APPLYING DEEP LEARNING TO SCENE SKETCH RECOGNITION AND 3D SKETCH-BASED 3D MODEL RETRIEVAL

by

Yuxiang Ye

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
May 2016

Committee Members:

Yijuan Lu, Chair

Byron Gao

Xiao Chen

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

Dedicated to my parents who have always loved me unconditionally and whose good examples have taught me to work hard and brilliant for the thing that I aspire to achieve.

ACKNOWLEDGEMENTS

I am deeply grateful to my supervisor Dr. Yijuan Lu, who guides me during the past two and half years, has been a great pleasure to me, with valuable guidance, financial support, encouragement, and patience. Without her guidance and persistent help, this thesis would not have been possible.

I am also grateful to my committee members, Dr. Byron Gao and Dr. Xiao Chen, for constant support and insightful comments in this thesis.

This manuscript was submitted on March 13, 2016.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

ABSTRACT

Human's sketch understanding is an important and challenging research problem. It has many applications in human computer interaction, multimedia, and computer vision. Most previous methods focus on single-object sketch recognition. Human's scene sketch understanding has not been studied. In this work, we make the first attempt to tackle this problem. We create the first scene sketch dataset "Scene250" and explore deep learning scheme for scene sketch understanding. We propose a deep Convolutional Neural Network (CNN) model "Scene-Net" and build a novel scene sketch recognition system based on this model. Our system has been tested on the collected scene sketch dataset and compared with other state-of-the-art CNNs and sketch recognition approaches. We also extend our work to sketch-based 3D model retrieval and design a complete retrieval system (CNN-SBR) based on deep CNN. It also achieves better performance than other comparison systems. The experimental results demonstrate the effectiveness and potential of our method.

I. INTRODUCTION

Since prehistoric times, sketching has been a unique communication method to visually render human's mind. In nowadays, with the increasing popularity of devices with touch screens (e.g., touchpads and touch phones), sketching has become one of the most natural schemes in human-computer interaction. It helps people retrieve similar images by sketching a scene on the smart phone. It simplifies the task of creating a 3D cartoon model. It is also an attractive method for children, especially pre-school kids, to effectively communicate with computers. Thus, enabling computers to understand human's hand-drawn sketch is crucial and it has attracted more and more attentions recently. Wide range of its applications have been investigated, including sketch recognition [Eitz et al. (2012a); Schneider and Tuytelaars (2014)], sketch-based image retrieval [Eitz et al. (2011); Hu and Collomosse (2013)], sketch-based 3D model retrieval [Wang et al. (2015)], and forensic sketch analysis [Klare et al. (2011); Ouyang et al. (2014)].

An effective sketch understanding system enables a computer to interact with human intelligently, perform efficient sketch-based search, enhance children education with computers, and improve game design. Although decades of graphics research demonstrate that sketching is almost the only way for most people to render visual context, there has never been a formal study of how scene sketch can be recognized by computers. We examine this topic for the first time and demonstrate applications of computational scene sketch understanding.

Sketch understanding is challenging. The difficulty is mainly because a sketch only contains abstract lines but does not have color, texture, and enough visual cues. Human sketches also often depict complicated high-level semantic information. Current sketch-related methods mainly focus on recognizing single-object sketches that only contain one single object Eitz et al. (2012a); Schneider and Tuytelaars (2014) in each image. Most prior work generally follow

the traditional vision classification architecture. That is, extracting hand-crafted features, e.g., SIFT, sHOG and fisher vector, from sketch images, and usually yielding the final represented features by Bag-of-Word (BoW), then feeding them to supervised learning models, *e.g.,* Support Vector Machine (SVM). Those conventional vision classification paradigms usually require the researchers have expert knowledge on drawing or specific domains.

Very recently, Yu et al. (2015) proposed a deep CNN model for single-object sketch recognition task, which inspired our work on scene sketch recognition. However, scene sketches are very different from commonly drawn sketches, which usually depict a complete scene, a beautiful sight or even a story. More powerful deep CNN model need to be proposed to defeat those difficulties. Compared with single-object sketch recognition, scene sketch understanding is more challenging due to the complexity, variation, and uncertainty of the scene sketch:

- Complexity: scene sketches usually contain multiple objects (Fig. I.1), which introduce a lot of difficulty for computer to recognize all the objects in one scene and understand the semantic information implicated.
- Variation: sketchers from different viewing positions may generate totally different sketches of the same scene (Fig. I.2: Classroom).
- Uncertainty: one thousand people may draw the same scene in one thousand different ways. For example, different objects may appear in the same scene (Fig. I.2: Desert). The sketches of "River" can be either several strokes to depict the river shape or a portrait with fine details of plants along the river (Fig. I.2: River).

So far, there is a lack of a comprehensive study of how human's scene sketches can be well understood by computers.

In this paper, we perform an initial study of scene sketch understanding. We create the first scene sketch dataset Scene250 and explore deep learning technology for scene sketch understanding. We propose a deep CNN model and

build a novel scene sketch understanding system based on this model. The performance of our system has been tested on the collected scene sketch dataset and compared with other state-of-the-art sketch recognition approaches. Our methods give much better results.

To our best knowledge, this work is the first attempt to explore scene sketch understanding and to investigate deep learning technology on scene sketch recognition. The implication of this work could not only accelerate the research on scene sketch understanding, but also shed inspiring light on human's sketch-related research work.

We also extend our work to SHREC'16 (`http://cs.txstate.edu/~yl12/SBR2016/`), the 3D sketch-based 3D shape retrieval challenge, and achieve the best performance among five competition teams. The objective of this track is to evaluate the performance of different 3D sketch-based 3D model retrieval algorithms using a hand-drawn 3D sketch query dataset on a generic 3D model dataset. We propose a creative 3D sketch retrieval system CNN-SBR and outperform over 100% than 2nd place in the competition.

In summary, our main contributions introduced in this paper are highlighted as follows:

- The first scene sketch dataset is created and open to public. It contains 250 scene sketches across 10 categories which consists of 4 common indoor categories and 6 common outdoor ones.
- A scene sketch oriented deep CNN model, Scene-net, is proposed for the first time to target the scene sketch recognition task.
- Comprehensive experiments have been conducted to evaluate the state-of-the-art sketch recognition approaches on human's scene sketch recognition and evaluated on both indoor and outdoor scenes.
- Outstanding performance on SHREC'16 and explore applying deep CNN on human 3D sketch retrieval problem.

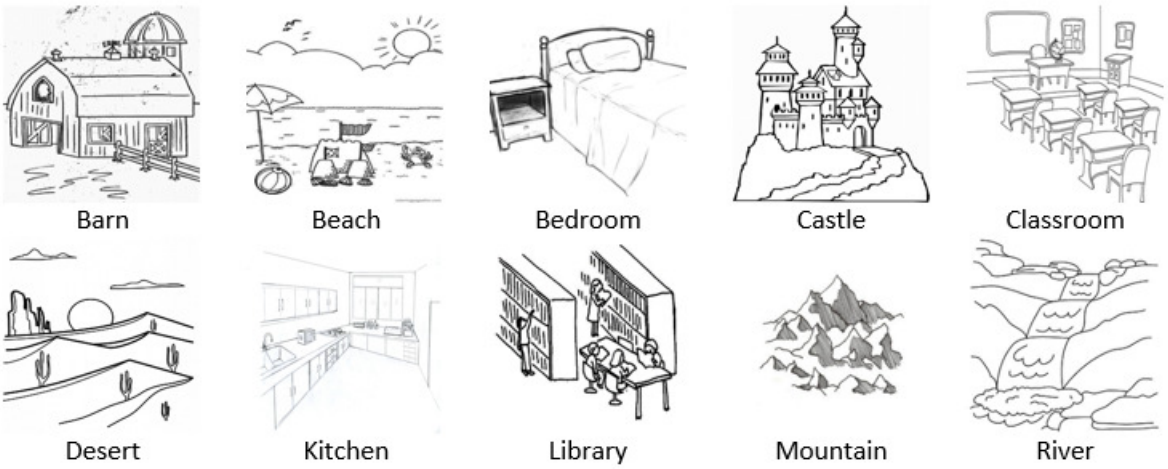


Figure I.1: Example scene sketches of our Scene250 dataset (one example per category).

- Our work will explicitly guide the research on human's sketch understanding and also provide a direction for sketch-based applications.

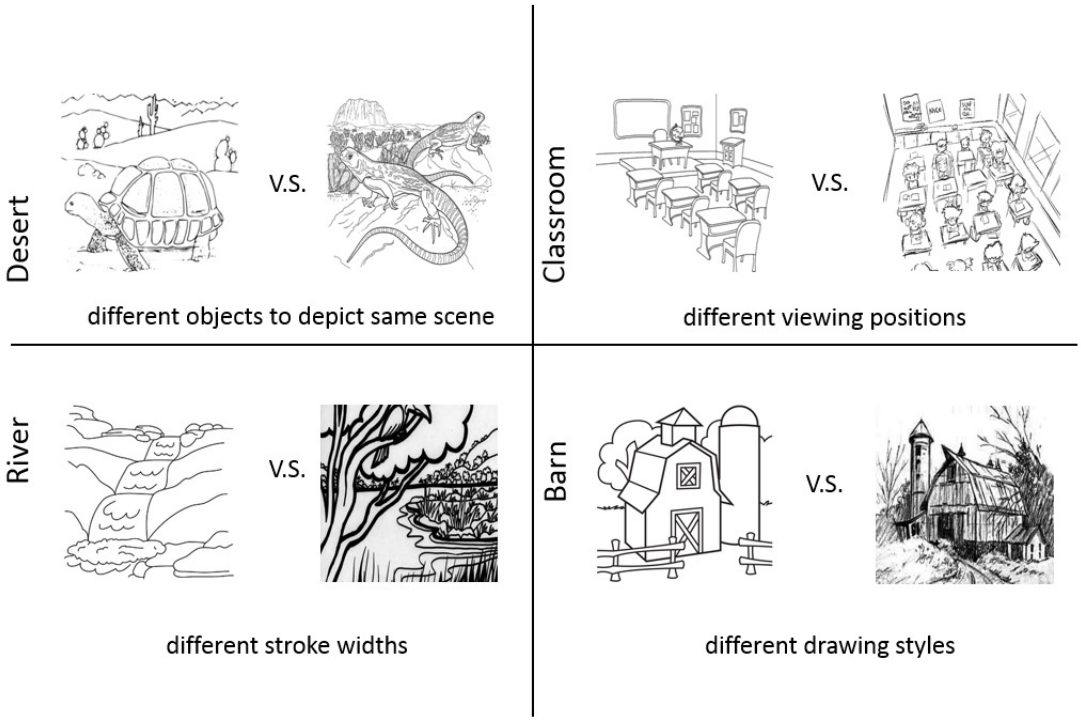


Figure I.2: Examples to demonstrate scene sketch recognition challenges.

II. BACKGROUND

In this chapter, we introduce the detailed background about the conventional sketch recognition methods and background about the deep learning algorithm, i. e. CNN, we use in our scene sketch recognition system.

Convolutional Neural Network

In this section, we review the paradigm of deep CNN design and its core four layer types, i. e. (i) convolutional, (ii) pooling, (iii) ReLU, and (iv) fully-connected [Stanford (2016)].

Like traditional Neural Network, CNN is made up of neurons which consist of learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole CNN could be expressed as a single differential score function, i. e. from the raw image pixels on one end to classification scores at the other.

CNN also employs backpropagation, which consists of forward pass and backward pass to adjust the learning weights, coupled with gradient descent optimization methods to train the learning model. At the end of the architecture, a loss function (e. g. SVM/Softmax) will be appended after the last fully-connected layer and all optimization technique developed for learning regular Neural Networks still apply.

To most common deep CNN models, including our Scene-Net, their architecture follows this pattern:

$$Input -> [[Conv -> ReLU] * N -> Pool?] * M -> [FC -> ReLU] * K -> FC$$

where $*$ denotes repetition, and $Pool?$ means pooling layer could be an optional layer. N, M, K are all integers greater than or equal to 0. All the deep CNNs mentioned in this work [Krizhevsky et al. (2012); Yu et al. (2015)], including our

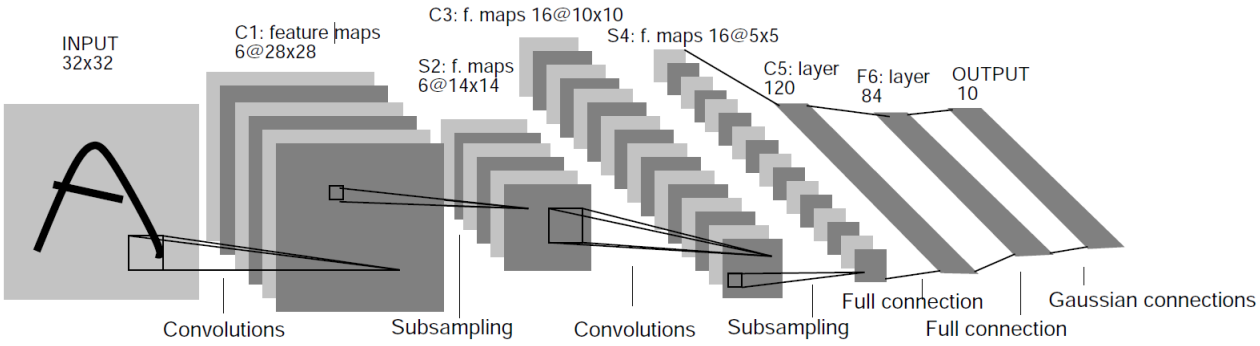


Figure II.1: Architecture of LeNet for digits recognition.

Scene-Net, accept this scheme to construct the actual deep CNN models. Figure II.1 shows the LeNet [Lecun et al. (1998)] as the example.

Input Layer

Input layer is the raw pixel data volume of the image in 3D space, i. e. $W \times H \times D$ where W, H indicates the image width and height and D indicates its depth or channel number. The depth number could be one (gray-scale image), three (RGB image), or even higher (aligned stereo model data).

Convolutional Layer

Convolutional layer could be considered as the keystone of a CNN, and compute the dot product between the layer weights and the local region they connected to in the input volume.

The convolutional layer's parameters are a set of kernel, which can be regarded as sliding matrices during computation procedure. As mentioned above, in the forward pass, we move the sliding matrix across the width and height of the input data and compute the dot product for each sliding matrix, yielding a 2D activation map of the kernel (see Figure II.3). When learning kernels see some specific type of feature at some spatial position in the input, they will construct the activation maps. Stacking these activation maps for all the kernels along the depths yields the output data volume. We now dive into the two key concepts:

(i) local connectivities and (ii) spatial arrangement.

Local connectivities. When dealing with high-dimentional inputs such as images, it is unrealistic to connect neurons to all neurons in the previous layer for the extremely high computation pressure. In practice, each neuron will only be connected with a local region of the input volume. The spatial extent of this connectivity is a hyper-parameter, which is a parameter of a prior distribution, called the receptive field of the neuron. The connectivity extent along the depth dimension is equal to the depth of the input data volume. The connections are local along width and height, while full along the entire depth dimension of the input volume. Figure II.2 explains this concept.

Spatial arrangement. We have reviewed the connectivity between neurons in the convolutional layer and the input volume. Now, we the number of neuron and the spatial arrangement in output volume. Three hyper-parameters control the size of output volume:

1. Output volume depth. It controls the number of neuron number in the convolutional layer which connect to the same region of the input layer.

2. Stride. If the stride is less than filter size, it would lead to the overlapping filter field and larger output volume. Otherwise, it lead to the non-overlapping filter and smaller output volume.

3. Zero-padding. To control the output size, it's convenient to pad zeros on the board of input volume.

The spatial size of the output volume can be computed using the input size W, the kernel size of the convolutional layer F, the stride step S, and the number of zero padding P. The formula can be expressed by $(W - F + 2P)/S + 1$.
We use an real-world example here to illustrate the formula. Our Scene-Net accepts the input image of size $[193 \times 193 \times 3]$. On the first convolutional layer, it used neurons with filter size $F = 7$, stride $S = 3$ and no zero padding $P = 0$. Since $(193 - 7)/3 + 1 = 63$, and since the convolutional layer has a depth of

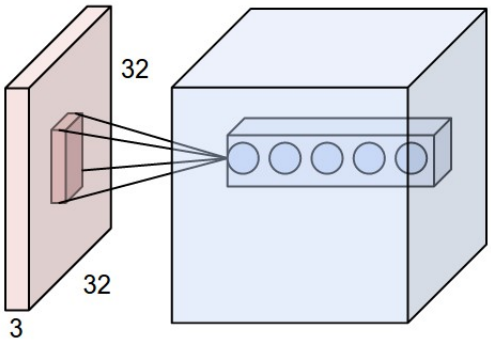


Figure II.2: Example for neuron connectivities.

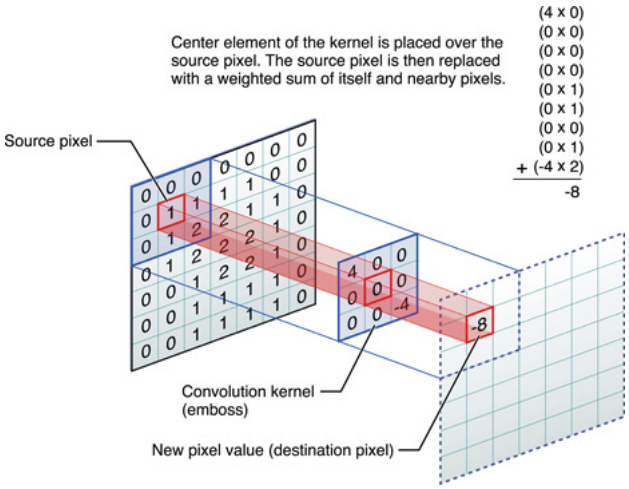


Figure II.3: Illustration of dot product in 2D convolution.

$K = 96$, the convolutional layer output volume has size of $[63 \times 63 \times 96]$. Each of the $63 \times 63 \times 96$ neurons in the volume was connected to a region of size $[7 \times 7 \times 3]$ in the input volume.

We now summarize that a convolutional layer:

- Accept the input volume of size $W_1 \times H_1 \times D_1$.
- Require four parameters: (i) kernel number K, (ii) kernel size F, (iii) stride size S, and (iv) the number of zero padding P.
- Yield the output volume of size $W_2 \times H_2 \times D_2$, where $W_2 = (W_1 - F + 2P)/S + 1$, $H_2 = (H_1 - F + 2P)/S + 1$, and $D_2 = K$.
- Applying parameter sharing introduces $F \cdot F \cdot D_1$ weights per kernel, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

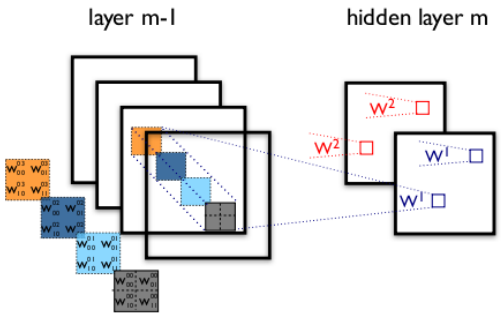


Figure II.4: Example of convolutional layer.

Pooling Layer

A pooling layer usually follows a convolutional layer in CNN scheme. The motivation for applying pooling layer is to reduce the number of neurons and the spatial size of the representation feature. The common used pooling types include max pooling (the most common one in modern CNNs), average pooling, and L2-norm pooling.

A widely used pooling form is a pooling layer with kernel of size 2×2 coupled with a stride of 2, which will downsamples every depth slice in the input by 2 and ignoring 75% of the activation neurons.

- Accept the input volume of size $W_1 \times H_1 \times D_1$
- Require two parameters: (i) kernel size F, (ii) stride size S
- Yield the output volume of size $W_2 \times H_2 \times D_2$, where
  $W_2 = (W_1 - F + 2P)/S + 1$, $H_2 = (H_1 - F + 2P)/S + 1$, and $D_2 = D_1$
- It is uncommon to use zero-padding for pooling layers

Note that there are only two types of commonly used max pooling: A overlapping pooling with $F = 3, S = 2$, and a non-overlapping pooling with $F = 2, S = 2$.

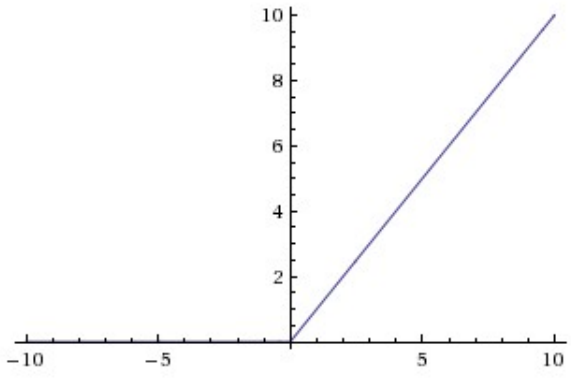


Figure II.5: ReLU activation function.

ReLU Layer

In modern deep neural network theory, the rectifier is an activation function defined as

$$f(x) = max(0, x) \tag{II.1}$$

while a neuron in deep CNNs employing the rectifier is called a ReLU. It will apply an element-wise activation function and leave the size of input volume unchanged.

Fully-connected Layer

Fully-connected layer have full connections to all activations in the previous layer. Thus, their activations can be simply computed using a matrix multiplication with a bias offset. We should note that the only difference between the fully-connected layer and the convolutional layer is convolutional layer that the activation in convolutional layer only connected to a local region in the input rather than the whole region in fully-connected layer. Thus, it is possible to convert between normal convolutional layer and fully-connected layer.

Backpropagation Algorithm in Convolutional Neural Network

Although employing 2D convolution layer, the mathematical theory of deep CNN should be exactly same to conventional neural network. In this section, we discuss the forward propagation and backward propagation, which consist of the mathematic base of deep CNN algorithm [Stanford (2015)].

Forward Propagation

Suppose there are m labeled training instances $\{x^{(1)}, y^{(1)}\}, \{x^{(2)}, y^{(2)}\}, \cdots, \{x^{(m)}, y^{(m)}\}$ and a L layers deep CNN model. For a specific training instance $\{x, y\}$, we can define the cost function as

$$J(W, b; x, y) = \frac{1}{2}\|h_{W,b}(x) - y\|^2 \tag{II.2}$$

where W denotes weights, b denotes bias term, and h denotes the heuristic function (or activation function).

Given a training set of m instances, then we can define the overall cost function as

$$\begin{aligned} J(W, b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W, b; x^{(i)}, y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{L}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\|h_{W,b}(x^{(i)}) - y^{(i)}\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{L}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2 \end{aligned} \tag{II.3}$$

In the above definition, the first term is the average sum-of-square error term and the second one is the regularization term (also called weight decay term), which used to decrease the magnitude of weights and alleviate over-fitting. The weight decay parameter λ controls the balance of the two terms. The above cost function can be applied to regression or classification problems. For regression, we re-scale the output y to $[0, 1]$ range (or if we were using *tanh* activation function, then $[-1, 1]$). For classification, we convert the output y to a 0-1 vector to represent the class labels (or if we were using *tanh* activation function, then -1

and +1 instead).

Backward Propagation

Our goal is to minimize the cost function $J(W, b)$ using neural network training. We first randomly initialize each parameter $W_{ij}^{(l)}$ and b_i^l and then apply batch gradient descent as optimization algorithm. Although $J(W, b)$ is a non-convex function and gradient descent usually reaches to local optima, gradient descent works well in practice.

The formula to update W, b in one iteration of gradient descent as follows:

$$\begin{aligned} W_{ij}^{(l)} &= W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b) \end{aligned} \tag{II.4}$$

where α denotes the learning rate. We now discuss the backpropagation algorithm which gives an efficient way to compute the partial derivatives above. In fact, we need to apply backpropagation algorithm to compute $\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y)$ and $\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y)$, the partial derivatives of the cost function $J(W, b; x, y)$ defined with respect to a single instance (x, y). Then, we can compute the derivatives of overall cost function $J(W, b)$ using:

$$\begin{aligned} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) &= \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \right] + \lambda W_{ij}^{(l)} \\ \frac{\partial}{\partial b_i^{(l)}} J(W, b) &= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \end{aligned} \tag{II.5}$$

The weight decay is applied to W but not b. Therefore, there is slightly difference between above two lines.

Now We can describe the backpropagation algorithm as follows:

1. We perform a forward propagation to compute the activations for layers $L_2, L_3, \ldots, L_n$

2. In layer n_l, for each output unit i, we set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)}) \tag{II.6}$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$ and each node i in layer l, we set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l+1} W_{ij}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{II.7}$$

4. Now we compute the partial derivatives, which are given as:

$$\begin{aligned} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) &= a_j^{(l)} \delta_i^{(l+1)}, \\ \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) &= \delta_i^{(l+1)}. \end{aligned} \tag{II.8}$$

For the convenience in real implementation, we can also re-write the above algorithm using matrix-vectorial notation. We use "$\bullet$" to denote the element-wise product operator (i. e."." in Matlab), so that if $a = b \bullet c$, then $a_i = b_i c_i$. Then, the algorithm is:

1. We perform a forward propagation to compute the activations for layers $L_2, L_3, \ldots, L_n$

2. In layer n_l, we have

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)}) \tag{II.9}$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$,

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \bullet f'(z^{(l)}) \tag{II.10}$$

4. Now we compute the partial derivatives, which are given as:

$$\begin{aligned} \nabla_{W^{(l)}} J(W, b; x, y) &= \delta^{(l+1)} (a^{(l)})^T, \\ \nabla_{b^{(l)}} J(W, b; x, y) &= \delta^{(l+1)}. \end{aligned} \tag{II.11}$$

We should note that, in this backpropagation pipeline, a max-pooling layer does not contribute to the learning process. Instead, it reduces the size of the input data volume by introducing sparseness. For example, in the 4th layer of our Scene-Net model, the max-pooling layer has filter size 3×3 with stride 2, and it reduces 3×3 blocks to a single value in forward propagation based on the pooling layer formula above. Then, in back propagation, we can get the error signal of this single value from its previous layer. This error is then forwarded to the place where it come from. Because it only come from one place in the 3×3 block, the distribution of backpropagated error from max-pooling layers are sparse.

III. RELATED WORK

Scene sketch recognition targets at recognizing scene sketch category given a hand-drawn sketch. In this chapter, we review two components related to our work: public sketch datasets and state-of-the-art sketch recognition approaches.

Object and Scene Classification

Recent advanced work in the computer vision community contribute to object and scene visual phenomena recognition using large-scale databases. These benchmarks include:

- Caltech-256 database [Griffin et al. (2007)]: object images
- SUN databse [Xiao et al. (2010)]: scene images
- LabelMe [Russell et al. (2008)] and Pascal VOC [Everingham et al. (2010)] databases: spatially labeled objects in scenes

With the building of these datasets, it allows the researchers to design algorithms to learn increasingly effective classifiers and compare the performance of recognition systems on these common benchmarks. Our recognition system follows the similar design scheme of many modern computer vision algorithms, however we are working on a challenging task and need novel methods.

Sketch Recognition

Most early sketch datasets are the small scale collections include:

- Artistic drawings [Sousa and Fonseca (2009)]
- Professional CAD figures [Mohamad et al. (2009)]
- specific domain structure sketches [Ouyang and Davis (2011); Lu et al. (2005)]

Recently, a large-scale collection of free-hand sketches (TU Berlin dataset [Eitz et al. (2012a)]) is open to public. It contains 20,000 single-object sketches in 250 daily object categories.

In earlier sketch recognition works Herot (1976); Sutherland (1964), sketching is introduced as a human-computer interaction technology in which mouse or pen is used to draw lines and curves. Recently, researchers explore more hand-crafted features, such as stroke length, stroke order and even stroke orientation to understand human sketch input at a higher level. Eitz et al. (2012a) demonstrated sHOG feature coupled with BoW methods on sketch understanding and achieved promising result of 56% accuracy in identifying unknown sketches on TU Berlin dataset.

Very recently, Schneider and Tuytelaars (2014) explore that Fisher Vectors, an image feature representation obtained by pooling local image features, can be applied to single-object sketch recognition and achieve human-like recognition accuracy (68.9% vs. 73.1% for human on the TU Berlin dataset). Despite these great efforts, hand-crafted features still require researchers to have solid knowledge on drawing or specific domains. More general sketch understanding approaches need to be explored.

Deep CNNs for Visual Recognition

Recently, deep CNNs have showed promising results on many vision recognition tasks in different domains. CNN was introduced in early 1980s, and was applied to solve simple and small vision recognition tasks like handwritten digit recognition [LeCun et al. (1990)].

In early time, the biggest bottleneck of deep CNN was the high computational cost when the classes number and input data volume are large, which significantly increase the number of neurons in CNN. However, with the proliferation of modern GPU, this bottleneck has been relieved. With the introduction of rectifier linear (ReLU) [LeCun et al. (2002)], max-pooling, local

response normalization (LRN) [Krizhevsky et al. (2012)], and dropout regularization units [Hinton et al. (2012)], CNNs become more effective, robust, and applicable. In particular, some benchmark top results of visual recognition challenge, e. g. ILSVRC [Deng et al. (2009)], have been dominated by deep CNN. Yu et al. (2015) designed a sketch-oriented deep CNN model "Sketch-a-Net" for sketch recognition task and achieved the accuracy of 74.9% on TU Berlin dataset. Nevertheless, most previous works only focus on single-object sketch recognition. However, in realistic world, people usually draw a sketch with abundant context information, i.e., a scene sketch, rather than a simple single-object sketch. Compared with single-object sketch recognition, scene sketch understanding is more challenging. Some examples are shown in Fig. I.2 to demonstrate the difficulties.

Sketch-Based 3D Model Retrieval

In computer vision community, the effort of sketch-based retrieval research work has been introduced for many years [Funkhouser et al. (2003)]. Early work on this task focus on global shape descriptors, such as distance functions [Kazhdan et al. (2002)] and shape statistics [Osada et al. (2002)]. Recently, researchers employ local features for partial matching [Funkhouser and Shilane (2006)] and use BoW [Bronstein et al. (2011)] for 3D model retrieval.

Most of the above 3D model representation methods are borrowed from traditional 2D image feature, such as BFDSIFT which is an extended SIFT feature with BoW [Furuya and Ohbuchi (2013)]. Therefore, it is important to choose a good representation of line drawing images for sketch-based 3D model retrieval. Recently, Eitz et al. proposed the GALIF and built on a collection of Gabor filters followed by a BoW method.

Instead of relying on the traditional 2D image feature, some methods also explore graph-based feature [Li et al. (2015)] and semantic labeling [Gong et al. (2013)] to facilitate the 3D model retrieval. In this work, we employ view based

methods and only use 2D sketch features in 3D model retrieval.

IV. METHODOLOGY

In this chapter, we describe the details of methodology applied on our Scene-Net model. Although our Scene-Net still follows the common used design scheme in other modern CNNs, we employ some novel features, e. g. fine-tuning, in our CNN and efficiently reduce the over-fitting and improve the performance of our recognition system.

Overview

We propose a novel scene sketch oriented deep CNN model "Scene-Net". Its architecture is illustrated in Fig. IV.1, while more detailed parameters can be found in Table IV.1.

As shown in Fig. IV.1, our Scene-Net CNN contains five convolutional layers (L1∼L5) followed by three fully-connected layers (L6∼L8). Each layer has a **ReLU!** (**ReLU!**) except for the layer L8. LRN units are appended to the layers L1 and L2 while max pooling units are appended to the layers L1, L2 and L5. We apply dropout units to the first two fully connected layers (L6 and L7). The third fully connected layer (L8), which is appended by a softmax loss, is the last layer of our CNN model. The output size of the last layer is 10 that corresponds to the 10 scene categories.

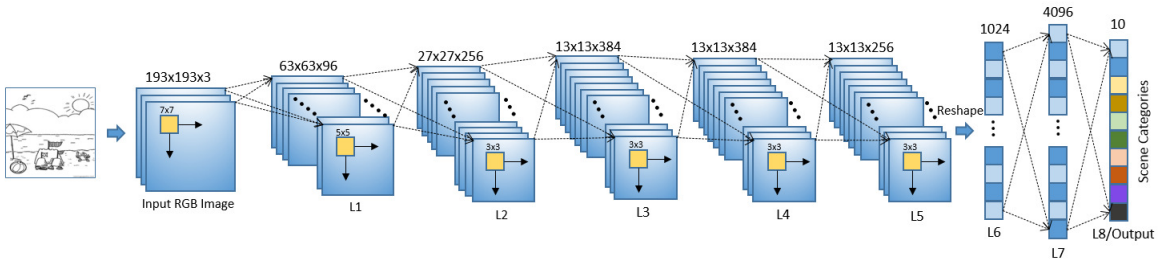


Figure IV.1: Overall architecture of Scene-Net model.

Commonalities Between Scene-Net and Other Deep CNNs

Motivated by prior advanced work on deep CNN, we follow the design scheme applied in other deep CNN models, such as AlexNet, Sketch-A-Net, GoogleNet [Szegedy et al. (2014)], etc.

Filter Number

Like most photo-oriented and sketch-oriented deep CNNs [Krizhevsky et al. (2012); Simonyan and Zisserman (2014); Yu et al. (2015)], the filter number increases with the model depth in our Scene-Net. We set the filter number of the first layer to 96 and increase it after each max pooling layer until 4096.

Padding

We apply zero-padding to the layers L3∼L5, which is used to ensure that size of input and output data volume are consistent in these layers.

Stride

We set the stride of the first convolutionaly layer to 3 while set it after the first layer to 1. Employing this configuration could keep as much data as possible after each convolution operation.

Local Response Normalization

LRN! (**LRN!**) is inspired by real neuroscience study. It performs a kind of "lateral inhibition" by normalizing over local input regions and create competition with different kernels for big activities among neuron outputs. It is also widely used in modern CNN recognition architecture.
In this paper, we denote the activity of a neuron computed by applying kernel i at position (x, y) as $a^i_{x,y}$ with ReLU non-linearity, and denote the

response-normalized activity as $b^i_{x,y}$, and give the expression of

$$b^i_{x,y} = a^i_{x,y}/(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)})^\beta \quad \text{(IV.1)}$$

where n denotes the local region size, N denotes the total number of kernels in the layer, α denotes the scaling parameter, and β denotes the exponent.
The constants k, n, α, β are hyper-parameters whose values are based on experimental results. We use $k = 2, n = 5, \alpha = 10^{-4}, and \beta = 0.75$ in this work and applied this normalization unit after first two **ReLU!** units (in the layers L1 and L2).

Dropout

Prior deep CNNs successfully deal with the over-fitting issue by combining the predictions of many different models. However, it is too expensive for modern deep CNN models, because of the longer training time for each model. Recent work [Hinton et al. (2012)] demonstrate that employing dropout regularization, which can reduce complex co-adaptations of neurons by randomly setting unit activation to zero. In this way, the "dropped out" neurons do not contribute to the forward pass and do not participate in back-propagation.
In our Scene-Net, we use dropout in the first two fully-connected layers and set the dropout rate as 0.5, which is a reasonable approximation and append it to the first two fully connected layers.

Novel Features in Our Scene-Net Architecture

In this section, we explain the novel features applied in our Scene-Net. Although few prior CNN research work discuss the detail why to set a parameter using a specific value, and it's also impossible to exhaustively verify the effectiveness of each hyper-parameter, we still propose some points which are different to conventional deep CNN models' setting and explain the reason. At the same

time, we also introduce fine-tuning, a very important technique to alleviate over-fitting problem in our small-scale benchmark Scene250.

Smaller Input Size

Input size in deep CNN is one of the most critical parameter, as all the following layers depend on the input layer. Many modern deep CNNs , e. g. AlexNet[Krizhevsky et al. (2012)] and Sketch-A-Net Yu et al. (2015), use large input size around 225×225. However, we find that smaller input size is more appropriate for scene sketch recognition. The major reason is that scene sketches usually contain multiple objects within one image. Smaller input size combined with random data augmentation can capture more sophisticated features. To this end, input size of 193×193 is used in our Scene-Net.

Overlapped Max Pooling

Some recent deep CNNs employ 2×2 max pooling with stride 2 [Simonyan and Zisserman (2014)]. This sort of non-overlapped max pooling can bring some spatial invariance and reduce the layer size by 75%. However, we find that the 3×3 pooling size with stride 2, i. e.overlapped max pooling, applied on our Scene-Net model could bring the improvement for our recognition system while reduce the layer size by 50%.

Fine-tuning

Fine-tuning is a commonly used method in deep CNN study to prevent over-fitting on small-scale dataset. It pre-trains a CNN model on a large ancillary dataset and learns the initial weights for the model. Then, it resumes the training process of the same CNN model on the target dataset (usually a small one) with the learned initial weights.
More specifically, in our work, we first pre-trained Scene-Net model on the TU Berlin dataset and obtained the initial learning weights for the Scene-Net model.

Table IV.1: Detailed architecture of Scene-Net

Index	Layer	Type	Filter Size	Filter Num	Stride	Pad	Output Size
0		Input	-	-	-	-	193×193
1	L1	Conv	7×7	96	3	0	63×63
2		ReLU	-	-	-	-	63×63
3		LRN($n = 5, \alpha = 10^{-4}, \beta = 0.75$)	-	-	-	-	63×63
4		Maxpool	3×3	-	2	0	31×31
6	L2	Conv	5×5	256	1	0	27×27
7		ReLU	-	-	-	-	27×27
8		LRN($n = 5, \alpha = 10^{-4}, \beta = 0.75$)	-	-	-	-	27×27
9		Maxpool	3×3	-	2	0	13×13
10	L3	Conv	3×3	384	1	1	13×13
11		ReLU	-	-	-	-	13×13
12	L4	Conv	3×3	384	1	1	13×13
13		ReLU	-	-	-	-	13×13
12	L5	Conv	3×3	256	1	1	13×13
13		ReLU	-	-	-	-	13×13
14		Maxpool	3×3	-	2	0	6×6
15	L6	Fully-connected	6×6	1024	1	0	1×1
16		ReLU	-	-	-	-	1×1
17		Dropout(0.5)	-	-	1	0	1×1
18	L7	Fully-connected	1×1	4096	1	0	1×1
19		ReLU	-	-	-	-	1×1
20		Dropout(0.5)	-	-	1	0	1×1
18	L8	Fully-connected	1×1	10	1	0	1×1
19		Softmax loss	-	-	-	-	1×1

Then, we resumed the training process on the Scene250 dataset to fine tune Scene-Net model for scene sketch recognition task. Because we want to predict 10 scene categories in fine-tuning step instead of 250 object categories in pre-training, we change the output number of last layer from 250 to 10.

We also decrease the learning rate to slow down the fine-tuned training. The motivation is, after pre-training, we have gotten the stable classification model and just want to apply this model to a new recognition task. Thus, the learning rate should be decreased to avoid change its stability.

In our experiments, we find applying fine-tuning would significantly increase the recognition accuracy and reduce the over-fitting problem.

V. EXPERIMENTS

Overview

In this chapter, we discuss details of experiment dataset collection and the compare our proposed deep CNN model with other state-of-the-art CNNs which are designed for similar tasks. We also demonstrate that our Scene-Net model get better performance than other comparison models.

Scene Sketch Recognition

Scene250 Dataset Collection

We create the first scene sketch dataset "Scene250" for scene sketch recognition task. Scene250 (Fig. I.1) contains 250 scene sketches collected from Internet. These sketches cover 10 different categories and each category has 25 sketches. More specifically, we searched on Google using a specific category name. Then we downloaded more than 100 sketches for each category and selected 25 representative ones. The following criteria are used for selecting 10 categories and 25 scene sketches in each category.

- Completeness. Although the categories of Scene250 cannot exhaustively cover all the scene categories, the selected 10 categories are the most common scenes in our daily life and it contains both indoor and outdoor scenes.
- Unambiguous. Each selected sketch contains no more than one scene content. For example, we don't select the sketches that contain both "desert" and "river".
- Recognizable. That is human can easily recognize these selected sketches with their shape alone and do not need additional context information.

- Diversiform. Sketches in the same scene category should be diversified. Thus, we don't select the sketches that are visually similar in order to guarantee the diversity.

We inspected the whole dataset by displaying sketches on a screen. All the scene sketches in the same category were displayed together, which allowed us to identify improper ones easily. We removed those sketches that are in the wrong category (e.g., a mountain scene in the bedroom category) or not qualified with our selection criteria. We also rescaled the dataset to contain exactly 25 scene sketches per category yielding final 250 scene sketches in the Scene250 dataset. Benefited from consistent size of each category, there is no need to correct for bias toward the categories with different size when performing training or testing.

Image Format

We rescaled all the images to 256×256 pixels and read the images as single channel or multiple channels, which depends on the architecture of CNNs. In our experiments, we find employing multiple channels, i. e. RGB channels, has a promising result on our Scene-Net model.

Data Augmentation

We performed data augmentation, which is a commonly used technique in machine learning study to prevent overfitting. In our experiments, we replicated our scene sketches by 500 times using random rotation, shift and flip. More specifically, we describe our algorithm as follow:
Based on the above algorithm, an image duplicate would be generated with one or more image transformations of shift, rotation and flip. We introduce this random data augmentation algorithm to increase the variety of the experiment dataset, which significantly reduces the sketching noise from different user hand-drawing.

Input : Original sketch dataset S
Output : Enlarged sketch dataset T with random shifts, rotations, and flips
initialization;
$w = width_{original} - width_{target}$;
foreach $I \in S$ **do**
  for $i \leftarrow 1$ **to** 500 **do**
    $C \leftarrow copy(I)$;
    $x_{shift} \leftarrow random(0, w)$;
    $y_{shift} \leftarrow random(0, w)$;
    $C \leftarrow shift(C, x_{shift}, y_{shift})$;
    $roll \leftarrow random(0, 1)$;
    if $roll < 0.5$ **then**
      $rd \leftarrow random(-5, 5)$;
      $C \leftarrow rotate(C, rd)$;
    end
    $roll \leftarrow random(0, 1)$;
    if $roll < 0.5$ **then**
      $C \leftarrow flip(C)$
    end
    $append(T, C)$;
  end
end

Algorithmus 1 : Data augmentation algorithms

Comparison

We compared our Scene-Net model with other two deep CNN models: AlexNet [Krizhevsky et al. (2012)] and Sketch-a-Net. Although AlexNet is a photo-oriented deep CNN model designed for ImageNet, it has shown great results in many applications (e.g., video, robotics, and bioinformatics). Sketch-a-Net is a sketch-oriented deep CNN model specifically designed for sketch classification. It even beats human recognition accuracy by 1.8% on TU Berlin dataset. Both AlexNet and Sketch-a-Net have five convolutional layers followed by three fully-connected layers (last fully-connected layer is the output layer). Sketch-a-Net also employs larger first layer filters (11×11), higher dropout rate (0.55), and overlapped pooling to achieve promising result on sketch-based object recognition. More detailed architecture of AlexNet and and Sketch-A-Net can be learned in V.1 and V.2.

We tested our Scene-Net on Scene250 dataset for scene sketch recognition and

Table V.1: Detailed architecture of AlexNet

Index	Layer	Type	Filter Size	Filter Num	Stride	Pad	Output Size
0		Input	-	-	-	-	224 × 224
1	L1	Conv	11 × 11	48	4	0	55 × 55
2		ReLU	-	-	-	-	55 × 55
3		Maxpool	3 × 3	-	2	0	27 × 27
4	L2	Conv	3 × 3	128	2	0	13 × 13
5		ReLU	-	-	-	-	13 × 13
6		Maxpool	3 × 3	-	1	1	13 × 13
7	L3	Conv	3 × 3	192	1	1	13 × 13
8		ReLU	-	-	-	-	13 × 13
9	L4	Conv	3 × 3	192	1	1	13 × 13
10		ReLU	-	-	-	-	13 × 13
11	L5	Conv	3 × 3	256	1	1	13 × 13
12		ReLU	-	-	-	-	13 × 13
13		Maxpool	3 × 3	-	2	0	7 × 7
14	L6	Fully-connected	7 × 7	2048	1	0	1 × 1
15		ReLU	-	-	-	-	1 × 1
16		Dropout(0.55)	-	-	1	0	1 × 1
17	L7	Fully-connected	1 × 1	2048	1	0	1 × 1
18		ReLU	-	-	-	-	1 × 1
19		Dropout(0.55)	-	-	1	0	1 × 1
20	L8	Fully-connected	1 × 1	10	1	0	1 × 1
21		Softmax loss	-	-	-	-	1 × 1

compared it with the above state-of-the-art CNN models. Following previous work, we used 2/3 sketches per category for training and 1/3 for testing. In addition, since there are two types of scene sketches in Scene250: indoor scene (i.e., bedroom, classroom, kitchen, and library) and out-door scene (i.e., barn, beach, castle, desert, mountain, and river), we also tested these CNN models on in-door scene and out-door scene recognition individually. The comparison results are listed in Table V.3 and V.4. From the results, we find that our Scene-Net model not only beats AlexNet and Sketch-a-Net on overall recognition performance but also beats them on both in-door and out-door scene recognition task.

Table V.2: Detailed architecture of Sketch-A-Net

Index	Layer	Type	Filter Size	Filter Num	Stride	Pad	Output Size
0		Input	-	-	-	-	225 × 225
1	L1	Conv	15 × 15	64	3	0	71 × 71
2		ReLU	-	-	-	-	71 × 71
3		Maxpool	3 × 3	-	2	0	35 × 35
4	L2	Conv	5 × 5	128	1	0	31 × 31
5		ReLU	-	-	-	-	31 × 31
6		Maxpool	3 × 3	-	2	0	15 × 15
7	L3	Conv	3 × 3	256	1	1	15 × 15
8		ReLU	-	-	-	-	15 × 15
9	L4	Conv	3 × 3	256	1	1	15 × 15
10		ReLU	-	-	-	-	15 × 15
11	L5	Conv	3 × 3	256	1	1	15 × 15
12		ReLU	-	-	-	-	15 × 15
13		Maxpool	3 × 3	-	2	0	7 × 7
14	L6	Fully-connected	7 × 7	512	1	0	1 × 1
15		ReLU	-	-	-	-	1 × 1
16		Dropout(0.55)	-	-	1	0	1 × 1
17	L7	Fully-connected	1 × 1	512	1	0	1 × 1
18		ReLU	-	-	-	-	1 × 1
19		Dropout(0.55)	-	-	1	0	1 × 1
20	L8	Fully-connected	1 × 1	10	1	0	1 × 1
21		Softmax loss	-	-	-	-	1 × 1

Table V.3: Scene sketch recognition performance comparison

Scene-Net	AlexNet	Sketch-a-Net
60.00%	52.50%	38.75%

Table V.4: In-door/Out-door scene sketch recognition comparison

	Scene-Net	AlexNet	Sketch-a-Net
In-door	50.00%	46.88%	25.00%
Out-door	66.67%	56.25%	47.92%

Running Cost

We implemented Scene-Net model using Matlab and the MatConvNet toolbox. All the experiments were executed on a server with an 8-core 3.50GHz CPU and a GeForce GTX Titan X GPU. The pre-training time for our Scene-Net model on the TU Berlin dataset is approximately 8 hours on GPU, while fine-tuning with Scene250 dataset is about 3 hours on GPU.

Sketch-Based 3D Model Retrieval

Introduction of SHREC'16

The objective of SHREC'16 is to evaluate the performance of different 3D sketch-based 3D model retrieval methods using a hand-drawn 3D sketch query dataset on a 3D model benchmark dataset (SHREC13STB [Li et al. (2013)]). 3D sketch-based 3D model retrieval is to retrieve relevant 3D models using corresponding 3D sketch as input. Prior sketch-based 3D model retrieval systems

Table V.5: Recognition performance for each category

	Barn	Beach	Bedroom	Castle	Classroom
Accuracy	62.50%	62.50%	25.0%	62.50%	75.00%
	Desert	Kitchen	Library	Mountain	River
Accuracy	62.50%	37.50%	62.50%	87.50%	62.50%

are built on 2D sketching technology, which require users to draw sketches on the 2D plane. However, 2D sketching lose the 3D or space information, which creating a huge gap between a 2D sketch and the accurate space motion representation of a 3D model.

Inspired by the above gap, 3D sketching technology is introduced to this task. 3D sketching offers more accurate space representation of a 3D model than conventional 2D sketching. In realist, a Kinect could be used to track the human hand's motion to construct a 3D sketch. Considering Kinect is a cheap and popular motion sensing input device developed by Microsoft and support RGB camera, depth sensor and multi-array microhone, we can easily obtain a large number of human 3D sketches.

In this challenge, we have two benchmark datasets: (i) 3D sketch dataset. The 3D sketch query set consists of 300 3D sketches (30 categories, 10 sketches per category, also called Kinect300), while 21 categories have the corresponding models in the target SHREC13STB dataset. The dataset is collected from 17 volunteers (13 males and 4 females) where each volunteer drew several sketch categories.. Figure V.1 shows some example 3D sketches. (ii) 3D model dataset. The 3D benchmark dataset is built on the SHREC13STB, which consists of 1258 models unevenly distributed in 90 categories. Figure V.2 shows some example 3D models.

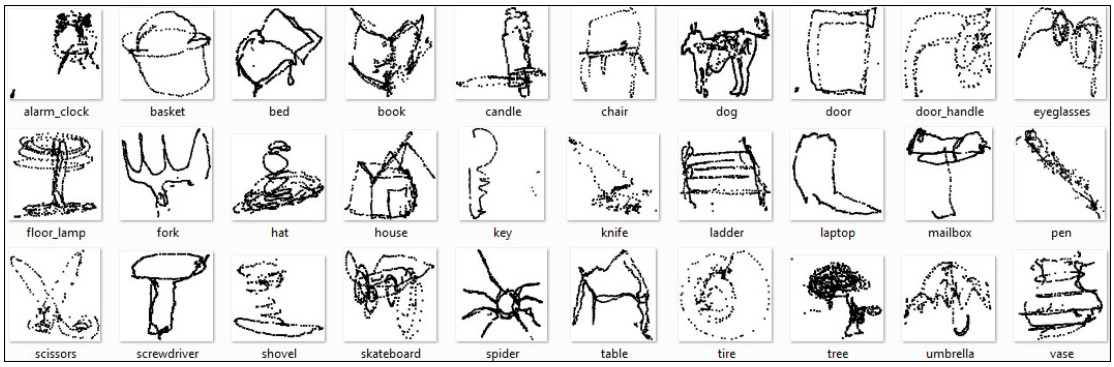


Figure V.1: Example 3D sketches of Kinect300 dataset

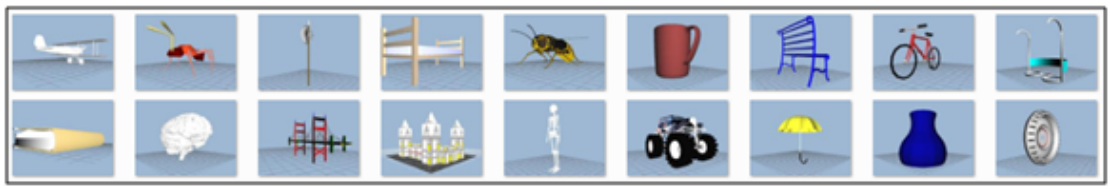


Figure V.2: Example 3D models of the SHREC13STB benchmark

Retrieval Architecture Design

Our 3D sketch-based shape retrieval architecture (CNN-SBR) is inspired by early sketch-based image retrieval work. We employ state-of-the-art deep CNN in sketch object recognition and combine multiple 3D model processing techniques in this work. We first pre-train our deep CNN model on TU Berlin dataset and obtain well learned weights for our CNN model. Then, we convert all the 3D sketches to multiple 2D sketch views for both training and query dataset, and perform data augmentation for these 2D sketch views, then fine-tune the CNN model using previous well learned weights. After that, we have the classification result for each query 3D sketch based on its 2D sketch views and fine-tuned CNN model. Finally, we use majority vote and simple label matching to generate the output result. Our CNN-SBR architecture is listed in Figure V.3.

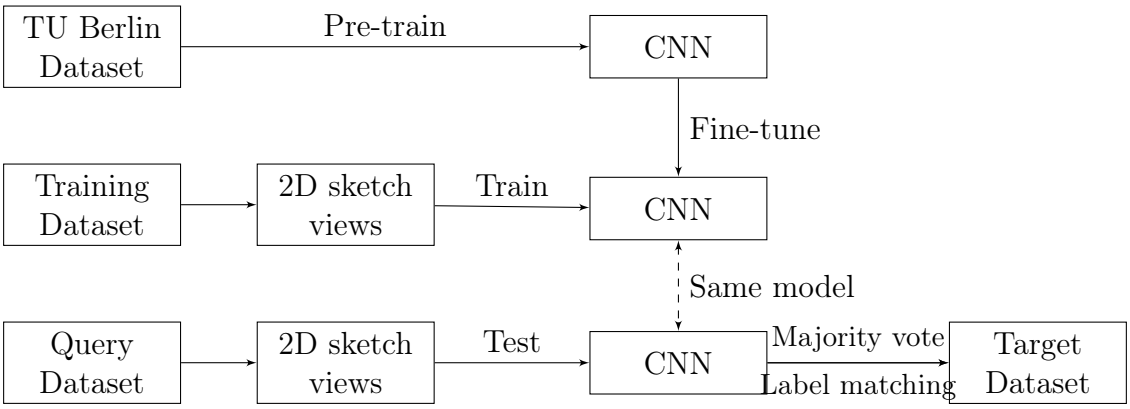


Figure V.3: Illustration of CNN-SBR architecture

To adapt the framework for 2D sketch-based CNN model, we need to convert the 3D sketches to 2D sketch views. We project all the coordinates in each 3D sketch to its six square faces, if we regard a 3D sketch as regular hexahedron, and

convert the coordinates to 2D depth image where the pixel value represents the distance to its view point (0 is the nearest while 255 is the furthest).
We use Sketch-a-Net as our core CNN model for our 3D model retrieval system. The motivation is the Sketch-a-Net is designed for single sketch recognition problem. We feed it with the TU Berlin dataset as the ancilary training dataset for pre-training. Although the TU Berlin dataset is obtained from human 2D sketching, it provides deep CNN a good representation of drawing features which could help deep CNN to build a initial learning weight for future fine-tuning on 3D sketch dataset.
In fine-tuning step, instead of employing mature learned pre-trained model (training epoch over 500), we choose the pre-trained model at epoch 50, which means the local optima has not been convergent and the learning weights are still flexible. We find using this method would significantly alleviate the over-fitting issue during fine-tuning.
For each 3D sketch, we use majority vote algorithm to choose the final classification label based on its six 2D sketch views. More specifically, for each 2D sketch view, we have a similarity vector for predicting categories. Thus, we have totally 6 similarity vectors and 6 top-1 labels for six sketch views. In order to rank the categories correctly, we use the following algorithm to evaluate the similarity vectors:

1. We rescale the similarity between a 3D sketch and target 3D model categories to range $[0, 1]$. Higher value means more similar.

2. We then compute the average similarity between this sketch and target 3D models based on 6 similarity vectors. Average similarity would also fall into range $[0, 1]$.

3. For each target 3D model category, we first count the number of top-1 label among 6 similarity vectors.

4. Finally, we rank all the target 3D model category using the score "top-1

label count + average similarity" for each 3D sketch.

Based on the above algorithm, we first consider those target 3D model categories have the most top-1 label count and rank them at the top places. We should note that, in the experiment, there are only 6 top-1 labels but 21 target 3D model categories. In this case, some target categories may have the same top-1 label count, then we need to compute the average similarity between the input 3D sketch and target 3D model categories and rank the rest of target models.

Comparison

There are 6 participants participate in the SHREC'16 challenge. In this section, we discuss three 3D model retrieval algorithms: (i) Localized Statistical Feature and Manifold Ranking (LSFMR), (ii) Histogram of Oriented Distances (HOD) and (iii) CNN-Point and CNN-Edge, which are proposed by Fan et al., Tabia et al. and Li, respectively.

LSFMR. LSFMR is based on Bag of Words (BoW) paradigm. It consists of two main components: (i) LSF extraction and (ii) Manifold ranking. In preprocessing, SVM is applied to remove the noise points of the 3D sketches while PCA is applied to normalize their position. Then, the local feature of the training 3D sketch data were clustered using k-means method. Note that the local feature is a new feature vector named as Localized Statistical Feature (LSF) proposed in this work. We will discuss this feature later. After clustering, a visual dictionary is built for these feature descriptors. In the implementation, the size of visual dictionary is 1024. After that, feature quantification is used to count the condition of the distribution from the different visual vocabulary in a 3D sketch. All the visual vocabulary weight of the 3D models are ranked based on (i) Primary index: the indexing of all 3D features applied visual vocabulary, (ii) Secondary index: the weight of the visual vocabulary. Figure V.4 shows the overall pipeline of LSFMR.

LSF feature describes the local region shape, which come with the dense grid

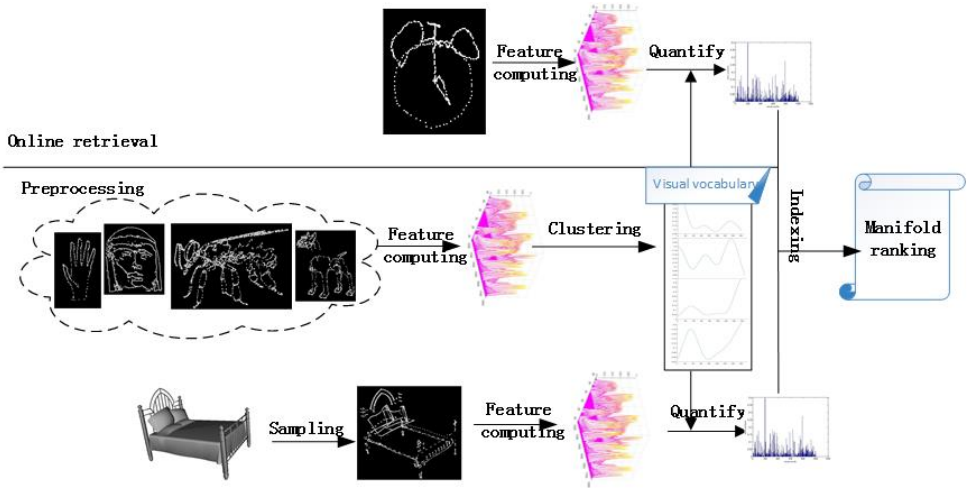


Figure V.4: Main steps of LSFMR

dividing, as the point statistical result. A dense network is built based on this feature and it capture both the local and global features. The local region is divided into $L \times L \times L$ cells to statistic the 3D points distribution in the local region. For each cell, the feature value is the number of points in the cell. The final representation of a local region is 1-D vector constructed by accumulation all the cell feature values. For comparing two LSF vectors, instead of employing Euclidean distance, χ^2 distribution is used for comparison.

$$\chi^2(F_1, F_2) = \sqrt{\sum_{c=1}^{L^3}(\frac{F_1(c) - E\chi(F_1)}{E\chi(F_1)}) + \sum_{c=1}^{L^3}(\frac{F_2(c) - E\chi(F_2)}{E\chi(F_2)})} \tag{V.1}$$

where F denotes LSF vector, $E\chi$ denotes the expectation of F.

In this work, manifold ranking is embedded into a high-dimensional Euclidean space that recovers the intrinsic structure of 3D shape. Two manifolds of LSF feature is created here. First one is from the comparison between a 3D sketch and each target 3D model, and is used to train the visual vocabulary. The second one is from the comparison of two different target 3D models. In second manifold, the 3D models are used as training data. In both two manifolds, precise retrieval results can be obtained using this method.

Given the LSF vectors $\chi = \{x_1, \cdots, x_q, x_{q+1}, \cdots, x_n\} \subset R^D$ of 3D sketches and

3D models, $r :\rightarrow R$ is defined as a ranking function which assigns to each point x_i a ranking score r_i, and a intial vector $p = [p_1, \cdots, p_n]^T$ is defined as a query 3D sketch and ith target 3D model. Thus, the cost function $C(r)$ can be described as:

$$C(r) = \frac{1}{2}\sum_{i,j=0}^{n} W_{ij}\|\frac{r_i}{\sqrt{D_{ii}}} - \frac{r_j}{\sqrt{D_{jj}}}\|^2 + \mu\sum_{i=0}^{n}\|r_i - p_i\|^2 \tag{V.2}$$

where W denotes the affinity matrix, D denotes the diagonal matrix $D_{ii} = \sum_j \sigma W_{ij}$, and $\mu > 0$ is a regularization parameter.
Lower cost function value means more accuracy of the ranking. Thus, $C(r)$ derivation operator is conducted by r and the convergent function of the sequence $r_i(t)$ can be defined as:

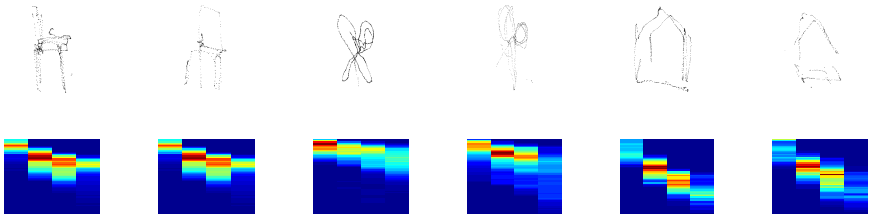
$$r^* = (I - \alpha S)^{-1}p \tag{V.3}$$

where S is derived from the symmetrical normalization of the matrix W, $S = D^{-1/2}WD^{-1/2}$, I denotes a unit matrix. Finally, we have the ranking mark $r = [r_1, \cdots, r_n]^T$.

HOD. The main idea of HOD is to build a dubbed Histogram of Oriented Distances descriptor on the joint distribution of two parameters accumulated in a 2D histogram. The algorithm is constructed as follows:

1. Randomly sample n points $P = p_i, i = 1 \ldots n$ from the 3D sketch.
2. Compute the Euclidean distance $d_i = \|p_i - p_j\|$ and measure the angle θ_{ij} between the two vectors $\overrightarrow{cp_i}$ and $\overrightarrow{cp_j}$, where c is the sketch's center of mass.
3. Compute the probability distribution of the distance $d \in \mathbb{R}^+$ and the orientation $\theta \in [0, \pi]$ of the sample pairs of points as a 2D histogram $h(d, \theta)$.

The global structure feature of the 3D sketches can be extracted using this representation. In addition, the similarity of the input 3D sketch between the



(g) (a) (h) (b) (i) (c) (j) (d) (k) (e) (l) (f)

Figure V.5: Example of HOD descriptors of some 3D sketch shapes.

target 3D model can be computed using the L_2 distance between the two distributions. This approach does not need to do extra preprocessing for both 3D sketch and target 3D model except the normalization for scale and random points sampling. Figure V.5 shows the histogram distribution of 64 bins for the distance and 4 bins for the orientation from six 3D sketches: (a) and (b) are from two chairs, (c) and (d) are from two scissors, while (e) and (f) are from two houses.

CNN-Point and CNN-edge. Li proposes the CNN-Point and CNN-Edge retrieval methods for this challenge. The only difference between these two methods is the processing techniques applied on query dataset. Based on this pipeline, all 3D models are converted to multiple representative images, then a classifier is trained on this image dataset, which consists of 1258 models and 120 images per each. To the target dataset, noise is randomly added on each 3D model according to their length. Then, this point cloud model is rendered to 128×128 gray-scale images from 120 uniformly distributed points of view. Meanwhile, the query dataset is also converted to point-based or edge-based 3D sketch according to pre-defined setting. Figure V.6 shows this 3D model conversion. The overall pipeline of CNN-Point and CNN-Edge is listed in Figure V.7.

To comprehensively evaluate the performance of the participant retrieval systems, we perform the comparison on 6 widely used evaluation metrics [Shilane et al. (2004)]: Nearest Neighbor (NN), First Tier (FT), Second Tier (ST), E-Measure (E), Discounted Cumulative Gain (DCG) and Average Precision (AP). The final result is listed in Table V.6.

Figure V.6: From left to right are 3D model, noised 3D model, point-based 3D sketch and edge-based 3D model

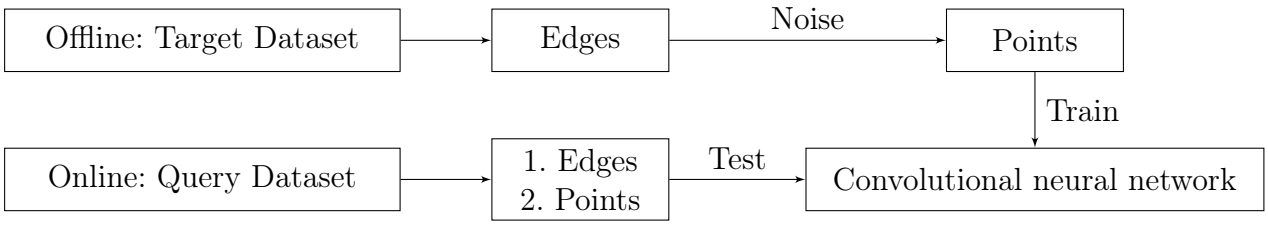


Figure V.7: Shape retrieval pipeline

From the above table, we can see that our CNN-SBR retrieval system beats other participant algorithms on the all evaluation metrics.

Table V.6: Performance metrics comparison on the SHREC'16 3D Sketch Track Benchmark.

Participant	**Method**	**NN**	**FT**	**ST**	**E**	**DCG**	**AP**
LL	3DSH	0.029	0.021	0.038	0.021	0.254	0.029
Fan	LSFMR	0.033	0.020	0.033	0.018	0.248	0.032
Li	CNN-Point	0.124	0.044	0.075	0.046	0.294	0.060
	CNN-Edge	0.114	0.056	0.084	0.051	0.302	0.063
Tabia	HOD1-4	0.029	0.015	0.035	0.026	0.259	0.032
	HOD64-1	0.052	0.031	0.053	0.034	0.274	0.044
	HOD64-2	0.067	0.031	0.057	0.032	0.272	0.044
	HOD64-4	0.124	0.019	0.022	0.013	0.230	0.026
Ye	CNN-SBR	0.222	0.251	0.320	0.186	0.471	0.314
Yin	CNN-Maxout-Siamese	0.000	0.031	0.108	0.048	0.293	0.072

VI. CONCLUSION AND FUTURE WORK

Recently, sketch recognition has been a highly popular research area due to its wide applications. Scene sketch understanding is a much more challenging research problem than conventional single-object sketch recognition tasks. However, there is no prior scene sketch benchmark and research work related to complexity scene sketch understanding. Prior research work focus on single-object sketch recognition and employ traditional photo-oriented schemes for this task. Recently, with the popularity of deep CNN, advanced deep CNN model has been proposed for single-object sketch recognition. We are inspired by this novel methodology and apply it to our scene sketch understanding work. In SHREC'16, we extend our work to 3D sketch-based 3D shape retrieval and build a novel CNN based 3D sketch retrieval system CNN-SBR which achieves very promising result and beats other competition retrieval systems.

In this work, we make the first effort to understand human's scene sketches. We build the first scene sketch dataset "Scene250" and open it to public. We also propose, implement, and test a novel deep CNN model "Scene-Net" on scene sketch recognition. We perform fine-tuning to reduce over-fitting and improve robustness of our "Scene-Net" model. The experiment results show that Scene-Net outperforms other two deep CNN models (AlexNet and Sketch-a-Net) by 7.50% and 21.25% on scene recognition respectively.

In SHREC'16 challenge, we explore the potential of deep CNN in 3D sketch-based shape retrieval problem which is an innovative and very challenging problem in current vision community. Integrated with 3D vision techniques, we convert the 3D sketches to the depth images of their 2D sketch views. We find applying this method would not only keep original 3D space information of a 3D sketch, but also adapt the 3D sketch data to traditional 2D deep CNN framework. In addition, we employ fine-tuning and data augmentation methods in this retrieval system and find it significantly alleviate the over-fitting problem.

From the evaluation result provided from challenge organizer, our CNN-SBR system wins the 1st place among 6 teams and its performance surpasses the 2nd place by about 100%.

In the future, further improvement work include collecting a large-scale scene sketches dataset, optimizing Scene-Net architecture to reduce training time, and applying our method to real world 3D model retrieval application.

APPENDIX SECTION

APPENDIX A

MatConvNet

MatConvNet (`http://www.vlfeat.org/matconvnet/`) is a MATLAB toolbox implementing CNNs for computer vision applications. It is distributed under BSD license. The main features include: It is simple, efficient, and can run and learn state-of-the-art deep CNNs. Many pre-trained deep CNNs for image classification, segmentation, face recognition, and text detection are available.

1. Flexible. CNN layers are written and implemented directly in Matlab code, which lets user to modify, extend, or integrate existing CNN easily.

2. Power. With the support of modern GPU and CUDA technology, MatConvNet can run large CNN models such as AlexNet.

3. Efficiency. The computation could be switched between CPU and GPU. Several pre-trained CNN model are also provided.

MatConvNet provides two wrappers for users to implement CNN models: (1) SimpleNN. A lightweight wrapper designed for CNN constructing of a single chain of layers. (2) DagNN. DagNN denotes directed acyclic graph neural network and is used to implement complex deep CNNs. The layers of DagNN are connected to local variables and vice-verse, which consists of a bipartite directed acyclic graph structure. In our implementation, we use SimpleNN wrapper due to the straightforward structures of the deep CNNs used in the experiments. We use MatConvNet as the main library to implement the deep CNN models. Since Matlab provides abundant APIs for image processing, we just need to focus on core deep CNN design.

APPENDIX B

Caffe

Caffe [Jia et al. (2014)] (`http://caffe.berkeleyvision.org/`) is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is the implemented in C++, and it also provides the wrappers for Matlab and Python. It is distributed under the BSD 2-Clause license.
Since Caffe has abundant existing deep CNN models where the parameter configuration is highly optimized, we transfer some deep CNN models from Caffe to MatConvNet with its highly optimized parameters. We also employ Caffe to validate the experimental results, that is re-run the same pipeline of a deep CNN model in both MatConvNet and Caffe and see if the result is consistent. Insert supplementary material here.

REFERENCES

Bronstein, A. M., Bronstein, M. M., Guibas, L. J., and Ovsjanikov, M. (2011). Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Trans. Graph.*, 30(1):1:1–1:20.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Eitz, M., Hays, J., and Alexa, M. (2012a). How do humans sketch objects? *ACM Trans. Graph.*, 31(4):44:1–44:10.

Eitz, M., Hildebrand, K., Boubekeur, T., and Alexa, M. (2011). Sketch-based image retrieval: Benchmark and bag-of-features descriptors. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1624–1636.

Eitz, M., Richter, R., Boubekeur, T., Hildebrand, K., and Alexa, M. (2012b). Sketch-based shape retrieval. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):31:1–31:10.

Everingham, M., Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338.

Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., and Jacobs, D. (2003). A search engine for 3d models. *ACM Trans. Graph.*, 22(1):83–105.

Funkhouser, T. and Shilane, P. (2006). Partial matching of 3d shapes with priority-driven search. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 131–142, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Furuya, T. and Ohbuchi, R. (2013). Ranking on cross-domain manifold for sketch-based 3d model retrieval. In *Cyberworlds (CW), 2013 International Conference on*, pages 274–281.

Gong, B., Liu, J., Wang, X., and Tang, X. (2013). Learning semantic signatures for 3d object retrieval. *IEEE Transactions on Multimedia*, 15(2):369–377.

Griffin, G., Holub, A., and Perona, P. (2007). Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology.

Herot, C. F. (1976). Graphical input through machine recognition of sketches. *SIGGRAPH Comput. Graph.*, 10(2):97–102.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

Hu, R. and Collomosse, J. (2013). A performance evaluation of gradient field {HOG} descriptor for sketch based image retrieval. *Computer Vision and Image Understanding*, 117(7):790 – 806.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

Kazhdan, M., Chazelle, B., Dobkin, D., Finkelstein, A., and Funkhouser, T. (2002). A reflective symmetry descriptor. In *7th European Conference on Computer Vision (ECCV 2002)*, pages 642–656.

Klare, B., Li, Z., and Jain, A. (2011). Matching forensic sketches to mug shot photos. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(3):639–646.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan-Kaufmann.

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

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. R. (2002). Neural networks: Tricks of the trade. In *Efficient BackProp*, pages 9–50. Springer.

Li, B., Lu, Y., Godil, A., Schreck, T., Aono, M., Johan, H., Saavedra, J. M., and Tashiro, S. (2013). SHREC'13 Track: Large Scale Sketch-Based 3D Shape Retrieval. In Castellani, U., Schreck, T., Biasotti, S., Pratikakis, I., Godil, A., and Veltkamp, R., editors, *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association.

Li, B., Lu, Y., Li, C., Godil, A., Schreck, T., Aono, M., Burtscher, M., Chen, Q., Chowdhury, N. K., Fang, B., Fu, H., Furuya, T., Li, H., Liu, J., Johan, H., Kosaka, R., Koyanagi, H., Ohbuchi, R., Tatsuma, A., Wan, Y., Zhang, C., and Zou, C. (2015). A comparison of 3d shape retrieval methods based on a large-scale benchmark supporting multimodal queries. *Computer Vision and Image Understanding*, 131:1 – 27. Special section: Large Scale Data-Driven Evaluation in Computer Vision.

Lu, T., Tai, C.-L., Su, F., and Cai, S. (2005). A new recognition model for electronic architectural drawings. *Computer-Aided Design*, 37(10):1053 – 1069.

Mohamad, M., Shafry, M., Rahim, M., Othman, N., and Jupri, Z. (2009). *A comparative study on extraction and recognition method of CAD data from CAD drawings*, pages 709–713.

Osada, R., Funkhouser, T., Chazelle, B., and Dobkin, D. (2002). Shape distributions. *ACM Trans. Graph.*, 21(4):807–832.

Ouyang, S., Hospedales, T., Song, Y.-Z., and Li, X. (2014). Cross-modal face matching: Beyond viewed sketches. *the 12th Asian Conference on Computer Vision*.

Ouyang, T. Y. and Davis, R. (2011). Chemink: A natural real-time recognition system for chemical drawings. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, IUI '11, pages 267–276, New York, NY, USA. ACM.

Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173.

Schneider, R. G. and Tuytelaars, T. (2014). Sketch classification and classification-driven analysis using fisher vectors. *ACM Trans. Graph.*, 33(6):174:1–174:9.

Shilane, P., Min, P., Kazhdan, M., and Funkhouser, T. (2004). The Princeton shape benchmark. In *Shape Modeling International*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

Sousa, P. and Fonseca, M. J. (2009). Geometric matching for clip-art drawing retrieval. *J. Vis. Comun. Image Represent.*, 20(2):71–83.

Stanford (2015). Backpropagation algorithm (http://ufldl.stanford.edu/wiki/index.php/backpropagation_algorithm).

Stanford (2016). Cs231n: Convolutional neural networks for visual recognition (http://cs231n.github.io/convolutional-networks).

Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE Design Automation Workshop*, DAC '64, pages 6.329–6.346, New York, NY, USA. ACM.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.

Wang, F., Kang, L., and Li, Y. (2015). Sketch-based 3d shape retrieval using convolutional neural networks. *CoRR*, abs/1504.03504.

Xiao, J., Hays, J., Ehinger, K., Oliva, A., and Torralba, A. (2010). Sun database: Large-scale scene recognition from abbey to zoo. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3485–3492.

Yu, Q., Yang, Y., Song, Y.-Z., Xiang, T., and Hospedales, T. (2015). Sketch-a-net that beats humans. In *arXiv:1501.07873*.