Interpreting and Diagnosing Deep Learning Models: A Visual Analytics Approach

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Junpeng Wang, B.E., M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2019

Dissertation Committee:

Han-Wei Shen, Advisor

Huan Sun

Hyunwoo Park

Abstract

Recently, Deep Neural Networks (DNNs) have demonstrated superhuman capabilities in solving problems across a wide range of applications. The outstanding performance of those models shows great potentials in replacing human labor with artificial intelligence in the near future. However, behind this beautiful picture, those deep learning models often work like black-boxes, i.e., people using them have a good sense of their inputs and outcomes, but have little knowledge on the complicated working mechanisms in between. Given that DNN models are not infallible, deploying them in real-world applications may put humans in a very dangerous situation. This is especially true for safety-critical applications, such as medical diagnosis and autonomous driving.

The internal working mechanisms of DNNs are not easily interpretable. A DNN approximates a very complex function through an intricate combination of numerous linear and non-linear functions, which are organized as multiple computational layers. The more layers a DNN has (i.e., deeper), the more powerful it could be. Although each individual linear/non-linear function is simple, the combination of them could be exponentially complicated. This is where the powerful expression of DNNs comes from, as well as the reason that makes DNNs hard to be interpreted, as the data transformation in high-dimensional spaces could easily go beyond humans' interpretation capability.

The emerging field of Explainable Artificial Intelligence (XAI) targets to open the black-boxes of DNNs, and many interpretation techniques have been proposed, e.g., sensitivity

analysis, guided back-propagation. Focusing on a specific data instance of interest, those techniques interpret a DNN by examining how the model output is resulted from the corresponding input. Still, domain experts lack the ability in flexibly exploring the detailed intermediate outcomes and combining the piece-by-piece outcomes to draw insightful conclusions or derive useful insight. These abilities are significantly important in deepening the experts' understanding on what was actually going on and diagnosing potential deficiencies of their models. We believe combing XAI techniques with visual analytics is a promising solution here, which uses graphical interface with powerful interactivity to construct a human-in-the-loop analysis pipeline and empowers deep learning experts to conveniently examine DNNs with multiple levels of details.

This dissertation investigates how visual analytics can assist the process of interpreting, diagnosing, and potentially improving deep learning models. Two general solutions of a Black-Box approach and a White-Box approach are introduced through three design studies, covering supervised, unsupervised, and reinforcement learning models. In the first study, we propose *DeepVID*, which brings together the power of deep learning and visual analytics to effectively interpret and diagnose supervised DNN classifiers. The second study focuses on a popular unsupervised deep generative model, i.e., Generative Adversarial Nets (GANs), and we design *GANViz* to unveil the mysterious power of the model in image generation. Last but not least, we present the design study of *DQNViz*, which demonstrates the training evolutions of deep reinforcement learning models, and contributes to potential improvements of the Deep Q-Networks (DQNs). This dissertation ends with actionable future research directions, as well as a summary of our contributions.

Dedicated to My Parents Zhenjun Wang and Shuhua Ma

Acknowledgments

My four years doctoral study in the Ohio State University has been very enlightening, intriguing, fulfilling, and also challenging. The experience helped me a lot in developing my professional skills, sharpening my mind, and shaping my values. I feel very fortunate to receive helps and supports from many very kind people, who I would like to express my greatest thanks to, at the beginning of this dissertation.

First, I would like to thank my research advisor Prof. Han-Wei Shen, who sets an example to be a good researcher and mentor to me. The research freedom and the helpful guidance he provided, as well as the continuous support he devoted to my Ph.D. study, were deeply appreciated. Without these, I would not have the chance to explore the topic of deep learning and conduct the research I am really interested in. My thanks also go to my thesis committee members, Prof. Huan Sun and Prof. Hyunwoo Park, who have provided insightful comments to improve my work.

Second, the internship experience at Visa Research contributed significantly to my Ph.D. study. I would like to sincerely thank my mentor Liang Gou, who always gave me the most interesting topics to work on and helped me a lot to write clear and concise research papers. My sincere thanks also go to my supervisors and collaborators at Visa Research, Hao Yang, Wei Zhang, Jing Huang, Yan Zheng, Liang Wang, Yanhong Wu, and Dean Galland.

I must also thank my fellow lab-mates and friends, who have helped me significantly to develop my research sensitivity, critical thinking, and other professional skills. They are:

Wenbin He, Xiaotong Liu, Soumya Dutta, Ayan Biswas, Subhashis Hazarika, Cheng Li, Xin Tong, Kewei Lu, Xiaonan Ji, Ko-Chih Wang, Jiayi Xu, Tzu-Hsuan Wei, and Chun-Ming Chen. The research discussions with them are always thorough, inspiring, and encouraging. Their good sense of humor and the unintentional jokes have brought a lot of fun to me and the entire group.

I would also like to extend my sincere gratitude to my advisor and lab-mates during my Master's study in Virginia Tech. Prof. Yong Cao brought me to the area of visualization, and taught me that research should always go into low-level details. My lab-mates Peng Mi, Chao Peng, Bin He, Run Yu, Sruthi Iyer, and Seung In Park, helped me a lot in the years when I first came to the United States. I thank them all and will never forget the happy and joyful time we spent together in the small and beautify city, Blacksburg.

Lastly, I thank the unconditional love, care, and encouragement from my parents and sisters back in China. No matter what decisions I made in my life, they were always the strongest supporters behind me and provided whatever they can to help me. It is my greatest honor to have them and I would spend the rest of my life to love and support them.

Vita

September 1988	Born, Beijing, China.
2011	B.E. in Software Engineering, Nankai University, Tianjin, China.
2011-2015	Graduate Teaching Assistant, Virginia Tech, Blacksburg, Virginia, USA.
2012	Summer Intern, HX5 LLC, Fort Walton Beach, Florida, USA.
2013	Summer Intern, Qualcomm Inc, San Diego, California, USA.
2014	Summer Intern, Qualcomm Inc, San Diego, California, USA.
2015	M.S. in Computer Science, Virginia Tech, Blacksburg, Virginia, USA.
2017	Summer Intern, Visa Research, Palo Alto, California, USA.
2018	Summer Intern, Visa Research, Palo Alto, California, USA.
2015-2019	Graduate Research Associate, The Ohio State University, Columbus, Ohio, USA.

Publications

Research Publications

Junpeng Wang, Liang Gou, Wei Zhang, Hao Yang, Han-Wei Shen "*DeepVID*: Deep Visual Interpretation and Diagnosis for Image Classifiers via Knowledge Distillation", *IEEE Transactions on Visualization and Computer Graphics*, 2019, (IEEE PacificVis, Accepted).

Junpeng Wang, Liang Gou, Han-Wei Shen, Hao Yang "*DQNViz*: A Visual Analytics Approach to Understand Deep Q-Networks", *IEEE Transactions on Visualization and Computer Graphics*, 2019 Jan;25(1):288-298. **[IEEE VAST Best Paper Honorable Mention Award]**.

Junpeng Wang, Xiaotong Liu, Han-Wei Shen "High-Dimensional Data Analysis with Subspace Comparison Using Matrix Visualization", *Information Visualization*, 2019 Jan; 18(1):94-109.

Junpeng Wang, Subhashis Hazarika, Cheng Li, Han-Wei Shen "Visualization and Visual Analysis of Ensemble Data: A Survey", *IEEE Transactions on Visualization and Computer Graphics*, 2018 (Early Access).

Junpeng Wang, Liang Gou, Hao Yang, Han-Wei Shen "*GANViz*: A Visual Analytics Approach to Understand the Adversarial Game", *IEEE Transactions on Visualization and Computer Graphics*, 2018 Jun;24(6):1905-1917. **[IEEE PacificVis Best Paper Award]**.

Junpeng Wang, Xiaotong Liu, Han-Wei Shen, Guang Lin "Multi-Resolution Climate Ensemble Parameter Analysis with Nested Parallel Coordinates Plots", *IEEE Transactions on Visualization and Computer Graphics (IEEE VAST)*, 2017 Jan;23(1):81-90.

Junpeng Wang, Fei Yang, Yong Cao "A Cache-Friendly Sampling Strategy for Texture-Based Volume Rendering on GPU", *Visual Informatics*, 2017; 1(2):92-105.

Junpeng Wang, Ji Wang, Chris North "Spectrum: A Visual Analytics Tool to Explore Movement Logs", In *2015 IEEE Conference on Visual Analytics Science and Technology (IEEE VAST)*, pages 175–176, 2015.

Mai Elshehaly, Denis Gračanin, Mohamed Gad, Junpeng Wang, Hicham G. Elmongui "Real-Time Interactive Time Correction on the GPU", In *2015 IEEE Conference on Scientific Visualization (IEEE SciVis)*, pages 145–146, 2015.

Junpeng Wang, Mai Elshehaly, Yong Cao "Cylindrical Acceleration Structures for Large Hexahedral Volume Visualization", In *IEEE 5th Symposium on Large Data Analysis and Visualization (IEEE LDAV)*, pages 25-31, 2015.

Junpeng Wang, Fei Yang, Yong Cao "Computation-to-Core Mapping Strategies for Iso-Surface Volume Rendering on GPUs", In *IEEE 8th Symposium on Pacific Visualization (IEEE PacificVis)*, pages 153-157, 2015.

Junpeng Wang, Fei Yang, Yong Cao "Cache-Aware Sampling Strategies for Texture-Based Ray Casting on GPU", In *IEEE 4th Symposium on Large Data Analysis and Visualization (IEEE LDAV)*, pages 19-26, 2014.

Weiwei Cai, Xusong Li, Yong Cao, Junpeng Wang, Lin Ma "Practical Aspects of Three-Dimensional Flame Imaging Using Tomographic Chemiluminescence", In *AIAA Science and Technology Forum and Exposition (SciTech)*, pages 1-9, 2014.

Neda Mohammadi, Junpeng Wang, Yong Cao, Mehdi Setareh "SMATS: Sketch-Based Modeling and Analysis of Truss Systems", In *Architectural Research Centers Consortium (ARCC) Conference Repository*, pages 143-147, 2013.

Fields of Study

Major Field: Computer Science and Engineering

Studies in:

Computer Graphics	Prof. Han-Wei Shen
Artificial Intelligence	Prof. Mikhail Belkin
High Performance Computing	Prof. Ponnuswamy Sadayappan

Table of Contents

List of Tables

Table **Page**

List of Figures

Chapter 1: Introduction

1.1 Background and Motivation

Recently, Deep Neural Networks (DNNs) have demonstrated superhuman capabilities in solving problems across a wide range of applications. For example, in 2015, Microsoft Research proposed a DNN (i.e., the ResNet [44]) to solve the image classification problem on the ImageNet dataset (i.e., classifying 10 million images into 1000 object categories), and the DNN was able to achieve an error rate that is even lower than human beings. In 2016, for the first time, professional human Go players were defeated by a computer program (i.e., the deep reinforcement learning agent AlphaGo [95]) developed by Google DeepMind. From 2014 to 2018, the rapid growth and development of deep generative models [38] made it possible to generate data (e.g., images [83], speech [89], 3D models [111]) that are almost indistinguishable from real data. Except these breakthroughs, DNNs have also enabled us to accomplish many tasks that were previously insurmountable, such as preventing and controlling diseases, combating and reducing crimes, investigating and predicting business trends, etc. As we have not been fully aware of, deep learning has started a new technological revolution.

On the other hand, deep learning models often work like "black-boxes" with opaque internal working mechanisms. Their mysterious power prevents people of the safety-critical

areas from using them, as those models are not infallible and their extraordinary good performance is not a measure of model trust. For example, a nearly perfect DNN model may achieve 99.9% accuracy in diagnosing heart diseases. However, except the diagnosis result, the model provides no clues on how the result was achieved and the model's internal working mechanism is hard to be directly interpreted by doctors. This would really concern the doctors, as they cannot explain their diagnosis to their patients and the very small chances of misdiagnosis could significantly affect their treatments. Similarly, in autonomous driving, misclassifying a stop sign as a speed-limit sign may happen very rarely, but it is already severe enough to lead to a grave consequence. With the fast growing of DNN models, being able to interpret why and how the models behaved in different situations becomes increasingly important, apart from blindly pursuing better performance.

The emerging field of Explainable Artificial Intelligence (XAI) [76, 91] targets to open the black-boxes of DNNs. A myriad of interpretation techniques have been proposed in this field, e.g., sensitivity analysis [76], class activation mapping [23, 93, 116], layer-wise relevance propagation [18, 75]. Through the well-trained model parameters and the connected computational graphs, those algorithms could interpret a DNN model by answering which part of the model input contributed positively/negatively to which part of the model output, which enables humans to verify if the model really learns the expected behaviors and build trust on it. However, solely relying on the above automatic algorithms is not sufficient to understand the intricate behaviors of DNNs, as well as the increasingly large, complex, and multi-faceted data generated from DNNs. These data include but are not limited to: DNN model architectures, intermediate neural activations, trained model parameters, training statistics, etc. To capture the temporal evolution, these multi-faceted

data may need to be collected every single training iteration, resulting in a very large and heterogeneous dataset.

Tailored for analyzing large and complex digital information, *exploratory data analysis* [104] combines fully-automatic approaches with interactive explorations by including humans in the data analysis loop. During analysis, users can iteratively propose hypotheses, assess assumptions, and derive insight, to progressively gain knowledge about their data. Moreover, *visual analytics* [26], which supports analytical reasoning with interactive graphical interfaces, further enhances exploratory data analysis by taking advantages of humans' visual perceptual capability. Through encoding complex digital information into intuitive graphical displays, users can more effectively perceive the valuable information hidden inside their data. As visual analytics has turned into one of the most promising solutions toward tackling the increasingly severe big data challenge, this dissertation leverages its power in analyzing the large and complex data generated from DNN models to serve the goal of model interpretation and diagnosis.

1.2 Scope of Research

According to the most widely accepted categorization in machine learning [82], there are three types of learning approaches:

- The *supervised learning* learns a function from a large number of input instances with desired labels, such that the function can predict the label for any unseen input instance. Typical supervised learning problems include classification and regression.

- The *unsupervised learning* learns a function to capture the hidden data patterns/structures from a large number of data instances without labels. The most notable unsupervised learning examples are cluster analysis and anomaly detection.

- The *reinforcement learning* trains an agent, which issues actions to maximize the cumulative rewards in a given environment. The learning is conducted on numerous data tuples of issued actions, environmental states, and achieved rewards. AlphaGo [95] is a famous reinforcement learning agent trained to play the Go game.

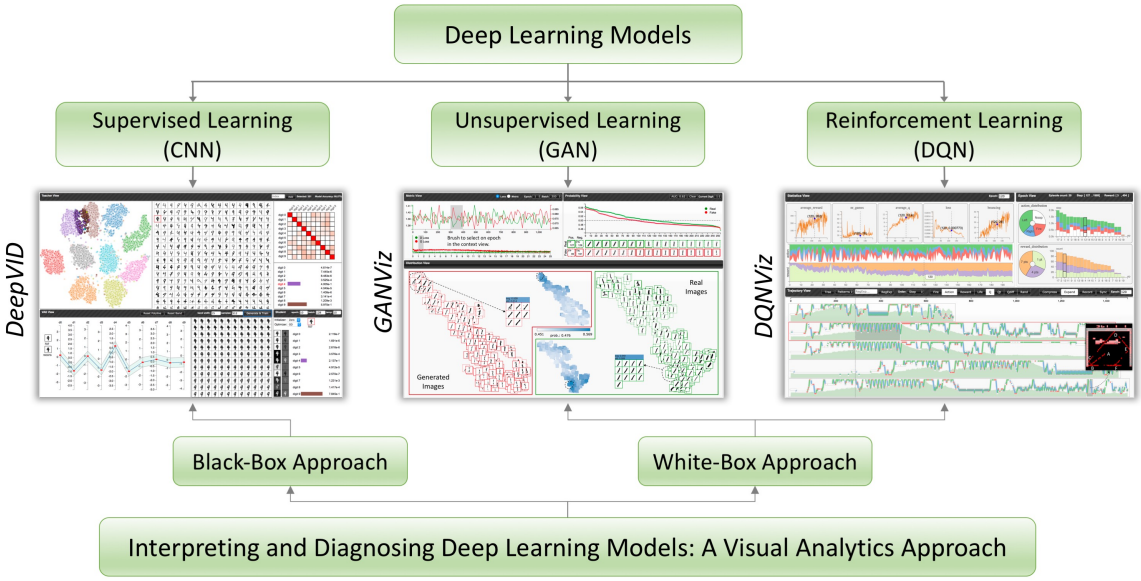

Figure 1.1: Our visual analytics framework introduces two general solutions of a Black-Box approach and a White-Box approach, covering supervised, unsupervised, and reinforcement learning models. Focusing on three DNN models in each learning approach (CNN, GAN, and DQN), three visual analytics prototypes, i.e., *DeepVID*, *GANViz*, and *DQNViz*, are proposed over our studies. They are explained with details in Chapter 3, 4, and 5 respectively.

DNNs have been used to implement all the three types of learning approaches. Among the many DNN models in each of them, we pick out the most representative ones to study in this dissertation. Specifically, we will study the Convolutional Neural Networks (CNNs) for supervised learning, the Generative Adversarial Networks (GANs) for unsupervised learning, and the Deep Q-Networks (DQNs) for reinforcement learning, as shown in Figure 1.1.

- **CNN** is one of the most widely used image classifiers, which chains a sequence of convolutional layers to extract features from images and employs fully-connected layers to map the extracted features to class probabilities.

- **GAN** is an unsupervised deep generative framework, which contains two sub-networks of a generator and a discriminator. The generator produces data from random vectors, whereas the discriminator discriminates the generated data from real data, and learns data features in an unsupervised fashion. The adversarial battle between the generator and discriminator makes both of them become stronger over the training.

- **DQN** is a CNN based reinforcement learning model proposed by Google DeepMind using the Q-learning theory [110]. The model has successfully trained an agent to play different Atari 2600 games and achieved superhuman performance.

These three types of models are also chosen based on the research interests of the deep learning experts that we collaborated with, so that they can provide valuable domain feedback to our work. It is worth mentioning that these models do not necessarily cover the features of all types of DNN models, but through the studies of them, we can effectively leverage the power of visual analytics, and think about how to extend the proposed visual analytics solutions to other more general DNN models.

Our research focuses on interpreting and diagnosing the three types of deep learning models by examining and analyzing the data generated from them with visual analytics approaches. The generated DNN data include but are not limited to the raw training data, the DNN model architecture, the intermediate neural activations, the trained model parameters, and the training statistics. We use image data to demonstrate and verify our idea in all studies, as the semantics of images is more easily perceivable by our visual channel compared to

other data types, e.g., texts or sounds. However, we discuss the feasibilities, as well as potential challenges, in extending our approach to other data types in individual studies, as well as our future works in Chapter 6.

1.3 Problem Statement

This dissertation targets to investigate problems on *interpreting* and *diagnosing* deep learning models through visual analytics and exploratory data analysis. Particularly, we target to shed some light onto the following two high-level research questions.

Question 1: *How can we exploit the power of coordinated multiple views with linked interactions to effectively demonstrate different aspects of the multi-faceted DNN data, and facilitate the model interpretation and knowledge discovery process?* Extracting useful implications from DNN data to assist the interpretation and diagnosis of the corresponding models often requires the analysis on different facets of the data simultaneously. However, due to the sheer size and high complexity of those data, as well as the limited load of humans' perceptual capability, only partial information can be presented at once. As a result, what parts/aspects of those data should be presented at the same time needs to be decided wisely. The order of presenting different parts/aspects also needs careful considerations. Building the connections between partial information presented at the same time and across the dynamic exploration process to support model interpretation and knowledge discovery, therefore, is a significantly important research question.

Question 2: *How can we take advantages of visual analytics and exploratory data analysis to diagnose DNN models and suggest model improvements beyond what deep learning experts can do with their conventional approaches?* Model diagnoses and improvements are very important to domain experts. However, due to their inability in investigating every

single detail of a model or the model training process, they often fall short of detecting certain model deficiencies. For example, to diagnose and improve a DQN agent trained to play the Atari games, the domain experts first need to find out how the agent succeeded or failed in different game scenarios, they can then examine the reasons behind those scenarios for further improvements. However, the typical agent behaviors are often buried in the huge amount of playing histories and cannot be easily dug out. We target to demonstrate how visual analytics and exploratory data analysis can help here. Additionally, we will present how they can be used to verify if certain remedies/improvements are effective to a DNN model or not, with thorough details beyond the numerical values provided by traditional summary statistics.

1.4 Proposed Solutions and Contributions

We propose two general solutions for interpreting and diagnosing DNN models in this dissertation, i.e., a Black-Box approach and a White-Box approach, which are demonstrated across the studies of the three focused DNN models explained in Section 1.2.

1.4.1 Interpreting and Diagnosing DNNs with a Black-Box Approach

The Black-Box approach interprets and diagnoses DNN models by probing their behaviors around a particular data instance of interest, using only the inputs and outputs of the models. The general idea of the Black-Box approach is to interpret and diagnose a cumbersome DNN model through a weak learner (local interpreter) derived from local approximations. For example, assuming we want to interpret the DNN binary classifier shown in Figure 1.2 (left), which separates two sets of data points in green and orange. The red curve denotes the true decision boundary of the DNN. Although depicting and interpreting the red curve (i.e., the complicated global boundary of the classifier) is difficult,

the classifier's local behavior around the sample of interest (the red point) can be explained with a simple linear model (the black dotted line), which can be trained using only the input and output of the original classifier around the sample of interest (i.e., points in the shaded region).

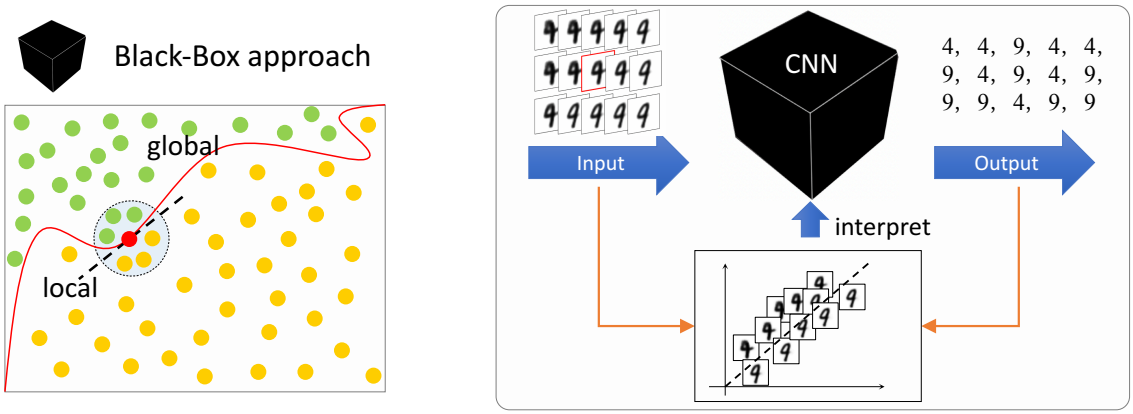

Figure 1.2: (Left) the Black-Box approach conducts model interpretation through local approximations. (Right) probing the decision boundary (between digit 4 and 9) of a CNN classifier, and interpreting it through a linear model (i.e., a local approximation of the CNN).

The right part of Figure 1.2 shows a real example, where we use the Black-Box approach to interpret how a CNN classifier differentiated hand-written digit 4 and 9. Our local approximated model (i.e., a linear regression model), trained using the input (images of digit 4, 9) and output (class probabilities) from the original CNN, considers each pixel as a feature dimension and assigns weights to those dimensions to reveal their contributions to the final prediction. By visualizing the linear model (i.e., the learned coefficients of the regression model), we can explain how the CNN worked in the interested local region without knowing its complicated internal architecture.

Chapter 3 of this dissertation introduces an interpretation framework, named *DeepVID*, which is an implementation of this Black-Box approach. Using that framework, we are able to interpret different types of CNN models and diagnose those models by dissecting why and how they succeed or fail in different scenarios.

1.4.2 Interpreting and Diagnosing DNNs with a White-Box Approach

As shown in Figure 1.3, the White-Box approach exposes the internal layer architecture of DNNs and enables users to examine the intermediate computational results, or the states of different network components, to interpret how the DNNs worked. In the illustrative example shown on the right of Figure 1.3, a human face image is predicted as the class of 'male' with 'glasses'. By visualizing the activations of the hidden layers, we can figure out what had been really extracted by different neurons, and which neurons identified the 'glasses' feature. Moreover, by sorting neurons of the same layer based on their contribution to the final predictions, one can identify their importance to the classification task. These findings can potentially help deep learning experts better decide the number of computational layers in their DNNs, as well as the number of neurons per layer [81].

In Chapter 4, we adopt the White-Box approach to study a deep generative model, i.e., Deep Convolutional Generative Adversarial Nets (DCGAN). To address the general challenge of effectively demonstrating the multi-layer structure of DNN models, we propose a novel compact visual design, called *TensorPath*, which lays out the multi-layer neural network structure along a zigzag path and embeds all the data of interests (e.g., activations, convolutional filters, trained parameters, etc.) onto the path. Tracking along the zigzag layout, deep learning experts can intuitively perceive how the input training data flow through the neural network. They can also record the intermediate computational results

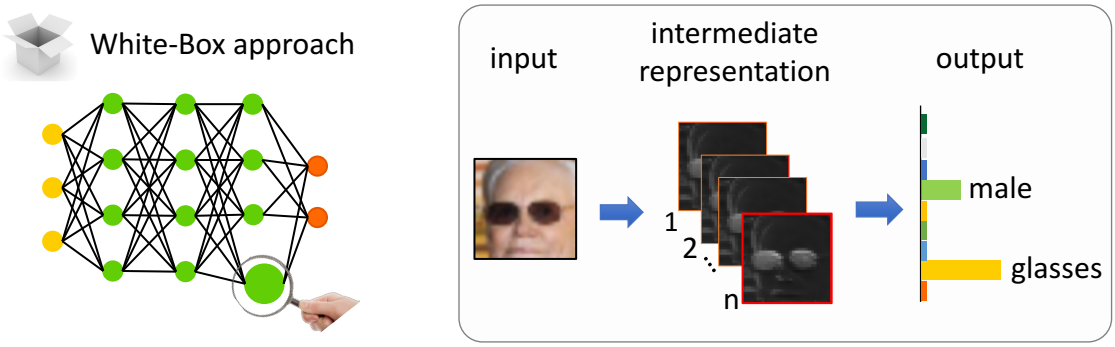

Figure 1.3: (Left) the White-Box approach enables users to examine the intermediate representations of the data or network components to interpret and diagnose DNN models. (Right) an illustrative example of using the White-Box approach to investigate which neurons really extracted the 'glasses' feature from the human face image.

(e.g., feature maps in CNNs) from one or multiple pairs of interested training instances along the zigzag path to compare them, across-class or across-time. Multiple visual analytics views are coordinated with the *TensorPath* view to empower users to flexibly explore and examine different components or computational results of the DCGAN. Integrating all those views, we design and develop a visual analytics system to the end, which is named *GANViz*.

We have also investigated the power of the White-Box approach in interpreting and diagnosing deep reinforcement learning models. In Chapter 5, we introduce a visual analytics system, i.e., *DQNViz*, which helps to identify the playing strategies that a DQN agent learned during the long-time training process. Those strategies effectively reveal what the agent was good at and what the agent was not capable of over the training. To the end, we work with several deep reinforcement learning experts to propose an improvement derived directly from our visual analytics to optimize the model training process of DQNs.

1.5 Outline and Overview

The rest of this dissertation is organized into the following four parts. Chapter 2 uses a story of the annual image classification competition to demonstrate the rise of deep learning in recent years. It also introduces the fundamental DNN concepts, as well as the state-of-the-art works related to model interpretations and diagnoses using visual analytics. Chapter 3 focuses on CNNs to illustrate the use of our Black-Box approach in supervised learning models. Along with the illustration, we propose an interpretation and diagnosis framework for image classification models, i.e., *DeepVID*. Focusing on explaining the White-Box approach, Chapter 4 and Chapter 5 introduce *GANViz* and *DQNViz*, in the context of interpreting and diagnosing unsupervised learning and reinforcement learning models, respectively. Finally, Chapter 6 and Chapter 7 conclude the dissertation by summarizing the findings from the preliminary results we have demonstrated and proposing several research directions that we plan to explore in the future.

Chapter 2: Background and Related Work

As our works are conducted with the rise of deep learning, we would like to start this background chapter with a story of the world-wide image classification competition to show the increasing impact of deep learning. Next, we dive into details to explain the fundamental concepts of DNN models to support our subsequent discussions. Lastly, we review the related works that used visual analytics to interpret or diagnose deep learning models.

2.1 The Rise of Deep Neural Networks

ImageNet [28] is a large-scale world-wide image database containing more than 14 million images in more than 20 thousand categories. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC [87]) picked out 1000 categories from the database for an annual competition, which challenges researchers, scholars, and developers across the world to propose more accurate object recognition and image classification algorithms. Around 1000 image instances are extracted for each selected image category, composing a dataset of more than 1.2 million images in 1000 categories.

Figure 2.1 shows the results of the ILSVRC competition from 2010 to 2015. The performance of the proposed algorithms is measured by top-five error rate (i.e., a prediction is incorrect if the target label is not in the five predicted labels with the highest probabilities). From the figure, it is obvious that the first drop of the error rate happened in 2012, which

is when DNNs were first introduced into this competition. The 8-layer AlexNet [59] (i.e., the winning algorithm in 2012) is much superior to the best traditional computer vision algorithms. From 2012 to 2014, the general trend is that the winning algorithms are always DNN-based and they are getting deeper and deeper (AlexNet [59], ZFNet [113], VGG [96], and GoogleNet [100]). To 2015, the ResNet [44] reached the error rate of 0.036, which is even lower than the error rate of human beings. The competition continued for two more years and it was finally stopped in 2017, as the challenge was considered solved.

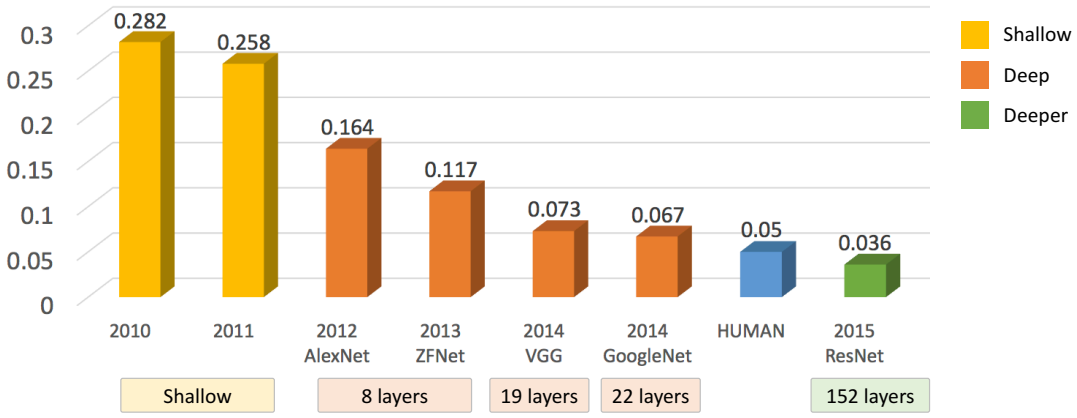

Figure 2.1: The top-five error rate of the winning algorithms for the ILSVRC annual competition. The first big drop happened in 2012, when DNNs were first introduced. To 2015, the DNN approach could achieve an error rate that is even lower than human beings.

The story of ILSVRC demonstrates how powerful DNN models could be and their fast-growing pace. Meanwhile, the complexity of those models, however, has gone far beyond humans' interpretability. For example, the ResNet has 152 computational layers and each layer has a large number of neurons, resulting in hundreds of millions of parameters being tuned in each training iteration. No one can precisely describe how the input data have been transformed through this complicated neural network, and it becomes more and more

difficult for deep learning experts to diagnose their models. Under such circumstances, model interpretation and diagnosis stepped onto the stage and became increasingly important.

2.2 The Basics of Deep Neural Networks

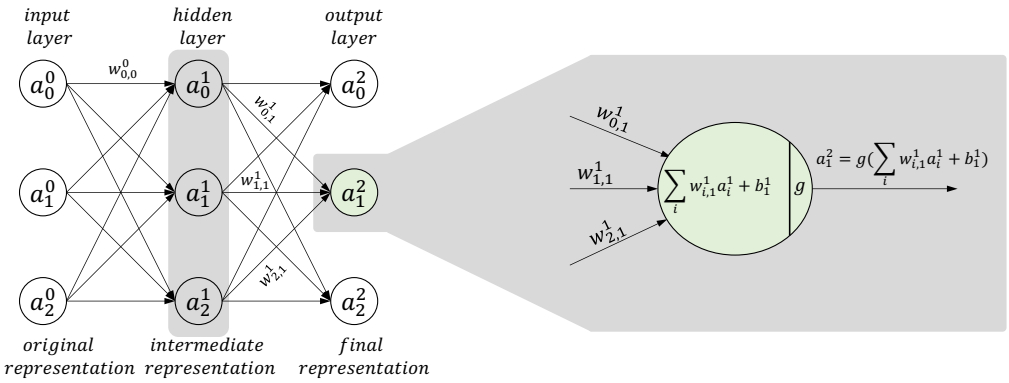

Figure 2.2: (Left) a DNN with three computational layers. The circles represent neurons and the lines connecting them represent the weights/parameters to be trained. (Right) a basic computational unit (a neuron). Activations from the previous layer are integrated through a weighted sum. The sum then goes through a non-linear activation function, i.e., the function g, to decide if the neuron should fire or not.

A DNN is used to approximate an extremely complex function: $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, through the combination of numerous simple linear and non-linear functions. In practice, a DNN is composed of multiple computational layers, which are chained one after another, as shown in Figure 2.2 (left). Each layer contains numerous neurons, which are the atomic computational units in the DNN. Figure 2.2 (right) gives an example of the computation details inside a neuron (superscripts are used to indicate the layer indices and subscripts are for neuron indices). This neuron computes a linear combination of the three activations from the previous layer ($\Sigma_i w_{i,1}^1 a_i^1$), adds a bias (b_1^1) to the combination, and applies a non-linear activation function, i.e. g, to the result to generate a new activation (a_1^2). It can be seen that

the data involved in a typical DNN are: weights, biases, and activations. The input and output of a DNN can be considered as activations of the first and last layer respectively.

2.2.1 The Architecture of DNNs

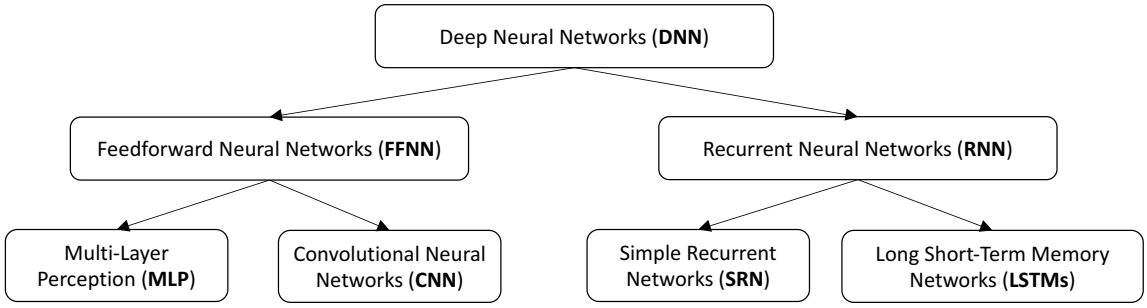

Figure 2.3: A general taxonomy of DNNs based on their architecture.

From the architecture point of view (i.e., how neurons are connected), there are two major types of DNNs: Feedforward Neural Networks (FFNN) and Recurrent Neural Networks (RNN). The fundamental difference between them is whether the neurons are recurrently connected or not. These two types of neural networks can be further divided into two nested groups respectively, as shown in Figure 2.3. FFNN can be further categorized into Multi-Layer Perceptron (MLP) and Convolutional Neural Networks (CNNs). Neurons from different layers of a MLP are fully-connected, whereas in CNNs, neurons from different layers are not fully-connected and some connections (i.e., weights) are shared between neurons from neighboring layers. With the weight sharing mechanism, CNNs can effectively capture spatial features, which makes them very popular in image related data analysis problems. There are two major variants of RNN: Simple Recurrent Networks (SRN) and Long Short-Term Memory networks (LSTMs). Both maintain a sequence of hidden states

and the only difference between them is how the hidden states are computed. Compared to SRN, LSTMs introduce different gates (i.e., input gate, forget gate, and output gate) to better preserve the long-term dependency. Within the scope of this dissertation, we focus only on DNNs with the feedforward architecture.

2.2.2 Three Types of Computational Layers in DNNs

The DNN models used in this dissertation cover three major types of computational layers: (1) fully-connected layers; (2) convolutional layers; (3) and transposed convolutional layers, which are detailed as follows.

In *fully-connected layers* (also known as dense layers), a neuron in layer l has connections to all neurons from its previous layer $(l-1)$ and its successive layer $(l+1)$, as shown in Figure 2.2 (left). The all-to-all connections make the size of training parameters (weights and biases) very large, though not all of them are equally important. A DNN contains only fully-connected layers is a MLP.

Convolutional layers are very similar to fully-connected layers, except two changes: (1) neighboring layers do not have the all-to-all connections; (2) some weights between different neurons of neighboring layers are shared. As shown in Figure 2.4 (left), there is no connection between the third neuron in the first layer and the first neuron in the second layer. Also, weight $w^0_{0,0}$, $w^0_{1,1}$, $w^0_{3,2}$, and $w^0_{4,3}$ share the same value. A convolutional layer expresses the convolutional computation, and the trained weights are the convolutional kernels. As convolutional layers are usually used to extract features from images, each of which is composed of many pixels organized as 2D, the neurons in convolutional layers are more frequently presented in a 2D manner, as shown in Figure 2.4 (right). The outputs of a convolutional layer, i.e., the activations organized in 2D, are often called as feature maps (the

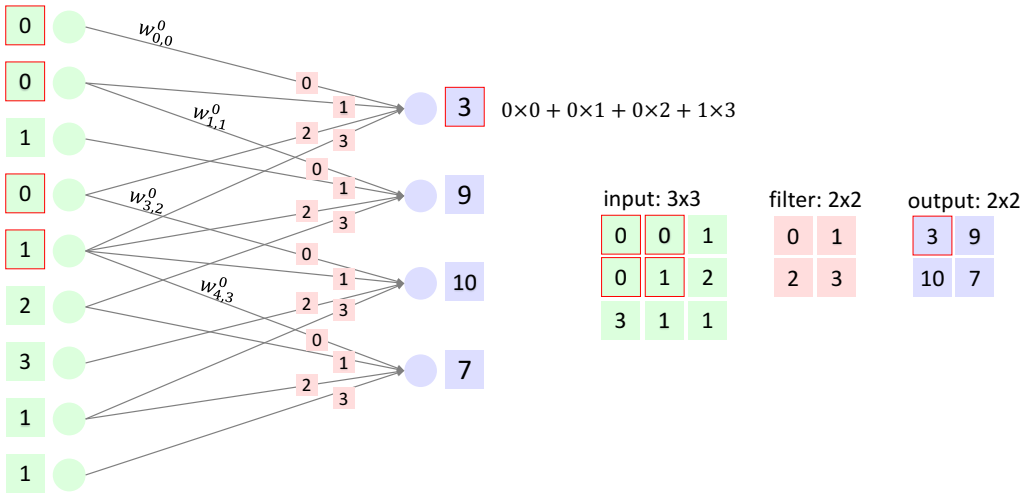

Figure 2.4: (Left) a typical convolutional layer. It can be seen that not all neurons are connected and some weights are shared between neurons from neighboring layers. (Right) a more commonly used representation for a convolutional layer. For simplicity, biases and activation functions are not shown in this figure.

blue patch in Figure 2.4, right), as they reveal what features have been extracted/activated by the layer. This way, the 2D spatial information is better presented. Except the neurons in different layers, the trained weights are often organized as 2D, as well (the red patch in Figure 2.4, right), and they are called as filters (i.e., convolutional kernels). From Figure 2.4, the relationship between a filter and the corresponding set of weights can be described as follows (the ith filter in the jth layer): $f_i^j = \{w_{0,i}^j, w_{1,i}^j, ..., w_{p-1,i}^j\}$, where p is the size (total number of pixels) of the filter.

Transposed convolutional layers are very similar to convolutional layers. However, instead of extracting features from images, they are often used to fill features into images to enlarge them. Figure 2.5 shows the example of a transposed convlutional layer. We can see the image patch of size 2×2 is increased to 3×3 through the transposed convolutional operation.

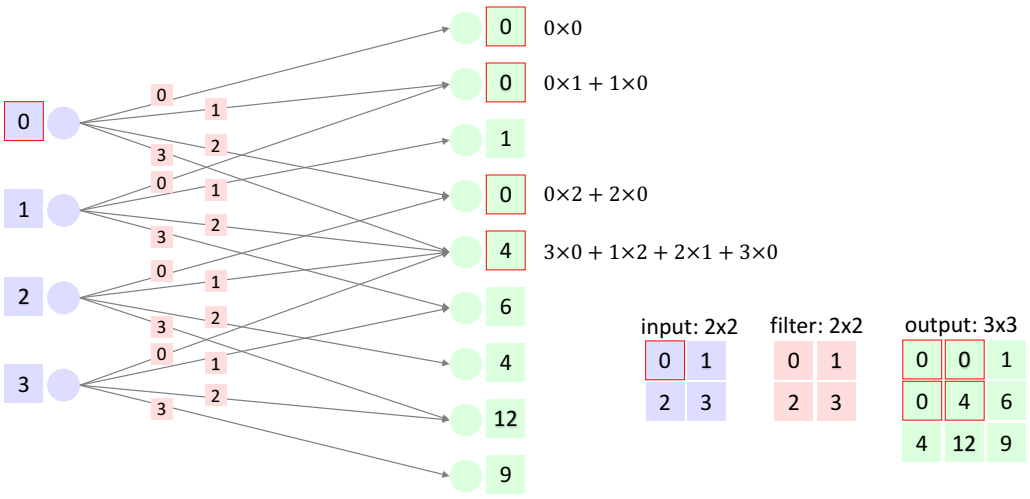

Figure 2.5: (Left) a typical transposed convolutional layer. It is very similar to the computation happened in a convolutional layer, except that the output layer has more neurons than the input layer (i.e., up-sampling the input). (Right) the 2D representation of the layer.

2.2.3 Fundamental Concepts Used in DNNs

This section takes a CNN as an example to explain how the fundamental concepts in DNNs are used. A CNN is usually composed of multiple convolutional layers, max-pooling layers (usually used after each convolutional layer to down-sample the 2D feature maps) and fully-connected layers (usually used at the end of a CNN to reshape the output to the desired size). Figure 2.6 (left) shows an example CNN architecture, and Figure 2.6 (right) zooms into the first convolutional layer to label the basic concepts:

- Neuron (circles in Figure 2.6, right): A basic computational unit conducting linear transformations of previous inputs.

- Layer (vertical rectangles in Figure 2.6, right): A collection of neurons that perform the same computation, but with different parameters. There are different types of layers

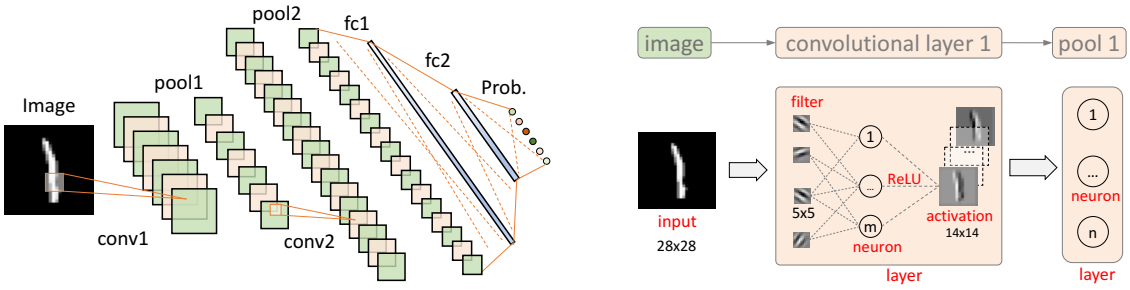

Figure 2.6: An example CNN architecture (left) and the enlarged version of its first convolutional layer (right).

according to their computation purposes (e.g., convolutional layer, fully-connected layer). Chaining layers one after another becomes a neural network. To make neural networks powerful, more and more layers are chained together, making the networks deeper and deeper.

- Activation: Activation occurs when the value of a neuron goes over a certain threshold, often decided by an activation function, e.g., Rectified Linear Unit (ReLU). In the image domain, individual activations are often composed together based on their spatial location to indicate what continues features are extracted. Those grouped activations constitute a feature map (i.e., thumbnails in Figure 2.6, right).

- Channel: Channel can be considered as the depth of one image. For a typical RGB image, the number of channels is 3 (i.e., the red, green, and blue color channels). In CNNs, the number of channels could be very large to capture various types of image features.

- Filter and weight: Parameters of the neurons used to conduct linear transformations on the inputs or previous layer activations. The parameters of neurons in fully-connected

layers are often called weights. For convolutional layers, those weights are shared and often organized as 2D (see Figure 2.4), which are called filters (i.e., convolutional kernels).

- Loss function: A function measures the cost of inaccuracy of predictions. The loss values are often considered as the indicators for the model quality. The training of a DNN is to minimize the loss value.

- Confusion matrix: A matrix measures the performance of a classifier by showing the number of test instances in four cells (for a binary classifier): true-positive, false-positive, true-negative, and false-negative. The matrix could be easily extended to multi-class classifiers, where the number of matrix cells is the square of the number of classes.

- Batch: A collection of input data instances (e.g., 64 images) used to train/update the neural network at one training iteration. A batch is often considered as the atomic unit to measure the progress of an iterative training process.

- Epoch: One epoch means all input data instances have been iterated once. For example, the MNIST dataset has 70,000 images of hand-written digits. If the batch size is 64, one epoch will have 1094 (70,000/64) batches. Depending on the complexity of the input data, it may take tens to hundreds of epochs to train a model.

2.3 Visual Analytics Efforts for Deep Learning Models

Many visualization and visual analytics techniques for interpreting and diagnosing DNNs have been proposed, as deep learning is gaining its popularity. This section provides a brief survey on those techniques from three different perspectives: (1) the DNN data that

the visualizations focused on; (2) the goals of visualizing DNN data; (3) the strategies used to explore DNNs.

2.3.1 Four Types of DNN Data

Based on the brief introduction of DNNs in Section 2.2, there are two types of DNN data: activations (e.g., input/output, feature maps from CNNs, hidden states from RNNs) and network parameters (e.g., weights, biases, convolutional filters). When working with a specific DNN model, the computational graph of the model (i.e., the model architecture/structure/-topology) often needs to be visualized for analysis as well. Moreover, different model statistics (e.g., loss value, model accuracy) over the training provide informative reflections regarding the training progress, and therefore, are often the focus of domain experts. In summary, there are four types of data from deep learning models: *(1) model architecture/structure/topology information; (2) model activations; (3) model parameters; and (4) model metrics*. Among them, activations are often the focus of analysis, as they encode what DNNs really learned from the training. In the following, we will explain some typical visualization techniques proposed for each of the four types of data.

Model Architectures. The architecture/structure of a DNN shows what computational layers the DNN has and how those layers are chained (layer-level information), as well as the number of neurons and how the neurons from different layers are connected (neuron-level information). The most popular visualization for presenting layer-level information is the node-link diagram [6, 51], from which users can have an overview of the network structure and quickly locate individual computational layers for detailed investigations. Diving into neuron-level structures, Liu et al. [66] modeled a CNN as a Directional Acyclic Graph (DAG), which is presented using a visualization similar to a Parallel Coordinates Plot (PCP).

Each parallel axis of the PCP is a computational layer and neurons in the layer are clustered and presented as groups of images. The images shown on the axes are subregions of the input images, extracted through the region proposal method [35], which can maximumly activate the corresponding neurons.

Model Activations. Most of the deep learning visualization works focus on visualizing the activations of DNNs, as they represent what features the networks really extracted. The straightforward way of visualizing activations is to normalize their values and map them to different visual channels, e.g., shape, color, etc. For example, in [66], the authors demonstrated the activations of a CNN using a matrix visualization. Each matrix cell represents the average activation (over all data instances) of a neuron to a class. Different rows of the matrix represent different neurons, and different columns represent different classes. The order of neurons (row order) is not important, whereas the order of classes (column orders) is very important to help users understand the activation strength to each class. Therefore, the authors reordered the rows (neurons) to augment/regularize the color patterns in the matrix and help users to more effectively perceive the activation pattern. Kahng et al. [51] adopted a similar approach of encoding neurons' active levels to colors in their ActiVis system. They can conduct both instance-level and subset-level analysis with those color-encoded activations. The instance-level analysis investigates the activation patterns when a particular data instance flows through the neural network; whereas the subset-level analysis aggregates the activations from instances of the same subset to reveal the collective pattern of the subset. Driven by different analytical tasks, the subset can be defined flexibly by users (e.g., data instances of the same class can be one subset).

For one data instance, the activations of all neurons in a particular layer can be considered as a high-dimensional vector, which is a learned intermediate representation of the original

data instance. The high-dimensional vectors for different data instances can be reduced to 2D through dimensionality reduction algorithms and presented in a scatterplot. Rauber et al. [84] successfully demonstrated the power of dimensionality reduction using the t-SNE algorithm [69]. Each point in their t-SNE projection represents one data instance, and the point is colored based on the class information of the corresponding data instance. Visualizing the activations from different layers in separate scatterplots can help users compare and understand the evaluation of the learned data representations across layers. For example, the two scatterplots generated for the first and last layers of the same DNN helped Rauber et al. [84] conclude that the forward propagation separated data instances from different classes. One can also compare the scatterplots generated for the same layer but in different training stages to see how the network evolves over time [51].

For CNN, the activations are often demonstrated as 2D images (i.e., feature maps). ConvNetJS [2] unrolled the CNN architecture into a sequential layout and visualized the activations layer-by-layer. From the visualization, each individual feature map (including the input image and output probability) can be quickly indexed and examined. Most other visualization works for CNNs adapted a similar approach for feature map visualization. For example, DeepVis Toolbox [112] visualized the learned filters and activated features as images as well.

For RNNs (both SRN and LSTMs), heat-maps have been used extensively to visualize the probability of predicted outputs (i.e., output layer activations). For example, LSTMVis [99] studied a word prediction application, and the top-k predicted words were sorted based on their probability and colored with blue colors from dark to light. The tool also employed a PCP to visualize the internal hidden states of RNNs. In detail, each hidden state $\mathbf{h_t}$ can be considered as a high-dimensional vector that encodes the contributions from $\mathbf{x_t}$ and

$\mathbf{h_{t-1}}$. Connecting values from the corresponding dimension of the hidden states across all time steps constituted a temporal sequence, which was presented as a polyline in the PCP. Users can select and search different patterns from the PCP through different interactions. Meanwhile, data instances sharing similar hidden state patterns could also be extracted to augment the common features in those instances.

Model Parameters. Tzeng and Ma [105] used MLP to classify the voxels from a 3D CT volume. As their MLP architecture was not very complicated, they directly visualized all neurons as nodes, and presented weights as lines connecting those nodes. The width of each line encoded the derived importance value of the corresponding connection. Using their visualization, they could successfully figure out what input dimensions were more important to the classification task. There are more works focusing on demonstrating the trained weights of CNNs, i.e., the convolutional filters. For example, in [2], the convolutional filters were presented as image patches. The gradients of individual filters could also be visualized to help people analyze the changes of weights over the training process.

Model Metrics. The metrics/statistics for DNN models can be either static or dynamic. When the analysis focuses on investigating how a well-trained model works, the model metrics are mostly static and collected from the testing stage. For example, Squares [85] studied the performance of classification models by analyzing the models' confusion matrix, which was presented using a PCP-like design and each PCP axis represented one class. The authors used different glyphs (e.g., solid squares, empty square, striped squares) with different colors to represent predictions in different categories (e.g., false-positives, true-negatives). The glyphs representing individual data instances presented an overall distribution of the test data, which was more comprehensive than a simple numerical accuracy value in assessing the model quality.

When the analysis of DNN data focuses on the entire DNN training process, more dynamic model statistics/metrics will be collected along with the training. For example, the Squares visualization toolkit [85] we discussed before can also be used to present and compare the confusion matrices from different training stages. By watching how the visualization changes over time, one can perceive whether the model is progressing towards the right direction or not. For most cases, the dynamic model statistics are simple time-series data, such as loss/accuracy values, and line chart visualizations are often the most straightforward choices for them.

The above discussions list some example works that visualize the four types of DNN data using different visualization techniques. A visualization technique usually visualizes multiple types of DNN data simultaneously, and conversely, the visualization of a single type of DNN data may include multiple visualization techniques. Therefore, there are both many-to-one and one-to-many relations between individual types of DNN data and different visualization techniques.

2.3.2 Three Objectives of Visual Analytics on DNNs

Aligned with the opinion of Liu et al. [66], we believe the visual analytics for DNNs should target on three objectives, i.e., interpretation, diagnosis, and improvement. This section explains these three objectives with detailed examples from the literature.

Model Interpretation. The objective of interpreting a DNN is to reveal the model details at multiple levels to explain how the model worked and enhance the understanding of the model. For example, AEVis [64] interpreted how an adversarial example could trick a CNN model by extracting the detailed information when the adversarial image and the corresponding normal image flowed through the network, i.e., their datapath. By

comparing the datapaths with multiple levels of details (i.e., layer level, feature map level, and neuron level), the authors were able to explain which regions of the normal image are more vulnerable to attacks. Another example is the work of Rauber et al. [84], which tried to visualize the hidden activities in DNNs through dimensionality reductions. The large amount of high-dimensional activations from each neural layer form a high-dimensional space. Projecting those activations from different layers, or from different training stages of the same layer, into 2D and comparing them in the lower dimensional space could expose how the high-dimensional activations were evolving in the original high-dimensional space. Rauber et al. [84] revealed how the activations were gradually separated through the forward propagation of a DNN by visualizing the dimensionality reduction results from all internal layers. Using the same approach, they could also interpret the training progress of a DNN by monitoring the dimensionality reduction results of the same layer across time.

Model Diagnosis. The objective of diagnosing a DNN is often to dissect why and how the model succeeded or failed in different scenarios, which can serve further improvements of the model. For example, DGMTracker [65] proposed a three-level framework to diagnose the long-time training process of deep generative models, i.e., snapshot-level, layer-level, and neuron-level. From this top-down analysis framework, the authors were able to examine the status of the network when different data instances flowed through it and find what neurons behaved irregularly, as well as the root cause for the abnormal behaviors. LIME [86] is another famous model-agnostic diagnosis framework, which breaks the studied image into many small fractions based on the homogeneous level of the image pixels. It then uses different combinations of those fractions to probe the behavior of the studied model and identifies the combination that can maximize the probability of a certain class. That combination of image fractions can therefore be considered as the reason leading the model

to the class. For instance, the authors explained why a 'Husky' image was misclassified as a 'wolf' image in their work. The extracted image fractions interpreted that the model considered the 'Husky' as a 'wolf' due to the white background of the image. From this interpretation, one may hypothesize that the misclassification is due to the high co-occurrence of white background with 'wolf' images (e.g., wolves in white snow). He/She can then verify the hypothesis and potentially improve the model by increasing the amount of 'wolf' images without white background (to make the classifier more robust).

Model Improvement. This objective targets to provide model improvements using visual analytics beyond what deep learning experts can do or have already conducted. We agree that improving DNN models should more rely on the algorithms, techniques and theoretical supports from deep learning experts. However, due to their inability in investigating all levels of details in the long-time model training process, they may not be able to effectively detect certain model deficiencies. As a complement, visual analytics could help in these cases. For example, Blocks [21] revealed the effects of the class hierarchy over the training of CNNs and took advantages of the hierarchy information as extra supervisions to improve the training of CNNs. DeepEyes [81] monitored the active level of different computational layers and neurons in a DNN over its training to detect degenerated neurons, and provide feedback to the design of the network architecture. Through this tool, domain experts can potentially introduce early interventions to optimize the structure of a DNN (e.g., changing the number of layers or the number of neurons per layer).

It is worth mentioning that these three objectives are not orthogonally independent. For example, the improvement of a DNN model may start from the model's deficiency detected over model diagnoses, and the diagnoses of the model will rely on the effective interpretations of the model's behaviors. Different visual analytics works may emphasize

certain objectives over others. Our three works in Chapter 3, 4, and 5 emphasize each of the three objectives respectively, i.e., *GANViz* emphasizes model interpretations; *DeepVID* focuses on model diagnoses; *DQNViz* targets more on model improvements. However, they all demonstrate different levels of model interpretations and diagnoses.

2.3.3 Two Exploration Strategies for DNNs

According to our study on the literature, a DNN model can be explored in two manners: (1) using only the input and output to probe the model's behavior without touching its internal structure; (2) tracking how data instances of interest flow through the model to expose all internal details of it. Both exploration strategies can accomplish the goal of exploring, interpreting, and diagnosing DNNs, and we call them the Black-Box and White-Box exploration, respectively.

The **Black-Box Exploration** uses only the input (i.e., training or test data instances) and output (e.g., predictions of DNN classifiers, loss values) of DNN models to probe the models' behaviors and provide insight into model interpretations and diagnoses [71, 86, 114]. As this approach does not need to touch the inner structure of DNNs (e.g., the neurons, activations), it is therefore *model-agnostic*. For example, Ming et al. [71] used model induction algorithms to infer a smaller and more interpretable model from a big classification model using the model's input and output. Their inferred model is a rule-based representation (or approximation) of the original model, and the derived rules are more easily understandable compared to the large number of inexplicable neurons of a DNN, or the complicated working mechanism of a SVM. They then designed a matrix-based visualization, i.e., RuleMatrix, to visualize those rules to help users interpret and diagnose the original model. As it can be seen, the Black-Box approach may not visualize all the four types of DNN data, but

may involve the visualization of other derived data (such as the inferred rules in this case). Another example of model-agnostic exploration is the Manifold [114]. Focusing on one data class, this visual analytics tool adopted a scatterplot to project input data instances onto a 2D space, in which the two predicted probabilities (for the class) from two different models were used as the horizontal and vertical coordinates. This way, the two classification models could be compared based on the agreement of their predictions, and the scatterplot was divided into four quadrants (e.g., the first quadrant shows data instances that both models agreed with each other). Extending to more data classes, Manifold employed multiple juxtaposed scatterplots (i.e., the small-multiple design) to present the agreement/disagreement between a model pair.

The **White-Box Exploration** interprets and diagnoses DNN models by directly visualizing the aforementioned four types of DNN data [21, 66, 81, 99, 113], as well as the interconnections between them. During explorations, users interact with different data portions and get findings related to the corresponding portions. Combining all the partial findings and integrating them with human intelligence usually provide valuable insight into the studied DNN models. For example, LSTMVis [99] employed a linked visualization of a PCP and multiple heat-maps. By revealing the dynamics in the hidden states of different input sentences using those views, users could open the black-box of LSTMs to identify what language patterns have been learned (e.g., noun phrases, verb phrases). For CNNs, Zeiler and Fergus [113] opened the CNN black-boxes by projecting the CNN extracted features back onto the input image space using deconvolutional networks. They conducted such projections layer-by-layer to investigate and compare the extracted features from different convolutional layers, which helped them a lot in better understanding and diagnosing CNNs. Specifically, the lower layers of a CNN usually extract basic shape or color information

from the input images, whereas the higher layers often identify more complicated features (which are the combinations of the basic features from lower layers), like the contour of a dog face. The White-Box exploration strategy is usually designed for a specific DNN model, and generalizing it to different models is usually limited by some model-specific settings, such as the model architecture (e.g., the type of layers, the number of neurons, etc.).

Our proposed two general model interpretation and diagnosis approaches (Section 1.4), namely, the Black-Box approach (i.e., *DeepVID*) and the White-Box approach (i.e., *GANViz* and *DQNViz*), are along the same line with these two exploration strategies. We demonstrate the detailed designs and requirements behind them, and show their effectiveness, in the following three chapters.

Chapter 3: *DeepVID*: Probe Supervised "Black-Boxes" with Knowledge Distillations

Deep Neural Networks (DNNs) have been extensively used in multiple disciplines due to their superior performance. However, in most cases, DNNs are considered as black-boxes and the interpretation of their internal working mechanism is usually challenging. Given that model trust is often built on the understanding of how a model works, the interpretation of DNNs becomes more important, especially in safety-critical applications (e.g., medical diagnosis, autonomous driving). In this chapter, we propose *DeepVID*, a ***Deep*** learning approach to ***V***isually ***I***nterpret and ***D***iagnose DNN models, especially image classifiers. In detail, we train a small locally-faithful model to mimic the behavior of an original cumbersome DNN around a particular data instance of interest, and the local model is sufficiently simple such that it can be visually interpreted (e.g., a linear model). *Knowledge distillation* is used to transfer the knowledge from the cumbersome DNN to the small model, and a deep generative model (i.e., variational auto-encoder) is used to generate neighbors around the instance of interest. Those neighbors, which come with small feature variances and semantic meanings, can effectively probe the DNN's behaviors around the interested instance and help the small model to learn those behaviors. Through comprehensive evaluations, as well as case studies conducted together with deep learning experts, we validate the effectiveness of *DeepVID*.

3.1 Introduction

Classification is a fundamental problem in machine learning, and there are numerous algorithms/models to solve this problem, e.g., naïve Bayes classifiers, Support Vector Machines (SVM) [27], and decision trees [88]. These models have been used extensively in different disciplines and achieved extraordinarily good performance. Recently, Deep Neural Network (DNN) classifiers demonstrated even better performance, and have made a breakthrough in the object recognition challenge of ImageNet via Convolutional Neural Networks (CNNs) [44, 59, 96].

However, the outstanding performance of the classification models does not lead to easy model interpretations or model trusts. Recently, people start concerning more about the trust and interpretability issues when deploying those accurate yet complex models, especially in safety-critical applications. For example, for self-driving cars, recognizing a stop sign as a speed-limit sign [31] will lead to a grave consequence. However, most of the classification models, especially DNN models, are considered as black-boxes and interpreting their internal working mechanism is usually challenging, as the data transformation in High-Dimensional (HD) space goes beyond humans' interpretation capability.

For image classifiers, model interpretation mostly aims at finding what pixels of an image contribute positively/negatively to which class probabilities [76, 86], and two types of interpretation techniques can be found from the literature. The first type attempts to open the black-boxes and is designed for specific DNN classifiers (e.g., CNN), such as guided back-propagations [98], layer-wise relevance propagations [18, 75], and class activation maps [116]. These techniques are limited to DNN classifiers and cannot be used if the network structure is unknown, which happens very often in real-world applications. The second type of techniques is model-agnostic, which treats the model to be interpreted as a

black-box and uses various approaches to probe and observe the black-box's behavior [33, 114]. One technique is to train a small model to locally probe and interpret the behavior of the original cumbersome classifiers at a specific data instance of interest, such as LIME [86]. The local model is trained using the perturbed neighbors around the data instance of interest, and the model is usually small and simple enough to be easily interpretable (e.g., linear regressors). Nevertheless, the existing ways of generating perturbed neighbors have fundamental limitations. First, perturbing a data instance in the input space is not efficient, as the input is usually high-dimensional and the combination of perturbed dimensions is massive. Second, the neighbors generated by directly perturbing a data instance in the input space may have no semantic meanings, leading to potential biases when being used to train the small model. Also, there are no accessible analysis tools integrating different pieces of the above interpretation pipeline.

In this chapter, we propose *DeepVID* (Figure 3.1), which brings together the power of ***Deep*** learning (i.e., knowledge distillation theories and generative models) and visual analytics to ***V***isually ***I***nterpret and ***D***iagnose image classifiers. Focusing on one image of interest, *DeepVID* trains a locally-faithful and explainable classifier (a small model) to mimic the behavior of an original cumbersome classifier (a big model). The training data of the small model are the localized neighbors of the interested image. Different from the existing model-agnostic approaches, *DeepVID* generates semantically meaningful training neighbors using deep generative networks, and the labels of the generated neighbors are obtained directly from the big model. Using the neighbors and their associated labels, the knowledge distillation mechanism [47] is then adopted to train the small model to effectively distill knowledge from the big model for interpretation. *DeepVID* is designed and implemented as a visual analytics system, and aims to help domain experts conveniently interpret classifiers

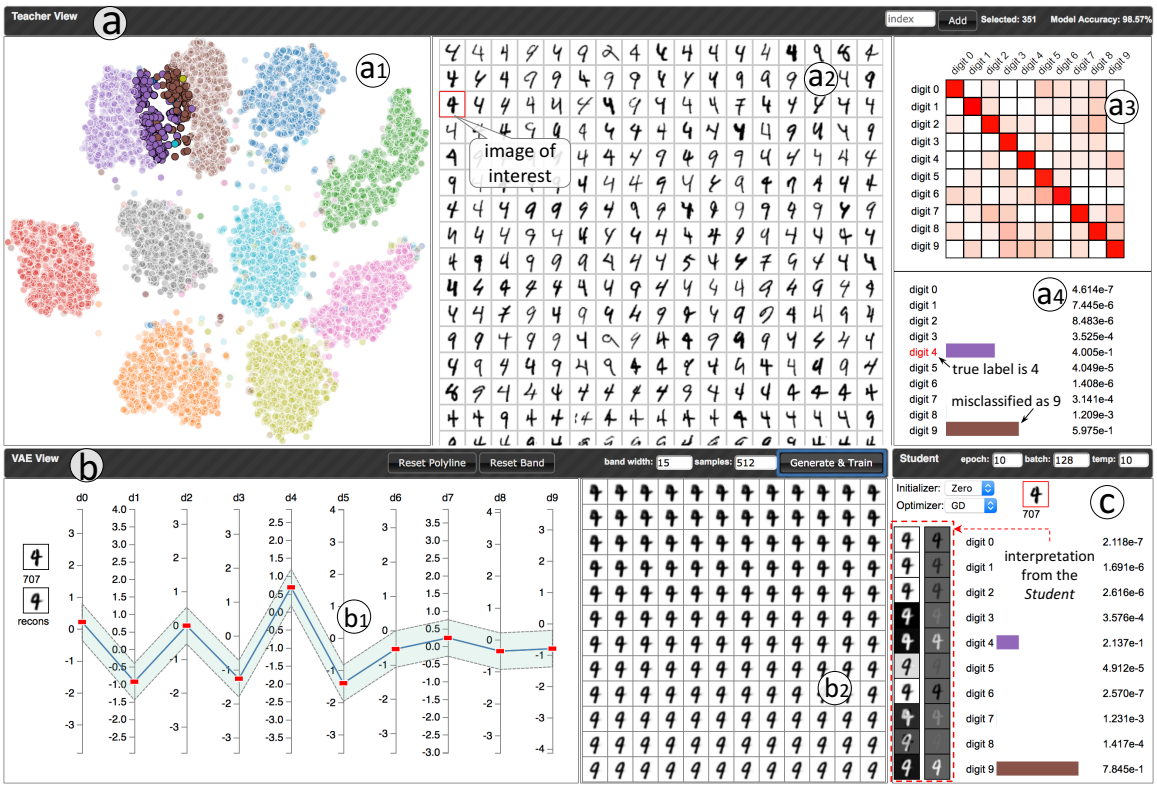

Figure 3.1: *DeepVID*. (a) Teacher view: explore test data and assess the *Teacher*'s performance with the t-SNE view (a1); Image-Grid view (a2); Confusion Matrix view (a3); Probability Distribution view (a4). (b) VAE view: explore the latent space with a parallel coordinates plot (b1); and the generated neighbors with the Neighbors view (b2). (c) Student view: visualize the trained *Student* for interpretation.

and diagnose their performance by dissecting why and how they succeed or fail in specific cases. Through comprehensive evaluations, we demonstrate the effectiveness of *DeepVID*.

To sum up, our contributions in this work include:

- We propose *DeepVID*, a model-agnostic interpretation framework that uses deep generative models (i.e., variational auto-encoders [54]) to generate local samples with semantic feature variance and adopts knowledge distillation for interpretation.

- We implement *DeepVID* as a visual analytics system, which empowers domain experts to interactively explore the feature space of their data and interpret their classifiers' behaviors.

- We validate the effectiveness of *DeepVID* through both qualitative and quantitative evaluations, as well as case studies conducted together with deep learning experts.

3.2 Existing Model Interpretation Solutions

Generally speaking, there are two types of existing solutions in interpreting classification models. The first type is model-specific and the second is model-agnostic. We briefly review them as follows.

DNN-Structure Aware Interpretation. There are three categories of techniques to interpret DNN classifiers using their network architecture (DNN-structure aware interpretation approaches). The *first* category is the gradient-based explanations (e.g., sensitivity analysis [76], guided back-propagations [98]), which compute the gradient of the network output (class probabilities) in respect of the network input (image pixels), layer-by-layer, and investigate how the output is produced from certain regions of the input. *Second*, the layer-wise relevance propagation approaches propose different policies [18, 75] to decompose the network output and redistribute them (layer-by-layer) back to the input to derive the positive/negative contribution of the input pixels. *Third*, the Class Activation Map (CAM) based approaches (e.g., CAM [116], Grad-CAM [93], Grad-CAM++ [23]) combine the extracted features (i.e., activations of the last convolutional layer) and the class-related weights (i.e., weighs of the last fully-connected layer) to relate class probabilities with input pixels. All the above interpretation approaches rely heavily on the architecture of

neural networks. However, if the architecture is unknown, which happens in most cases of real-world applications, they are no longer applicable.

Localized Model-Agnostic Interpretation [33, 114]. Another category of techniques interprets a classifier by approximating the classifier's local behavior around a particular data instance of interest with a simple interpretable model (e.g., linear regressions, decision trees). These techniques are model-agnostic, as only the input and output of the original classifier are used to train the simple model and the internal architecture of the classifier is not required. The most notable examples include LIME [86] and meta-predictors [33]. One common limitation of the techniques in this category is the way of generating training data for the small interpretable model. To better localize the training samples around the data instance of interest, they are often generated by perturbing the instance of interest *in the image space*. For example, LIME decomposes the image of interest into many fractions and considers different combinations of those fractions as local neighbors of the interested instance for training. The approach of meta-predictors blurs a particular region of interest and probes the original classifier's behavior. However, those perturbed neighbors (combinations of image fractions, partially blurred images) are not natural images and lack of semantic meanings, which may bias the training of the interpretable model. Also, these approaches perturb the data in the input space which is not efficient, as the dimensionality of the input is usually very high. Our proposed approach is also model-agnostic. However, we take advantage of the recent advances in deep generative networks to generate localized neighbors with semantic meanings. Our approach is more efficient as we use the generative model's *latent space* to sample neighbors, which is much smaller than the input image space.

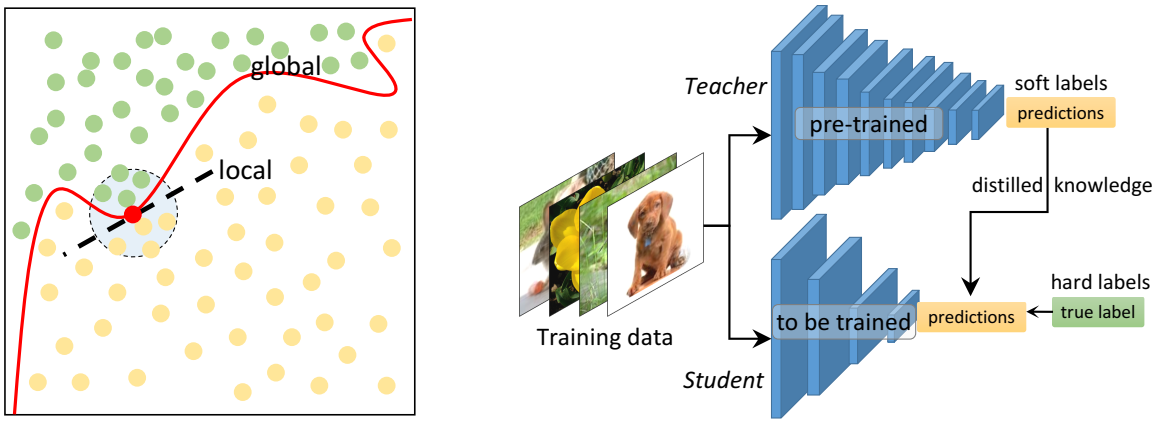

Figure 3.2: Left: a small model (the dotted line) trained with local samples can be used to approximate and interpret the global boundary of a big model (the red curve). Right: the knowledge distillation mechanism distills knowledge from a big model (*Teacher*) and uses the knowledge to train a small model (*Student*) for model compressions [47].

3.3 Background

Interpretation via Localization. The idea of localization is to approximate the behavior of a classifier at a point of interest using a simple interpretable model [86]. Figure 3.2 (left) shows an example: the data points from two different classes are colored in green and orange; a classifier's decision boundary is denoted by the red curve; and the red point represents the sample of interest. Although interpreting the red curve (i.e., the complicated global boundary of the classifier) is difficult, the classifier's local behavior around the sample of interest can be explained with a simple linear model (the black dotted line), which is trained using the samples in the shaded region (neighbors of the sample of interest). Our approach adopts a similar idea of local approximation to interpret and diagnose a complicated model, but improves the localization strategy with a deep generative approach to enhance the efficiency and accuracy.

Knowledge Distillation [47] was proposed to compress cumbersome models into light-weighted models for various purposes (e.g., simplifying the deployment of a model). The idea is to train a small model (*Student*) using knowledge distilled from a pre-trained cumbersome model (*Teacher*). A commonly used method is to match the logits from the two models, such as the example shown in Figure 3.2 (right). The *Teacher* is a pre-trained DNN, which takes images as input and outputs their class probabilities. The *Student* is also a DNN but contains much fewer layers, which can be trained using the same image inputs. The *Student*'s training loss contains two parts, which minimize the difference between the predicted labels and the true labels (i.e., *hard labels*), as well as the predictions from the *Teacher* (i.e., *soft labels*). The soft labels (relative probabilities of different classes) provide rich information about how the *Teacher* thinks of the input data. For example, a 'truck' image may have some probabilities to be misclassified as a 'car', but very small chances to be misclassified as a 'dog'.

Mathematically, we can denote an image dataset as $\{X, Y\}$ (X: images; Y: image labels). By feeding an image $x \in X$ into the *Teacher* and *Student*, we get two sets of logits $\boldsymbol{Z_t}$ and $\boldsymbol{Z_s}$. The prediction (i.e., probability distribution) from the *Teacher* and *Student* can be denoted as $\boldsymbol{P_t} = softmax(\boldsymbol{Z_t}/T)$ and $\boldsymbol{P_s} = softmax(\boldsymbol{Z_s}/T)$, where T is the distillation temperature controlling the entropy in $\boldsymbol{P_t}$ and $\boldsymbol{P_s}$ ($T=1$ by default). The `softmax` function is defined as: $softmax = \frac{e^{y_i/T}}{\sum_i e^{y_i/T}}$. The training loss for the *Student* is:

$$l_s = \alpha L_{hard}(\boldsymbol{P_s}, \boldsymbol{y}) + \beta L_{soft}(\boldsymbol{P_s}, \boldsymbol{P_t}), \tag{3.1}$$

where α, β are two coefficients, $\boldsymbol{y}$ is the true label (hard label) for x (a one-hot vector), and L_{hard}, L_{soft} are measured by cross-entropy. Increasing the value of T will increase the entropy in $\boldsymbol{P_s}$, and thus enhance the *Student* to learn the relative probabilities of different

classes from the pre-trained *Teacher*. However, if T is too large, the probability of irrelevant classes will also be over-emphasized.

In this work, instead of compressing models, we adopt the idea of knowledge distillation to distill and transfer knowledge between models for the purpose of interpretation. In the traditional model compression applications, the *Student* is usually required to share similar model natures (e.g., network structures) with the *Teacher*. In our case, however, as the purpose is to locally approximate but not fully mimic a *Teacher*'s behaviors, the *Student* can be much simpler and more explainable to serve our interpretation goal.

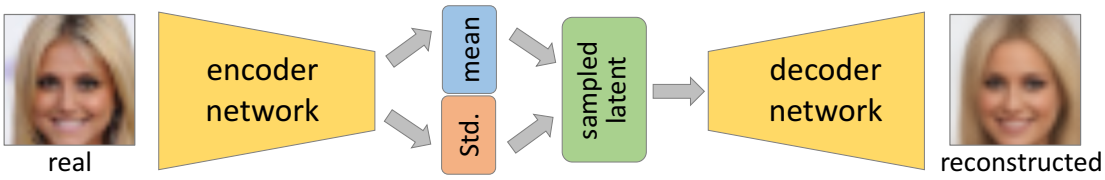

Figure 3.3: The architecture of VAE models, which is constituted of two sub-networks, an encoder network and a decoder network.

Variational Auto-Encoder (VAE) [54] is an unsupervised neural network model, which learns a latent representation of the training data and then reconstructs the data from the learned representation. It consists of two sub-networks of an encoder and a decoder (Figure 3.3). The encoder compresses the input $x \in X$ into a latent vector z, i.e., $z = encoder(x) \sim q(z|x)$; whereas the decoder reconstructs an image x' from z, i.e., $x' = decoder(z) \sim p(x|z)$. The training of VAE is conducted by: (1) minimizing the difference between x and x'; and (2) limiting the distribution of z to be a unit Gaussian distribution, i.e., $p(z) = \mathcal{N}(0,1)$. The training loss of a VAE is:

$$l(\theta,\phi) = -E_{z\sim q_\theta(z|x)}[\log p_\phi(x|z)] + KL(q_\theta(z|x)||p(z)), \tag{3.2}$$

where θ and ϕ are the trainable parameters. The power of VAE falls into two folds: (1) capturing the complicated data features from the input space and compressing them into a smaller latent space; (2) generating unseen samples with meaningful semantics in an on-demand fashion. Our method leverages these two strengths to efficiently generate local samples to probe a model's behavior.

3.4 Framework Overview

With the power of knowledge distillation and VAE, we propose a new model interpretation framework, i.e., *DeepVID* (Figure 3.4), following the idea of *interpretation via localization*.

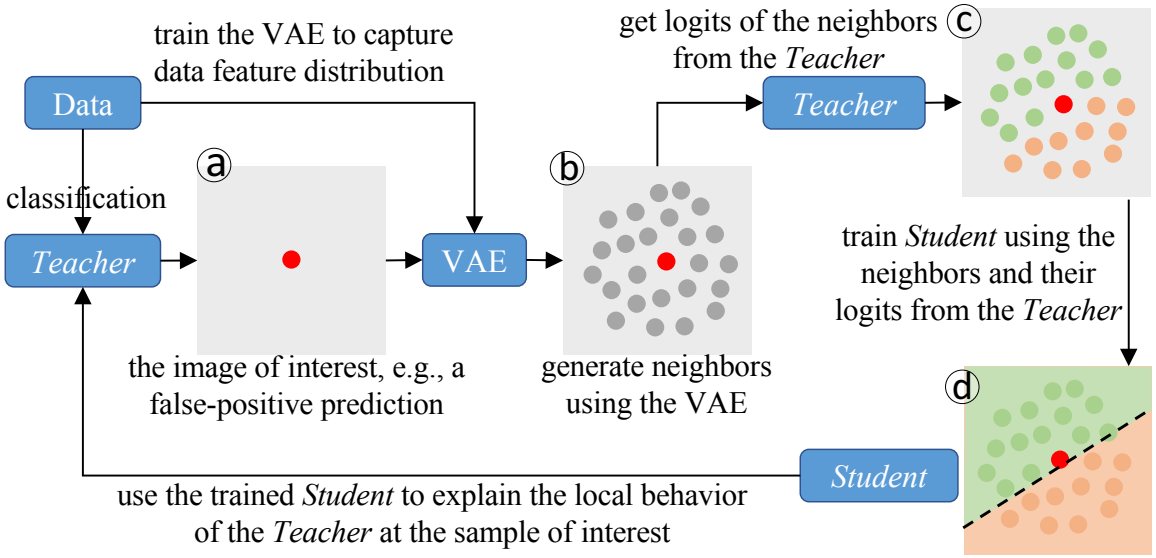

Figure 3.4: Overview of the *DeepVID* framework. The two boxes with "*Teacher*" are the same model, i.e., the model to be interpreted.

With a dataset and a classification task, a complex model (i.e., *Teacher*) is trained and to be interpreted. Also, a VAE model is pre-trained to capture the feature distribution

in the dataset. Now, given a data instance of interest identified based on the *Teacher*'s performance (e.g., a false-positive image), shown as the red point in Figure 3.4a, we use the pre-trained VAE to generate semantically meaningful neighbors around the instance (gray points in Figure 3.4b). Those neighbors are then fed to the *Teacher* to probe its behavior and distill its knowledge (i.e., deriving logits, Figure 3.4c). Next, we train a linear *Student* model using those neighbors and their logits from the *Teacher* (i.e., soft labels, note that those neighbors have no hard labels and only soft labels are used to train the *Student*) to mimic the behavior of the *Teacher* (Figure 3.4d). Finally, by visualizing the *Student*, we can interactively interpret the *Teacher*'s behavior on the data instance (e.g., why the image instance is misclassified).

Algorithm 1 Pseudocode for our *DeepVID* framework.

Require: *Teacher*: the classifier to be interpreted
Require: *VAE_encoder*, *VAE_decoder*: pre-trained VAE model
Require: x: data instance of interest (an image)
Require: *num_neighbors*: the number of neighbors needed
Require: T: temperature value to train the *Student*
Require: ***Ranges***: the value ranges of all latent dimensions
Require: *Sampling_Percent*: sampling within this percentage

```
lat_x = VAE_encoder(x) // derive the latent vector for x
band = Ranges × Sampling_Percent     // the sampling band
interval = band / num_neighbors
neighbors = [ ]              // training data for the Student
neighbors_logits = [ ]     // labels for the training data
for i=0; i<num_neighbors; i++ do
    lat_nbr=lat_x−band/2+i×interval  // sampled latent vector
    neighbor = VAE_decoder(lat_nbr)
    logit = Teacher(neighbor)
    neighbors.append(neighbor)
    neighbors_logits.append(logit)
end for
Student = Linear(neighbors, neighbors_logits, T).train()
Visualize(Student.weights, Student.weights × x)
```

Algorithm 1 explains our interpretation framework with more details. From the image of interest, x, we first derive its latent vector, ***lat_x***, using the encoder of the pre-trained VAE (line 1). Instead of sampling the input space (i.e., perturbing image x), which is performed in the previous works [33, 86], to generate neighbors, we sample the latent space (line 7) and use the VAE decoder to reconstruct neighbors from the sampled latent vectors (line 8). The sampling is centered at the latent vector of the interested image and within a user-specified percentage (*Sampling_Percent*) of the entire latent space (***Ranges***). This neighbor generation process is further explained with details in Section 3.6.2. With those generated neighbors, as well as their soft labels from the *Teacher* (line 9), we train the linear *Student* with temperature T (line 13). Finally, the trained *Student* (i.e., coefficients of the model) will be visualized (line 14) to interpret the *Teacher*'s behavior at x.

3.5 Visual Analytics Design Requirements

To implement the proposed interpretation framework as a visual analytics system, we worked closely with four deep learning experts and elicited the following three main themes of design requirements:

- ***R1: Visual understanding of the Teacher's behavior.*** The target system should provide a visual understanding of the test data and enable users to explore the prediction results from the *Teacher*. In detail, this theme needs the system to be able to:
  - R1.1: visualize and explore the feature space of the test data;
  - R1.2: show the *Teacher*'s overall performance over the test data, and the probability scores for a specified data instance;

  - R1.3: help users to easily identify the failure cases, such as hard samples (i.e., false-positives) and uncertain samples.

- ***R2: Flexibly explore the HD space of the generated neighbors.*** The quality of the generated neighbors is critical to train a high-quality *Student*, and thus, the system should enable users to:

  - R2.1: understand the semantics encoded in the latent space of the generator (i.e., the VAE). For example, for MNIST hand-written digits, some latent dimensions encode the boldness, and others encode the italic style of the digits;
  - R2.2: interactively adjust the generated neighbors, such that they are sufficiently close to the instance of interest;
  - R2.3: generate training samples between two instances of interest across class boundaries, e.g., generating samples that smoothly morphing from a user-interested digit 4 to a digit 9.

- ***R3: Interactively train and interpret the Student model.*** Tuning and understanding the performance of a *Student* should be an iterative process, and thus, the system needs to help users:

  - R3.1: understand the performance of the *Student* model and compare it with the *Teacher*'s performance;
  - R3.2: interpret the parameters learned by the *Student* model;
  - R3.3: interactively adjust the training hyper-parameters, such as the distillation temperature, optimizer, epoch number, etc.

3.6 Visual Analytics System: *DeepVID*

Following the design requirements, we designed and developed *DeepVID* with three modules, visualizing the information of the *Teacher* (R1), the VAE (R2), and the *Student* (R3) respectively.

3.6.1 Teacher View

The Teacher view (Figure 3.1a) demonstrates an overview of all data instances in test, along with the *Teacher*'s prediction performance on specified instances. It contains four juxtaposed sub-views.

The t-SNE view projects all test images into 2D and presents them as a scatterplot (Figure 3.1-a1), so that users can overview their clustering result and select groups of similar image instances on-demand (R1.1). Each point represents one image, and the point is colored by the true label of the image. The position of points is from the t-SNE algorithm [69], where the activations from the last hidden layer of the *Teacher* model are used as the t-SNE input. The projection provides an overview of all data samples with similar ones clustered together. Through lasso selections, users can select instances into the Image-Grid view (Figure 3.1-a2) to visualize them.

The Confusion Matrix view (Figure 3.1-a3) presents the confusion matrix of the *Teacher* derived from the test data (R1.2). The value at the ith row and jth column indicates the number of data instances of class i but predicted as class j. The cells along the diagonal are the correctly classified instances, and other cells are wrongly classified ones. The color from white to red encodes the cell values from small to large with the logarithmic scale. Clicking on any cell will select the images falling in that cell into the Image-Grid view, from where users can have a quick overview of the images (R1.3).

The Image-Grid view (Figure 3.1-a2) shows the selected images (either through lasso selections in the t-SNE view or by clicking cells in the Confusion Matrix view) without overlap. To effectively use the limited screen space, images with resolutions larger than 32×32 are scaled down to 32×32. This view is also scrollable if all images cannot be presented in the limited screen space. Hovering over any image in this view will highlight the corresponding point in the t-SNE view to help users build connections between views. Clicking on any image will update the Probability Distribution view (Figure 3.1-a4) to show the *Teacher*'s prediction on it.

The Probability Distribution view demonstrates the *Teacher*'s prediction for the selected image as a bar chart (Figure 3.1-a4, R1.2). The true label of the selected image is highlighted in red texts.

3.6.2 VAE View

The VAE view empowers users to explore the feature space (i.e., the latent space from the VAE) of the data, and presents an overview of the generated neighbors to reveal the feature trend in them.

Parallel Coordinates Plot (PCP) view. The latent vector for the image of interest is a HD vector (10D in Figure 3.1-b1). We present it with a PCP, which can effectively visualize HD data and allow users to interactively explore different dimensions of the corresponding HD space (i.e., the dragging interaction explained later). Each axis of the PCP shows the available range of the corresponding latent dimension and the red rectangle on the axis marks the value of the current latent vector on that dimension. The polyline connecting the red rectangles across all axes represents the latent vector of the currently selected image. Users can drag the red rectangle along each axis to see how the corresponding latent dimension

affects the visual appearance of the reconstructed image (i.e., the image with the label "recons" on the left of the PCP), in comparison with the original image, i.e., image 707 in Figure 3.1-b1 (R2.1).

The light blue band in the PCP shows the available perturbation range of the 10 latent dimensions. We generate n neighbors of the selected image by: (1) perturbing its 10D latent vector in the band to generate n 10D latent vectors; and (2) reconstructing from those latent vectors (using the VAE decoder) to generate n neighbors. In detail, the band is generated with a user-specified percentage value, which is 15% by default (it can be changed through the interface on the header of the VAE view). For each latent axis, a 15% range centered at the latent value of that axis is generated. Connecting the ranges from all axes forms the perturbation band. Next, n polylines are generated evenly within this band. Currently, we generate them by uniformly sampling 10D vectors within the band (see line 7 of Algorithm 1). This is not the only sampling method, but the most straightforward choice that meets our need. One can definitely resort to more sophisticated methods if necessary (e.g., exhausting all possible combinations of the values from the 10 axes).

If users have obtained a good understanding on the semantics of different latent dimensions (after interacting with the PCP view), they can manually brush each axis to specify the perturbation band, which will control and optimize the generated neighbors (R2.2).

The ***Neighbors view*** demonstrates m ($m<=n$, $m=144$ in Figure 3.1-b2) evenly sampled images (out of the total n generated neighbors), and those images are sorted from top-left to bottom-right following the increasing order of the corresponding latent vectors (i.e., the ***lat_nbr*** in line 7 of Algorithm 1). From the visual trend of those images, one can easily identify their feature distribution (R2.1).

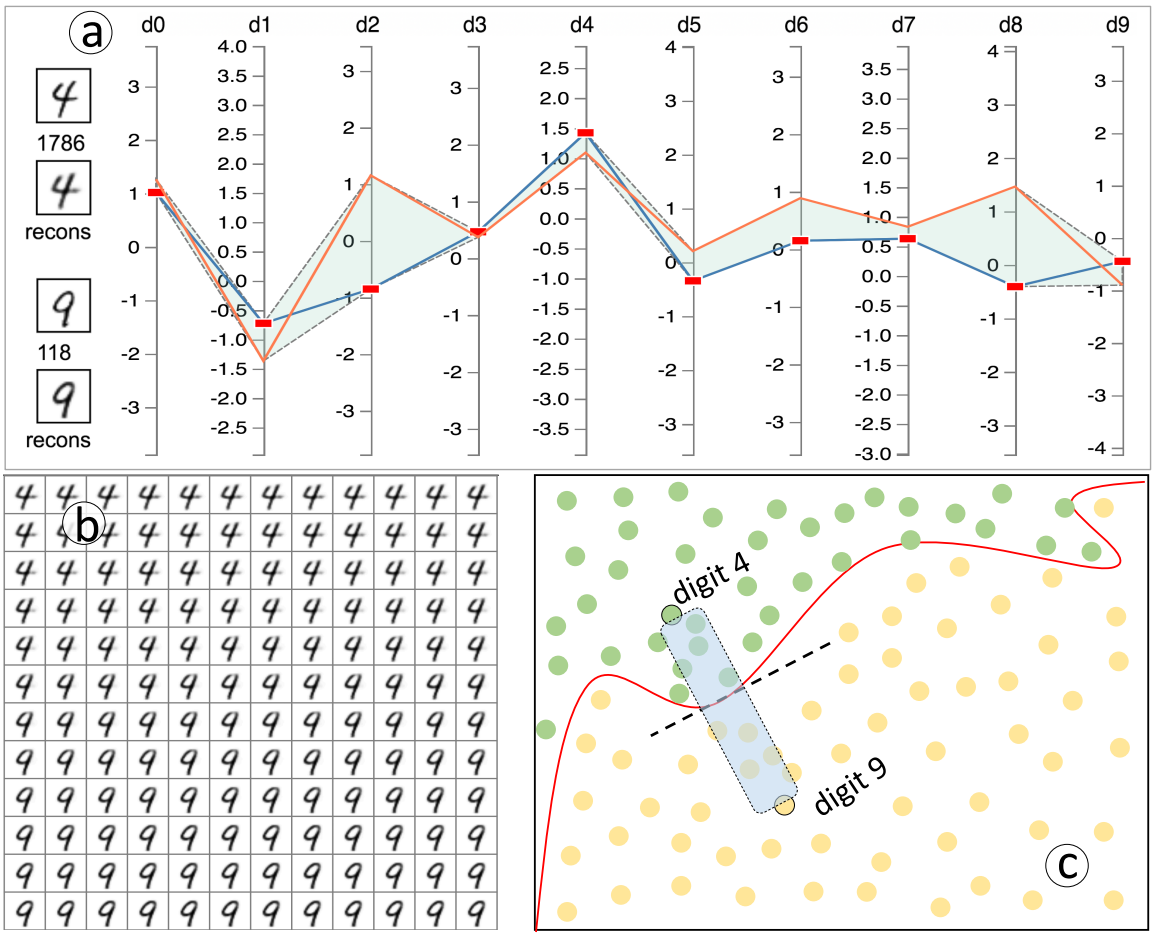

Figure 3.5: Morphing from a digit 4 (image 1786) to a digit 9 (image 118): (a) the perturbation band covers the two polylines in the PCP; (b) the 144 generated samples showing how image 1786 (the top-left image) is morphing to image 118 (the bottom-right image); (c) a semantic illustration explaining where the 144 samples are generated from.

Morphing between two images. One can also select two images of interest to investigate how the *Teacher* differentiated them (R2.3). In this case, *DeepVID* generates image samples between the two selected images. In Figure 3.5a, image 1786 and 118 (a digit 4 and 9) are selected, and their latent vectors are presented as polylines in blue and orange respectively in the PCP. We generate the perturbation band by connecting the covered ranges (of the two polylines) across all axes, and the perturbed latent vectors are generated by performing a

linear interpolation between the two latent vectors. For example, the Neighbors view in Figure 3.5b shows 144 generated neighbors, which present how image 1786 is smoothly morphing to image 118. Since the two images belong to different classes, the generated samples are images that across a certain point of the boundary between those two classes (the shaded region in Figure 3.5c). The *Student* trained on these samples will be able to explain how the two images are differentiated by the *Teacher*.

3.6.3 Student View and *DeepVID*'s Interpretation

The Student view helps to train the *Student* with customized hyper-parameters (R3.3). It also visualizes the trained *Student* (R3.2) and its predictions (R3.1). The top part of this view shows the selected image and the hyper-parameters. For example, in Figure 3.1c, the batch size is 128; the temperature is 10; the *Student*'s weights are initialized as zeros; and the optimizer is Gradient-Decent (GD).

The *Student* is a linear model, which takes an image (a set of pixels) as input and outputs a probability value. Mathematically, it can be expressed as $f = softmax(\sum_i w_i a_i + b)$, where a_i are the feature dimensions (pixels), w_i are the weights (coefficients), b is the bias, and f is the produced probability. To interpret f, two types of information should be visualized: (1) the learned weight for each feature/pixel, i.e., w_i; (2) the weighted feature, i.e., $w_i a_i$. Taking the MNIST data as an example, the input and output dimensions of the *Student* are 784 (pixels) and 10 (probabilities) respectively. Therefore, the weights are in the shape of 784×10, which can be considered as ten 28×28 weight matrices. Each matrix works like a template that can be multiplied with the 28×28 input image (pixel-wise) to weight different pixels. Figure 3.6 shows the detailed architecture of our *Student*, where the *Teacher* is the

LeNet [62] (a CNN). We use different line colors (blue, red, and green) to differentiate different weight matrices of the *Student*.

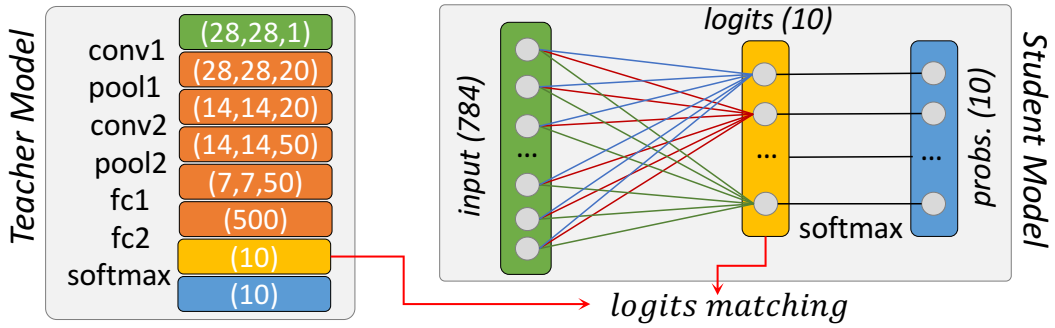

Figure 3.6: An example architecture of knowledge distillation. Left: the *Teacher* in this example is the LeNet. Right: the *Student* is a linear model, which is trained using knowledge distilled from the *Teacher* (i.e., the logits of different data instances from the *Teacher*).

The Student view uses images to visualize the trained *Student*'s weights and the weighted features. As highlighted in the red box of Figure 3.1c, the 10 weight matrices are normalized individually and visualized as the first column of images. The second column of images show the weighted features (i.e., multiplying individual matrices with the image of interest). Images in this column are normalized together to highlight how different pixels contribute to the 10 classes (values from small to large are mapped to colors from black to white). In Figure 3.1c, the selected image has high probabilities to be digit 4 and 9 due to the extracted white pixels in the second column of images. Hovering over any image in this view will pop up an enlarged version of that image to present more details.

The Student view also presents the *Student*'s predicted probabilities, which should be as close to the probabilities from the *Teacher* as possible, i.e., the bar chart in Figure 3.1c should match with the bar chart in Figure 3.1-a4. When the *Student*'s prediction is not

similar to the *Teacher*'s prediction, the explanations should not be trusted. In that case, one should try to adjust the hyper-parameters (e.g., modifying the temperature value or the sampling bandwidth) and re-train the *Student* to more accurately mimic the *Teacher*'s behavior. When training the *Student*, *DeepVID* will also print the *Student*'s training loss in the back-end (measured by cross-entropy), i.e., how well the *Student*'s predictions on the generated neighbors agree with the *Teacher*'s predictions, to monitor the training progress.

3.7 Case Studies with Domain Experts

We worked with four domain experts to explore the power of *DeepVID* using the commonly used image datasets and CNN classifiers. All experts have more than 10 years experience with machine learning and 4$\sim$6 years experience with deep learning.

3.7.1 Datasets and Network Structure

Three image datasets: MNIST [5], QuickDraw [4], and CelebA [1, 67] were used in our experiments. Figure 3.7 shows the typical images from each class of the three datasets. The MNIST dataset contains 65000 (train:55000; test:10000) hand-written digits in 10 classes (digit 0-9), and each image instance is a gray scale image with the resolution of 28×28. The QuickDraw dataset also has 10 image classes, which we extracted from the Google Quick-Draw game [4]. Each class in this dataset has 11000 image instances. As a result, there are 110000 gray scale images in total (train:100000, test:10000), and each image is also of size 28×28. To demonstrate the scalability of our approach, we have also worked with the CelebA dataset, which contains 202599 (train:192599; test:10000) human face images (in the resolution of 178×218) with annotations of 40 binary attributes per image. Those RGB images are first cropped into 148×148 and then rescaled to 64×64 for our usage. We randomly selected 3 binary attributes ("Male", "Blond Hair", "Glasses") to separate the

202599 images into eight (2^3) classes. For example, the first class is the images that have false values on all the three binary attributes.

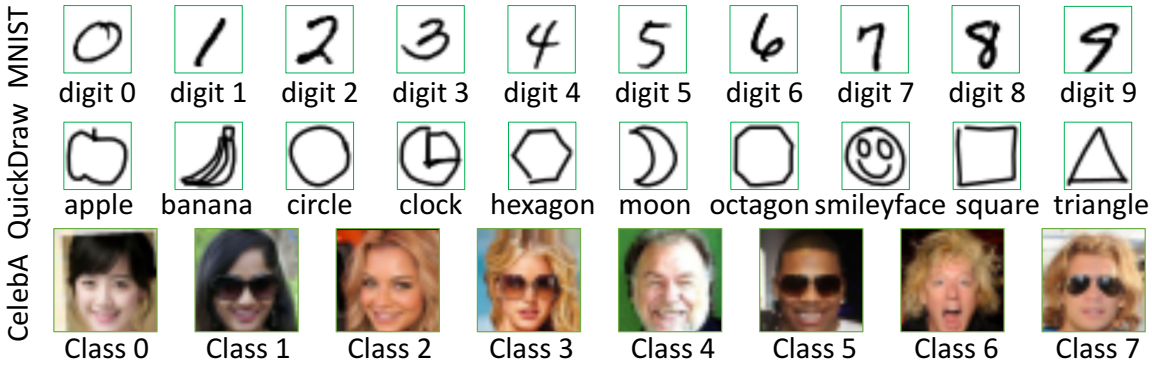

Figure 3.7: The experimental data: MNIST, QuickDraw, and CelebA.

Two CNN classifiers, LeNet [62] and VGG16 [96], were often used by our domain experts, and we used *DeepVID* to interpret their behavior on the above datasets. As shown in Table 3.1, the MNIST and QuickDraw datasets were used to train two LeNets and achieved accuracy 98.57% and 89.73% respectively. The VGG16 model is more complicated (deeper) than the LeNet. It was used to train the CelebA dataset and achieved accuracy 93.74% after 100 training epochs. For MNIST and QuickDraw, the feature space is relatively small. A VAE with two fully-connected layers [10] and a 10D latent space is able to capture the feature space (Figure 3.8, left). For CelebA, however, the feature space is much larger ($64\times64\times3$ dimensions). After several attempts, we finally adopted a four convolutional layers DFC-VAE [8] to capture the complicated feature space (Figure 3.8, right). DFC-VAE (Deep Feature Consistent VAE [49]) is a variant of the regular VAE. It improves the reconstruction performance by feeding both the input and reconstructed images into a pre-trained CNN model, and minimizing the difference between activations from all the

intermediate layers. Moreover, for CelebA, we used 100D latent vectors to preserve more information for better reconstructions.

Table 3.1: Details of different *Teacher* and VAE models.

Dataset	*Teacher*	Epochs	Accuracy	Generative Model	Latent	Epochs
MNIST	LeNet	20	98.57%	regular VAE	10D	150
QuickDraw	LeNet	20	89.73%	regular VAE	10D	150
CelebA	VGG16	100	93.74%	DFC-VAE	100D	50

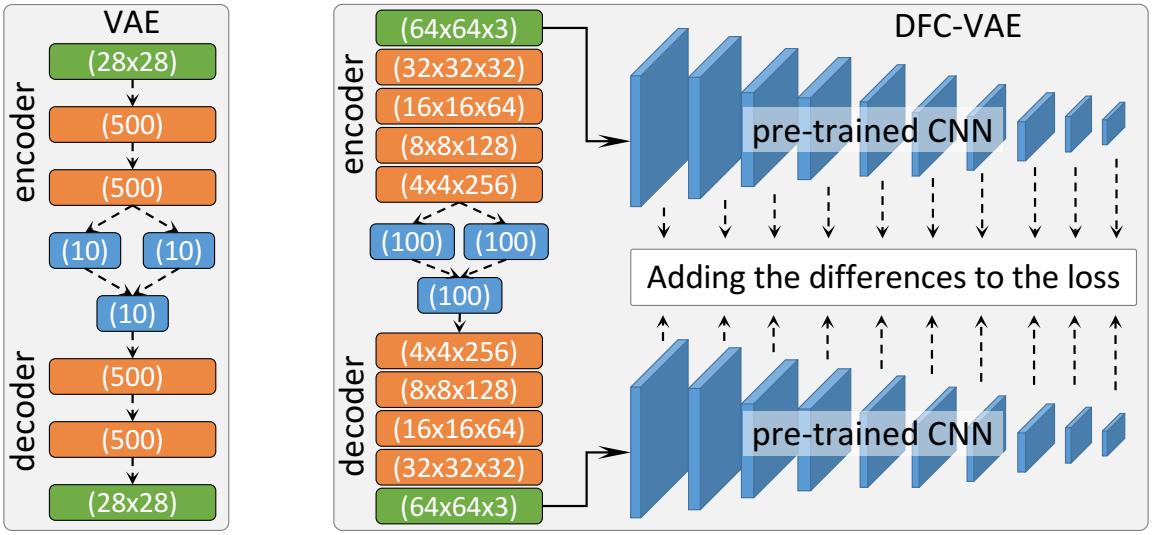

Figure 3.8: The structures of our VAE and DFC-VAE models.

3.7.2 Interpreting the *Teachers* with Domain Experts

Overview Teacher's Behavior. The experts first explored with the MNIST dataset to understand the performance of the LeNet (*Teacher*). From the t-SNE view (Figure 3.1-a1), which lays out all data instances using their last hidden layer representation from the *Teacher*,

the experts knew details on the *Teacher*'s performance beyond a single accuracy value (R1.1, R1.2). For example, from the clear separation of the 10 clusters, the experts became aware of the good performance of the *Teacher*. By hovering over different data points, they realized what digits different clusters represented for. They commented that digit 4 and 9 might be more difficult to be differentiated, as the purple and brown cluster (clusters for digit 4 and 9) are much closer than others. The Confusion Matrix view (Figure 3.1-a3) echoes the good performance (R1.2), as most of the data instances fall into cells along the diagonal (i.e., true-positives).

Explore Test Data and Identify Instance of Interest. Next, the experts selected data instances along the boundary of the purple and brown cluster in the t-SNE view to analyze the uncertain samples along the decision boundary between digit 4 and 9. The juxtaposed Image-Grid view (Figure 3.1-a2) visualizes the selected instances as images (R1.1). From there, the experts randomly clicked different images to explore their probability predicted by the *Teacher* (R1.2). For example, Figure 3.1-a4 shows the probability of a misclassified digit whose true label is digit 4 (highlighted in red) but misclassified as digit 9 with 59.75%. The experts were very interested in why the LeNet misclassified such a normal-looking digit 4 into digit 9 (R1.3). They then selected it for further interpretation.

Visualize the Latent Space and the Generated Neighbors. Figure 3.1b shows the VAE view for the selected digit 4 (i.e., image 707 shown on the left of this view). The latent vector of this selected digit is visualized in the PCP (Figure 3.1-b1), and the 144 evenly sampled neighbors (out of the total 512 neighbors) generated within the 15% sampling band are shown in the Neighbors view (Figure 3.1-b2). From the visual appearance of those neighbors, the experts saw the smooth transition from digit 4 (top-left) to 9 (bottom-right). Figure 3.9 shows our post-analysis on the probability of all the 512 generated neighbors

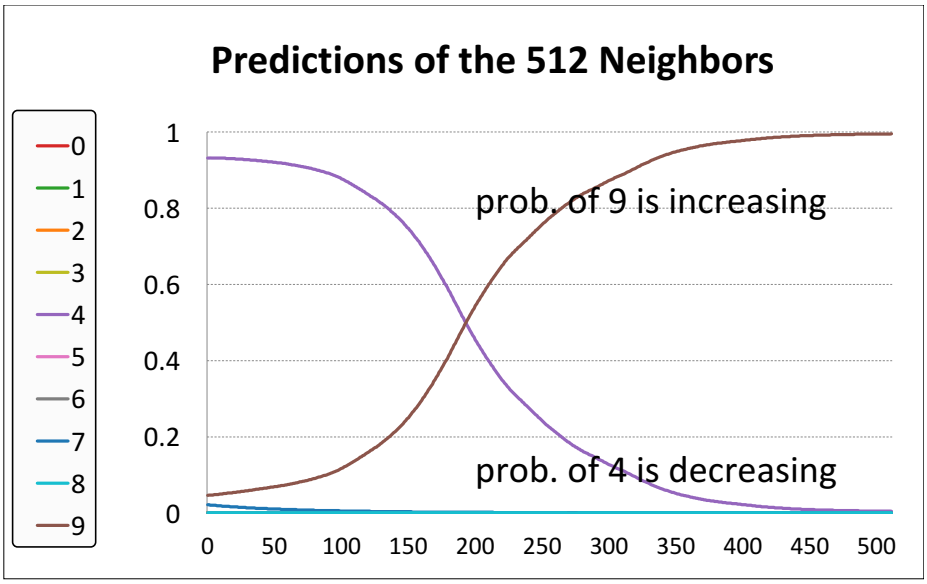

Figure 3.9: Probabilities of the 512 generated neighbors. Their probability of being digit 9 is increasing from left to right, whereas their probability of being digit 4 is decreasing.

predicted by the *Teacher* (the horizontal axis aligns the 512 neighbors, whereas the vertical axis shows their probabilities). From left to right, we can see that the probability of digit 4 is decreasing while the probability of digit 9 is increasing, indicating the VAE did a good job in generating neighbors from one side of the *Teacher*'s decision boundary (between 4 and 9) to the other side. Those neighbors' probabilities for other classes are very low.

Interpreting Misclassifications. Figure 3.1c shows the trained *Student* using the 512 generated neighbors. Initially, the experts set the temperature as 2 when training the *Student*. As the predicted probability from the *Student*, i.e., 97.24%/2.76% to be digit 4/9 (Figure 3.10a), is not very similar to the *Teacher*'s prediction, i.e., 40.05%/59.75% to be digit 4/9 (Figure 3.1-a4), they tried to adjust the temperature value. After some explorations, they found that the *Student* could achieve similar prediction performance when the temperature was 10 (Figure 3.10b, R3.1, R3.3). From the weighted features (the second column of images

in Figure 3.10b), the experts understood that the misclassification is because the highlighting of pixels in the top of digit 9 (comparing Figure 3.10-b1 and 3.10-b3) and the suppression of pixels in the little tail of digit 4 (comparing Figure 3.10-b2 and 3.10-b4, R3.2). The experts also trained the *Student* with different optimizers and initializers (R3.3), and the results are shown in Figure 3.10c-e. It can be seen that the *Student* achieved similar performance with the *Teacher* in all the three configurations (see the probabilities in Figure 3.10c-e). Although the trained weight matrices are different (due to different parameter update mechanisms), the weighted features (the second column of images) are similar.

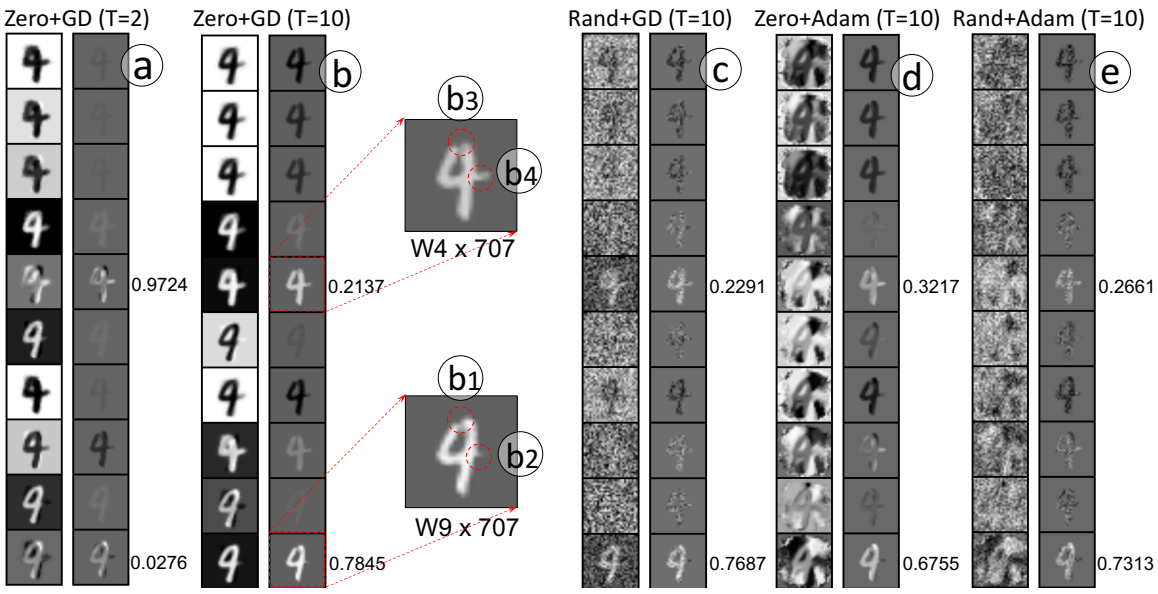

Figure 3.10: The effects of different hyper-parameters.

Additionally, the experts explored different misclassified images by directly selecting them from the Confusion Matrix view (clicking non-diagonal matrix cells). Figure 3.11 shows the case that the *Teacher* misclassified a bold digit 1 (image 9071) into digit 8. As shown in Figure 3.11c, the *Teacher* thinks the image has 41.78% to be digit 1 and

57.73% to be digit 8. Figure 3.11a shows the generated neighbors around image 9071, and Figure 3.11b shows the interpretation from the trained *Student* with the given hyper-parameters. The experts observed that the 10 trained weight matrices (the first column of images in Figure 3.11b) work like 10 masks, which can enhance or suppress certain pixels when being multiplied with the image of interest. For example, the weight W_8 enhances most pixels of the bold digit 1, but suppresses pixels in the two holes of digit 8. When multiplying it to image 9071, the bold digit 1 looks more like a digit 8, as the enlarged view shown in the bottom-right of Figure 3.11.

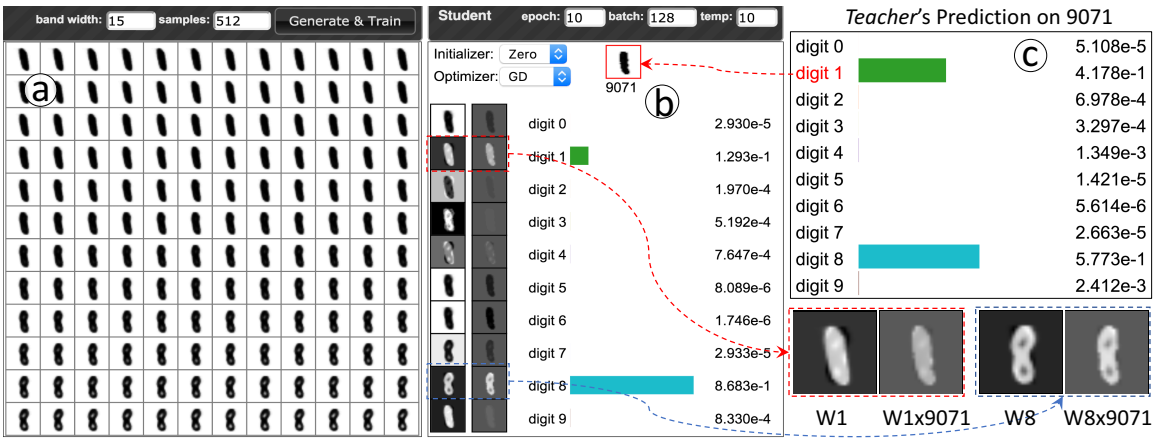

Figure 3.11: Explain why the selected digit 1 (image 9071) is classified as a digit 8.

Explore the Data Manifold and Control Neighbor Generation. In certain cases, the reconstructed image may not be similar enough to the image of interest (depending on the VAE quality). In those cases, users might want to explore the feature space to manually tune the sampling band and generate qualified neighbors. For example, in Figure 3.12a, a true-positive image of digit 7 (*Teacher*'s probability 99.83%) is selected for analysis. The reconstructed image, however, is more similar to a digit 9, so as to the generated neighbors

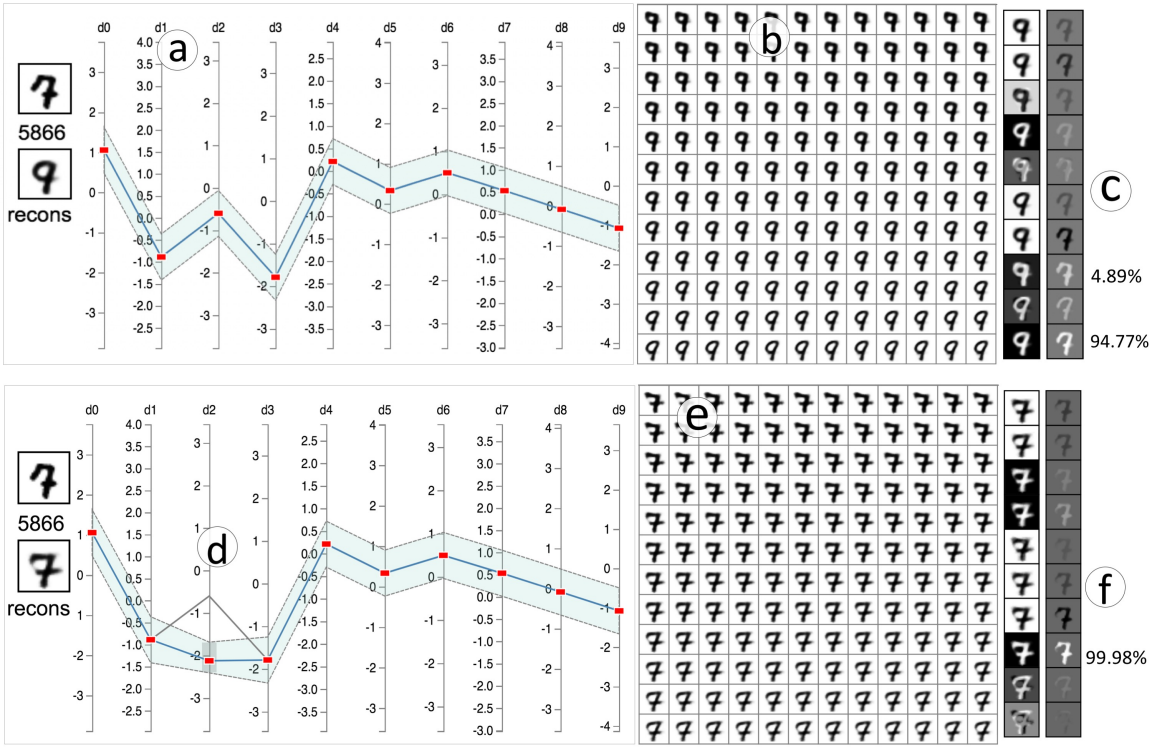

Figure 3.12: *DeepVID* allows users to flexibly brush the PCP axes to change the perturbation/sampling band, and generate better neighbors.

around it in Figure 3.12b. When feeding those neighbors to the *Teacher* (to derive logits), the *Teacher* recognized most of them as digit 9. As a result, the trained *Student* incorrectly predicted the image of interest as a digit 9 with probability 94.77% (Figure 3.12c). The interpretation from the *Student* cannot be trusted, as the behavior of the *Student* does not match with the *Teacher*.

To help the *Student* better mimic the *Teacher*'s behavior, the experts used *DeepVID* to explore the feature space of the data and generate better neighbors (R2.1, R2.2). As shown in Figure 3.12d, the experts dragged the red rectangle on each latent dimension to see its effects to the reconstructed image. From the exploration, they found that adjusting "d2" made the reconstructed image more similar to the image of interest (image 5866). As

a result, they manually brushed a region on "d2" to change the sampling band, and the reconstructed neighbors from the new band are shown in Figure 3.12e. By training the *Student* with those new neighbors, the *Student* could produce almost the same prediction with the *Teacher* (Figure 3.12f), and the *Student*'s interpretation is more meaningful.

Differentiate Two Images of Interest. As explained in Figure 3.5, *DeepVID* allows users to select two images and interpret how the classifier differentiated them (R2.3). We worked with the experts to exploit this function using the QuickDraw dataset. Figure 3.13 shows how the classifier (LeNet) differentiated a triangle from a square (image 4167 and 7386). First, the samples that morphing from the triangle to the square are generated (Figure 3.13a), and they are then fed to the *Student* to train the 10 weight matrices (the first column of images in Figure 3.13b). The triangle and square are the 9th and 8th class in QuickDraw. So, W_9 and W_8 are more import (Figure 3.13c).

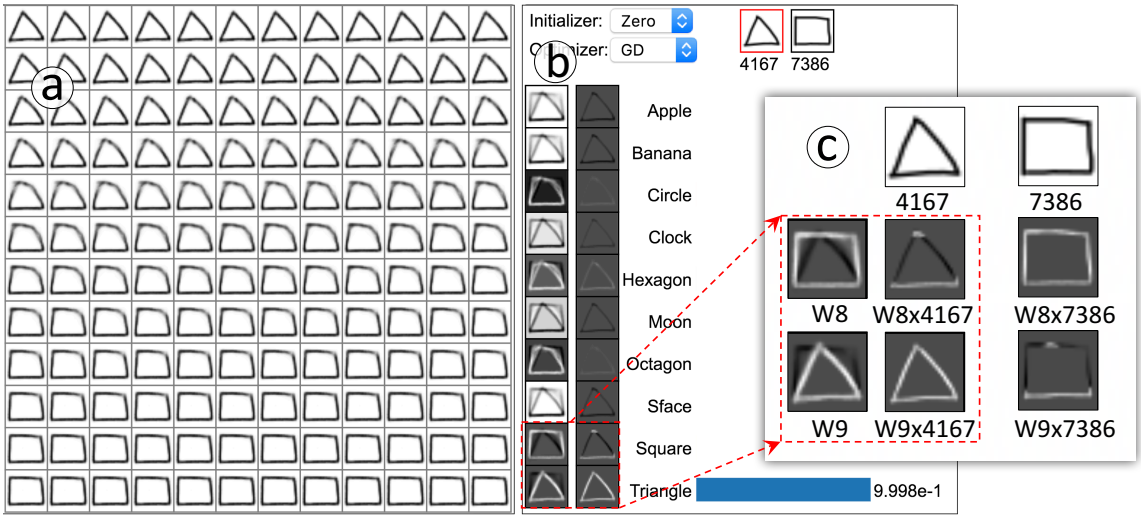

Figure 3.13: How a triangle and a square are differentiated.

We found that W_9 enhanced the three edges of the triangle and suppressed the top three edges of the square; whereas W_8 enhanced the four edges of the square and suppressed the top two edges of the triangle. When multiplying W_8 with the triangle image ($W_8 \times 4167$), only the bottom edge was extracted and the probability of being a square was very low. However, when multiplying W_9 with the same image ($W_9 \times 4167$), all three edges of the triangle were extracted and the probability of being a triangle was very high. On the other hand, when multiplying W_8 with the square image ($W_8 \times 7386$), all four edges of the square were extracted and the probability of being a square was very high. However, when multiplying W_9 with the square image ($W_9 \times 7386$), only the bottom edge was extracted, and consequently, the probability of being a square was very low. The experts observed the weight matrices were like masks that only allow pixels in certain regions of the input image to pass through.

Explore Larger Datasets. We have also explored the more complicated CelebA dataset with the experts. Figure 3.14 shows an example of interpreting how image 8170 is classified as "Female, Blond Hair, No Glasses" (class 2). The image in comparison (image 1682) is from class 0 ("Female, No Blond Hair, No Glasses"), and the feature variance of the two images is the existence of the blond hair or not. The Neighbors view on the left shows the images generated between the two images and we can see the smooth transition from the face with black hair to the face with blond hair. The enlarged second weight matrix (learned by the *Student*) works like a mask, which allows the blond hair from image 8170 to pass through it and blocks the rest pixels. This interpretation made the experts feel confident in trusting the classifier as the hair features were in use.

When exploring the CelebA dataset, the experts also found one limitation in terms of scaling *DeepVID* to larger datasets with higher-dimensional feature spaces. For example,

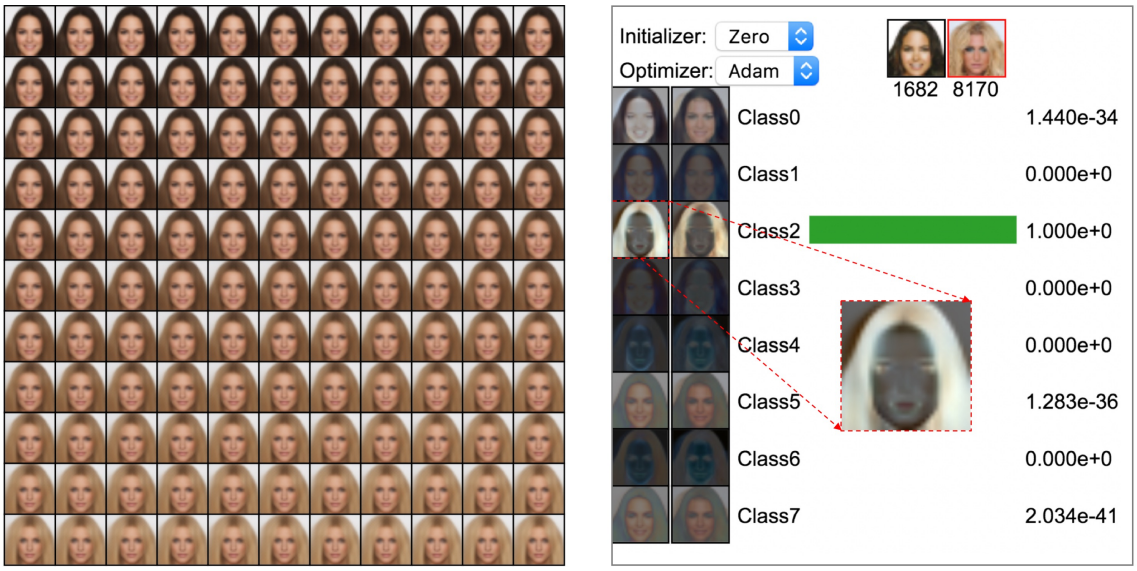

Figure 3.14: *DeepVID* identifies the blond hair feature in image 8170.

when differentiating a face image with glasses and a face image without glasses, if those images also have feature variances in other regions (e.g., one with hat and one without), those variances could disturb the training of the *Student* from focusing only on the class-sensitive features. We discussed this limitation with the experts and brain-stormed potential remedies of using advanced generative models to better control the neighbor generation process.

3.7.3 Domain Experts' Feedback

We conducted open-ended interviews with the four deep learning experts (E1∼E4). All experts gave their positive feedback on both the interpretation framework and visual analytics system of *DeepVID*, towards helping them interpret and diagnose their models. Their comments and suggestions are summarized as follows.

First, we observed that the experts' exercise with *DeepVID* deepened their understanding and confidence on the approach of local approximation for model interpretation. E1

explained that the case studies "*really helped me understand how and why the local approximation works*", after exploring the visual patterns of the generated neighbors around the interested data point, as well as the corresponding prediction results from both the DNNs and the linear explainer. E3 commented that "*this (DeepVID) is a strong case showing what a weak leaner (the linear explainer) can do (fitting local patterns) and cannot do (capture the overall data patterns)*".

All experts agreed that the visual explanations provided by the *Student* are "*insightful and actionable*". E1 expressed that "*the weighted features are very straightforward to show how the meaningful pixels contributed to the predictions*", and "*they are even more important than the trained weights (coefficients)*". E4 commented that "*this (DeepVID) helps me re-think the assumption about feature distribution, and we do need more data on some rare cases*". E3 was inspired to "*adopt a boosting method that classifies easy instances first and then works on the hard ones*", after observing that some contributing features of the misclassified images were also hard to be captured by the original models.

Notably, the VAE view attracted the most attention from the experts and raised some thoughtful discussions. One observation resonated among all experts is that the neighbor generation process is "*critical to the interpretation performance*", and they commented that "*sampling from the latent space (instead of the input space) is a nice trick*". The experts "*enjoyed*" exploring the VAE latent space and observing the reconstructed images to make sense of the semantics encoded in the latent space. E3 even developed his own sampling strategies with the latent space in an iterative fashion and commented that "*a more efficient way to generate samples is to fix some features, which did not contribute to the prediction in my previous exploration, and then adjust other dimensions*". E1 also suggested other

efficient sampling strategies by "*identifying some correlated latent dimensions first to shrink the sampling space*".

The experts also offered insightful suggestions. E3 discussed the feasibility of a hybrid approach that combines both model-agnostic and model-specific techniques. He commented that "*DeepVID cannot offer insight into improving the neural network structures*", and it may be helpful to have "*a layer-by-layer local approximation to show interesting findings on layer designs, such as different choices of filter sizes, activation functions, etc.*" E2 suggested to provide more flexible exploration to understand the feature space of the generated samples, and help users make sense of the semantics encoded in the latent space with some representative samples.

3.8 Evaluation and Comparison

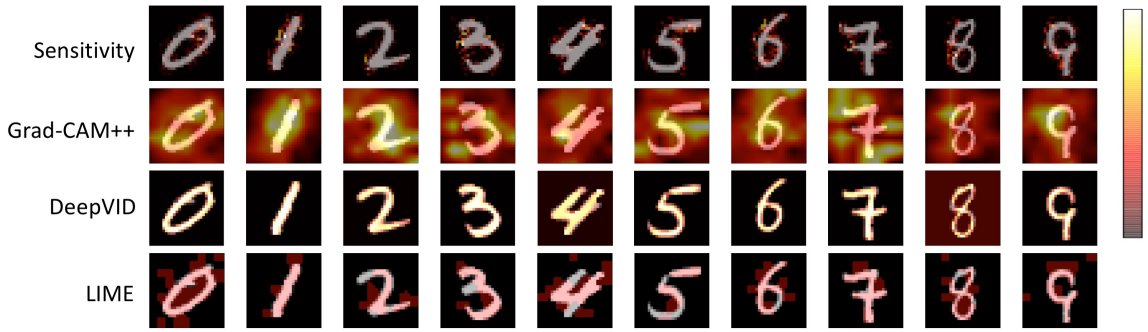

Figure 3.15: Qualitative comparisons by overlaying the pixel-importance map (derived from different approaches) onto the image in analysis.

Qualitative Evaluation. We compared the interpretation generated from *DeepVID* (i.e., the second column of images from the Student view) with three existing solutions: sensitivity analysis [76], Grad-CAM++ [23], and LIME [86]. When interpreting the behavior of an

image classifier, most approaches generate a map that colors individual pixels based on their importance to the final predictions (of the corresponding class). By overlapping the map with the image of interest, we can visually quantify if the map really captures the class-sensitive features. Following this, we randomly selected 10 images from the 10 classes of MNIST, and generated their pixel-importance maps (Figure 3.15). The foreground and background of the original MNIST images are shown with gray and black color respectively, whereas the pixel-importance maps are overlaid on those images. For sensitivity analysis, Grad-CAM++, and *DeepVID*, the color map [black-red-white] reflects pixels with the increasing importance. LIME only outputs binary results, i.e., positive to the class or not, and the positive pixels are colored in red. From the comparison, we can see that *DeepVID* can more accurately extract the gray pixels (the foreground strokes) of different digits.

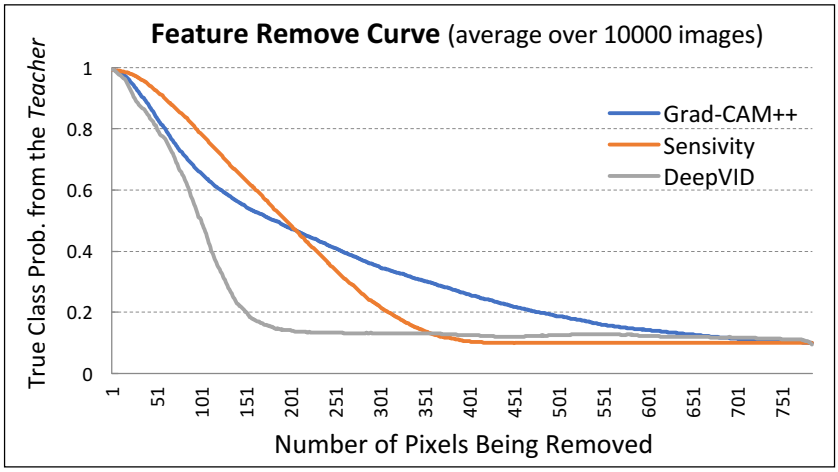

Figure 3.16: Quantitative comparisons using feature-remove curves (averaged over curves for all the 10000 images from the test data).

Quantitative Evaluation. To eliminate the potential cherry-picking comparisons and have a more rigorous evaluation, we conducted the quantitative evaluation using feature-remove curves. This comparison is conducted by sequentially removing the most important pixel from an image based on its order in the pixel-importance maps derived from different approaches, and feeding the modified image into the *Teacher* to see its predicted probability. The horizontal axis of the feature remove curve is the number of pixels being removed, whereas the vertical axis is the *Teacher*'s probability (for the true class). We generate such a curve for each test image in the MNIST dataset and average all those curves (10000 curves) derived from individual methods, as shown in Figure 3.16. A sharper drop at the beginning of the curve means more important features are captured and removed, indicating a more accurate interpretation. The curve trend in the later stages (e.g., after 200 pixels being removed) is not important, as the image has been modified significantly. From the comparison, the curve of our method drops the fastest (before 200 pixels being removed). The sharper drop also reflects the consensus between the *Student* and *Teacher* on class-sensitive pixels. LIME is not compared here, as it only outputs binary results (i.e., no order for the pixels).

3.9 Discussion, Limitations, and Future Work

Latency on Training the Student. *DeepVID* is implemented using Python and Javascript, and the Flask library [3] is used to enable the communication between them. The online training of the *Student* is conducted on the Python side using TensorFlow. As the *Student* is simple, its training latency is usually manageable. For MNIST and QuickDraw, the latency of generating 512 neighbors and training the *Student* for 10 epochs is less than 0.5 second on a modern laptop. For CelebA, the same experiment costs around 19 seconds on average.

Dynamically revealing the training progress (e.g., a progress bar) could improve users' experience, especially when generating more neighbors or working with larger datasets.

Hyper-Parameters Tuning. When training the *Student* with *DeepVID*, users need to tune a few hyper-parameters: the perturbation bandwidth, the number of generated neighbors, the number of training epochs, the batch size, the distillation temperature, the type of weight initializer and model optimizer. The default settings for them are: 15%, 512, 10, 128, 10, the random initializer, and the GD optimizer, respectively. These default values work well in most cases of our explorations, except that the temperature value needs to be adjusted frequently. To alleviate the issue of manually adjusting these parameters, some hyper-parameter tunning approaches can be adopted in the future. For example, a naïve grid search algorithm could be very helpful, considering the rather small searching space of the hyper-parameters in *DeepVID*.

Neighbors Generation. There are two categories of neighbor generation approaches in general. The first category generates neighbors through perturbations in the original data space, like LIME and *k*NN (searching neighbors in the original data space). This group of approaches can be easily conducted, but the number of feature dimensions is usually large and the generated neighbors may not be semantically meaningful. The second category of approaches encodes the original data into a new space (e.g., a latent or frequency space) and generates neighbors through perturbations in that space. *DeepVID* is in this category, where it uses a VAE to build the mapping between the original and the latent space. Usually, the latent space is much smaller and the latent dimensions are semantically meaningful. Thus, the neighbor generation process is more controllable. However, getting a high-quality data generator is the bottleneck for this category of approaches. With the clear pros-and-cons of these two categories, a hybrid solution seems to be promising. For example, one can

perform kNN using the latent vectors of all images (rather than the raw pixels). The latent vectors of the identified kNNs can also be used to guide the perturbation process in the latent space to generate new neighbors.

Large Latent Space. In certain cases, the dimensionality of the latent space could be very high (e.g., it is 100 when working with the CelebA dataset), which will bring challenges when perturbing and exploring the latent space. Working with domain experts, we have thought of two potential directions to address this problem. The first one is to resort to more advanced generative models (e.g., progressive GAN [53]) to derive latent dimensions with more stable and controllable semantics and group the dimensions based on their semantics. The second direction is to enhance the PCP interface, so that it can progressively learn users' preferences when exploring the latent space and automatically adjust the positions of the control points on different PCP axes. One such successful attempt has been proposed in color enhancement of images [57].

One ***limitation*** that we planned to address in the future is the scalability of *DeepVID*. In some sophisticated real-world applications, the number of input images, latent dimensions, and output classes could be far more than what we have demonstrated. In those cases, some *DeepVID* views will have to be redesigned to satisfy the demanding need. Another limitation of our approach is that it requires a high-quality pre-trained VAE model on the target dataset to provide good interpretations. For example, we have also worked with the CIFAR10 dataset [58] with the domain experts. However, we were not able to train a high-quality VAE, even using DFC-VAE [49], as the number of training images for each class of CIFAR10 is too small (i.e., only 5000 images per class), but the feature space is relatively large, i.e., each image has $32\times32\times3$ dimensions (32×32 pixels with the red, green, and blue channels). Figure 3.17 shows an example of interpreting a 'cat' image. The

VGG16 model (the *Teacher*) predicts the image to be a 'cat' with 72.45% and a 'dog' with 27.49% (Figure 3.17a); but the *Student* predicts the image as a 'deer' with almost 100% (Figure 3.17c). Adjusting the hyper-parameters could not improve the performance too much. The fundamental problem is that the training data of the *Student* model are not really localized neighbors for the image of interest (due to the poor performance of the pre-trained DFC-VAE model), as shown in Figure 3.17b. The semantic illustration in Figure 3.17d provides more explanations. The image of interest is on the class boundary between 'cat' and 'dog', but the generated neighbors (samples in the shaded region) are in the area of the 'deer' class. When feeding the generated neighbors to the *Teacher*, the *Teacher* will assign them with high probabilities to be 'deer' (soft labels). As a result, the *Student* can only distill the knowledge that the perturbed neighbors are 'deer', which leads to the wrong interpretation.

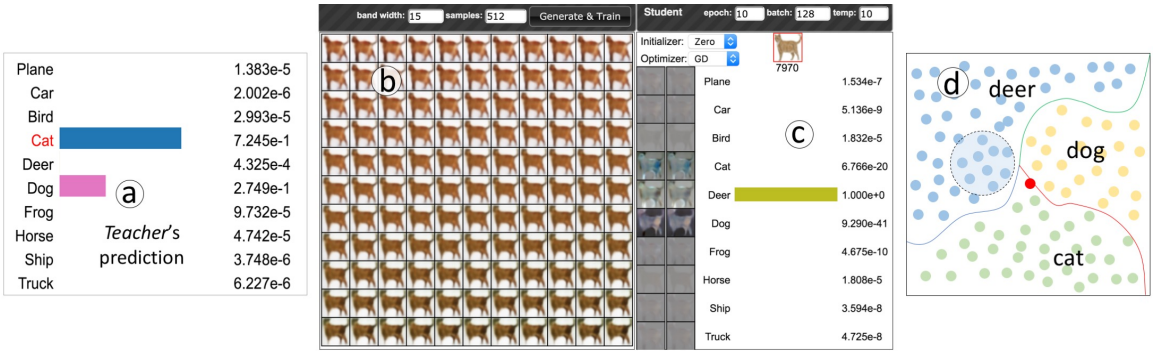

Figure 3.17: The pre-trained VAE on CIFAR10 cannot capture the complicated feature space with the insufficient number of training samples.

More Advanced Interpretations. Apart from interpreting the behavior of image classifiers, there are emerging requirements on interpreting and diagnosing more advanced models. We would like to extend *DeepVID* towards two interesting directions in the future. One is

to adapt *DeepVID* to interpret adversarial attacks [101]. Generating adversarial features to probe a model's behavior could reveal how vulnerable the model is, and thus further improve it from there. The other direction is to apply *DeepVID* for model comparisons. Domain experts usually have multiple candidate models for an application. With *DeepVID*, they can train smaller explainable models to better interpret and compare those candidates, and thus, making proper decisions for their application.

Other Classifiers and Other Data Types. We focus on interpreting DNN classifiers in this work. However, as *DeepVID* does not rely on the classifiers' internal structure (only the input and output are needed for knowledge distillation), other non-DNN classifiers (e.g., SVM, random forest) could be plugged into it for interpretation. Apart from extending *DeepVID* to non-DNN classifiers, we also plan to explore *DeepVID* with other types of data in the future. For example, we plan to work with text data to investigate which word in a particular sentence influences the sentiment of the sentence. This is very similar to examine which pixel of an image influences the classification result of the image.

3.10 Conclusion

In this work, we propose *DeepVID*, a visual analytics system that takes advantages of two deep learning solutions to perform visual interpretation for image classifiers. The first solution of knowledge distillation guided us to train a simple interpretable model using the knowledge distilled from a cumbersome classifier to interpret the classifier itself. The second deep learning solution of generative models helped us generate semantically meaningful neighbors around the data instance of interest, which can effectively probe the behaviors of the cumbersome model and train the small explainer. By virtue of the interactive power of

DeepVID, we conducted thorough studies with domain experts, compared it with existing model interpretation solutions, and validated its effectiveness.

Chapter 4: *GANViz*: Open the "Black-Boxes" of Unsupervised Deep Generative Models

Generative models bear promising implications to learn data representations in an unsupervised fashion with deep learning. Generative Adversarial Nets (GAN) is one of the most popular frameworks in this arena. Despite the promising results from different types of GANs, in-depth understanding on the adversarial training process of the models remains a challenge to domain experts. The complexity and the potential long-time training process of the models make it hard to evaluate, interpret, and optimize them. In this work, guided by practical needs from domain experts, we design and develop a visual analytics system, *GANViz*, aiming to help experts understand the adversarial process of GANs in-depth. Specifically, *GANViz* evaluates the model performance of two subnetworks of GANs, provides evidence and interpretations of the models' performance, and empowers comparative analysis with the evidence. Through our case studies with two real-world datasets, we demonstrate that *GANViz* can provide useful insight into helping domain experts understand, interpret, evaluate, and potentially improve GAN models.

4.1 Introduction

Generative models bear promising implications to learn data representation in an unsupervised fashion. Generative Adversarial Nets (GAN) [38] is one of the most popular

frameworks in this field. Since the GAN idea was first proposed by Goodfellow et al. [38] in 2014, numerous extensions have been proposed (e.g., DCGAN [83], InfoGAN [24]) and they have successfully demonstrated the powerful capability of the GAN framework. The basic idea of GANs is to set up two deep neural nets to compete against each other with opposite target functions. One neural net, called Generator (G), generates fake data and uses them to fool the other neural net, called Discriminator (D), who learns to discriminate fake data from real data. These two nets are trained against each other, and to the end, G develops the capability of generating data that are amazingly similar to real data, and D learns the data representations in an unsupervised fashion. Researchers can further use the trained G and D for different purposes. For example, well-trained Gs can generate data in fields where data is not always sufficient [63]. Ds are often used as pre-trained parameters for later fine-tuned trainings [30].

Despite the promising results from GANs, optimizing, interpreting, and evaluating such models remain challenging to domain experts [37, 112]. First, the adversarial process is complicated and has no theoretical guarantee of convergence so far, which presents challenges for model optimizations. Second, it is a non-trivial task to interpret the details inside GANs during trainings. For example, what features does D learn to differentiate real/fake data? what patterns do the results generated by G have, and how do the patterns evolve over time? Finally, model evaluation is also quite challenging. For example, domain experts are eager to know: is the network architecture an evenly matched game for both D and G? Is G learning data distributions only in a local feature space (i.e., the mode collapse issue [37])? However, these questions cannot be directly answered by model statistics (e.g., losses, activation means, parameter gradients) captured from conventional approaches.

One attempt to alleviate these issues comes from visual analytics. There are some visualization tools (e.g., TensorBoard [6]) to understand the training statistics of general neural nets. Also, an increasing amount of model-specific visualizations have been proposed recently, including MLP visualization [105], convolutional network visualization [66, 112, 113], and seq-to-seq model visualization [52, 99].

However, few visual analytics tools have been proposed to help domain experts understand, interpret, and evaluate GANs. The most recent work in this thread is the DGM-Tracker [65]. While this tool is very helpful to understand the overall training process and diagnose potential training failures for *general* generative models, we are still puzzled with the challenges of GAN models mentioned above (e.g., the detailed analysis/evaluation of D/G, the mode collapse issue). Thus, it is worthwhile to study GAN models in-depth for both machine learning and visualization community.

In this work, we focus on a popular GAN model, DCGAN [83], to design and develop a visual analytics system, i.e., *GANViz* (Figure 4.1). The system helps domain experts understand the adversarial process of GANs in-depth, including evaluating the model performance for both D and G, providing evidence and interpretations of the performance, and conducting comparative analysis with the evidence. *GANViz* consists of five components. The first component of metric view provides an overview of model dynamics in the training process. The three components of probability view, distribution view, and *TensorPath* view offer rich evidence and interpretations of the model performance, such as the decision features of D, feature patterns of both real and fake data, neural network structures, and associated decision-critical data in the structures (e.g., neurons, activations, parameters, etc.). Finally, the comparative analysis component allows users to understand how the decision-critical data are evolved and how they work for the data from different sources (e.g.,

fake/real, true/false predictions). We demonstrate the effectiveness of the system through case studies of two real-world datasets with domain experts. To sum up, the contributions of this work include:

- We design and develop an interactive visual analytics system, *GANViz*, to provide an in-depth understanding of the adversarial process of GAN models.
- We introduce a compact matrix-based visual design, *TensorPath*, to reveal the full-spectrum information inside deep neural networks. The design is independent of specific models, and thus can be applied to many other deep neural network models.
- From case studies with domain experts, we summarize feedback and derive insights with important implications on understanding, evaluating, and improving GANs.

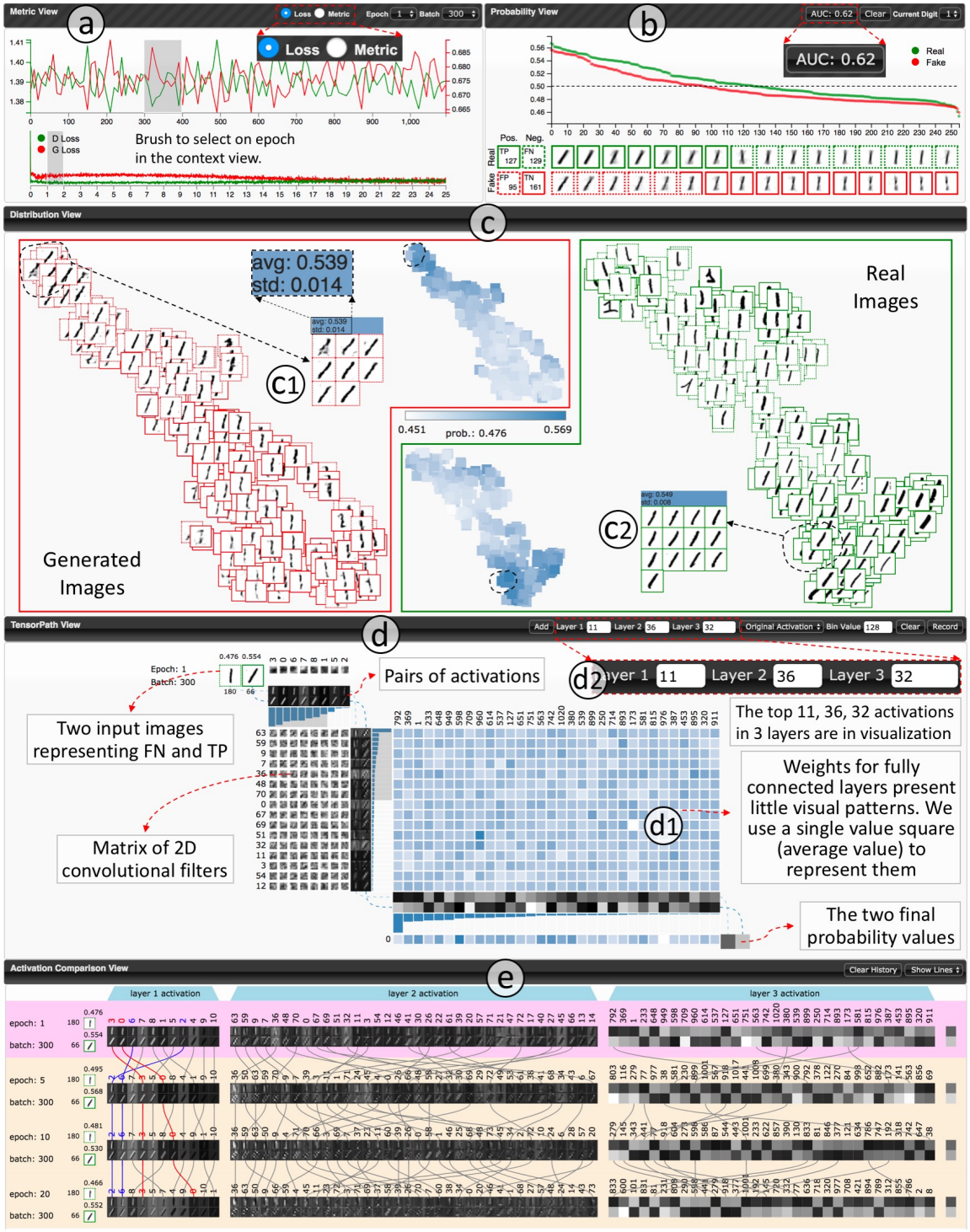

Figure 4.1: *GANViz*: (a) the metric view with the *focus+context* support; (b) the probability view shows the probability of real/fake images; (c) the distribution view reveals the feature distributions of real/fake images; (d) the *TensorPath* view highlights important activations, filters, and weights of *D*; (e) the activation comparison view tracks the changes of activations.

4.2 GAN Models and Model Challenges

Deep generative models are emerging as an active area for unsupervised learning. The power of these models lies in learning and extracting features from large scale unlabeled data, and later using them for other tasks, such as classification and clustering. One of the most popular frameworks is Generative Adversarial Nets (GAN) [38], which trains two networks of Generator (G) and Discriminator (D) as feature extractors.

GAN works in the following way. Suppose we have a dataset X (e.g., a collection of images, sounds, texts, etc.), and want to generate a new data item, x', that is similar to those in X. GAN achieves this goal by finding the distribution of X, i.e., $P_t(x)$. Generating x' is, then, similar to take a sample from $P_t(x)$. However, finding the exact $P_t(x)$ is challenging, as X can be infinitely large (e.g., images of all human faces) but a model can only take a finite set of inputs. GAN, therefore, resorts to find a distribution $P_g(x)$ that can best approximate $P_t(x)$. In detail, it takes a random sample z from a simple distribution $P_z(z)$ (e.g., a uniform or Gaussian distribution) and maps it to x' through a deep neural network (called Generator, G). To make x' similar to data items in X, GAN tunes the parameters of G with the help of another deep neural network (called Discriminator, D), which learns data features in X and discriminates x' from data items in X. During the training, they compete with each other iteratively by optimizing a minimax target function, i.e., Equation 4.1 [38]. To the end, G learns $P_g(x)$ and can generate data that are almost indistinguishable from real data; and D learns to extract reusable features from real data.

$$\min_G \max_D \mathbb{E}_{x\sim P_t(x)}[logD(x)] + \mathbb{E}_{z\sim P_z(z)}[log(1-D(G(z)))] \tag{4.1}$$

There are many extensions of the original GAN [38]. Their goal is to improve the quality of feature extractions by modifying the network structure of G and/or D. For

G, the extensions mainly improve G in the latent variable space, e.g., BiGAN [29] and InfoGAN [24]. For D, the extensions take advantages of the additional label information, e.g., semi-supervised GAN [77] and auxiliary classifier GAN [78]. Some other methods introduce additional components for both G and D. For example, conditional GAN [72, 78] uses class information to separate data in G and D. One popular such extension for image data is the DCGAN (Deep Convolutional GAN [83]), shown in Figure 4.2. In this model, G is a deep neural network with multiple transposed convolutional layers, which unrolls features layer by layer from random inputs (i.e., z vectors) and generates images. D is a deep convolutional network (a binary classifier similar to CNN [66, 113]), which discriminates the generated images from real images. This model is also our focus in this work. Further details and source code of the model can be found in [83] and [7].

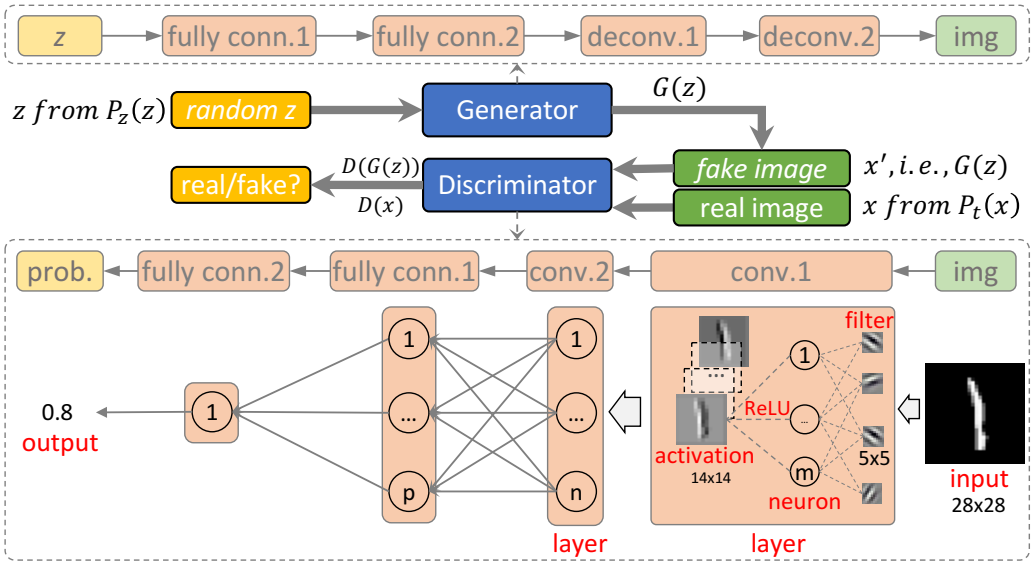

Figure 4.2: Simplified architecture of the DCGAN model used in this work. G and D are two deep neural networks with multiple transposed convolutional layers and convolutional layers, respectively.

There are several challenges to understand and train GANs [37]. One is the stability and convergence of the models. Current research is debating the theoretical guarantee on the convergence of the GANs game [37]. Some approaches, like MixGAN [14], can only guarantee an approximate equilibrium between D and G. Another challenge is the mode collapse issue [70], i.e., the generator can only learn a local distribution of the real data. Although WGAN [13] introduces the earth-mover distance to alleviate the problem, it is still unknown if the learned features really capture the distribution of real data or not. The third challenge is to understand and evaluate GAN models. The current evaluation methods are either through measuring the performance of additional classification tasks or qualitatively observing some generated results in a cherry-picking way. In this work, we adopt a visual analytics approach and attempt to shed some light on the above challenges of training and understanding GANs.

4.3 Requirement Analysis

We worked with three domain experts to elicit the design requirements to understand GANs. The design and development went through three stages: (1) initial understanding of GANs (how the model works, the current challenges); (2) prototyping with CNN visualization approaches [113] for DCGAN; (3) iterative design and development of *GANViz*.

Along with the iterative design process and our interaction with the experts, three main themes: ***model evaluation***, ***model interpretation*** and ***comparative analysis***, are emerging to aid their understanding and improving GAN models.

R1: Supporting *model evaluation* during the dynamic training process. It is a challenging task to evaluate GANs because of the nature of unsupervised learning and the dynamic training process. The existing evaluation method of using additional classification

tasks only gives a score without in-depth justification, and eyeballing the generated results is subjective and time-consuming. Both methods fail to reveal the model quality over the training process. Therefore, we distilled the following requirements under this theme:

- *R1.1: What is the overview of GANs' quality during the training?* For example, how do the loss functions of *D*/*G* evolve over time? Besides the loss functions, what other measures can we use to help us understand the model convergence and mode collapse problem?

- *R1.2: How is the performance of D evolved in different stages?* For *D*, it is important to measure and present its capability to distinguish fake and real data over the training.

- *R1.3: How is the performance of G changed in different training stages?* One challenge with *G* mentioned by domain experts is the mode collapse issue (i.e., *G* tends to learn local information and generate similar results again and again). Therefore, the tool should enable users to monitor the quality of *G*, to diagnose the mode collapse problem.

R2: Enabling *model interpretation* for detailed analysis on model quality. Besides the overall understanding of model quality, it is desirable to interpret how and why the quality is achieved. Furthermore, it is also helpful to interpret the features learned and extracted, especially for *D* who extracts reusable features that can be applied to other tasks. Towards the two ends, we embraced the following goals:

- *R2.1: Revealing the features and data distribution learned by both G and D.* For *G*, it is helpful to see the typical features and the distribution of the generated images in comparison with the real images. For *D*, the domain experts want to know what visual features are used to make the decision of real or fake images.

- *R2.2: Demonstrating the network architecture of D and interpreting the extracted features.* This aims to present hidden features (e.g., important neurons, learned features) of the *D*'s network to support detailed investigations.

R3: Providing *comparative analysis* to enrich the understanding of model dynamics. One need mentioned by the domain experts is to compare how image features are learned and used from two dimensions: (1) the image sources, e.g., positive/negative predictions, real/fake, and different classes, such as *apple* or *orange*; (2) the time dimension: how the learned features evolve during a training. This type of comparative analysis assists users to develop further understandings on the power of *D* and model evolution.

- *R3.1: Between-Source Comparison: comparing two images from different sources (e.g., true-positive (TP) vs. false-positive (FP) images; apple vs. orange), to examine how D could differentiate them.* For example, comparing the two images representing TP and FP, and revealing the differences between their corresponding activations can help users understand why *D* uses certain features to make the decision, and what neurons are important in extracting these features.

- *R3.2: Temporal Comparison: tracking the same pair of images across the training process.* Users should be able to compare the same two images in different stages, to track how the importance of neurons (in differentiating them) changes over time. This would lead users towards meaningful inferences on the formation of *D*'s decision over time.

4.4 *GANViz* Framework Overview

With the aforementioned design requirements, we designed and developed an interactive visual analytics system, i.e., *GANViz*, for domain experts to analyze GAN models. This section provides an overview of the system (Figure 4.3).

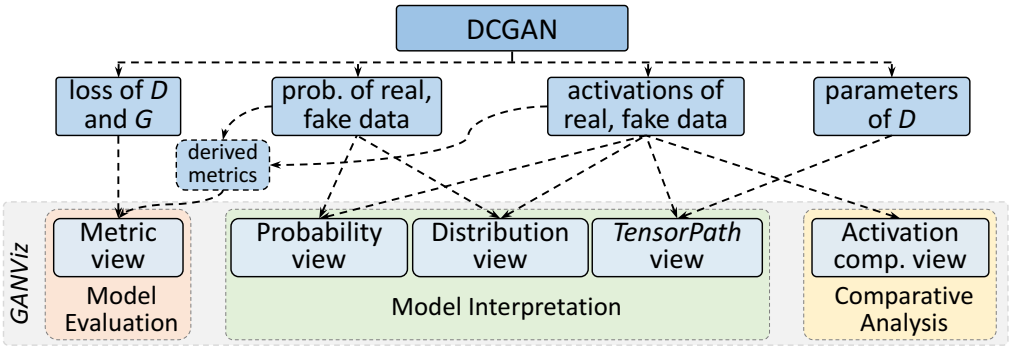

Figure 4.3: Framework of our visual analytics system.

Model Execution and Data Collection. The first step is to collect necessary data from the model training for later visualization and analysis. During the training process, we collect four types of data with different frequencies. The **first** type of data is the values of loss functions for both D and G. These values are common indicators for the quality of the model, and thus, should be collected frequently to closely monitor the training progress. The **second** type of data is the probability values (likelihood to be real) for each real/fake image. The value is the prediction result of the D network with an image as input. We sampled a subset of real/fake images as the inputs. For real data, 256 images are randomly taken from the training images; for fake data, 256 random z vectors are fed to G to generate 256 fake images. As a result, we collected 512 probability values (256 for real images and 256 for fake images), in total, at each timestamp.

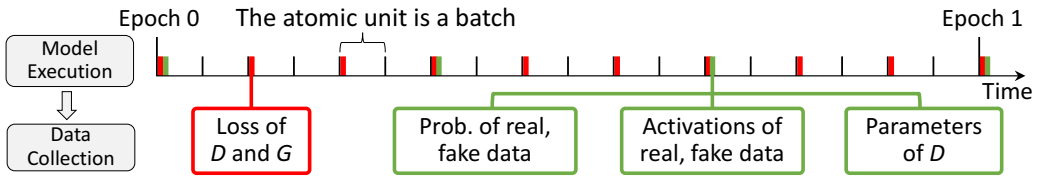

Figure 4.4: Collecting data with different frequencies. In this example, losses are collected every two batches (red ticks); probability values, activations, and parameter data are collected every six batches (green ticks).

The **third** type of data is the activations (from all layers) for each real/fake image when it goes through the D network. For example, the first layer of activations for an image is the image itself; the second layer of activations is the image after it went through the first convolutional layer; the last layer of activations is the probability value of the image. The **fourth** type of data is the parameters of D, i.e., the filters of convolutional layers and weights of fully-connected layers. These parameters, especially the convolutional filters, will help users understand how activations are generated and how they lead to the final probability values. Different types of data were collected with different frequencies to reduce the size of data to be collected. As shown in Figure 4.4, the loss values are collected more frequently than others.

Besides the directly collected data, we also provide three derived metrics to better evaluate the model quality:

AUC (Area Under Curve) reflects the performance of D, measured by the area under its Receiver Operating Characteristic (ROC) [32] curve. A ROC curve plots the true positive rate ($TP/(TP+FN)$) against the false positive rate ($FP/(FP+TN)$) of a binary classifier under different decision thresholds (Figure 4.5). More points on the top-left corner (larger

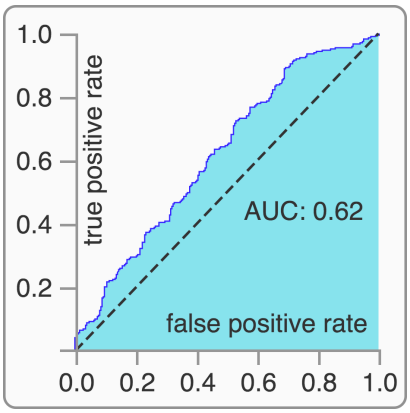

Figure 4.5: AUC in cyan.

area under the ROC curve) of this plot (i.e., high true-positive rate, low false-positive rate) indicate a better performance of D.

Diversity quantifies the diversity in the generated images, which is used to detect the mode collapse problem in G, i.e., the value of this metric should keep high if no mode collapse happens. The metric first derives a similarity value among all generated images, using structure similarity (SSIM) [108]. The inverse of this value is then used to indicate the diversity for the set of images. If the number of images is n, then,

$$Diversity = 1/\left(\frac{1}{n(n-1)/2}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(SSIM(img_i, img_j)\right)\right). \tag{4.2}$$

Dist-Diff (Distribution Difference) measures the difference between the fake data distribution, $P_g(x)$, and real data distribution, $P_t(x)$. The metric is derived through two steps: (1) approximate $P_g(x)$ and $P_t(x)$ from a subset of fake and real images respectively; (2) measure the difference between the two distributions. For the first step, we use PCA to reduce images to lower dimensional space, and employ Gaussian Mixture Models (GMM) [22] to fit them as a distribution. The second step measures the KL divergence [61] between the two GMMs

through Monte-Carlo samples (Equation 4.3).

$$Dist\text{–}Diff = KL\big(P_t(x)||P_g(x)\big),\ \ P_t(x) = GMM(PCA(X)),\ \ P_g(x) = GMM(PCA(G(\mathbf{z}))) \tag{4.3}$$

Visual Analytics Modules and Components. Aligned with the design requirements, the components of *GANViz* are organized into three modules, including quality monitoring, model interpretation, and comparative analysis. The three modules are designed upon the top-down exploration flow elicited from domain experts and also organized by Shneiderman's information seeking mantra, i.e., "overview first, zoom and filter, then details-on-demand" [94].

In the three modules, five views (Figure 4.3) are designed to analyze different data in a top-down style. In the quality monitoring module, the metric view helps users understand the overall quality (R1) by visualizing various model metrics. Then, the probability view, distribution view, and *TensorPath* view enable users to examine and interpret the model at a specific training stage of their interest (R2, also R1.2). Over the journey of exploration, the activation comparison view supports users to collect relevant model information and conduct comparative analysis (R3). The five views are coordinated with each other, and rich interactions (e.g., brushing, filtering, selection, etc.) are provided to link the views for flexible exploration.

4.5 *GANViz* Visual Analytics System

In this section, we explain the design rationales and visual interface for the five components of *GANViz* (Figure 4.1). The components are: the metric view (Figure 4.1a) for model quality dynamics (R1); three components of probability view, distribution view, and *TensorPath* view (Figure 4.1b, 4.1c, 4.1d) for model interpretations (R2); and the activation comparison view (Figure 4.1e) for detailed comparative analysis (R3). The layout of these

five views follows the analysis workflow suggested by domain scientists (i.e., from high-level overviews to detailed explorations/comparisons), and the views are linked together with coordinated visualizations.

4.5.1 Metric View

The metric view (Figure 4.1a, 4.6) provides an overview on the quality, and quality evolution, of the model. Four types of metrics are analyzed in this view: loss values of D/G, AUC, Diversity, and Dist-Diff scores. These metrics varying over the training process are essentially multiple time-series data [60, 80], and organized into two sub-groups: model convergence with loss values (R1.1); model performance for D (AUC, R1.2) and G (Diversity, Dist-Diff, R1.3).

This view is presented by a line chart, which is an intuitive and straightforward representation to show multiple time-series [48, 50]. In the chart, the horizontal axis represents time (epochs/batches) and vertical axis represents values of different metrics. The two sub-groups of metrics are presented in their own views (Figure 4.1a, 4.6). Users can easily switch between them by checking the radio box on the top (the inset in Figure 4.1a). Notice that the second sub-group of metrics do not share the same value range, we normalized them to be in [0, 1] before visualization.

A $focus{+}context$ zooming is also provided to explore the model dynamics over a lot of epochs/batches. The context view (the bottom half of Figure 4.1a) shows the whole training in the unit of epoch over the horizontal axis. The focus view (the top half) presents a detailed view of the metrics from the selected epoch and the axis is at the scale of batches. Both views are equipped with a snapped brushing (i.e., the brushed regions will be rounded to one

epoch/batch automatically) for easy selection. Once the interested epoch/batch is selected (epoch 1, batch 300 in Figure 4.1a), other views will be updated accordingly.

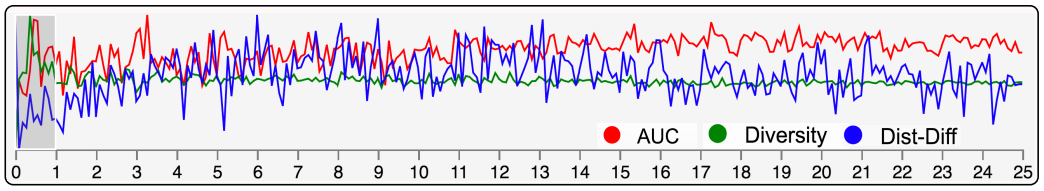

Figure 4.6: Metric view: measuring model quality with three metrics.

4.5.2 Probability View

The probability view (Figure 4.1b) provides detailed information for the performance of D (R2.1) at a timestamp of selection (e.g., x epoch, y batch). This view presents the predicted probabilities of the sample data (256 real/fake images) along with their distributions in the confusion matrix. It contains two components: (1) a line chart showing the probability; (2) image thumbnails with confusion matrix information.

Probability Chart. To clearly demonstrate the separation of probabilities from real/fake samples, we sort those probabilities and present them in a line chart, as shown in Figure 4.1b. Two curves show the probability of the 512 (256+256) images sampled from the real (in green) and fake (in red) data. The horizontal axis represents the order of the 256 images sorted by their probability value (in a decreasing order) and the vertical axis shows the probability values. The horizontal black dashed line indicates the current decision threshold of D: images with probability value less than this threshold are considered as fake, otherwise they are considered as real. Users can drag this line to adjust the current decision threshold. As the decision threshold moves, we can obtain a metric of AUC score [32], which indicates

the prediction power of D. Users can also check the AUC region from this view by clicking the AUC button on the top. Figure 4.5 shows the pop-up AUC window for the current timestamp.

Aggregated Image Thumbnails. To reveal a visual trend of the images, we place the aggregated image thumbnails below the horizontal axis. We aggregate every 16 consecutive images and present them as one thumbnail by averaging the pixel values to save the screen real estate. As a result, only 16 (256/16) thumbnails are shown. The two rows of thumbnails in Figure 4.1b represent both real (top) and fake (bottom) image samples. Clicking on one thumbnail will pop up a small window to show the 16 original images. The thumbnails from left to right reveal how visual features change with the decrease of their probability value. In Figure 4.1b (from left to right), we can see how the italic style of digit 1 affects the probability values derived from D.

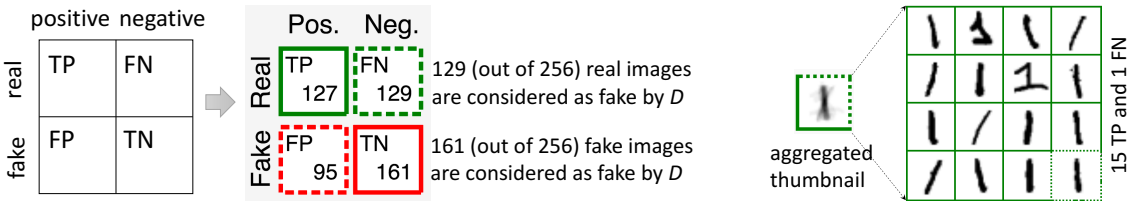

Figure 4.7: Left: confusion matrix information with glyphs; right: a thumbnail aggregates 15 TP and 1 FN image (the bottom-right one).

Confusion Matrix Visualization. The confusion matrix (Figure 4.7 left) shows the classification results with current decision threshold indicated by the horizontal black dashed line in Figure 4.1b. The border style of images indicates whether D has classified them correctly (solid border) or not (dashed border). The bottom left corner of Figure 4.1b shows the legends for the four categories in the confusion matrix (an enlarged version is

shown in Figure 4.7 left). The number in each legend indicates the number of images falling into that category. A thumbnail with both solid and dashed borders indicates that the thumbnail averages images from two categories. For example, Figure 4.7 (right) shows the 8th thumbnail in the top row of Figure 4.1b. The top and right borders of this thumbnail are in dashed style; whereas the bottom and left borders are in solid style. Clicking on this thumbnail allows users to check the original 16 images. From borders of the original images, we know that this thumbnail averages 15 TP images and 1 FN image.

4.5.3 Distribution View

The distribution view (Figure 4.1c) provides supporting information to understand the performance of G (R2.1, Diversity and Dist-Diff). It shows an overview on the appearance of all real/fake images, their feature and probability distributions.

Image Overview and Feature Distribution. To provide an overview of the visual features of the sampled (real/fake) images, we visualize those images and cluster them with the t-SNE [69] algorithm to reveal their feature distribution. The 256 fake/real images are shown on the left/right of Figure 4.1c. Similar to the probability view, the category of each image is encoded with its border style and color. Additionally, users can perform lasso selections in this view. Selected image will be sorted (in probability decreasing order) and displayed in a pop-up window. The header of the window will show the average and standard deviation of probabilities of the selected images (Figure 4.1-c1). From this view, features of images in different clusters can also be easily observed. For example, the selected images in Figure 4.1-c1 are generated images with italic styles.

Probability Distribution. We show the probability distribution, along with the feature distribution, of the sampled images to reveal what features have tricked D. Other design

choices, such as encoding the probability values with glyphs placed aside the images, were also considered but introduced more visual cluttering. In the middle of Figure 4.1c, the probability distribution of fake/real images is presented with a group of colored squares on top/bottom. One square corresponds to one image; the color of the square (from white to blue) represents the corresponding image's probability value (from small to large). The layouts of the squares and images are synchronized, which allows users to find the correspondence between them. Interactions are introduced to link the two parts, as well as the corresponding data in other views. For example, hovering over one image will highlight the image and its probability square. The corresponding sample in other views will also be highlighted.

4.5.4 *TensorPath* View

Two design objectives for the *TensorPath* view are: (1) revealing the details of *D*'s network architecture (layers, activations, parameters, R2.2); (2) showing and comparing the important features that *D* extracts to make predictions (R3).

The network architecture of *D* can be considered as a chain of layers (Figure 4.8 left). There are two categories of data of our interest: (1) the inputs (in green), activations (in orange), and outputs (in yellow); (2) the filters and weights (in blue) learned during the training process. The activations and outputs are generated, as input images (real/fake) pass through the network (the activations in each convolutional layer are the extracted features of that layer). The four numbers in each non-blue rectangle represent the number of images/activations, the number of channels, width and height of each image/activation. The first number is always 2 since we always compare a pair of images.

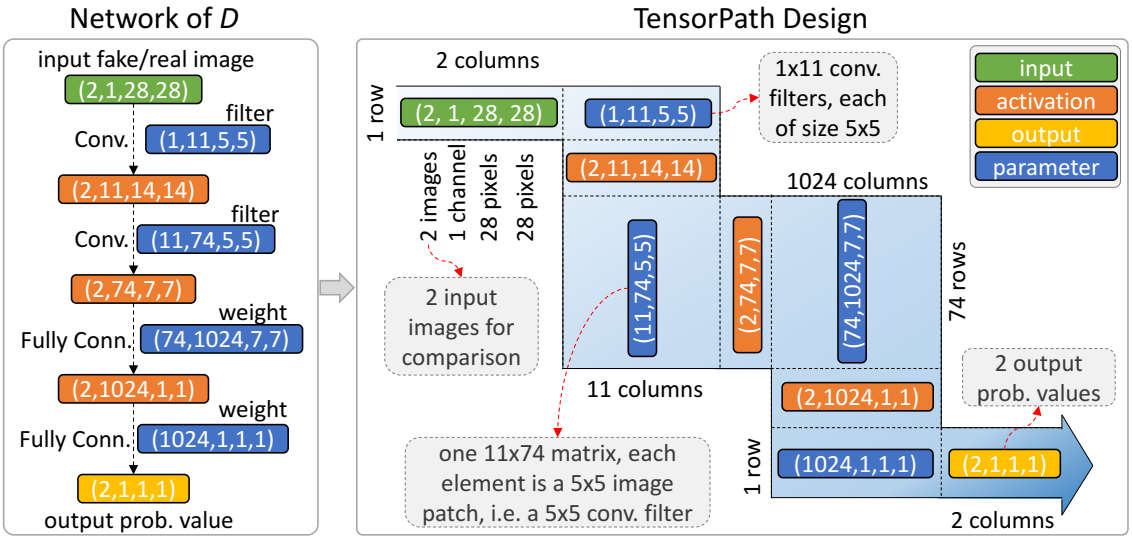

Figure 4.8: Left: the neural network architecture of *D* in the DCGAN model [7]; right: a zigzag layout design of the *TensorPath* view.

The design of the *TensorPath* view is illustrated in Figure 4.8 (right). The design goal is to demonstrate the network structure of D, and the flow of various types of data in a compact way. To efficiently use the space, we design a zigzag path visualization, which we called as *TensorPath*, to illustrate the network structure and dataflow in *D*. The inputs, activations, and outputs in neighboring layers are placed to be orthogonal to each other, and connected by filters or weights in the zigzag path. This design is to compactly demonstrate layer structures, easily identify important activations/parameters, and intuitively track data across layers of *D*. Figure 4.1d shows our implementation of the *TensorPath* view. The enlarged top-left corner is shown in Figure 4.9. Pairs of activations (Figure 4.9a) in each layer are sorted by the activation difference of the pairs (mean square error between two activations) in a decreasing order. The blue bars next to the activations encode the difference between them (Figure 4.9b). They are also used to select and filter out the activations shown in the view.

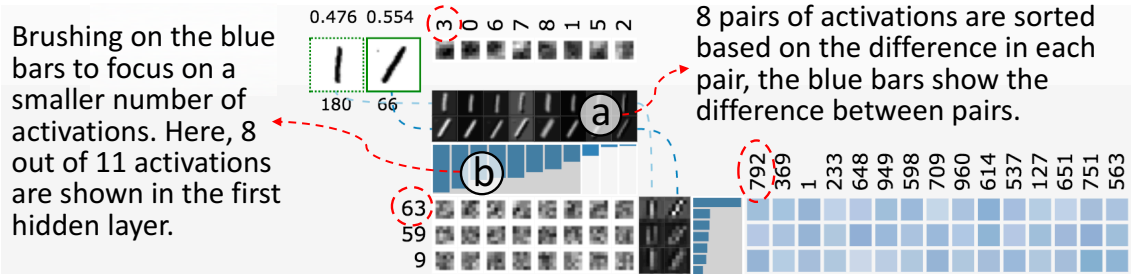

Figure 4.9: *TensorPath* view: the enlarged top-left corner of Figure 4.1d.

The data of interest (input/activation/output/filter/weight) are all presented as 2D images in *TensorPath*. Hovering over an image will show the enlarged version of it. However, those images may have different sizes, e.g., the resolutions of input images and convolutional filters are 28×28 and 5×5 respectively. To put them consistently in *TensorPath*, we introduce three operations on a 2D image:

- *Scaling* is to scale up/down one image. Scaling up an image is to copy each pixel to its neighboring pixels. For example, to scale an image two times up, each pixel will be duplicated twice horizontally and vertically. Scaling down an image is to reproduce the image but with a fixed stride.

- *Padding* is to append rows and columns of white pixels around an image. In certain cases, the desired image size cannot be achieved by scaling only. For example, if the original image size is 5×5 and the desired image size is 12×12. In this case, the image can first be scaled up twice, then padded with two rows and two columns of pixels.

- *Aggregation* is to average pixels of an image and use the average value to represent the image. This is a similar idea to heat-map visualization and very efficient to visualize

the large amount of parameters (weights) in fully-connected layers (Figure 4.1-d1, the colors from white to blue represent values from small to large).

We provide two interactions to address the scalability issue in *TensorPath*. First, users can use a threshold value to filter out less important activations from the visualization. With our interface (Figure 4.1-d2), users can flexibly change these values. By default, we use 11 (out of 11), 36 (out of 74) and 32 (out of 1024) for the three hidden layers of *D* in the DCGAN model used in this work. Second, users can brush on the difference bars to focus on a smaller but more important set of activations, as we have shown in Figure 4.9b.

4.5.5 Activation Comparison View

The activation comparison view allows users to compare multiple pairs of activations across sources or timestamps (R3.1 and R3.2). This is a typical visual analysis task for high-dimensional data. Among all design alternatives, we use a design that is similar to the Parallel Coordinates Plot (PCP), which can effectively visualize and compare high-dimensional data [107]. As shown in Figure 4.1e, three horizontal PCPs are used for three layers of activations. Each parallel axis (each row) is a sequence of sorted activation pairs from the *TensorPath*. Across rows of individual PCPs, the pairs of activations with the same indices are linked together with Bézier curves. The number of curves between neighboring rows indicates the number of shared activations in distinguishing different pairs of input images.

This view and the *TensorPath* view work together to support both cross-source and cross-time comparative analysis. During users' explorations in the *TensorPath* view, they can compare a pair of images and record the pair into the activation comparison view (as one row). The new pair will then be compared with other existing pairs (rows) in the

view. Clicking on one row in the activation comparison view will trigger the update in the *TensorPath* to show network details for the row of images. The pink row in Figure 4.1e is in selection now (i.e., the corresponding images of that row are currently presented in the *TensorPath* view).

4.6 Case Studies and Domain Experts' Feedback

In this section, we show the effectiveness of *GANViz* by presenting case studies with two real-world datasets (with different complexities), along with the insights derived during the studies.

4.6.1 Datasets: MNIST and QuickDraw

The MNIST [5] and Google QuickDraw [4] datasets were used in our case studies. Both datasets have 70,000 images (with resolution 28×28) in 10 classes (for QuickDraw, we randomly selected 7,000 images from each of the 10 selected classes). Figure 4.10 shows one sample from each class of these two datasets. As we can see, the two datasets come with different complexities of visual features, i.e., the drawings in QuickDraw are more complex than the digits in MNIST.

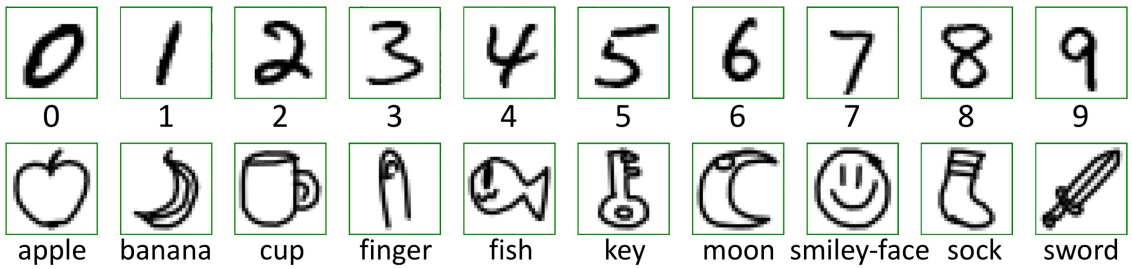

Figure 4.10: The two datasets: top: MNIST [5]; bottom: QuickDraw [4].

We trained the DCGAN model with the two datasets separately for 25 epochs. The batch sizes for both trainings were 64, so there were 1094 (70,000/64) batches in each epoch. Loss function values were collected every 10 batches; other data (see Section 4.4) were collected every 100 batches.

4.6.2 Findings from Case Studies with Domain Experts

This section describes our findings from the case studies following the three themes of design requirements to show how *GANViz* satisfied them.

4.6.2.1 Model Evaluation

With *GANViz*, we can easily observe the patterns of model training dynamics. From the metric view, we observed a quick drop of losses after several training epochs for both datasets (Figure 4.1a, 4.11a). However, we also found there were two different patterns for them: (1) the losses dropped much faster for MNIST (after the first epoch) than QuickDraw (after 3 epochs); (2) the losses of both *D* and *G* oscillated in a relatively smaller range in MNIST than those in QuickDraw.

We also observed some shared trends from the three quality metrics of MNIST and QuickDraw (Figure 4.6, 4.11b): (1) the capability of *D* is improving as the AUC curves (in red) show an increasing trend for both cases (R1.2); (2) the Diversity curves (in green) remain high throughout the 25 epochs, which indicates that mode collapse did not happen in *G* (R1.3); (3) with the training process moving forward, the Dist-Diff curves (in blue) demonstrate a decreasing trend (after 14 epochs), which means *G* is progressing towards the right direction to capture the real data distributions (R1.3).

The metric view also reveals a couple of model quality differences between the two datasets. First, *D* is struggling more in discriminating real/fake digits (from MNIST) than

real/fake drawings (from QuickDraw), as the AUC curve oscillates a little more significantly in the MNIST case. This indicates that G can generate better fake digits (compared to fake drawings) to trick D. It also verifies our expectation that it is easier for G to learn simple visual features than sophisticated ones. Second, the oscillation of the Dist-Diff curve becomes larger in later epochs of the QuickDraw data. In contrast, the oscillation degree of the Dist-Diff curve in MNIST does not show obvious changes. The lager oscillation degree in QuickDraw indicates that, for G, capturing the real data distribution of QuickDraw is more difficult.

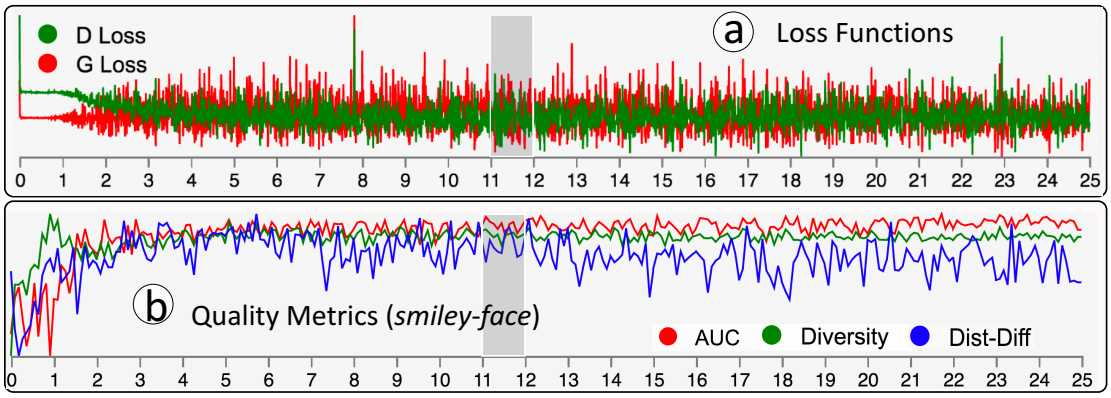

Figure 4.11: Loss and metric values of QuickDraw in the metric view.

4.6.2.2 Model Interpretation

Decision features of D (R2.1). We used the probability view to explore what decision features D had used, and how D used them over the training process. For MNIST, D learned to use various font styles (italic/bold) to make decisions. For example, from the two rows of thumbnails in Figure 4.1b, we observed the trend that images with higher probabilities are digit 1s with italic styles; and in Figure 4.12a, the decision feature of D is the boldness of

digits. For QuickDraw (the more intricate data), *D* used more features to make decisions, including: the size of *apples*, the orientation of *keys*, the contour shape of *smiley-faces*, as shown in Figure 4.12d, 4.12e, 4.12f.

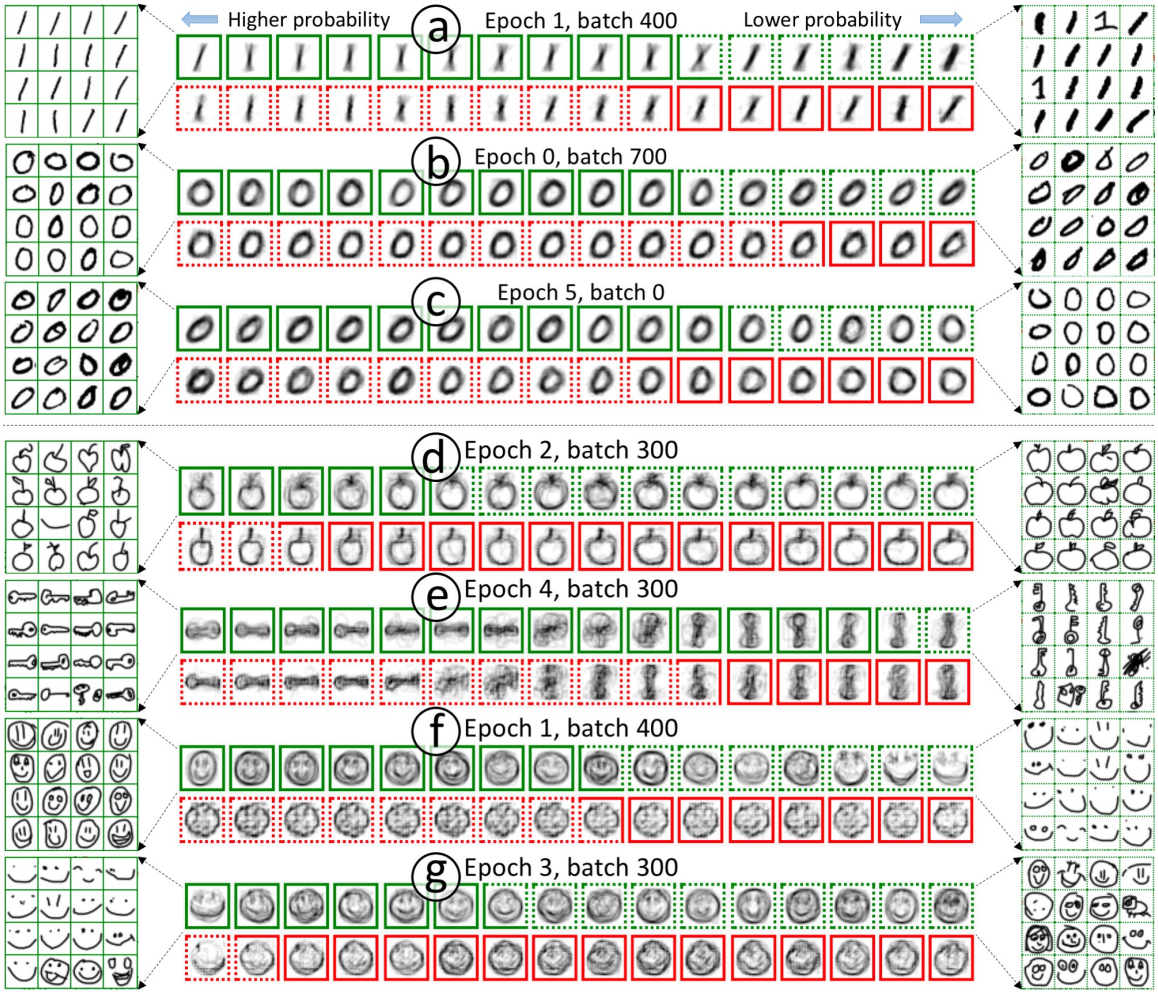

Figure 4.12: (a) Digit 1 with bold style as the decision feature; (b, c) digit 0 with the flipping of decision feature (i.e., the italic style) in two timestamps; (d, e, f) the decision features of *D* in different classes of drawings in QuickDraw; (f, g) the flipping of decision feature in *smiley-faces*.

More interestingly, we found that *D* may flip its decision features in different training stages, in both datasets. For example, Figure 4.12b, 4.12c show that *D* used the italic style of digit 0 as its decision feature, and it flipped its decision between the two timestamps. For QuickDraw, we saw that *smiley-faces* with circular boundary were considered to have higher probabilities to be real in epoch 1 batch 400 (Figure 4.12f); however, in epoch 3 batch 300 (Figure 4.12g), images with those features received lower probabilities. More examples on this decision flipping from the QuickDraw data are shown in Figure 4.13.

Based on our explorations, we notice that the decision feature of *D* is not predictable, and *D* may flip its decisions back and forth several times during a training. Even with the same model, same dataset, *D* may have different decision features or decisions, in the same timestamp of two separate trainings. This interesting observation calls for more theoretical explorations from the domain experts.

Visual explanation for the performance of G (R2.1). We used the distribution view (Figure 4.1c) to verify and explain the overall performance of *G* discussed in the metric view (i.e., the Diversity and Dist-Diff curves). This view provides an overview on the quality of images generated by *G*, as well as the visual feature distributions of fake and real images.

We first observed that *G* always attempted to learn the decision-critical features over the training, even though *G* could not capture the full distribution of real data in the very early stages. In Figure 4.1-c1, although most of the generated images were still blurry (*G* was not very sophisticated) at this early stage (epoch 1, batch 300), some digits with italic styles had successfully cheated *D*, and received high probability values to be real (the average probability value of the selected images is 0.539). Similar phenomena were also found in the QuickDraw data. For example, in Figure 4.14, *G* learned to generate horizontal *keys* to

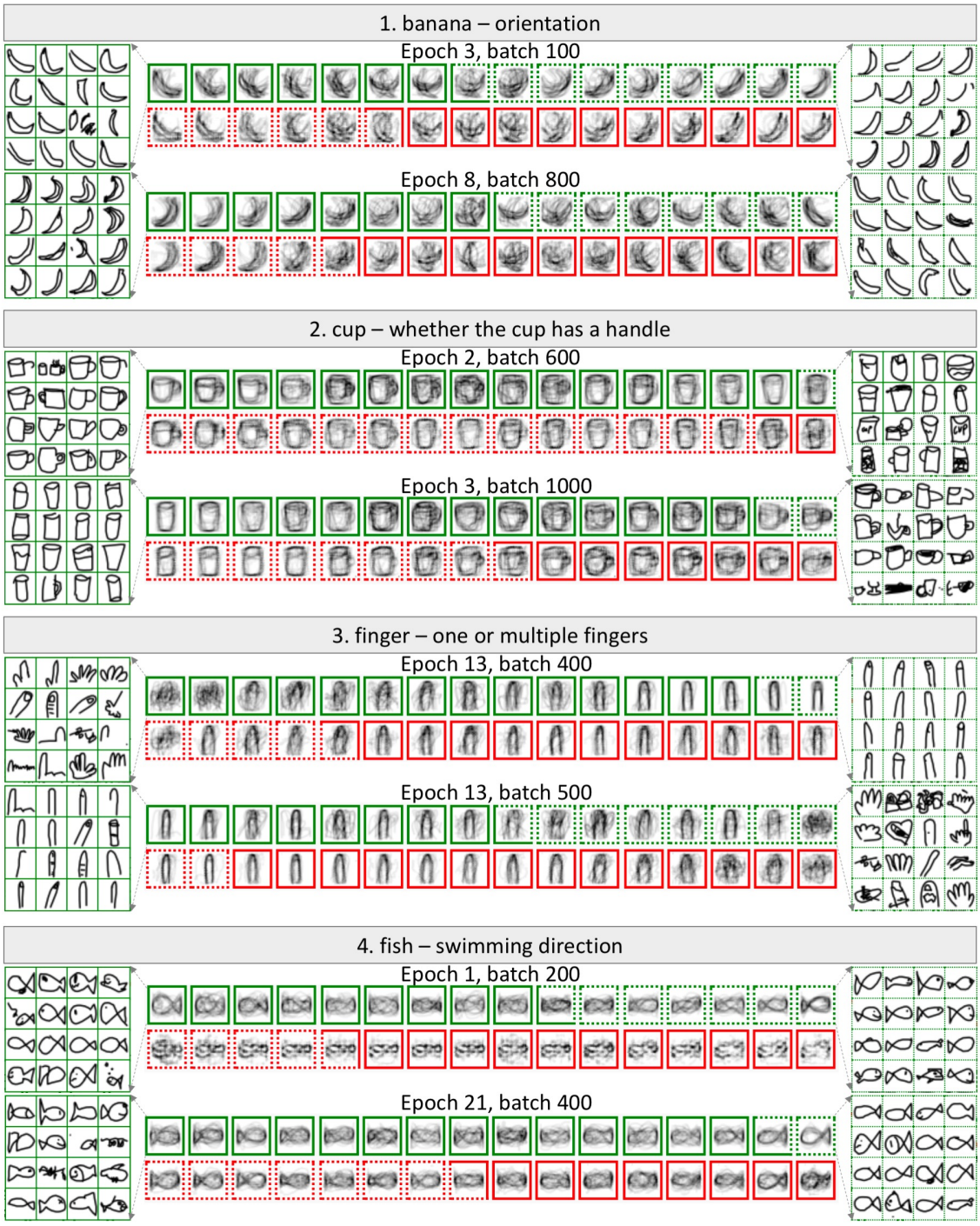

Figure 4.13: The flipping of D's decision was found in all classes of the QuickDraw data. This figure shows four more examples on the classes of *banana*, *cup*, *finger*, and *fish*.

trick *D*, and the image clusters with horizontal orientation in the top-right corner received higher probability values.

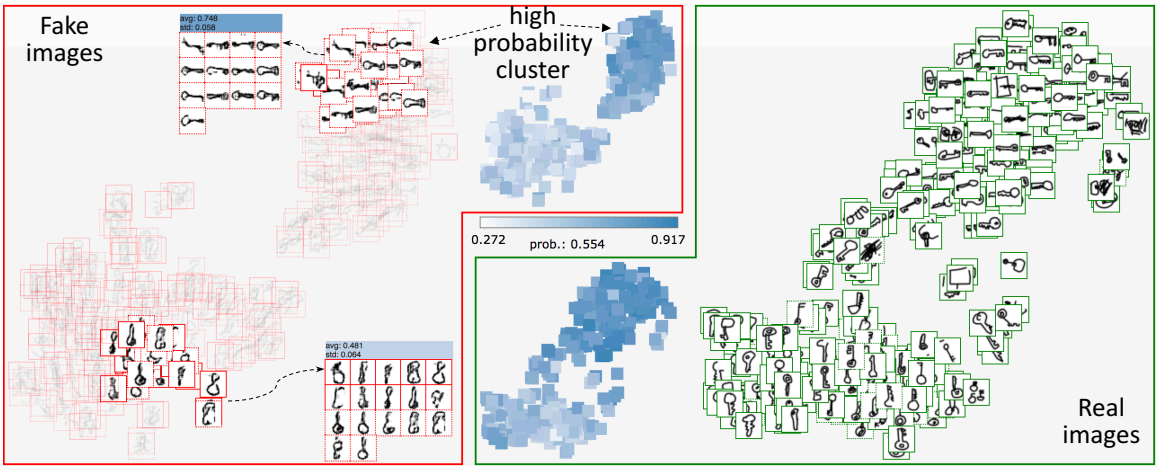

Figure 4.14: Distribution view for the drawings *key* from QuickDraw.

Moving to later stages, we observed that *G* could generate diverse images with good qualities, which explained the high Diversity and low Dist-Diff values shown in the metric view, and proved that the mode collapse issue did not happen. For example, in epoch 24, batch 100 of the MNIST training (Figure 4.15), *G* learned that the real data have many different features (e.g., italic/vertical, bold/slim) and could generate images that are very similar to the real data. As a result, *D* got really confused and could not tell what features lead to real data. The even distribution of white/blue squares in the middle of Figure 4.15 shows this.

Understanding the process of extracting decision-critical features by D (R2.2). The power of generative models comes from their capability of extracting reusable features. From previous explorations, we knew that both *D* and *G* learned the decision-critical features

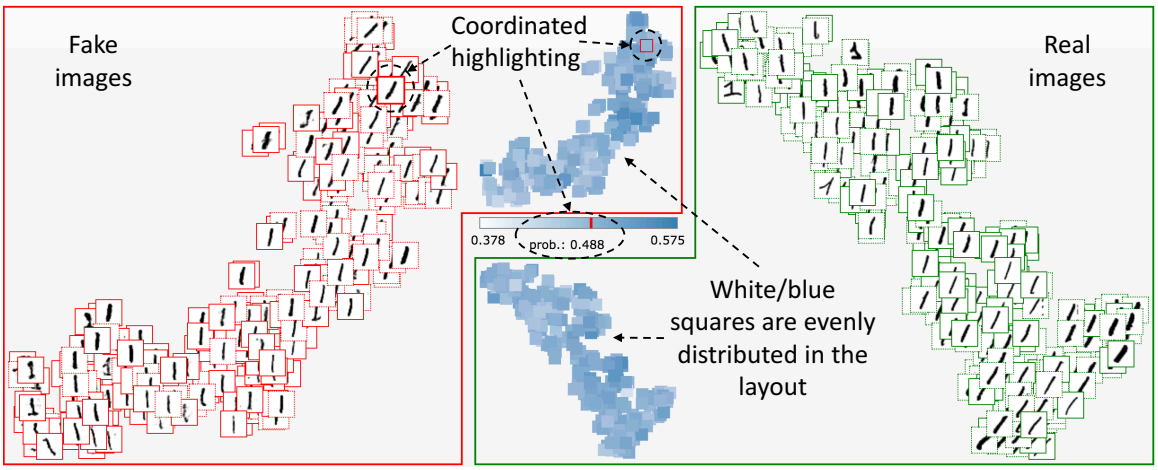

Figure 4.15: Distribution view in epoch 24, batch 100. The probability distributions (color squares) are more even, compared with Figure 4.1c.

(e.g. italic, bold styles in MNIST digits; size, orientation, contour shape in QuickDraw drawings) over the training process. Next, we used *GANViz* to understand and interpret how those features were extracted through the network. Specifically, we focused on *D*, which usually serves as feature extractors for other tasks.

Towards this goal, we used the *TensorPath* view to open the hood of *D* and observe how features were extracted through the layers of *D*'s network. From Figure 4.1b, we already knew that the decision feature of *D* in this timestamp is the italic styles of digit 1. We then selected two images of digit 1 from false-negative (i.e., without italic style) and true-positive (with italic style) to conduct detailed investigations in the *TensorPath*. Figure 4.1d shows how this pair of images flow through *D*'s architecture with important details, e.g., filters, activations. For example, it is obvious that the most important filters in discriminating the pair of images are: 3, 63, and 792 in the three hidden layers respectively (Figure 4.9).

Semantic interpretation of the extracted decision-critical features by D (R2.2). After getting an overview on how the pair of images flowed through the network, we further

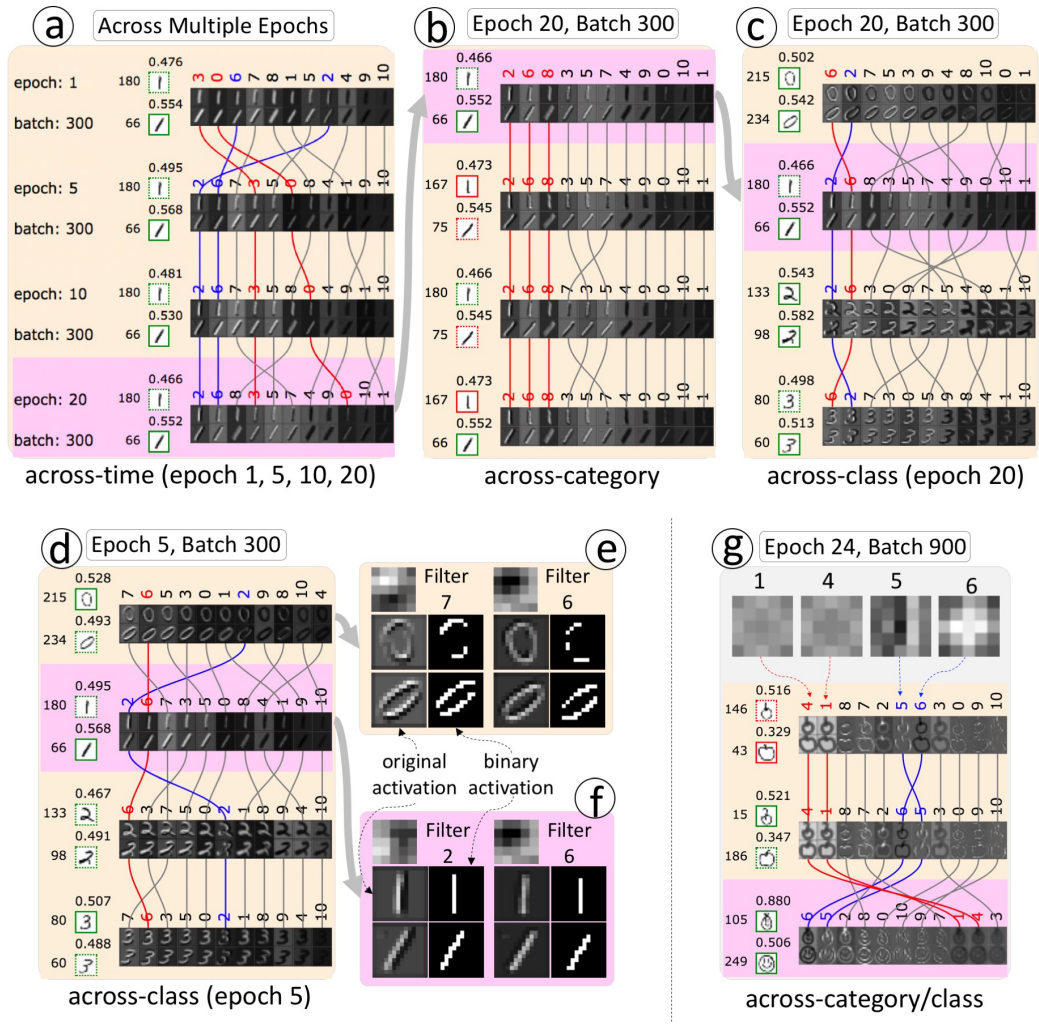

Figure 4.16: (a) Tracking a pair of digit 1s across four timestamps; (b) comparing vertical/italic digit 1s across categories; (c, d) comparing vertical/italic digits across classes in epoch 20, epoch 5; (e, f) enlarged important filters/activations of digit 0s, 1s from *TensorPath*; (g) comparing across categories (top two rows) and across classes (bottom two rows) in QuickDraw, along with three circular filters: 1, 4, 6; and one directional filter: 5.

investigated the details of filters/activations to see how the decision-critical features were extracted through the filters. For example, Figure 4.16f shows the enlarged view of the two most important filters in differentiating the two previously selected digit 1s, in a later training stage (epoch 5, batch 300, we chose to show a later stage as the filters were formed

more completely). Their corresponding activations from the *TensorPath* view are also shown. From the figure, we can see that filter 2 and 6 are two directional filters. The effect of a directional filter is similar to cast a light from different directions to the content of the original image, and the filter decides the source and direction of the light. The left side of filter 2 is white, and the color changes to black towards the right side. This indicates the light starts from left and is cast towards the right. Under this filter, the shape of digit 1s (both vertical and italic) can be extracted. The binary activations illustrate the difference of the extracted features more clearly. These activations are binary images derived from the original activations by thresholding (i.e., pixels with values greater than 128 are set to 255, otherwise they are set to 0). Filter 6 can be interpreted in the same way, as it is also a directional filter (from bottom-right to top-left). In Figure 4.16e, the most important filters of vertical/italic digit 0s in the same timestamp present another example. Under the effect of filter 7 (a directional filter from top-left to bottom-right), two sides of the italic digit 0 are activated. However, only a small portion of the vertical digit 0 is activated under this filter. As a result, filter 7 plays a very important role in differentiating the two digit 0s.

When it comes to the QuickDraw data, we observed that filters with more functions were involved in differentiating two drawings. For example, in Figure 4.17a, a pair of *smiley-face* images with different visual features are selected into *TensorPath*. In the enlarged version of this view (Figure 4.17c), one can figure out that, in addition to the two directional filters, i.e., 8 and 5, which capture the horizontal and vertical features, filter 2 (that captures the contour of one image) is also important to differentiate the two images. The differences in the three pairs of activations in Figure 4.17c help D generate different probability values for these two drawings.

4.6.2.3 Comparative Analysis

Understanding the evolution of decision-critical features (R3.2). *GANViz* also enables users to understand how the decision-critical features evolve over time. We observed that for both MNIST and QuickDraw, the important filters to differentiate two images became stable after several epochs, even *D* may still flip its decision features. For example, the activation comparison view in Figure 4.1e records four rows of activations (of the selected digits 1s) in four different timestamps. Figure 4.16a is an enlarged version of the first hidden layer. From this figure, we observed that filter 3 and 0 were the most important in epoch 1, but they became less and less important over time. In contrast, filter 2 and 6 became more important since epoch 5, and they kept being the most important two filters in later training stages.

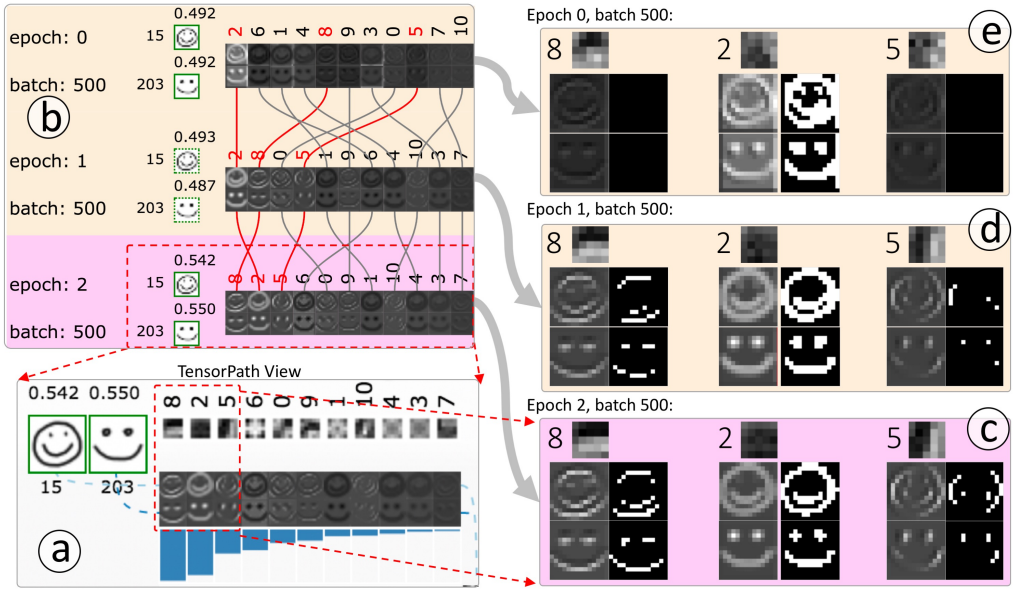

Figure 4.17: Across-time comparison: (a) the *TensorPath* view shows filter 8, 2, 5 are the most important; (b) trace the three filters in epoch 0, 1, 2; (c, d, e) enlarged filters and activations in the three epochs.

Also, *GANViz* can reveal how filters are formed and what features are extracted over time. Figure 4.17b shows the activation comparison view of two *smiley-faces* in three different epochs (epoch 0, 1 and 2, we present three early epochs here, as filters change more significantly at the beginning). From the activation comparison view, we realized that filter 8 was becoming more and more important in discriminating the two images. In the enlarged view of the three epochs, we observed that: (1) the horizontal filter 8 was not formed completely in epoch 0, and few features were extracted by it (Figure 4.17e, left); (2) the filter was roughly formed in epoch 1 (Figure4.17d, left) and could extract some disjointed pixels in this epoch; (3) in epoch 2 (Figure 4.17c, left), the filter could extract continuous regions from images and we can roughly recognize the *smiley-face* from the activated pixels. The same learning pattern can also be found in filter 5. Filter 2 extracts the contour of a *smiley-face* image. Over the three epochs, the extracted contour became more and more precise.

Comparing decision-critical features over different decision categories (R3.1). For images of the same class, we found that the decision-critical features were fairly consistent for images categorized into different categories (e.g., FN, TP). For MNIST, as shown in Figure 4.16b, although the four pairs of images (one vertical, one italic) represent different pairs of categories (category information is encoded in the images' border style and color), we can see that the four orders of filters in differentiating the four pairs are mostly the same. For example, filter 2, 6 and 8 (filter 0, 10 and 1) are always the most (least) important. The similar orders indicate that D treats images from different sources equally. Before seeing the visualization results, the experts were uncertain about this phenomenon. They responded that the comparative analysis enhanced their understanding about D.

The decision-critical features were also observed to be consistent in QuickDraw. From the top two rows of Figure 4.16g, we noticed that *D* used similar decision features when differentiating fake (the top row) and real (the second row) *apples*, as the orders of activation pairs were mostly the same. Big *apples*, either from fake or real data, were considered as fake by *D*; small *apples* were considered as real. Circular filter 1 and 4 played the most important roles.

Comparing the decision-critical features over different image classes (R3.1). At a specific timestamp, the domain experts were wondering if the decision-critical features are also consistent in differentiating images from different classes (e.g., digit 0 vs. 1). For simple images (from MNIST), we found that the decision-critical features were more stable in later training stages. Figure 4.16c shows the comparison of vertical/italic digit 0, 1, 2 and 3 in epoch 20. Filter 2, 6 are always very important in differentiating vertical and italic digits, which indicates the learned decision-critical features are quite stable. However, in a relatively earlier stage, such as epoch 5 (Figure 4.16d), the importance of filter 2 is different in different classes, though filter 6 is always very important.

For complicated images (from QuickDraw), however, we found the decision-critical features varied even in later training stages. The last two rows in Figure 4.16g show the comparison of pairs of *apples* and *smiley-faces* (across-class), in epoch 24. Filter 6, 5 replaced filter 1, 4 and became the most important filters in differentiating small (oval) and large (circular) *smiley-faces*, and we did not observe consistent decision-critical features across classes in QuickDraw.

4.6.3 Feedback from Domain Experts

We conducted case study workshops with three domain experts ($E1$, $E2$, $E3$), who are senior researchers with 3+ years experience in deep learning, and 5$\sim$10 years experience in machine learning. We used a guided exploration and think-aloud approach as the study protocol. We first went over the high-level goals and introduced the components of *GANViz*, and then guided them to walk through the case study scenarios with both datasets, as explained in Section 4.6.2. The walk-through was in an interactive way to think aloud the insights, pros and cons of the design. The study ended up with an exit interview to collect feedback and suggestions. We summarized the excitements, frustrations and comments emerged in the studies as follows.

Overall, all experts appreciated the rich details behind the scene of GANs' training revealed by *GANViz*. $E2$ stated that the components offered him new insights at different levels, especially, the probability distribution of D and the generated results of G over the training. He also commented that the top-down analysis flow bears the intrinsic logics to understand GAN models. The experts all agreed that "*only looking at loss function is quite limited and sometimes even misleading*". $E1$ "*liked the distribution view most*" and observed some generated drawings with broken strokes in the distribution view even in the later training stages, and she suspected the training should carry on. They all liked the AUC curve for the performance of D, which is usually overlooked in many evaluations. $E3$ mentioned that the two datasets together revealed the strength and weakness of GAN models. Both $E1$ and $E2$ were impressed by the comparative analysis, which showed how the decision features were evolved, and commented "*it is enlightening to know the top filters were playing different roles in D in the two datasets*".

More interestingly, their explorations triggered some discussions on model improvements. The probability view of D inspired $E2$ to think about the way of separating the training data (based on the currently predicted scores) for the next training iteration, as he observed that some real images with poor quality (e.g., bad doodling in QuickDraw) actually distracted the training of D. Both $E1$ and $E3$ noticed that D made incorrect predictions for some creative drawings in the real data (e.g., *smiley-faces* with a stuck-out tongue), and they discussed about extending GANs with the capability of generating and discriminating creative data. $E1$ even threw out a paradigm-shift question: "*is it reasonable to have the one-to-one mapping from the data point in z sample space to the generation space in GANs?*" Further investigations on this could nurture a novel GAN framework.

The experts also mentioned some frustrations and desirable features. One frustration is to understand the heat-map like weights of the fully-connected layers in the *TensorPath* view, because of the "*overwhelming information without sufficient interpretations*". One metric suggested by $E2$ is looking at the variation of the generated images by G over the training process to understand how stable G is. $E3$ suggested that the Diversity metric should also consider the complexity level of real images. $E3$ also mentioned that the feature of problem diagnosing is desirable to help users quickly locate patterns with potential pitfalls, such as detecting the mode collapse issue in certain epochs. $E1$ talked about how to migrate *GANViz* to other domains, like text generative models, in which interpreting the activations for words and sentences would be more challenging.

4.7 Discussion, Limitations, and Future Work

Model quality measurement. Measuring the quality of a GAN model is still an open question in the machine learning community [37]. The loss functions cannot reflect sufficient

details about the model quality, as we have demonstrated in our case studies. Proposing the three new metrics to measure GANs' quality is an initial attempt. The metrics did reveal certain aspects of model quality, however, their effectiveness in different scenarios requires further validations, which is out of the scope of this work. Also, those metrics can be easily replaced by others to plug into our system.

Flipping of decision-critical features. One interesting observation we had (and the domain experts also thought "*quite phenomenal*") is that the model flipped the decision-critical features from time to time, and such flipping occurred more frequently for simple images than complicated ones. One possible explanation is that: as *G* gets stronger, it is hard for *D* to select critical features to make decisions. Therefore, measuring the decision flipping could potentially be a useful metric to monitor the training process and model quality. This requires more explorations with theoretical supports, and we consider it as an important extension of our work.

Representative image selection. We selected images with representative features into *TensorPath* and compared their details in *D*'s network to examine how *D* differentiated them. However, this selection is inevitably subjective. Different people may select different images of digit 1 to represent images with/without italic styles. We consider this as one limitation of our work, and a remedy we have planned for the future is to derive some summary statistics or priority scores for each image to facilitate the selection.

In the future, we would like to generalize *GANViz* from two perspectives: (1) analyzing more complex image data; (2) extending to other types of GANs. For the first part, we target on images with higher resolutions and more complex features. As we can easily scale/-pad/aggregate images, we do not expect big challenges in adapting *GANViz* to them, though the decision-critical features may become more complicated and not easily interpretable.

For the second part, we plan to explore GANs for other types of data (e.g., texts, sounds). We believe most parts of our framework are still reusable, such as the metrics we proposed to measure GANs' quality and the idea of comparing activations. The challenge appears in the visualization of individual data instances, i.e., how to present a snippet of sounds or texts. We observe that those data can easily be vectorized (e.g., texts are sequences of characters), and thus we should be able to transfer/reorganize them into 2D spaces/shapes and visualize them as image thumbnails in *GANViz*. Moreover, some views may need to undergo a new design process. For example, for text data, the distribution view may use a word cloud to demonstrate the distribution of real/fake texts.

4.8 Conclusion

In this work, we designed and developed *GANViz*, a visual analytics system that helps to evaluate, interpret, and analyze GANs. *GANViz* has five visualization components: the metric view, probability view, and distribution view allow users to conduct model evaluation and interpretation without introducing neural network structures; the *TensorPath* view and activation comparison view enable users to dive into details of network architectures, and perform detailed comparative analysis. All five views are linked together to facilitate users' explorations and visual reasonings. Important findings from real-world datasets and positive feedback from domain experts verified the effectiveness of *GANViz*.

Chapter 5: *DQNViz*: Improve Reinforcement Learning Agents with Visual Analytics

Deep Q-Network (DQN), as one type of deep reinforcement learning model, targets to train an intelligent agent that acquires optimal actions while interacting with an environment. The model is well known for its ability to surpass professional human players across many Atari 2600 games. Despite the superhuman performance, in-depth understanding of the model and interpreting the sophisticated behaviors of the DQN agent remain to be challenging tasks, due to the long-time model training process and the large number of experiences dynamically generated by the agent. In this work, we propose *DQNViz*, a visual analytics system to expose details of the blind training process in four levels, and enable users to dive into the large experience space of the agent for comprehensive analysis. As an initial attempt in visualizing DQN models, our work focuses more on Atari games with a simple action space, most notably the Breakout game. From our visual analytics of the agent's experiences, we extract useful action/reward patterns that help to interpret the model and control the training. Through multiple case studies conducted together with deep learning experts, we demonstrate that *DQNViz* can effectively help domain experts to understand, diagnose, and potentially improve DQN models.

5.1 Introduction

Recently, a reinforcement learning (RL) agent trained by Google DeepMind [73, 74] was able to play different Atari 2600 games [19] and achieved superhuman level performance. More surprisingly, the superhuman level performance was achieved by taking only the game screens and game rewards as input, which makes a big step towards artificial general intelligence [36]. The model that empowers the RL agent with such capabilities is named Deep Q-Network (DQN [74]), which is a deep convolutional neural network. Taking the Breakout game as an example (Figure 5.1, left), the goal of the agent is to get the maximum reward by firing the ball to hit the bricks, and catching the ball with the paddle to avoid life loss. This is a typical RL problem (Figure 5.1, right), which trains an *agent* to interact with an *environment* (the game) and strives to achieve the maximum reward by following certain strategies. Through iterative trainings, the agent becomes more intelligent to interact with the environment to achieve high rewards.

Despite the promising results, training DQN models usually requires in-depth know-how knowledge due to the following reasons. First, different from supervised/unsupervised learnings that learn from a predefined set of data instances, reinforcement learnings learn from the agent's experiences, which are dynamically generated over time. This requires dynamic summarizations of the experiences to achieve a good understanding of the training data. Second, interpreting the behavior of a DQN agent is also challenging. For example, when the agent moves the paddle left, what does the agent really see? Is this an intentional move or it is just a random choice? These questions are important to understand the agent, but cannot be directly answered by model statistics captured from conventional approaches. Finally, DQN models take a certain amount of random inputs during training (e.g., randomly moving the paddle in the Breakout game). The random inputs give the

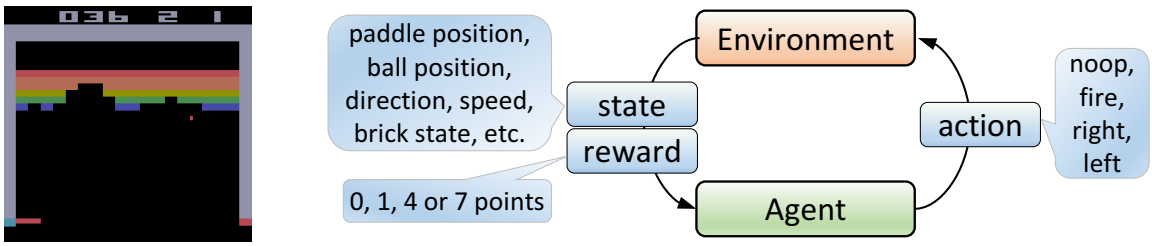

Figure 5.1: The Breakout game and the reinforcement learning problem.

agent more flexibilities to explore the unknown part of the environment, but also prevent the agent from fully exploiting its intelligence. Therefore, a proper random rate is crucial to the training.

In recent years, we have witnessed the success of many visual analytics approaches to understand deep neural networks. These approaches cover supervised (e.g. CNNVis [66]) and unsupervised (e.g. GANViz [106]) deep learning models and are able to expose the models with multiple levels of details. However, to date, few visual analytics works have been reported for deep RL models. Additionally, visualization has played an increasingly important role in model diagnosis and improvement. For example, Liu et al. [65] showed how visualization can help in diagnosing training failures of deep generative models by disclosing how different neurons interact with each other. Through visualization, Bilal et al. [21] demonstrated the effects of the class hierarchy in convolutional neural networks and they successfully improved the models by considering the hierarchy. We believe similar attempts of diagnosing and improving models are also promising for deep RL models.

In this work, we propose *DQNViz* (Figure 5.2), a visual analytics system to understand, diagnose, and potentially improve DQN models. *DQNViz* helps domain experts understand the experiences of a DQN agent at four different levels through visualization. The agent's

experiences are not only the inputs for next training stages, but also the outputs from previous training stages. Therefore, they decide what the agent will learn next, and also reflect what the agent has learned previously. By thoroughly studying those experiences with domain experts using *DQNViz*, we have identified several typical action and reward patterns, which are very useful in understanding the behavior of the agent, evaluating the quality of the model, and improving the performance of the training. For the challenge of understanding the agent's mind when performing an action, we dive into the structure of DQN and use guided back-propagations [98] to expose the features that different convolutional filters extracted. To sum up, the contributions of this work include:

- We present **a visual analytics system**, *DQNViz*, which helps to understand DQN models by revealing the models' details in four levels: *overall training* level, *epoch* level, *episode* level, and *segment* level.

- We propose **a visual design** for event sequence data generated from DQN models. The design can effectively reveal the movement patterns of an agent, and synchronize multiple types of event sequences.

- Through comprehensive studies with domain experts using *DQNViz*, we propose **an improvement in controlling random actions** in DQN models, which is directly resulted from our visual analytics.

Figure 5.2: *DQNViz*: (a) the *Statistics* view presents the overall training statistics with line charts and stacked area charts; (b) the *Epoch* view shows epoch-level statistics with pie charts and stacked bar charts; (c) the *Trajectory* view reveals the movement and reward patterns of the DQN agent in different episodes; (d) the *Segment* view reveals what the agent really sees from a selected segment.

5.2 DQN Model and Model Challenges

Reinforcement learning (RL) aims to generate an autonomous agent interacting with an environment to learn optimal actions through trial-and-error. Researchers have developed three main approaches to address RL problems: value function based approaches [16, 110], policy based approaches [55], and actor-critic approaches [56]. Different approaches have their respective merits and frailties, which have been thoroughly discussed in [15, 16]. Our work focuses on DQN [73, 74], a value function based approach, to present an initial effort in probing RL problems with a visual analytics approach. DQN learns a Q-value

function [16, 110] for a given state-action pair with deep neural networks to handle the large number of input states (e.g., playing Atari games). We explain it with details in Section 5.3.

Recently, there are several important extensions of DQN models. Wang et al. [109] proposed dueling networks to learn a value function for states and an advantage function associated with the states, and combined them to estimate the value function for an action. Double DQN [41] tackles the over-estimation problem by using double estimators. Another important extension is the prioritized experience replay [92] that samples important experiences more frequently. Other extensions to reduce variability and instability are also proposed [12, 43, 45].

Despite these advances, there are several challenges to understand and improve DQNs. The first one is to understand the long-time training process. A DQN infers the optimal policy by enormous trial-and-error interactions with the environment, and it usually takes days/weeks to train the model [25]. It is not easy to effectively track and assess the training progress, and have intervention at the early stages. Secondly, it is not clear what strategies an agent learns and how the agent picks them up. The agent generates a huge space of actions and states with strong temporal correlations during the learning. It is helpful yet challenging to extract the dominant action-state patterns and understand how the patterns impact the training. Thirdly, it is a non-trivial task to incorporate domain experts' feedback into the training. For example, if experts observe some good/bad strategies an agent learned, they do not have a tool to directly apply such findings to the training. Finally, similar to other deep learning models, it needs numerous experiments to understand and tune the hyper-parameters of DQNs. One example is to understand the trade-off between explorations and exploitations [34], which is controlled by the exploration rate [79]. In this work, we try to shed some light on these challenges through a visual analytics attempt.

5.3 Background on Deep Q-Networks (DQN)

DQN [73, 74], as one type of reinforcement learning, aims to train an intelligent *agent* that can interact with an *environment* to achieve a desired *goal*. Taking the Breakout game as an example (Figure 5.1, left), the environment is the game itself, and it responds to any action (e.g., moving left) from an agent (the trained player) by returning the state of the game (e.g., paddle position) and rewards. With the updated state and achieved reward, the agent makes a decision and takes a new action for next step. This iterative interaction between the agent and environment (Figure 5.1, right) continues until the environment returns a terminal state (i.e., game-over), and the process generates a sequence of states, actions, and rewards, denoted as: $s_0, a_0, r_1, s_1, a_1, r_2, ..., r_n, s_n$. The desired goal is maximizing the total reward achieved by the agent.

How to maximize the total reward? The total reward for one game episode (i.e., from game-start to game-over) is: $R{=}r_1 + r_2 + ... + r_n$. Suppose we are at time t, to achieve the maximum total reward, the agent needs to carefully choose actions onwards to maximize its future rewards: $R_t{=}r_t + r_{t+1} + ... + r_n$ (nothing can be done for the previous $t-1$ steps as they have already happened). To accommodate the uncertainty introduced by the stochastic environment, a discount factor, $\gamma \in [0, 1]$, is usually used to penalize future rewards. Therefore, $R_t{=}r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} ... + \gamma^{n-t} r_n{=}r_t + \gamma R_{t+1}$, i.e., the maximum reward from time t onwards equals the reward achieved at t plus the maximum discounted future reward. Q-learning [110] defines the maximum future reward as a function of the current state and the action taken in the state, i.e.,

$$Q(s,a){=}r{+}\gamma \max_{a'} Q(s',a'), \text{ and } s'/a' \text{ is the state/action after } s/a. \tag{5.1}$$

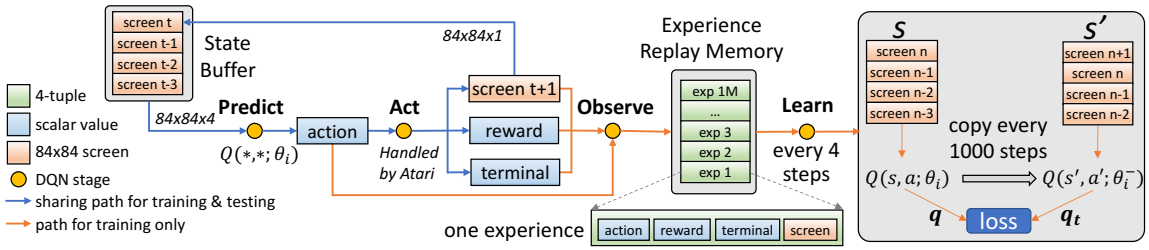

Figure 5.3: The four stages of DQN: *Predict*, *Act*, *Observe*, and *Learn*.

This equation is well known as the Bellman equation [20]. The problem of maximizing the total reward is to solve this equation now, which can be conducted through traditional dynamic programming algorithms.

Why DQN is needed? One problem in solving the Bellman equation is the algorithm complexity, especially when the number of states is large. In Breakout, the states should reflect the position, direction, speed of the ball and the paddle, the remaining bricks, etc. To capture such information, RL experts use four consecutive game screens as one state, which contains both static (e.g., paddle position) and dynamic (e.g., ball trajectory) information. As a result, each state has $84\times84\times4$ dimensions (each screen is a gray scale image of resolution 84×84), and the total number of states is $256^{84\times84\times4}$. Solving the Bellman equation with input in this scale is intractable. DQN, which approximates $Q(s,a)$ through a deep neural network, emerges to be a promising solution.

How does DQN work? The core component of DQN is a Q-network that takes screen states as input and outputs the q-value (expected reward) for individual actions. One popular implementation of the Q-network is using a deep convolutional neural network, which shows strong capabilities for image inputs [96]. The DQN framework consists of four major stages, as shown in Figure 5.3:

- The *Predict* stage (conducted by the Q-network with its current parameters θ_i) takes the latest state (4 screens) as input and outputs the predicted reward for individual actions. The action with the maximum reward, i.e., $arg\max_a Q(s,a;\theta_i)$, is the predicted action; and the maximum reward is the predicted q(uality) value, i.e., $q{=}\max_a Q(s,a;\theta_i)$.

- The *Act* stage is handled by the environment (an Atari game emulator, we used ALE [19] in this work). It takes the predicted action as input and outputs the next game screen, the resulted reward, and whether the game terminates or not. The new screen will be pushed into the State Buffer (a circular queue storing the latest four screens) and constitute a new state with the three previous screens, which is the input for the next *Predict* stage.

- The *Observe* stage updates the Experience Replay memory (ER, a circular queue with a large number of experiences) by pushing a new tuple (of the predicted action, the reward of the action, the new screen, and the terminal value) as an experience into the ER.

- The *Learn* stage updates the Q-network by minimizing the following loss at iteration i [74], i.e., the mean square error between q and q_t:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim ER}\left[\left(r+\gamma\max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right] \tag{5.2}$$

  where (s,a,r,s') are random samples from the ER and θ_i^- are the parameters of the Q-network used to generate the *target* q at iteration i, i.e., $q_t{=}r{+}\gamma\max_{a'} Q(s',a';\theta_i^-)$. To stabilize q_t during training, θ_i^- are updated much less frequently (every C steps) by copying from θ_i [73, 74] ($C{=}1000$ in Figure 5.3 as well as our model training process).

Exploration and Exploitation Dilemma [34]. In the *Predict* stage, actions are not always from the Q-network. A certain percentage of the actions are randomly generated. The reason is that we should not only exploit the intelligence of the agent to predict actions, but also explore the unknown environment with random actions. Usually, the ratio between exploration and exploitation is dynamically updated in the training (i.e., the value of ε in Equation 5.3 decays over time). However, choosing a proper value for this ratio (to handle the trade-off between exploration and exploitation) is still a very challenging problem.

$$predicted\ action = \begin{cases} random\ action, & \text{with probability } \varepsilon \\ arg\max_a Q(s,a;\theta_i), & \text{with probability } 1-\varepsilon \end{cases} \tag{5.3}$$

5.4 Requirement Analysis and Approach Overview

5.4.1 Design Requirements

We maintained weekly meetings with three domain experts in deep learning for more than two months to distill their requirements of a desired visual analytics system for DQN. All the experts have 3+ years experience in deep learning and 5$\sim$10 years experience in machine learning. Through iterative discussions and refinements, we finally identified the following three main themes of requirements:

R1: Providing in-depth *model statistics* over a training. Having an overview of the training process is a fundamental requirement of the experts, and it is a prerequisite for later detailed analysis. In particular, the experts are interested in the following questions:

- *R1.1: How does the training process evolve, in terms of common statistical summaries? Typical examples of these summaries include: the average rewards per episode, the number of games per epoch, the loss values, etc.*

- *R1.2: What are the distributions of actions and rewards, and how do the distributions evolve over time?* For example, will the action distribution become stable (i.e., a roughly fixed ratio among different actions in an epoch) in later training stages? Is there any relationship between the distributions of actions and rewards over time?

- *R1.3: Can the overview reflect some statistics of the agent's action/movement/reward behaviors?* For example, are there any desired patterns that happen more often than others over time?

R2: Revealing the agent's *behavior patterns* encoded in the experience data. Demonstrating the action/movement/reward patterns of the agent is a strong need from the experts [15, 16], given that few existing tools are readily applicable for this purpose. This requirement includes:

- *R2.1: Revealing the overall action/movement/reward patterns from a large number of steps.* Facing with a large number of experiences, the experts need an effective overview to guide their pattern explorations.

- *R2.2: Efficiently detect/extract patterns of interest to understand the agent's behavior.* It is nearly impossible to scrutinize the numerous data sequences to spot all interesting patterns (merely with visualization). To aid users' exploration, a mechanism of pattern detection/extraction is desirable.

- *R2.3: Being able to present other types of data on-demand to facilitate comprehensive reasoning.* The q, q_t values, random actions, etc., are important context information when analyzing the agent's behaviors. Users should be able to bring them up flexibly.

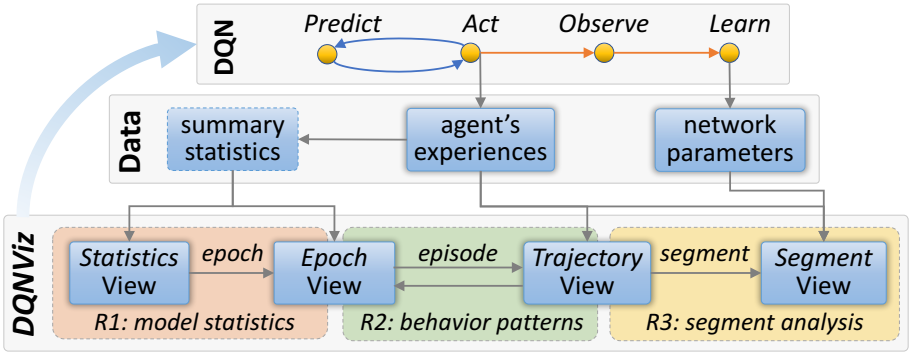

Figure 5.4: The overview of our framework to analyze and improve DQN models.

R3: Empowering *segment analysis and comparisons* by looking through the agent's eyes. This requirement enables users to dive into the architecture of DQN, to analyze how the network works and how it makes the decision (i.e., predicts the action) for an input state.

- *R3.1: Revealing what important features are captured and how they are captured by the agent.* Specifically, the experts are curious about the functionalities of each convolutional filter in terms of extracting features and making action predictions over various experience segments.

- *R3.2: Comparing convolutional filters when processing different experience segments at the same training stage.* For example, domain experts are interested in if the same filter always extracts the same feature when handling different segments in the same epoch.

- *R3.3: Comparing filters when processing the same experience segment at different stages.* This requirement aims to reveal if the agent treats the same experience segment differently in different epochs, and how the agent becomes more and more intelligent.

5.4.2 Approach Overview

Figure 5.4 shows an overview of our approach to understand DQN with *DQNViz*. **First**, we train the DQN model and collect two types of data during the training: (1) the agent's experiences, which are heterogeneous time-varying sequences (Section 5.5); (2) the model losses and network parameters, which are used to assess the model quality and *read the agent's mind* at different training stages. A pre-processing on the experience data is performed to derive useful summary statistics, which include the average reward, average q values, etc. **Second**, the *DQNViz* system takes the two types of data and the derived statistics as input, and presents them to domain experts (Section 5.6). Aligned with the design requirements, the components of *DQNViz* are organized into three modules: (R1) model statistics, (R2) behavior patterns, and (R3) segment analysis, which are implemented through four visualization views. The four views, following a top-down exploration flow, present the collected data at four levels of details: overall training level (*Statistics* view), epoch-level (*Epoch* view), episode-level (*Trajectory* view), and segment-level (*Segment* view). **Lastly**, we demonstrate several case studies in which the knowledge learned from *DQNViz* has helped domain experts to diagnose and improve a DQN model (Section 5.7).

5.5 Training Process and Data Collection

We focus on the Breakout game to present *DQNViz* in this work, as it is one of the most frequently tested/used games in previous works, and a DQN agent can achieve superhuman performance on it.

In Breakout, the agent has five lives in each game episode. Life loss happens when the agent fails to catch the ball with the paddle. The game terminates if the agent loses all five lives. There are four possible actions: no-operation (*noop*), firing the ball (*fire*), moving left

(*left*), and moving right (*right*). The agent receives rewards of 1, 4 and 7 when the ball hits bricks in the bottom two rows, middle two rows, and top two rows respectively. Otherwise, the reward is 0. Each of the six rows has 18 bricks and the maximum reward is 432 (i.e., $(1+4+7)\times 2\times 18$). On the top of the game scene, two numerical values indicate the current reward and the number of lives left (e.g., they are 36 and 2 in Figure 5.1, left).

We trained the DQN model from [9] for 200 epochs. Each epoch contains 250,000 training steps and 25,000 testing steps. The testing part (the blue paths in Figure 5.3) does not update the model parameters, and thus is used to assess the model quality. At each testing step, we collected the following eight types of data:

1. *action*: a value of 0, 1, 2 or 3 representing *noop*, *fire*, *right*, and *left*.
2. *reward*: a value of 0, 1, 4 or 7 for the reward from an action.
3. *screen*: an array of 84×84 values in the range of [0, 255], representing the gray-scale pixel values of the current game scene.
4. *life*: a value in [1, 5] for the number of lives the agent has currently.
5. *terminal*: a boolean value indicating if an episode ends or not.
6. *random*: a boolean value indicating if an action is a random one or not.
7. q: the predicted q (a floating-point value) for the current step.
8. q_t: the target q value, i.e., q_t (see Section 5.3), for the current step.

At the training stage, the random rate ε starts with 1, decays to 0.1 in one million steps (i.e., 4 training epochs), and keeps to be 0.1 to the end. For testing, ε is always 0.05. During

data collection, if an action is a random one, we still use the DQN to derive its q and q_t value, though the action to be executed will be the randomly generated one.

There is an inherent hierarchy in the collected experiences. A ***step*** that composed by the eight types of data is an atomic unit. A ***segment*** is a consecutive sequence of steps in an episode with a customized length. An ***episode*** includes all steps from a game-start to the corresponding game-end (five lives). A testing ***epoch*** contains 25,000 steps (experiences are collected during the test stage only). In summary, the relationship for them is: $step \subseteq segment \subseteq episode \subseteq epoch$.

5.6 Visual Analytics System: *DQNViz*

Following the requirements (Section 5.4.1), we designed and developed *DQNViz* with four coordinated views, which present the data collected from a DQN model at four different levels in a top-down order: overall training level (*Statistics* view), epoch-level (*Epoch* view), episode-level (*Trajectory* view), and segment-level (*Segment* view).

5.6.1 *Statistics* View: Training Process Overview

The *Statistics* view shows the overall training statistics of a DQN model with line charts and stacked area charts. Those charts present the entire DQN training process over time, along the horizontal axis.

The line charts (revealing the trend of different summary statistics over the training) are presented as small-multiples [103] (R1.1, R1.3). As shown in Figure 5.2a, the five line charts track five summary statistics (from left to right): *average rewards per episode*, *number of games per epoch*, *average q value*, *loss value*, and *number of bouncing patterns*. Users are able to plug other self-defined statistics into this view, such as *maximum reward per episode*, *number of digging patterns*, etc.

The two stacked area charts demonstrate the distribution of actions and rewards over time (R1.2). The evolution of action/reward distributions provides evidence to assess the model quality. For example, by looking at the distribution of reward 1, 4, and 7 in Figure 5.2-a2, one can infer that the model training is progressing towards the correct direction, as the high rewards of 4 and 7 take increasingly more portions over the total reward. To the rightmost, the even distribution of reward 1, 4, and 7 reflects that the agent can hit roughly the same number of bricks from different rows, indicating a good performance of the agent.

All charts in this view are coordinated. When users hover one chart, a gray dashed line will show up in the current chart, as well as other charts, and a pop-up tooltip in each individual chart will show the corresponding information, as shown in Figure 5.2a (the mouse is in the stacked area chart for the reward distribution). Meanwhile, the hovering event will trigger the update in the *Epoch* view (presented next).

5.6.2 *Epoch* View: Epoch-Level Overview

The *Epoch* view presents the summary statistics of the selected epoch with a combined visualization of a pie chart and a stacked bar chart, as shown in Figure 5.2b (R1.2, R2.1). The pie chart shows the action/reward distribution of all steps in the current epoch; whereas the stacked bar chart presents the action/reward distribution of each individual episode in the epoch. As shown in Figure 5.2-b2, there are 20 episodes in the current epoch (one stacked bar for one episode), and the stacked bars are sorted decreasingly from left to right to help users quickly identify the episode with the maximum number of steps/rewards.

The two types of charts are linked with each other via user-interactions. For example, when hovering over the white sector of the pie chart (representing *noop* actions), the *noop* portion of all stacked bars will be highlighted, as the area of the sector is the sum of all

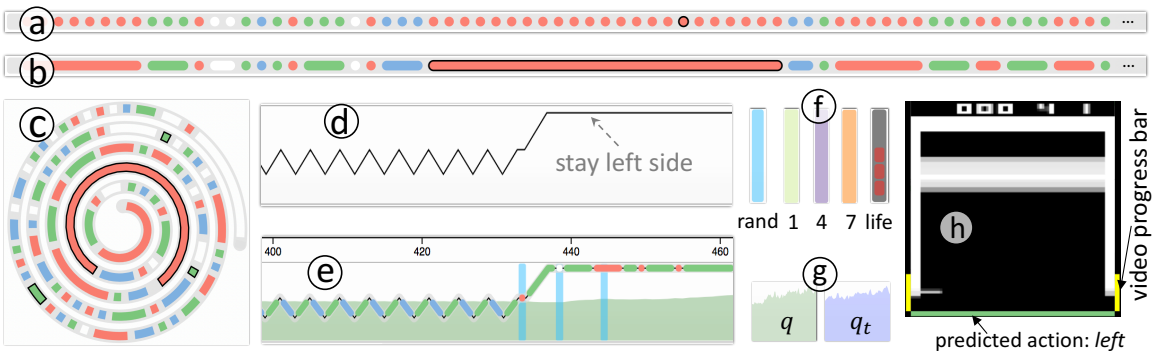

Figure 5.5: Different designs for the event sequence data from DQN.

white regions from the stacked bars. By default, the distributions of actions and rewards are presented in this view, as these two are of the most interest. But, users can also plug in other variables (e.g., the step distribution in different lives of the agent) by modifying the configuration file of *DQNViz*.

All views of *DQNViz* share the same color mapping. For example, the red color always represents the *fire* action; and the purple color always indicates 4-point reward. Therefore, the pie charts (with text annotation in different sectors) also serve as the legends for *DQNViz*.

5.6.3 *Trajectory* View: Episode-Level Exploration

The *Trajectory* view aims to provide an overview of all steps in one epoch, and reveal the action/reward patterns in all episodes of the epoch. Also, it is designed to facilitate users' detailed examinations on the eight types of data (see Section 5.5).

5.6.3.1 Visual Design for DQN Event Sequence Data

The data collected from one episode is essentially an event sequence (also called as a *trajectory* by experts [25]). We, therefore, start the design of this view with event sequence visualization solutions, and several key design iterations are briefly discussed as follows.

Our first design presents one episode with a line of circles representing different types of actions, as shown in Figure 5.5a. The color white, red, blue, and green represent *noop*, *fire*, *right*, and *left* respectively. The circles with the black stroke represent the actions with a reward. Many previous works (e.g. [40, 115]) have adopted this type of design, as it is straightforward and easy to understand. However, the design cannot reflect the agent's movement patterns and it is also not scalable. To overcome these limitations, we tried to merge consecutive circles representing the same actions as one line (Figure 5.5b), which can effectively reveal the repeat of different actions. We also explored the spiral layout to address the scalability issue (Figure 5.5c). The spiral layout can present one entire episode on the screen, but compactly arranging all episodes (in one epoch) with varying lengths becomes a problem, as the spiral layout for those episodes will have different widths and heights.

To reveal the movement patterns of the agent (R2.2), we visualize the displacement of the paddle to the right boundary over time (Figure 5.5d). The design is based on our observation that the moving behavior of the agent is visually reflected by the position of the paddle. For example, the oscillation in the first half of Figure 5.5d indicates that the agent keeps switching between *left* and *right* to adjust the position of the paddle. Also, this design is scalable. It allows us to flexibly compress the curve horizontally, like a spring, to get an entire view of a long episode (R2.1). One limitation is that the paddle positions, though reflecting the agent's movement patterns, cannot accurately reflect the action taken by the agent. For example, in the right half of Figure 5.5d, the paddle stays at the leftmost position (the top side). The action that the agent is taking now can be *noop*, *fire* (*fire* does not change the paddle position), or *left* (the paddle is blocked by the left boundary and cannot go further). To address this issue, we overlay the action circles/lines onto the curve. From

the visualization (Figure 5.5e), we found that the agent takes three types of actions (i.e., *left*, *noop*, and *fire*) in the right half of Figure 5.5d, and most of the time, the agent is repeating the *left* action.

Lastly, this design can synchronize other types of data with the action data (R2.3). As shown in Figure 5.5e, some actions are highlighted with background bars in cyan, indicating they are random actions. Bars with color light green, purple, and orange encode the reward of 1, 4, and 7 respectively (Figure 5.5f, 5.2-c5). We also design glyphs for actions with a life loss, as shown in the rightmost of Figure 5.5f. This glyph is a gray bar with 0$\sim$4 dark red rectangles inside, indicating the number of remaining lives after the life loss action. The *terminal* information is also encoded in this glyph (gray bars with 0 dark red rectangle). The q and q_t values are presented as transparent area charts with green and blue color respectively in the background (Figure 5.5g, 5.5e). When users click the action circles/lines, a video clip (Figure 5.5h) will pop up and show the *screen* data (a sequence of screens). The two vertical yellow bars on the two sides of the video are the progress bars. We found they are very useful in reflecting the progress of static videos, e.g., when the agent is repeating the *noop* action. The color bar on the bottom shows the predicted action. It changes from frame to frame when the video is playing. To avoid visual clutters in the *Trajectory* view, users can show/hide the eight types of data on-demand through a set of check-box widgets (Figure 5.2-c2).

5.6.3.2 Segment Clustering and Pattern Mining

It is not a trivial task to visually identify the common patterns from the large number of actions in one epoch. To address this issue, we adopted two techniques of segment clustering (R2.1) and pattern mining (R2.2). To cluster segments, we first cut the episodes in one epoch into many smaller segments and cluster them using hierarchical clustering [42]. A

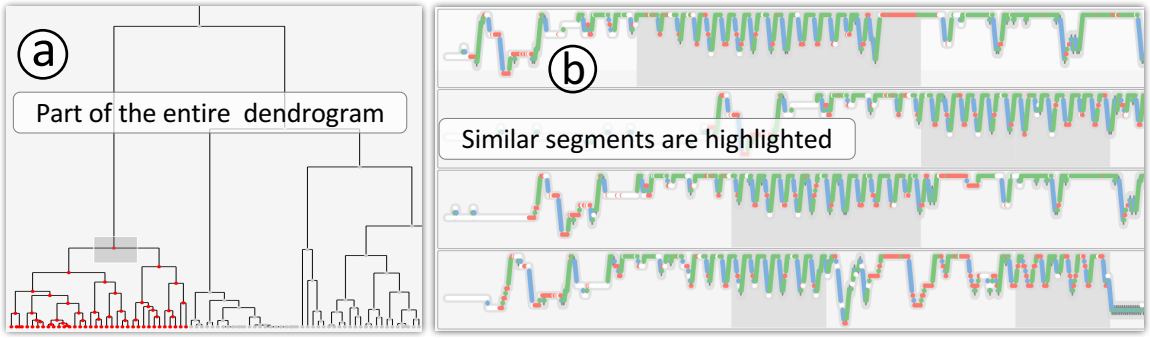

Figure 5.6: Clustering segments (the segment length is 100) in epoch 120.

segment is expressed by a sequence of values, indicating the paddle positions. The segment length is set to 100 by default, but it can be adjusted on-demand from the configuration file of *DQNViz*. To better quantify the similarity between segments, we used the Dynamic Time Warping algorithm (DTW) [90]. Specifically, for a pair of segments (i.e., two temporal sequences), the DTW algorithm can find the best temporal alignment between them and derive a more comprehensive similarity score. Applying DTW to all pairs of segments will derive a similarity matrix for all segments in one epoch, which is the input of the clustering algorithm. When clicking the "Tree" button in Figure 5.2-c1, a dendrogram visualization of the clustering results will show up (Figure 5.6a). Selecting different branches of the dendrogram will highlight different clusters of segments in the *Trajectory* view (Figure 5.6b).

We found some typical movement patterns of the agent while exploring the clustering results. By defining those patterns and mining them in other epochs, we can provide more insight into the agent's behaviors. The regular expression [11, 102] is used to define a pattern as it is simple and flexible. For example, an action sequence can be expressed as a string (of 0, 1, 2, and 3) and a predefined regular expression can be used to search on the string to find

when and where a particular pattern happens. Table 5.1 presents two example movement patterns, i.e., *repeating* and *hesitating* (frequently switching between *left* and *right*).

Table 5.1: Formalizing action/reward patterns with regular expressions.

Pattern	Regular Expression	Explanation
repeating	`0{30,}`	repeating *noop* (0) for at least 30 times.
hesitating	`(20*30*){5,}`	switching *left* (2) and *right* (3) for at least 5 times. There might be multiple *noop* actions between the *left* and *right*.
digging	`10+10+40+` `40+70+70+`	the two 1s, 4s, and 7s are where the ball hits the bottom, middle and top two rows of bricks, the 0s in between are the round trip of the ball between the paddle and bricks.
bouncing	`(70+){5,}`	hitting top two rows of bricks for at least 5 times.

Reward patterns can be defined similarly. For example, we found that the agent becomes very smart in later training stages, and it always tries to dig a tunnel through the bricks, so that the ball can bounce between the top boundary and the top two rows to achieve 7-point rewards. The *digging* and *bouncing* pattern can be defined using regular expressions, as shown in Table 5.1. We can also visually verify the patterns from the bars in the *Trajectory* view (Figure 5.2-c5). It is worth mentioning that the regular expression for each pattern can be relaxed. For example, the *digging* pattern in Table 5.1 can be relaxed to `10+40+40+70+`. More patterns can be defined similarly in this way.

Tracking patterns is an effective way to provide insight into the evolution of the agent's behaviors. For example, the decreasing of *repeating* indicates the agent became more flexible in switching among actions. The increasing of *digging* reflects the agent obtained the trick of digging tunnels. The number of a pattern can be defined as a metric (R1.3) and plugged into the *Statistics* view for overview (Figure 5.2-a1).

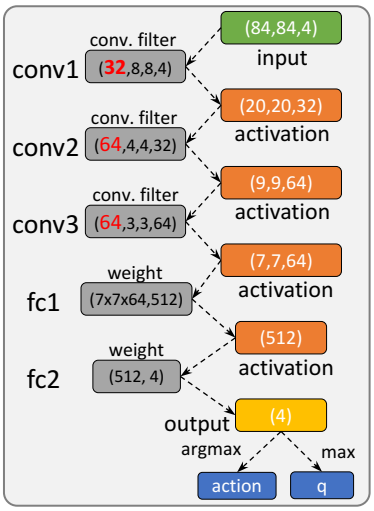

Figure 5.7: The structure of DQN, which contains three convolutional layers and two fully-connected layers. The first, second, and third convolutional layers contain 32, 64, and 64 convolutional filters, respectively (160 filters in total).

5.6.4 *Segment* View: Segment-Level Interpretation

5.6.4.1 See through the agent's eyes

The *Segment* view targets to reveal what the agent really sees and how it gains such vision in the screen states from a trajectory segment. To achieve this, we dive into the DQN structure [74] (Figure 5.7) and reveal what have been extracted by the neural network (R3.1). The input of the network is a state of size $84 \times 84 \times 4$ (in green), and the output is a vector of four values (in yellow) representing the predicted rewards for the four actions. Among the four values, the maximum one is the predicted q, and its index is the predicted action. Between the input and output are three convolutional and two fully-connected layers.

The filters in the convolutional layers are the basic computational units that extract features from the input states. We focus on them to interpret what the agent sees. The four numbers in each gray rectangle of Figure 5.7 represent *the number of filters*, *the width and*

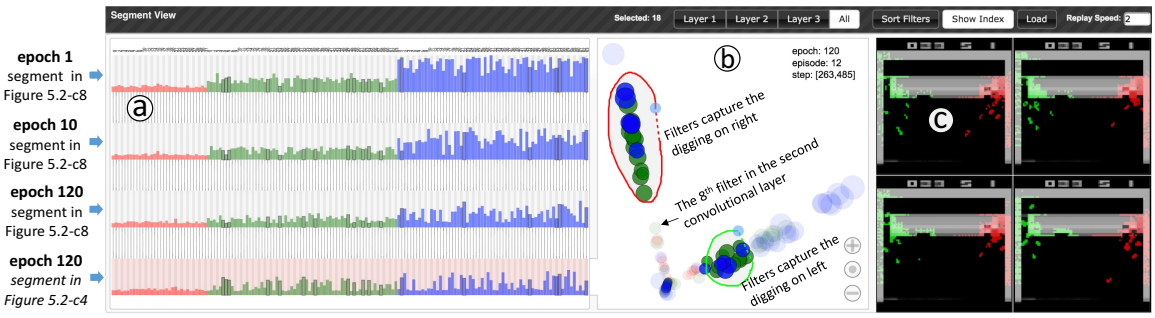

Figure 5.8: *Segment* View: (a) bar charts view, (b) PCA view with lasso selections, (c) aggregated screens with saliency maps.

height of each filter, and *the number of channels*. There are *32*, *64*, and *64* filters in the first, second, and third convolutional layer (160 in total).

Algorithm 2 Picking out the maximumly activated state (`max_state`) from a segment (`screens`), and generating the corresponding saliency map (`map`) of the state for each convolutional filter in each layer.

1: $screens = [s_1, s_2, ..., s_n]$ // input: a segment of n screens
2: $states = [\{s_1, s_2, s_3, s_4\}, ..., \{s_{n-3}, s_{n-2}, s_{n-1}, s_n\}]$ // forming n-3 states
3: **for** $i = 0;\ i < layers.length;\ i++$ **do**
4: **for** $j = 0;\ j < layers[i].filters.length;\ j++$ **do**
5: $activations = DQN.f\_prop(layers[i].filters[j],\ states)$
6: $max\_idx = argmax(activations)$
7: $max\_state = states[max\_idx]$
8: $max\_activation = activations[max\_idx]$
9: $map = DQN.b\_prop(layers[i].filters[j],\ max\_activation)$
10: $output[i][j] = blend(max\_state, map)$ // algorithm output
11: **end for**
12: **end for**

Algorithm 2 shows how we visualize what features each filter extracted from an input state. The algorithm includes three main steps. **First**, given a segment, we find the state that is maximumly activated by each of the 160 filters. Specifically, for each filter in each layer,

we first apply forward propagations on all the input states of the segment (Algorithm 2, *line 5*) to get the state (`max_state` in *line 7*) that is maximumly activated by the filter. **Then**, using the activation of this state (`max_activation` in *line 8*), we perform guided back-propagations to generate a saliency map (`map` in *line 9*) for the state. The saliency map will have the same size with the input state (i.e., $84 \times 84 \times 4$), and the pixel values in the map indicate how strong the corresponding pixels of the input state have been activated by this filter (the back-propagation computes the gradient of the maximum activation on the input state, details can be found in [98]). **Finally**, we blend the input state with its corresponding saliency map (`blend` in *line 10*). The blending image can expose which region of the input state has been seen by the current filter (like an eye of the agent). For example, Figure 5.10a shows the blending result of the second screen of a state with its corresponding saliency map. We can see that the filter extracts the ball from the screen.

5.6.4.2 Analysis components with the agent's eyes

The *Segment* view enables users to analyze the 160 filters along with the 160 states they have maximumly activated in three sub-views (Figure 5.8): a parallel bar charts view, a Principal Component Analysis (PCA) view, and a view showing the average state of the input segment.

The bar charts view (Figure 5.8a) shows the size of features that individual convolutional filters extracted from the input states. Each row is a bar chart representing one segment (four rows are in the view in Figure 5.8a). Each bar in each row represents a filter from the DQN, and the height of the bar indicates the size of the feature that the filter extracted (i.e., the number of activated pixels in the corresponding saliency map, see Algorithm 2). The color red, green, and blue indicate the filter is from the first, second, and third layer respectively. Different rows represent the filters for different source segments selected by users, and the

corresponding filters are linked together across rows for comparisons (R3.2, R3.3). Clicking the "Sort Filters" button in the header of this view will sort the bars based on their height. Users can focus on filters in layer 1, 2, 3 or all of them for analysis by interacting with the widgets in the header (currently all filters are in analysis). The row with the pink background (the 4th row) is the segment in selection and currently analyzed in the other two sub-views. Clicking on different rows will update contents of the other two sub-views.

The PCA view (Figure 5.8b) presents how the filters can capture similar or dissimilar features from the screen states (R3.1). It projects the 160 convolutional filters of the selected row based on their saliency map, i.e., reducing the $84 \times 84 \times 4$ dimensional saliency maps to 2D using PCA. Each circle in this view represents one filter, and the color red, green, and blue indicate the filter is from the first, second, and third layer. The size of circles encodes the size of features extracted by the filters. Moreover, the circles in this view are coordinated with the bars of the selected row in the *bar charts view*. Clicking any bars/circles will pop up a four-frame video showing the blending result of the input state and the corresponding saliency map. Figure 5.10a shows the second frame of the pop-up video when clicking the 8th filter in the second layer (position indicated in Figure 5.8b). Semantic zoom (the user interfaces in the bottom right of this view) is also enabled to reduce visual clutters. This interaction can enlarge the view while maintaining the size of circles, which will mitigate the overlap of the circles by increasing the distances among them.

The four screens (Figure 5.8c) show the aggregated results of states from a selected segment. For example, the top-left screen is the result of averaging the first screen from all input states of the selected segment. We also introduce some interactions to help users easily observe which part of the screen is seen by the filters. For example, when users select different convolutional filters from the *bar charts view* (via brushing) or from the *PCA view*

(via lasso selection), the union of the corresponding saliency maps will be highlighted on the aggregated screens. In Figure 5.8c, the two selected clusters of filters (Figure 5.8b, circles in the green and red lasso) capture the features that the ball is digging the left and right corner of the bricks respectively (R3.1).

5.7 Case Studies

We worked with the same three domain experts (E1, E2, and E3) in the design stage of *DQNViz* on several case studies. Here, we present two of them: one emphasizes how *DQNViz* can help the experts understand the DQN training process; the other shows how the experts use *DQNViz* to diagnose and improve the exploration rate of the DQN model.

5.7.1 Unveiling the Model Training Evolution

We use an exploration scenario in this section to demonstrate how *DQNViz* can expose the four levels of details in a typical DQN training.

Overall training statistics. The experts all confirmed that the training was successful based on observations from the *Statistics* view in Figure 5.2a. First, both *average_reward* and *average_q* value are increasing, indicating the agent expected and was able to receive higher and higher rewards. Meanwhile, the decreasing number of games (*nr_games*) shows the agent could survive longer in individual episodes (R1.1). Second, the experts observed that the agent became more intelligent and strategic to receive higher points as the training evolves (R1.3). For example, in Figure 5.2-a1, the agent adopted more *bouncing* patterns that can collect high points along with the training. Also, the reward distribution in Figure 5.2-a2 echoes the same observation as the agent obtained an increasing amount of 4 and 7 points over time (R1.2).

One interesting observation pointed by one expert is that the action distribution is very diverse even in the later training stages (R1.2). However, the agent can still achieve high rewards with diverse action distributions. This indicates that by merely looking at the action distribution, it is difficult to understand why and how the agent achieved high rewards. It also calls for the investigation of detailed action patterns, which is conducted using the *Trajectory* view later on by the experts.

The experts also spotted some abnormal epochs. For example, the reward distribution in epoch 37 did not follow the general trend in the stacked area chart. This observation led the experts to select epoch 37 and drill down to explore its details in the *Epoch* view.

Epoch statistics. From the statistics shown in the *Epoch* view, the experts suspected that the agent repetitively moved the paddle left and right in epoch 37, but those moves were mostly useless. As shown in Figure 5.9a, the *left* and *right* actions take 31% and 47% of the total 25,000 steps in this epoch (R1.2). Meanwhile, it can be imagined that the agent mostly hit bricks in the bottom two rows as the achieved rewards were mostly 1-point reward (Figure 5.9b). From the stacked bar charts in Figure 5.9a and 5.9b, the experts observed that most episodes in this epoch lasted for less than 0.6k steps (much shorter than normal episodes) and achieved less than 5 points. Therefore, the large number of *left* and *right* did not contribute much to a better performance, but led to low rewards and short life cycles (R2.1).

Action/Reward patterns in episodes. Drilling down to the agent's movement patterns in the *Trajectory* view, the experts confirmed that the large number of *left* and *right* actions in epoch 37 were mostly useless. As shown in Figure 5.9c, the agent repeated a lot of *hesitating* and *repeating* patterns, which contributed nothing to achieving rewards. By highlighting the random actions (cyan bars in Figure 5.9c), the experts also learned that random actions

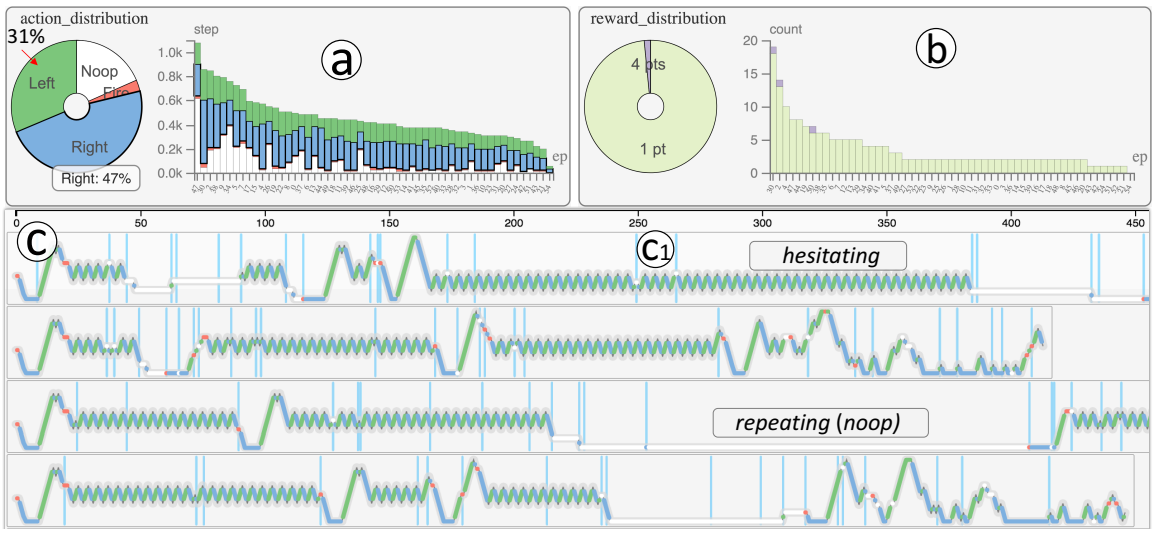

Figure 5.9: The *Epoch* (a, b) and *Trajectory* (c) view of Epoch 37.

play an important role in terminating the *hesitating* and *repeating* patterns. Moreover, terminating those patterns may need multiple random actions (e.g., the two random actions in Figure 5.9-c1 are not sufficient to terminate the *hesitating* pattern).

Except the failure cases, the experts were also interested to know the agent's behaviors in normal epochs. Among the many epochs whose statistics follow the general trend, the experts randomly selected epoch 120 to explore. A few *hesitating* and *repeating* patterns can still be found in this epoch as shown in Figure 5.2-c3 and 5.2-c7. Moreover, many *digging* and *bouncing* patterns were found in this normal epoch. From the exploration of the hierarchical clustering result (Figure 5.6), the experts found many similar movement patterns shown in Figure 5.2-c4 and 5.2-c6. By zooming into Figure 5.2-c4, visualizing the reward data, and replaying the video clips, the experts found that the agent was digging a tunnel through the bricks and a *bouncing* pattern appeared right after the *digging* pattern, as shown in Figure 5.2-c5 (R2.2, R2.3). In the digging pattern, the agent periodically moved

the paddle to catch the ball and received 1-point, 4-point, and 7-point rewards with regular step intervals. In the successive *bouncing* pattern, the agent received 7-point rewards very frequently.

It is also obvious that the q value kept increasing in *digging*, but immediately started decreasing after the tunnel was created. The experts interpreted this observation as follows. During digging, the agent can see the progress of the tunnel (from input states), and the expected reward keeps increasing as the tunnel will potentially result in *bouncing* (i.e., keep getting high rewards). However, when the *bouncing* starts, the bricks (especially in the top two rows) keep being destroyed, therefore, the expected future reward starts decreasing. The experts also found that when *bouncing* happens, the paddle mostly stays at the leftmost position, which is *"quite phenomenal"*. This deepens their understanding on the successful playing strategies of the agent and they would like to conduct further theoretical investigations on this.

Segment-level investigations. Intrigued by *digging* and *bouncing* patterns, the experts were curious about how the agent can observe states and make the action predictions. This motivated them to select some segments of interest and analyze them in the *Segment* view (R3.1). From there, they found that filters from higher convolutional layers usually have stronger activations and capture larger and more diverse features from the input states. For example, to further investigate the *digging* pattern in Figure 5.2-c4, the experts selected that segment (step 263∼485 of episode 12 in epoch 120) into the *Segment* view (the fourth row in Figure 5.8a). By exploring it, they found that: (1) the height of bars representing filters from layer 1 (red), 2 (green), and 3 (blue) shows an increasing trend (Figure 5.8a); (2) the circles representing filters from three layers show an inner-to-outer layout in the PCA view (Figure 5.8b).

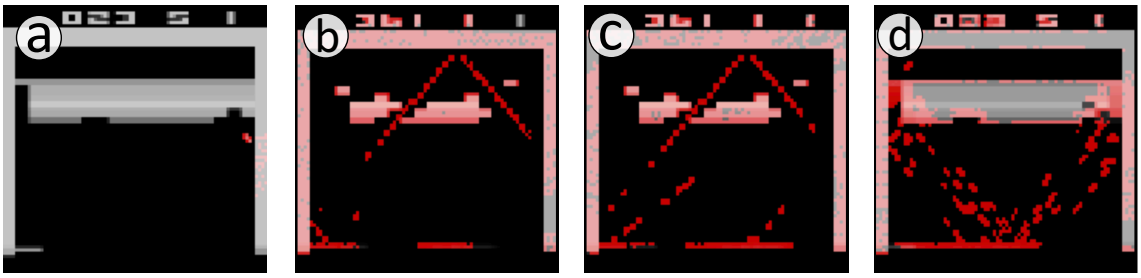

Figure 5.10: (a) Blending a state with its saliency map. (b,c,d) what the agent sees from the segment in the 1st, 2nd, and 4th row of Figure 5.8a.

By examining the *digging* segment, the experts also learned that the agent dug tunnels from both sides of the bricks and they identified what filters captured the *digging* on different sides. Specifically, the two groups of filters in the green and red lasso of Figure 5.8b are filters that captured the *digging* on the left and right of the scene (Figure 5.8c). From the aggregated saliency map of all filters in Figure 5.10d, it can be seen that the agent moves the paddle between the left boundary and the middle of the scene to catch the ball and dig tunnels from both sides.

Compare segments from the same epoch (R3.2). To have more understandings on the functionality of different filters, the experts also compared the above *digging* segment with other segments in this epoch. For example, one expert selected another segment shown in Figure 5.2-c8, where the agent moved the paddle all the way to the left then to the right. Figure 5.2d shows the fourth average screen and the aggregated saliency map from all filters for this segment. It is clear that the agent tracked the moving path of the ball, i.e., A-B-C-D-E-F, and it moved the paddle all the way to the left then to the right to catch the ball at B and F. By comparing the filters in these two segments, the experts found that certain filters behave similarly, e.g., filter 16 from the third convolutional layer traces the

ball in both segments. However, some filters also have dramatically different functions, e.g., filter 23 in the second layer stares at the top-left tunnel in the *digging* segment; while it traces the ball in the other segment. This result provides evidence that filters in the same stage may not always have the same functions when parsing different states.

Track a segment over time (R3.3). The experts also wanted to understand how filters evolved over time by comparing the same segment across epochs. As shown in the top three rows in Figure 5.8a, one expert tracked the segment in Figure 5.2-c8 in epoch 1, 10 and 120. The blended saliency maps are shown in Figure 5.10b, 5.10c, and 5.2d (the saliency maps from all filters on the fourth average screen, the result of other three average screens are similar). From them, the expert understood that the agent did not have a clear vision on the input states in early stages (Figure 5.10b), and it gradually developed its attention on important parts, e.g., the moving path of the ball (Figure 5.10c, 5.2d).

5.7.2 Optimizing the Exploration Rate (Random Actions)

One expert was very interested in the random actions, especially after he saw that random actions can terminate bad movement patterns in Figure 5.9c. Here, we describe the experiments that we worked with him to diagnose and improve the use of random actions using *DQNViz*.

Experiment 1: No random action. With the certain level of understanding on the DQN model, the expert first hypothesized that random actions are not necessary after the model is well trained. The logic behind this is that an action predicted by a well-trained agent should be better than a randomly generated one. To test this, the expert set the exploration rate (ε in Equation 5.3) to 0, after 200 training epochs, and used the agent to play the Breakout

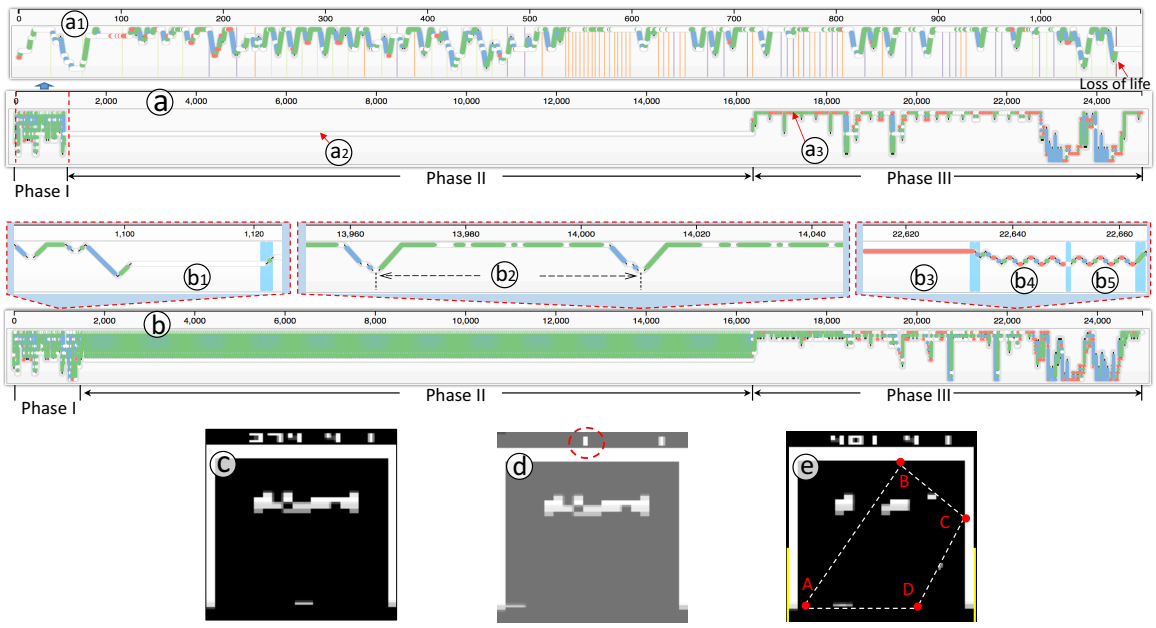

Figure 5.11: (a, b) Result of Experiment 1, 2; (c, d) game screen at a2, a3; (e) ball trajectory for segment b2, the ball follows the loop A-B-C-D-C-B-A.

for 25,000 steps (a testing epoch) to see if anything will go wrong (by default ε=0.05 in testing).

The result of the 25,000 steps in the *Trajectory* view is shown in Figure 5.11a. The expert had two observations: (1) there is only one episode in the 25,000 steps, and the episode is very long; (2) the agent keeps repeating the *noop* action in roughly 60% of the episode.

In detail, the single and long episode can roughly be cut into three phases as labeled in Figure 5.11a. In phase **I** (Figure 5.11-a1), the agent played very well in the first $\sim$1,080 steps, and this phase ended with a life loss. In phase **II**, the agent kept repeating *noop* for $\sim$15,000 steps (i.e., *trapped* by the environment). By looking at the *screen* data (Figure 5.11c) at the position indicated in Figure 5.11-a2, the expert understood that the paddle stays around

the middle of the scene and the ball is not in the scene (as there is no *fire* action). This indicates that the agent does not know that he needs to *fire* the ball at the beginning of a game, but just keeps waiting for the ball. The expert was very surprised about the agent's movement pattern in phase **III**, i.e., how did the agent get out of the trap without the help of random actions? After checking the screen data at the position indicated by Figure 5.11-a3, he realized that the game has crashed actually, as the numbers for reward and life disappear and the entire scene becomes lighter (Figure 5.11d).

Experiment 2: Random actions on demand. From Experiment 1, the expert learned that random actions are necessary. Next, he was wondering if one can control the initiation of random actions when they are really needed. The expert hypothesized that random actions are needed when the agent keeps repeating the same patterns but gets no reward (e.g. *hesitating* in Figure 5.9c, and *noop* in Figure 5.11-a2).

Experiment 2 tested this hypothesis with a Pattern Detection (PD) algorithm, which can be explained as follows. First, a buffer that stores the latest 20 steps is maintained. At each step of the game, if the agent received rewards in the latest 20 steps, no random action is needed. However, if the agent did not receive any reward, but kept repeating the same action/pattern in those steps (detected using regular expressions, see Section 5.6.3.2), a random action would be introduced. As observed before, a repeating pattern usually has a basic repeating unit which is very short, e.g., the basic unit of the *hesitating* pattern in Figure 5.5e is *left-left-right-right* and the pattern length is 4. Experiment 2 checks pattern length from 2$\sim$7, and a random action is introduced if a pattern has been repeated for 3 times. For example, if the latest three actions are 230 (*right-left-noop*) and this pattern can be found 3 times in the latest 9 steps, then the next action will be a random action.

Figure 5.11b shows the result of applying the PD algorithm to DQN. Similarly, only one episode is generated in the 25,000 steps, and the episode can be cut into three similar phases. The expert first noticed that the PD algorithm worked well in terminating the *repeating* of one action in phase **I** (before $\sim$1,800 steps). For example, in Figure 5.11-b1, the *noop* action has been repeated for 20 times, and the agent introduced several random actions and got out of the repeating of *noop* finally.

However, the expert also observed that the agent was trapped by the environment again in phase **II**. After zooming into this phase, the expert found that the agent kept repeating a long pattern with the length of around 50 steps (Figure 5.11-b2). By replaying the game, as shown in Figure 5.11e, he realized that the agent kept moving the paddle between point A and D to catch the ball, and the ball repeated the loop A-B-C-D-C-B-A. No random action was introduced, as the length of the repeating pattern exceeded the threshold (i.e., 7) in the PD algorithm.

The game crashed again in phase **III**. However, by exploring different segments in this phase, we could still see that the PD algorithm worked well in introducing random actions (e.g., Figure 5.11-b3, b4, b5).

Experiment 3: Improved random actions. With the lessons learned in Experiment 2, the expert applied the following changes to the PD algorithm: (1) changing the maximum pattern length from 7 to 50; (2) increasing the buffer size from 20 to 100; (3) introducing a random action if a pattern repeats twice. With these changes, the agent was able to play the game very well and no longer trapped by the environment. In 25,000 steps, the agent played 12 episodes and received 5,223 total rewards. The number of random actions in those steps is 501, which is much less than 1,250 (i.e., 5% of 25,000 in the original setting).

Table 5.2: Statistics of random actions per epoch (averaged over 10 runs).

	steps	episodes	total rewards	random actions
ε=0.05 (5%)	25,000	16.6	4198.6	1269.4
PD Algorithm	25,000	11.4	4899.2	503
ε=0.02 (2%)	25,000	9.9	3780.8	492.1

To quantitatively evaluate this improvement, the expert compared the PD algorithm with other two baselines using ε=0.05 and ε=0.02, as shown in Table 5.2 (results are averaged over 10 runs). As we can see, the PD algorithm worked better than the other two random-only methods. Compared to the method of ε=0.05, the PD algorithm introduced less random actions, but achieved $\sim$700 more rewards in 25,000 steps. Also, it led to fewer life losses, as the number of episodes is less than the baseline using ε=0.05. Compared to the method of ε=0.02, the PD algorithm obtained much higher total rewards in 25,000 steps, though the number of random actions was similar. This comparison further verifies that the PD algorithm can effectively control random actions.

5.7.3 Feedback from Domain Experts

We collected the experts' feedback via in-depth interviews after the case study sessions. Overall, all experts believed that the tool is *"extremely helpful to have actionable insight into the (model) evolution"*, and *"has great potentials to comprehend and improve deep RL models"*.

In terms of aiding their understanding of the model, all experts agreed that the overall training statistics are the basic need as they *"use them in their daily model building practice"*. The overall trend and distribution of DQN-specific metrics (e.g., the number of *bouncing*

pattern) really *"provide a glimpse into the evolution details"* and *"would jump-start their diagnosing towards the hurdle of model training"*.

All experts believed the most useful component is the *Trajectory* view, where they *"could explore, observe, and extract meaningful patterns"* and *"ponder how the agent developed its playing strategies"*. All experts spent the most time on this view to explore useful patterns that could help them comprehend and potentially improve the model. *"It is also quite entertaining to look at the video play-back. This level info is absent in other tools."*, commented by E3. Another example is the three experiments on random actions. Towards the end of that study, E1 concluded that *"the visualization had significantly improved his understanding about random actions"*. He explained that the final choice of *"the (pattern) length to be 50 is not random"*. In fact, that value should be (or close to be) the upper bound for the number of steps that the ball needed for a round trip (between the paddle and bricks).

The experts expressed that the *Segment* view is *"very fun to play with"* and appreciated this view *"provides many details about how the agent parses the game screens"*. E1 and E2 both made a connection between this view and the *"attention mechanism [68, 97]"* by commenting *"it is very enlightening to know layers have attentions over different parts of the screens"*. E1 even proposed a hypothesis that *"filter activation in a complicated network may correspond to different playing strategies"*.

Additionally, the experts also mentioned several desirable features and suggested some improvements. E2 was first confused with the visual encoding of paddle positions in the *Trajectory* view, though later on she understood it with some explanations. She felt *"(there is) a disconnection between the horizontal location in the game interface and the vertical position encoding"*, and *"some labels (at the vertical axis of the Trajectory view) may be helpful"*. For the *Segment* view, two experts thought *"there is a high learning curve"*, and

"it is especially true for the layout of filters". *"Showing network structures and highlighting where those filters come from may help"*, suggested by E1. The three experts also shared their concerns on the generalization of *DQNViz*. E3 pondered *"how this tool can visualize games with a larger action space"* or *"even real games, like AlphaGO [95]"*. These comments are very useful in guiding our further improvements of *DQNViz*.

5.8 Discussion, Limitations, and Future Work

Dimensionality Reduction. The *Segment* view uses a dimensionality reduction algorithm to project the high-dimensional saliency maps (resulted from individual filters) to 2D for visualization and interaction. The use of the PCA algorithm came out in one of our discussions with the domain experts and it turns out that the algorithm is simple, sufficient for our objective, and involves few parameters to tune with. However, it also suffers from many drawbacks as shown in previous works [17, 46], and we do believe that other dimensionality reduction algorithms can be used here as alternative solutions. Most notably, we have tested the t-SNE algorithm [69] and demonstrated some results of it with different parameter settings in Figure 5.12. Both algorithms lay out filters of higher layers in outer locations, indicating higher layer filters usually capture more diverse features.

Generalization. *DQNViz* can be applied to several other Atari games that involve simple movements, e.g., Pong and Space-Invader. However, as a preliminary prototype, it is not readily applicable to all Atari games, especially the ones with sophisticated scenes and large action spaces, like Montezuma's Revenge. This limitation motivates us to generalize *DQNViz* from several directions in the future. The *first* direction is to adapt the *Trajectory* view to games with a larger action space. Currently, this view can only capture movement patterns in 1D (i.e., moving horizontally or vertically). Enabling the view to capture 2D

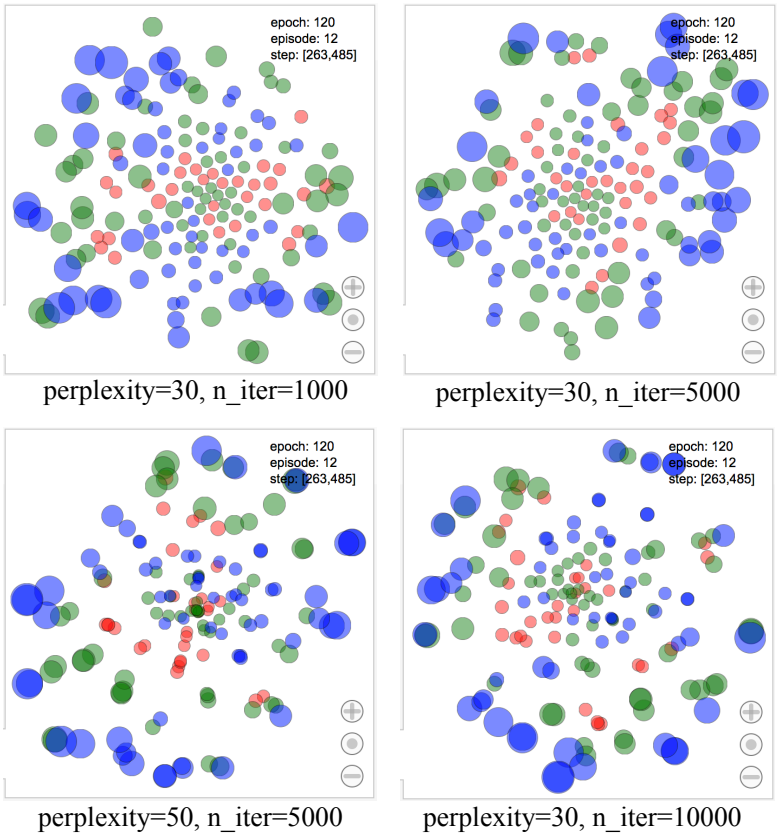

Figure 5.12: Replacing the PCA algorithm in the *Segment* view with the t-SNE algorithm. This figure shows the effects of two important parameters of t-SNE, i.e., perplexity and n_iter (the number of iterations).

movement patterns is our first planned extension. In detail, we can divide the possible actions into three categories: horizontal movement actions, vertical movement actions, and other actions. Multiple *Trajectory* views can be used to visualize actions in different categories, and coordinated interactions can be used to connect those views to retrieve screen states and observe action patterns. *Second*, we believe visual analytics can help the diagnoses and improvements of DQN far more than optimizing the exploration rate. Our next attempt targets on prioritizing the agent's experiences [92] through visualization to accelerate the training. *Lastly*, we also want to explore if any components of *DQNViz* or the designs in our

work can be reused to analyze other RL models. For example, the four-level exploration framework may be directly applicable to other RL models, though the details in each level will have many differences. The idea of looking from the agent's eyes through guided back-propagations could also be reused in other deep RL models.

Scalability. One potential challenge with *DQNViz* is its scalability, including large parameter settings, games with a very long training process, and so on. As to address the large parameter settings, some components of *DQNViz* should be improved to accommodate them. For example, the *Trajectory* view currently presents all actions in one epoch containing 25,000 steps. However, if the number of steps in an epoch is very large, the *Trajectory* view will have to aggregate and smooth those steps, and use semantic zoom to help users explore the sequence patterns. Similarly, the scalability problem may also occur when extending *DQNViz* to other games with a very long training process. For example, in certain games, the length of individual game episodes may be too long to be presented in the *Trajectory* view. In those cases, we can first cut a long episode into many very short segments and use the most frequently appeared action in each segment to represent the segment (i.e., binning and voting). In short, intelligent data aggregations, effective uses of the visualization space, and friendly user-interactions always deserve more considerations in addressing the scalability problem.

More Future Works. We are also interested in exploiting the power of regular expressions in pattern mining, and more user-defined regular expressions may be used to explore the agent's experiences. It may also be possible to extract certain patterns using automatic data mining algorithms. Additionally, our system analyzes DQN models off-line currently (i.e., after training). Enabling domain experts to directly interact with the model training process (i.e., guiding the agent to learn specific behaviors during training) is an important and

interesting direction for us to investigate in the future. Lastly, the current version of *DQNViz* targets to serve domain experts with certain knowledge on DQN models. Simplifying the complex interface and generalizing the domain-specific components of *DQNViz* to extend the tool to common users are also potential research directions that worth to be explored.

5.9 Conclusion

In this work, we present *DQNViz*, a visual analytics system that helps to understand, diagnose, and potentially improve DQN models. The system reveals the large experience space of a DQN agent with four levels of details: overall training level, epoch-level, episode-level, and segment-level. From our thorough studies on the agent's experiences, we have identified typical action/movement/reward patterns of the agent, and those patterns have helped in controlling the random actions of the DQN. The insightful findings we demonstrated, the improvements we were able to achieve, and the positive feedback from deep learning experts validate the effectiveness and usefulness of *DQNViz*.

Chapter 6: Future Research Directions and Potential Challenges

To this part of the dissertation, we have demonstrated how visual analytics can be used to interpret, diagnose, and potentially improve DNN models through the study on three types of popular DNN models. This chapter describes our future plans on extending our study in the following three directions: (1) generalizing the proposed visual analytics solutions to more data types and more DNN models; (2) investigating the powerful capability of visual analytics in analyzing DNNs with a recurrent network architecture; (3) examining the scalability of visual analytics for larger models in industry-scale.

6.1 Towards the Generalization of Visual Analytics Solutions

We first want to extend our current visual analytics works from image data to other types of data, such as texts, logs, audios, or credit card transactions. Compared to images, there is no straightforward choice to directly visualize those types of data. Therefore, the first research question is to effectively encode them into different visual channels and present them onto the graphical display. For example, to visually analyze a DNN trained on audio data, we first have to think about the way of visualizing a snippet of audio (i.e., an input instance) and the intermediate representations (i.e., activations) of the instance when it flows through the DNN. One potential solution here is to cut each audio instance into atomic elements and map them to pixels with different colors. However, whether this

encoding can effectively reflect the subtle difference between individual audio instances needs rigorous studies. Another challenge in visualizing those new types of data is their high dimensionality. Traditional dimensionality reduction algorithms or visualization techniques may fall short of effectively preserving or revealing the important information inside them. New visual designs, or the combination of multiple conventional designs (or visualization algorithms/techniques), are waiting to be proposed to cope with those new data types in different application scenarios.

We also want to generalize our visual analytics works to more types of machine learning models (both DNN and non-DNN models). For example, although we proposed *DeepVID* as a model-agnostic interpretation and diagnosis framework, our demonstrated cases currently focused on DNN models only. Other non-DNN models, e.g., SVM and decision trees, could be used to verify the power of *DeepVID* and find potential issues of the system for further improvements. For the study of *GANViz*, we particularly focused on analyzing DCGAN models. To be more general, we are considering to study more types of GANs (e.g., InfoGAN [24], WGAN [39]) to extract their common model architecture (e.g., a generator and a discriminator) and investigate their shared model training problems (e.g., mode collapse, unstable training). *GANViz* can then be generalized by emphasizing those architectural commonalities and diagnosing those shared model training problems. For *DQNViz*, we currently focused more on simple Atari 2600 games, i.e., the Breakout. The first step in generalizing this system is to cover more complicated games, such as games with a higher dimensional action space. Next, we plan to investigate RL models, in which the environment is more complicated than Atari games and the input/output space is much larger. All these extensions will require a lot of interdisciplinary collaborations, as well as the inputs from machine learning experts.

6.2 Towards Analyzing DNNs with Recurrent Network Architectures

Feedforward Neural Networks (FFNNs) and Recurrent Neural Networks (RNNs) are the two most common DNN architectures. The demonstrated visual analytics works in this dissertation all focus on FFNNs due to the current focuses of our domain collaborators and our limited application contexts. However, we do have plans to cover DNNs with recurrent network architectures, most notably LSTMs, in the near future. We believe some of our proposed visual designs or interpretation/diagnosis solutions can be reused. For example, adopting the small-multiple design (e.g., the line chart, stacked area chart in *DQNViz*) to show model statistics at different levels is also applicable to RNNs. The idea of using the knowledge distillation mechanism to train a small model interpreter (i.e., a weak learner) to explain a complicated model should also work for RNNs.

Meanwhile, we expect the following challenges along our way of investigating RNNs. First, as RNNs are mostly for time-varying data, the challenge could come from the way of effectively visualizing the training/test data instances. For example, how to encode text data and present them through different visual channels to facilitate visual analytics? Should the time dimension in the data instances be treated equally with other dimensions? Second, new visual designs are needed to effectively illustrate the recurrent architecture of RNNs. Our existing sequential designs, like *TensorPath*, have to undergo different levels of adaptations to cope with this new type of architecture. Third, the focused model-related problems may be different in DNNs with recurrent architectures. For example, a common problem in RNNs is the gradient vanishing issue, i.e., the gradients of the training parameters become infinitely small and the networks cannot be further updated/improved, which is often due to the recurrent loop in the RNN architecture. Revealing how and when this problem occurs in a RNN will be very helpful for domain experts to diagnose/improve the model. Intensive

interactions and close collaborations with domain experts are required to extract the essence of those model-related problems.

6.3 Towards Scalable Visual Analytics for Industry-Scale DNNs

The proposed visual analytics solutions have great potential to be applied to industry-scale real-world DNN applications. However, we may have to face with three scalability challenges when extending our works. First, industry-level DNN applications usually involve much larger datasets. For instance, Facebook maintains the status information for millions of users every day; VISA processes billions of credit card transactions every month; Google takes care of trillions of Internet searches every year. Visual analytics of DNNs trained on datasets with those scales needs to cope with the extraordinarily large number of data instances. For example, to diagnose a DNN model, domain experts usually examine the behavior of the model when taking certain data instances (e.g., a suspicious error-causing instance) as input. However, it could be extremely difficult to identify such instances from the large-scale training datasets. Effective data summarizations and efficient anomaly detections may help in these cases. The second challenge comes from the increasingly deeper neural network structures (e.g., from the AlexNet [59] proposed in 2012 to the ResNet [44] proposed in 2015, the number of neural layers has increased from 8 to 152). A practical visual analytics solution should no longer provide all layers' details to overwhelm users. Instead, it should offer high-level overviews/summarizations of all layer-related information at the very beginning to guide users to more important layers, and lead them towards more actionable insight. Third, the growing respect of users' privacy would also pose new challenges in analyzing industry-level DNN models. For example, knowing which dimension of a credit card transaction represents the location information (i.e., where the

transaction happened) could help a lot in interpreting a DNN fraud detection model, as the change of locations usually causes the model to behave differently. However, to protect the information related to users' purchase and consumption habits, the semantics of different dimensions is usually not available to the model designer/analysts, which prevents them from using their prior-knowledge to better interpret what happened in the model. Aggregating the subset of instances that behave similarly (i.e., clustering) could augment the data features and may help the interpretation of DNNs in this case.

Chapter 7: Conclusion

In this dissertation, we demonstrated the prospect of interpreting and diagnosing DNN models through interactive data explorations and visual analytics. Specifically, we focused on three representative DNN models to investigate three types of machine learning problems: (1) CNN for supervised learning; (2) GAN for unsupervised learning; (3) DQN for reinforcement learning, and covered three main tasks that visual analytics can help, i.e., model interpretation, model diagnosis, and model improvement. This chapter concludes the dissertation by first summarizing our contributions. It then closes the dissertation with additional comments on the broader implications of visual analytics for deep learning.

7.1 Summary of Contributions

Our contributions can be summarized into the following three categories. First, from the way of exploring DNN models, we reviewed the literature and elicited two general model interpretation and diagnosis approaches, i.e., the Black-Box approach and the White-Box approach. Second, from the system perspective, we proposed three integrated visual analytics prototypes to analyze three types of DNN models, i.e., *DeepVID* for supervised learning models, *GANViz* for unsupervised learning models, and *DQNViz* for reinforcement learning models. Lastly, we introduced two novel visual designs tailored for the large-scale

multi-faceted DNN data, i.e., the *TensorPath* view in *GANViz* and the *Trajectory* view in *DQNViz*. We briefly review them as follows:

Two general model interpretation/diagnosis approaches:

- Black-Box Approach: The Black-Box approach is a model-agnostic interpretation and diagnosis approach. It uses only the model inputs and outputs to probe the behaviors of DNN models, and does not require to know the internal model details (e.g., the number of layers, the types of layers, etc.). *DeepVID* (Chapter 3) is a realization of this approach in investigating different image classification models (CNNs).

- White-Box Approach: The White-Box approach is a model-specific interpretation and diagnosis approach. It opens the "black-boxes" and exposes all internal details of DNN models to enable users to flexibly interpret and diagnose them with visual evidence. This approach requires users to have the full access to every layer/neuron of the analyzed DNN models. *GANViz* (Chapter 4) and *DQNViz* (Chapter 5) are designed following this approach.

Three visual analytics systems:

- *DeepVID*: Targeting on interpreting and diagnosing image classification models, we designed and developed *DeepVID*. This visual analytics system helps users identify data instances of interest (e.g., false-positive predictions) and employs knowledge distillation to train a weak learner to interpret the original DNN classifiers' behavior on the interested data instances. Through interpretations with visual evidence, *DeepVID* helps users diagnose classification models by dissecting why and how the models succeeded or failed in different scenarios.

- *GANViz*: In order to understand the adversarial game between the generator and the discriminator of GAN models, we studied the DCGAN model and designed *GANViz*. This visual analytics prototype is developed to satisfy three specific requirements from domain experts, i.e., model evaluation, model interpretation, and comparative analysis. For evaluation, we introduced several new metrics to better assess the training quality of GAN models. For interpretation, five coordinated visualization views were implemented to empower users to explore and interpret the DCGAN model with different levels of details. Lastly, we proposed a design similar to PCP, which allows users to conduct comparative analysis across-time and across-class.

- *DQNViz*: To better understand the mysterious power of DQN models, we developed the visual analytics system *DQNViz*, which reveals the models' details in four levels: overall training level, epoch level, episode level, and segment level. The system allowed users to effectively identify typical movement or reward patterns of the trained agent (e.g., the *digging* and *hesitating* pattern in the Breakout game). By examining the ineffective movement patterns, *DQNViz* helped domain experts propose a new strategy in controlling the initiation of random actions. This is an important case demonstrating that visual analytics can not only help to interpret and diagnose DNN models, but also help to improve and refine the models.

Two novel visual designs:

- The *TensorPath* View (Figure 4.8, Figure 4.9): Comparing a pair of data instances and examining how they flowed through the multi-layer architecture of DNNs is a very important task in interpreting and diagnosing DNNs. We propose a compact zigzag design, named *TensorPath*, which lays out neighboring layers of DNNs orthogonally

and fills the orthogonal space with the trained parameters (i.e., network weights, convolutional filters, etc.). Activations (or feature maps) for the pair of data instances in comparison are sorted based on the difference within individual activation pairs and placed along the axes representing respective DNN layers. From top-left to bottom-right, this visual design can show how a pair of interested images flowed through the neural network layer-by-layer, and reveal all necessary details to facilitate the pair-wise comparisons. We demonstrated how *TensorPath* can be used to analyze the discriminative network of GAN models in Chapter 4. It can also be used to investigate any DNNs with the feedforward network architecture.

- The *Trajectory* View (Figure 5.2c): Exploring the large-scale multi-faceted event sequence data generated from reinforcement learning models is not a trivial task. While studying the DQN model, we proposed a new visual design for the visualization and analysis of this type of data, i.e., the *Trajectory* view of *DQNViz*. This design can: (1) visualize different aspects of the multi-faceted data simultaneously and synchronize different temporal events based on time; (2) effectively reveal the data patterns from the large amount of event sequences. Focusing on the Breakout game, this visual design breaks a long action sequence based on game episodes and uses a 2D plot to present each episode. The horizontal and vertical axis of the 2D plot represent the time and paddle displacement respectively. Through pattern mining and segment clustering, users can easily identify typical movement patterns of the reinforcement learning agent trained for this game. Additionally, other types of data (e.g., rewards, number of lives) can be synchronized to the 2D plot according to their temporal information (i.e., the horizontal axis) through user-interactions. Those data can also be disabled from the plot on-demand to reduce potential visual clutters.

Through comprehensive evaluations, as well as case studies conducted together with multiple deep learning experts, we validated the effectiveness of our contributions in interpreting and diagnosing DNNs through visual analytics.

7.2 Closing Remarks

Living in the digital era, we have witnessed multiple breakthroughs from the deep learning field, such as the adoption of CNNs in image classifications, the success of AlphaGo in outperforming human beings. With the growing complexity of DNNs and the demanding need in understanding these "black-boxes", we believe the next breakthrough will come from model interpretations. Visual analytics is currently the most promising solution towards this direction and our current works presented in this dissertation are part of the numerous initial attempts. With these and many more on-going attempts, we embrace the bright future of visual analytics and look forward to more exciting works for the interpretation and diagnosis of DNN models.

Bibliography

[1] CelebA data. http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. Accessed: 2017-09-08.

[2] ConvNetJS MNIST Demo. https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html. Accessed: 2018-05-07.

[3] Flask microframework. http://flask.pocoo.org/docs/1.0/.

[4] Google Quick Draw game. https://quickdraw.withgoogle.com/. Accessed: 2017-08-08.

[5] MNIST data. http://yann.lecun.com/exdb/mnist/. Accessed: 2017-08-08.

[6] TensorBoard. https://www.tensorflow.org/get_started/summaries_and_tensorboard. Accessed: 2017-08-22.

[7] TensorFlow code for DCGAN on GitHub. https://github.com/carpedm20/DCGAN-tensorflow. Accessed: 2017-08-08.

[8] TensorFlow code for DFC-VAE on GitHub. https://github.com/yzwxx/vae-celebA. Accessed: 2018-08-20.

[9] TensorFlow code for DQN on GitHub. https://github.com/tambetm/simple_dqn. Accessed: 2018-02-08.

[10] TensorFlow code for VAE on GitHub. https://github.com/fastforwardlabs/vae-tf/tree/master. Accessed: 2018-07-21.

[11] Alfred V. Aho and Jeffrey D. Ullman. *Foundations of Computer Science (Chapter 10: Patterns, Automata, and Regular Expressions)*. Computer Science Press, Inc., New York, NY, USA, 1992.

[12] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pages 176–185, 2017.

[13] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[14] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (GANs). In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 224–232, 2017.

[15] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[16] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[17] Michaël Aupetit. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing*, 70(7-9):1304–1330, 2007.

[18] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 2015.

[19] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[20] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Inc., New York, NY, USA, 2003.

[21] Alsallakh Bilal, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren. Do convolutional neural networks learn class hierarchy? *IEEE Transactions on Visualization and Computer Graphics*, 24(1):152–162, 2018.

[22] Jeff A Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *International Computer Science Institute*, 4(510):126, 1998.

[23] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847. IEEE, 2018.

[24] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing

generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[25] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4302–4310, 2017.

[26] Kristin A Cook and James J Thomas. Illuminating the path: The research and development agenda for visual analytics. Technical report, Pacific Northwest National Laboratory (PNNL), Richland, WA (US), 2005.

[27] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[29] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[30] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[31] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018.

[32] James Fogarty, Ryan S Baker, and Scott E Hudson. Case studies in the use of ROC curve analysis for sensor-based estimates in human computer interaction. In *Proceedings of Graphics Interface*, pages 129–136, 2005.

[33] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017.

[34] Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. *NIPS Deep Reinforcement Learning Workshop, arXiv preprint arXiv:1512.02011*, 2015.

[35] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[36] Ben Goertzel and Cassio Pennachin. *Artificial General Intelligence*. Springer-Verlag Berlin Heidelberg, 2007.

[37] Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, December 2016. arXiv: 1701.00160.

[38] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[39] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[40] Shunan Guo, Ke Xu, Rongwen Zhao, David Gotz, Hongyuan Zha, and Nan Cao. EventThread: Visual summarization and stage analysis of event sequence data. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):56–65, 2018.

[41] Hado V Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[42] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction (second edition)*. Springer, 2009.

[43] Frank S He, Yang Liu, Alexander G Schwing, and Jian Peng. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *arXiv preprint arXiv:1611.01606*, 2016.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[45] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[46] Nicolas Heulot, Jean-Daniel Fekete, and Michael Aupetit. Proxilens: Interactive exploration of high-dimensional data using projections. In *VAMP: EuroVis Workshop on Visual Analytics using Multidimensional Projections*. The Eurographics Association, 2013.

[47] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[48] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[49] Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141. IEEE, 2017.

[50] Waqas Javed, Bryan McDonnel, and Niklas Elmqvist. Graphical perception of multiple time series. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):927–934, 2010.

[51] Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Polo Chau. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018.

[52] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[53] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[54] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[55] Jens Kober and Jan R. Peters. Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems*, pages 849–856. 2009.

[56] Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, April 2003.

[57] Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. Selph: Progressive learning and support of manual photo color enhancement. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2520–2532, New York, NY, USA, 2016. ACM.

[58] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[60] Milos Krstajic, Enrico Bertini, and Daniel Keim. CloudLines: Compact display of event episodes in multiple time-series. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2432–2439, 2011.

[61] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[62] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[63] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4681–4690, 2017.

[64] Mengchen Liu, Shixia Liu, Hang Su, Kelei Cao, and Jun Zhu. Analyzing the noise robustness of deep neural networks. *arXiv preprint arXiv:1810.03913*, 2018.

[65] Mengchen Liu, Jiaxin Shi, Kelei Cao, Jun Zhu, and Shixia Liu. Analyzing the training processes of deep generative models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):77–87, 2018.

[66] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017.

[67] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015.

[68] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

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

[70] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[71] Yao Ming, Huamin Qu, and Enrico Bertini. RuleMatrix: Visualizing and understanding classifiers with rules. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):342–352, 2019.

[72] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[73] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. In *NIPS Deep Learning Workshop, arXiv preprint arXiv:1312.5602*. 2013.

[74] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[75] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[76] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[77] Augustus Odena. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

[78] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651, 2017.

[79] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[80] Charles Perin, Frédéric Vernier, and Jean-Daniel Fekete. Interactive horizon graphs: Improving the compact visualization of multiple time series. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3217–3226. ACM, 2013.

[81] Nicola Pezzotti, Thomas Höllt, Jan Van Gemert, Boudewijn PF Lelieveldt, Elmar Eisemann, and Anna Vilanova. DeepEyes: Progressive visual analytics for designing deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):98–108, 2018.

[82] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, 2016.

[83] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[84] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, 2017.

[85] Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, 2017.

[86] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you? explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[87] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[88] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, 1991.

[89] Yuki Saito, Shinnosuke Takamichi, and Hiroshi Saruwatari. Statistical parametric speech synthesis incorporating generative adversarial networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(1):84–96, 2018.

[90] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[91] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[92] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[93] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[94] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343. IEEE, 1996.

[95] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[96] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[97] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent Q-network. *arXiv preprint arXiv:1512.01693*, 2015.

[98] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[99] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676, Jan 2018.

[100] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[101] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[102] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[103] Edward R. Tufte. *Beautiful Evidence*. Graphics Press, 2006.

[104] John W Tukey. *Exploratory Data Analysis*. Pearson, 1977.

[105] Fan-Yin Tzeng and Kwan-Liu Ma. Opening the black box - data driven visualization of neural networks. In *VIS 05. IEEE Visualization, 2005*, pages 383–390. IEEE, 2005.

[106] Junpeng Wang, Liang Gou, Hao Yang, and Han-Wei Shen. GANViz: A visual analytics approach to understand the adversarial game. *IEEE Transactions on Visualization and Computer Graphics*, 24(6):1905–1917, 2018.

[107] Junpeng Wang, Xiaotong Liu, Han-Wei Shen, and Guang Lin. Multi-resolution climate ensemble parameter analysis with nested parallel coordinates plots. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):81–90, 2017.

[108] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[109] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1995–2003, 2016.

[110] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[111] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[112] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[113] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[114] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):364–373, Jan 2019.

[115] Jian Zhao, Michael Glueck, Fanny Chevalier, Yanhong Wu, and Azam Khan. Egocentric analysis of dynamic networks with egolines. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5003–5014. ACM, 2016.

[116] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, 2016.