TOWARDS BETTER UNDERSTANDING OF DEEP LEARNING WITH VISUALIZATION

by

HAIPENG ZENG

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Supervised by Prof. Huamin Qu

October 2016, Hong Kong

TABLE OF CONTENTS

TOWARDS BETTER UNDERSTANDING OF DEEP LEARNING WITH VISUALIZATION

by

HAIPENG ZENG

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

ABSTRACT

Deep learning can learn representations of data for different kinds of tasks by using computational models with multiple processing layers. Remarkable progress has been made in detection and classification tasks in recent years. However, there is still no clear understanding of the inner working mechanisms. Usually, to get a better deep learning model, people have to undergo a substantial amount of trial-and-error procedures, which is very inconvenient and time-consuming. Consequently, there has been a dramatical interest in using visualization to help people better understand and train deep learning models intuitively. Existing work mainly focuses on three aspects, i.e., feature visualization, relationship visualization and process visualization, which show the clear advantages in helping understand the reasoning behind deep learning models.

In this survey, we first introduce the background and characteristics of deep learning and then give a comprehensive review of how visualization techniques are used to help understand and train deep learning models. Finally, we conclude the survey with a discussion of future research directions.

CHAPTER 1

INTRODUCTION

In this section, we first briefly introduce the background of deep learning, and then discuss the motivation behind using visualization to help understand deep learning. After that, an overview of this survey is given.

1.1 Background and Motivation

Deep learning, as a subfield of machine learning, allows computational models with multiple processing layers to extract features from raw data and discover the hierarchical representations needed for different kinds of tasks (LeCun et al., 2015). In recent years, it has achieved great progress in many Artificial Intelligence (AI) tasks, such as image recognition (He et al., 2015) (Krizhevsky et al., 2012), speech recognition (Mohamed et al., 2012) (Seide et al., 2011), and natural language processing (Mikolov et al., 2011) (Bordes et al., 2012), attracting a lot of attention.

Actually, deep learning has been studied for many years. However, few researchers paid much attention to it in the past because other machine learning models, such as SVMs, can outperform it. Also it is difficult to train a deep neural network well, and a deep model tend to yield worse results than a shallow model due to limited computation ability and the optimization challenges (Deng & Yu, 2014). It was not until 2006 Hinto et al. proposed the unsupervised pretraining technique using greedy layer-wise procedures that made training better deep neural networks possible (Hinton et al., 2006) (Bengio et al., 2007) (MarcAurelio Ranzato et al., 2007). Then the success of AlexNet (Krizhevsky et al., 2012) in the ImageNet challenge attracted great attention from both academia and industry. After that people began to realize the powerful capacity of deep learning. Generally speaking, three main factors contribute to the vigorous development of deep learning. The first is the powerful computing capability. In particular, GPU implementations make it possible to train a larger deep learning model in a relatively short time (Raina et al., 2009) (Ciresan et al., 2011). The second is the availability of large label data sets (Deng et al., 2009) (Lin et al., 2014). The last factor is the considerable advances in the availability of algorithms and techniques, such as the backpropagation algorithm (Rumelhart et al., 1988), dropout (Hinton et al., 2012) and rectified linear units (Glorot et al., 2011).

Although deep learning has achieved remarkable progress, there is still no clear understanding of the inner working mechanisms of deep learning models. The obstacle is mainly due to their complex network structures, millions of parameters and non-linear mathematical function components. Compared with many other machine learning models, such as SVMs and decision trees, deep learning works like a black box. Consequently, people do not know why it works, how to improve it and when it fails. Therefore, in order to get a better model, people have to work through a substantial amount of trial-and-error procedures, which is very inconvenient and time-consuming. Also, when deep learning models are applied to real scenarios, such as healthcare and insurance, people become concerned about their performance in the decision-making process. It is not easy for people to trust a black box model.

Recently, to better understand and train a deep learning model, researchers have proposed visualization techniques, in the hope of shedding light into the black box model and gaining valuable insight. For both newcomers and experts who want to learn more about deep learning, visualization can not only help them understand deep learning models intuitively, but also help them to tune a better model efficiently. For customers who adopt deep learning models to make decisions, visualization techniques could involve them in and thus make their decisions more interpretable.

This survey attempts to summarize existing work for visualization techniques used on deep learning. It examines how visualization can better help us understand and tune deep learning models, which can be further explored in the future work.

1.2 Overview

The remainder of this paper is organized as follows. In Chapter 2, we discuss the taxonomy in detail. In Chapter 3, a brief introduction to deep neural networks (CNNs and RNNs) is given. In Chapter 4, we describe some methods used for visualizing the features learned by deep learning models. In Chapter 5, we introduce visualization techniques used for visualizing the relationships in deep learning models. In Chapter 6, we introduce visualization techniques used for visualizing the process of deep learning models. Finally, in Chapter 7, we give a summary of this survey and point out some potential research directions for the future work.

CHAPTER 2

TAXONOMY

There can be many different taxonomies for visualization of deep learning. Considering what kinds of visualization techniques are used, potential classes could be grid-based techniques, network-based techniques and so on. While considering the targets that the visualization is used on, other potential taxonomies could be neuron visualization, layer visualization, edge (connection) visualization and so on. In this survey, we propose a taxonomy based on the challenges that deep learning faces and the purposes that visualization techniques serve.

In the deep learning field, to better understand a deep neural network, researchers are usually concerned about how the model works and how to improve it. Therefore, some of existing work focuses on visualizing which features are extracted by neurons in the network and how they relate to each other. This helps to understand what the model has learned and what the inner working mechanism is. Other visualization work concentrates on visualizing the whole training process and training information, which helps to design and train a better model. Therefore, based on the purposes the visualization used for, we categorize existing work into three classes, namely feature visualization, relationship visualization and process visualization, which are shown in Figure 2.1.

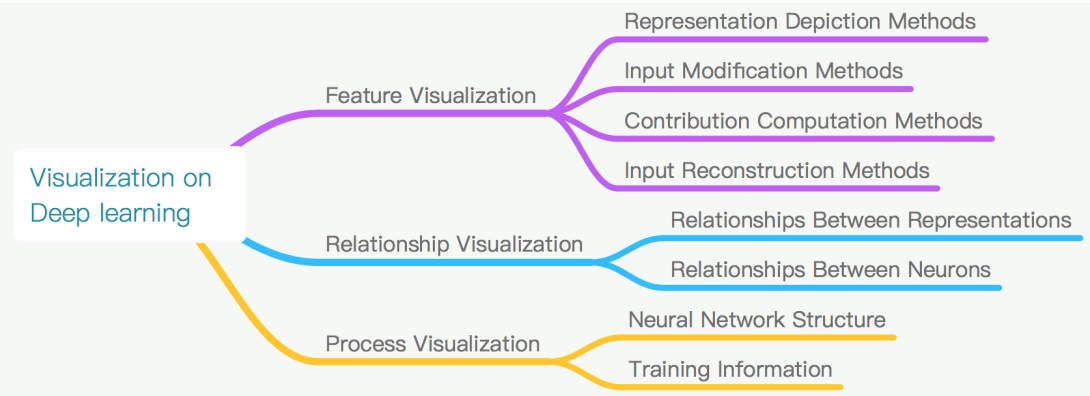


Figure 2.1: The Taxonomy

Feature Visualization: Feature visualization methods are used to visualize the features learned by the neurons of interest. These methods can be further categorized into four classes, namely representation depiction methods, input modification methods, contribution computation methods, and input reconstruction methods.

Relationship Visualization: In relationship visualization, we concentrate on the relationships between the learned representations and the relationships between neurons, where projection and clustering techniques are leveraged by visualization techniques such as scatter-plot and DAG-based visualization, respectively.

Process Visualization: We mainly focus on two aspects in process visualization: neural network structure visualization and training information visualization. Process visualization captures the whole working flow of a deep learning model.

There are various machine learning techniques and architectures in the deep learning field, which can be broadly categorized into three major classes, namely deep networks for unsupervised or generative learning, deep networks for supervised learning, and hybrid deep networks (Deng, 2012). However, this three-way categorization is too general and does not consider special structures of different neural networks, which is not suitable for our survey. Basically, to apply visualization techniques, we need to consider the structure of the networks carefully, since different structures could make a great difference. In general, neural networks can be categorized by their architectures, such as Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Deep Belief Networks (DBNs). Among them, CNNs and RNNs are the most widely used and have achieved numerous state-of-the-art results. To the best of our knowledge, many existing work about visualization is being applied to these two architectures owing to their prevalence and practicality. Therefore, in this survey, our examples are mostly related to these two architectures, especially for CNNs.

Finally, we give a table summarizing the related papers in this survey (Table 2.1).

Table 2.1: Visualization on Deep Learning

	Detail Categories	Related Paper
Feature Visualization	Representation Depiction Methods	(Karpathy, 2014a), (Yosinski et al., 2015), (Karpathy et al., 2015), (Strobelt et al., 2016), etc.
	Input Modification Methods	(Zeiler & Fergus, 2014), (Zhou et al., 2014), (Girshick et al., 2014), etc.
	Contribution Computation Methods	(Zeiler & Fergus, 2014), (Simonyan et al., 2013), (Springenberg et al., 2014), (Bach et al., 2015), (Zhou et al., 2015), (Bahdanau et al., 2014), (Socher et al., 2013), (Hermann et al., 2015), (Goyal et al., 2016), etc.
	Input Reconstruction Methods	(Long et al., 2014), (Erhan et al., 2009), (Simonyan et al., 2013), (Alexander et al., 2015), (Mahendran & Vedaldi, 2015), (Yosinski et al., 2015), (Mahendran & Vedaldi, 2016), (Dosovitskiy & Brox, 2015), etc.
Relationship Visualization	Relationships Between Representations	(Maaten & Hinton, 2008), (Cho et al., 2014), (Karpathy, 2014b), (Rauber et al., 2016), etc.
	Relationships Between Neurons	(Liu et al., 2016), (Rauber et al., 2016), etc.
Process Visualization	Neural Network Structure (Model)	(Karpathy, 2014a), (Yosinski et al., 2015), (Smilkov et al., 2015), (Harley, 2015), (Chung et al., 2016), (Liu et al., 2016), (Bruckner, 2014), (Google, 2015), etc.
	Training Information (Data)	(Google, 2015), (Bruckner, 2014), (Smilkov et al., 2015), (Skymind, 2013), (Chung et al., 2016), etc.

CHAPTER 3

DEEP LEARNING MODELS

In this chapter, we briefly introduce some basic knowledge about deep learning models, focusing on convolutional neural networks and recurrent neural networks, which are useful for subsequent chapters.

3.1 Concepts and Foundations

Deep learning models usually refer to neural networks with multiple hidden layers of neurons between the input and output layers (Bengio, 2009). A deep neural network (DNN) is a typical deep learning model. As shown in Figure 3.1, the leftmost layer is called the input layer; the rightmost layer is called the output layer; the middle layer is called the hidden layer. We call a node in the network a neuron or unit, just like a biological neuron. It performs a scalar product with its input and weights, added by the bias and then the result is applied with an activation function (Figure 3.2). The final value of the neuron is called as an activation and a vector of activations in a hidden layer is a representation of an original input.

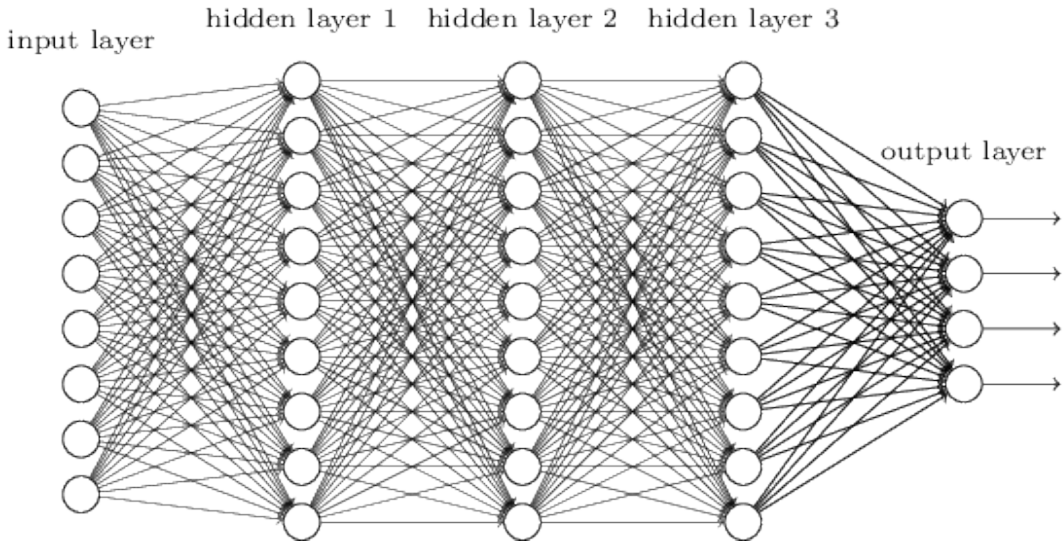


Figure 3.1: A typical architecture of DNN.[1]

[1]http://neuralnetworksanddeeplearning.com/chap6.html

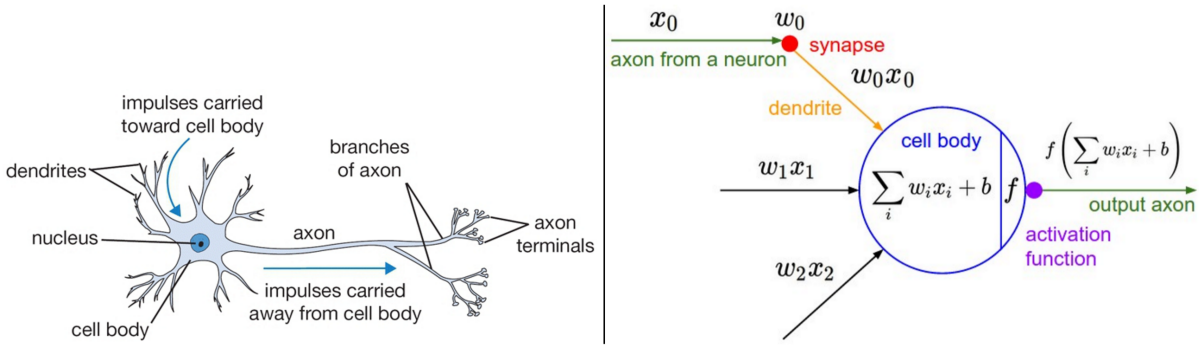


Figure 3.2: A sketch of a biological neuron (left) and a mathematical model (right).[2]

Here we make it more formal in mathematics based on the notation from Nielsen's book (Nielsen, 2015). w_{jk}^l is used to stand for the weight for connection from the k-th neuron in the $(l-1)$-th layer to the j-th neuron in the l-th layer. Similarly, b_j^l is used to represent the bias of the j-th neuron in the l-th layer. While z_j^l is used for the linear weight computation of a former layer and a_j^l is used for the activation of the j-th neuron in the l-th layer. In addition, σ is used for an activation function, which could be a sigmoid function, a tanh function, a ReLU (rectifier linear unit) function and so on. While in the last layer, usually a softmax function is applied. The computation equations are shown as follows:

$$\begin{cases} z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \\ a_j^l = \sigma(z_j^l) \end{cases} \tag{3.1}$$

They can be written in more elegant and compact vectorized form:

$$\begin{cases} z^l = w^l a^{l-1} + b^l \\ a^l = \sigma(z^l); \quad l = 2, 3, ...L \end{cases} \tag{3.2}$$

where a^1 indicates the input and a^L indicates the output.

We can define quadratic cost function for a single training example x:

$$C = \frac{1}{2}\|y - a^L\| = \frac{1}{2}\sum_j (y_j - a_j^L)^2 \tag{3.3}$$

where $a^L = a^L(x)$ is the actual output from the network and $y = y(x)$ is the corresponding desired output. Meanwhile, the cost function could actually be defined into other forms, such as the cross-entropy cost function.

[2]http://cs231n.github.io/neural-networks-1/

It is easy to get four fundamental equations behind the backpropagation (Nielsen, 2015):

$$\begin{cases} \delta^L = \nabla_a C \odot \sigma'(z^L) \\ \delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \\ \dfrac{\partial C}{\partial b_j^l} = \delta_j^l \\ \dfrac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \end{cases} \tag{3.4}$$

Where δ indicates the error and $\odot$ indicates the Hadamard product.

To train the model, usually we use the backpropagation algorithm and the stochastic gradient descent. Here we show the procedure for a given mini-batch of m training examples (Nielsen, 2015):

1. Input a set of training examples;
2. For each training example x: Set the corresponding input activation $a^{x,1}$, and perform the following steps:
   - Feedforward: Compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$ for each $l = 2, 3, ..., L$.
   - Output error $\delta^{x,L}$: Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \delta'(z^{x,L})$.
   - Backpropagation: Compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \delta'(z^{x,l})$ for each $l = L - 1, L - 2, ..., 2$.
3. Gradient descent: Update the weights and the biases according to the rule for each $l = L, L - 1, ..., 2$.

$$\begin{cases} w^l \to w^l - \dfrac{\eta}{m} \sum\limits_x \delta^{x,l} (a^{x,l-1})^T \\ b^l \to b^l - \dfrac{\eta}{m} \sum\limits_x \delta^{x,l} \end{cases} \tag{3.5}$$

   where η is the learning rate.

3.2 Convolutional Neural Networks

Convolutional neural networks are a special type of network, which are usually used for image-related tasks. Here we introduce some important components of a CNN.

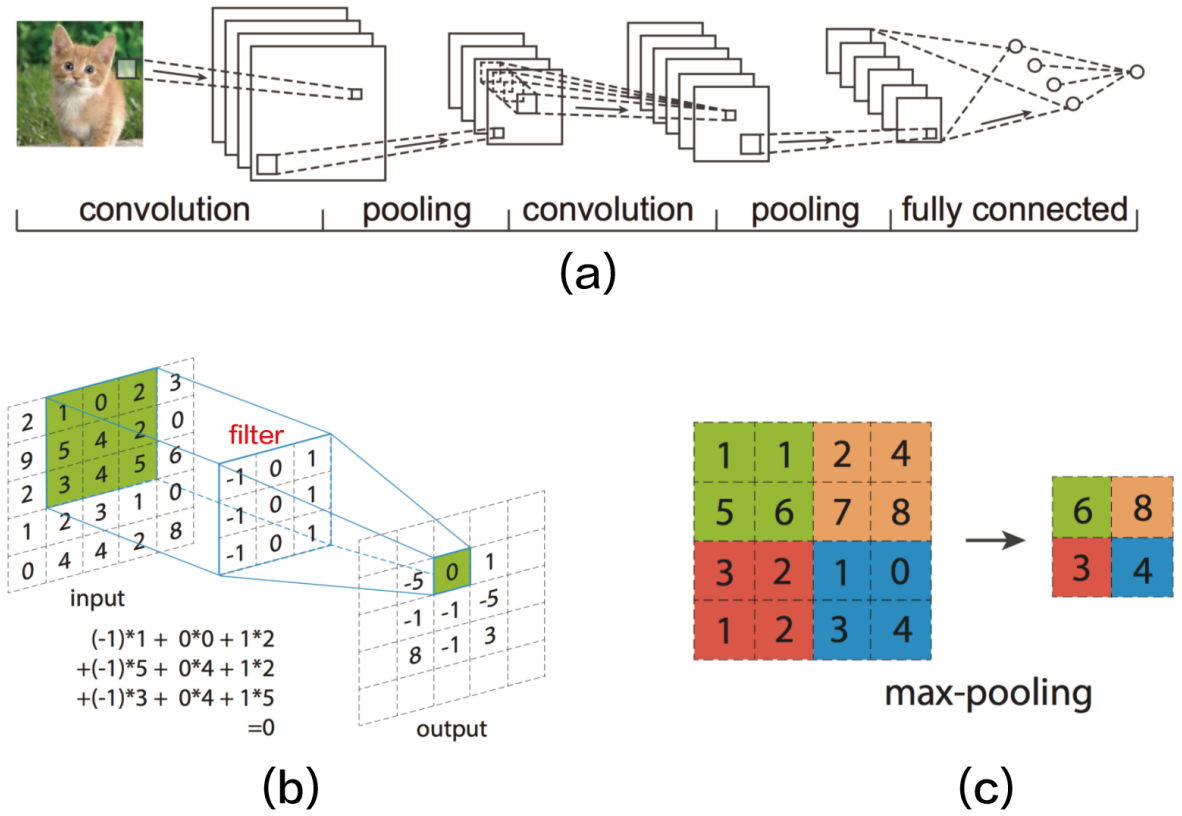


Figure 3.3: (a) A typical architecture of CNN. (b) convolution. (c) max-pooling.[3]

Architecture. As shown in Figure 3.3(a), a typical architecture of CNN usually contains convolution layers, pooling layers and fully connected layers.

Convolution. A convolution operation is shown in Figure 3.3(b), which calculates the weighted sum of the input pixels covered by the window. Usually we use a sliding window called a filter or kernel to go over an image and do the convolution operation, then we can get a corresponding map. We can use different filters to get different maps.

Activation Function. After a convolution operation, an activation function, such as a sigmoid, tanh or ReLU (rectifier linear unit) function, is applied to each neuron. We usually call the result an activation and a set of activations in the same map the activation map or the feature map.

Pooling. A max-pooling operation is shown in Figure 3.3(c), which down samples the image feature from the previous layer. We apply a pooling operation to each feature map, so that we can get the corresponding smaller map with some level of translation invariance.

Fully connected layer. At the end of CNNs, there are usually fully connected layers,

[3]https://arxiv.org/pdf/1604.07043v3.pdf

which are similar to DNN. In the last layer, we perform a softmax function to normalize them to add up to 1. For the image classification, the output of the network is a vector of class probabilities.

3.3 Recurrent Neural Networks

Recurrent Neural Networks are another special type of network with a loop, which are usually used for handling sequential data. Here we introduce some important characteristics of an RNN.

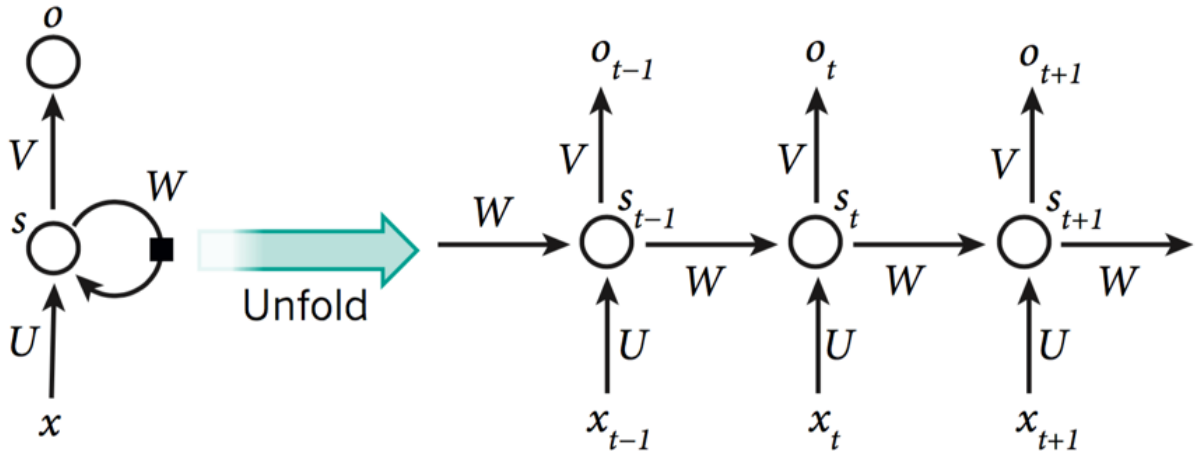


Figure 3.4: A recurrent neural network and the unfolding computational graph.[4]

Architecture. As shown in Figure 3.4, a RNN could be unfolded into a computational graph, which is easier to understand. x_t, s_t and o_t are the input, hidden state ("memory") and output at time step t respectively. The formulas are given as follow:

$$\begin{cases} s_t = f(Ux_t + Ws_{t-1}) \\ o_t = softmax(Vs_t) \end{cases} \tag{3.6}$$

Where f is a non-linear function, such as a tanh function and a ReLU function. It is generally believed that RNNs have a memory which can capture information calculated before, but they have only very limited memory due to the gradient vanish or explode problem, which can only look back a few steps.

LSTMs. Long Short Term Memory networks (LSTMs) are special kinds of RNNs, which use long short-term cells to help the learning of long-term dependencies in RNNs. As shown in Figure 3.5, Figure 3.5(a) is a standard RNN, while Figure 3.5(b) is a LSTM.

[4]http://www.nature.com/nature/journal/v521/n7553/pdf/nature14539.pdf

The main difference is about the calculation of the hidden state, where three gates (input gate, forget gate and output gate) are introduced into LSTM. Here are the changed formulas for LSTM:

$$\begin{cases} f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \\ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t = o_t * tanh(C_t) \end{cases} \tag{3.7}$$

Where f_t, i_t, $\widetilde{C}_t$, C_t, o_t and h_t represent the forget gate, input gate, new candidate, new cell state, output gate and hidden state respectively.

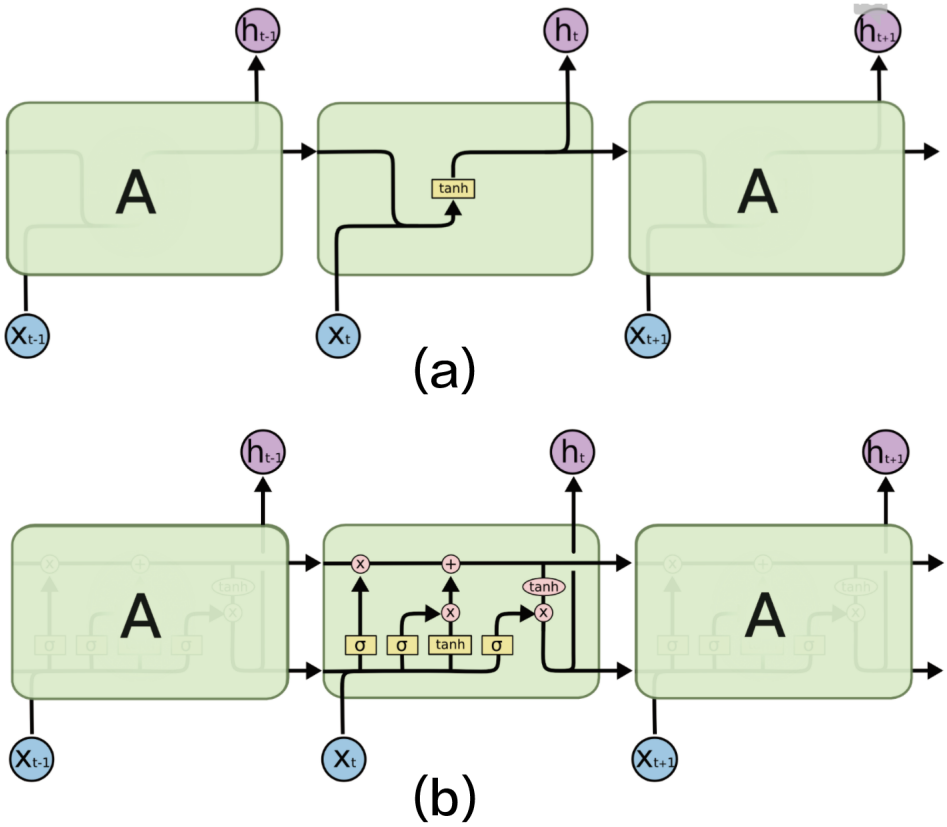


Figure 3.5: a) A standard RNN with a single layer; b) A LSTM with four interacting layers.[5]

[5]http://colah.github.io/posts/2015-08-Understanding-LSTMs/

CHAPTER 4

FEATURE VISUALIZATION

In this chapter, we introduce different visualization methods to explore what the neural networks have learned. There has been research on visually understanding how a deep learning model works, especially what roles neurons play in and what features they extract. These existing work could be further categorized into four classes, based on their goals and the algorithms involved: representation depiction methods, input modification methods, contribution computation methods, and input reconstruction methods. In the following sections, we will introduce them one by one.

4.1 Representation Depiction Methods

As previously mentioned, a representation for an input refers to a vector of activations in a hidden layer of a deep neural network. To unfold what the neurons have learned, the most intuitive way is to visually depict the representation directly.

Activation Map (Feature Map) In CNNs, an activation map is a representation for an input, which can be directly shown as a bitmap. For example, Karpathy proposed a javaScript library called ConvNetJs, which uses a single browser to train and visualize neural networks (Karpathy, 2014a). We can directly observe the changes of each activation map during the training process (Figure 4.1). Similarly, Yosinski et al. proposed an interactive visualization system, which helps deal with image and webcam input in real time with a trained model (Yosinski et al., 2015). As shown in Figure 4.2, for each input, we can observe the activation maps in each layer in the middle part, and a deconv image, as well as top nine images from the training set that result in the highest activations for the selected channel. It is very intuitive to locate an activation map of interest since all activation maps are changing in real time when the input object is moving. In this case (Figure 4.2), we find that the activation map indicated by a green rectangle is highly related to face detection.

Hidden State In RNNs, hidden states also refer to representations for inputs. It is widely believed that the hidden states are capable to capture the important information

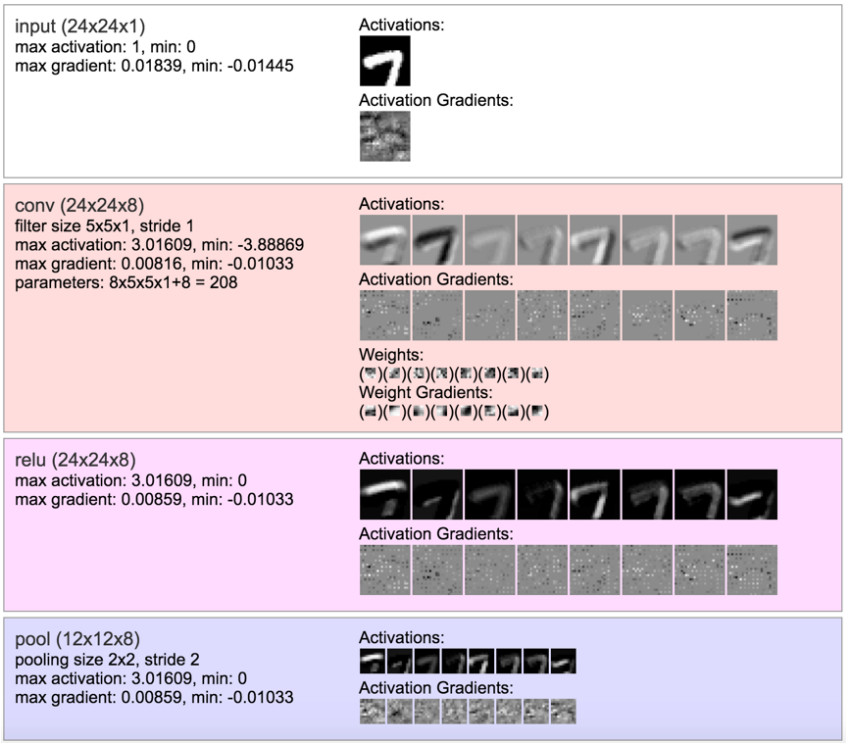


Figure 4.1: A snippet of ConvNetJS MNIST demo. Image courtesy of (Karpathy, 2014a).

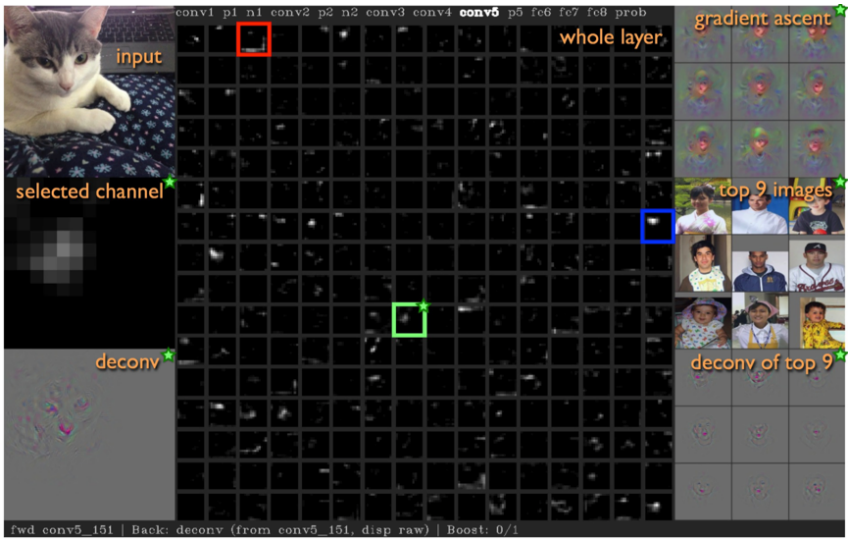


Figure 4.2: Deep visualization system with different components. Image courtesy of (Yosinski et al., 2015).

from the input. However, to trace what they have captured in a neural network is rather difficult. Karpathy et al. used heatmap to show activations in text input (Karpathy et al., 2015), which reveals that hidden state captures the structure of input text, such as length of lines and quotes (Figure 4.3). Strobelt et al. visualized hidden state dynamics by adopting parallel coordinates (Strobelt et al., 2016). In their system, users can freely select a range of text and those texts with similar hidden state patterns in the dataset

would be shown in the match view (Figure 4.4).

Directly visualizing representations can reveal what neurons have learned in a neural network to some extent; however it may be difficult to be explained in some cases. Therefore, researchers try to link inputs with representations, attempting to find methods with high interpretability.

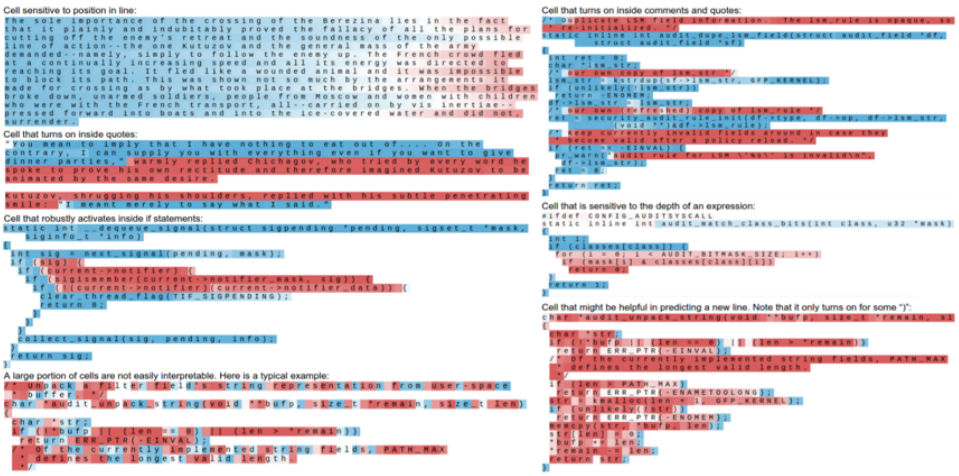


Figure 4.3: Interpretable activations of texts in Linux Kernel and War and Peace LSTMs. Blue is for high activate value and red is for low value. Image courtesy of (Karpathy et al., 2015).

Figure 4.4: The interface of LTSMVis. Users can select a range of text in the select view and similar hidden state patterns are shown in the match view. Image courtesy of (Strobelt et al., 2016).

4.2 Input Modification Methods

Input modification methods refer to those methods where we modify the input and then measure the changes of the output or activations in hidden layers (Grün et al., 2016). We can therefore observe how the input affects the output. Here we give some examples.

The occlusion method proposed by Zeiler et al. aims to find out which part of an image is important to the classification result (Zeiler & Fergus, 2014). In their method, a gray square is used to systematically occlude different parts of the input images and then measure the changes in activations. Any change in activations can reflect the importance of an area on an image. If the area is covered up and activations change a lot, it means the area is important and affects the classification result tremendously. If the area is covered up but does not affect the activations too much, it means the area is unimportant to the classification result. As Figure 4.5(b) shows, the heatmap reflects changes in activations with the red color meaning high value and the blue color meaning low value. It is obvious that when the face of the dog is covered up, the activations in the middle turn blue (Figure 4.5(b)) and the probability of classifying it as a dog is very low (Figure 4.5(c) and (d)), which means the face of the dog is very important in the classification result. In other words, it is the feature learned by the neural networks. However, Zeiler et al. used a mono colored gray square, which might affect the result due to sensitivity of CNNs to edges. Zhou et al. extended their work by using a randomized pixel values patch (Zhou et al., 2014), which is shown in Figure 4.6. A small randomized pixel values patch is contained in each sliding-window stimuli (Indicated here as the red arrow in Figure 4.6(a)). The highlighted parts of the middle and right images show their sensitivity to the sliding window stimuli (Figure 4.6(b) and (c)).

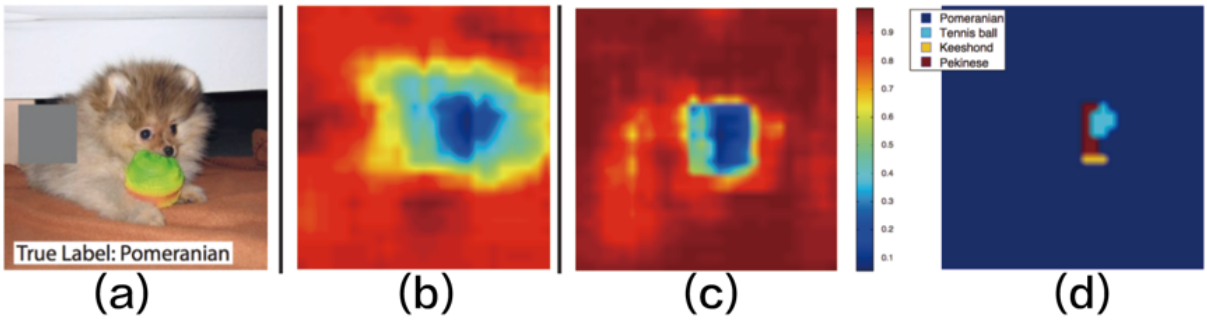


Figure 4.5: (a) an input Image with a gray square; (b) a heatmap for strongest feature map in layer 5, where heatmap value indicates total activation for each position of the gray scale; (c) a heapmap of correct class probability, as a function of the occluder position; (d) the most likely label for each position of the occluder. Image courtesy of (Zeiler & Fergus, 2014).

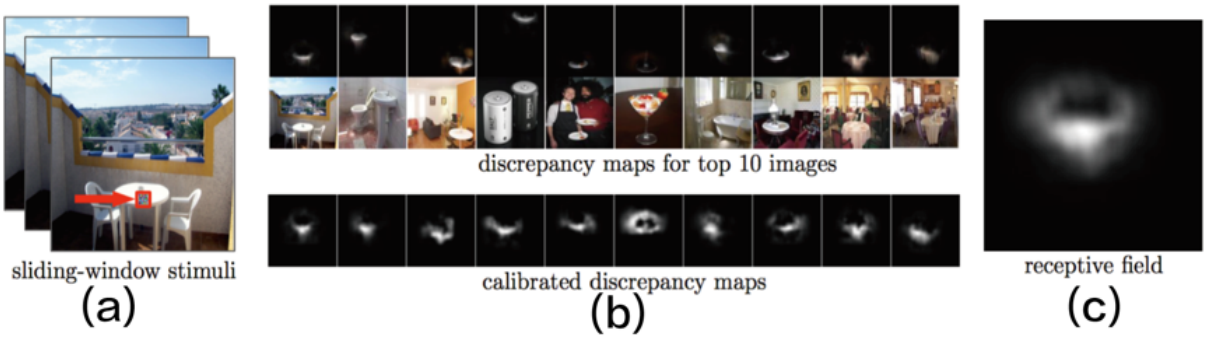


Figure 4.6: (a) images with a sliding-window stimuli (a small randomized patch) (b) discrepancy maps show the activation response of the sliding-window stimuli; (c) the actual receptive field. Image courtesy of (Zhou et al., 2014).

There is another method proposed by Girshick et al., which uses different proposed regions of the image as input instead modifying the original image (Girshick et al., 2014). They first need to rescale the proposed regions to the same size of the original image and then put them into the CNNs model. They then measured the response of neurons of interest to find out what proposed regions of the image activate neurons most. As shown in Figure 4.7, the neuron in the first row fires on human faces, while the neuron fires on dog faces and dot arrays in the second row. The other neurons are interested in a red blob, text, light dot and so on. This method is simple and intuitive which has been successfully applied to CNNVis (Liu et al., 2016). However, it rescales different proposed regions regardless of the size and aspect ratio and does not consider the location of proposal regions in the whole image, which may affect the final result.

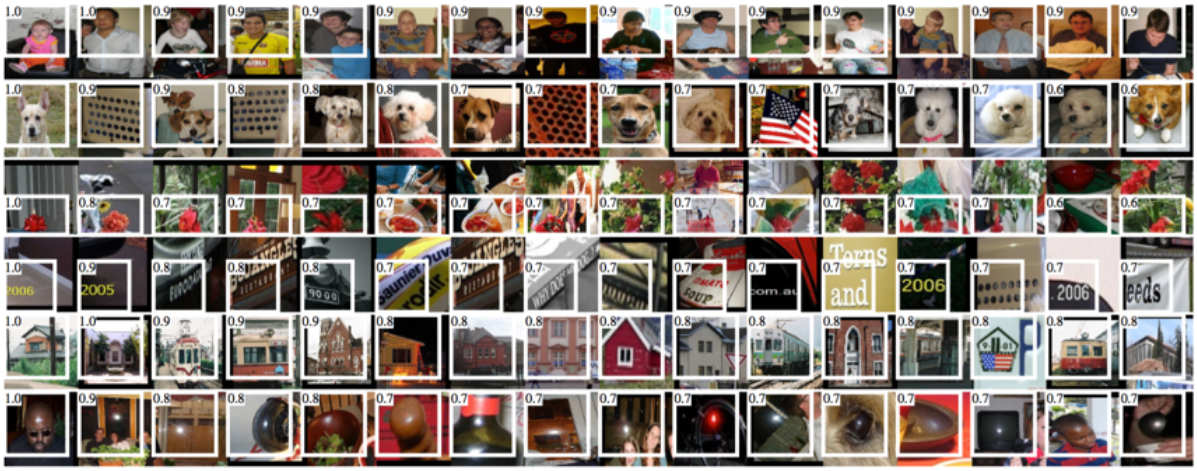


Figure 4.7: Top regions for six units in pool 5. Activation values and receptive fields are shown in white rectangle. Different units are interested in different objects, such as people (row 1), text (row 4) and specular reflections (row 6). Image courtesy of (Girshick et al., 2014).

4.3 Contribution Computation Methods

Contribution computation methods refer to the methods which compute the contributions of origin image pixels to the activation of interest (Grün et al., 2016). Usually, starting from the activation of interest, we compute the contribution of each neuron in the lower layer to this activation interactively until the input layer is reached. In this way we get a visualization of the features which are most related to the activation of interest.

Zeiler et al. proposed a multi-layered deconvolutional network based on their previous work (Zeiler et al., 2011), in which the activations are projected from the original feature space to the input space (Zeiler & Fergus, 2014). However, their method only focus on a single activation and cannot visualize the joint activity in a layer. The result (Figure 4.9) shows the complex invariances learned by the network layers, from which we can get some interesting insight. For example, the images in row 1, column 2 of layer 5 (inside the red rectangle), show the neurons focus on the background (grass) instead of the foreground.

While Simonyan et al. proposed a backpropagation method based on Baehrens et al. 's work (Baehrens et al., 2010), which computes the derivative of the class score in regards to the input image (Simonyan et al., 2013). As shown in Figure 4.8, the images above are the input images, while the images below are the corresponding saliency maps, from which we can see which parts are responsible for the class result.

Figure 4.8: The images above show the input images (top-1 predicted class in ILSVRC-2013), and below show the corresponding saliency maps. Image courtesy of (Simonyan et al., 2013).

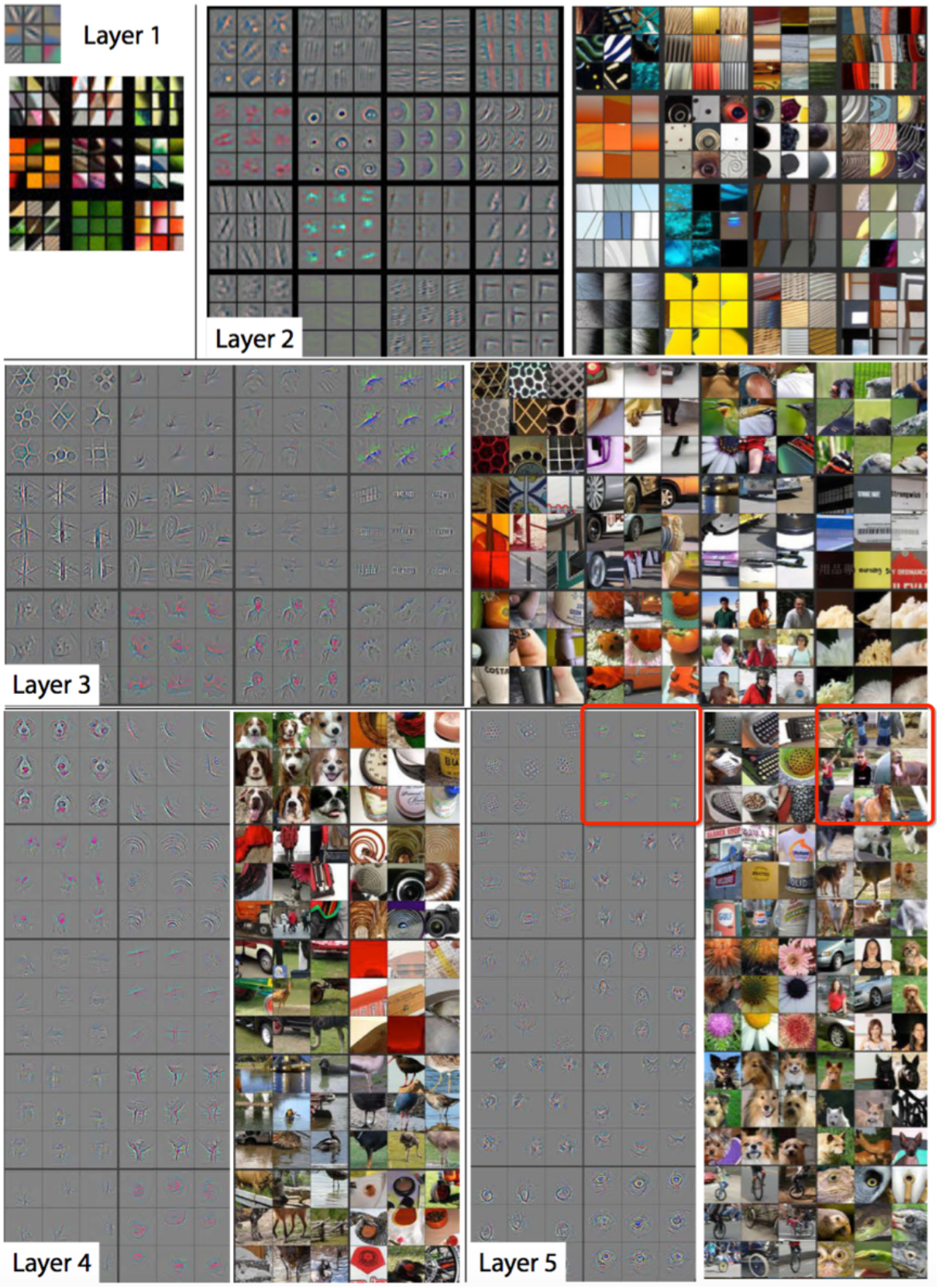


Figure 4.9: Visualization of features in a fully trained model using a deconvolutional network approach. In layers 2-5, the top nine activations in a random subset of feature maps across the validation data are shown, projecting down to the input image pixel space. Image courtesy of (Zeiler & Fergus, 2014).

Based on Zeiler et al., and Simonyan et al.'s work, Springenberg et al. proposed a method called Guided Backpropagation (Springenberg et al., 2014), which makes the projection clearer. However, their method is only used for CNNs without max pooling layers. Figure 4.10 shows the visualization result of guided backpropagation and the image generated is more meaningful and clearer than previous work. It is worth noting that the difference among these three methods falls in how they propagate the contribution values through ReLU and the convolutional layers (Grün et al., 2016).

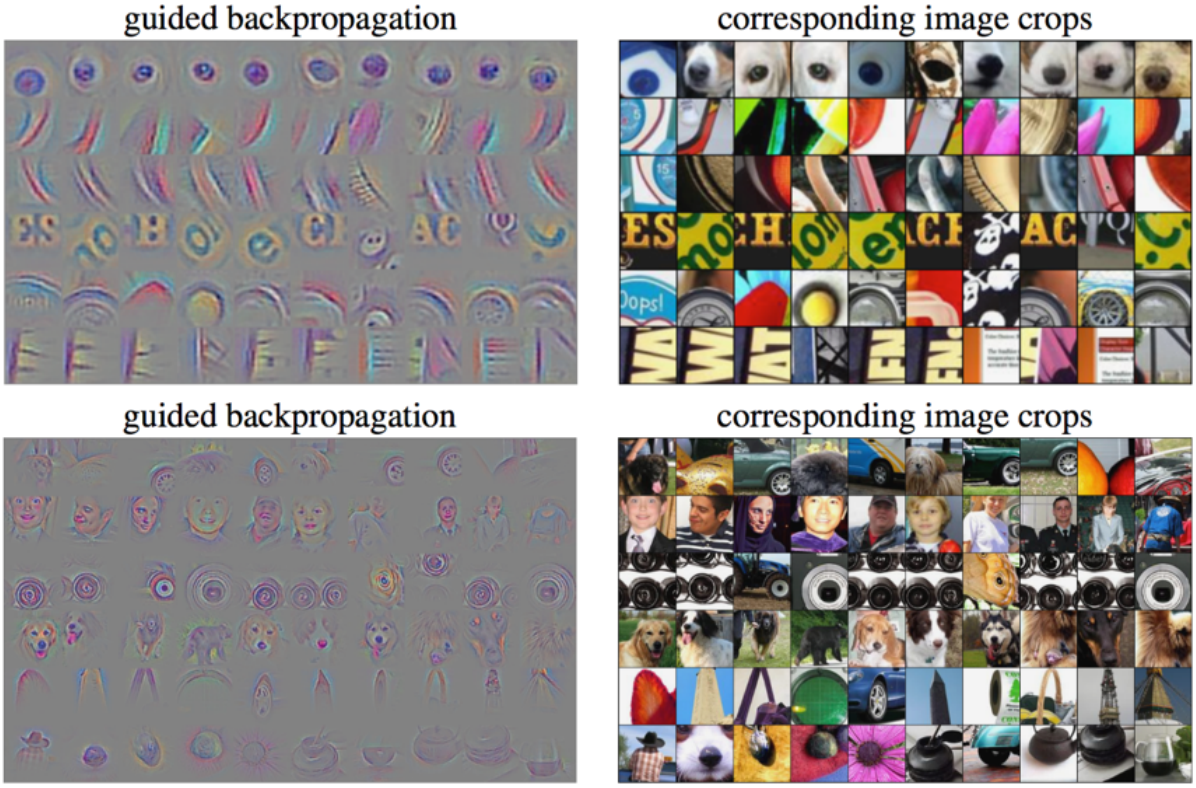


Figure 4.10: Guided backpropagation results of top 10 images patches for 6 units in layer conv6 (top) and layer conv9 (bottom). Image courtesy of (Springenberg et al., 2014).

After that, Bach et al. proposed a more general method called relevance propagation (Bach et al., 2015). The contributions of each pixel to the final result is visualized by using pixel-wise decomposition (Figure 4.12). Grün et al. has compared the effect of these four methods mentioned above (Grün et al., 2016). It is obvious that the guided backpropagation has the sharpest visualization result in Figure 4.11.

Another method constructing a class activation map is proposed by Zhou et al. (Zhou et al., 2015), which uses global average pooling on convolutional layer feature maps to generate a class activation map showing which parts of the input image are responsible for the output class of interest. The process is shown in Figure 4.13.

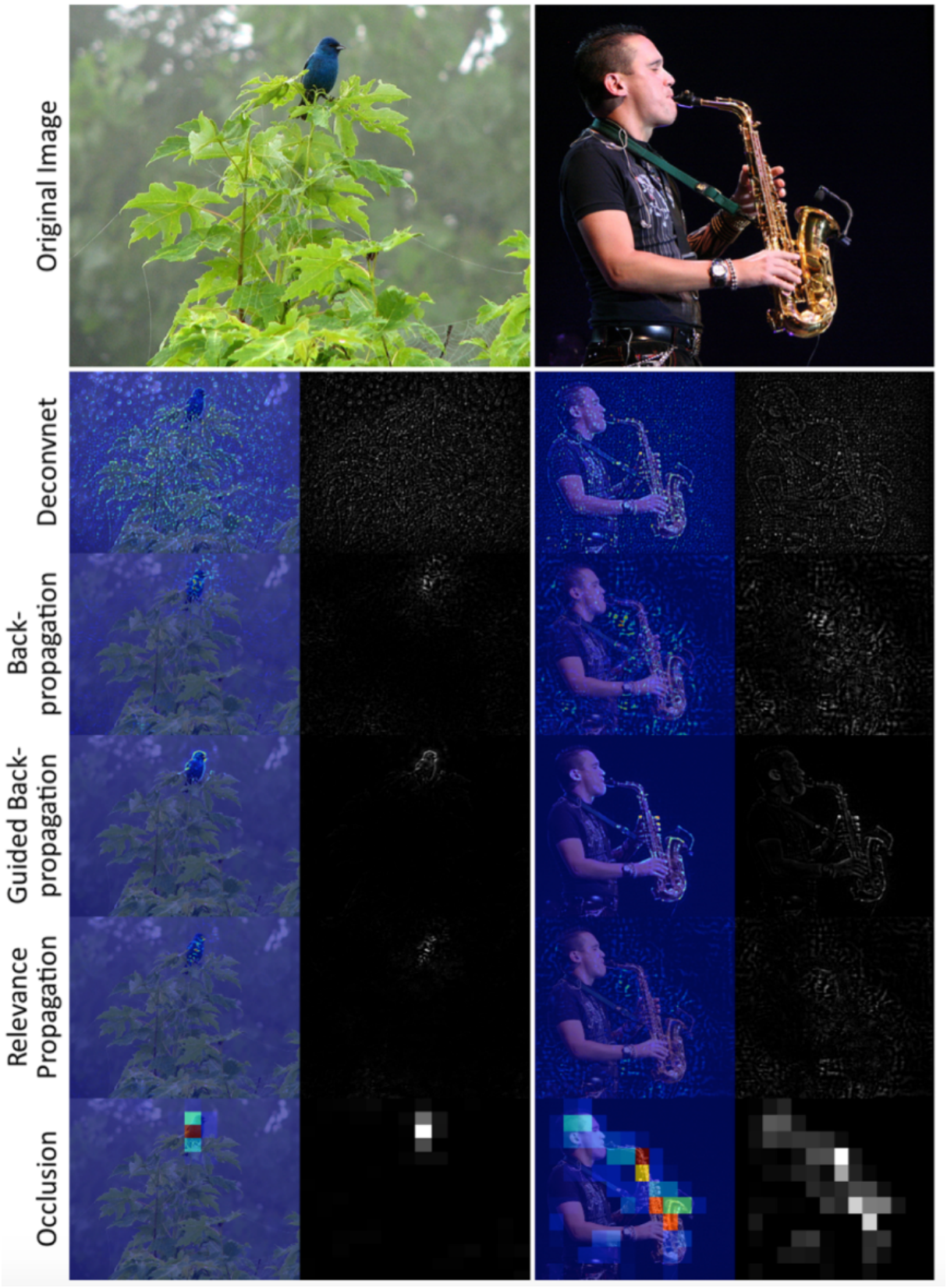


Figure 4.11: Comparation of different visualization methods for two images. Image courtesy of (Grün et al., 2016).

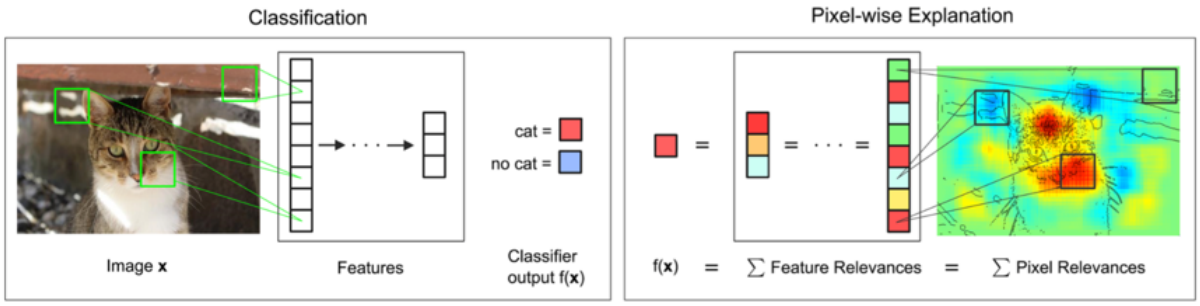


Figure 4.12: The process of the pixel-wise decomposition visualization. The heatmap shows the contributions of single pixels to the prediction. Image courtesy of (Bach et al., 2015).

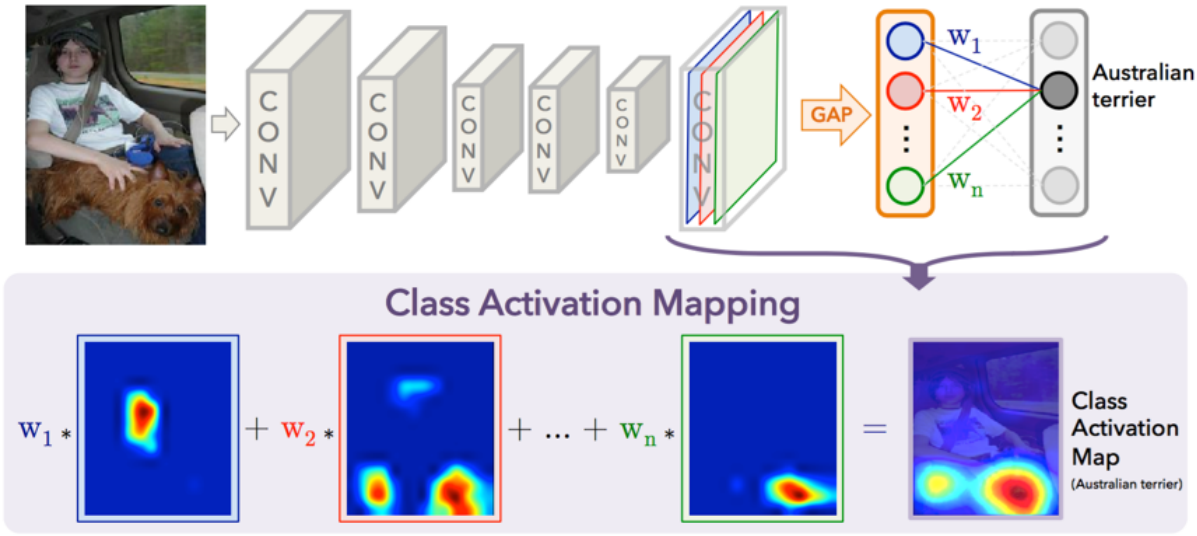


Figure 4.13: The process of class activation mapping. The class activation map highlights the class-specific discriminative regions. Image courtesy of (Zhou et al., 2015).

These examples above are all about images. Besides, there are some work on texts. For example, Socher et al. adopted a tree structure to analyze the sentiment of a sentence, showing how each word contributes the final result (Socher et al., 2013), as shown in Figure 4.14. Bahdanau et al. visualize the attention weight matrix, which represents the contribution of each French word to its corresponding English one in translation (Figure 4.15(a)) (Bahdanau et al., 2014). Hermann et al. adopted heatmap to highlight the important parts that respond different queries (Figure 4.15(b)) (Hermann et al., 2015). Meanwhile, it is worth mentioning that input modification methods and contribution computation methods have been used in a visual answer system to show the inference process (Goyal et al., 2016). For example, in Figure 4.16, important words in text and pixels in images to responsible for answering the question are both highlighted.

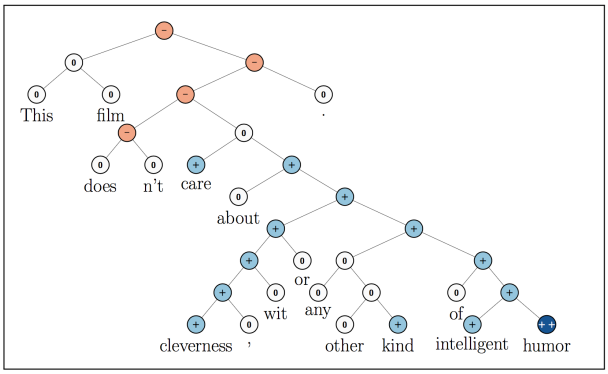


Figure 4.14: Tree structure to interpreter the sentiment of a sentence. Five sentiment classes, very negative to very positive (–, -, 0, +, ++). Image courtesy of (Socher et al., 2013).

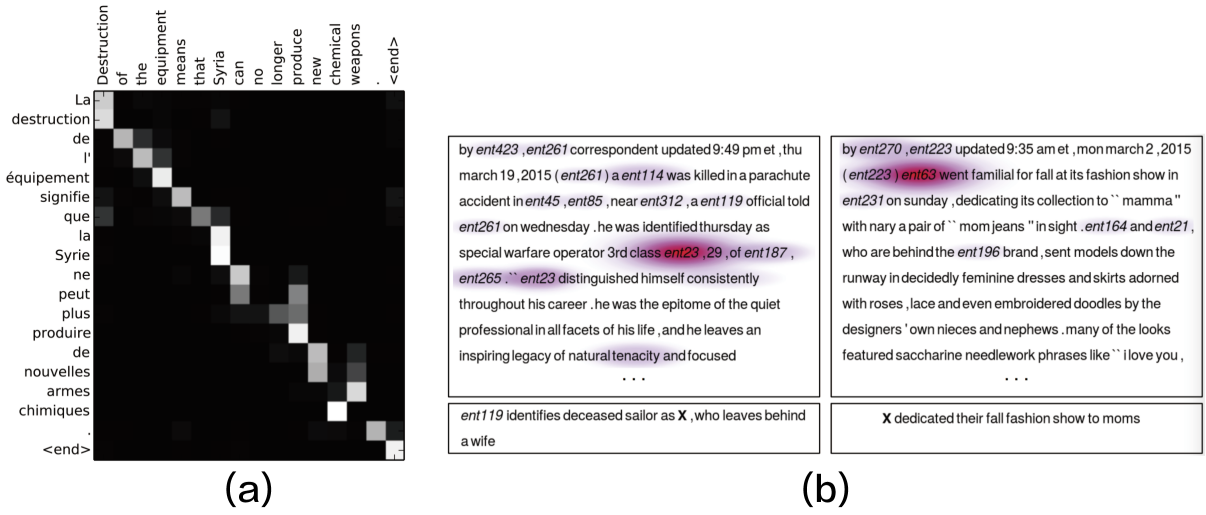


Figure 4.15: (a) Visualizing the attention weight matrix in language translation (Bahdanau et al., 2014). (b) Attention heat maps for answering queries (Hermann et al., 2015).

Figure 4.16: Examples of interpreting a visual question answering model. Image courtesy of (Goyal et al., 2016).

4.4 Input Reconstruction Methods

Input reconstruction methods refer to the methods of reconstructing an image based on representations in a neural network (Grün et al., 2016). It is widely believed that representations capture important information of input. Therefore, the reconstructing input actually reveals what the neural network learns to some extent. Some papers (Nguyen et al., 2016) (Mahendran & Vedaldi, 2016) (Liu et al., 2016) group these methods into two groups, namely activation maximization and code inversion, which is not comprehensive. A more general taxonomy is given by (Grün et al., 2016), which is based upon the methods used, namely replacement, gradient descent and generative networks. Here we adopt the latter one, since activation maximization and code inversion can be categorized into using gradient descent to reconstruct the input images.

Based on the HOGgles approach, instead of pair dictionary learning, Long et al. used replacement with top-k nearest neighbors in feature space to visualize the feature detected by a CNN (Figure 4.17) (Long et al., 2014). This method is called the replacement method.

Figure 4.17: Replacement with top-k nearest neighbors. Image courtesy of (Long et al., 2014).

As for the gradient method, we consider two types, activation maximization and code inversion. Activation maximization is the method which aims to find an image that maximally activates the neuron of interest, revealing what features the neuron detects. This method can be used for both each of the hidden neurons (Erhan et al., 2009) and the output neurons (Simonyan et al., 2013). The result of Simonyan et al. is shown in Figure 4.18. We can see some unclear corresponding objects in the images, where there are some repeats. It is worth mentioning that a famous application called deep dream

(inceptionism) adopts a similar idea (Alexander et al., 2015). For image input, it allows a trained deep learning model to amplify the feature selected by users or enhance whatever it has detected, which generates some fancy images (Figure 4.19). More interestingly, images with art styles can be generated if the algorithm is applied interactively with zooming after each iteration (Figure 4.20). Code inversion is another method aiming to synthesize an image starting from the encoded image representation. Mahendran et al. used a gradient descent optimization to invert representations (Mahendran & Vedaldi, 2015). Some results are shown in Figure 4.21. Since the function is not uniquely invertible, there are some different alternatives, from which we can observe the invariances. It is worth noting that to achieve recognizable images, both activation maximization and code inversion need to incorporate natural image priors into the optimization process. Different regularization techniques have been discussed in (Yosinski et al., 2015) and (Mahendran & Vedaldi, 2016).

Figure 4.18: Reconstructing images with activation maximization method for three classes, i.e. goose, ostrich and limousine. Image courtesy of (Simonyan et al., 2013).

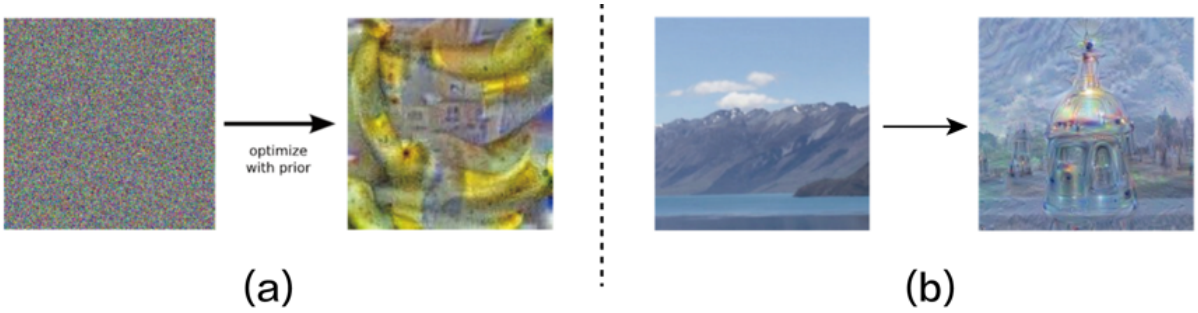


Figure 4.19: (a) generate a banana image; (b) generate an image with towers and pagodas. Image courtesy of (Alexander et al., 2015).

The last method proposed by Dosovitskiy et al. is to use a generative network to reconstruct the input image from the representations of different layers, which need to

Figure 4.20: Images generated by iterations and zooming from random noise. Image courtesy of (Alexander et al., 2015).

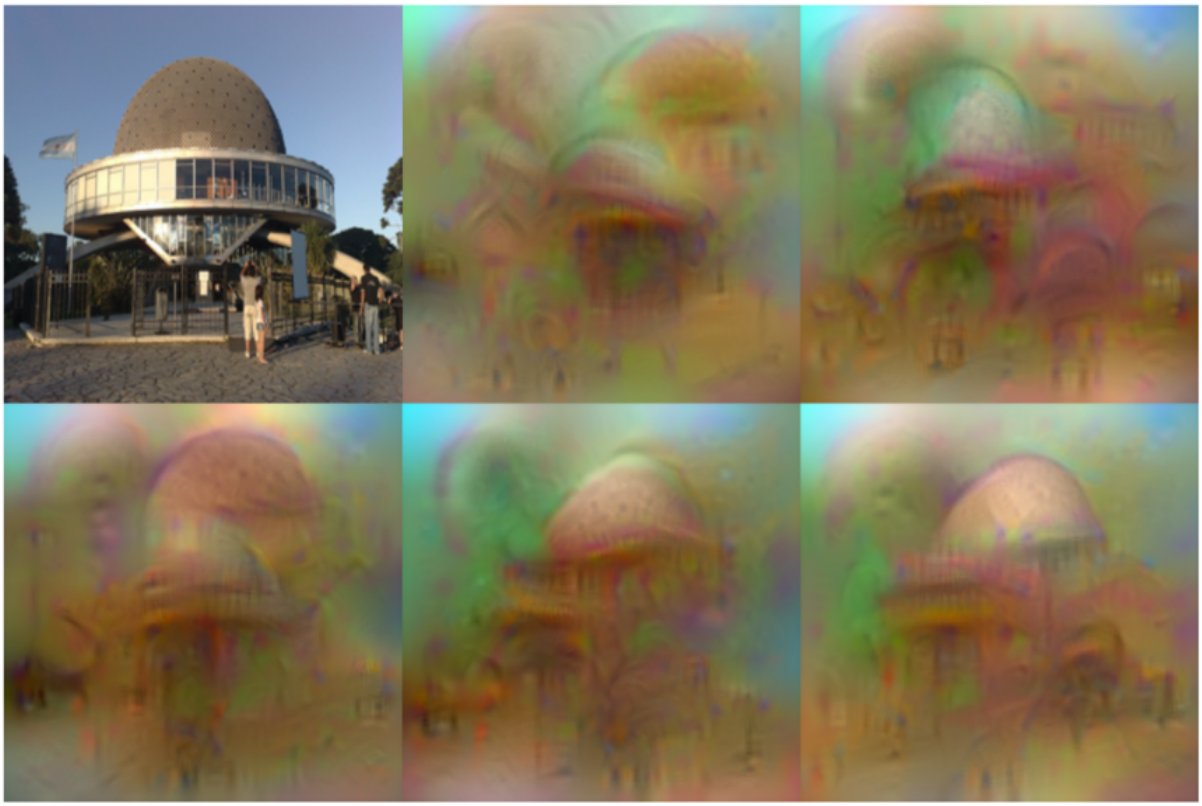


Figure 4.21: Five possible reconstructions of the reference image with code inversion method.Image courtesy of (Mahendran & Vedaldi, 2015).

train an up-convolutional neural network (Dosovitskiy & Brox, 2015). As shown in Figure 4.22, compared with Figure 4.22(b), reconstructions (Figure 4.22(a)) from FC7 and FC8 are still similar to the input image, which means this method outperforms Mahendran's method (Mahendran & Vedaldi, 2015). However, we need to train an up-convolutional neural network before using it.

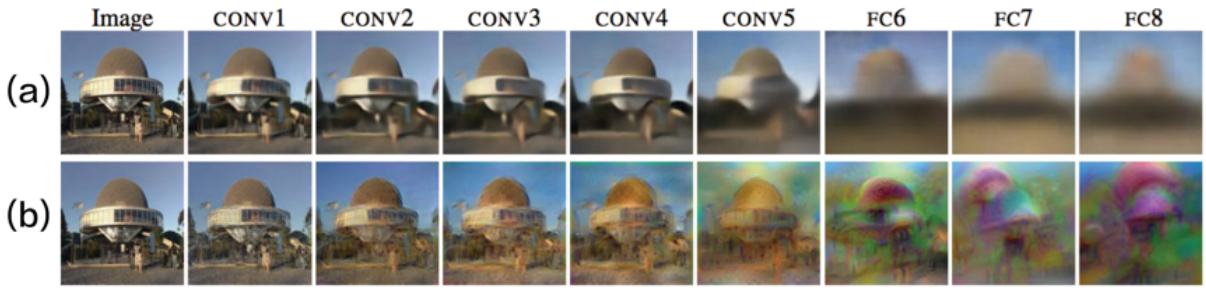


Figure 4.22: Reconstruction from layers of AlexNet. (a) an up-convolutional neural network; (b) gradient descent (Mahendran & Vedaldi, 2015). Image courtesy of (Dosovitskiy & Brox, 2015).

In summary, to reveal what neurons have learned, several simple but effective visualization methods have been proposed, mainly in the form of heatmap and bitmap, as well as image reconstruction. These methods facilitate users' understanding of what a deep neural network has captured in an intuitive manner.

CHAPTER 5

RELATIONSHIP VISUALIZATION

Apart from revealing what the neurons have learned in a deep learning model, researchers have also given continuous attention to relationships in a deep learning model, i.e., relationships between representations and relationships between neurons. In a typical deep neural network, there are usually hundreds of or thousands of neurons and also millions of connections between them, which is difficult to be visualized and analyzed. By using some techniques, such as dimensionality reduction (t-SNE, MDS, etc.), clustering and bicluster-based edge bundling, etc., different visualizations have been proposed to solve these problems.

5.1 Relationships Between Representations

In a deep learning model, learned representations refer to the representations of input data in hidden layers, which are usually high-dimensional vectors of activations. In pattern classification, although the deep learning model is considered as a black box model, it is widely believed that it has learned high-level representations of the origin observations. There are many approaches to visualize high-dimensional data, among which, t-SNE (Maaten & Hinton, 2008) has been widely and successfully applied to visualize learned representations. For example, Cho et al. used t-SNE to visualize the learned phrase representations in an RNN encoder-decoder model (Cho et al., 2014), as shown in Figure 5.1, while Karpathy used it to show the learned image representations (Karpathy, 2014b). From their work, we can see that the learned representations of similar original data tend to be projected close to each other.

Similarly, Rauber et al. have also used the t-SNE dimension reduction technique and provided more analyses about the projection (Rauber et al., 2016), which confirms the known and reveal the unknown. In their work, the relationships between learned representations are visualized by projecting high dimensional vectors to scatterplots in a 2D plane, providing insightful visual feedback about neural networks. Here we introduce Rauber et al.'s work in detail.

Figure 5.2(a) shows the t-SNE projection of the last MLP (Multilayer perceptron, a feedforward artificial neural network model) hidden layer activations on the SVHN test

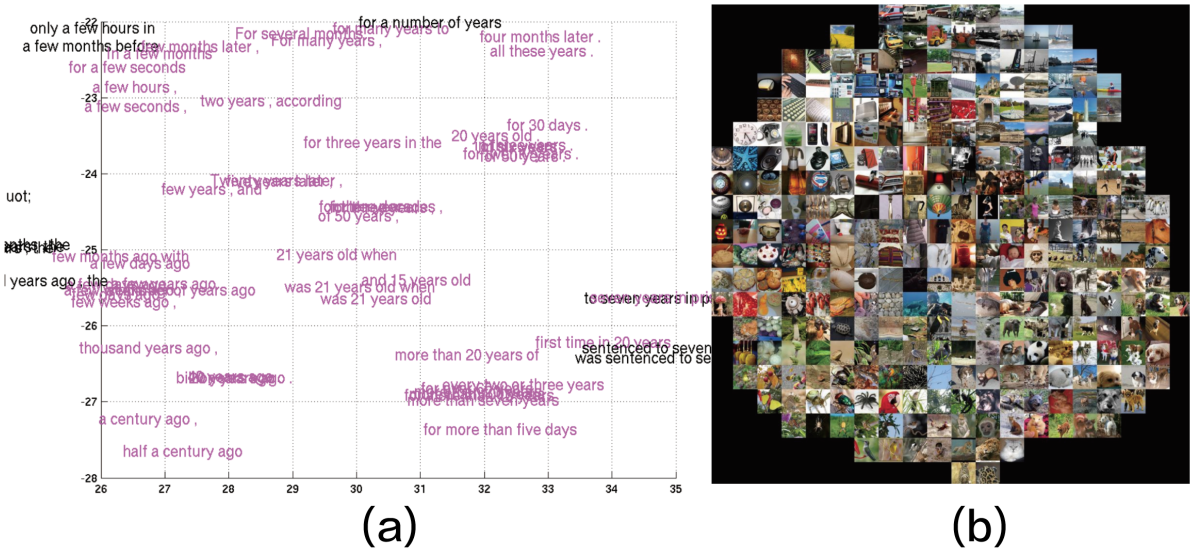


Figure 5.1: (a) t-SNE projection of the learned phrase representations in an RNN encoder-decoder model; Image courtesy of (Cho et al., 2014). (b) t-SNE projection of the learned image representations in a CNN; Image courtesy of (Karpathy, 2014b).

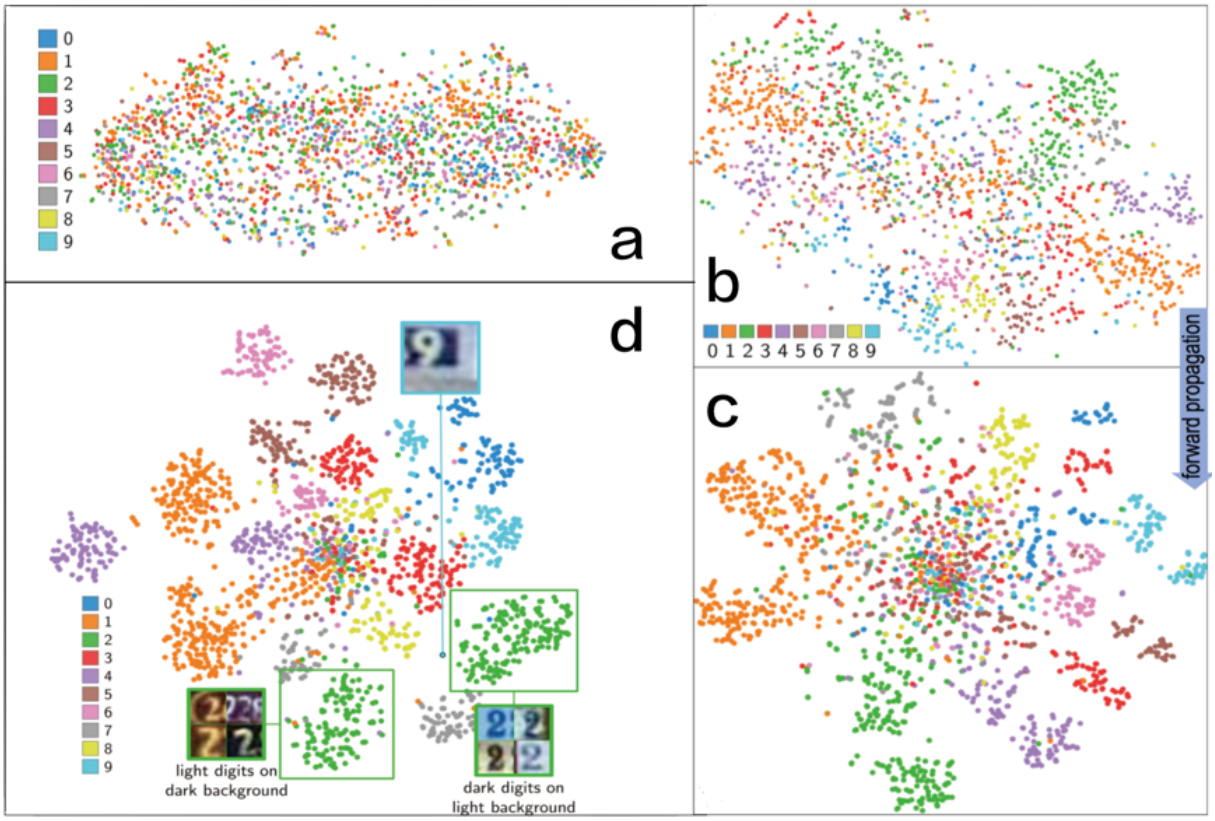


Figure 5.2: t-SNE projection of layer activations on SVHN test subset. a) last MLP (Multilayer perceptron, a feedforward artificial neural network model) hidden layer, before training; b) first MLP hidden layer, after training; c) last MLP hidden layer, after training; d) last CNN hidden layer, after training. Image courtesy of (Rauber et al., 2016).

subset before training, which shows poor class separation. Figure 5.2(b) and (c) show the projection of trained MLP hidden layer activations, i.e., the first and the last hidden layers respectively. And the projection of the last CNN hidden layer activations is shown in Figure 5.2(d). Comparing Figure 5.2(a) with (c), the visual separation between classes in Figure 5.2(c) becomes better, from which we can confirm that neural networks indeed learn to detect higher-level features, which are useful for class discrimination. While comparing Figure 5.2(b) with (c), we find the later layers in properly trained neural networks could discriminate higher-level features of the original observations. Comparing Figure 5.2(c) with (d), it is obvious that in this case CNN outperforms MLP, which indicates that we can use projection to evaluate and compare the performance of deep learning models. More interestingly, we notice that each class seems to be split into two visual clusters (Figure 5.2(d)). After exploring two green clusters corresponding to digit "2", we find that these two clusters correspond to dark digits on light backgrounds and light digits on dark backgrounds. Noting this, we can improve the classification accuracy by data preprocessing. Actually this pattern is not easy to be discovered with other conventional methods, while it is easily detected by projection. Also we can explain some misclassifications easily. For example, there is a misclassification point (digit "9") in one green cluster (digit "2"). After checking it, we find the dark border of digit "9" is similar to "2".

5.2 Relationships Between Neurons

In a deep learning model, neurons in different layers are connected by edges. To some extent, the weights of these edges can indicate different relationships between neurons across different layers. However, directly visualizing all of them easily results in severe visual clutter. To resolve this problem and to better analyze the relationships between neurons and how they relate to each other, Liu et al. proposed CNNVis, which aims for a better analysis of deep convolutional neural networks (Liu et al., 2016). In CNNCVis, they formulated a CNN as a directed acyclic graph (DAG) and proposed a hybrid visualization to reveal features learned by neurons and the interactions between them, aiming to disclose the inner working mechanism of CNNs. In order to display a large network, they first aggregated related layers, and then clustered neurons based on average activation value among a number of input images. As shown in Figure 5.3, each large rectangle represents a neuron cluster (Figure 5.3(A)), and Figure 5.3(B1) shows learned features of each neuron and Figure 5.3(B2) shows a reorder activation matrix. In the network, we can observe that the pictures tend to be higher level and more similar in the later layers, which means the

neuron clusters have learned the corresponding representations. In the reorder activation matrix, the average activation value of the i-th neuron in the j-th class is represented by the color of a cell in the i-th row and the j-th column, from which we can not only gain some insight about the activation situation of the neuron cluster, but also know the relationships between neurons and classes. After using biclustering-based edge bundling to reduce visual clutter, the connections (edges) are revealed between different layers. There is an "in-between" layer between two neuron clusters, where small green and red regions are used to indicate the proportion of the number of positive (green) edges and negative (red) edges (Figure 5.3(C)). By observing the connections between different neurons, we can get some insight into the network training situation.

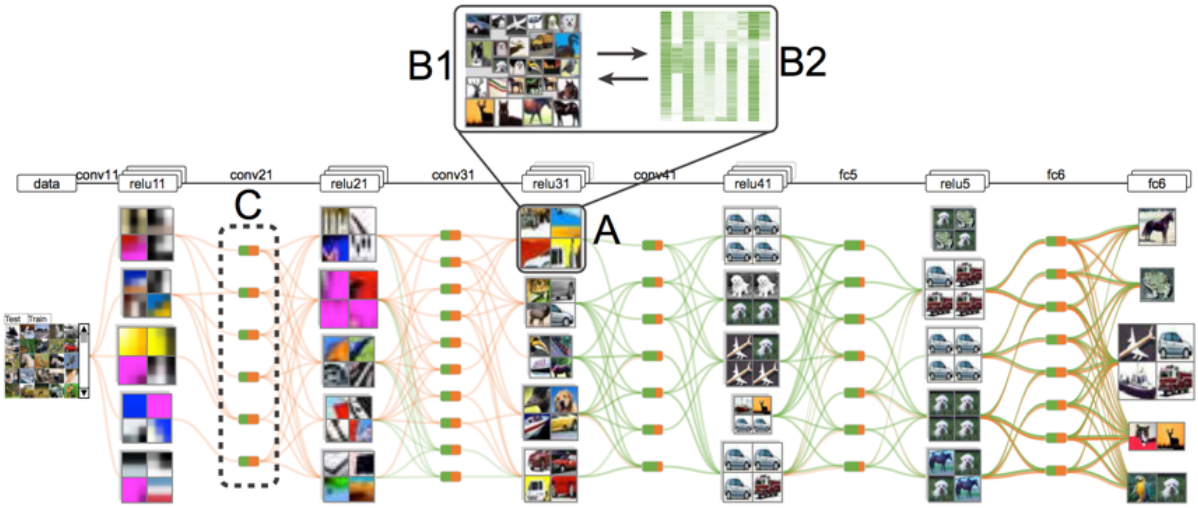


Figure 5.3: The interface of CNNVis. (A) a large rectangle for a neuron cluster; (B1) learned features for neurons; (B2) a reorder activation matrix; (C) "in-between" layer. Image courtesy of (Liu et al., 2016).

To reveal the relationships between neurons in the same layers, besides clustering neurons based on average activation value, the relationships between neurons can also be visualized by projection based on the Pearson's correlation coefficient between neurons (Rauber et al., 2016). Figure 5.4 shows the activations and neuron projections of the last CNN hidden layer before and after training on a MNIST test subset. Before training, if we brush on class "8" (yellow color), the corresponding neurons disperse in different places over the projection (Figure 5.4(a) and (b)). After training, the class clusters are separated much better, and if we brush on class "8", the corresponding neurons form a cluster in the neuron projection (Figure 5.4(c) and (d)). This shows sets of highly related neurons are created and work together in the classification task after the training process.

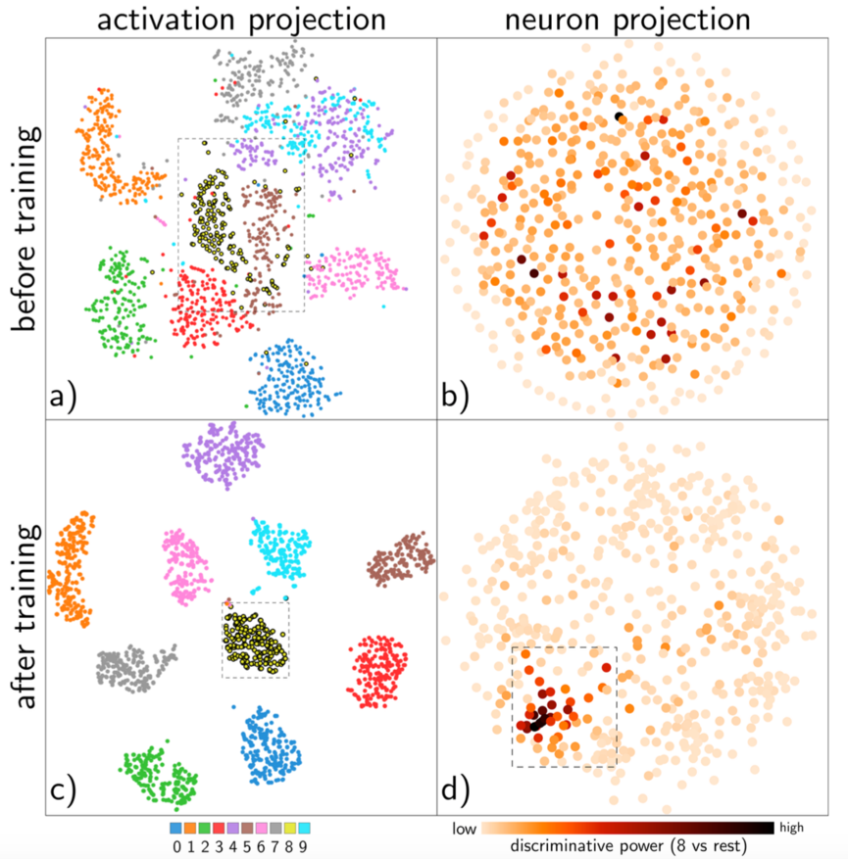


Figure 5.4: Activation projection (left) and corresponding neuron projection (right) of the last CNN hidden layer activations before training (top) and after training (bottom) performing on MNIST test subset. Image courtesy of (Rauber et al., 2016).

In this chapter, the examples have demonstrated the potential of projection techniques to explore the relationships between representations of observations learned by neural networks and the relationships between neurons. We can gain some highly valuable qualitative information from the projection. However, this visualization technique also has some drawbacks. First, visual clutter exists in the scatter-plot visualization. Second, projection (such as t-SNE) is time-consuming. Last, since the result of projection could vary every time, it is hard to trace a point of interest and do some comparisons. Compared with the scatter-plot visualization used in projection, the DAG-based visualization is more intuitive. The DAG-based visualization shows the whole network structure and learned features of each neuron. In addition, the relationships between neurons and classes are more intuitive by using the reorder activation matrix. However, since it uses directed acyclic graph, it is hard to extend to other networks with loop, such as RNNs. Also in their implementation, CNNVis currently only supports the offline training process, which focuses on analyzing snapshots of the CNN model.

CHAPTER 6

PROCESS VISUALIZATION

Some researchers are interested in the whole process of deep learning models, and have proposed interactive systems to visualize neural network structures and training information, which can facilitate people in learning and training deep learning models. In this chapter, we first describe some interactive systems to visualize neural network structures, and then introduce some visualizations on training information.

6.1 Neural Network Structure

To help design and train neural network models, many interactive systems have been proposed. Basically, we can categorize them into three classes, namely grid-based diagrams, node-link diagrams and block diagrams. While some of the previous work mainly focuses on shallow neural network, such as (Zell et al., 1994) (Streeter et al., 2001) (Tzeng & Ma, 2005), etc., in this survey, we mainly describe recent and representative systems here.

Grid-Based Diagrams Grid-based diagrams are the visualization techniques which visualize a network's layer-by-layer activation patterns or learned features. Therefore, grid-based diagrams mainly apply feature visualization techniques. Here are two examples.

One example is the ConvNetJs proposed by Karpathy (Karpathy, 2014a). Figure 4.1 shows a snippet of ConvNetJS MNIST demo, from which we can observe the activations of each layer. However, ConvNetJs does not support users to interactively create new inputs. Another example is the deep visualization toolbox proposed by Yosinski et al (Yosinski et al., 2015). In contrast, their system enable to deal with image and webcam input in real time with a trained model. Different components are shown in Figure 4.2, such as the input in the left-top corner and the whole layer of conv5 activations in the middle. For each input, we can observe the activations in each layer, and a deconv image, as well as top nine images from the training set that result in the highest activations for the selected channel.

Grid-based diagrams allow us to observe the activations of each layer easily. However, the main drawback is obvious. The activation patterns and features learned are not well

organized enough to show the architecture of a network. Therefore, it is hard for us to capture the whole picture of a neural network.

Node-Link Diagrams In contrast to grid-based diagrams, node-link diagrams visualize neurons as nodes and connections as edges, which easily give us an overview of what a network looks like. A typical node-link diagram is shown in Figure 3.1. Here we introduce some interesting examples.

Figure 6.1 shows a toy visualization system called the tensorflow playground, which is designed for non-experts to gain an understanding of deep learning (Smilkov et al., 2015). Input units are two abstract real-value features ($x_1, x_2 \in [-1, 1]$) and various mathematical combinations. Units in hidden layers are shown as small boxes, where the activation heatmaps inside indicate the situation of the neurons. The connections between units are shown as curves whose width and color indicate weight values (blue for positive, while orange for negative). The output of the network is shown as a square with a heatmap indicating the classification results. There are many parameters we can change, such as the number of layers, the number of neurons in each layers, the input features and learning rate. After setting parameters, we can press the play button to see the working flow of the whole network, which is very intuitive.

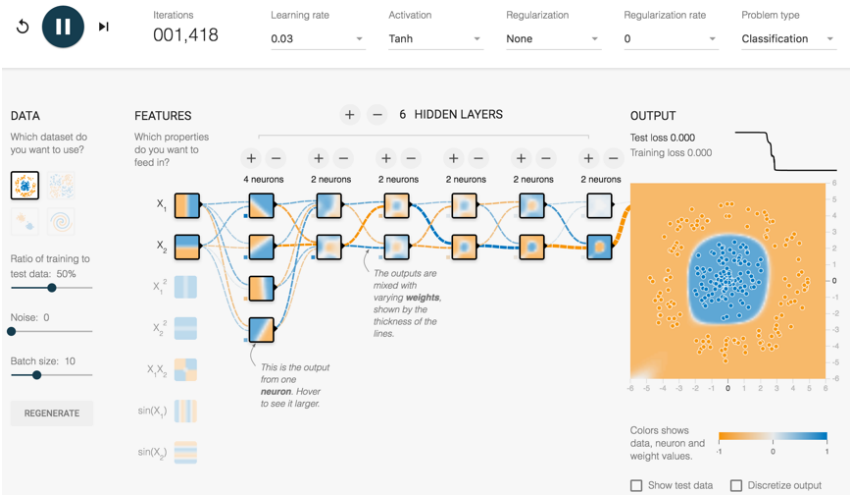


Figure 6.1: The tensorflow playground.[1]

However, the visualization that directly depicts all the neurons and edges easily leads to the visual clutter problem when applied to a large scale network. Some methods are proposed to alleviate this problem.

As shown in Figure 6.2, based on the idea of "search, show context, expand on demand" (Van Ham & Perer, 2009) and the focus-plus-context techniques (Shneiderman,

[1]http://playground.tensorflow.org/

1996), Harley hid the edges rather than depicted them directly. (Harley, 2015). When users hover on a node, the corresponding edges are shown. While when users click on a specific node, the corresponding information is shown. Through this 3D visualization of a CNN, we can easily study the architecture of the neural network and observe the input-ouput process intuitively. However, 3D visualization has the occlusion and distortion problem. A similar idea is to only show edges whose values belong to the top 50% (Figure 6.3) (Chung et al., 2016).

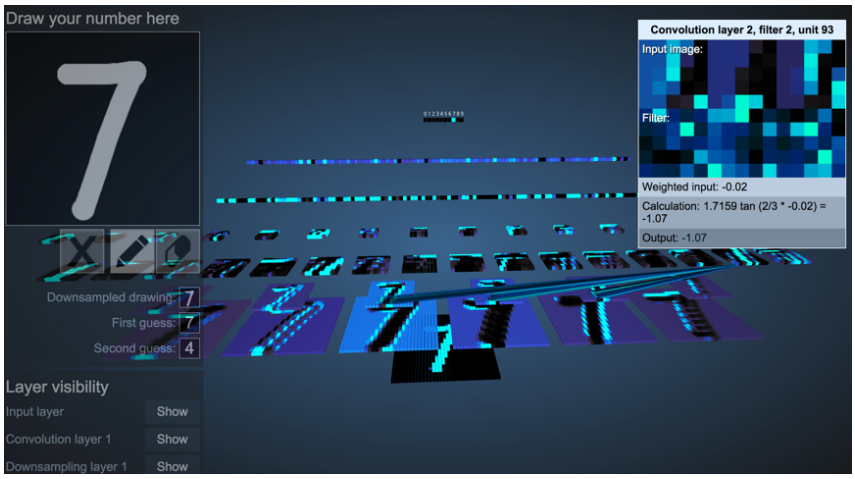


Figure 6.2: A 3D interactive node-link diagram of a CNN trained to recognize handwritten digits. Hue and brightness indicate the activation level of each node.[2]

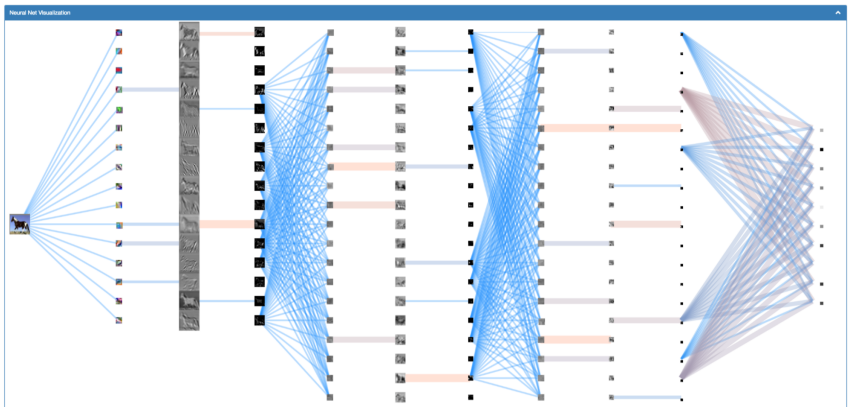


Figure 6.3: The network visualization of ReVACNN. Image courtesy of (Chung et al., 2016).

Another method to alleviate the visual clutter problem is proposed by Liu et al. (Liu et al., 2016). As shown in Figure 5.3, they first aggregated related layers, and then clustered neurons based on their average activation value among a number of images. Also they used a biclustering-based edge bundling technique to reduce visual clutter.

[2]http://scs.ryerson.ca/~aharley/vis/conv/

Node-Link Diagrams are widely used to visualize neural networks due to their simplicity and intuitiveness. However, the scalability problem is the main weakness of this visualization.

Block Diagrams Block diagrams replace each layer of nodes with a solid block and replace the connections between neurons with a single arrow line pointing from one block to another block. In this way, we can somehow solve the scalability problem. A typical block diagram is shown in Figure 6.4 (Bruckner, 2014). However, since neurons of a layer are represented as a block, it does not allow for interaction or a detailed analysis.

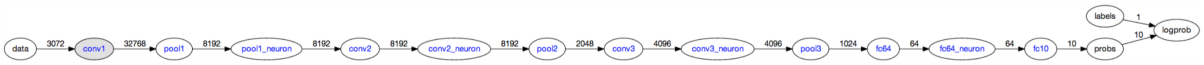


Figure 6.4: A typical block Diagram. Image courtesy of (Bruckner, 2014).

To solve this problem, a possible solution is to use a hierarchical layout. A representative example is the tensorboard (Google, 2015), which is shown in Figure 6.5. For example, when users double click a block, the block will be expanded and some detail information is shown at the same time, which allow users to do some further exploration.

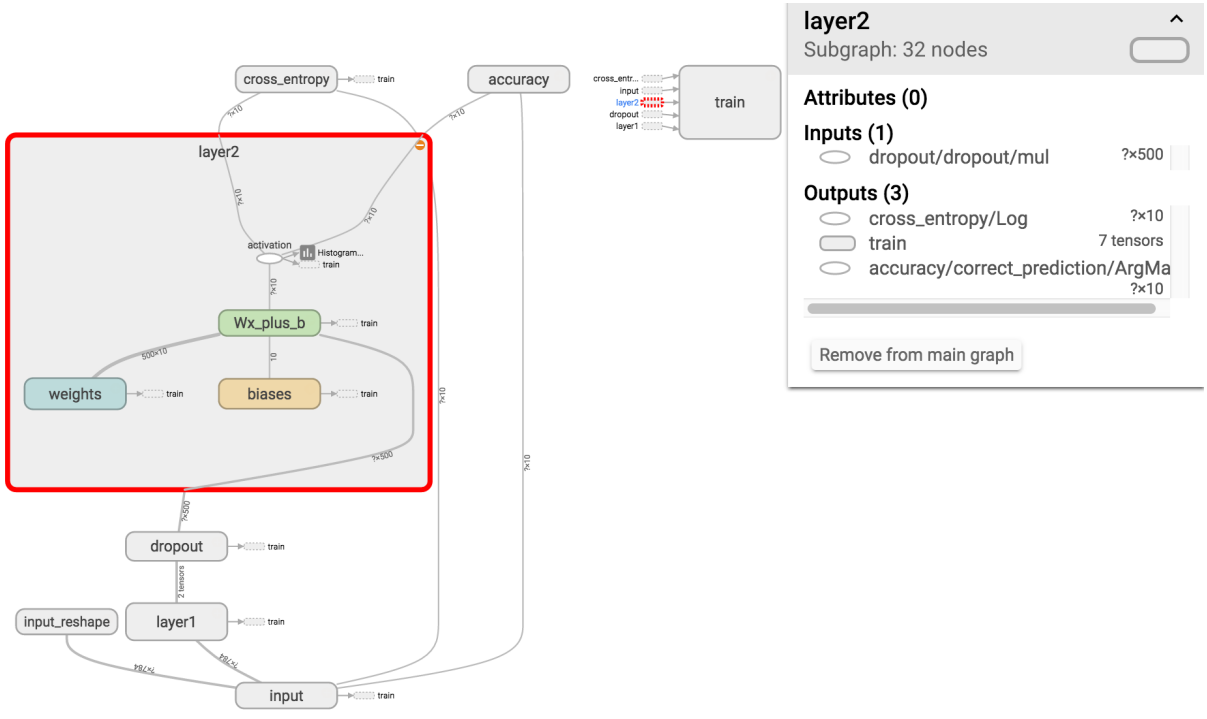


Figure 6.5: A tensorboard visualization of a simple neural network. Image courtesy of (Google, 2015).

6.2 Training Information

Meanwhile, in order to get some insight from the training process, some visualization work are done on visualizing training information, which could help to debug and design a better deep learning model. The training information contains many different types of data, such as input data (images, text, etc.), hidden layer data (activation maps, filters, hidden state, etc.), output data (loss function, classification results, etc.), some parameters (weights, biases, etc.), as well as hyper-parameters (the number of layers, the number of neurons in each layer, learning rate, batch size, etc.).

Basically, there are two ways to use these data. First, we can store the training information during training, and visualize it after training is finished. For example, the tensorBoard uses line chart and histogram to visualize the log event data , which are shown in Figure 6.6 and Figure 6.7 (Google, 2015). While Bruckner et al. used the confusion matrix to analyze the classification results (Figure 6.9) (Bruckner, 2014). The purpose is to examine the process of training and try to find some patterns. Second, we can visualize the training information during training, so that we can detect some exceptions, and then stop, refine and restart the training. An example of this is the tensorflow playground (Smilkov et al., 2015), which shows the neurons situation and loss function value during training (Figure 6.1). Another example is shown in Figure 6.8. Similarly, loss function, parameter values and gradient values are visualized by using a line chart and histogram (Skymind, 2013). Further, it is better to allow users to directly adjust the parameters and manipulate the system during the training. However, to implement a real-time manipulation is rather difficult, due to the intensive and demanding computation. Chung et al. proposed a proof-of-concept visual analytic system to steer a neural network (i.e., the capabilities of dynamic node/layer addition/removal) in real time during the training process (Chung et al., 2016). They find that the “20-20-21” model (i.e., three convolutional layers followed by ReLU and pooling layers after each convolutional layer; 20, 20 and 21 filters in each layer) could not directly trained well. On the other hand, if they train the “20-20-20” model first and add a filter after 15 epochs, a better training result on the ”20-20-21” model could be obtained, as shown in Figure 6.10.

Through visualizing the training information, we can have a better understanding and are able to train a deep learning model well. However, the existing work merely leverage relatively simple visualization techniques, such as line chart, bar chart, and histogram, etc., which are intuitive but not that informative. On the other hand, current visualization methods require many coordinated views and figures to visualize the training information, which may increase the burdens on users’ mental processing and hinder gaining insight

into the data.

In this chapter, we describe the process visualization of deep learning models, which consists of two aspects: model visualization (the structure of neural networks) and training information visualization. To better understand, design and train deep learning models, there still exists significant room for improvement regarding the process visualization.

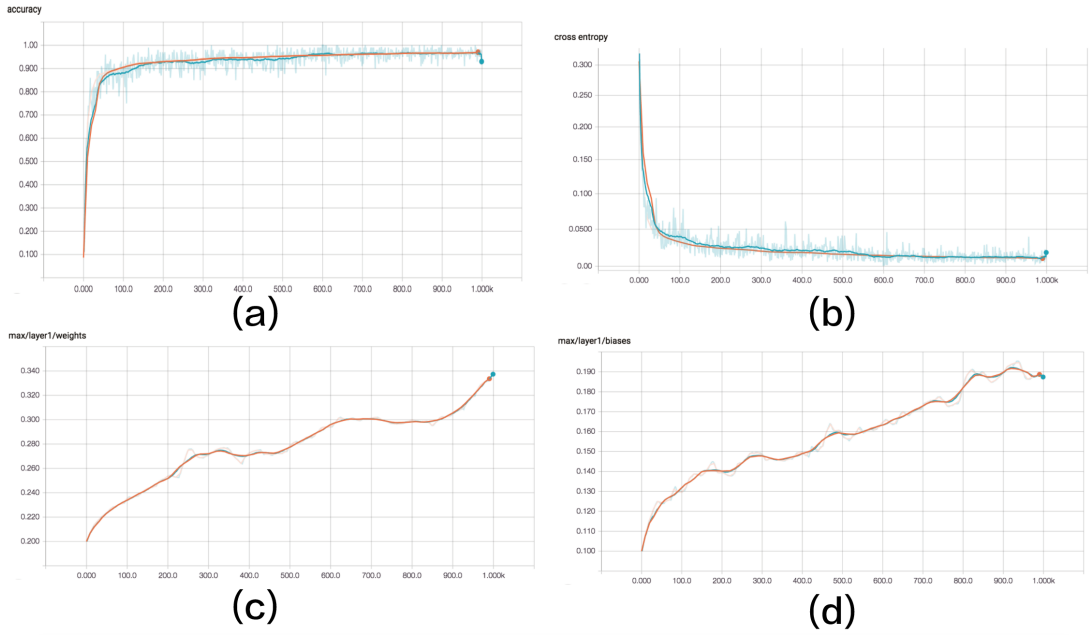


Figure 6.6: Tensorflow Events. (a) accuracy; (b) cross entropy; (c) max/layer1/weights; (d) max/layer1/biases. Image courtesy of (Google, 2015).

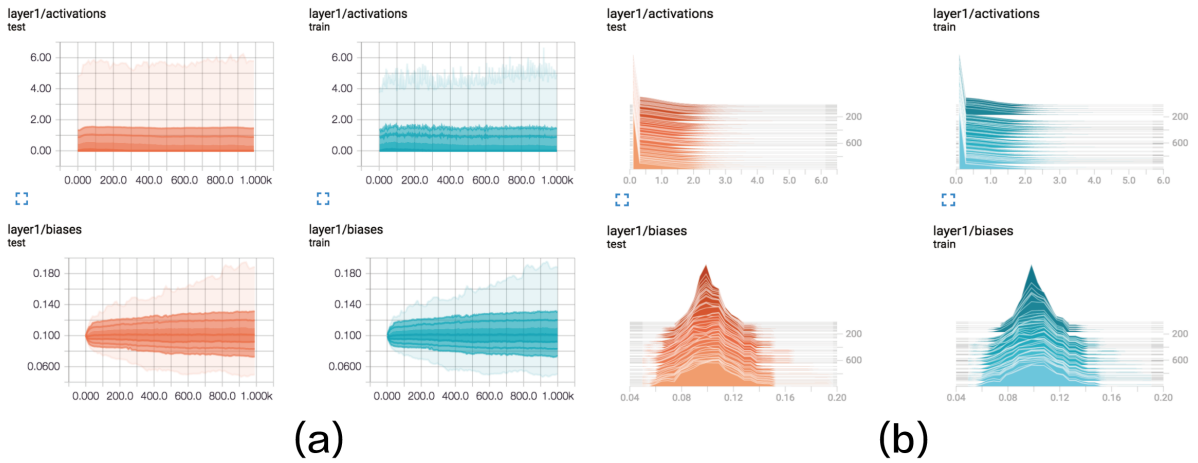


Figure 6.7: Tensorflow Distributions and Histograms. (a) Distributions, layer1, activations/biases, train/test; (b) Histograms, layer1, activations/biases, train/test. Image courtesy of (Google, 2015).

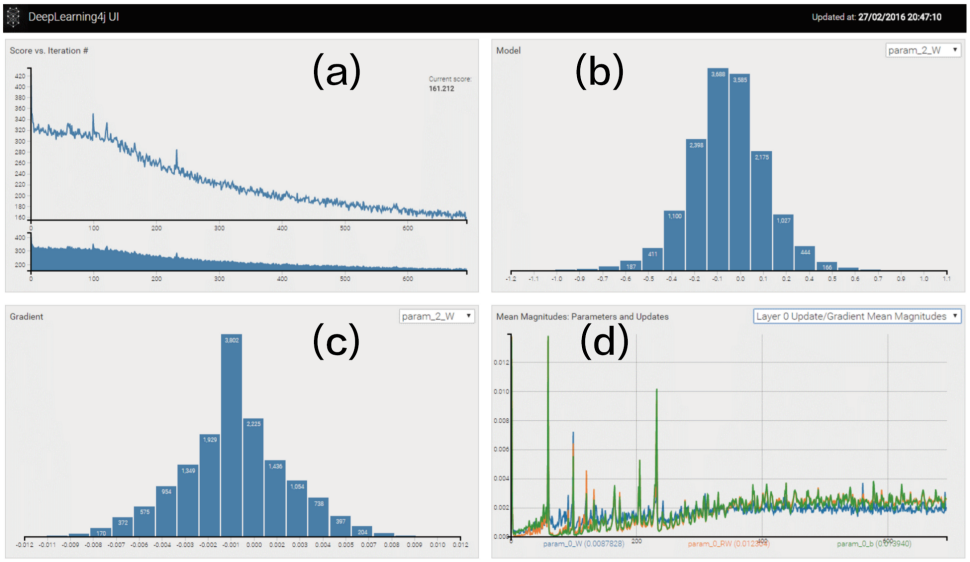
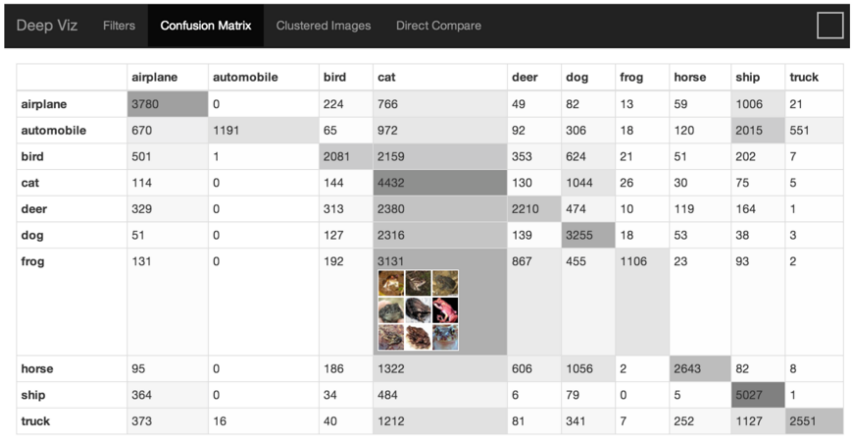


Figure 6.8: Visualization of deeplearning4j[3]. (a) loss function of the current minibatch and iteration; (b) histogram of parameter values; (c) histogram of gradient values; (d) line chart of parameters and updates. Image courtesy of (Skymind, 2013).



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	3780	0	224	766	49	82	13	59	1006	21
automobile	670	1191	65	972	92	306	18	120	2015	551
bird	501	1	2081	2159	353	624	21	51	202	7
cat	114	0	144	4432	130	1044	26	30	75	5
deer	329	0	313	2380	2210	474	10	119	164	1
dog	51	0	127	2316	139	3255	18	53	38	3
frog	131	0	192	3131	867	455	1106	23	93	2
horse	95	0	186	1322	606	1056	2	2643	82	8
ship	364	0	34	484	6	79	0	5	5027	1
truck	373	16	40	1212	81	341	7	252	1127	2551

Figure 6.9: Confusion Matrix. Rows indicate actual classes and columns indicate predicted classes. Image courtesy of (Bruckner, 2014).

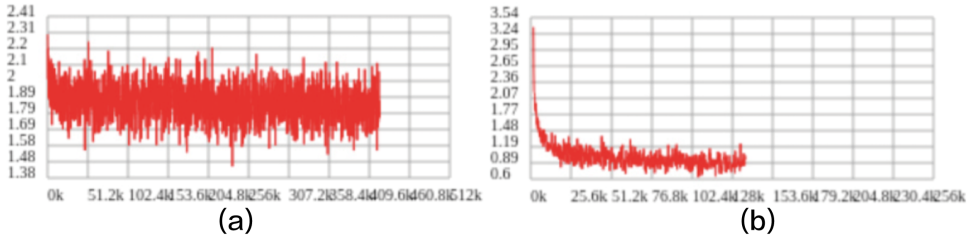


Figure 6.10: Effects of dynamic filter addition. (a) the "20-20-21" model without filter addition; (b) the "20-20-21" model with filter addition. Image courtesy of (Chung et al., 2016).

[3]http://deeplearning4j.org/visualization

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this survey, we first reviewed the basic background of deep learning. Then we introduced two common problems in the deep learning field: how deep learning models work and how to design and improve deep learning models. According to existing visualization work trying to tackle these problems, we classify them into three categories, namely feature visualization, relationship visualization and process visualization. After that we introduced several studies on these three categories. From this survey, we know several ways to integrate visualization with the deep learning models: 1) visualizing which features have been learned by deep learning models; 2) visualizing the relationships between learned representations of observations and the relationships between neurons; 3) visualizing the whole working flow of the deep learning models; and 4) visualizing training information to diagnose and refine the deep learning model interactively. There are several advantages of doing so: 1) understand what features the deep learning models learn; 2) grasp the inner working mechanism of deep learning models; 3) facilitate people to design and train better deep learning models; and 4) make the deep learning models more understandable and accessible to people.

7.2 Future Work

As previously mentioned, visualization techniques have demonstrated great advantages in understanding and improving deep learning models. However, the current applications of visualization on deep learning still have room for improvement.

Firstly, most of the existing work focuses on image data and CNNs. We can extend visualization techniques to other data (e.g., video) and models (e.g., Deep Q-networks), and may potentially gain more insights into them. In addition, more applications could be explored by combining visualization with deep learning models. For example, they could be applied to summarize and analyze massive online video data, which help obtain more thematic information and get more semantic-rich results.

Secondly, in order to better visualize model structures and the process of training data, new designs are needed, especially when encountering scalability issues. Traditional designs often result in a clutter graph, which may hinder users understanding. Researchers have already developed several approaches to alleviate visual clutter (Ellis & Dix, 2007), such as sampling, topological distortion and animation. We can further utilize them to resolve potential scalability problems. Another issue is that the current designs are not informative, involving too many coordinated different views. Although some of them are straight forward, it is difficult to reduce user's misinterpretations and the attentions required for the users, especially when there is too much information needed to be shown at a time. Therefore, there still exists room for improvements on effective designs.

Last but not least, to help users better steer and train neural network models, a visual analysis system that can give instant and iterative feedback as well as provide a friendly user interface and smooth interaction is needed. However, to design a real-time system, we may face the computation capability issues. In the future, we may overcome this bottleneck for real-time visual analysis with the aid of fast-growing computing power.

Bibliography

Alexander, M., Christopher, O., & Mike, T. (2015). Inceptionism: Going deeper into neural networks. `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`. [Online; accessed 10-October-2016].

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., & Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, *10*(7), e0130140.

Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., & MÃžller, K.-R. (2010). How to explain individual classification decisions. *Journal of Machine Learning Research*, *11*(Jun), 1803–1831.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, *2*(1), 1–127.

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, *19*, 153.

Bordes, A., Glorot, X., Weston, J., & Bengio, Y. (2012). Joint learning of words and meaning representations for open-text semantic parsing.. In *AISTATS*, Vol. 351, pp. 423–424.

Bruckner, D. (2014). Ml-o-scope: a diagnostic visualization system for deep machine learning pipelines. Tech. rep., DTIC Document.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chung, S., Suh, S., Park, C., Kang, K., Choo, J., & Kwon, B. C. (2016). Revacnn: Real-time visual analytics for convolutional neural network..

Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Vol. 22, p. 1237.

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

Deng, L. (2012). Three classes of deep learning architectures and their applications: a tutorial survey. *APSIPA transactions on signal and information processing*.

Deng, L., & Yu, D. (2014). Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, *7*(34), 197–387.

Dosovitskiy, A., & Brox, T. (2015). Inverting convolutional networks with convolutional networks. *CoRR abs/1506.02753*.

Ellis, G., & Dix, A. (2007). A taxonomy of clutter reduction for information visualisation. *IEEE transactions on visualization and computer graphics*, *13*(6), 1216–1223.

Erhan, D., Bengio, Y., Courville, A., & Vincent, P. (2009). Visualizing higher-layer features of a deep network. *University of Montreal*, *1341*.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks.. In *Aistats*, Vol. 15, p. 275.

Google (2015). Tensorboard. `https://www.tensorflow.org/tensorboard/index.html`. [Online; accessed 10-October-2016].

Goyal, Y., Mohapatra, A., Parikh, D., & Batra, D. (2016). Towards transparent ai systems: Interpreting visual question answering models. *arXiv preprint arXiv:1608.08974*.

Grün, F., Rupprecht, C., Navab, N., & Tombari, F. (2016). A taxonomy and library for visualizing learned features in convolutional neural networks. *arXiv preprint arXiv:1606.07757*.

Harley, A. W. (2015). An interactive node-link visualization of convolutional neural networks. In *International Symposium on Visual Computing*, pp. 867–877. Springer.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., & Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pp. 1693–1701.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Karpathy, A. (2014a). Convnetjs-deep learning in your browser, 2015. *URL http://cs. stanford. edu/people/karpathy/convnetjs*.

Karpathy, A. (2014b). t-sne visualization of cnn codes. `http://cs.stanford.edu/people/karpathy/cnnembed/`. [Online; accessed 10-October-2016].

Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pp. 740–755. Springer.

Liu, M., Shi, J., Li, Z., Li, C., Zhu, J., & Liu, S. (2016). Towards better analysis of deep convolutional neural networks. *arXiv preprint arXiv:1604.07043*.

Long, J. L., Zhang, N., & Darrell, T. (2014). Do convnets learn correspondence?. In *Advances in Neural Information Processing Systems*, pp. 1601–1609.

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, *9*(Nov), 2579–2605.

Mahendran, A., & Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *2015 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 5188–5196. IEEE.

Mahendran, A., & Vedaldi, A. (2016). Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 1–23.

MarcAurelio Ranzato, C. P., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In *Proceedings of NIPS*.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., & Cernockỳ, J. (2011). Empirical evaluation and combination of advanced language modeling techniques.. In *INTERSPEECH*, No. s 1, pp. 605–608.

Mohamed, A.-r., Dahl, G. E., & Hinton, G. (2012). Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(1), 14–22.

Nguyen, A., Yosinski, J., & Clune, J. (2016). Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*.

Nielsen, M. A. (2015). *Neural Networks and Deep learning*. Determination Press.

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pp. 873–880. ACM.

Rauber, P. E., Fadel, S., Falcao, A., & Telea, A. (2016). Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization & Computer Graphics*, pp. 1–1.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, *5*(3), 1.

Seide, F., Li, G., & Yu, D. (2011). Conversational speech transcription using context-dependent deep neural networks.. In *Interspeech*, pp. 437–440.

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pp. 336–343. IEEE.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Skymind (2013). Deeplearning4j: Open-source distributed deep learning for the jvm. `http://deeplearning4j.org/visualization`. [Online; accessed 10-October-2016].

Smilkov, D., Carter, S., Sculley, D., Viégas, F. B., & Wattenberg, M. (2015). Direct-manipulation visualization of deep networks..

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, Vol. 1631, p. 1642. Citeseer.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Streeter, M. J., Ward, M. O., & Alvarez, S. A. (2001). Nvis: An interactive visualization tool for neural networks. In *Photonics West 2001-Electronic Imaging*, pp. 234–241. International Society for Optics and Photonics.

Strobelt, H., Gehrmann, S., Huber, B., Pfister, H., & Rush, A. M. (2016). Visual analysis of hidden state dynamics in recurrent neural networks. *arXiv preprint arXiv:1606.07461*.

Tzeng, F.-Y., & Ma, K.-L. (2005). Opening the black box-data driven visualization of neural networks. In *VIS 05. IEEE Visualization, 2005.*, pp. 383–390. IEEE.

Van Ham, F., & Perer, A. (2009). search, show context, expand on demand: Supporting large graph exploration with degree-of-interest. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 953–960.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pp. 818–833. Springer.

Zeiler, M. D., Taylor, G. W., & Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision*, pp. 2018–2025. IEEE.

Zell, A., Mache, N., Hübner, R., Mamier, G., Vogt, M., Schmalzl, M., & Herrmann, K.-U. (1994). Snns (stuttgart neural network simulator). In *Neural Network Simulation Environments*, pp. 165–186. Springer.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2014). Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2015). Learning deep features for discriminative localization. *arXiv preprint arXiv:1512.04150*.