Interpreting and visualizing the decisions of Deep Neural Networks

Dissertation

For the award of the degree

"Doctor rerum naturalium" (Dr.rer.nat.)

Of the Georg-August-Universität Göttingen

Within the doctoral program Computer Science (PCS)

Of the Georg-August University School of Science (GAUSS)

Submitted by

Aisha Aamir

From Rawalpindi, Pakistan

Göttingen, 2022

Abstract

Machine learning and its associated algorithms involving deep neural networks have gained widespread admiration in the computer vision domain. In this regard, significant progress has been made in automating certain application-dependent tasks, especially in the fields of medicine, autonomous driving and robotics. Moreover, considerable improvements have already been made and work is underway to make automated systems secure and robust against failures. Nonetheless, researchers are struggling to find ways to give reasons and explanations for why a machine learning model made a certain decision. In particular, deep neural networks are considered to be “black-box” in this regard, and the reason is that their distributed encoding of information makes it even more challenging to interpret their decision-making capability.

In view of the above challenges, this dissertation aims to establish methods to visualize and interpret the decisions of these complex machine learning models in an image classification task. We opt for three types of post hoc methods, i.e., global, hybrid and local interpretability to understand and assess the reasons and type of image features that are vital for a decision in an image classification task. Hence, we call our approach “visualizing and interpreting the decision of deep neural networks".

On a global level, we investigate and assess the deep network architecture as a whole, keeping in view the internal connections between adjacent layers, filters and functioning of different hidden layers. Hence, we have proposed a visualization method in the form of a Caffe2Unity plugin to construct and visualize a complete AlexNet architecture in a virtual reality environment. This novel approach allows the user to become part of the virtual network and gives liberty to explore and visualize the internal states of the network. Exploring and visualizing the network in a virtual environment for global assessment, working and understanding of deep neural networks benefits both novices and experts among our target audience.

Using a hybrid approach, we gave a local interpretable module within our global virtual model that allowed the user to visualize and interpret the network in real-time. We permitted the user to add an occlusion block on an image and visualize the results, as well as verify the decision of the network via our reframed integrated Shapley values approach. In this way, we achieved our goal of finding a good reason to determine which part of the image the network considers important for making its decision.

At the local interpretable level, we proposed a layer-wise approach using influence scores to gain deeper insights into the pre-trained models' decision-making capability. We used the layer-wise influence score to determine what each layer has learned and which training data is most influential in the decision. By finding a contrast between the influential image and the network's decision, we also identified the biased nature of the network towards the texture of the images.

The proposed methods analyze different kinds of explainable and interpretable perspectives to study and unlock the "black-box" nature of deep neural networks in image classification tasks. We can augment the visualizing and interpretability approaches in many other applications, particularly in understanding action predictions in robotics and object scene understanding.

Acknowledgements

My sincerest gratitude and all praise are for Almighty Allah who is the entire source of wisdom and special honor for the Holy Prophet Muhammad (P.B.U.H) whose entire life is a source of guidance for humankind.
On the successful completion of this dissertation, first of all I would like to express my heartiest gratitude to my supervisor Prof. Dr. Florentin Wörgötter for his continuous guidance, and encouragement in the supervision of this work. Thank you for providing me this great research environment and giving me the liberty to explore and develop the original idea for this thesis and for your fatherly care besides my thesis. My co-supervisor Prof. Dr. Minija Tamosiunaite for her fruitful discussions, and spending precious hours for repeated revisions of my work, without which all this would not have been accomplished. A special thanks to my committee member Prof. Dr. phil.-nat. Jens Grabowski for his keen review to steer the focus of this thesis.

I would also like to thank all my former and current group members Michael, Tatyana, Jannik, Timo, David, Christian, Tomas, Moritz, Cheng, Osman, Sebastian, Fatemeh, Florian, Erenus, Lennart, Shirin, Sebastian, Shijia, for their direct and indirect support. A very special thanks to Ursula and Thomas who were a great help too.

I would also like to thank my Parents: Muhammad Azeem Zaki and Munwar Sultana for their everlasting love, prayers and boosting my moral whenever I felt hopeless. Without their support & prayers I would not have achieved this goal. A big thanks to my sisters: Ammara & Annam and my brother Ahmad for cheering me up and to all the silly and joyful moments we spent together.

Lastly, my loving husband Aamir Manzoor for his everlasting love, support and reason for continuous motivation & encouragement during the challenging times of my PhD work. Thank you for listening to all my problems and giving me the shoulder to cry on. Not forgetting my daughter Mahnoor, the reason of my life and the unforgettable times we spend together. I really don't have the words to describe the love and affection I have for both of you and thank you for making me complete.

Aisha Aamir

Dedicated to my parents,

My loving husband &

My sweet little daughter

Table of Contents

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
AV	Augmented Virtuality
BVLC	Berkeley vision and learning center
CAFFE	Convolutional Architecture for Fast Feature Embedding
CNNs	Convolutional Neural Networks
CPU	Central Processing Unit
CDBNs	Convolutional Deep Belief Networks
DCNNs	Deep Convolutional Neural Networks
DBN	Deep Belief Network
DNNs	Deep Neural Networks
DLL	Dynamic Link Library
GUI	Graphical User Interface
GPUs	Graphics Processing Units
HMDs	Head Mounted Displays
HVP	Hessian Vector Products
LRP	Layer-wise Relevance Propagation
MCDNN	Multicolumn Deep Neural Networks
ML	Machine Learning
RBMs	Restricted Boltzmann Machines
UE	Unreal Engine
VR	Virtual Reality

CHAPTER 1

Introduction

1.1 Motivation and Problem Definition

Today's modern world mainly revolves around Artificial Intelligence (AI), machines and their ability to perform tasks efficiently within a limited amount of time. This popularity is due to Machine Learning (ML), which is playing a pivotal role in advancing technology in industry and the scientific sector. We not only see this at work in large industrial sectors, but it has become a part of our everyday lives. Consider smartphones with their ability to scan irises, fingerprints and even detect and track faces [87], strategic video games [88] and [89], various online tools and services that detect and translate languages [90], human-computer interaction, visual object recognition, physics and many computer vision tasks. The reason for the proliferation of AI-based machine learning is the availability of high computational processing units, access to vast volumes of datasets and enhanced algorithmic approaches that are able to achieve excellent performance in solving a large number of complex everyday tasks, in most cases reaching near human-level performance.

In this regard, Deep Neural Networks (DNNs) have shown outstanding performance and are considered to be at the forefront, but due to their distributed non-linear encoding of information, they are perceived to be "black-boxes". Their non-linear nature makes it very difficult to determine how these complex structures arrived at a certain decision. Moreover, their internal operations, behavior and complexity in achieving the best performance are deeply unsatisfactory from a scientific standpoint because the development of improved models is limited to trial-and-errors [91], [92], [93] and [94]. Such a hindrance can be encountered in various resource-restricted devices [95] or in security risk-related applications, where a lack of robustness to adversarial attacks can greatly hinder performance [96]. Furthermore, in certain applications, a lack of interpretability and transparency reduces trust in the decisions made by these deep networks. Solving the above challenges is a hot topic in this domain, and there is a lot of research underway to find the best possible solution to gain better insights into these high-end networks.

In this thesis, we will focus on the interpretability of the decisions made by these large non-linear structures in the computer vision domain. Our main emphasis will be on the classification task, where we first try to gain a general understanding of network architecture in a virtual reality environment. At the same time, we further extend the interpretability approach to shed light on the assessment of deep neural network architecture by finding reasons for why the network correctly or incorrectly classifies an image. The goals of this work are summarized below:

Global interpretability

- To interpret the model and gain a general understanding of the deep neural network. To achieve this goal, we designed and visualized a pre-trained AlexNet [97] architecture on the ImageNet [98] dataset in a virtual reality environment of the gaming engine Unity.
- To evaluate the internal details and visualize the layers of the model, we configured the Caffe framework in a dynamic link library to be used in the Unity engine as a plugin.

Hybrid Interpretability (local + global)

- To countercheck the classification decisions, we employed and reframed Shapley values to interpret the results in real-time using a virtual setup. We also added occlusion on the images to see how the network behaves and which part of the test images are relevant and which areas negatively contribute to the decision-making capability of the network.

Local Interpretability

- To further evaluate the model's decisions, we used influence scores in order to find out which training data is responsible for making the decisions.
- To evaluate which image features are important at different layers of the network and for which features the network is biased.

1.2 Organization of the thesis

To achieve our objectives and define the basis, we have organized the structure of this thesis as follows:

Chapter 2: We will describe the foundations and fundamentals required to understand the topics that are used later in the subsequent chapters. Next, we will describe machine learning in a more general way and then narrow it down to its learning types, i.e., supervised, unsupervised and reinforcement learning, while focusing on the neural network models in particular. Our main focus will be to describe these models in the computer vision domain and discuss what kinds of features are learned by the neural network model. We will briefly discuss the frameworks used for deep learning models (e.g., TensorFlow, Caffe, Keras) to give the user a bird's eye view of the relevant features of these frameworks. Then, we will focus on Convolutional Neural Networks (CNNs) as these networks play a pivotal role in most computer vision tasks. In this section, we will discuss how the input is processed through the different layers and how the input changes as it passes through different layers of the network. We will also describe the role of pooling layers, fully connected and softmax layers, stride and filters, etc. In general, this chapter will lay the foundation of our visualization strategy that we will describe in more detail in the following chapters. Lastly, we will introduce the use of computer technology to simulate the environment in virtual reality. We will describe the tools and simulating environments that are available to execute the task in virtual reality, while placing particular focus on the Unity gaming engine.

Chapter 3: We will discuss in detail the related work that will be used in this thesis. In particular, we will primarily focus on how visualization has been used to understand DNNs, the methods that already exist and the goal of visualizing DNNs. We will also give a taxonomy of visualization, distinguishing between different methods and goals to assess these models in the computer vision domain.

Chapter 4: In this section, we will describe the making of a dynamic link library for the Caffe framework – how it was compiled and what software-based prerequisites were needed for a successful compilation process. We will give details of the classes and main functions used in the execution of the current task. In this section, we will also cover how to link the Caffe framework to the Unity gaming environment alongside a demo program to demonstrate its successful compilation.

Chapter 5: This chapter will detail the use of the DLL and the creation of a real-time interactive visualization model for deep neural networks in the Unity engine. In addition, we will describe how a user can interact with this immersive model and gain an in-depth understanding and interpret the model's decisions using Shapley values, hence giving the details of the manuscript.

Chapter 6: We will discuss the semantics and deeper insights learned and interpreted by DNNs. It is also a post hoc interpretability approach where we want to understand and interpret why the network fails to classify an input image. The main focus of this study will be to interpret learned images and evaluate the reason for network decisions by using layer-wise influence scores.

Chapter 7: Lastly, we will draw conclusions and outline future work.

CHAPTER 2

Foundations

This thesis is a combination of various models and techniques, particularly focusing on DNNs. The methods adopted for conducting the research are not only from the computer vision domain, but we have also implemented methods from the fields of game theory and robust statistics. In order to provide an in-depth understanding of the forthcoming chapters, we briefly describe the foundations of the context relevant to the content of this thesis. This chapter, while laying the foundations and describing some general information, will be particularly useful for readers who are novices in this field.

2.1 Machine learning

Learning is a fundamental capability of any machine learning model. Learning instructions are usually algorithms to determine the appropriate weights and/or other learning parameters. A machine can become intelligent if it learns from its changing environment and adapts to those changes in similar situations. In other words, we say that a machine can learn to solve problems by identifying patterns in the data through computational modelling. These models can be automated via classical programming languages in the form of an algorithm. Any machine learning model observes the patterns in its training data and predicts an output based on the given input pattern [99]. This input can be in the form of text, videos, signals and images. In this thesis, however, we deal with images as inputs for the machine learning model. Although there are various techniques to train these models, we here briefly describe supervised learning, unsupervised learning and reinforcement learning to give a general understanding. Below, we give a brief overview of these learning techniques.

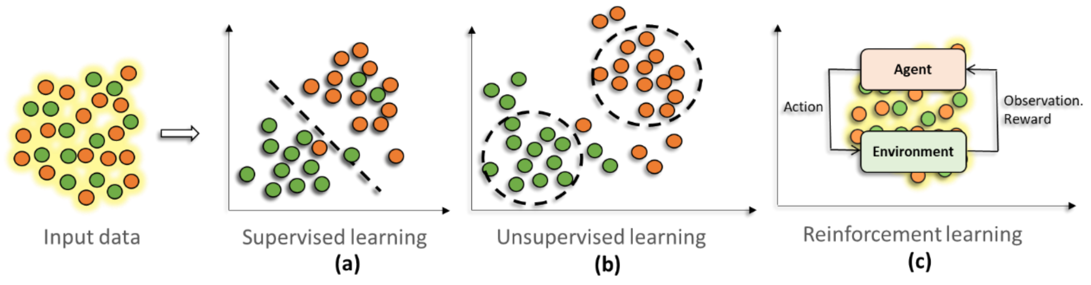


Figure 2.1: Three most common types of machine learning techniques showing **(a)** classification of data into two groups with color shown in red and green **(b)** clustering the data based on similarity of patterns and **(c)** reinforcement learning in which an agent learns from its environment and performs certain action and based on correct action it gets a reward.

Supervised learning

Supervised learning is one of the most common methods used for machine learning today, which involves learning from examples. Training through supervision involves finding patterns and correlating the inputs with the corresponding correct outputs. The objective of this type of learning is to identify the correct labels based on unseen input data. In its very simple form, the supervised learning algorithm can be expressed in the form of $y_a = f(x_a)$, where $a \in \{0, N\}$ for N sample input images. After learning, the model can predict and map each input x_a to its corresponding output label y_a. Generally, supervised learning is divided into classification and regression problems depending upon the output space. If the algorithm predicts discrete values and assigns the input image to a particular group or class, we say it is a classification problem, as shown in Fig. 2.1(a). However, if the output prediction is continuous (e.g., income, temperature or sales, etc.), we say it's a regression problem.

Unsupervised learning

It is a type of self-organized learning where the model tries to group the data and establishes a comparison relationship to guess the predicted output. There is no labelled data with unsupervised learning, and the model finds patterns in this unlabelled dataset and groups them according to known patterns. Clustering is a type of unsupervised learning and usually is employed on items/objects (e.g., exam scores). The features, for instance, can be grades, averages, pass/fail, etc., which are grouped according to similar patterns, as shown in Fig. 2.1(b). Generally, with unsupervised learning, the model is given x_a input vectors, but unlike with supervised learning, there are no labelled output vectors.

Reinforcement learning

This type of learning basically involves learning and interacting with environments. There is no labelled or unlabelled data, but with this type of learning, the model/agent learns from the environmental state or whether a certain action was right or wrong. The selection of a particular action is called a policy, and if the agent is able to perform a correct action based on its changing environment, it is rewarded in the form of a signal. This reward may depend on the previous action or on the preceding actions that have already taken place, as shown in Fig. 2.1(c). The ultimate goal of reinforcement learning is to maximize the duration of the reward based on its optimal policy using trial and error.

For DNNs, learning can be regarded as a non-linear optimization problem that tries to determine certain network parameters, minimizing the cost function for the given task [100]. DNNs were first introduced in the early 1990s and showed good performance in handwritten digit classification and face detection related tasks [101]. Until now, several research investigations have concluded that DNN models deliver robust performance on challenges related to visual classification tasks and have become the architecture of choice for large-scale image recognition [7] and [35]. Their work delivered state-of-the-art performance on CIFAR-10 datasets and ImageNet 2012 classification benchmarks, and the model achieved an error rate of 16.4%. Authors in [102] used DNNs to visualize unsupervised features of hidden layers in Deep Belief Networks (DBNs). Experiments performed in [16] employed deep network models and developed innovative methods that led to the maximizing of neural activity of interest through optimization using gradient ascent in the image space. The main reason for these improvements and the high performance of DNNs was the availability of large training sets with millions of labelled examples, as well as the application and availability of powerful and fast Graphics Processing Units (GPUs) [102].

To understand these complex machine learning models and device changes and improvements, we need a framework to store and process our data. This framework serves as a tool or interface, allowing the user to utilize pre-compiled and readily available components without the hassle of building algorithms from scratch. Although there are quite a few frameworks available, we will give a brief overview of the three most popular ones used in the computer vision community, namely Keras, Tensorflow and Caffe, which are most relevant to the context of this thesis.

Tensorflow

Keras is an open-source library that provides an end-to-end interface for experts and trainees in the field of machine learning. It was initially developed by a team of researchers at Google Brain and supports many pre-written codes as one of the most popular deep neural network models. This framework also has very strong community support, documentation and very flexible architecture support for CPUs and GPUs. In addition, it supports codes written in popular languages such as C++, Python and R, and is successfully deployed in audio, video, text and image related tasks. Although there are many components involved in this framework, the two most relevant for the context of this work are TensorFlow [103] and TensorBoard.

Keras

Although TensorFlow is very popular among the experts, Keras provides a more advanced high-level API for machine learning novices using the popular Python programming language. This framework is flexible in that it can provide support to run on Theano, TensorFlow and CNTK, making it ideal for research purposes. Like TensorFlow, it also supports most of the popular deep neural network architectures with CPU and GPU support. Keras provides two options for implementing deep neural networks: sequentially or via the Keras functional API, depending on the type of model being implemented.

Caffe

Caffe [104] is another popular open-source framework for deep neural networks, developed and maintained by researchers at the University of California, Berkeley. The main reason for its popularity is that it is extremely fast compared to the other frameworks, especially for computer vision related tasks. Also, it offers the functionality of software wrappers for multiple programming languages such as Python, C, C++ and MATLAB. In addition, it has many popular pre-trained deep learning architectures and weights that can be used to solve machine learning tasks (e.g., audio, video, text, images, etc.) on CPUs and GPUs.

2.2 Convolutional Neural Networks

With huge advances in computing power and GPUs, machine learning models such as DNNs have gained immense popularity. These models have been shown to outperform in a wide variety of fields such as speech recognition, human-computer interaction, visual object recognition, physics and many kinds of computer vision tasks. However, these models are still considered to be a "black-box" due to their distributed encoding of information, internal operations, behavior and complexity in achieving the best performance. For these reasons, these models are inadequate from a research perspective because the development of improved models is limited to trial-and-error [91], [92], [93] and [94]. It is difficult to understand how these complex networks arrive at a particular classification decision because of the interaction of a large number of non-linear parts and the size of the networks. Understanding what image features are learned through CNNs is interesting in its own right, but it is also important for further improvement of the models. For example, visualization using a deconvolution technique led to state-of-the-art work being done on the ImageNet benchmark in 2013 [82]. This method used image features and observed different convolutional filters to extract important features at different hidden layers of the DNN model. Since these complex models are a combination of different components,

layers and filters, we will briefly describe the function of some of these components used in any typical convolutional neural network.
Although providing more elaborate information on these models is beyond the scope of this thesis, readers new to this field can find a more detailed overview of machine learning in the computer vision domain in the textbooks authored by Goodfellow, Bengio and Courville [105], and Murphy [106].

2.2.1 Components of DNNs

Neural networks are basically inspired by the functioning of the biological neural system of the human brain, in which neurons are connected by synapses and form a dense network that propagates neural activity. Each neuron produces an output activation depending upon the incoming connection. In general, when the incoming charge exceeds a certain threshold, an action potential is triggered and propagates along the axon to release neurotransmitters at the synapses. A detailed working and chemical exchange of ions after the action potential of these biological neurons can be found in the book authored by Dayan and Abbott [107]. Similarly, we simplify the activation propagation of artificial neurons via synaptic weights. These weights are the learnable parameters for the neural network that learns and acquires knowledge during its training.

This subsection aims to address the basic components involved in the understanding and functioning of an artificial neural network. In particular, we focus on a deep neural architecture in this work as a non-linear hierarchical parameterized structure that associates its parameters θ with input x and output y given as $y = f(x, \theta)$. In a generalized way, we can say that a neural network can be represented as a function f_i of inputs $x = (x_1, x_2, \dots, x_n)$ with the weight vectors $w_i = (w_{i,1}, w_{i,2}, \dots, w_{i,n})$, bias b_i and activation function $\emptyset$ given in a mathematical form as:

$$y_i = f_i\,(x) = \emptyset(\langle w_i, x\rangle + b_i).$$

Usually, the choice of activation function $\emptyset$ depends on the type of function performed by the neural network. The most commonly used activation functions for classification-related tasks are rectified linear units ReLU and logistic or sigmoid functions σ, given as:

$$\sigma = \frac{\exp(x)}{\exp(x) + 1} \text{ and } ReLu(x) = \max(0, x).$$

Theoretically, an architecture is defined by neurons arranged in layers, in which neurons interact with other neurons from adjacent layers. As the number of layers increases, the network becomes large and complex, with many neurons and connections. Hence, the name "deep" is used for neural networks with multiple stacks of layers. Although there are different types of deep neural architectures, we will briefly discuss the layers and components commonly used in CNNs. In a typical CNN for an image processing task, activations are represented in the form of 3-dimensional tensors used for height, width and channel. These channels are RGB colors for an input image that store useful features for deeper layers. However, it is necessary to understand the difference between the channels and the spatial components involved.

Convolutional Layer: Each convolutional layer accepts a specific spatial layout of its input that is convolved (or multiplied on pixel grids) with spatial filters. These kernels are usually of a fixed size (typically 3x3, 5x5 or 7x7) depending upon the depth (or location) of that specific layer in the network. Each filter/kernel produces one output channel. These filter coefficients are the parameters for the CNNs. As the filter hovers over the pixel grids of the input, it produces feature maps that later serve as inputs for the successor layers, as shown in Fig. 2.2.

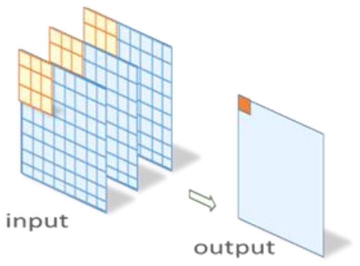


Figure 2.2: A typical 2D convolution operation with a kernel of size 3x3 shown in orange applied on three input channels to produce one output element.

In a convolutional layer with n number of filters/Kernel K applied on 2D image I of size $i \times j$ can be expressed as:

$$(K * I)(i,j) = \sum_{m,n} K(m,n)I(i+n, j+m).$$

Receptive fields: The feature maps are formed as a result of receptive fields, which are the parts of the images where a particular neuron is most sensitive. As the size of the receptive field grows, the computational cost increases and it is usually highly complex with very deep networks. However, there are a number of methods to sufficiently reduce the size of these receptive fields. A common approach to sub-sample the activation maps is by pooling layers.

Pooling: A common way to retain the spatial features of the image without losing important detailing is to have a low-resolution version of the input across all the channels. This can be efficiently achieved by pooling layers that are placed on alternate positions after the convolutional layers. Depending on the size of the kernel and image, mostly two common types of pooling are performed: max pooling or average pooling of the spatial features to sub-sample the feature maps, as shown in Fig. 2.3. Usually, a filter with a small stride is considered that determines the pixel shift over an input grid.

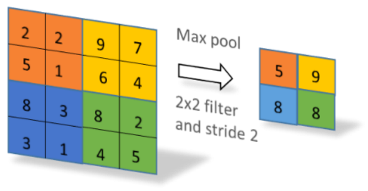


Figure 2.3: Max pooling operation showing a sub-sampling of feature maps with 2x2 kernel size with stride 2.

Fully Connected Layer: In these layers, the neurons are connected one-to-one with the neurons of the next layers. It is more or less like a multilayer perceptron with usually 2-3 layers that are actually responsible for mapping the activations from the previous layers into class probability distributions for the output. For a network, if we consider that l-1 is the fully connected layer, $m^{(l-1)}$ are the feature maps for the layers, $w_{i,j}^{(l)}$ are the weight parameters, we can mathematically define it as:

$$y_i^{(l)} = f\left(x_i^{(l)}\right) \text{where } x_i^{(l)} = \sum_{j=1}^{m^{(l-1)}} w_{i,j}^{(l)} y_i^{(l-1)}$$

Softmax layer: It is usually the layer before the output layer of the network containing an equal number of nodes as in an output layer. The softmax layer is responsible for assigning probabilities to classes in a multi-class classification problem and is given as:

$$f_j(x) = \frac{e^{x_j}}{\sum_k e^{x_i}}$$

A generic structure showing the flow of an input image through various layers of the network during a convolutional and de-convolutional operation is shown in Fig. 2.4. Understanding which computations are carried out at each layer of CNNs is an important research direction in image recognition tasks. In particular, visualizing these layers to find ways of improving and interpreting these models is also becoming increasingly popular these days. An example is the interactive plotting of activations on each layer of trained networks in the case of static (2D images) and dynamic (video) inputs. Static images only give slow and detailed investigations of a particular input, whereas video can be processed live from a user's camera and is especially helpful if someone wants to move different items in the field of view [79]. Another network analysis method is to study each layer as a group and analyze computations involving a set of neurons on a specific layer [46].

Past studies examining the interpretation and computing of the functions performed by individual neurons in the layers have been divided into two categories: data-centric and model-centric. The data-centric approach keeps the model constant, makes augmentations in the dataset and improves performance via individual units causing high or low activations.

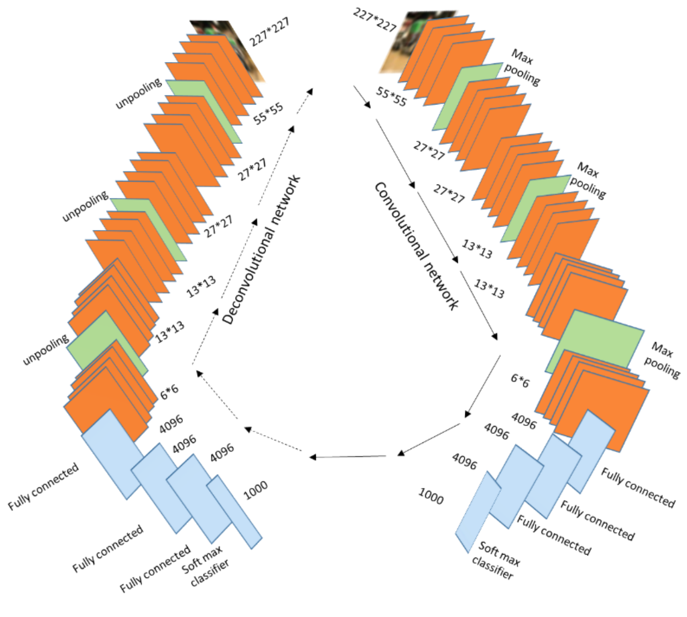


Figure 2.4: CNN with convolutional and de-convolutional paths defining the transition of input image as it passes through a convolutional, max pooling and fully connected layers, whereas a reverse strategy is seen in the de-convolutional pathway.

This approach is easy and fast compared to model-centric approaches but is computationally expensive as it deals with large amounts of data to improve the learning of these models. An example of such an approach is the deconvolution method given in [82], which represents a specific image responsible for the firing of each neural unit. On the other hand, model-centric techniques directly analyse a network, keeping the data constant and modifying the code for performance enhancements. Usually, these methods are slow and difficult to modify, for example, by making adjustments in the augmentations of the learning representation [16] or to generate synthetic datasets.

Different researchers have devised methods to gain more transparent and deeper insights into these complex networks by means of visualization. Their research varies depending upon the type of network architecture used, visualization methods and goals for better understanding of machine learning models. A detailed review of these visualization strategies is given in chapter 3.

2.3 Virtual Reality

Virtual reality (VR) is basically a computer-generated digital world or mimicry of the real-world environment. It gives the user the opportunity to create a wide variety of digital content in a number of application domains such as the advertising industry, media, gaming, education, medicine, flight simulations, scientific visualizations, and even in various sub-branches of engineering domains [108] and [109]. In a virtual reality setup, a user can physically become a part of the virtual scenery around them and interact with virtual objects to show movements as if they were in the real world. Although VR systems have been around since the 1960s, with the gradual rise of information technology and advances in the graphics industry, this field of study has gained considerable attention. However, its use was initially confined to a few academic and corporate research labs where these expensive VR devices were strictly kept for use by only a limited number of people.

Today, whatever the industry, VR devices are readily available at reasonable prices from a number of brands, the most popular being HTC Vive, Oculus, Google, Samsung and Microsoft. VR is a great way to understand, simulate, learn and interpret abstract concepts, and practice real skills before applying them in real-world scenarios to identify faults in the simulations and save budget costs for actual projects.

There are various forms of reality if considered on a virtual continuum scale, ranging from a real environment (in which we live) at one end and a virtual environment at the opposite end of the scale. However, there is another form of reality that lies between these opposite ends, known as mixed reality (or augmented reality [AR]), as shown below in Fig. 2.5. It is also sometimes further sub-divided into augmented reality or augmented virtuality (AV). How we try to replicate the real world and its object interactions in a virtual world depends on the goal of the application. However, in an augmented environment, there is no virtual world, but we can add certain interactive cues in the real world. In an AV, we can merge real-world objects into the virtual world, just like in immersive films. It's a matter of perspective between the two worlds that creates an illusion in the human mind. On the other hand, a VR environment is a computer-generated artificial environment that gives the user the feeling of being immersed in a completely new world that is very different from the real world in which they live, and makes the user become a part of that particular application.

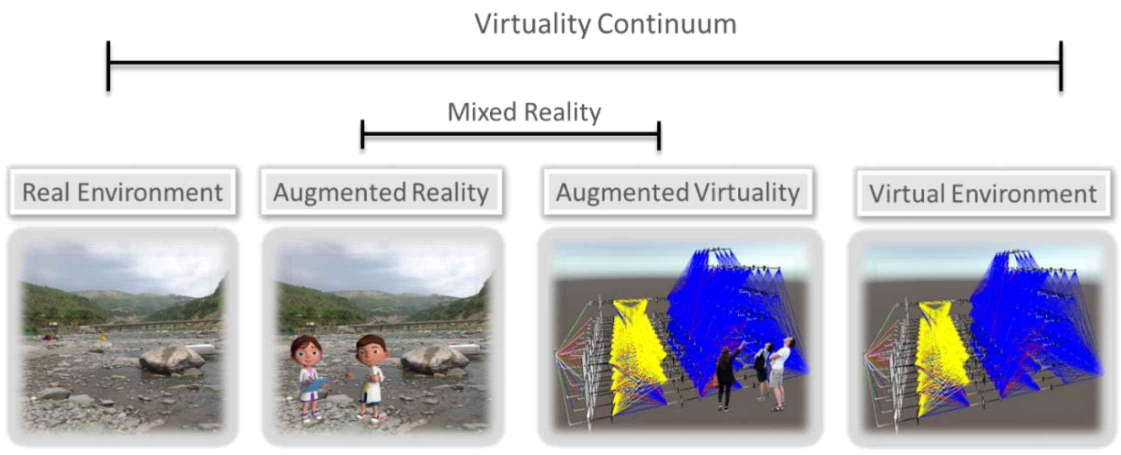


Figure 2.5: Different forms of reality shown on a virtuality continuum.

In order to communicate between the human and different reality worlds, we need specialized hardware devices that serve as input/output. These hardware components enable a user to have that immersive feeling of being part of a virtual world. This is done via cyclic operations that take input from the user, pass it on to different components of the virtual world and return the output to the user and vice versa, as shown in Fig. 2.6. This cycle involves taking **input** from the user (either through the eyes where the user is looking or from the hands that interact with virtual objects) to the virtual world application. This is where the **application's** objects are simulated, user interaction takes place and the dynamic geometry of the world is updated. To give the user that immersive experience, **rendering** is performed, which is basically the transformation of the computer-generated simulation into a user-friendly form that gives a sense of touch (haptics), visual rendering and even auditory rendering. The user receives an **output** as a physical representation in the form of sound via headphones or pixel displays in the form of images.

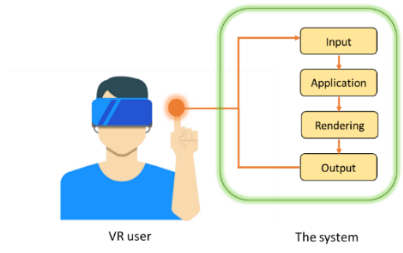


Figure 2.6: Cyclic representation of a VR system

2.3.1 VR system setup and components

Every VR setup requires some sort of installation and the necessary devices to allow the user to fully experience the immersive environment. In general, we can categorize the components into two main types: (i) hardware components and (ii) software components.

Usually, the hardware components include a personal computer to which other sensory devices can be connected that are necessary for interaction in the VR world. These sensory devices include a Head-mounted Display (HMD) for visuals along with headphones for the audio. In order to have uninterrupted communication, additional graphics and sound cards are added, and HMDs should be in a visible range for tracking devices to function properly. Input devices are some of the other VR hardware components without which interaction in a VR world is impossible. Typically, these devices include joysticks, hand controllers, electronic gloves to track the movement of the hands, a microphone, etc. These devices enable the user to touch or manipulate virtual objects during visual exploration and work in coordination with the tracking devices.

To actually build a VR environment, we need auxiliary graphics software that allows a user to design the virtual scenery and objects, assign specific visual properties and textures and even enhance the visual appearance of virtual objects. We can also integrate audio/video features into an object to give the impression that it is mimicking real-world objects. Finally, we need simulation software to tie all the components together and define the rules for the virtual world, such as how the objects will interact with each other, etc. In this work, we used the VR hardware supplied by HTC, and the HTC Vive VR gear set was used in the Unity gaming engine as simulation software. Chapters 4 and 5 describe in more detail how we utilized the VR setup and the application we demonstrated in the virtual environment in order to interpret the decisions of the deep neural network.

CHAPTER 3

Literature Review

The underlying theme of this work is to understand, visualize and assess the decisions of deep neural networks, particularly in the computer vision domain. Different researchers have devised methods to gain more transparent and deeper insights into these complex networks by means of visualization. Their research varies depending upon the type of network architecture used, visualization methods and goals for better understanding machine learning models

In this regard, we have complied a list of useful articles published in renowned conferences and journals for past ten years in a form of a review article. This work was compiled and reviewed during a training work at TU Dresden and is published as a book chapter with citation:

Seifert, C.; **Aamir, A**.; Balagopalan, A.; Jain, D.; Sharma, A.; Grottel, S. & Gumhold, S. Visualizations of Deep Neural Networks in Computer Vision: A Survey In Transparent data mining for Big and Small Data, pages 123–144. Springer.2017

The content of this chapter is identical to the original published version but written here to match the format of this thesis. All the references are compiled under a single list of references at the end of the thesis.

Visualizations of Deep Neural Networks in Computer Vision: A Survey

Christin Seifert[1], Aisha Aamir, Aparna Balagopalan, Dhruv Jain, Abhinav Sharma,

Sebastian Grottel and Stefan Gumhold

Abstract

In recent years, Deep Neural Networks (DNNs) have been shown to out-perform the state-of-the-art in multiple areas, such as visual object recognition, genomics and speech recognition. Due to the distributed encodings of information, DNNs are hard to understand and interpret. To this end, visualizations have been used to understand how deep architecture work in general, what different layers of the network encode, what the limitations of the trained model was and to interactively collect user feedback. In this chapter, we provide a survey of visualizations of DNNs in the field of computer vision. We define a classification scheme describing visualization goals and methods as well as the application area. This survey gives an overview of what can be learned from visualizing DNNs and which visualization methods were used to gain which insights. We found that most papers use pixel displays to show neuron activations. However, recently more sophisticated visualizations like interactive node-link diagrams were proposed. The presented overview can serve as a guideline when applying visualizations while designing DNNs.

3.1 Introduction

Artificial Neural Networks for learning mathematical functions have been introduced in 1943 [48]. Despite being theoretically able to approximate any function [8], their popularity decreased in the 1970's because their computationally expensive training

Christin Seifert e-mail: Christin.42.Seifert@gmail.com, Aisha Aamir e-mail: aishaaamir7@gmail.com,
Aparna Balagopalan e-mail: aparna.balagopalan@gmail.com, Dhruv Jain e-mail: dhruvjain.1027@gmail.com,
Abhinav Sharma e-mail: abhinav0301@gmail.com , Sebastian Grottel e-mail: sebastian.grottel@tu-dresden.de,
Stefan Gumhold e-mail:stefan.gumhold@tu-dresden.de
Technische Universit¨at Dresden, Germany

was not feasible with available computing resources [49]. With the increase in computing power in recent years, neural networks again became subject of research as Deep Neural Networks (DNNs). DNNs, artificial neural networks with multiple layers combining supervised and unsupervised training, have since been shown to outperform the state-of-the-art in multiple areas, such as visual object recognition, genomics and speech recognition [36]. Despite their empirically superior performance, DNN models have one disadvantage: their trained models are not easily understandable, because information is encoded in a distributed manner.

However, understanding and trust have been identified as desirable property of data mining models [65]. In most scenarios, experts can assess model performance on data sets, including gold standard data sets, but have little insights on how and why a specific model works [82]. The missing understandability is one of the reasons why less powerful, but easy to communicate classification models such as decision trees are in some applications preferred to very powerful classification models, like Support Vector Machines and Artificial Neural Networks [33].

Visualization has been shown to support understandability for various data mining models, e.g. for Naive Bayes [2] and Decision Forests [66]. In this chapter, we review literature on visualization of DNNs in the computer vision domain. Although DNNs have many application areas, including automatic translation and text generation, computer vision tasks are the earliest applications [35]. Computer vision applications also provide the most visualization possibilities due to their easy-to-visualize input data, i.e., images. In the review, we identify questions authors ask about neural networks that should be answered by a visualization (visualization goal) and which visualization methods they apply therefore. We also characterize the application domain by the computer vision task the network is trained for, the type of network architecture and the data sets used for training and visualization. Note that we only consider visualizations which are automatically generated. We do not cover manually generated illustrations (like the network architecture illustration in [35]). Concretely, our research questions are:

RQ-1 Which insights can be gained about DNN models by means of visualization?

RQ-2 Which visualization methods are appropriate for which kind of insights?

To collect the literature we pursued the following steps: since deep architectures became prominent only a few years ago, we restricted our search starting from the year 2010. We searched the main conferences, journals and workshops in the area of computer vision, machine learning and visualization, such as: IEEE International Conference on Computer Vision (ICCV), IEEE Conferences on Computer Vision and Pattern Recognition (CVPR), IEEE Visualization Conference (VIS), Advances in Neural Information Processing Systems (NIPS). Additionally, we used keyword- based search in academic search engines, using the following phrases (and combinations): "deep neural networks", "dnn", "visualization", "visual analysis", "visual representation", "feature visualization". This chapter is organized as follows: the next section introduces the classification scheme and describes the categories we applied to the collected papers. Section 3 reviews the literature according to the introduced categories. We discuss the findings with respect to the introduced research questions in section 4, and conclude the work in section 5.

3.2 Classification Scheme

In this chapter we present the classification scheme used to structure the literature: we first introduce a general view, and then provide detailed descriptions of the categories and their values. An overview of the classification scheme is shown in Fig 3.1. First, we need to identify the purpose the visualization was developed for. We call this category visualization goal. Possible values are for instance general understanding and model quality assessment. Then, we identified the visualization methods used to achieve the above mentioned goals. Such methods can potential cover the whole visualization space [51], but literature review shows that only a very small subset has been used so far in the context of DNNs, including heat maps and visualizations of confusion matrices. Additionally, we introduced three categories to describe the application domain. These categories are the computer vision task, the architecture type of the network and the data sets the neural network was trained on, which is also used for the visualization. Note, that the categorization is not distinct. This means that one paper can be assigned multiple values in one category. For instance, a paper can use multiple visualization methods (CNNVis uses a combination of node-link diagrams, matrix displays and heatmaps [44]) on multiple data sets.

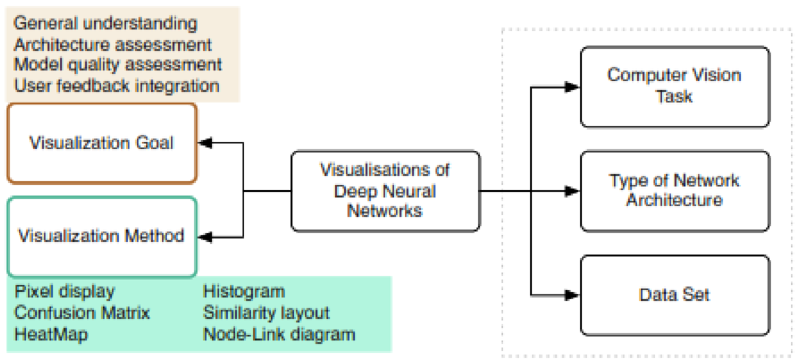


Figure 3.1: Classification Scheme for Visualizations of Deep Neural Networks. The dotted border subsumes the categories characterizing the application area.

Related to the proposed classification scheme is the taxonomy of Gr¨un et al. for visualizing learned features in convolutional neural networks [25]. The authors categorize the visualization methods into input modification, de-convolutional and input reconstruction methods. In input modification methods, the output of the network and intermediate layers is measured while the input is modified. De-Convolutional methods adapt a reverse strategy to calculate the influence of a neuron's activation from lower layers. This strategy demonstrates which pixels are responsible for the activation of neurons in each layer of the network. Input reconstruction methods try to assess the importance of features by reconstructing input images. These input images can either be real or artificial images, that either maximize or lead to an output invariance of a unit of interest. This categorization is restricted to feature visualizations and therefore narrower as the proposed scheme. For instance, it does not cover the general application domain, and is restricted to specific type of visualizations, because it categorizes the calculation methods used for pixel displays and heatmaps.

Visualization Goals

This category describes the various goals of the authors visualizing DNNs. We identified the following four main goals:

- *General Understanding*: This category encompasses questions about general behavior of the neural network, either during training, on the evaluation data set or on unseen images. Authors want to find out what different network layers are learning or have learned, on a rather general level.
- *Architecture Assessment*: Work in this category tries to identify how the network architecture influences performance in detail. Compared to the first category the analyses are on a more fine-grained level, e.g. assessing which layers of the architecture represent which features (e.g., color, texture), and which feature combinations are the basis for the final decision.
- *Model Quality Assessment*: In this category authors have focused their research goal in determining how the number of layers and role played by each layer can affect the visualization process.
- *User Feedback Integration*: This category comprises work in which visualization is the means to integrate user feedback into the machine learning model. Examples for such feedback integration are user-based selection of training data [58] or the interactive refinement of hypotheses [21].

Visualization Methods

Only a few visualization methods [51] have been applied to DNNs. We briefly describe them in the following.

- *Histogram*: A histogram is a very basic visualization showing the distribution of univariate data as a bar chart.
- *Pixel Displays*: The basic idea is that each pixel represents a data point. In the context of DNN, the (color) value for each pixel is based on network activation,

reconstructions or similar and yield 2-dimensional rectangular images. In most cases the pixels next to each other in the display space are also next to each other in the semantic space (e.g., nearby pixels of the original image). This nearness criterion is defined on the difference from Dense Pixel Displays [32]. We further distinguish whether the displayed values originate from a single image, from a set of images (i.e., a batch), or only from a part of the image.

- *Heat Maps*: Heat maps are a special case of pixel displays, where the value for each pixel represents an accumulated quantity of some kind and is encoded using a specific coloring scheme [73]. Heat maps are often transparently overlaid over the original data.
- *Similarity Layout*: In similarity-based layouts the relative positions of data objects in the low-dimensional display space is based on their pair-wise similarity. Similar objects should be placed nearby in the visualization space, dissimilar objects farther apart. In the context of images as objects, suitable similarity measures between images have to be defined [53].
- *Confusion Matrix Visualization*: This technique combines the idea of heatmaps and matrix displays. The classifier confusion matrix (showing the relation between true and predicted classes) is colored according to the value in each cell. The diagonal of the matrix indicates correct classification and all the values other than the diagonal are errors that need to be inspected. Confusion matrix visualizations have been applied to clustering and classification problems in other domains [70].
- *Node-Link Diagrams*: are visualizations of (un-)directed graphs [1], in which nodes represents objects and links represent relations between objects.

Computer Vision Tasks

In the surveyed papers different computer vision tasks were solved by DNNs. These are the following:

- *Classification*: The task is to categorize image pixels into one or more classes.
- *Tracking*: Object tracking is the tasks of locating moving objects over time.

- *Recognition*: Object recognition is the task of identifying objects in an input image by determining their position and label.
- *Detection*: Given an object and an input image the task in object detection is to localize this object in the image, if it exists.
- *Representation Learning*: This task refers to learning features suitable for object recognition, tracking etc. Examples of such features are points, lines, edges, textures or geometric shapes.

Network Architectures

We identified six different types of network architectures in the context of visualization. These types are not mutually exclusive, since all types belong to DNNs, but some architectures are more specific, either w.r.t. the types of layers, the type of connections between the layers or the learning algorithm used.

- DNN: Deep Neural Networks are the general type of feed-forward networks with multiple hidden layers.
- *CNN*: Convolutional Neural Networks are a type of feed-forward networks specifically designed to mimic the human visual cortex [22]. The architecture consists of multiple layers of smaller neuron collections processing portions of the input image (convolutional layers) generating low-level feature maps. Due to their specific architecture CNNs have much fewer connections and parameters compared to standard DNNs, and thus are easier to train.
- *DCNN*: The Deep Convolution Neural Network is a CNN with a special eight layer architecture [35]. The first five layers are convolutional layers and the last three layers are fully connected.
- *DBN*: Deep Belief Networks can be seen as a composition of Restricted Boltzmann Machines (RBMs) and are characterized by a specific training algorithm [27]. The top two layers of the network have undirected connections whereas the lower layers have directed connection with the upper layers.
- *CDBN*: Convolutional Deep Belief Networks are similar to DBNs, containing Convolutional RBMs stacked on one another [38]. Training is performed similar to

DBNs using a greedy layer-wise learning procedure i.e. the weights of trained layers are fixed and considered as input for the next layer.

- *MCDNN*: The Multicolumn Deep Neural Networks is basically a combination of several DNN stacked in column form [7]. The input is processed by all DNNs and their output aggregated to the final output of the DNN.

In the next section we will apply the presented classification scheme (cf. Figure 1) to the selected papers and provide some statistics on the goals, methods and application domains. Additionally, we categorize the papers according to the taxonomy of Gr¨un [25] (input modification methods, de-convolutional methods and input reconstruction) if this taxonomy is applicable.

3.3 Visualizations of Deep Neural Networks

Table 1 provides an overview of all papers included in this survey and their categorization. The table is sorted first by publication year and then by author name. In the following, the collected papers are investigated in detail, whereas the subsections correspond to the categories derived in the previous section.

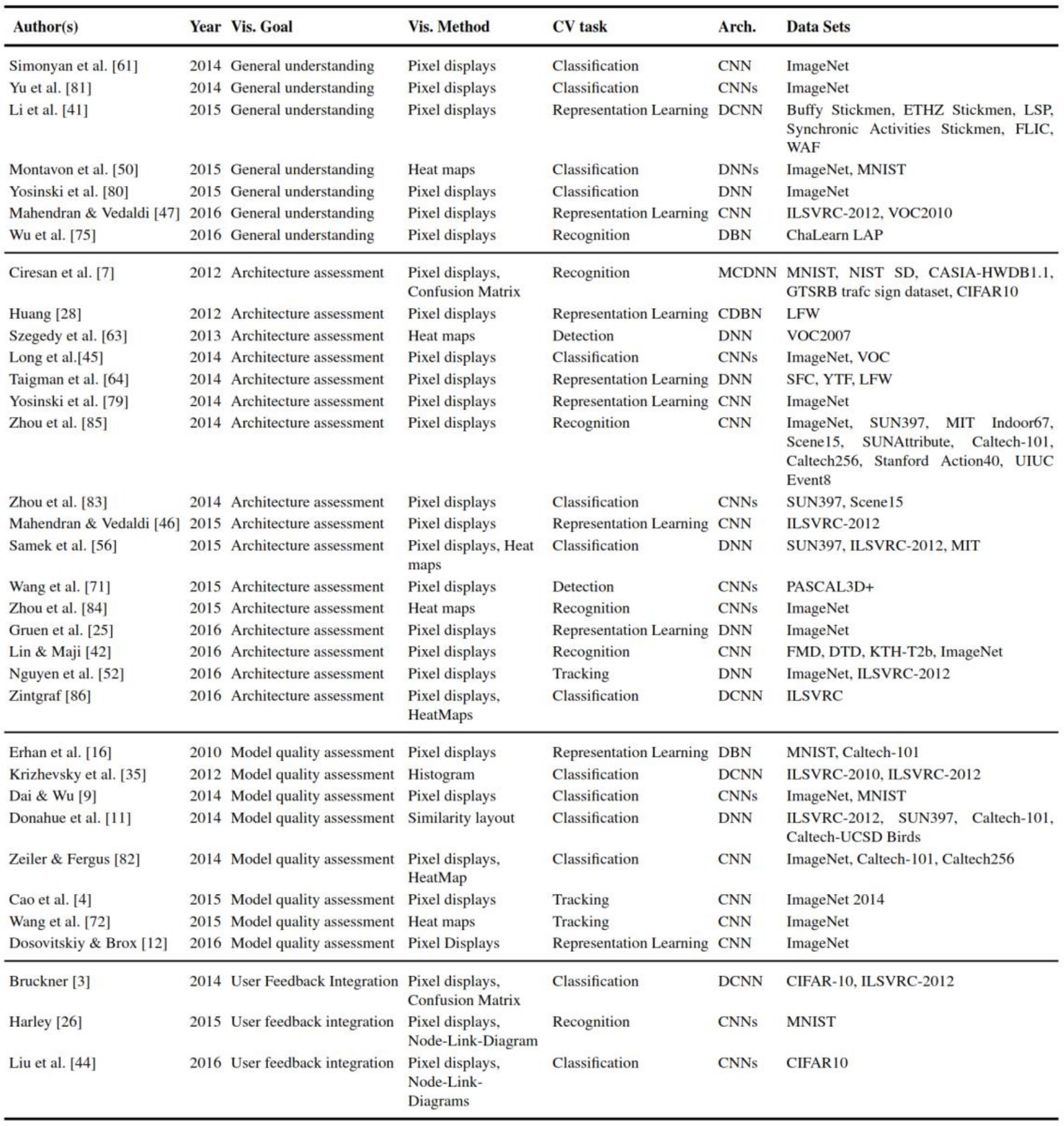
3.3.1 Visualization Goals

Table 2 provides an overview of the papers in this category. The most prominent goal is architecture assessment (16 papers). Model quality assessment was covered in 8 and general understanding in 7 papers respectively, while only 3 authors approach interactive integration of user feedback. Authors who have contributed work on visualizing DNNs with the goal general understanding have focused on gaining basic knowledge of how the network performs its task. They aimed to understand what each network layer is doing in general. Most of the work in this category conclude that lower layers of the networks contains representations of simple features like edges and lines, whereas deeper layers tend to be more class-specific and learn complex image features [61, 41, 47]. Some authors developed tools to get a better understanding of learning capabilities of convolutional networks[2][80, 3]. They demonstrated that such tools can provide a means to visualize the activations produced in response to user inputs and showed how the network behaves on unseen data. Approaches providing deeper insights into the architecture were placed into the category architecture assessment. Authors focused their research on determining how these networks capture representations of texture, color and other features that discriminate an image from another, quite similar image [56]. Other authors tried to assess how these deep architectures arrive at certain decisions [42] and how the input image data affects the decision making capability of these networks under different conditions. These conditions include image scale, object translation and cluttered background scenes. Further, authors investigated which features are learned, and whether the neurons are able to learn more than one feature in order to arrive at a decision [52]. Also, the contribution of image parts for activation of specific neurons was investigated [86] in order to understand for instance, what part of a dog's face needs to be visible for the network to detect it as a dog. Authors also investigated what types of features are transferred from lower to higher layers [79, 80], and have shown for instance that scene centric and object centric features are represented differently in the network [85]. Eight papers contributed work on model quality assessment. Authors have focused their research on how the individual layers can be effectively visualized, as well as the effect on the network's performance. The contribution of each layer at different level greatly influence their role played in computer vision tasks. One such work determined how the convolutional layers at various levels of the network show varied properties in tracking purposes [72]. Dosovitskiy & Bronx have shown that higher convolutional layers retain details of object location, color and contour information of the image [12]. Visualization is used as a means to improve tools for finding good interpretations of features learned in higher levels [16]. Kriszhesvsky et al. focused on performance of individual layers and how performance degrades when certain layers in the network are removed [35]. Some authors researched user feedback integration. In the

[2] Tools available http://yosinski.com/deepvis and https://github.com/bruckner/deepViz, last accessed 2016-09-08

interactive node-link visualization in [26] the user can provide his/her own training data using a drawing area. This method is strongly tied to the used network and training data (MNIST hand written digit). In the Ml-O-Scope system users can interactively analyze convolutional neural networks [3]. Users are presented with a visualization of the current model performance, i.e. the a-posteriori probability distribution for

Table 1 Overview of all reviewed papers



Author(s)	Year	Vis. Goal	Vis. Method	CV task	Arch.	Data Sets
Simonyan et al. [61]	2014	General understanding	Pixel displays	Classification	CNN	ImageNet
Yu et al. [81]	2014	General understanding	Pixel displays	Classification	CNNs	ImageNet
Li et al. [41]	2015	General understanding	Pixel displays	Representation Learning	DCNN	Buffy Stickmen, ETHZ Stickmen, LSP, Synchronic Activities Stickmen, FLIC, WAF
Montavon et al. [50]	2015	General understanding	Heat maps	Classification	DNNs	ImageNet, MNIST
Yosinski et al. [80]	2015	General understanding	Pixel displays	Classification	DNN	ImageNet
Mahendran & Vedaldi [47]	2016	General understanding	Pixel displays	Representation Learning	CNN	ILSVRC-2012, VOC2010
Wu et al. [75]	2016	General understanding	Pixel displays	Recognition	DBN	ChaLearn LAP
Ciresan et al. [7]	2012	Architecture assessment	Pixel displays, Confusion Matrix	Recognition	MCDNN	MNIST, NIST SD, CASIA-HWDB1.1, GTSRB trafc sign dataset, CIFAR10
Huang [28]	2012	Architecture assessment	Pixel displays	Representation Learning	CDBN	LFW
Szegedy et al. [63]	2013	Architecture assessment	Heat maps	Detection	DNN	VOC2007
Long et al.[45]	2014	Architecture assessment	Pixel displays	Classification	CNNs	ImageNet, VOC
Taigman et al. [64]	2014	Architecture assessment	Pixel displays	Representation Learning	DNN	SFC, YTF, LFW
Yosinski et al. [79]	2014	Architecture assessment	Pixel displays	Representation Learning	CNN	ImageNet
Zhou et al. [85]	2014	Architecture assessment	Pixel displays	Recognition	CNN	ImageNet, SUN397, MIT Indoor67, Scene15, SUNAttribute, Caltech-101, Caltech256, Stanford Action40, UIUC Event8
Zhou et al. [83]	2014	Architecture assessment	Pixel displays	Classification	CNNs	SUN397, Scene15
Mahendran & Vedaldi [46]	2015	Architecture assessment	Pixel displays	Representation Learning	CNN	ILSVRC-2012
Samek et al. [56]	2015	Architecture assessment	Pixel displays, Heat maps	Classification	DNN	SUN397, ILSVRC-2012, MIT
Wang et al. [71]	2015	Architecture assessment	Pixel displays	Detection	CNNs	PASCAL3D+
Zhou et al. [84]	2015	Architecture assessment	Heat maps	Recognition	CNNs	ImageNet
Gruen et al. [25]	2016	Architecture assessment	Pixel displays	Representation Learning	DNN	ImageNet
Lin & Maji [42]	2016	Architecture assessment	Pixel displays	Recognition	CNN	FMD, DTD, KTH-T2b, ImageNet
Nguyen et al. [52]	2016	Architecture assessment	Pixel displays	Tracking	DNN	ImageNet, ILSVRC-2012
Zintgraf [86]	2016	Architecture assessment	Pixel displays, HeatMaps	Classification	DCNN	ILSVRC
Erhan et al. [16]	2010	Model quality assessment	Pixel displays	Representation Learning	DBN	MNIST, Caltech-101
Krizhevsky et al. [35]	2012	Model quality assessment	Histogram	Classification	DCNN	ILSVRC-2010, ILSVRC-2012
Dai & Wu [9]	2014	Model quality assessment	Pixel displays	Classification	CNNs	ImageNet, MNIST
Donahue et al. [11]	2014	Model quality assessment	Similarity layout	Classification	DNN	ILSVRC-2012, SUN397, Caltech-101, Caltech-UCSD Birds
Zeiler & Fergus [82]	2014	Model quality assessment	Pixel displays, HeatMap	Classification	CNN	ImageNet, Caltech-101, Caltech256
Cao et al. [4]	2015	Model quality assessment	Pixel displays	Tracking	CNN	ImageNet 2014
Wang et al. [72]	2015	Model quality assessment	Heat maps	Tracking	CNN	ImageNet
Dosovitskiy & Brox [12]	2016	Model quality assessment	Pixel Displays	Representation Learning	CNN	ImageNet
Bruckner [3]	2014	User Feedback Integration	Pixel displays, Confusion Matrix	Classification	DCNN	CIFAR-10, ILSVRC-2012
Harley [26]	2015	User feedback integration	Pixel displays, Node-Link-Diagram	Recognition	CNNs	MNIST
Liu et al. [44]	2016	User feedback integration	Pixel displays, Node-Link-Diagrams	Classification	CNNs	CIFAR10

Table 2 Overview of visualization goals

Category	# Papers	References
Architecture assessment	16	[7, 25, 28, 42, 45, 46, 52, 56, 63, 64, 71, 79, 83, 84, 85, 86]
Model quality assessment	8	[4, 9, 11, 12, 16, 35, 72, 82]
General understanding	7	[41, 47, 50, 61, 80, 75, 81]
User feedback integration	3	[3, 26, 44]

input images and pixel displays of activations within selected network layers. They are also provided with a user interface for interactive adaption of model hyper-parameters. A visual analytics approach to DNN training has been proposed recently [44].The authors present 3 case studies in which DNN experts evaluated a network, assessed errors and found directions for improvement (e.g. adding new layers).

3.4 Visualization Methods

In this section we describe the different visualization methods applied to DNNs. An overview of the methods is provided in Table 3. We also categorize the papers according to Grün's taxonomy [25] in Table 4. In the following we describe the papers for each visualization method separately.

Table 3 Overview of visualization methods

Category	Sub-Category	# Papers	References
Pixel displays	single image	24	[4, 7, 9, 12, 16, 25, 26, 41, 42, 44, 45, 46, 47, 52, 56, 61, 72, 71, 75, 79, 80, 81, 82, 86]
	image batch	4	[3, 28, 35, 85]
	part of image	2	[64, 83]
Heat maps		6	[50, 56, 63, 72, 82, 84, 86]
Confusion matrix		2	[3, 7]
Node-Link-Diagrams		2	[26, 44]
Similarity layout		1	[11]
Histogram		1	[35]

Table 4 Overview of categorization by Gr¨un [25]

Category	# Papers	References
Deconvolution	24	[3, 4, 7, 9, 12, 16, 26, 28, 35, 41, 45, 50, 52, 56, 61, 63, 64, 71, 72, 81, 83, 84, 85]
Input modification	6	[44, 75, 79, 80, 82, 86]
Input Reconstruction	4	[42, 46, 47, 61]

3.4.1 Pixel displays

Most of the reviewed work has utilized pixel based activations as a means to visualize different features and layers of deep neural networks. The basic idea behind such visualization is that each pixel represents a data point. The color of the pixel corresponds to an activation value, the maximum gradient w.r.t. to a given class, or a reconstructed image. The different computational approaches for calculating maximum activations, sensitivity values or reconstructed images are not within the scope of this chapter. We refer to the survey paper for feature visualizations in DNNs [25] and provide a categorization of papers into Gr¨un's taxonomy in Table 4. Mahendran & Vedaldi [46],[47] have visualized the information contained in the image by using a process of inversion using optimized gradient descent function. Visualizations are used to show the representations at each layer of the network (cf. Fig. 3.2). All the convolutional layers maintain photographically realistic representations of the image. The first few layers are specific to the input images and form a direct invertible code base. The fully connected layers represent data with less geometry and instance specific information. Activation signals can thus be invert back to images containing parts similar, but not identical to the original images. Cao et al. [4] have used pixel displays on complex, cluttered, single images to visualize their results of CNNs with feedback. Nguyen et al. [52] developed an algorithm to demonstrate that single neurons can represent multiple facets. Their visualizations show the type of image features that activate specific neurons. A regularization method is also presented to determine the interpretability of the images to maximize activation. The results suggest that synthesizing visualizations from activated neurons better represent input images in terms of the overall structure and color. Simonyan et al. [61] visualized data for deep convolutional networks. The first visualization is a numerically generated image to maximize a classification score. As second visualization, saliency maps for given pairs of images and classes indicate influence of pixels from the input image on the respective class score, via back-propagation.

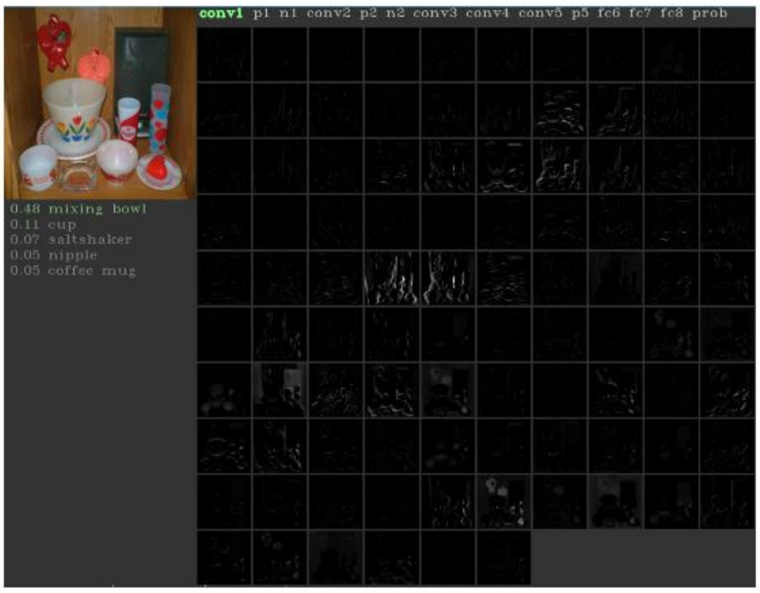


Figure. 3.2: Pixel based display. Activations of first convolutional layer generated with the DeepVis toolboxfrom [80] from https://github.com/yosinski/deep-visualization-toolbox/.

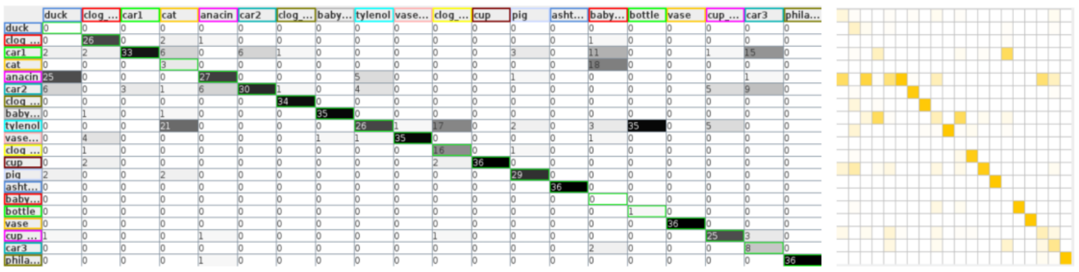
3.4.2 Heat Maps

In most cases, heat maps were used for visualizing the extend of feature activations of specific network layers for various computer vision tasks (e.g. classification [82], tracking [72], detection [84]). Heat maps have also been used to visualize the final network output, e.g. the classifier probability [63], [82]. The heat map visualizations are used to study the contributions of different network layers (e.g. [72]), compare different methods (e.g., [50]) or investigate the DNNs inner features and results on different input images [84]. Zintgraf et al. [86] used heat maps to visualize image regions in favor of, as well as image regions against, a specific class in one image.

Authors use different color coding's for their heat maps: blue-red-yellow color schemes [72], [82], [84], white-red scheme [50], blue-white-red [86] and also a simple grayscale highlighting interesting regions in white [63].

3.4.3 Confusion Matrix and Histogram

Two authors have shown the confusion matrix to illustrate the performance of the DNN w.r.t. a classification task (see Fig 3.3). Bruckner et al. [3] additionally encoded the value in each cell using color (darker color represents higher values). Thus, in this visualization dark off-diagonal spots correspond to large errors. In [7] the encoding used is different: each cell value is additionally encoded by the size of a square. Cells containing large squares represent large values; a large off-diagonal square corresponds to a large error between two classes. Similarly, in one paper histograms have been used to visualize the decision uncertainty of a classifier, indicating using color whether the highest-probable class is the correct one [35].



	duck	clog_...	car1	cat	anacin	car2	clog_...	baby...	tylenol	vase...	clog_...	cup	pig	asht...	baby...	bottle	vase	cup_...	car3	phila...
duck	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
clog ...	0	26	0	2	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
car1	2	2	33	6	0	6	1	0	0	0	0	0	3	0	11	0	0	1	15	0
cat	0	0	0	3	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0
anacin	25	0	0	0	27	0	0	0	5	0	0	0	1	0	0	0	0	0	1	0
car2	6	0	3	1	6	30	1	0	4	0	0	0	0	0	0	0	0	5	9	0
clog ...	0	0	0	0	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0	0
baby...	0	1	0	1	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0
tylenol	0	0	0	21	0	0	0	0	26	1	17	0	2	0	3	35	0	5	0	0
vase...	0	4	0	0	0	0	0	1	1	35	0	0	0	0	1	0	0	0	0	0
clog ...	0	1	0	0	0	0	0	0	0	0	16	0	1	0	0	0	0	0	0	0
cup	0	2	0	0	0	0	0	0	0	0	2	36	0	0	0	0	0	0	0	0
pig	2	0	0	2	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0
asht...	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0
baby...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bottle	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
vase	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0
cup ...	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	25	3	0
car3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	8	0
phila...	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36

Figure. 3.3: Confusion Matrix example. Showing classification results for the COIL-20 data set. Screenshots reproduced with software from [59].

3.4.4 Similarity based layout

In the context of DNNs, similarity based layouts so far have been applied only by Donahue et al. [11], who specifically used t-distributed stochastic neighbor embedding (t-SNE) [68] of feature representations. The authors projected feature representations of different networks layers into the 2-dimensional space and found a visible clustering for the higher layers in the network, but none for features of the lower network layer. This finding corresponds to the general knowledge of the community that higher levels learn semantic or high-level features. Further, based on the projection the authors could conclude that some feature representation is a good choice for generalization to other (unseen) classes and how traditional features compare to feature representations learned by deep architectures. Fig 3.4 provided an example of the latter.

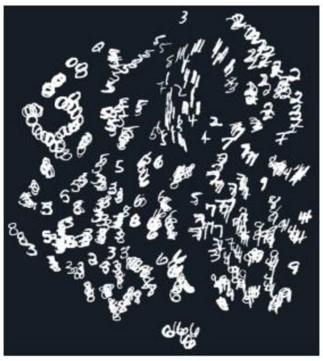
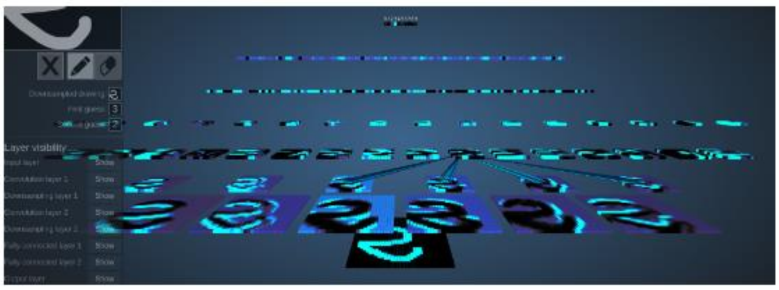


Figure. 3.4: Similarity based layout of the MNIST data set using raw features. Screenshot was taken with a JavaScript implementation of t-SNE[67] https://scienceai.github.io/tsne-js/.

3.4.5 Node-Link Diagrams

Two authors have approach DNN visualization with node-link diagrams (see examples in Fig 3.5). In his interactive visualization approach, Adam Harley represented layers in the neural networks as nodes using pixel displays, and activation levels as edges [26]. Due to the denseness of connections in DNNs only active edges are visible. Users can draw input images for the network and interactively explore how the DNN is trained. In CNNVis [44] nodes represent neuron clusters and are visualized in different ways (e.g., activations) showing derived features for the clusters.



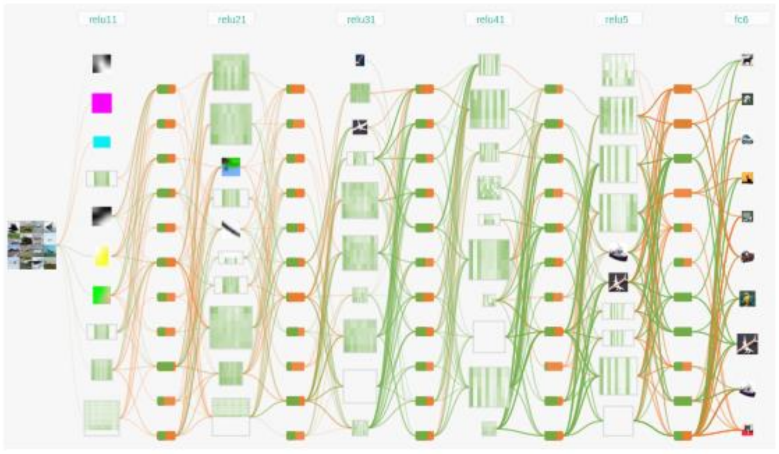


Figure.3.5: Node-link diagrams of DNNs. Top: Example from [26] taken with the online application at http://scs.ryerson.ca/aharley/vis/conv/. Bottom: screenshot of the CNNVis system [44] taken with the online application at http://shixialiu.com/publications/cnnvis/demo/.

3.5 Network Architecture and Computer Vision Task

Table 5 provides a summary of the architecture types. The majority of papers applied visualizations to CNN architectures (18 papers), while 8 papers dealt with the more general case of DNNs. Only 8 papers have investigated more special architectures, like DCNN (4 papers), DBNs (2 papers), CDBN (1 paper) and MCDNNs (1 paper). Table 6 summarizes the computer vision tasks for which the DNNs have been trained. Most networks were trained for classification (14 papers), some for representation learning and recognition (9 and 6 papers, respectively). Tracking and Detection were pursued the least often.

Table 5 Overview of network architecture types

Category	# Papers	References
CNN	18	[4, 9, 12, 26, 42, 44, 45, 46, 47, 61, 71, 72, 79, 81, 82, 83, 84, 85]
DNN	8	[11, 25, 50, 52, 56, 63, 64, 80]
DCNN	4	[3, 35, 41, 86]
DBN	2	[16, 75]
CDBN	1	[28]
MCDNN	1	[7]

Table 6 Overview of computer vision tasks

Category	# Papers	References
Classification	14	[3, 9, 11, 35, 44, 45, 50, 56, 61, 80, 81, 82, 83, 86]
Representation learning	9	[12, 16, 25, 28, 41, 46, 47, 64, 79]
Recognition	6	[7, 26, 42, 75, 84, 85]
Tracking	3	[4, 52, 72]
Detection	2	[63, 71]

3.6 Data Sets

Table 7 provides an overview of the data sets used in the reviewed papers. In the field of classification and detection, the ImageNet dataset represent the most frequently used dataset, used around 21 times. Other popular datasets used in tasks involving detection and recognition such as Caltech101, Caltech256 etc. have been used 2-3 times (e.g. in [11], [56], [82], [85]). While ImageNet and its subsets (e.g. ISLVRC) are large datasets with around 10,000,000 images each, there are smaller datasets such as the ETHZ stickmen and VOC2010 which are generally used for fine-grained classification and learning. VOC2010, consisting of about 21,738 images, has been used twice, while more specialized data sets, such as Buffy Stickmen for representation learning, have been used only once in the reviewed papers [41]. There are datasets used in recognition with fewer classes such as CIFAR10, consisting of 60,000 color images, with about 10 classes; and MNIST used for recognition of handwritten digits.

Table 7 Overview of data sets sorted after their usage. Column"#" refers to the number of papers in this survey using this data set.

Data Set	Year	# Images	CV Task	Comment	#	References
ImageNet [10]	2009	14,197,122	classification, tracking	21841 synsets	21	[80, 61, 56, 42, 47, 52, 86, 79, 4, 11, 72, 35, 46, 25, 82, 86, 56, 52, 85, 12, 3]
ILSVRC2012 [55]	2015	1,200,000	classification, detection, representation learning	1000 object categories	7	[52, 47, 56, 46, 3, 11, 35]
VOC2010 [18]	2010	21,738	detection, representation learning	50/50 train-test split	3	[63, 47, 45]
Caltech-101 [19]	2006	9146	recognition, classification	101 categories	3	[85, 82, 11]
Places [85]	2014	2,500,000	classification, recognition	205 scene categories	2	[85, 56]
Sun397 [77]	2010	130,519	classification, recognition	397 categories	2	[85, 56]
Caltech256 [23]	2007	30,607	classification,recognition	256 categories	2	[85, 82]
LFW [29]	2007	13,323	representation learning	5,749 faces, 6,000 pairs	2	[64, 28]
MNIST [37]	1998	70,000	recognition	60,000 train, 10,000 test 10 classes, hand-written digits	2	[16, 7]
DTD [6]	2014	5640	recognition	47 terms (categories)	1	[42]
ChaLearn LAP [17]	2014	47933	recognition	RGB-D gesture videos with 249 gestures labels, each 249 for train/testing/validation	1	[75]
SFC [64]	2014	4,400,000	representation learning	photos of 4030 people	1	[64]
PASCAL3D+ [76]	2014	30899	detection		1	[71]
FLIC [57]	2013	5003	representation learning	30 movies with person detector, 20% for testing	1	[40]
Synchronic Activities Stickmen[15]	2012	357	representation learning	upper-body annotations	1	[40]
Buffy Stickmen [30]	2012	748	representation learning	ground-truth stickmen annotations, anno-tated video frames	1	[40]
SUNAttribute [54]	2012	14,000	recognition	700 categories	1	[85]
CASIA-HWDB1.1 [43]	2011	1,121,749	recognition	897,758 train, 223,991 test ,3755 classes, chinese handwriting	1	[7]
GTSRB traffic sign dataset[62]	2011	50,000	recognition	>40 classes, single-image, multi-class classification	1	[7]
Caltech-UCSD Birds [69]	2011	11,788	classification	200 categories	1	[11]
YTF [74]	2011	3425	representation learning	videos, subset of LFW	1	[64]
Stanford Action40 [78]	2011	9532	recognition	40 actions, 180-300 images per action class	1	[85]
WAF [14]	2010	525	representation learning	downloaded via Google Image Search	1	[40]
LSP [31]	2010	2000	representation learning	pose annotated images with 14 joint loca-tion	1	[40]
ETHZ Stickmen [13]	2009	549	representation learning	annotated by a 6-part stickman	1	[40]
CIFAR10 [34]	2009	60000	recognition	50000 training and 10000 test of 10 classes	1	[7]
FMD [60]	2009	1000	recognition	10 categories, 100 images per category	1	[42]
UIUC Event8 [39]	2007	1579	recognition	sports event categories	1	[85]
KTH-T2b [5]	2005	4752	recognition	11 materials captured under controlled scale, pose, and illumination	1	[42]
Scene15 [20]	2005	4485	recognition	200 to 400 images per class of 15 class scenes	1	[85]
NIST SD 19 [24]	1995	800000	recognition	forms and digits	1	[7]

3.7 Discussion

In this section we discuss the implications of the findings from the previous section with respect to the research questions. We start the discussion by evaluating the results for the stated research questions. RQ-1 (Which insights can be gained about DNN models by means of visualization) has been discussed along with the single papers in the previous section in detail. We showed by examples which visualizations have previously been shown to lead to which insights. For instance, visualizations are used to learn which features are represented in which layer of a network or which part of the image a certain node reacts to. Additionally, visualizing synthetic input images which maximize activation allows to better understand how a network as a whole works. To strengthen our point here, we additionally provide some quotes from authors: Heat maps: "*The visualization method shows which pixels of a specific input image are evidence for or against a node in the network*." [86]. Similarity layout: "*[. . .] first layers learn 'low-level' features, whereas the latter layers learn semantic or 'high-level' features. [. . .] GIST or LLC fail to capture the semantic difference [. . .]*" [11]. Pixel Displays: "*[. . .] representations on later convolutional layers tend to be somewhat local, where channels correspond to specific, natural parts (e.g. wheels, faces) instead of being dimensions in a completely distributed code. That said, not all features correspond to natural parts [. . .]*" [80]

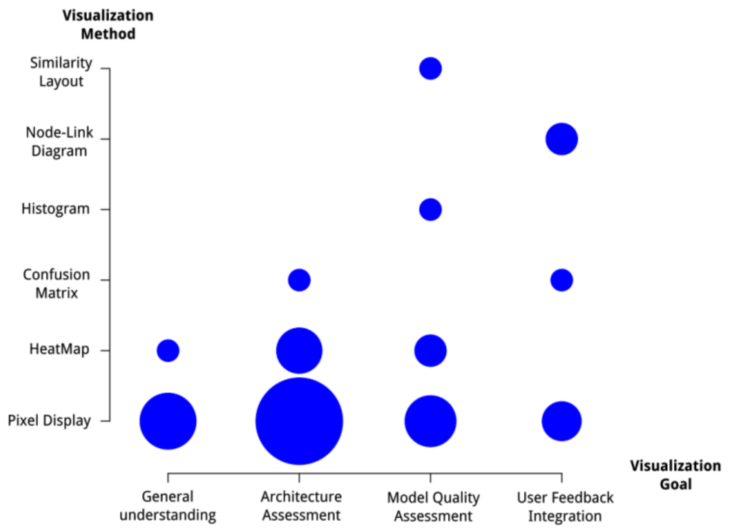


Figure. 3.6: Relation of visualization goals and applied methods in the surveyed papers following our taxonomy. Size of the circles corresponds to the (square root of the) number of papers in the respective categories. For details on papers see Table 1.

The premise to use visualization is thus valid, as the publications agree that visualizations help to understand the functionality and behavior of DNNs in computer vision. This is especially true when investigating specific parts of the DNN. To answer **RQ-2** (Which visualization methods are appropriate for which kind of insights?) we evaluated which visualizations were applied in the context of which visualization goals. A summary is shown in Fig 3.6. It can be seen that not all methods were used in combination with all goals, which is not surprising. For instance, no publication used a similarity layout for assessing the architecture. This provides hints on possibilities for further visualization experiments. Pixel displays were prevalent for architecture assessment and general understanding. This is plausible since DNNs for computer vision work on the images themselves. Thus, pixel displays preserve the spatial-context of the input data, making the interpretation of the visualization straight-forward. This visualization, however, method has its own disadvantages and might not be the ideal choice in all cases. The visualization design space is extremely limited, i.e. constrained to a simple color mapping. Especially for more complex research questions, extending this space might be worthwhile, as the other visualization examples in this review show. The fact that a method has not been used w.r.t. a certain goal does not necessarily mean that it would not be appropriate. It merely means that authors so far achieved their goal with a different kind of visualization. The results based on our taxonomy, cf. Fig.3.6 and Table. 1, hint at corresponding white spots. For example, node-link diagrams are well suited to visualize dependencies and relations. Such information could be extracted for architecture assessment as well, depicting which input images and activation levels correlate highly to activations within individual layers of the network. Such a visualization will neither be trivial to create nor to use, since this first three part correlation requires suitable hyper-graph visualization metaphor, but the information basis is promising. Similar example ideas can be constructed for the other white spots in Fig. 3.6 and beyond.

3.8 Summary and Conclusion

In this chapter we surveyed visualizations of DNNs in the computer vision domain. Our leading questions were: “Which insights can be gained about DNN models by means of visualization?” and “Which visualization methods are appropriate for which kind of insights?” A taxonomy containing the categories visualization method, visualization goal, network architecture type, computer vision task and data set was developed to structure the domain. We found that pixel displays were most prominent among the methods, closely followed by heat maps. Both is not surprising, given that images (or image sequences) are the prevalent input data in computer vision. Most of the developed visualizations and/or tools are expert tools, designed for the usage of DNN/computer vision experts. We found no interactive visualization allowing to integrate user feedback directly into the model. The closest approach is the semi-automatic CNNVis tool [44]. An interesting next step would be to investigate which of the methods have been used in other application areas of DNNs, such as speech recognition, where pixel displays are not the most straight-forward visualization. It would be also interesting to see which visualization knowledge and techniques could be successfully transferred between these application areas.

CHAPTER 4

Connecting Deep learning to Unity gaming engine

4.1. Introduction

Over the past few years, the computer vision community has enjoyed tremendous popularity in terms of hardware and software upgradations. In particular, faster computing devices, graphical resources and the availability of larger datasets [110] and [18] have enhanced the industry with efficient and robust solutions. Large datasets for training and testing allow the user to manipulate complex machine learning models, such as a deep neural network [35]. Although the success of these models is increasing significantly, such complicated structures suffer from interpretability and high computational demands in terms of space and time to perform tasks such as image annotation or even segmentation. Past studies have shown that it can be much easier if large amounts of image datasets are synthetically generated, annotated and semantically segmented to mimic real-world applications [111], [112] and [113].

Robotics research can greatly benefit from these synthetic 3D objects and scenes where an agent can learn to execute tasks in the real world via robotic simulations. Learning new actions in a simulated world, such as in a virtual world, can reduce excess training time for the robots. Human actions can be recorded to be simulated many times over instead of physically executing tasks. Artificial intelligence (AI) has paved its way to success in fields such as autonomous driving [114], intelligent games, various medical domains, simulating engineering-related tasks, etc. All this is possible because of synthetic or artificially simulated environments like virtual reality [114]. In this regard, the video game industry has developed high-end platforms to help researchers in different fields design synthetic objects and virtual worlds. Among these are Unity 3D and Unreal Engine (UE), which provide many tools to create 3D objects, generate synthetic datasets and develop built-in applications with readily available virtual worlds. Incorporating the power of machine learning models into these gaming engines can greatly reduce the time and effort required to generate results, which is currently not fully available. To bridge the gap between these machine learning models, in particular, deep neural networks and gaming engines, we compiled a Windows-based Caffe framework as a dynamic link library that can be integrated as a plugin into the Unity gaming engine that we named Caffe2Unity. Our compiled plugin can open many doors in the gaming industry since the user is able to fully utilize the features of a deep neural network framework in the Unity environment.

4.2 Related Work

Many robotics simulators use synthetic objects and virtual worlds to mimic real-world scenarios in robotics research and industry to study different design principles [114] and [116]. Executing robotics tasks in a virtual world can provide a powerful interface to study physics dynamics, interactions and rendering capabilities. However, these simulators need to be regularly updated in order to be useful to the computer vision community.

In this regard, Unity and Unreal Engine both have up-to-date built-in scenes and objects that are upgraded and maintained by a large community. There is also a large repository containing many 3D objects, virtual sceneries and backgrounds that are actively maintained within the computer vision community [117], [118], [119] and [120]. Furthermore, there is an interesting open-source tool designed for use in the Unreal Engine that incorporates several computer vision tasks like segmentation, object masking and generation of virtual scenes [121]. Our work, however, integrates a Caffe-based complete deep neural network framework within the Unity gaming engine instead of using it as a separate application programming interface (API). It can be customized for any computer vision task, such as detection, segmentation, classification, etc. Although it is still at an early development stage, we hope it can help many researchers build customizable projects and generate interesting results in the near future.

4.3 Methodology

Convolutional Architecture for Fast Feature Embedding (Caffe) is a deep learning framework developed and maintained by the Berkeley Vision and Learning Center (BVLC). The framework is open-source with a BSD license written in C++ along with its MATLAB and Python wrappers. A Caffe-based CNN consists of two parts:

- The structure, e.g., the layer types that it contains, parameters describing each layer, the order of the layers and how the layers interact with each other.
- The CNN weights: These are the result of the CNN's training and are used to transform the input of each layer to layer's output.

The structure of a layer is described in a *.prototxt file. This file is written in a human-readable format that can be opened by a text editor such as Notepad and reveals the definition of the entire CNN. The weights of a CNN network are contained in a *.caffemodel file, which is not readable. It is loaded after the CNN has been defined as a structure. Normally, the first layer of a CNN is a data layer – this is where the input is stored, usually as an image. Next, each layer takes as input the output of a previous layer and produces its own output. There are two main functionalities associated with these layers: A forward pass that takes inputs and yields the output, and secondly, a backward pass that takes gradients to produce the output. Caffe's framework basically contains different layers, which normally include pooling, convolutional, rectified linear units, softmax, etc. These layers vary depending upon the network architecture being used. The temporal storage between layers is called activation maps or feature maps. They are known as maps because each one is the output of a single neuron or a filter when this filter travels across the layer input. Whereas, all numerical values in caffe are contained in blobs. Blobs are usually a 4-dimensional array which provides memory to hold images or other data. Usually when an image is loaded from the disk it is stored in a blob. This loading of image or data is a synchronized operation of CPU and GPU. A numerical value contained in a CNN blob can be:

- A weight: This remains constant over time and is used to process the input of the CNN.
- A value: It is an intermediate result of processing the input of the CNN. This value is contained in an activation map.

Therefore, numerical values contained in blobs are either weights or activation maps. Schematically, CNN blobs can thus be shown as in Fig. 4.1 below:

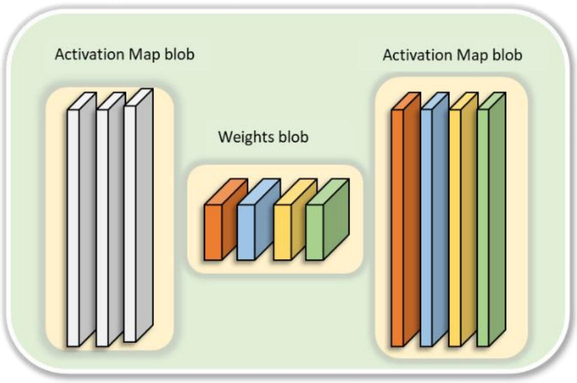


Figure 4.1 : Schematic representation of CNN activation maps blobs, and weight blobs

In the above schema, the 3 grey blocks on the left represent an input block with 3 channels (channels 0, 1, 2). This is a blob with 3 dimensions, namely channel, width and height. The weight blob (which always belongs to a layer) in the middle has 4 dimensions. The first dimension is an ordinal number (orange = 0, blue = 1, yellow = 2, green = 3). The second dimension is where the weight reads from, i.e., the channel of the previous blob (0, 1, 2). The third and fourth dimensions are width and height respectively. Four filters in the layer in the middle produce four activation maps on the right. Each activation map corresponds to one filter. Again, we have a blob of 3 dimensions on the right. Based on the above information, we configured the Caffe framework as a dynamic link library (DLL) for the Unity gaming engine.

4.3.1 Prerequisites of compiling of DLL and composing the network architecture

In order to use the functionality of a deep neural network in the Unity engine, we need to have a configured Caffe framework on Microsoft Windows. The reason for this Windows-based configuration is that Unity is not fully functional or compatible with the Linux operating system. So, we used the following prerequisite versions in order to compile Caffe on Microsoft Windows:

- Microsoft Visual Studio 2015
- CMake
- Git version 2.151.2
- CUDA 8.0
- Python 3.5.4(64 bit)
- NVIDIA CuDNN v5

We used a static compilation of Caffe to generate its *.lib files using Microsoft Visual Studio 2015 and statically linked the lib files to another project, forming a dynamic link library. The overall process is shown in Fig. 4.2.

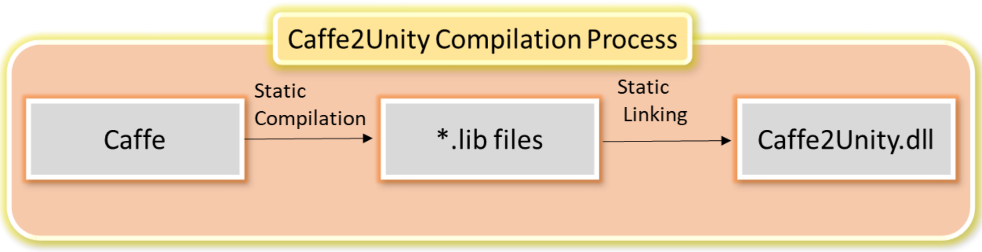
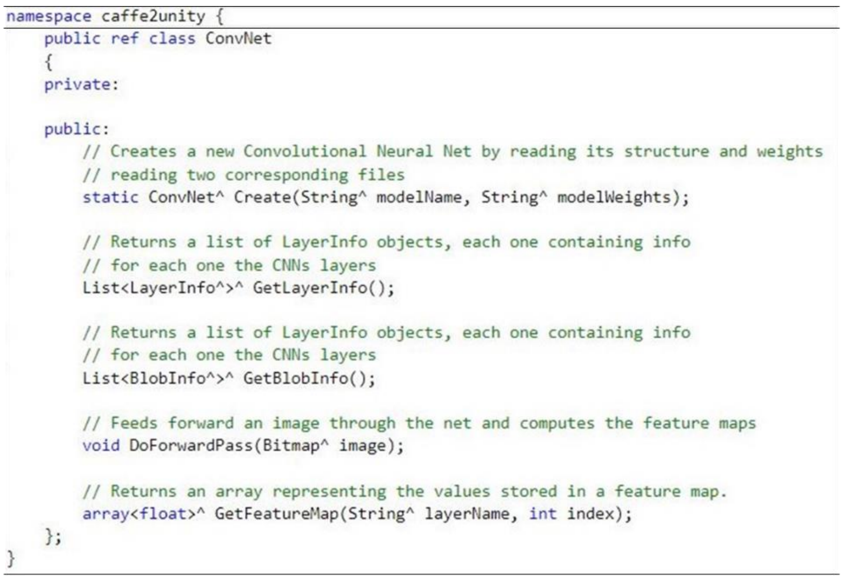


Figure 4.2: Compilation process of Caffe2unity DLL.

After the configuration and compiling process, we chose to test the AlexNet architecture. The reason for its selection is its medium-sized architecture with a total of 11 layers. We chose the following arrangement of all the layers and filters to use the network in a Unity environment. There are a total of 11 layers between input and output, and we listed the dimensions of the layers and filters that were used to design and test our DLL in a Unity environment.

Layer 0: Represents the input image to be passed to the network with a size of $227 \times 227 \times 3$. **Layer 1**: This is the first convolutional layer containing 96 filters of size 11×11, stride = 4 and padding = 0. Its overall size can be estimated as: $55 \times 55 \times 96$, which is calculated as $((227 - 11))/((4 + 1) = 55)$. Here, 96 is the depth (since each set represents 1 filter), and in total we have 96 filters for the first convolutional layer. **Layer 2**: The max-pooling layer, where we down-sample the input received from the previous layer with a 3×3 filter and stride 2, giving a size of $27 \times 27 \times 96$, which is calculated as $((55 - 3))/((2 + 1) = 27)$, representing the size of the outcome. In this layer, the depth remains the same as in the previous layer. **Layer 3:** This is the second convolutional layer containing 256 filters with size 5×5, stride = 1 and padding = 2. Similar to the previous layers, its size can be calculated as $27 \times 27 \times 256$. Since in this layer the padding = 2, we can restore the original size of the image, i.e., $((5 - 1))/(2 = 2)$. **Layer 4:** The next alternate layer is again a max-pooling layer with a 3×3 filter and stride = 2, with size of $13 \times 13 \times 256$, which is calculated as $((27 - 3))/((2 + 1) = 13)$. **Layer 5:** The third convolutional layer contains 384 kernels of size 3×3, stride = 1 and padding = 1. Its size is $13 \times 13 \times 384$, which is calculated as $((3 - 1))/(2 = 1)$. **Layer 6:** It is the fourth convolutional layer containing a depth of 384 filters of size 3×3, stride =1 and padding =1. Its overall size can be obtained as $13 \times 13 \times 384$, which is similar to previous layer. **Layer 7:** This is the last convolutional layer containing a depth of 256 filters of size 3×3, stride =1 and padding =1. Hence, its size can be shown as $13 \times 13 \times 256$, with its original size being restored, it can be calculated as $((3 - 1))/(2 = 1)$. **Layer 8:** The last max-pooling layer of the network contains a kernel of size 3×3 and stride = 2. Here, the overall size is $6 \times 6 \times 256$, which is calculated as $((13 - 3))/((2 + 1) = 6)$. **Layer 9:** The network's first fully connected layer containing 4096 neurons, where $6 \times 6 \times 256$ =9216 pixels serve as input to 4096 neurons, and the weights are determined using back-propagation. **Layer 10**: This is the second fully connected layer with 4096 neurons, which have the same properties as the previous layer. **Layer 11**: The last fully connected layer of the AlexNet architecture contains 1000 neurons corresponding to the 1000 classes of the ImageNet dataset.

We designed the architecture in a virtual environment based on the above calculations and chose to do an in-depth study of image classification as a computer vision task. Although, this DLL can also be used for other customizable tasks such as 3D rendering and modelling, segmentation, etc. To give a general overview of the integrated Caffe framework in a DLL, we give a snapshot of the ConvNet class template, as shown in Fig. 4.3. It basically uses ConvNet objects that contain two lists. The LayerInfos list contains information about the layers and weight blobs contained in each layer, and the BlobInfos list contains information about the activation maps. Later on, we added and customized functionality to interpret and visualize the AlexNet architecture in a real-time scenario (see chapter 5).



```
namespace caffe2unity {
    public ref class ConvNet
    {
    private:

    public:
        // Creates a new Convolutional Neural Net by reading its structure and weights
        // reading two corresponding files
        static ConvNet^ Create(String^ modelName, String^ modelWeights);

        // Returns a list of LayerInfo objects, each one containing info
        // for each one the CNNs layers
        List<LayerInfo^>^ GetLayerInfo();

        // Returns a list of LayerInfo objects, each one containing info
        // for each one the CNNs layers
        List<BlobInfo^>^ GetBlobInfo();

        // Feeds forward an image through the net and computes the feature maps
        void DoForwardPass(Bitmap^ image);

        // Returns an array representing the values stored in a feature map.
        array<float>^ GetFeatureMap(String^ layerName, int index);
    };
}
```

Figure 4.3: Template of ConvNet class showing the main functions to be implemented.

Additionally, we created the Visualizer class with helper functions for creating images from weight and activation blobs to check if the configuration was correct. Shown below in Fig. 4.4 are the four main classes with the attributes and methods used in the DLL.

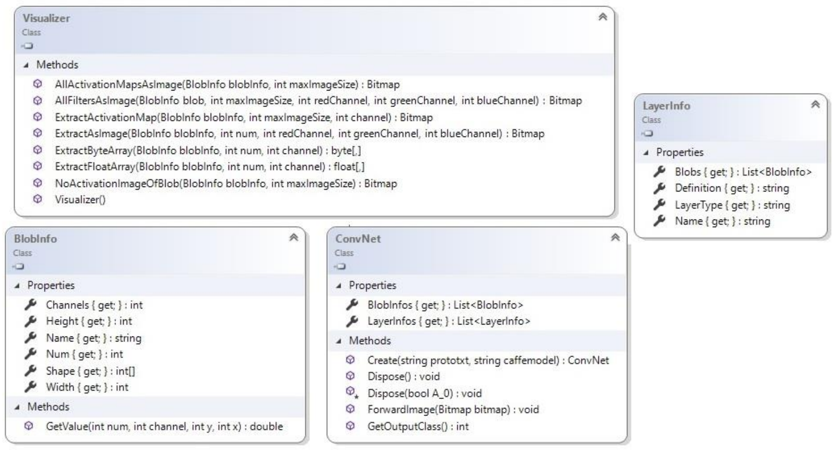

Visualizer
Class
Methods
AllActivationMapsAsImage(BlobInfo blobInfo, int maxImageSize) : Bitmap
AllFiltersAsImage(BlobInfo blob, int maxImageSize, int redChannel, int greenChannel, int blueChannel) : Bitmap
ExtractActivationMap(BlobInfo blobInfo, int maxImageSize, int channel) : Bitmap
ExtractAsImage(BlobInfo blobInfo, int num, int redChannel, int greenChannel, int blueChannel) : Bitmap
ExtractByteArray(BlobInfo blobInfo, int num, int channel) : byte[,]
ExtractFloatArray(BlobInfo blobInfo, int num, int channel) : float[,]
NoActivationImageOfBlob(BlobInfo blobInfo, int maxImageSize) : Bitmap
Visualizer()
LayerInfo
Class
Properties
Blobs { get; } : List<BlobInfo>
Definition { get; } : string
LayerType { get; } : string
Name { get; } : string
BlobInfo
Class
Properties
Channels { get; } : int
Height { get; } : int
Name { get; } : string
Num { get; } : int
Shape { get; } : int[]
Width { get; } : int
Methods
GetValue(int num, int channel, int y, int x) : double
ConvNet
Class
Properties
BlobInfos { get; } : List<BlobInfo>
LayerInfos { get; } : List<LayerInfo>
Methods
Create(string prototxt, string caffemodel) : ConvNet
Dispose() : void
Dispose(bool A_0) : void
ForwardImage(Bitmap bitmap) : void
GetOutputClass() : int

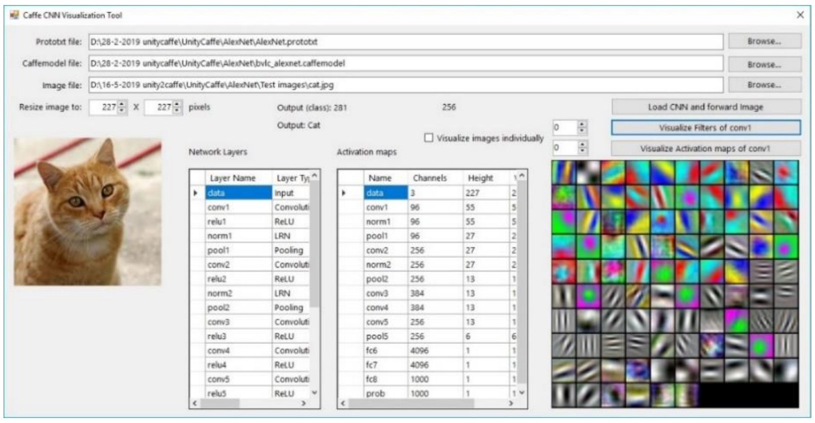
Figure 4.4: Main classes of Caffe2unity DLL with the corresponding methods and attributes used to access the deep neural network

4.4 Results and Application

4.4.1 Perquisites of using the Caffe2Unity plugin in Unity

To check the working of our DLL, we needed the following software and hardware: Windows 10 PC, Nvidia GeForce GTX, Microsoft Visual Studio 2019, SteamVR, Unity version 2019.1.1f1 (64-bit), HTC Vive headset, hand controller and two base stations. Given that the DLL itself is not an executable file, we had a demonstration program TestCaffe2Unity.exe to check its functionality. We followed the below steps to check the functionality of our DLL. After its correct compilation, the resulting interface should look like Fig. 4.5.

- First, select a prototxt file.
- Select a caffemodel file to load weights.
- Select an image to load and forward to the network.
- Press the "Load AlexNet and forward image" button.
- Press "Visualize filters" in order to visualize the filters of the first convolutional layer


Caffe CNN Visualization Tool
Browse...
Browse...
Browse...
Load CNN and forward image
Visualize Filters of conv1
Visualize Activation maps of conv1
Visualize images individually

Prototxt file: D:\28-2-2019 unitycaffe\UnityCaffe\AlexNet\AlexNet.prototxt
Caffemodel file: D:\28-2-2019 unitycaffe\UnityCaffe\AlexNet\bvlc_alexnet.caffemodel
Image file: D:\16-5-2019 unity2caffe\UnityCaffe\AlexNet\Test images\cat.jpg
Resize image to: 227 X 227 pixels
Output (class): 281
Output: Cat
256

Network Layers

Layer Name	Layer Ty
data	Input
conv1	Convoluti
relu1	ReLU
norm1	LRN
pool1	Pooling
conv2	Convoluti
relu2	ReLU
norm2	LRN
pool2	Pooling
conv3	Convoluti
relu3	ReLU
conv4	Convoluti
relu4	ReLU
conv5	Convoluti
relu5	ReLU

Activation maps

Name	Channels	Height
data	3	227
conv1	96	55
norm1	96	55
pool1	96	27
conv2	256	27
norm2	256	27
pool2	256	13
conv3	384	13
conv4	384	13
conv5	256	13
pool5	256	6
fc6	4096	1
fc7	4096	1
fc8	1000	1
prob	1000	1

Figure 4.5: Demonstration program showing the network layers, activation maps and filter outputs of first convolutional layer

This demo confirmed the correct configuration of our framework and can be imported as a plugin into the assets folder in the Unity environment. We used the extern "C" {...} qualifier and enclosed it in brackets to contain a block of function prototypes used in our DLL project to make it accessible for Unity.

4.4.2 Setting up the room and Gear for VR

Before we could visualize our network in the Unity environment, we needed to set up the room for navigation in the virtual reality space. We used the setup as described by SteamVR [122], which is shown in Fig. 4.5.

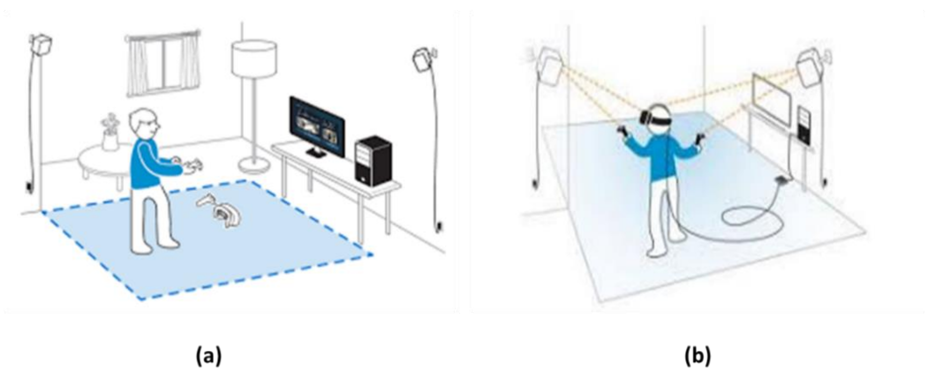


Figure 4.5: (a) Left: Marking the play area shown in blue depending on the available space in the room. (b) Right: The diagonal placement of base station should be in visible range for the headset and hand controllers (fig adopted from [122]).

We used an HTC Vive headset with a room scale tracking feature to enable 360-degree movement in virtual space. The Vive base stations are responsible for tracking the user-initiated movement and interact with the environment using the hand controllers. Fig. 4.6 shows the complete HTC Vive set.

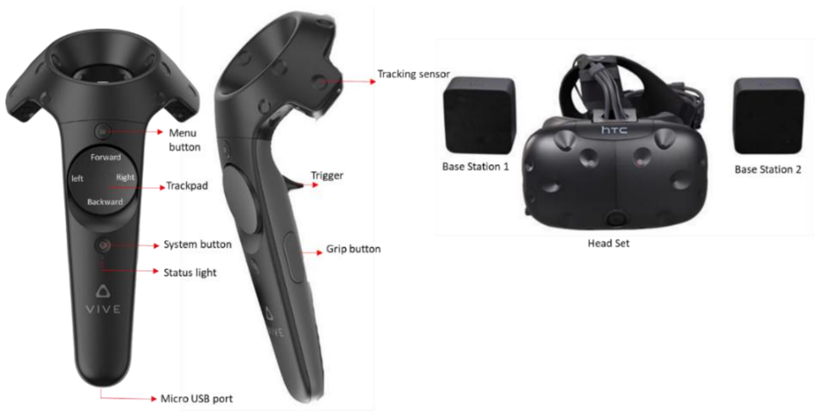


Figure 4.6: HTC VIVE headset, base stations and two hand controller showing the required functionality of its buttons.

4.4.3 Interactivity, navigation controls and topological arrangements

For the initial study, we kept the network model static in the Unity environment, and all changes were implemented before the user entered the VR environment. The arrangement of the network was initially kept fixed. As a preliminary test run of Caffe2Unity in a virtual reality environment, we tested the classes from the ImageNet dataset to see if the classification was correct. We designed the AlexNet architecture using cubes to represent the neurons and filters, and lines to show the connection between the layers. As a visualization strategy, we chose a weight-centric approach that aimed to represent the relationship of filters and layers via their learnable parameters (i.e., weights). But, since the scene became extremely cluttered with millions of parameters, we employed weighted thresholds to minimize the weighted connections for each layer, as shown in Fig. 4.7.

Although modelling the architecture in VR offers freedom to navigate between different layers and visualize filter connections, at the same time, it becomes difficult to analyze the actual working of the model due to cluttered information. In order to overcome this issue, we used weighted thresholds to reduce the number of connections. In convolutional layers, in particular, to visualize useful semantic information, we used weighted threshold connections amongst the layers and assigned color codes to distinguish and remove cluttered information. We considered absolute weight activation values and converted them to percentage ranges. For example, if the WeightValue==0, assign the color yellow; if WeightValue>0 && WeightValue<4, assign blue; if WeightValue>=4 && WeightValue<7, assign red; if WeightValue>=7 && WeightValue<10, assign white; if WeightValue>=10 && WeightValue<20, assign green. This effect can be seen in Fig. 10(b), where all the weight values in range 0 are assigned yellow, but after applying the threshold, the cluttering is reduced.

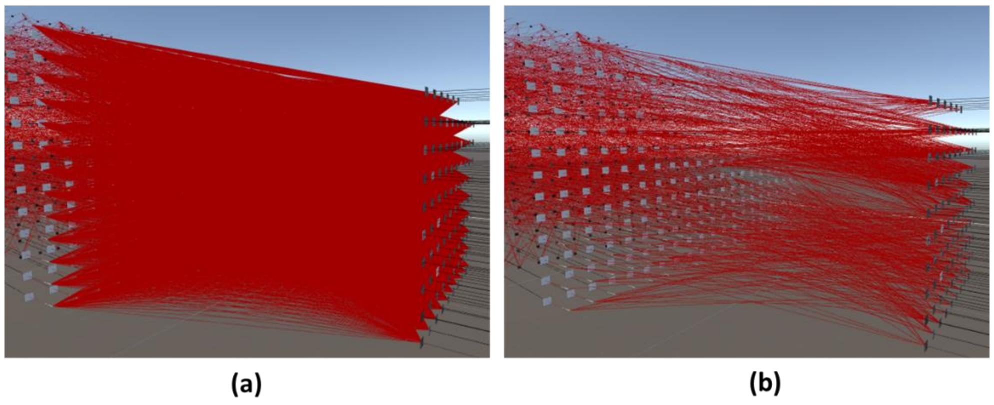


Figure 4.7: Snapshot of AlexNet architecture shown in VR demonstrating weighted threshold approach (a) all the connection are shown giving a cluttered scene (b) after applying a threshold limit to 50% reduction in the cluttered connection.

Once the cluttering is removed, a user can better understand and interact with individual filters and neurons with a laser pointer to see the stored values and size of the filters applied to the neurons to produce activation maps. Furthermore, the user has the freedom to load different trained datasets, change the topology of the network model, adjust the filter space and layer width between the connection layers, regenerate the model according to their desired formation and visualize the corresponding results. In order to steer around the immersive model, the user can use the Vive controllers and teleportation to move throughout the network in the direction the user is looking, i.e., movement is synchronous with the headset's camera. The panel used to modify the network properties and hand controllers' navigation is shown in Fig. 4.9. In addition, we reprogrammed the hand controllers to provide an additional teleportation capability in the virtual environment. We demonstrated the above settings on a smaller network with a limited number of neurons and hidden layers to give flexibility to the VR users and allow topological rearrangement to gain a better understanding of neural networks (see Fig. 4.8). Even though the model's visualization was in VR, most of the interactions were not performed in real-time. Before entering the VR mode, the user has the freedom to configure the required settings of the layers, adjust the weighted threshold values and the settings of the connection lines, set the spacing between the neurons and adjacent layers, etc. However, once in VR mode, the user can move back and forth and up/down in the network as many times as they want. A laser pointer can be used to select the properties of any neuron or filter to see the height/width of the activation maps, its layer name and its current number with the actual value or weight (see Fig. 4.9). Additional visualization results for the input/output layers and the weighted threshold results are shown in Fig. 4.10.

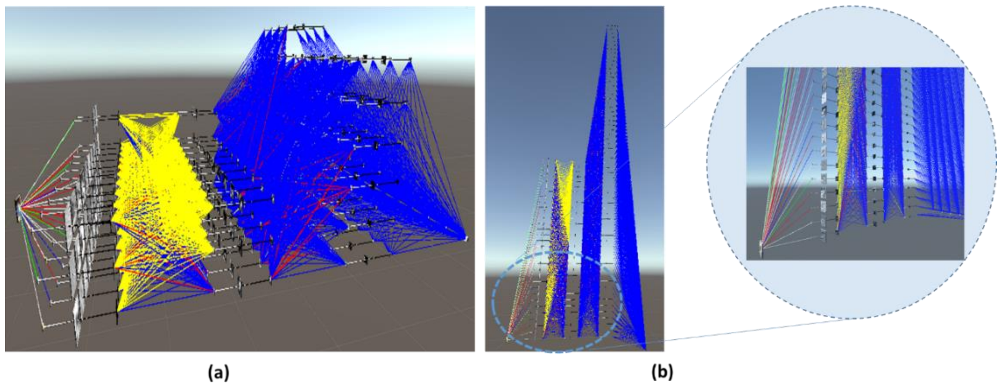
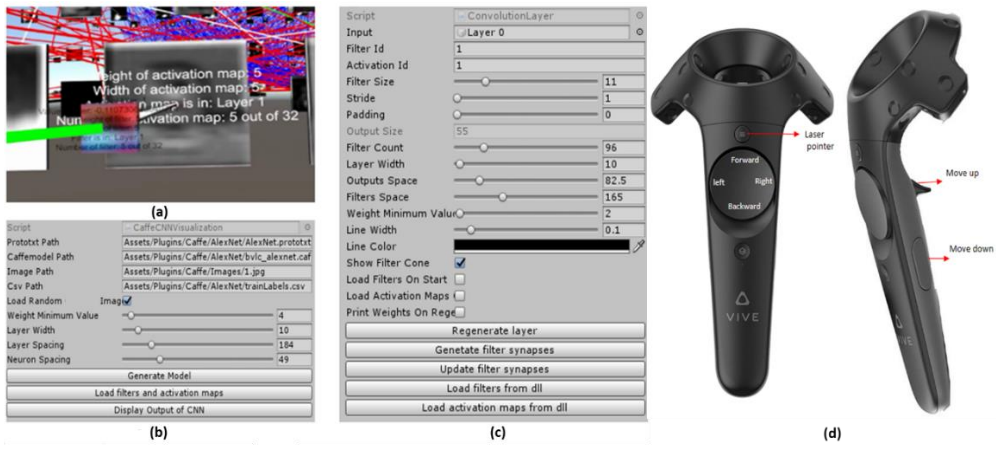


Figure 4.8: Topological arrangement of the network (a) 1 -1 arrangement of network architecture (b) Row and column arrangement of network architecture with zoom in view shown in the circle



Script	CaffeCNNVisualization
Prototxt Path	Assets/Plugins/Caffe/AlexNet/AlexNet.prototxt
Caffemodel Path	Assets/Plugins/Caffe/AlexNet/bvlc_alexnet.caf
Image Path	Assets/Plugins/Caffe/Images/1.jpg
Csv Path	Assets/Plugins/Caffe/AlexNet/trainLabels.csv
Load Random Imag	✓
Weight Minimum Value	4
Layer Width	10
Layer Spacing	184
Neuron Spacing	49

Generate Model
Load filters and activation maps
Display Output of CNN

Script	ConvolutionLayer
Input	Layer 0
Filter Id	1
Activation Id	1
Filter Size	11
Stride	1
Padding	0
Output Size	55
Filter Count	96
Layer Width	10
Outputs Space	82.5
Filters Space	165
Weight Minimum Valu	2
Line Width	0.1
Line Color	⬛
Show Filter Cone	✓
Load Filters On Start	☐
Load Activation Maps	☐
Print Weights On Rege	☐

Regenerate layer
Genetate filter synapses
Update filter synapses
Load filters from dll
Load activation maps from dll

Figure 4.9: Snap shot of the settings panel (a) attributes of a particular neuron (b) Path selection for the network weight, and its layers etc. (c) Different control settings for individual layers which the VR user adjust before entering the VR mode and (d) reprogrammed hand controllers buttons (trackpad for moving up, down right and left, menu button for laser pointer, trigger button for moving up and gripper buttons for moving down in the network.) for navigation and interaction in the VR environment.

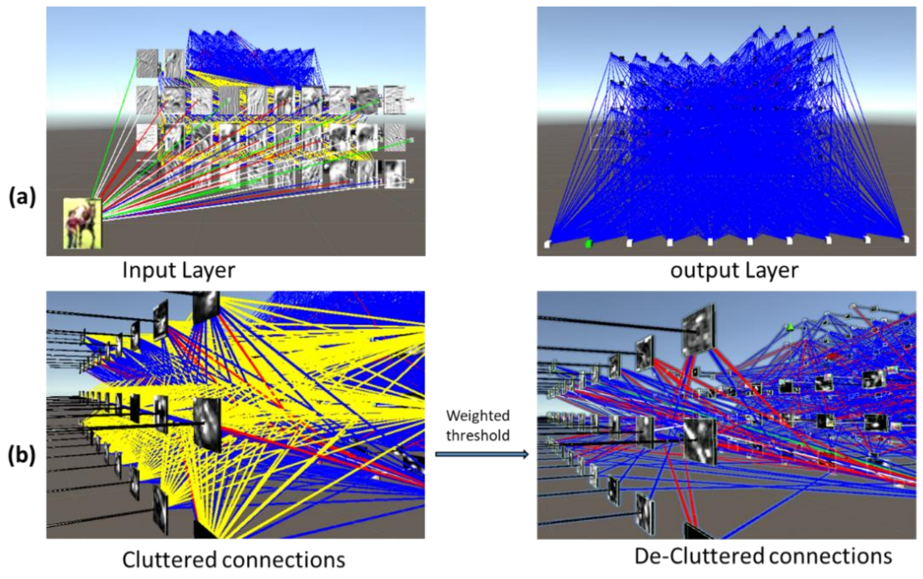


Figure 4.10: (a) showing the visualization of input image and its corresponding activation maps of the first convolutional layer. The colored connection lines show different weighted thresholds, whereas on the left shows the output layer with 10 classes represented as cubes. A green color cube indicates the probability of the correct prediction class. (b) Before and after view of applying weighted threshold in first convolutional layer with weight range set to WeightValue>0 && WeightValue<4 removes the weight range containing yellow color code. As a result all the weights shown in yellow connecting lines were eliminated afterwards giving a decluttered view of the activation maps and the adjacent layers.

4.5 Conclusion

In this work, we developed a dynamic link library for the Caffe framework of a deep neural network, which is used as a plugin in the Unity gaming engine. We provided a demonstration version of the plugin to show how it works in Microsoft Visual Studio, giving the user the opportunity to interact with the plugin and utilize the functionalities of the Caffe framework. In this particular work, we tried to visualize a deep neural network in a virtual reality environment, where most of the VR settings were predefined. The presented model network gives the user the freedom to navigate between the layers and visit all the filters and neurons showing the feature activations, modify the network topology and use thresholds to assign color codes to different weight ranges in the convolutional layers to understand its internal representations. The same Caffe2Unity plugin reappears in the subsequent chapter, where we introduce a modified version of this plugin that made use of the virtual environment, gave the user more freedom regarding interactivity and converted all the functionality in real-time. In addition, we also provided an interpretation module to gain a better understanding of these complex machine learning models.

CHAPTER 5

Caffe2Unity: An Immersive Visualization and Interpretation of Deep Neural Networks

In the previous chapter, we compiled a plugin for window-based version of the Caffe framework. We tested its functionality in a virtual environment using a small network configuration of few layers. The features we delivered in the previous version was purely offline i.e. the user was not able to interact with the immersive model or its features. All the interaction was carried out before the user could enter in the virtual environment to visualize the model. However, in this chapter we expand our work on the plugin and include an interpretation module in the Caffe2Unity.dll and interpret all the interaction and results in real time. This gives more freedom to the user to explore and understand the network model, given the advantages of a virtual environment. We tested our updated version of the DLL with a medium size deep neural network with all the interaction and interpretation in real time.

Details of the work given in the following sections are identical to the published version of the article. However the presentation style is adopted to match the format of the thesis and the references are given at the end as we maintain a single list for the whole thesis. The citation for the article is given as:

Aamir, A.; Tamosiunaite, M.; Wörgötter, F. Caffe2Unity: Immersive Visualization and Interpretation of Deep Neural Networks. *Electronics* 2022, *11*, 83. https://doi.org/10.3390/electronics11010083

Caffe2Unity: Immersive Visualization and Interpretation of Deep Neural Networks

Aisha Aamir [1,*], Minija Tamosiunaite [1,2] and Florentin Wörgötter

Abstract: Deep neural networks (DNNs) dominate many tasks in the computer vision domain, but it is still difficult to understand and interpret the information contained within these networks. To gain better insight into how a network learns and operates, there is a strong need to visualize these complex structures, and this remains an important research direction. In this paper, we address the problem of how the interactive display of DNNs in a virtual reality (VR) setup can be used for general understanding and architectural assessment. We compiled a static library as a plugin for the Caffe framework in the Unity gaming engine. We used routines from this plugin to create and visualize a VR-based AlexNet architecture for an image classification task. Our layered interactive model allows the user to freely navigate back and forth within the network during visual exploration. To make the DNN model even more accessible, the user can select certain connections to understand the activity flow at a particular neuron. Our VR setup also allows users to hide the activation maps/filters or even interactively occlude certain features in an image in real-time. Furthermore, we added an interpretation module and reframed the Shapley values to give a deeper understanding of the different layers. Thus, this novel tool offers more direct access to network structures and results, and its immersive operation is especially instructive for both novices and experts in the field of DNNs.

Index Terms— Deep Neural Networks, Visualizations, Virtual Reality, Unity

5.1 Introduction

A Virtual Reality (VR) environment is a great way to interact with complex scientific visualizations, immense datasets, and complicated 3D structures [123]. Head-mounted displays used in VR have been shown to be a great source of learning in different domains (e.g., complex scientific concepts in astronomy and engineering domains) compared to traditional learning environments [124], [125], [126] and [127]. It is believed that a user can learn and explore multifaceted structures more easily by navigating around them and exploring different characteristics for better understanding [123], [128].

In this regard, DNNs have gained much popularity in the past decade due to the availability of faster computing devices that can minimize their computationally expensive training times. Although these models can process multidimensional forms of data, including audio, video, images, etc., in the computer vision domain, image analysis has been the most prevalent. In addition, visualizations of different machine learning models, particularly DNNs, have also played a significant role in the understanding of input-output processes between successive layers [80], [52]. Until now, DNN visualizations have been restricted to 2D visualization of images, which is not sufficient if we want to visualize the internal representations and interpretations of the network [123], [128] and [127]. Despite their tremendous performance in many computer vision tasks, these models are still considered to be "black-boxes" and require more transparency and interpretation to understand their decision-making process. In this regard, a VR environment can offer many advantageous visualization solutions that can help in understanding deep learning models. Earlier works regarding visualizations have particularly focused on expert users with an in-depth understanding of DNNs and were specially tailored for developers in this field [7], [9], and [103], [80] and [26].

In this paper, our primary goal is to provide a general understanding and architectural assessment of DNNs in a VR environment while targeting both non-experts and experts as our audience. We focused on the immersive visualization and interpretation of DNNs and provided a virtual walk through the network layers exploring different image activation features and connections across different layers. Additionally, an interpretation module was used to analyze the network's decisions by highlighting the positive and negative contributions of different regions of the test input images. Our main contributions in this paper are as follows:

- We compiled a Windows-based Caffe framework as a dynamic link library to be used as a plugin in the Unity gaming engine.
- We designed a simplified, layered GUI for analyzing the AlexNet architecture [35] for novices and experts in the field.
- We developed an interactive real-time module for adding occlusion on the input images to understand the behavior of the network.
- We provided an interpretation module to reframe the Shapley values algorithm [130] for analyzing the network's decisions based on three hidden layers.

5.2 Related Work

In order to gain an intuitive understanding of the different features and responses of individual layers, researchers have developed tools and libraries to visualize the neural activations in different layers. In this regard, visualizations can be divided into two categories, namely static 2D image-based visualizations, and browser- and VR-based interactive visualizations, for which a brief description is given in the following sub-sections.

5.2.1 Static 2D image-based visualizations

In this category, much work has been carried out on the 2D visualization of images. In [80] and [3], interactive software tools were used for understanding, and an exploratory analysis was done on how we can visualize the activations produced at each layer in response to user input. In [56], Layer-wise Relevance Propagation (LRP) based heatmaps were used for region perturbation and highlighting the important parts in an image during the classification task. While in [52] and [86], it was demonstrated that a single neuron can represent multiple facets and be visualized via a synthetically generated image, highlighting the specific areas in natural images that best activate the neurons. In [47], different visualization techniques based on natural images were studied, involving activation maximization in identifying and highlighting visual patterns. In [82], the visualization of intermediate layers was studied during an image classification task, showing properties such as discriminability, compositionality and invariance among the features as the higher layers were approached. In [46], an analysis of the information contained in an image was performed by the process of inversion using an optimized gradient descent function. Visualizations were used to show the representations at each convolutional layer while maintaining the photographically realistic representations of the images.

5.2.2 Web browser and VR - based interactive visualizations

Other interactive visualization techniques include web-based representations based on the TensorflowJS library called ConvNetJS ((http://cs.stanford.edu/people/karpathy/convnetjs/ (accessed on 08.04.2020))) that allows for training the network in the browser. In [10], the authors presented a real-time interactive tool for convolutional neural networks. In addition, TensorBoard (https://www.tensorflow.org/versions/r0.8/how_tos/graph_viz/index.html(accessed on 08.04.2020)) (Google's TensorFlow library) and TensorFlow Playground (http://playground.tensorflow.org/(accessed on 08.04.2020)) also provide web-based interfaces that give the user the freedom to interact with and visualize detailed neural network configurations. NVIDIA has also implemented a web-based deep learning library called DIGITS (https://developer.nvidia.com/digits (accessed on 08.04.2020)) which allows users to design, train, monitor, and visualize neural network models in a browser. The method developed by [83] allows the user to pick and choose certain data and visualize their highest activations in the subsequent layers. In [26], a 3D interactive convolutional network was designed to allow users to draw images and simultaneously visualize the trained network. These methods are intuitive but lack the freedom of navigation, restricting the user's movements with only mouse and keyboard interactions.

Despite having outstanding tools and methodologies, none of the visualization methods described above provide the user with the freedom to engage in a virtual reality environment. Few studies exist that deal with the visualization of CNNs in the VR domain. The work by [26] and [176] visualized a simplistic model that gives novices in this field a sense of the basic operations that occur in the different hidden layers. Investigations by [129] developed an immersive node-link diagrams-based tool to visualize the internal representations of CNNs. The work by [177] gave a VR model of DNN, giving the user freedom to move layers and objects in tangible manner. In [178] authors provide interactive tool with various levels of details to explore DNN model, whereas in [179] hand gestures based interactive VR interface using DNNs is described using Tensorflow as a deep learning framework.

In our paper, we present immersive visualization and interpretation of CNNs, while focusing on an image classification task using the AlexNet architecture. Our study involves the configuration of Caffe on the .Net framework using Microsoft Visual Studio to provide a plugin for the "Unity" gaming engine. The next sections provide a detailed description of this "Caffe2Unity" plugin and its application for visualizing and interpreting the AlexNet architecture on an image classification task.

Methods

A Windows-based Caffe framework was generated in the form of a dynamic link library (DLL) to create a model of the CNN in the gaming engine Unity, details of which are presented next.

5.3.1 Compilation Process of the Caffe2Unity.dll

In order to utilize the Caffe framework and its associated functionality to bridge the gap between deep learning and a gaming engine, the Caffe2Unity.dll was developed. Caffe was initially configured on a Windows platform and compiled to create a static library producing *.lib files to make the code platform and application independent. The compilation process was carried out using Microsoft Visual Studio 2015 and CMake to obtain the relevant make files for the compiler environment. Other prerequisites in the compilation process were Git (version 2.151.2), CUDA (version 8.0), Python (version 3.5.4), and NVIDIA CuDNN (version 5). Once the Caffe framework was compiled and *.lib files were created, another C++ mixed-mode DLL project was developed to link statically to the static libraries of Caffe. To use routines from the Caffe static library, a reference was added in the C++ mixed-mode DLL project to utilize the *.lib files of Caffe. For execution of the Caffe2Unity.dll, static library files of Caffe were selected as a default project. To test the current Caffe2Unity.dll, we used a pre-trained ImageNet dataset on the AlexNet architecture but to train a different network architecture with different dataset, levelDB or Lmdb format and caffe.exe is required. Figure 5.1 shows the conceptualization of the visualization strategy.

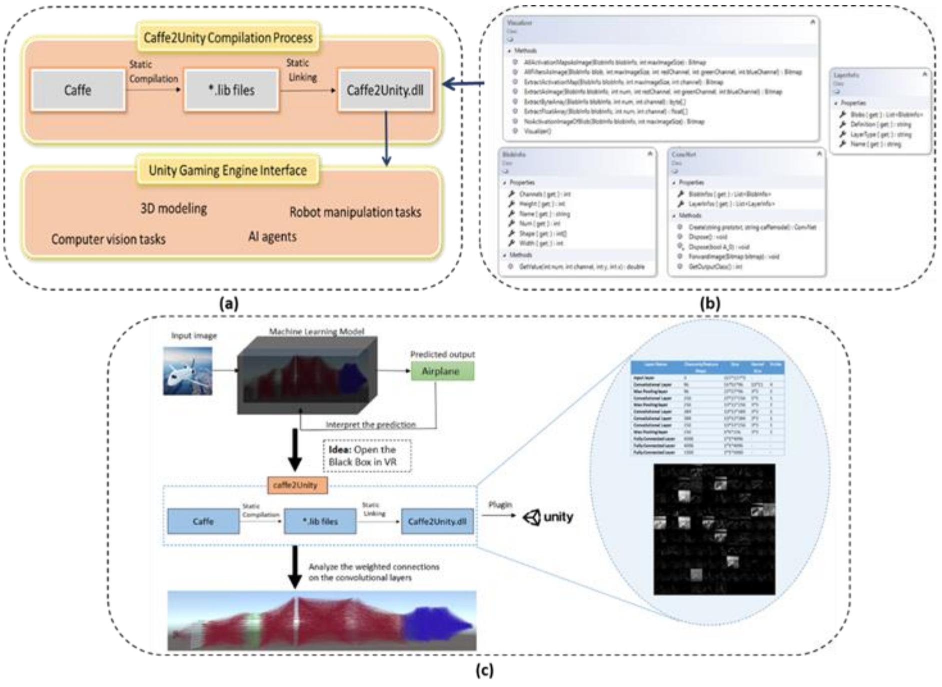


Figure. 5.1: (**a**) Flow chart of the compilation process of the Caffe2Unity.dll and its interfacing with Unity to perform various tasks. In its current form, we used the DLL for image classification as an application domain. (**b**) Prototypes and attributes of the main classes used in the Caffe2Unity.dll to achieve our goal. (**c**) Snapshot of the visualization strategy illustrating the immersive model of AlexNet with details of all layers and activation maps that are used in the VR environment.

The compiled DLL was used as a plugin in the asset folder of the Unity project where routines were called, and the immersive model of deep network was constructed. To give a general overview of the integrated Caffe framework in a DLL, we basically use ConvNet objects that contain two lists (Figure. 5.1b). The LayerInfos list contains information about the layers and weight blobs contained in each layer, and the BlobInfos list contains information about the activation maps. Additionally, we created the Visualizer class with helper functions for creating images from weight and activation blobs to check the correct configuration of the DLL.

After the configuration and compiling process, we chose to test the AlexNet architecture. The reason for its selection is its medium-sized architecture with a total of 11 layers. We made the following assumptions to calculate the size of the output and filters objects designed in Unity. Suppose we have an image size of $N \times N$ and a filter size of $F \times F$ for convolutional layer, P_s for filters of pooling layer and stride S, we can calculate the output size of the convolutional and pooling layers respectively using the expressions: $output\_size_{conv} = \frac{(N-F)}{S+1}$ and $output\_size_{\max\_pool} = \frac{(N-P_s)}{S+1}$.

Given that the DLL itself is not an executable file, we had a demonstration program TestCaffe2Unity.exe to check its functionality. The successful execution of the demo confirmed the correct configuration of our framework and was imported as a plugin into the assets folder in the Unity environment (see Figure 5.1c). We used the extern "C" {...} qualifier and enclosed it in brackets to contain a block of function prototypes used in our DLL project to make it accessible for Unity. The function prototypes used are given in the above Figure 5.1b. Once a function is added to the DLL project, it can be defined in the Caffe2Unity.cs file in Unity just like any normal function definition. One interesting feature is that any model trained on the Caffe framework can be used in the DLL and its functionality is independent of the different versions of Unity. In the current settings, the layers of AlexNet are hard-coded to avoid crashes but for customization, a single line of code can add or remove layers, thereby altering the network using a "for loop" $\mathrm{for(int\ i = 0;\ i < 9;/* data.layers.Count - 1 */; i++)}$. It is currently set to 9 layers + 1 output layer. The following section provides details of the neural network model in the Unity gaming engine.

5.3.2 Immersive model, Interactivity, and Navigation Controls

A deep neural network is a hierarchical representation of linear and non-linear transformations between layers. Here, convolutional layers play a key role when applied to input images to produce the corresponding feature maps indicating the learned local features. In this paper, we used a standard convolutional AlexNet model developed in [35] which consists of five convolutional and three fully connected layers. We used image classification as a standard task for this network and for our visualization method. Other classification tasks could be considered, too, but their visualization might be less straightforward. In the current setup, training of CNNs is not part of the visualization procedures. Thus, we work with a pre-trained AlexNet architecture. Deeper network models pre-trained with Caffe framework can also be used in the DLL as the functionality and basic operations remain almost the same. However, since our goal was to interpret, visualize, and give a general understanding of a DNN in a VR environment, the medium-sized AlexNet was an appropriate choice. Moreover, using AlexNet for the goal of assessing and understanding the network features in real time is more efficient with such a medium-sized network as compared to dealing with deeper and more recent networks.

To provide freedom of navigation in the VR environment, the user interacts with a virtual graphical interface and first selects a desired path within the architecture given as a *.prototxt file. This file is in a human-readable format and describes the structure of the CNN in terms of model parameters, layer types, their order, and how these layers interact with each other. Next, the user selects the *. Caffemodel file, which is not human-readable and contains the weights of the trained CNN. This model file is loaded after the structure of the CNN has been defined. Finally, the user selects the output-label file and an input image for forwarding to the network. We introduced a very simple user-friendly GUI in the VR environment so that a user can select the model architecture files from a directory, load the output image-labels, and choose an input image for the network. We also provide an option to edit the image and add an occlusion block of any size on the input images to see how this affects the network in real time. To draw such an occlusion block, the user can press and drag the VR-controller's trigger button and release the trigger at the end. Both hand-controllers can alternatively be used for navigation in the network, the trackpad on the right-hand controller with laser pointer is used to move forward, backward, left, and right, whereas the left-hand controller allows the user to move up and down in the network. The VR-imagery from the head mounted device is real-time synchronous with the movements of the user. Modeling the architecture in the VR environment gives users the opportunity to navigate within different layers and to visualize filters and connections as often as wished. Hence, the user is truly immersed in the network architecture not being restricted to fixed operations of mouse or keyboard as in non-VR systems. He or she can easily use the hand controllers for "teleportation", this way moving through the network in all directions with real-time visualization by the headset.

We created a layered VR-representation architecture to keep it as close to the network's theoretical description and to—this way—allow for a more intuitive understanding because it allows step-by-step access of the structure. This way one can obtain a more detailed intuition for the operation of the different layers as compared to simple non-VR-based visualizations. An additional advantage of this layered structure is that it also makes it easier to interpret neural-specific operations in case of image occlusions. In addition to this, the user can select to highlight specific connections to understand the activity flow from the input data passing a particular filter with its corresponding feature maps which is then forwarded to the deeper layers of the network. Aside from moving in the forward direction to view the deeper layers, the user can also move to previous layers at any time during visual exploration, which is usually not possible without a VR setup. At the output layer of the network, users can visualize the top five network classification results with their colored legend shown next to the output layer icon in a descending order of relevance.

5.4 Results and Discussion

5.4.1 Visualizations, Interpretation, and Analysis of the Network Decisions

The immersive AlexNet architecture and its features are presented in Figure 5.2, which was selected as an example case; however, all methods can also be applied to other

DCNNs. In addition, we also provide a graphical user interaction video alongside a step-by-step guide to using the application as a separate Supplementary Materials. Analysis and interpretation of classification results is important as it helps to understand the decisions of the network, because at times even a perfectly trained network model misclassifies an image. Thus, finding the reasons for network decisions can help in improving model accuracy, thereby making it more trustworthy.

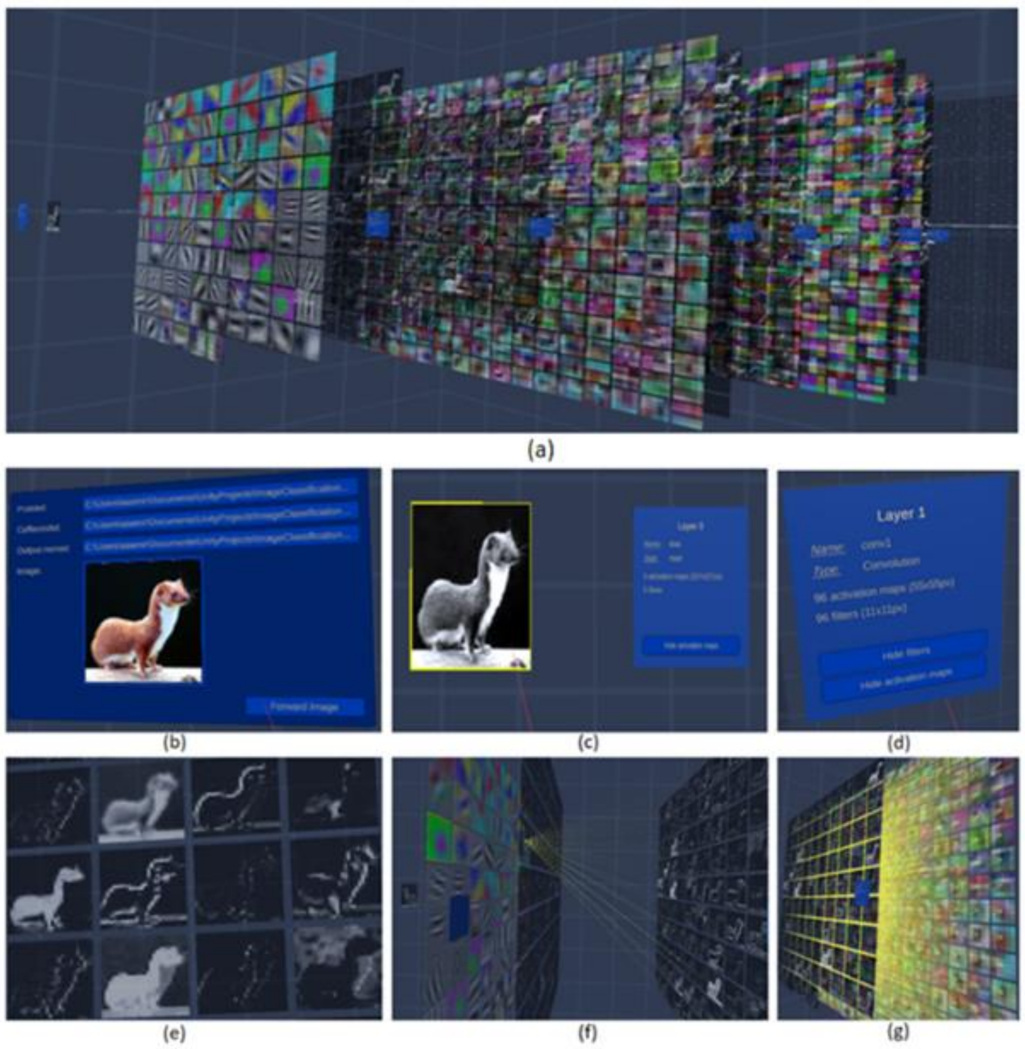


Figure 5.2. (**a**) An immersive visualization of different layers of the AlexNet architecture in Unity, (**b**) virtual GUI for selecting paths for model weights, CNN structure, labels, and input image, (**c**) input layer, (**d**) first convolutional layer icon, (**e**) snapshot of activation maps of the 1st convolutional layer, (**f**) selecting the connection of a particular filter of the 1st convolutional layer to visualize its activity across

the network, and (**g**) deeper layer interactions of the selected filter. The user can visualize and hide filters and activation maps by selecting the corresponding option on the layer icon using a laser pointer.

In the current paper, we re-framed the use of Shapley values [130] to find those regions in an image which play an important role in the model's decision. Our layered structure enables us to go to individual layers to identify the highest activation regions that contribute in the Shapley value-based evaluation. The size of the individual features increases as we move to the deeper layers and the Shapley values grow because in the deeper layer, the network gains more confidence about its predictions. This kind of visualization is only possible in a VR environment. An example of the Shapley value results for a weasel as input image is shown in the VR environment in Figure 5.3. The red color indicates positive contributions to the network's decision of these image regions while blue stands for negative contributions, which decrease the likelihood for a certain decision. White shows that these pixels have no effect on the decision of the network and occluding those regions will not change the network output. Shapley value-based interpretations are shown for three convolutional layers of the network. The top five potential classification results for the weasel image were "weasel", "mink", "polecat", "black-footed ferret", and "mongoose", with highest probability of being a "weasel". Figure 5.4 shows a zoomed-in view of Shapley-value results given in the VR environment for the weasel picture representing a "weasel" (left), and for the alternative four classification possibilities. For all five potential recognition hypotheses, red areas are concentrated around the face region of the images, which are, thus, considered as the most important regions of interest for the network.

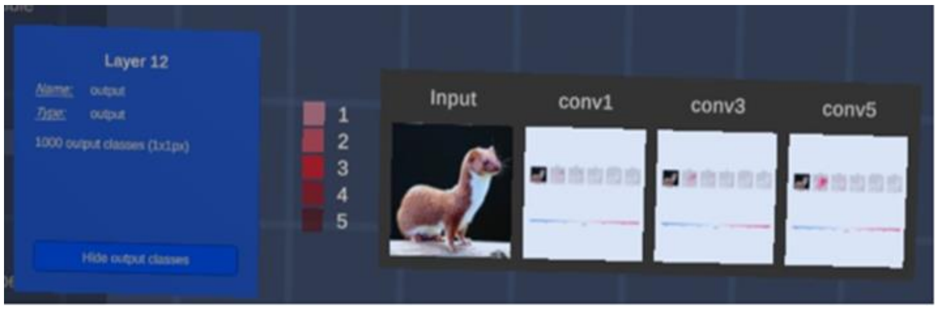


Figure. 5.3: Analyzing the network decision using Shapley-value-based interpretations in the VR

environment. These interpretations are shown for three hidden layers namely conv1, conv3, and conv5, respectively (see also Figure 5.4 for zoomed in view).

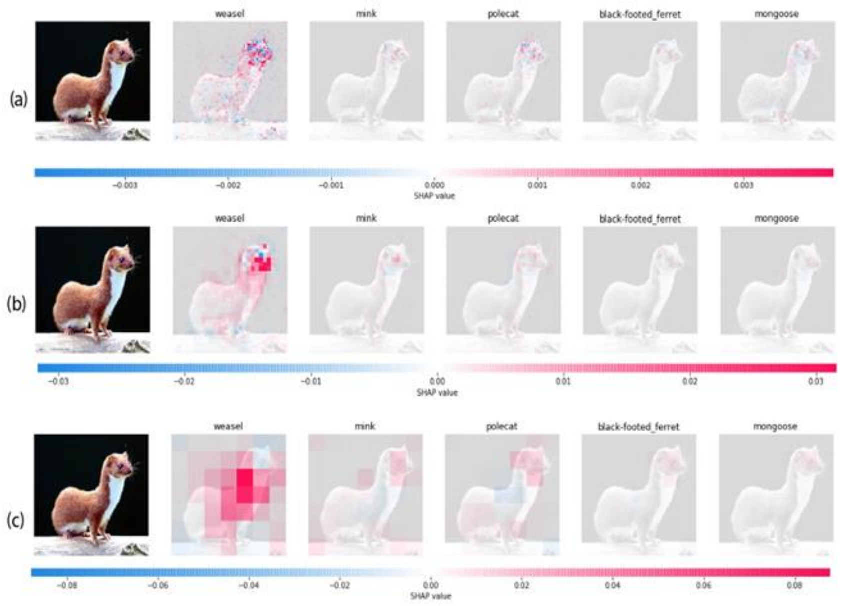


Figure. 5.4: Zoomed in view of VR-based Shapley value results from Figure 5.3 showing individual interpretation of three hidden layers, (**a**) conv1, (**b**) conv3, and (**c**) conv5 of the weasel image.

However, in general, for this image, evidence remains far lower for all four 'wrong' classification hypotheses than for weasel. As we move to the deeper layers, features increase in size and become more abstract, finally providing the model confidence to make its decision as shown in Figure 5.4. To further enhance the usefulness of our VR system, the user can also add an occlusion block on the input images at random locations to hide part of the image. This is simply a black rectangle overlaid on the image using the laser pointer as shown in Figure 5.5. The user has the option to select an image of his/her choice from the sample image folder, add an occluded block of any size at any random location in the image, and save the image so it can be forward to the network. The classification results can then be interpreted using the Shapley values to see the effect of different image regions responsible for the final decision.
As an example, we calculated the Shapley values for the weasel image by occluding the part that most strongly contributes to the network decision as shown above. The effect of the occluded block is evident in the activation of certain neurons and the feature maps that contribute to the final decision. The network now concludes that this

is a "guinea pig". Shapley value-based interpretation of the partially occluded weasel image can explain this decision (Figure 5.5).

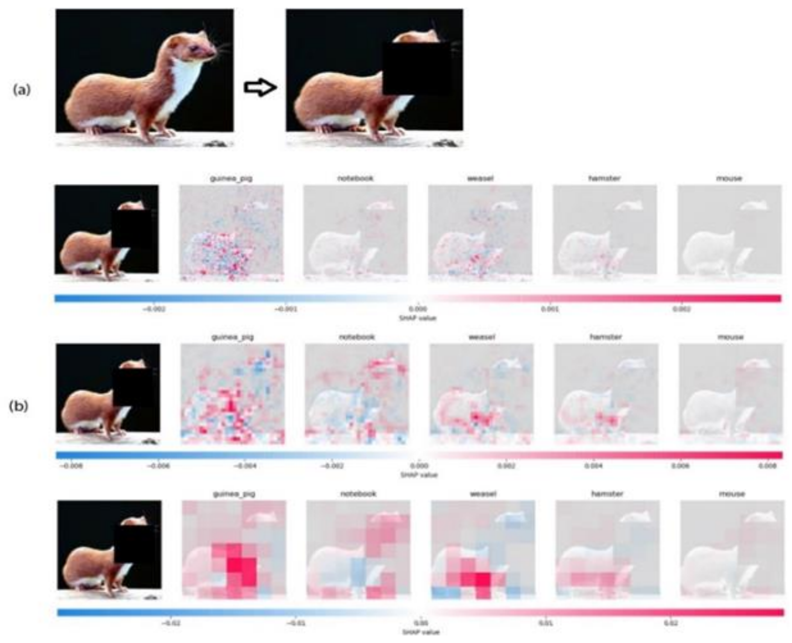


Figure 5.5. Zoomed in view of VR-based results, (**a**) original image on the left whereas the right image shows the overlaid block occlusion, and (**b**) Shapley-value-based interpretation of the weasel image after being occluded by the VR user. Results are shown for three hidden convolutional layers, where the network strongly predicted the image as being a "guinea pig" instead of a "weasel".

From the remaining image, the network now considers the front of the animal body as most important where "guinea pig" and "weasel" yield high positive values, but a higher one for "guinea pig". Interestingly, a black occlusion is (necessarily) also interpretable as part of an image and the network suggests that this part may be indicative of the image being a "notebook" with a total score even higher than for "weasel". This occlusion-based method allows us to analyze network interpretations not only from the correct classification results but also from a such-distorted image, exploring the individual neurons and layers in the architecture.

5.5 Conclusions

In this paper, we developed a dynamic link library for the Caffe framework of deep neural networks, which was used as a plugin for the Unity gaming engine. Users can interact with the plugin and utilize the functionalities of the Caffe framework. We used this plugin for immersive visualization of the AlexNet architecture with its application in image classification. The interactive immersive neural network model not only offers the freedom of navigation within the network layers but also provides the option to select certain connections to examine the activity flow of particular neurons. The novel aspect of integrating a Caffe framework with a gaming engine allows for using our DLL and the user can visualize the functioning of a deep neural network in a gaming engine without the need for additional programming dependencies. Initially, we used it for image classification tasks. As a future work, one could instead perform other tasks like detection, segmentation, image retrieval, design intelligent games, or other application specific solutions. In general, to get deeper insights as well to understand the learnt representations, this kind of visualization is easy to explore and debug given the design of our model. Our simple layered design is especially well-suited for novices to better understand network function.

To dig deeper into a network's internal representation, we analyzed and interpreted the network decision using Shapley values to evaluate the regions of interest with and without occluding the input image. We observed a massive shift in the output decision and of the activation maps of the network when we added an occlusion block on the most relevant image features. Note that this is a real-time feature, which is a novel aspect of this work. A user can, in the VR setup, go back and forth many times to compare and visualize the most influential parts of the image features as given by the interpretation module. Without the VR setup, going back to visualize the hidden layers as well the shapely results in parallel is far more difficult. On the other hand, no change in the network decision was seen by occluding less important image regions as calculated by Shapley values (shown in Supplementary Materials). Thus, the interpretation results obtained by the Shapley values provide visual evidence of how different convolutional layers weigh important features in an input image.

In summary, we think that one can better assess and interpret DNN models via immersive exploration of different layers and activation maps rather than by using non-VR based visualization. Being able to visualize the flow from the input image into the network and the changes it undergoes is beneficial for getting a deeper understanding of these complex structures and allows us to some extent to interpret the network's decision process. We have shown through Shapley values how important the regions of interest are for the network to make its decision. Using such observations, it may be possible to improve the model's decision by retraining it with different sets of hybrid features associated with the interpretation of decision processes. This could be done, for example, by considering the importance of contextual information and how this affects the classification output, which will be one focus of our future work.

Additional aspects for future work could be to try to arrive at a more formal quantification of the different aspects shown here. The visualization of network function shown here is mainly targeted at human viewers and there is no generally agreed baseline how to assess the quality of systems like ours and formal criteria cannot easily be established. Hence, to address the aspect of usefulness for a user, possibly the best way would be to perform user studies with people who have different backgrounds in understanding network operation and function (e.g., from a novice to an expert). The second aspect shown in this study concerns the visualization of network-decision regions via Shapley values. This lends itself to a more rigorous quality assessment and we are currently performing an assessment of the influence of different aspects in images on a network's decision. This assessment, however, is very detailed and would exceed the scope of the current study.

5.6 Supplementary Material

5.6.1 Installations

As an initial step, install the Unity software from the download section on the official website https://unity3d.com/get-unity/download. Also install the software relevant for your virtual reality hardware device, in our case we installed SteamVR and the appropriate software for the HTC Vive. We followed the step by step installation guide for the HTC Vive given on the official website https://support.steampowered.com/steamvr/HTC_Vive/. All the installations are done according to the Windows operating system and specifications of the computer used.

5.6.2 How to use the application

After all these software installations, copy the Caffe2Unity.dll in the plugin folder of the Unity project. The step-by-step process for using this application is described below.

Step1: Set the path for relevant files i.e. prototxt file, caffemodel file, output labels file and images to forward to the network. All these files are placed in the folder named "SampleData\". We have used the images from ImageNet dataset with a .png extension. We used a customized binding to adjust the functionality of the HTC Vive hand controllers. The "North" play mode of the Trackpad of one controller is used to move to the "next_layer" and the "south mode" is used to move to the "previous_layer" in the network. The trigger button is used to interact with the user interface. The controller, which has a laser pointer, is used for "pointing" and "selecting" an object using the trigger button and also for moving back and forth in the network. The trackpad of the controller, without the laser pointer, is used to move up, down, left and right in the virtual environment. The hand controllers and blank interface of our application are shown below:

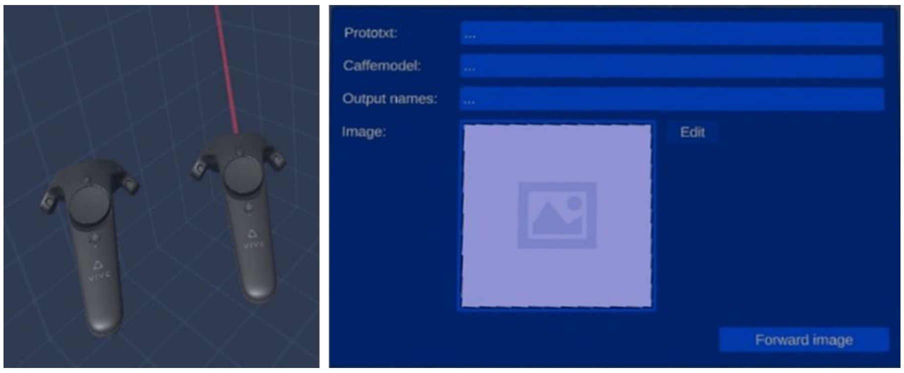
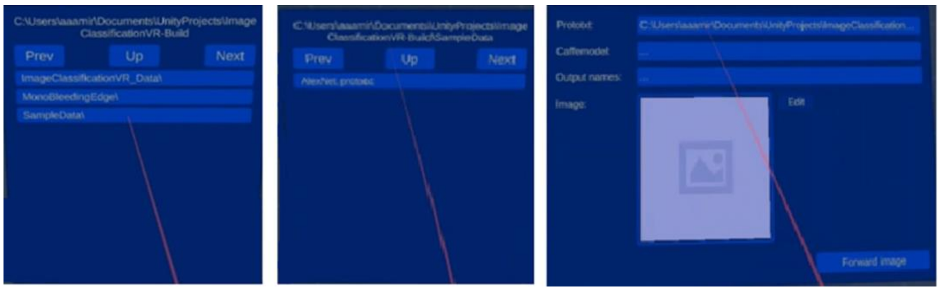


Fig. 1: Hand controllers and GUI of the application.

Step2: Select the necessary paths as shown below, press the "Forward Image" button to pass the image to the neural network.


C:\Users\aaamir\Documents\UnityProjects\Image ClassificationVR-Build

ImageClassificationVR_Data\
MonoBleedingEdge\
SampleData\

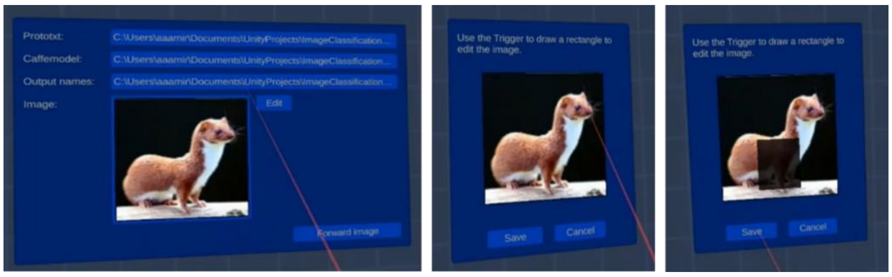
C:\Users\aaamir\Documents\UnityProjects\Image ClassificationVR-Build\SampleData

AlexNet.prototxt

Prototxt: C:\Users\aaamir\Documents\UnityProjects\ImageClassification...
Caffemodel: ...
Output names: ...

Fig. 2: Selecting relevant paths for prototxt file, caffemodel and output label files using the laser pointer.

Alternatively, the user can press the "Edit" button to add a block occlusion on the image and then forward it to the network, as shown below.

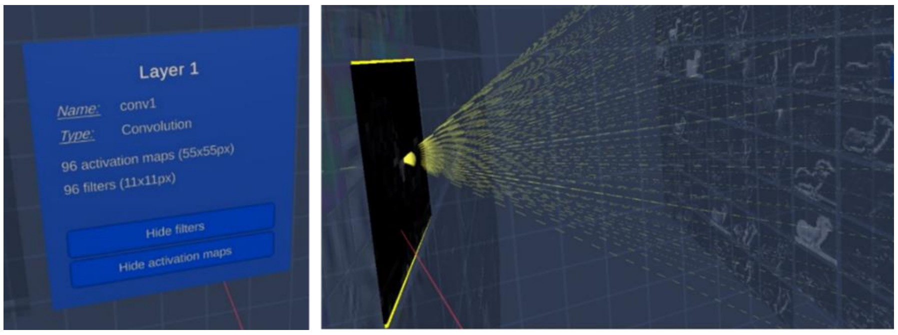

Prototxt: C:\Users\aaamir\Documents\UnityProjects\ImageClassification...
Caffemodel: C:\Users\aaamir\Documents\UnityProjects\ImageClassification...
Output names: C:\Users\aaamir\Documents\UnityProjects\ImageClassification...

Use the Trigger to draw a rectangle to edit the image.

Use the Trigger to draw a rectangle to edit the image.

Fig. 3: Input image selection and pressing the Edit button to add an occlusion block on the image.

Step 3: After the image is loaded and forwarded to the network, the VR user can point with a laser to select and visualize or hide the activation maps/filters using the trigger button of the controller. This is shown below. The user can select a specific filter using the laser pointer to see its related connections (depicted below).



Layer 1

Name: conv1

Type: Convolution

96 activation maps (55x55px)

96 filters (11x11px)

Fig. 4: Convolutional layer 1 icon and selecting a specific activation map to see the activity flow along the layers.

In this way, the user can explore all the layers until the output layer is reached, which shows the top five prediction results of the input, as depicted below.

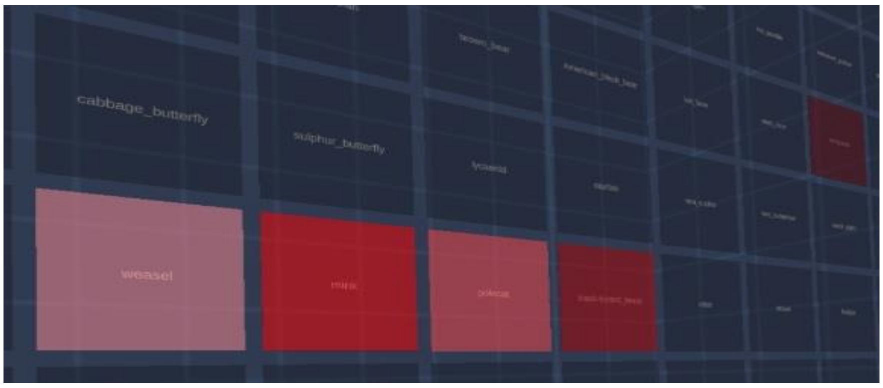
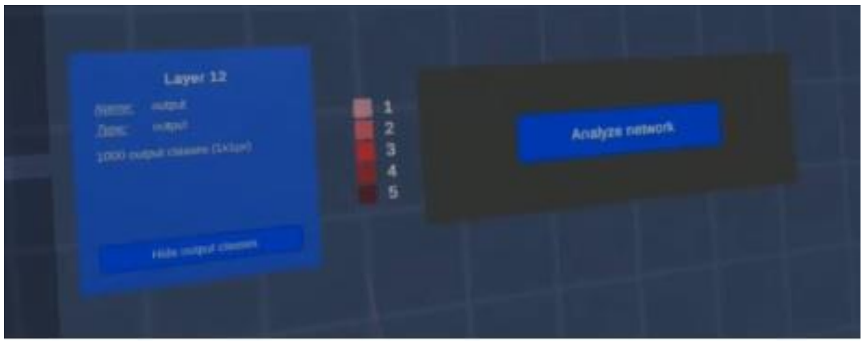


Fig. 5: Sample output layer showing the top five prediction results.

Step 4: As the last layer, the user can also analyze the network by pointing the laser on "Analyze network", and use the trigger button from the hand controller to select that option. Upon clicking this button, you will see the label on the button "Analyzing network", and depending on the processing power of the computer, you will be able to get the Shapley values results as an interpretation of the network decision within few minutes of delay.



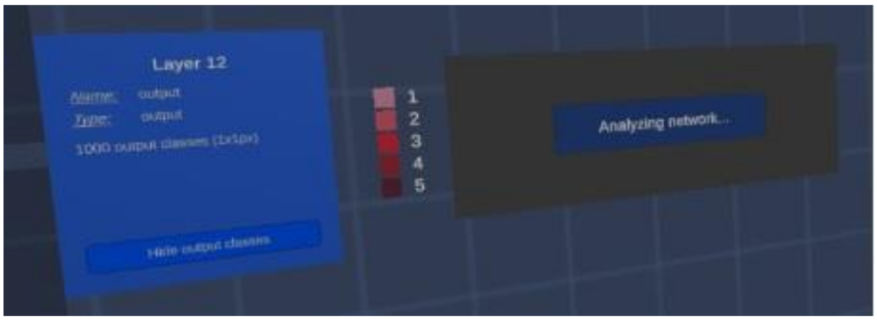


Fig. 6: Output layer icon, and legend for top five prediction results (1: highest probability) and icon to Analyze the network predictions for Shapley values results.

After processing few minutes, you will get the results as shown below.

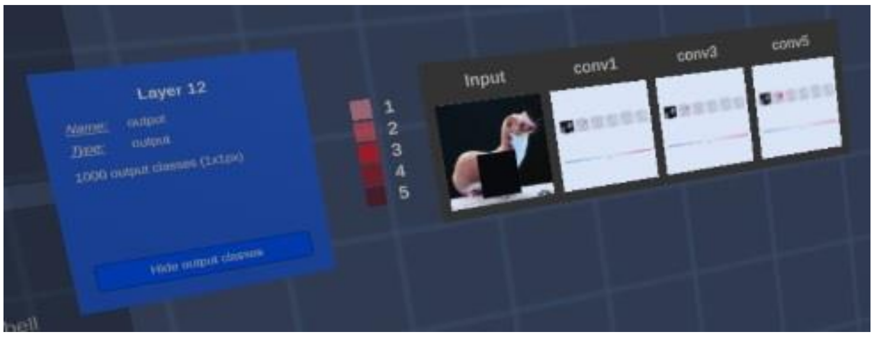


Fig. 7: Shapley values results shown in VR for three convolutional layers.

5.5.3. Additional Visualization Results

Some more results on a particular neuron selection and a 'zoomed in' view for Shapley values for occlusion and misclassification results are also shown below:

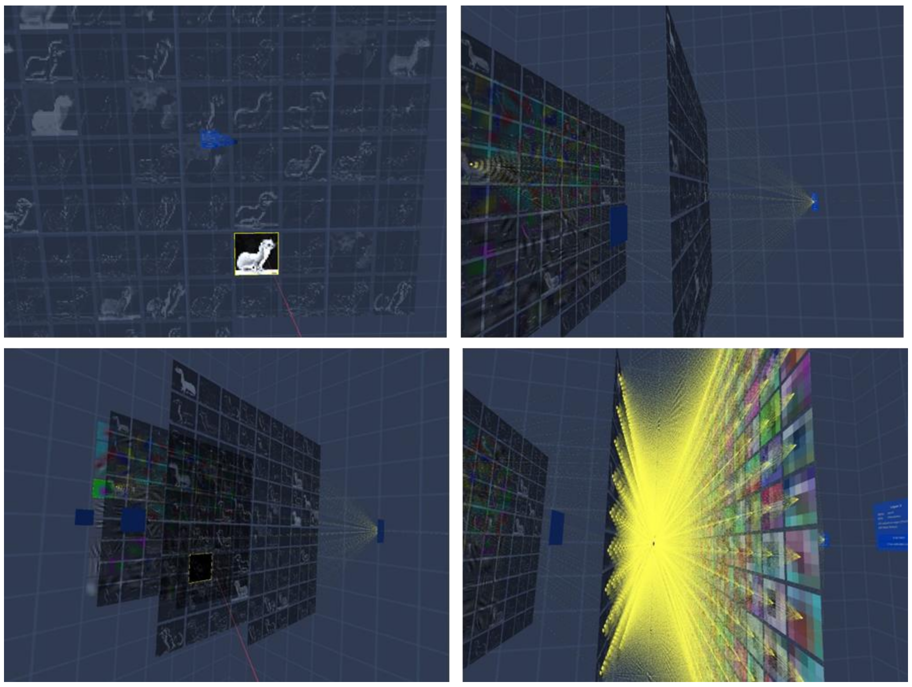


Fig. 8: Particular neuron selection and its linkages in the deeper layers.

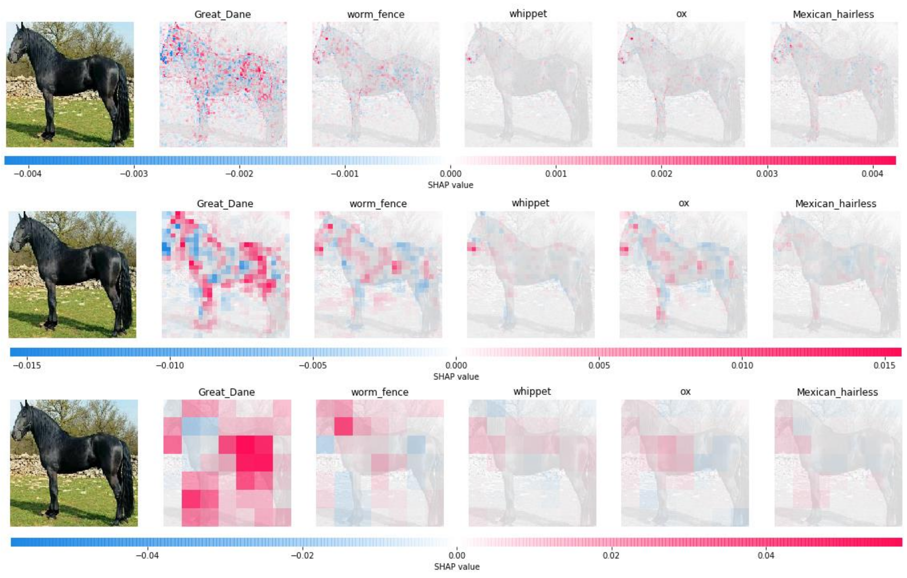


Fig. 9: Zoomed in view of VR based Shapley value interpretation showing a misclassification result of Black horse identified as Great Dane.

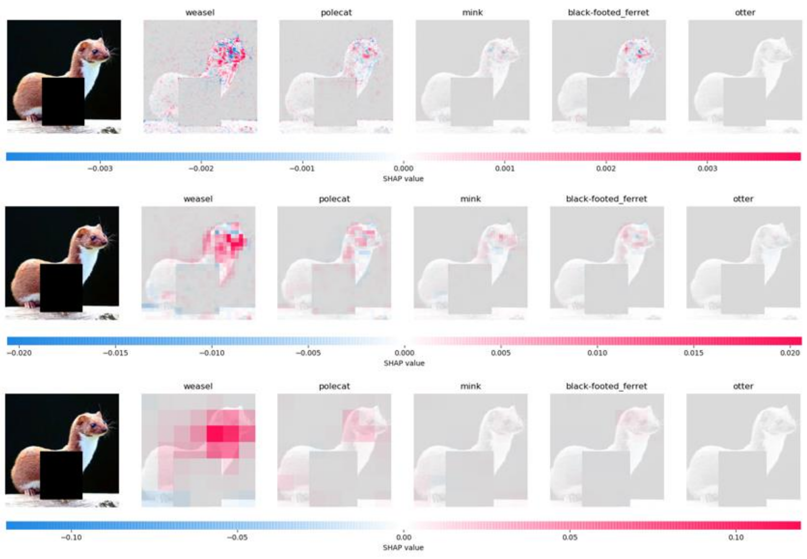


Fig. 10: Zoomed in view of VR based Shapley values for adding occlusion block on the less relevant region of the image. The network is still able to correctly identify the image despite adding block occlusion.

CHAPTER 6

Interpreting the decision of CNNs via influence functions

Model interpretability is at the core of understanding deep neural network decisions. It provides explanations that are understandable to humans and allows avoiding the risk of bias in model predictions. In the previous chapters, we dealt with interpreting the decisions of deep neural network based on test input images. However in this section we will investigate and interpret the model output based on images in training dataset i.e. which training image influences the decisions of the network. Here, we use the notion of influence functions that are commonly used in statistics to measure a change in a parametric observations and its effect on an estimator, hence useful for comparing the robustness and bias of the estimator [175]. In deep learning domain, we can use this notion to debug the results of a network model in respect to the training dataset. In this work, it was applied for analyzing perturbation of the loss functions to understand the behavior (specifically, class prediction) of deep learning models both in case of original test set images as well as when those images were exposed to disturbances. We considered three types of disturbances in the original images of ImageNet dataset: texture, style and background elimination. For images with those disturbances we calculated so called influence scores [153] layer-wise, at different hidden layers of the VGG16 network. The influence scores allowed to identify the most influential training images for the testing images from ImageNet as well as in case of those images with aforementioned disturbances.

Furthermore using hierarchical clustering, based on influence scores we grouped different disturbances and identified type of disturbances which bias network predictions the most. Layer-wise cluster analysis showed that the background information is more dominant on the lower layers, style is more dominant on the middle layers, while object and texture information prevails on the higher layers. Object was considered important for making decisions when images without background information were classified correctly. Our layer-wise influence scoring is an effective method to extract deeper semantic representations and could help improve classification accuracy by retraining the relevant hidden layers.

6.1 Introduction

Today's machine learning algorithms based on deep networks have outperformed humans in solving tasks in various fields, not only in the computer vision domain but are also excelling in the industrial and medical domains [131]. Moreover, their ability to make predictions, exploratory data analysis and visualization are remarkable [132] and [133]. The success of deep networks is due to the availability of high-end computing devices [134], larger datasets for learning [10] and [135], and improved deep learning techniques [136] and [137]. However, despite their success in many domains, these complex structures suffer from a lack of interpretability and transparency in their learned representations [138]. One of the main reasons for this is their "black-box" nature and the distributed encoding of the data from which they generalize and learn the representations [138]. In order to understand the input-output relation of these complex models [139], we need to probe the individual or cluster of neurons to visualize and encode the acquired concepts [140] and [141]. There are certain approaches that deal with prototype building of learned representations in an abstract manner. For instance, to explain the concept of a "bird", these methods learn the representations by creating prototype images of the class of interest [142], [143], [61] and [80]. These kinds of learned representations generally adopt activation maximization as their basis and have proved to be an effective tool for providing a more transparent and visual understanding of CNNs. Another means to make CNN decision-making transparent is to consider individual predictions, i.e., heatmaps highlight the pixels most relevant for making a decision [144] and [61]. In [145] and [146], the authors used layer-wise relevance propagation to explain predictions applicable in many machine learning models such as CNNs [145], LSTMs [147] and support vector machines [148]. In order to understand the predictions based on model behavior, the authors in [149] presented the spectral relevance analysis, which identifies individual heatmaps and clusters the learned concepts for classification.

In this chapter, we used influence functions, which have a rich history in statistics, but not much has been found in the literature regarding their use in the deep learning domain. In order to identify the influence of a training instance on a particular model prediction, a simple way is to delete the training instance and retrain the model. Taking the difference between predictions can determine the influence, but retraining large networks requires a lot of time. To simplify this, one can establish if the network has a loss function that is twice differentiable with regard to its parameters. If so, we can approximate the influence of any instance on the model parameters and its predictions. Another reason why influence functions have not been so popular is because of the high computational cost of determining if a model's loss function is twice differentiable, which limits their use in today's machine learning models. However, there are methods that can efficiently and accurately approximate the influence functions using second-order optimization techniques [150] and [151]. Furthermore, [152] and [153] have used influence functions to present example/image-based explanations by identifying the most influential training images responsible for classification. These methods are useful for identifying model errors and biases, identifying mislabeled datasets and debugging models for better predictability, but they lack interpretability in terms of identifying the learned representations. In [153], authors approximate Influence functions using the second-order optimization technique to represent model's behavior through training data [153]. We used this method and calculated the layer-wise influence score of each training image with regard to each test image, finding the most influential training images at each layer. As mentioned above, we used three different types of disturbances to the input images: texture, style and removing the background. For layer-wise influence scores, we applied cluster analysis to determine which of the aforementioned disturbances are learned at each layer.

6.2 Related work

Despite the many challenges in determining an appropriate method that yields insights into deep network performance, one must be sure that the explanations of these methods give a true reflection of the internal functionality of these models [154]. Even the best-performing deep learning models in various computer vision domains need a more transparent explanation of their predictions. For instance, interpretability via perturbation of data points has been extensively studied by [153], [130], [155], [156], [157] and [158], and they evaluated the changes in model outcomes either globally or locally. Moreover, perturbation-based methods are often inconsistent in their explanations, which might be true for one data point but not for its neighboring point or with similar data points within the same class.

The saliency-based method is mostly studied for the interpretability of local features in image classification tasks [159], [160] and [161]. These methods emphasize the importance of individual pixels in image classification tasks, however, conclusions obtained for one image cannot be transferred to another image, and thus these local explanations are not sufficient to reflect model decisions. Instead, we need methods that address the distributed encoding of a neural network in a systematic way. In this regard, influence functions are a technique originating in statistics that has been used in machine learning tasks to trace predictions back to the training data [153], and to study robustness and cross-validation within a model [162], [163] and [164]. A similar method is used to estimate the Cook's distance for prioritizing the training points [165], whereas an influence-based distance metric is defined to understand and configure the classifiers [166]. Other methods use influence functions where the model is given adversarial examples to interpret the model decisions [167]. Meanwhile, changing the class labels in the subset of the training set improves the network prediction for incorrect test inputs [168]. These methods performed well in different machine learning models, but using them as a robust statistical approach for post hoc interpretability is an emerging research direction in the deep learning domain. We used influence functions to identify the added disturbances in images to which the network is most sensitive , and explored how the hidden layers differ in terms of their influence scores in case of different types of disturbances.

6.3 Methods

6.3.1 Influence score and data set

Let us assume that the CNN is pre-trained for the task of image classification, and we want to find out the importance of different hidden layers of the CNN. The non-linear nature of CNNs gradually untangles the semantic information as the activation moves towards the deeper layers [169], [170] and [171]. We describe the details of our methodology based on the schematic diagram as shown below in Fig. 6.1. Specifically, we will use Influence functions [153] to analyse network decisions in case of regular test set images as well as when disturbances have been added to those images.

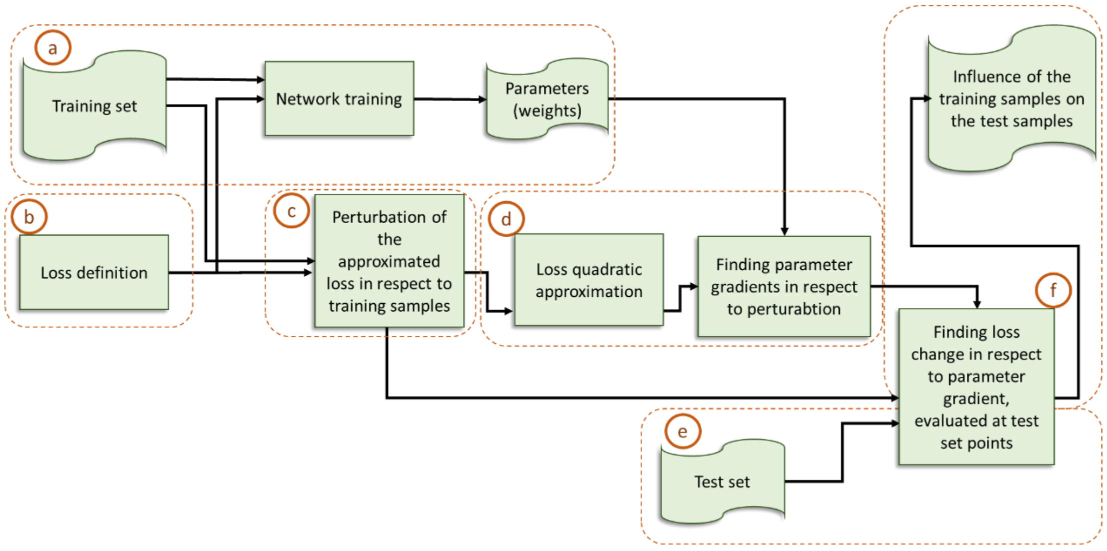

a
Training set
Network training
Parameters (weights)
Influence of the training samples on the test samples
b
Loss definition
c
Perturbation of the approximated loss in respect to training samples
d
Loss quadratic approximation
Finding parameter gradients in respect to perturabtion
f
Finding loss change in respect to parameter gradient, evaluated at test set points
e
Test set

Figure 6.1: Schematic diagram illustrating the methodology used to interpret the model decisions using the notion of influence functions. The description of blocks enclosed within dotted line is marked (a-f) and is described alongside its relevant text and equations below.

To accomplish this, we first define a training dataset for a neural network as $R_t = \{x_1, x_2, \ldots, x_n\}$, where n is the number of training samples; $x_i = (a_i, b_i)$ where a_i are class images of the size 224 x 224 provided to the input of the neural network (marked as [a]), and b_i is the output of the network, defined as one-hot encoding in c output lines, where c is the number of classes. We then define a loss function [b] given as:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(x_i\,, \theta) \tag{1}$$

Where L is the categorical-cross entropy loss with which the network was pre-trained i.e. a softmax followed by cross entropy loss written as: $L = -log\left(\frac{e^{a_p}}{\sum_j^C e^{a_j}}\right)$ where a_p is the positive class. Formally we can write as: $L = -\sum_{i=1}^{c} b_i \log\left(f(a)_i\right)$ where $f(a)_i$ is the probability of each class and b_i is the true label class. For c number of output classes in the network model, the output of the softmax will be the probabilities for each of the classes. We can obtain these probabilities using $f(a)_i = \left(\frac{e^{a_i}}{\sum_j^C e^{a_j}}\right)$, i.e., we take exponentiation (e^{a_i}) which is the output of each neuron for each class divided by the sum of all the exponents $(\sum_j^C e^{a_j})$ to obtain $(f(a)_i)$ i.e. the probability of each class. Let us say that all learning samples in the beginning are contributing to the loss equally with coefficients $1/n$ as defined in eq. (1) above. We will probe the loss function $L(\theta)$ by decreasing contributions of individual learning samples. Specifically, we will investigate the perturbation of the loss in respect to the training samples[c]. Our aim is to calculate how the network parameters θ would change in the case of changing the contribution of a specific sample $x_j \in (j = 1,2, \dots n)$ to a loss by a small quantity ϵ. For that, first we need to evaluate optimal network parameters for the loss function with the perturbation:

$$\hat{\theta}_{\epsilon,x_j} \overset{\text{def}}{=} \arg min_{\theta\in\Theta} \frac{1}{n}\left(\sum_{i=1}^{n} L(x_i\,, \theta) + \epsilon L(x_j\,, \theta)\right) \tag{2}$$

In [153], it was shown that the rate of change in optimum network weights $\hat{\theta}$ in respect to ϵ the way the latter quantity is defined in eq. (2), under the assumption of quadratic approximation of the loss function, can be expressed as follows[d]:

$$\begin{aligned} I_{mod,params}(x_j) &= \frac{d\hat{\theta}_{\epsilon,x_j}}{d\epsilon}\Big|_{\epsilon=0} \\ &= -H_{\hat{\theta}}^{-1}\nabla_\theta L(x_j, \hat{\theta}) \end{aligned} \tag{3}$$

Where, $\nabla_{\theta} L(x_j, \hat{\theta})$ is the perturbation of approximated loss gradient in respect to the training sample at the point $(x_j, \hat{\theta})$ and $H_{\hat{\theta}}$ is defined as follows:

$$H_{\hat{\theta}} = \frac{1}{n} \sum_{i=0}^{n} \nabla_{\hat{\theta}}^{2} L(x_i, \hat{\theta}) \qquad (4)$$

Let us now define the test set ⓔ (including both original test images and images with disturbances) as $R_c = \{\hat{x}_{c_1}, \hat{x}_{c_2}, \dots \hat{x}_{c_m}\}$. We will calculate the influence of the training image x_j on the loss at the test image $\hat{x}_{c_k}$ following the approximation given in [153]:

$$\begin{aligned} I_{mod,loss}(x_j, \hat{x}_{c_k}) &= \frac{dL\left(\hat{x}_{c_k}, \hat{\theta}_{\epsilon,x_j}\right)}{d\epsilon} |_{\epsilon=0} \\ &= \nabla_{\theta} L(\hat{x}_{c_k}, \hat{\theta})^{T} \frac{d\hat{\theta}_{\epsilon,x_j}}{d\epsilon} |_{\epsilon=0} \\ &= -\nabla_{\theta} L(\hat{x}_{c_k}, \hat{\theta})^{T} H_{\hat{\theta}}^{-1} \nabla_{\theta} L(x_j, \hat{\theta}) \qquad (5) \end{aligned}$$

Since directly computing the Hessian matrix $H_{\hat{\theta}}$ and its inverse as given in Eqs. (3-5) is computationally expensive, we used stochastic gradient to obtain HVP's and their inverse on mini batch of training images. Specifically, we calculate HVP's for each layer, which is the product between the Hessian matrix $H_{\hat{\theta}}$ and gradient vector of loss.

In [153] the gradients in Eq. (3) and influence scores in Eq. (5) were calculated based on the entire parameter set θ of the neural network. We expand the approach, by performing the analysis layer-wise, by separately finding gradients and Hessian Vector Products HVPs for each layer l in the network, thus obtaining layer-wise influence scores as given below:

$$\begin{aligned} I_{scr\_l}(x_j, \hat{x}_{c_k}) = I_{mod,loss}^{l}(x_j, \hat{x}_{c_k}) &= \frac{dL\left(\hat{x}_{c_k}, \hat{\theta}_{\epsilon,x_j}^{l}\right)}{d\epsilon} |_{\epsilon=0} \\ &= \left(-\nabla_{\theta} L(\hat{x}_{c_k}, \hat{\theta})^{T} H_{\hat{\theta}}^{-1} \nabla_{\theta_i} L(x_j, \hat{\theta}_{\epsilon,x_j}^{l})\right) \qquad (6) \end{aligned}$$

In the above expression, $\hat{\theta}^{l}$ represents the parameters for the l^{th} layer of the network. The computation for Eq.(6) can be efficiently obtained via Tensorflow expression given later in the text that will return the layer-wise gradients of the loss $L(\hat{\theta}_i)$ for an image x_j which we denote as $\nabla_{\theta_i} L\left(x_j, \hat{\theta}^l_{\epsilon,x_j}\right)$. By varying the parameters indices of $\nabla_{\theta_i} L\left(x_j, \hat{\theta}^l_{\epsilon,x_j}\right)$ in the above equation we can extract the influence scores for each individual layer. To calculate layer-wise score, we will be evaluating $\textcircled{f}$ i.e., expression (6) for each possible triplet of a training set image, network layer, and test set image $(x_j, \hat{x}_{c_k})$, j = 1,2, ... n and k = 1,2, ... m and will call it an **influence score**. The image in the training set x_j with the highest influence score for the test set image $\hat{x}_{c_k}$ will be called the (most) *influential image* in layer l formally defined as: $\hat{x}\, I_{inf}(l,k) = arg\, max_{j=1,2..n}\ I^l_{mod,loss}(x_j, \hat{x}_{c_k})$. We also analyze compound influence score based on layer-wise influence scores:

$$I_{total} = \sum_{l=1}^{n} I_{scr\_l}\ (x_j, \hat{x}_{c_k}) \tag{7}$$

An overall layer-independent (most) *influential image* can also be obtained using the above expression as $\hat{x}\, I_{inf}(k) = arg\, max_{j=1,2..n}\ I_{mod,loss}(x_j, \hat{x}_{c_k})$.We use the Tensorflow implementation of *get_Hv_op*() defined in *pyhessian* as (Hv = flatten (tf.gradients (tf.math.multiply (flatten (tf.gradients (L , $\hat{\theta}_i$)) , tf.stopgradient (v)) , params))) to efficiently calculate the HVP's similar to the one used in [153]. To extract layer-wise influence score for any layer l we use the above mentioned implementation to calculate layer-wise HVP's (L_{HVP}), mathematically represented in the following equation:

$$L_{HVP} = v^T H\ (\hat{\theta}_i)|_{\theta=\hat{\theta}_i} \in \mathbb{R}^P \tag{8}$$

In the above equation, $H(\hat{\theta}_i)$ is a matrix of second derivatives of loss and v is a matrix for gradient vectors of loss containing the vectors $v_1 = \nabla_{\theta_1} L(\hat{\theta}_1)$, $v_2 = \nabla_{\theta_2} L(\hat{\theta}_2), \ldots \ldots v_P = \nabla_{\theta_{P_l}} L(\hat{\theta}_{P_l})$ for different layers with P number of parameters in the network. The product between $H(\hat{\theta}_i)|_{\theta=\hat{\theta}_i}$ and v^T gives a matrix L_{HVP} with gradient parameters vectors $\hat{\theta}_i = [\theta_1\ \ \theta_2\ \ \theta_3\ \ \ldots\ \theta_{P_l}] \in \mathbb{R}^P$ for layer l. In the above equation, H is the same Hessian matrix from Eq. (3-5) (and used later in the text). Furthermore, we multiply element-wise the gradient vectors in v and H to obtain layer-wise HVP's (L_{HVP}). For each layer, we extract the L_{HVP} of the layer l cutting off the rest of the layers for which the product is not needed, and we repeat this for all the layers until we get separate L_{HVP} for all layers of the model.

We save the influence score of each layer into associated arrays or dictionaries and extract the *influential image* based on the highest influence score for the purpose of analysis. In addition, we take the intra-class mean of layer-wise influence score for each test set image with disturbance and save this score in the dictionaries. Whereas, a total influence for a particular image $\hat{x}_{c_k}$ is the sum of all layer-wise influences calculated in Eq. (7) as a compound influence score.

Later, we show a comparison and establish a relationship of the intra-class mean influence score calculated for a particular test image with disturbances (using I_{total} from Eq. (7)) and the corresponding non-disturbed image as a control group. In this study, we have taken an average of the intra-class influence score over the samples within each class given as: $I_{avg} = \frac{\sum I_{total}(for\ each\ image\ per\ class)}{No\ of\ images\ per\ class}$ to identify an influence based trend between the image disturbance and their controlled groups.

In order to test our methodology, we used the VGG16 network architecture, and defined the cues we wanted to quantify. We evaluated our method on the ImageNet dataset with 10 classes (chair, cat, elephant, zebra, screwdriver, bird, cup, toaster, bus and bicycle) using three types of disturbances: (1) foreground placed on a white-background, (2) with texture added (Textured) and (3) with style added (Styled), where original images were used as controls (Fig. 6.2). We simplified the class labelling, i.e., ImageNet's Persian cat is labelled as "cat" and the hummingbird as "bird", etc. All original images are from the ImageNet dataset, and we used a subset of approximately 30,000 images (30*1000) as our training set and 200 images as a testing dataset, where the latter contains disturbances (200=5 images *10 classes*(1 original + 3 disturbance types)) .

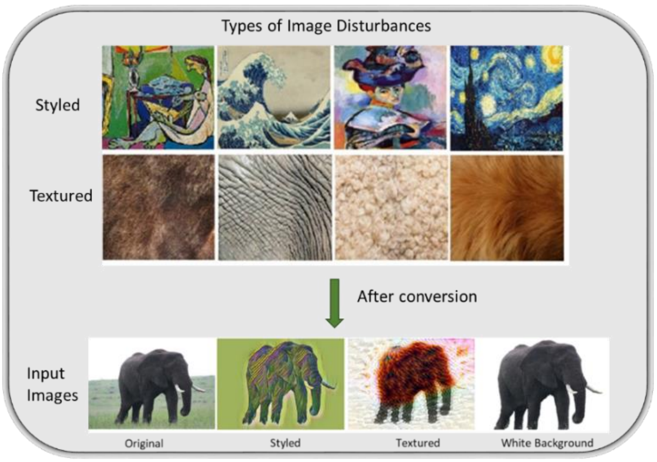


Figure 6.2: Sample styled and textured images used for adding disturbances and the converted input images of elephant class after the added disturbances.

For white-background images (five per class) we eliminated the background and replaced it with white colour. To obtain styled images, we adopted the method from [172] where we applied the style transfer to the white-background images. We transferred four different styles to 50 images (5 per class) where style assignment was randomized. To obtain texture images (50 overall, 5 per class), we used the method from [173] for different texture patterns like skin, fur, wool, feathers, etc. We summarized the overall methodology in Algorithm 1.

```
Algorithm 1: layer-wise influence score
Input:  training set and testing set images: $\boldsymbol{R_t} = \{x_1, x_2, \ldots, x_n\}$ $\boldsymbol{R_c} = \{\hat{x}_{c_1}, \hat{x}_{c_2}, \ldots \hat{x}_{c_m}\}$
Output: network prediction influence score $\boldsymbol{I_{scr\_l}}(x_j, \hat{x}_{c_k})$ and most influential learning images for test images
pre-process images of style and texture using [172][173]respectively
Step 1: prepare _data to create archive
  -rt   :  labelled training images
  -rc   :  labelled test images
  -mod  :  modified images
  -o    :  output filename of archive
Step 2: calculate $\boldsymbol{I_{scr\_l}}$
        -Create network model instance
  For each image from $\boldsymbol{R_c}$, $\boldsymbol{R_t}$ and for each layer $\boldsymbol{l}$
     -Using eq. (2) calculate the optimal parameters for the perturbed loss $\hat{\boldsymbol{\theta}}_{\epsilon,x_j}$
     - calculate layer-wise HPV's $\boldsymbol{L_{HVP}}$ using eq. (8) for each layer w.r.t $\boldsymbol{R_c}$
     - expanding eq. (5) for each layer and employing the obtained $\boldsymbol{L_{HVP}}$ ,calculate parameter gradients $\boldsymbol{I^l_{mod,loss}}(x_j, \hat{x}_{c_k})$
     -calculate influential images of layer-wise $\boldsymbol{I_{scr\_l}}$, and total influence using eq.(6)and(7)
     -return layer-wise influence score , influential images
  End
```

6.3.2 Clustering and cluster variables

We used hierarchical clustering as a statistical method to determine the similarity among the various images based on the observed influence scores. We started by considering the images with added disturbances and the intra-class influence score of all layers of the VGG16 as separate clusters, i.e., a cluster of singletons. We took an average of the individual layers from ($I_{l\_avg} = l_{1\_avg}, \dots, l_{16\_avg}$) for all the intra-class images in the testing dataset (i.e., 5 images*4 cases*10 classes) as feature vectors. The clustering was based on 4 cases: original image, image with white background, textured and styled images. At each iteration step, we selected two singletons (i.e. images with disturbances and original images) and measured the similarity S_{ij} (%) between cluster-singletons "i" and "j". We also calculated the average Euclidean distance d_{ij} between singletons from cluster-variables (i.e. original, styled, textured and white background) using their layer-wise influence score and merged the pair with the least distance. We then calculated a correlation distance matrix using Pearson's correlation ρ_{ij} [174] to identify the common characteristics based on the layer-wise influence score to identify at which layers the disturbances in the images were more noticeable.

Using the clustering analysis procedure given as Algorithm 2, we identified three clusters to show which image disturbances were learned at hidden layers of the network. The clustering results were calculated based on the average of the intra-class influence scores. Our clustering method considered original and white-background images as one cluster, style and textured images as separate cluster. We also double-checked with test image interpretability (i.e., Shapley values) to confirm the findings we had obtained using training image interpretability (i.e., using influence functions). In order to accomplish this, we used Shapley values [130] to interpret the network predictions. In doing so, we analyzed the predictions based on which features of the test image are important for the network. We used both of these schemes to analyze the findings and to estimate where the network remained consistent in making its decisions.

```
Algorithm 2: layer-wise hierarchical cluster analysis
Input:  layer-wise influence score from eq. (6) for each image disturbance and its control images
Output: cluster variable and layer-wise cluster observations
For each $x_j$ in $R_c$
    -1. For each class, select the average intra-class layer-wise influence scores of four different cases: original images as well as white background, textured and styled images as single separate clusters
    -2. Calculate similarity and least distance between each cluster singleton using $S_{ij} = \frac{(1- d_{ij})}{d_{max}} * 100$.
    -3. Merge clusters with least distance
    -4. Calculate Pearson's correlation distance using $D_{ij} = 1 - \rho_{ij}$ between each cluster variable
    -5. Update distance matrix
    -6. Repeat step 2 to 5 until one cluster is left
End
return cluster_ variables based on similarity, layer-wise cluster observations
```

6.4. Result and discussion

We performed experiments on the VGG16 architecture using the ImageNet dataset, with our training set consisting of 30,000 images. In our study, we calculated influence scores for 200 test images and the corresponding layer-wise influence scores to determine the basis for the network's decisions.

6.4.1. Image classification with input image disturbances

As an initial experiment, we tested the classification response for all types of disturbances. Nearly all images from the "original" and "white-background" categories were correctly classified by the network (Fig. 6.3). It is worth noting that the images with a white background showed predictions comparable to images in the original category, making the object an important cue for decision making. For the styled images, prediction probabilities were slightly less compared to the original and white-background images. In contrast, textured images for all classes failed to classify correctly due to the texture coming from completely different classes. The decision was made based on the added texture rather than the class itself. Hence, the decision was biased, suggesting that the texture of the image is an important cue for network decisions.

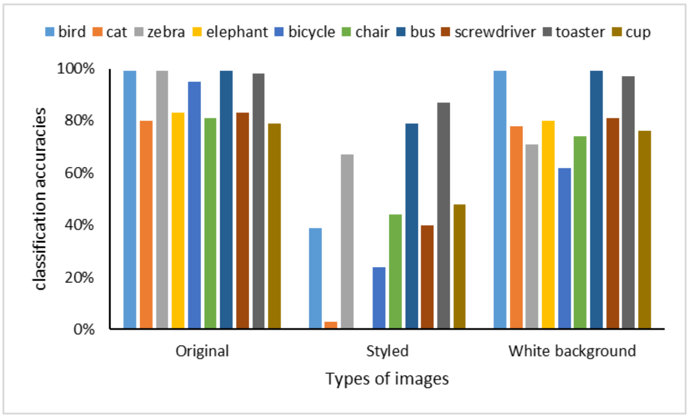


Figure 6.3: Classification accuracies of the classes with varying image disturbances. All textured images failed to give correct predictions (0%) hence are not shown in figure above.

Texture-style analysis

To verify the importance of texture for the network's decisions, we tested the network with texture patches as input. The network was able to correctly identify the texture with classification probabilities given in Table 1. Further, we transferred the textured patches to 50 white-background images using the method in [173] (Fig. 6.4) to confirm our findings. Again, the network was making its decision based on the texture but not on the content of the image (Table 1, second column).

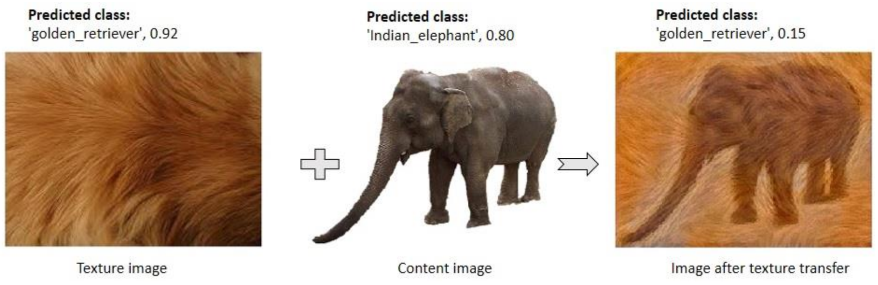


Figure 6.4: Example showing predicted class label and accuracy for elephant class before and after transfer of texture.

Styled images were also observed based on the activations of the hidden layers of the VGG16 network. We used this style-textured cue to see how the influence score varied along the network layers, as we will show next.

Table 1: Classification accuracies (%) of example texture image patches and class labels in case of texture transfer onto content images

Before texture transfer		**After texture transfer**	
Texture image	**Accuracy (%)**	**Content image**	**Predicted class label**
Zebra	0.93	Bird	Zebra
Leopard	0.93	Cat	Leopard
Elephant	0.95	Bus	Indian elephant
Golden retriever	0.92	elephant	Golden retriever

6.4.2. Influential image score and learned representations

In the previous sections, we calculated influence scores for each of the 16 layers of the VGG16 network and named it as layer-wise influence score (I_{scr_l}). We also gave total influence scores I_{total} as a sum of all the 16 layers to analyze the global network predictions. To identify influential images, we used this total influence score for all images with added disturbances as well as the original images to identify which training image influenced the prediction of a testing image with the biggest influence score.

Although all the original images were predicted correctly and were given the same predicted class label as their influential image, we also observed that their influential images were very sensitive to the background color (Fig. 6.5a). For white-background images, the network's prediction was also correct, and the influential training image also used to belong to the same class as the original. The only difference was that this time the influence score was slightly less compared to the case of the original image. Thus, we can conclude that the object itself is important and must have shaped the features of the CNN. Removing the background information in this case had very little effect on the final decision. For this, we calculate the average of the intra-class influence score to show a comparison of the images with disturbances with their original images as control. This effect can be seen from the average influence scores (I_{avg}) plotted for all the classes in Fig. 6.6c.

As for the styled and textured images, one interesting observation was the similarity of the spatial features of the test images with their most influential training images (Fig. 6.5 b, c) and relatively fluctuating influence scores (Fig. 6.6 a, b) with their corresponding original images. However, in this case, the influential images played a negative role in the sense that the predicted class labels for the test images were very different from their influential image class, indicating a strong dependence on the styling and texture representations of the input images. Although, for styled and white-background images, the average influence score shows comparable fluctuations with their control groups, indicating their strong influence on correct network classifications, as is evident from Fig. 6.3. However, the average influence score of textured images as well as their control group is very low, indicating that influential training images are not considered for prediction, but the added texture was more dominant in network decisions. In all three cases, only the bird class showed a high influence score, indicating that there exists a strong similarity between the test and training instances as learned by the model.

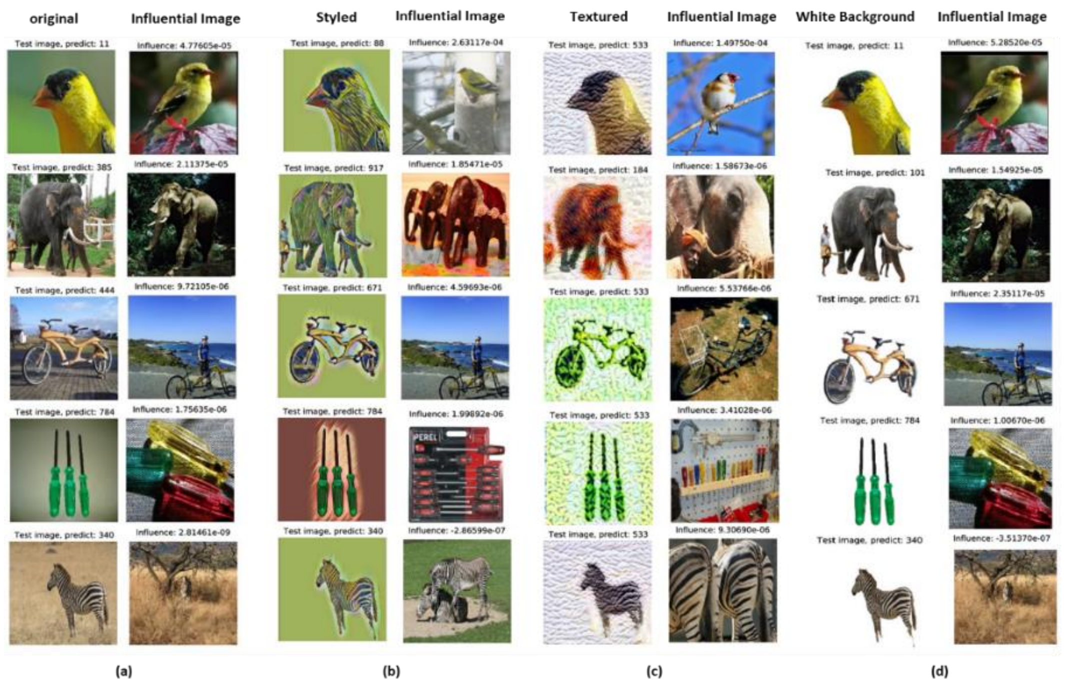


Figure 6.5: Intra-class influential images with highest influential score calculated using (I_{total}) among the training set and corresponding test inputs of (a) original, (b) styled, (c) textured, and (d) white background images.

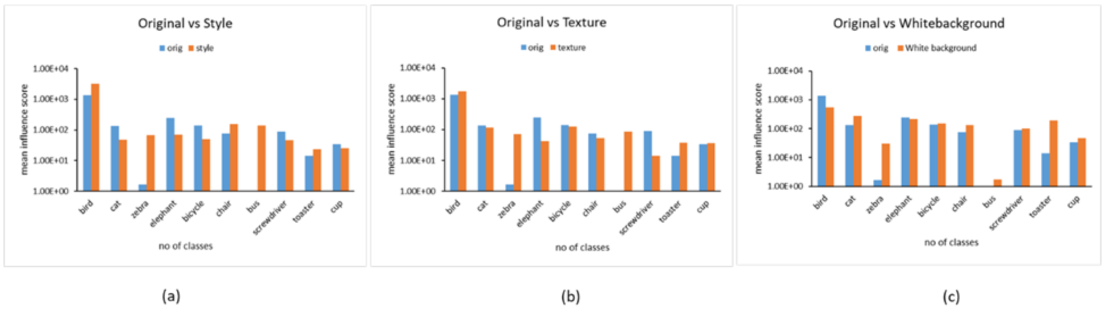


Figure 6.6: Comparison of intra-class scores showing a plot between average influence scores (I_{avg}) : (a) original vs. styled images, (b) original vs textured images and (c) original vs white background images.

6.4.3. Interpretation of influence scores via Shapley values

To further evaluate and interpret the learned representations, we identified Shapley value-based [130] influential regions between different types of test inputs (Fig. 6.7). Here, we did not calculate layer-wise Shapley values but only considered the test images to see which image regions were important for the network. The reason for this analysis is that we wanted to identify the similarities in terms of what the network looks at in making its decision through the test image and its training images. We considered a region to be influential if the Shapley values in that region were higher (marked by red color in the figure). We observed that in the original and white-background images, high Shapley values were concentrated on an object, so we could consider the object to be an influential region (Fig. 6.7 a, d). For the styled and textured images (Fig. 6.7 b, c), the influential regions are wider than in their corresponding original images. Thus, we can conclude that style and texture are important in network decisions.

In addition, we detected that a change in the amount of added style-texture (%) changed the network's decisions based on the fluctuation in influence scores. We observed an inverse relationship between the percentage of style-texture and the influence score as well as network classification accuracy, i.e., as the amount of style or texture increased on the test images, the influence that a training instance had on the network's accuracy in correctly classifying the images decreased. We showed a common trend of the inverse relationship of texture varying from zero to 50% in iteration steps among the individual classes. The generalized relationship plot was based on the average of the texture-based influence scores and accuracy among the classes, which showed a similar trend and is shown in Fig. 6.8. As we increased the amount of texture on the test images, with each iteration, the influence score was turning negative and showing less resemblance to the training images, resulting in incorrect object recognition. So, we can say that the texture of the image becomes more influential on the network's decisions than the object itself.

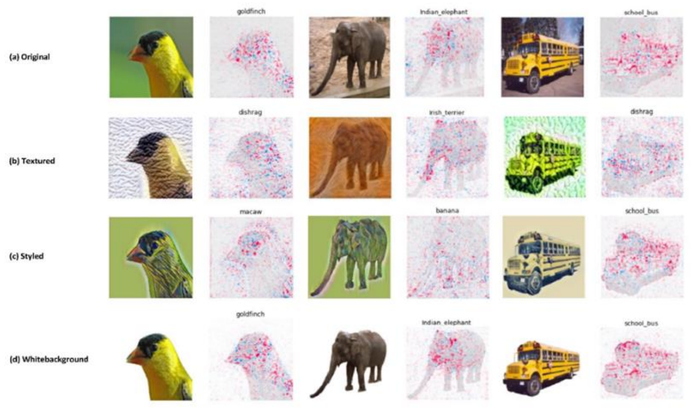


Figure 6.7: Shapley-value based influential regions of images with disturbances, (a) original (b) textured (c) styled and (d) white background images.

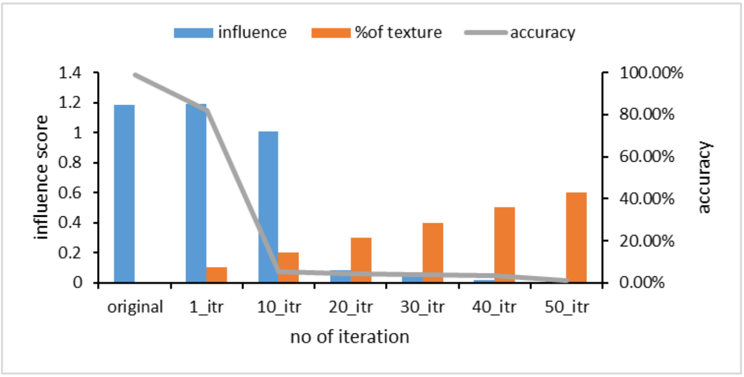


Figure 6.8: A generalized inverse-relationship of various classes showing a similar trend between averages (I_{avg})of texture based intra-class influence scores and accuracy.

6.4.4. Interpretation via cluster-analysis

Moreover, we performed the cluster analysis based on Pearson's distance correlation of the average of the layer-wise influence scores of input disturbances and the control groups from the same class. As an illustration, we present the results for three classes (bird, elephant and bicycle).

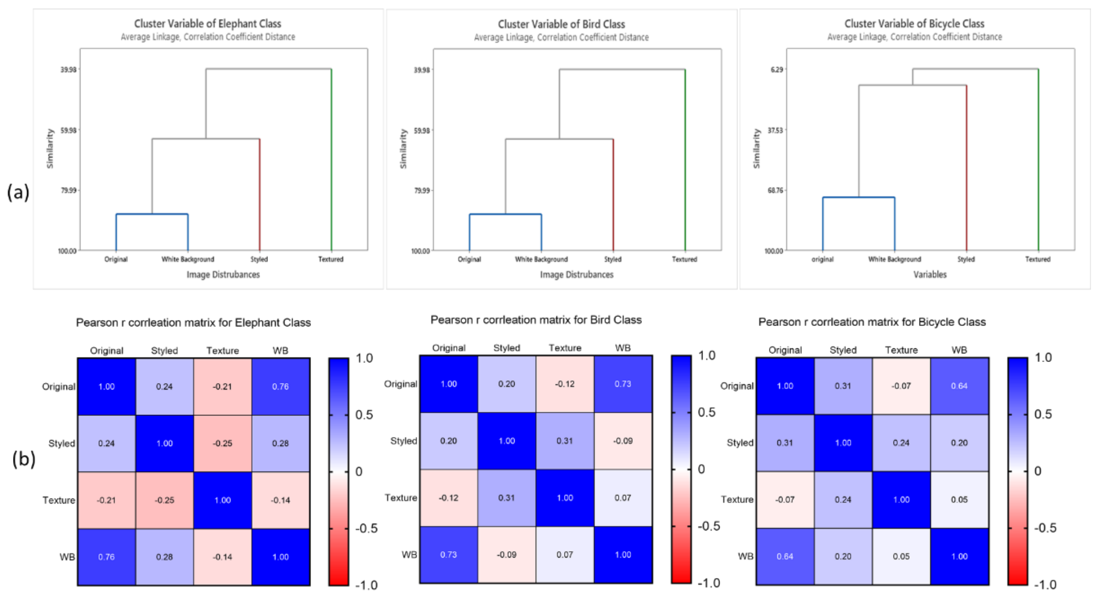


Figure 6.9: Layer-wise intra-class cluster analysis of the influence score on VGG16 network shown for three image classes, (a) identified three cluster , evaluated by Pearson´s distance correlation based on I_{avg}. The original and white background images were determined as one cluster in terms of their similarity while style-textured images were identified as separate clusters based on dissimilarity from their controlled groups. (b) Their corresponding correlation matrix based on intra-class average layer-wise influence scores (I_{l_avg}) for the above classes.

We identified three clusters: original and white background as one cluster shown in blue, styled in red and textured in green (Fig. 6.9a). The color association of the cluster variables is assigned depending on how similar the observations are to better visualize the common characteristics in clustering pattern. The similarity is determined based on the global influence score calculated for each intra-classes using I_{avg}. To further learn about the common patterns that the model considers during decision making, we identify similarities among different images at individual layers using I_{l_avg}. So in general, one could get a global view via clustering as identified in Fig 6.9a and also in the correlation matrix (Fig 6.9 b) but to get a deeper understanding of why these clusters are formed we illustrate this relationship among the images with disturbances in Fig 6.10. Hence finding reasons and interpretation of the common features among different types disturbance in images. The analysis showed a high similarity of the layer-wise influence score between the original and white-background images which were identified as one cluster (in Fig. 6.9a) making the object serve as a learned representation for prediction in higher layers. In contrast, the background information was more dominant at the lower layers and showed a high similarity (%) to the influence score between the original and white-background images (Fig 6.10 a, b). There is clear evidence of similarity among its influence score that can be visible in both Fig 6.10 a, b in first few layers and in the prediction layer. Where for white background images the prediction layer shows slightly high values for influence score hence making a correct classification of the object. This is also evident from our results presented in Fig. 6.5 a, d and Fig. 6.7 a, d. Since the results are presented for all classes and they all seem to follow a common pattern across different layers depending upon the type of added image disturbances hence we represent these results as layer-wise clustering relationship. There is a very similar layer-wise clustering pattern for original and white background images as compared to the style-textured images.

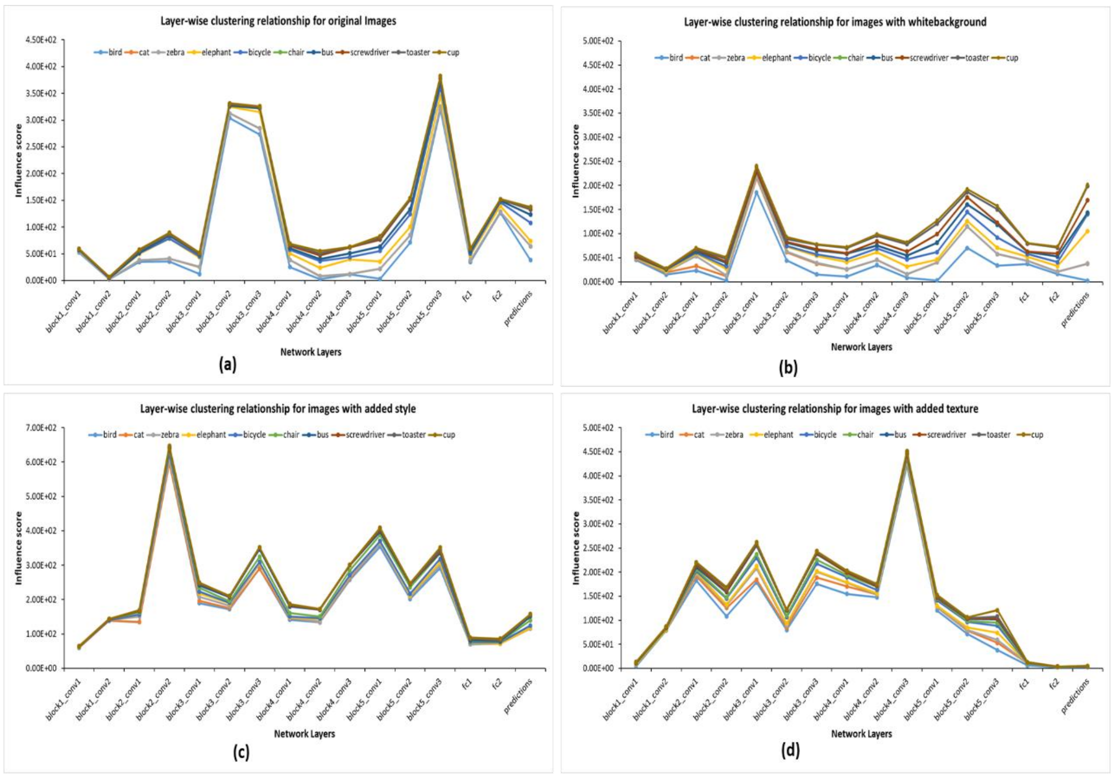


Figure 6.10: Intra-class layer-wise influence scores calculated from training set onto its corresponding images with disturbances show a clustering relationship of influence scores varying across the network layers for: (a) original images, (b) images with white background, (c) images with added style and (d) images with added texture for all the classes.

The layer-wise influence scores of the styled images fluctuated more towards the middle layers when compared with their controlled groups, indicating that the deeper layers of the network observe more abstract representations (Fig. 6.10 c). Styling the images modifies their appearance, making it difficult for the network to correctly correlate with its original image features in decision making. The values for influence score fluctuate for the middle layer as compared to the original images. However, despite the fluctuation and added style, the network does try to show some correct classification results which is also visible from the last layers. In contrast, no similarity is identified in terms of influence scores among the images with added texture and its corresponding original images. A very low layer-wise influence scores at the higher layers result in 0% classification accuracy of the textured images compared with their original images. This provides evidence that influence of embedded textures at the higher layers giving more semantic representations of the learned texture (Fig 6.10 d). Hence, the textured images formed a separate cluster (see Fig 6.9 a) as there was a high dissimilarity and increased distance between the layer-wise influence scores compared with its control group.

6.5 Conclusions

In this paper, we used influence score to analyze and interpret the decisions of the VGG16 network. We presented the network with various types of input images to observe changes in its predictions. We demonstrated that the network model strongly relies on the style-textured cues of the images. However, the type of input image and object-background information also play an important role in network decisions. We analyzed the network predictions using layer-wise influence scores to determine the influential images and gave evidence for the fluctuations in the decisions. Through cluster analysis, we identified at which layers the disturbances in an image were more noticeable. We observed a sudden drop in the network's recognition accuracy when we varied the style-texture ratios and finally concluded that the pre-trained model under study is biased towards style and textures in decision making.

As a future work, we can use the influence score to gain deeper insights into more abstract representations of various other network models. For instance, considering the layer-wise influence score, we can improve the classification accuracy and debug the model to achieve improved results, which can be a promising direction for future research avenues.

CHAPTER 7

Conclusions

In this thesis, we argue that visualizing and interpreting decisions through visuals can be useful for understanding complex "black-box" models. Images, objects, scenes, etc., are core to any classification problem in the computer vision domain. Gaining an in-depth understanding of why these visuals can be important for model predictions, such as deep neural architectures, can help unlock these "black-box" models. These visuals can give substantial weight to providing logical explanations, assessments and understanding of the distributed information encoded within these networks. We use these visual representations to interpret the decision-making capabilities of deep neural networks, focusing on the goals and methods adopted to accomplish our task. In this chapter, we will first highlight the important findings of our approaches used to accomplish our objective, and secondly, we will conclude this work with some future perspectives and remarks.

7.1 Summary

After a thorough review and analysis of the different methodologies that already exist in this field, we gave a concise amalgamated review of the literature on visualization techniques for deep neural networks in the computer vision domain. Building upon our review, we presented the idea of opening the "black-box" model in a virtual reality environment. The reason we chose this visualization strategy was that up till now, previous work on visualizing DNNs has been limited to 2D images, as it is easier to directly map the data that DNNs work on. Images, however, do not suffice to visualize the internal functions of a network. Thus, to fully capture the planned information, we moved towards 3D visualizations. Still, this would not create good visualizations due to occlusion and scatter in the 3D formations. In order to cope with the complexity of 3D visualization, we intended to build an interactive exploration and planned to use a virtual reality-based visualization. The intention was to enable the user to intuitively navigate through the immersive information space and gain a better visualization understanding. The user would be able to develop new visualization strategies, see the activation produced in the neurons, determine the features that make an image distinct from others, and last but not least, interpret the decision-making capabilities of these deep architectures.

To the best of our knowledge, the work presented to integrate a Windows-based Caffe framework into the Unity gaming engine is the first of its kind. We developed Caffe2Unity.dll as a plugin for the gaming engine, allowing the user to utilize the functionality of a complete deep neural network and make the application platform-independent.

As a global interpretability, we performed a complete visualization and interpretability of individual layers, attributions of neurons and filters and activation maps in a virtual reality environment. We gave solutions for both real-time and non-real-time situations depending upon the availability of computing resources, targeting both experts and novices in this field.

We used our hybrid interpretability approach, which incorporates both a local and global interpretation method, to visualize the decisions of the deep neural network. We reframed the Shapley values approach in a VR environment and added an occlusion block to gain a real-time understanding of the network decisions. Varying the size of the block at any random location on the test images gives the VR user the freedom to understand the important regions within the images that the network deems relevant in its final prediction.

Later, we extended our work to include a reverse interpretability approach, giving the idea of using layer-wise influence scores to determine which training data is responsible for the network's decisions, as opposed to the test images. To achieve our goal of interpretability, we used images with disturbances to see the variations in the network decisions. We observed an inverse relationship between the styled-textured influence scores and the accuracy of the correct predictions. By analyzing individual layers, we observed which insights were being learned at individual layers and identified the bias of the pre-trained model towards the varying texture of the images.

7.2 Future perspective and remarks

Based on our findings and the insights gained during this work, we suggest the following improvements and extensions for a future work:

- In this current work, we used Caffe2Unity for the purpose of interpreting and visualizing the deep neural network. However, we plan to extend its usability for other computer vision-related tasks such as image retrieval, segmentation and detection to be performed within a virtual reality environment, making the task more application-oriented and scalable.
- The key idea behind integrating a deep network framework into a gaming engine was to reduce the extra burden of human effort when executing and interpreting action-oriented tasks for robots. Performing simulations repeatedly in virtual scenes can enable researchers in this field to improve their existing annotation results, avoid costly training time and incorporate more challenging tasks.
- One very interesting connection exists between computer vision tasks and natural language processing, i.e., the ability to understand images in a human-understandable language. We can improve our interpretation and understanding of machine learning models if the learned concepts are interpreted in a natural language instead of pixel representations. Incorporating such changes into our layer-wise influence approach can significantly improve the gist behind concept learning in deep neural networks.

References

[1] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. Computational Geometry, 4(5):235 – 282, 1994.

[2] B. Becker, R. Kohavi, and D. Sommerfield. Visualizing the simple bayesian classifier. In KDD Workshop Issues in the Integration of Data Mining and Data Visualization, 1997.

[3] D. M. Bruckner. Ml-o-scope: a diagnostic visualization system for deep machine learning pipelines. Technical Report UCB/EECS-2014-99, University of California at Berkeley, 2014.

[4] C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu, D. Ramanan, and T. S. Huang. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. In 2015 IEEE International Conference on Computer Vision (ICCV), 2015.

[5] B. Caputo, E. Hayman, and P. Mallikarjuna. Class-specific material categorisation. In Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, 2005.

[6] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. CoRR, abs/1311.3618, 2014.

[7] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In Computer Vision and Pattern Recognition (CVPR), pages 3642–3649, 2012.

[8] G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2(4):303–314, 1989.

[9] J. Dai and Y. N. Wu. Generative modeling of convolutional neural networks. CoRR, abs/1412.6296, 2014.

[10] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255, June 2009.

[11] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In ICML, 2014.

[12] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[13] M. Eichner and V. Ferrari. Better appearance models for pictorial structures. In Proceedings of the British Machine Vision Conference, pages 3.1–3.11. BMVA Press, 2009. doi:10.5244/C.23.3.

[14] M. Eichner and V. Ferrari. We are family: Joint pose estimation of multiple persons. In Proceedings of the 11th European Conference on Computer Vision: Part I, ECCV'10, pages 228–242, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] M. Eichner and V. Ferrari. Human pose co-estimation and applications. IEEE Trans. Pattern Anal. Mach. Intell., 34(11):2282–2288, 2012.

[16] D. Erhan, A. Courville, and Y. Bengio. Understanding representations learned in deep architectures. Technical Report 1355, Universit´e de Montr´eal/DIRO, October 2010.

[17] S. Escalera, X. Bar´o, J. Gonzalez, M. A. Bautista, M. Madadi, M. Reyes, V. Ponce-L´opez, H. J. Escalante, J. Shotton, and I. Guyon. Chalearn looking at people challenge 2014: Dataset and results. In Workshop at the European Conference on Computer Vision, 2014.

[18] M. Everingham, L. van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 88(2):303–338, 2010.

[19] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. IEEE transactions on pattern analysis and machine intelligence, 2006.

[20] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02, CVPR '05, pages 524–531, Washington, DC, USA, 2005. IEEE Computer Society.

[21] R. Fuchs, J. Waser, and E. Gr¨oller. Visual human+machine learning. Proc. Vis 09, 15(6):1327– 1334, October 2009.

[22] K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. Pattern Recognition, 15(6):455 – 469, 1982.

[23] G. Griffin, A. Houlub, and P. Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007.

[24] P. J. Grother. Nist special database 19 handprinted forms and characters database. National Institute of Standards and Technology, 1995.

[25] F. Gr¨un, C. Rupprecht, N. Navab, and F. Tombari. A taxonomy and library for visualizing learned features in convolutional neural networks. In Proceedings of the International Conference on Machine Learning 2016, 2016.

[26] A. W. Harley. An Interactive Node-Link Visualization of Convolutional Neural Networks, pages 867–877. Springer International Publishing, Cham, 2015.

[27] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. Neural Comput., 18(7):1527–1554, July 2006.

[28] G. B. Huang. Learning hierarchical representations for face verification with convolutional deep belief networks. In Proceedings Conference on Computer Vision and Pattern Recognition, CVPR, pages 2518–2525, Washington, DC, USA, 2012. IEEE Computer Society.

[29] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 0749, University of Massachusetts, Amherst, October 2007.

[30] N. Jammalamadaka, A. Zisserman, M. Eichner, V. Ferrari, and C. Jawahar. Has my algorithm succeeded? an evaluator for human pose estimators. In European Conference on Computer Vision, 2012.

[31] S. Johnson and M. Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In Proceedings of the British Machine Vision Conference, 2010. doi:10.5244/C.24.12.

[32] D. Keim, P. Bak, and M. Sch¨afer. Dense pixel displays. In Ling Liu and M. Tamer ¨Oszu, editors, Encyclopedia of Database Systems, pages 789–795. Springer US, 2009.

[33] R. Kohavi. Data mining and visualization. Invited talk at the National Academy of Engineering US Frontiers of Engineers (NAE), 9 2000.

[34] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[36] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. Nature, 521:436–444, 5 2015.

[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, November 1998.

[38] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[39] L. Li and L. Fei-Fei. What, where and who? Classifying events by scene and object recognition.In IEEE Intern. Conf. in Computer Vision (ICCV). 2007, 2007.

[40] S. Li, Z.-Q. Liu, and A. B. Chan. Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014.

[41] S. Li, Z.-Q. Liu, and A. B. Chan. Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network. International Journal of Computer Vision, 113(1):19–36, 2015.

[42] T.-Y. Lin and S. Maji. Visualizing and understanding deep texture representations. In Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[43] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang. Casia online and offline Chinese handwriting databases. In 2011 International Conference on Document Analysis and Recognition, 2011.

[44] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. CoRR, abs/1604.07043, 2016.

[45] J. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? CoRR, abs/1411.1091, 2014.

[46] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2015.

[47] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. In International Journal of Computer Vision, 2016.

[48] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

[49] M. Minsky and S. Papert. Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, MA, USA, 1969.

[50] G. Montavon, S. Bach, A. Binder, W. Samek, and K.-R. M¨uller. Explaining nonlinear classification decisions with deep taylor decomposition. CoRR, abs/1512.02479, 2015.

[51] T. Munzner. Visualization Analysis and Design. A K Peters Visualization Series. CRC Press, 2014.

[52] A. M. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. CoRR, abs/1602.03616, 2016.

[53] G. P. Nguyen and M. Worring. Interactive access to large image collections using similarity based visualization. J. Vis. Lang. Comput., 19(2):203–224, 4 2008.

[54] G. Patterson. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), CVPR '12, pages 2751–2758, Washington, DC, USA, 2012. IEEE Computer Society.

[55] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large

Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015.

[56] W. Samek, A. Binder, G. Montavon, S. Bach, and K.-R. M¨uller. Evaluating the visualization of what a deep neural network has learned. CoRR, abs/1509.06321, 2015.

[57] B. Sapp and B. Taskar. Modec: Multimodal decomposable models for human pose estimation. In In Proc. CVPR, 2013.

[58] C. Seifert and M. Granitzer. User-based active learning. In Proceedings of 10th International Conference on Data Mining Workshops (ICDM), pages 418–425, 2010.

[59] C. Seifert and E. Lex. A novel visualization approach for data-mining-related classification. In Proc. of the International Conference on Information Visualisation (IV), pages 490–495. Wiley, July 2009.

[60] L. Sharan, R. Rosenholtz, and E. Adelson. Material perception: What can you see in a brief glance? Journal of Vision August 2009, Vol.9, 784. doi:10.1167/9.8.784, 2009.

[61] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualizing image classification models and saliency maps. CoRR, abs/1312.6034, 2014.

[62] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In Neural Networks (IJCNN), The 2011 International Joint Conference on, 2011.

[63] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pages 2553–2561. Curran Associates, Inc., 2013.

[64] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In 2014 IEEE Conference on Computer Vision and Pattern Recognition, pages 1701–1708, June 2014.

[65] K. Thearling, B. Becker, D. DeCoste, W. Mawby, M. Pilote, and D. Sommerfield. Information Visualization in Data Mining and Knowledge Discovery, chapter Visualizing data mining models, pages 205–222. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[66] S. Urbanek. Exploring statistical forests. In Proc. of the 2002 Joint Statistical Meeting, 2002.

[67] L van der Maaten and G. E. Hinton. Visualizing high-dimensional data using t-sne. Journal of Machine Learning Research, 9:2579–2605, 2008.

[68] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. Journal of Machine Learning Research, 9:2579–2605, 11 2008.

[69] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[70] J. Wang, B. Yu, and L. Gasser. Classification visualization with shaded similarity matrices. Technical report, GSLIS University of Illinois at Urbana-Champaign, 2002.

[71] J. Wang, Z. Zhang, V. Premachandran, and A. L. Yuille. Discovering internal representations from object-cnns using population encoding. CoRR, abs/1511.06855, 2015.

[72] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In IEEE International Conference on Computer Vision (ICCV), 2015.

[73] L. Wilkinson and M. Friendly. The history of the cluster heat map. The American Statistician, 63(2):179–184, May 2009.

[74] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. In in Proc. IEEE Conf. Comput. Vision Pattern Recognition, 2011.

[75] D. Wu, L. Pigou, P. J. Kindermans, N. D. H. Le, L. Shao, J. Dambre, and J. M. Odobez. Deep dynamic neural networks for multimodal gesture segmentation and recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(8):1583–1597, Aug 2016.

[76] Y. Xiang, R. Mottaghi, and S. Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In IEEE Winter Conference on Applications of Computer Vision, 2014.

[77] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In Computer vision and pattern recognition (CVPR), 2010 IEEE conference on, 2010.

[78] B. Yao, X. Jiang, A. Khosla, A. L. Lin, L. Guibas, and L. Fei-Fei. Human action recognition by learning bases of action attributes and parts. In ICCV, 2011.

[79] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N.d. Lawrence, and K.q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 3320–3328. Curran Associates, Inc., 2014.

[80] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. In Proceedings of the International Conference on Machine Learning 2015, 2015.

[81] W. Yu, K. Yang, Y. Bai, H. Yao, and Y. Rui. Visualizing and comparing convolutional neural networks. CoRR, abs/1412.6631, 2014.

[82] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In Computer Vision 13th European Conference (ECCV) 2014, 2014.

[83] B. Zhou, A. Khosla, `A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene cnns. CoRR, abs/1412.6856, 2014.

[84] B. Zhou, A. Khosla, `A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. CoRR, abs/1512.04150, 2015.

[85] B. Zhou, `A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 487–495. Curran Associates, Inc., 2014.

[86] L. M. Zintgraf, T. Cohen, and M. Welling. A new method to visualize deep neural networks. CoRR, abs/1603.02518, 2016.

[87] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi: You only look once: unified, real-time object detection. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788 (2016)

[88] M. Moravčík, M .Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, DeepStack: Expert-level artificial intelligence in heads-up no-limit. Science 356(6337), 508–513 (2017)

[89] D. Silver, J. Schrittwieser, K. Simonyan , I. Antonoglou , A. Huang , A. Guez , T. Hubert , L. Baker , M. Lai , A. Bolton ,Y. Chen, T. Lillicrap , F. Hui , L. Sifre ,G. van den Driessche , T. Graepel , D. Hassabis: Mastering the game of Go without human knowledge. Nature 550(7676), 354–359 (2017)

[90] D. Bahdanau, K. Cho, Y. Bengio : Neural machine translation by jointly learning to align and translate. In: International Conference on Learning Representations (ICLR) (2015)

[91] C. Seifert, A. Aamir, A. Balagopalan, D. Jain, A. Sharma, S. Grottel and S. Gumhold. “Visualizations of Deep Neural Networks in Computer Vision: A Survey”. In: Transparent Data Mining for Big and Small Data. Ed. by Tania Cerquitelli, Daniele Quercia, and Frank Pasquale. Cham: Springer, 2017, pp. 123–144

[92] J.-D. Fekete, 2013. Visual analytics infrastructures: From data management to exploration. Computer Institute of Electrical and Electronics Engineers, 2013, Visual Analytics: Seeking the Unknown, 46 (7), pp.22-29.

[93] M. Liu, J. Shi, Z. Li, C. Li, J.J.H Zhu, S. Liu: Towards better analysis of deep convolutional neural networks. IEEE TVCG 23 (1), 91–100. 2017http://dx.doi.org/10. 1109/IEEETVCG.2016.2598831

[94] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, M. Streit: Opening the black box: Strategies for increased user involvement in existing algorithm implementations. IEEE TVCG 20 (12), 1643–1652. 2014, http://dx.doi.org/10.1109/ IEEETVCG.2014.2346578.

[95] S. Han, J. Pool, J. Tran, Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems (NIPS), pp. 1135–1143 (2015)

[96] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (ICLR) (2018)

[97] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems 25 (NIPS 2012)

[98] J. Deng, W. Dong, R. Socher, L.-J Li,K. Li, and L. Fei-Fei Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255) (2009).

[99] C. M. Bishop. Pattern recognition and machine learning. Springer, 2006.

[100] K.-L. Du and M. N. S. Swamy, Neural Networks and Statistical Learning, DOI: 10.1007/978-1-4471-5571-3_2, Springer-Verlag London 2014

[101] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel: Backpropagation applied to handwritten zip code recognition. Neural Comput. 1(4), 541–551 (1989)

[102] G.E. Hinton, N. Srivastave, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov: Improving neural networks by preventing co-adaptation of feature detectors. In: arXiv:1207.0580 (2012).

[103] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng : TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[104] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell,: Caffe: Convolutional Architecture for Fast Feature Embedding, CoRR, abs/1408.5093,2014.

[105] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT press, 2016.

[106] K. P. Murphy. Machine Learning: A Probabilistic Perspective. The MIT Press, 2012.

[107] P. Dayan and L. F. Abbott. Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. MIT press, 2001.

[108] W. Winn, “A conceptual basis for educational applications of virtual reality,” Technical Publication R-93-9, Human Interface Technology Laboratory of the Washington Technology Center, Seattle:University of Washington, 1993.

[109] S. Zhang, J. Teizer, J.-K. Lee, C. M. Eastman, and M. Venugopal, “Building information modeling (bim) and safety: Automatic safety checking of construction models and schedules,”Automation in Construction, vol. 29, pp. 183–195, 2013.

[110] J. Deng, W. Dong, R. Socher, L.J Li, K. Li, L. Fei-Fei: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. pp. 248–255. IEEE (2009)

[111] H. Hattori, V. N. Boddeti, K.M. Kitani, and T. Kanade: Learning scene-specific pedestrian detectors without real data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3819–3827 (2015)

[112] X. Peng, B. Sun, K. Ali, and K. Saenko: Learning deep object detectors from 3d models. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1278–1286 (2015)

[113] H. Su, C.R. Qi, Y. Li, and L.J. Guibas: Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2686–2694 (2015)

[114] C. Chen, A. Seff, A. Kornhauser, and J. Xiao: Deepdriving: Learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2722–2730 (2015)

[115] N. Koenig, A. Howard: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. vol. 3, pp. 2149–2154. IEEE (2004)

[116] E. Todorov, T. Erez, and Y. Tassa: Mujoco: A physics engine for model-based control. In IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012)

[117] A.X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, and S. Song, H. Su et al: Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012 (2015)

[118] S. Choi, Q.Y. Zhou, S. Miller, V. Koltun: A large dataset of object scans. arXiv preprint arXiv:1602.02481 (2016)

[119] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla: Scenenet: Understanding real world indoor scenes with synthetic data. arXiv preprint arXiv:1511.07041 (2015)

[120] R. Mottaghi, M. Rastegari, A. Gupta, A. Farhadi:" what happens if..." learning to predict the effect of forces in images. arXiv preprint arXiv:1603.05600 (2016)

[121] W. Qiu, F. Zhong, Y. Zhang, S. Qiao,Z. Xiao, T. S. Kim, Y. Wang, A. Yuille," UnrealCV: Virtual Worlds for Computer Vision", ACM Multimedia Open Source Software Competition,2017

[122] https://www.vive.com/nz/support/vive-prohmd/category_howto/setting-up-room-scale-play-area.html

[123] A. V. Dam, A. S. Forsberg, D. H. Laidlaw, J. J. LaViola and R. M. Simpson, "Immersive VR for scientific visualization: a progress report," in *IEEE Computer Graphics and Applications, vol. 20, no. 6*, pp. 26-52, Nov.-Dec. 2000. doi: 10.1109/38.888006

[124] A. G. Abulrub, A. N. Attridge, and M. A. Williams, "Virtual reality in engineering education: The future for creative learning." 2011 IEEE Global Engineering Education Conference (EDUCON), pp. 751–757, 2011.

[125] W. Alhalabi, "Virtual reality systems enhance students achievements in engineering education." Behaviour & Information Technology, vol. 35, no. 11, pp. 919–925, Jul. 2016. [Online]. Available: https://doi.org/10.1080/0144929X.2016.1212931.

[126] R. Mintz, S. Litvak, and Y. Yair, "3d-virtual reality in science education: An implication for astronomy teaching," Journal of Computers in Mathematics and Science Teaching, vol. 20, no. 3, pp. 293–305, 2001. [Online]. Available: https://www.learntechlib.org/p/9543

[127] C. Dede, M. C. Salzman, and R. B. Loftin, "Sciencespace: Virtual realities for learning complex and abstract scientific concepts." in Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium, 1996, pp. 246–252

[128] H. Gerwin, K. Michal and P. Frits: "Towards intuitive exploration tools for data visualization in VR", VRST '02, November 11-13, 2002

[129] M. Bellgardt, C. Scheiderer and T. W. Kuhlen, "An Immersive Node-Link Visualization of Artificial Neural Networks for Machine Learning Experts," *2020 IEEE International Conference on Artificial Intelligence and Virtual*

Reality (AIVR), 2020, pp. 33-36, doi: 10.1109/AIVR50618.2020.00015.

[130] S.M. Lundberg and S.I.Lee. "A Unified Approach to Interpreting Model Predictions", 31st Conference on Neural Information Processing Systems (NIPS) Long Beach, CA, USA, 2017

[131] S. Chmiela, H.E. Sauceda, K.R. Müller, A. Tkatchenko: Towards exact molecular dynamics simulations with machine-learned force fields. Nat. Commun. 9(1), 3887 (2018)

[132] A.W. Thomas, H.R. Heekeren, K.R. Müller, W. Samek: Analyzing neuroimaging data through recurrent deep learning models. arXiv preprint arXiv:1810.09945 (2018)

[133] D. Wu, L. Wang, and P. Zhang: Solving statistical mechanics using variational autoregressive networks. Phys. Rev. Lett. 122(8), 080602 (2019)

[134] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym: NVIDIA tesla: a unified graphics and computing architecture. IEEE Micro 28(2), 39–55 (2008)

[135] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei: Largescale video classification with convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1725–1732 (2014)

[136] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436–444 (2015)

[137] LeCun, Y.A., Bottou, L., Orr, G.B., M¨uller, K.-R.: Efficient backprop. In: Montavon, G., Orr, G.B., M¨uller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 7700, pp. 9–48. Springer, Heidelberg (2012). https://doi.org/10.1007/ 978-3-642-35289-8 3

[138] W. Samek, T. Wiegand, K.R. Müller: Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. 2017. arXiv preprint arXiv:1708.08296

[139] A. Fernandez, F. Herrera, O. Cordon, MJ. Del Jesus, F. Marcelloni: Evolutionary fuzzy systems for explainable artificial intelligence: why, when, what for, and where to? IEEE Computational Intelligence Magazine. 2019

[140] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba: Network dissection: quantifying interpretability of deep visual representations. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6541–6549 (2017)

[141] J. Li, W. Monroe , D. Jurafsky: Understanding neural networks through representation erasure. arXiv preprint arXiv:1612.08220, 2016b.

[142] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, J. Clune: Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In: Advances in Neural Information Processing Systems (NIPS), pp. 3387–3395 (2016)

[143] A. Nguyen, J. Yosinski, J. Clune: Understanding neural networks via feature visualization: a survey. In: W. Samek, G. Montavon, A. Vedaldi, L.K. Hansen,K.R. M¨uller (eds.) Explainable AI. LNCS, vol. 11700, pp. 55–76. Springer, Cham (2019)

[144] G. Montavon, W. Samek, and K.R. M¨uller: Methods for interpreting and understanding deep neural networks. Digit. Signal Process. 73, 1–15 (2018)

[145] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.R. M¨uller, W. Samek: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLoS ONE 10(7), e0130140 (2015)

[146] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, K.R. M¨uller: Layer-wise relevance propagation: an overview. In: Samek, W., Montavon, G., Vedaldi, A., Hansen, L.K., M¨uller, K.-R. (eds.) Explainable AI. LNCS, vol. 11700, pp. 193–209. Springer, Cham (2019)

[147] L. Arras, G. Montavon, K.R. M¨uller, W. Samek: Explaining recurrent neural network predictions in sentiment analysis. In: EMNLP 2017 Workshop on

Computational Approaches to Subjectivity, Sentiment & Social Media Analysis (WASSA), pp. 159–168 (2017)

[148] J. Kauffmann, K.R. M¨uller, G. Montavon: Towards explaining anomalies: a deep Taylor decomposition of one-class models. arXiv preprint arXiv:1805.06230 (2018)

[149] S. Lapuschkin, S. W¨aldchen, A. Binder, G. Montavon, W. Samek, K.R. M¨uller: Unmasking clever hans predictors and assessing what machines really learn. Nat. Commun. 10, 1096 (2019)

[150] N. Agarwal, B. Bullins, and E. Hazan: Second order stochastic optimization in linear time. arXiv preprint arXiv:1602.03943, 2016.

[151] J. Martens: Deep learning via hessian-free optimization. In International Conference on Machine Learning (ICML), pp. 735–742, 2010

[152] R. Khanna, B. Kim, J. Ghosh, and O. Koyejo: Interpreting black box predictions using fisher kernels. arXiv preprint arXiv:1810.10118 (2018)

[153] P.W. Koh, P. Liang: Understanding black-box predictions via influence functions. In: International Conference on Machine Learning (ICML), pp. 1885–1894 (2017)

[154] B. Goodman, and S. Flaxman: European union regulations on algorithmic decision-making and a "right to explanation". arXiv preprint arXiv:1606.08813, 2016

[155] J. Li, W. Monroe, and D. Jurafsky: Understanding neural networks through representation erasure. arXiv preprint arXiv:1612.08220, 2016b.

[156] P.J. Kindermans, K. T. Schütt, M. Alber, K.R. Müller, D. Erhan, B. Kim, S. Dähne: Learning how to explain neural networks: patternnet and patternattribution. In: International Conference on Learning Representations (ICLR) (2018)

[157] A. Datta, S. Sen, and Y. Zick: Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In Security and Privacy (SP), 2016 IEEE Symposium on, pp. 598–617, 2016.

[158] P. Adler, C. Falk, S.A. Friedler, G. Rybeck, C. Scheidegger, B. Smith and S. Venkatasubramanian: Auditing black-box models for indirect influence. arXiv preprint arXiv:1602.07043, 2016.

[159] P. Dabkowski, and Y. Gal: Real time image saliency for black box classifiers. arXiv preprint arXiv:1705.07857, 2017.

[160] D. Erhan, Y. Bengio, A. Courville, and P. Vincent: Visualizing higher-layer features of a deep network. University of Montreal, 1341:3, 2009.

[161] R.R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra: Grad-cam: Why did you say that? arXiv preprint arXiv:1611.07450, 2016.

[162] M. Debruyne, M. Hubert, and J. A.K. Suykens: A Model selection in kernel based regression using the influence function. Journal of Machine Learning Research (JMLR), 9 (0):2377–2400, 2008

[163] A. Christmann, and I. Steinwart: On robustness properties of convex risk minimization methods for pattern recognition. Journal of Machine Learning Research (JMLR), 5(0):1007–1034, 2004.

[164] Y. Liu, S. Jiang, and S. Liao: Efficient approximation of cross-validation for kernel methods using Bouligand influence function. In International Conference on Machine Learning (ICML), pp. 324–332, 2014

[165] M. Wojnowicz, B. Cruz, X. Zhao, B. Wallace, M. Wolff, J. Luan, and C. Crable: "Influence sketching": Finding influential samples in large-scale regressions. arXiv preprint arXiv:1611.05923, 2016.

[166] M. Kabra, A. Robie, and K. Branson: Understanding classifier errors by examining influential neighbors. In Computer Vision and Pattern Recognition (CVPR), pp. 3917– 3925, 2015.

[167] I.J. Goodfellow, J. Shlens, and C. Szegedy: Explaining and harnessing adversarial examples. In International Conference on Learning Representations (ICLR), 2015

[168] G. Cadamuro, R.G. Bachrach, and X. Zhu: Debugging machine learning models. In ICML Workshop on Reliable Machine Learning in the Wild, 2016

[169] A. Guillaume and B. Yoshua: Understanding intermediate layers using linear classifier probes. arXivpreprint arXiv:1610.01644, 2016

[170] M. Raghu, J. Gilmer, J. Yosinski, and J. SohlDickstein: Svcca: Singular vector canonical correlation analysis for deep understanding and improvement. arXiv preprint arXiv:1706.05806, 2017.

[171] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba: Network dissection: Quantifying interpretability of deep visual representations. In Computer Vision and Pattern Recognition, 2017

[172] J. Johnson, A. Alahi and L. F.-Fei: Perceptual Losses for Real-Time Style Transfer and Super-Resolution, arXiv 2016

[173] L. A. Gatys, A. S. Ecker, and M. Bethge: A Neural Algorithm of Artistic Style. Preprint, submitted September 2, 2015. https://arxiv.org/abs/1508.06576

[174] G. Marti, F. Nielsen, P. Donnat, S. Andler:(2017) On Clustering Financial Time Series: A Need for Distances Between Dependent Random Variables. In: Nielsen F., Critchley F., Dodson C. (eds) Computational Information Geometry. Signals and Communication Technology. Springer, Cham. https://doi.org/10.1007/978-3-319-47058-0_8

[175] Fisher, A.J., & Kennedy, E.H. (2018). Visually Communicating and Teaching Intuition for Influence Functions. *The American Statistician, 75*, 162 - 172.

[176] Meissler, N; Wohlan, A; Hochgeschwender,N; Schreiber ,A. Using Visualization of Convolutional Neural Networks in Virtual Reality for Machine Learning Newcomers. In 2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR) (pp. 152-1526),2019.

[177] VanHorn, K. C; Zinn, M; Cobanoglu, M. C. Deep Learning Development Environment in Virtual Reality. CoRR, abs/1906.05925,2019.

[178] Schreiber, A; Bock, M. Visualization and Exploration of Deep Learning Networks in 3D and Virtual Reality. In International Conference on Human-Computer Interaction (pp. 206-211). Springer,2019.

[179] Kang, T.; Chae, M.; Seo, E.; Kim, M.; Kim, J. DeepHandsVR: Hand Interface Using Deep Learning in Immersive Virtual Reality. *Electronics* **2020**, *9*, 1863. https://doi.org/10.3390/electronics9111863