An Investigation into Visual Content Understanding Based on Deep Learning and Natural Language Processing

Ke Sun

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy

July 2018

Abstract

A long standing goal of artificial intelligence is to enable machines to perceive the visual world and interact with humans using natural language. To achieve this goal, many computer vision and natural language processing techniques have been proposed during the past decades, especially deep convolutional neural networks (CNNs). However, most previous work mainly focus on the two sides separately, and few work have been done by connecting the vision and language modalities. Hence, the semantic gap between the two modalities still exists.

To solve this, the overall objective of my PhD research is to design machine learning algorithms for visual content understanding by connecting the vision and language modalities. Towards this goal, we have developed several deep learning models combined with natural language processing techniques to represent and analyze the image/video and text data. We focus on a series of applications to demonstrate the effectiveness of the proposed models and obtain promising results.

Firstly we show that tag-based image annotations exhibit many limitations for visual content representation, and then develop techniques to discover visual themes as an alternative by re-organizing the original image and tag set into a group of visual themes. More concretely, we extract visual feature and semantic feature from two trained deep learning models respectively, and then design a method to effectively evaluate the similarity between a pair of tags both visually and semantically. Next we cluster these tags into a set of visual themes based on their joint similarities. We conduct human-based evaluation and machine-based evaluation to demonstrate the usefulness and rationality of the discovered visual themes and indicate their potential usage for automatically managing user photos.

Secondly we develop a novel framework for understanding complex video scenes involving objects, humans, scene backgrounds and the interactions between them. In this work, we propose to automatically discover semantic information for these videos from an unsupervised manner. We do this by introducing a set of semantic attributes

derived from a joint image and text corpora. Then we re-train a deep constitutional neural network to produce visual and semantic features simultaneously. Our trained model encodes the complex information including the whole visual scene, object appearance/property and motion. We apply our model to solve the video summarization problem by adopting a partially near duplicate image discovery technique to cluster visually and semantically consistent video frames together. The experimental results demonstrate the effectiveness of the semantic attributes in assisting the visual features in the video summarization and our new technique (SASUM) achieves state-of-the-art performance.

Thirdly we decide to depict and interpret the traffic scene using vehicle objects. Firstly, we collect an hour-long traffic video with a resolution of 3840×2160 at 5 busy intersections of a megacity by flying an UAV during the rush hours. We then build a UavCT dataset containing over 64K annotated vehicles in 17K 512×512 images. In the next, we design and train a deep constitutional neural network from scratch to detect and localize different types of road vehicles (i.e. car, bus and truck), and propose a fast online tracking method to track and count vehicles in consecutive video frames. We design vehicle counting experiments on both image and video data to demonstrate the effectiveness of the proposed method.

Fourthly we extend our work in the previous stage and explore more potential applications of our deep model for traffic scene understanding. Specifically, we first track all the target vehicles from the original video, and then recognize vehicle behaviors based on nearest neighbor search (clustering) and bidirectional long short-term memory (classification). By conducting comparative studies, we further demonstrate the effectiveness and versatility of our approach for object-based visual scene understanding.

As last, we focus on TV series video understanding and develop techniques to recognize characters in these videos. Using label-level supervision, we transform the problem to multi-label classification and design a novel semantic projection network (SP-Net) consisting of two stacked subnetworks with specially designed constraints. The first subnetwork is aiming to reconstruct the input feature activations from a trained single-label CNN, and the other one functions as a multi-label classifier which predicts the character labels as well as reconstructing the input visual features from the mapped semantic label space. We show such kind of mutual projection significantly benefits the character recognition by conducting experiments on three popular TV series video datasets. We also show that region-based prediction strategy could further improve the overall performance.

Our work contribute to a developing research field that demonstrates the power of deep learning techniques in solving different visual recognition problems by connecting the vision and language modality and advance the state-of-the-art on various tasks.

Acknowledgements

This thesis would not have been possible without the guidance, support, and love from many people around me in the four amazing years of my PhD life.

First and foremost, I would like to especially thank my supervisor Prof. Guoping Qiu for his encouragement, insight and patience. I am fortunate to be one of the students in Guoping's research group, and have appreciated his seemingly unlimited supply of passion and kindness. I have learned a lot from him, not only how to conduct high-quality research, but also invaluable advices in my life.

I am also grateful to my close collaborators who have become my role models in various respects. It has been my great honor to collaborate with Prof. Jiasong Zhu from whom, I learned incredible cross-discipline knowledge. I would like to thank Qian Zhang, Chao Zhang, Zhan Xu, Bolei Xu, Jingxin Liu, Zhuo Lei, Fei Yang, Qing Li and Rui Cao for the insightful research discussions and great moments. I am also thankful to Dr. Bozhi Liu for his kind help during my visiting in Shenzhen University.

I would like to thank the University of Nottingham for supporting my research, including the International Doctoral Innovation Centre (IDIC) and the Horizon Centre for Doctoral Training (CDT). I would also like to thank Emma Juggins and Jessica Wang for their kind administrative support in UNUK and UNNC.

I also want to thank many people in my daily life who create many precious memories. I would like to thank Xianxu Hou, Dongsheng Zhao, Long Wen, Fengyu Zhang, Chuang Gao, Xiaoping Jiang, Liming Xu, Linbin Yao, Jorge Mendez Astudillo, Yitong Huang, Kate Arnold, Sam Doehren, Gregor Engelmann, Iona Fitzpatrick, Shadab Mashuk, Obrien Sim, Pepita Stringer, Tatiana Styliari, Matt Voigts and Alex Young.

Last but not least, I would not be where I am today without the amazing support, encouragement and love from my parents Guilian Liu and Yuancheng Sun, and my elder brother Kai Sun. It's the passion, exploration and adventure combined with determination that I learnt from you. My appreciation to them is beyond what I can express in words.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Overview

Humans can easily achieve to classify a given image or identify different objects depicted in that image. A quick glimpse at an image is enough for us to catch all the important visual details. Besides, we can easily obtain adequate semantic information from the visual world including the types and locations of objects within it, and the concurrent actions/events, etc. Taking Figure 1.1 as an example, we can immediately point out the "a male cartoon character" with just a glance, and then notice the man is holding "a grey golf club", and the "golf ball above the lawn". Some people may simply summarize the whole image as "a man playing golf".

Since we do visual recognition so naturally and effortlessly, we may not realize how hard this task is for computer vision (CV) algorithms. Visual recognition and other machine vision perception cannot be easily solved by manually designing rules which are used for processing visual inputs. In the scene above, a man hits a golf ball with a grey golf club. On the CV side, it involves many interpretations of the depicted cartoon image to be recognized by computers from an object level. And the recognized objects (including interactions) need to be reconstructed back together to form accurate and complete recognition results [5]. Because an image stored in computers is made up of tiny dots named pixels, the vision algorithms must transform these pixel values into high-level semantic meanings such as a "man" and a "golf ball". Besides, different races of men with various poses could still be described using same sentence, but the lighting condition, camera angles and the artistic styles (e.g. cartoons, painting, photos, etc.) might be totally different. On another aspect, some similar low-level image patterns (e.g. furry pattern) might be components of a large number of different object (e.g. hats, shoes, tigers, wolves, etc.).

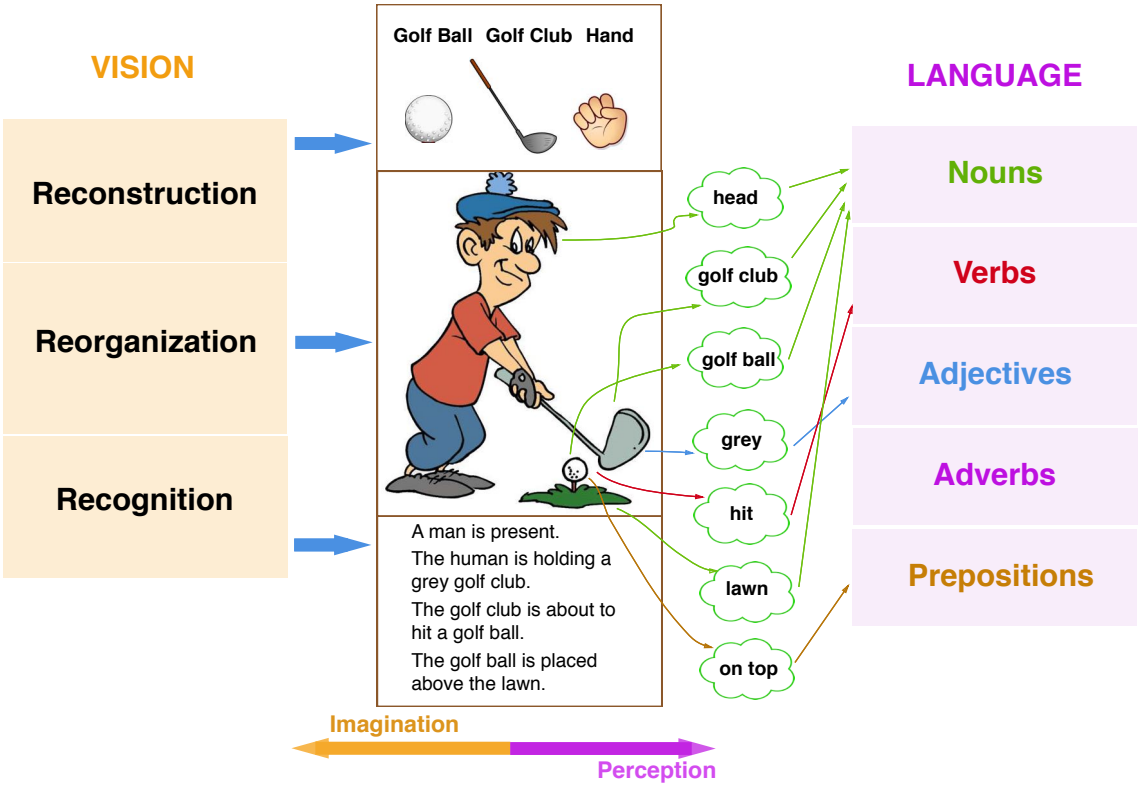


Figure 1.1: The connection between vision and language. **Left:** the process of visual perception. **Middle:** the example image and the natural language description. **Right:** the process of natural language understanding.

The story is also not simple on the language side. A natural language description such as "a man playing golf" might be rephrased using many different words (e.g. synonyms) and phrases (e.g. variation of word order), due to human's subjectivity of language understanding. For example, images containing lakes, rivers and ocean could all be annotated with *water*, or we can just use *lake*, *river*, *ocean* respectively. Some people may just describe such an image as a "a beautiful scene". It is a very natural case but this often confuses CV algorithms since they are forced to distinguish similar visual contents. Additionally, some words in an image caption often refer to particular parts in that image, and understanding such an image actually involves a complicated region-based pattern recognition process which is aiming to detect and identify salient regions from a wide range of pixels. The aforementioned issues bring forward the problem of semantic image/video understanding, i.e. enabling computers to effectively connect the two modalities of vision and natural language, and accurately extract human understandable meaning from the vast visual world.

1.2 Contributions

In this thesis we focus on developing models and machine learning techniques to connect the vision and language modalities for computer vision applications.

We first develop techniques to discover Visual Theme (VT) as a better (more compact, efficient and effective) alternative for tag-based image annotations. A visual concept is denoted as a subset of human language vocabularies that refer to particular visual entities (e.g. fireman, policeman). We discover visual themes by clustering user tags according to the combination of visual similarity (based on deep features) and semantic similarity (based on word embeddings) using a spectral clustering algorithm. The discovered visual themes are demonstrated to be more effective via user study and common computer vision tasks.

In the second stage, we show that tag based image annotation is subjective and suffers from the diversity of natural language understanding, and consider generating semantic representation from the visual information directly. More concretely, we train a deep convolutional neural network to automatically obtain joint (visual and semantic) feature representation from the images themselves without having to explicitly rely on human supervision. We validity our proposal by conduct the video summarization experiment and obtained promising results.

In the third stage, we focus on the total scene understanding by inspecting various objects in it. We first collect a large-scale high resolution traffic video dataset by flying an UAV during rush hours, and then annotate the video frames using bounding boxes and vehicle types. Next, build a deep learning based vehicle counting framework which is capable of detecting, tracking and counting vehicles in 4K high resolution UAV videos. To evaluate the proposed framework, we conduct experiments on five complicated traffic scenes by counting different types of vehicle in testing images and videos. The results demonstrate the effectiveness and superiority of the proposed method over other state-of-the-art methods.

In the fourth stage, we extend our work in the previous stage and recognize vehicle behaviors by analyzing their trajectories in the traffic videos. We focus on behavior recognition at road intersections and approach this problem via unsupervised (nearest neighbor search) and supervised (bidirectional long short-term memory network) manners respectively. The experimental results demonstrated the effectiveness and superiority of the proposed methods.

In the final stage, we focus on TV series video understanding and design a method to automatically recognize characters in them. The proposed semantic projection net-

work (SPNet) consists of two stacked subnetworks with specially designed constraints for different purposes. More specifically, the first subnetwork is a contractive autoencoder which focuses on reconstructing visual feature activations extracted from a pre-trained CNN, while the second subnetwork functions as a multi-label classifier with additional constraints which require to reconstruct input visual features from the projected semantic space. Besides, we also design the region-based classification strategy to further improve the overall performance. Experiment results on three challenging TV series show that the proposed method achieves very promising performance, and demonstrate the effectiveness of autoencoders for character recognition.

1.3 Outline

In order to make each part of the thesis self-contained, we will leave the background and related work to each individual chapter. The rest of thesis is organized as follows.

In **Chapter 2**, we describe relevant mathematical knowledge for unsupervised learning, supervised learning, convolutional neural networks, and general strategies of network architectural design for visual information processing. We also provide some fundamental knowledge of natural language processing, including word segmentation, part-of-speech tagging and word embedding.

In **Chapter 3**, we introduce our work on visual theme discovery. Given an dataset containing images and tag-based annotations, we cluster tags into visual themes based on their visual similarity and semantic similarity measures using a spectral clustering algorithm. We conduct both user studies and experiments to evaluate the effectiveness and rationality of the discovered visual themes, and obtain promising results. The content of this chapter published in [6].

In **Chapter 4**, we describe our work on semantic attributed based user video summarization. Firstly, we use a natural language processing tool to discover a set of keywords from an image and text corpora to form the semantic attributes of visual contents. Secondly, we train a deep constitutional neural network to extract visual features as well as predict the semantic attributes of video segments which enables us to represent video contents with visual and semantic features simultaneously. Thirdly, we construct a temporally constrained video segment affinity matrix and use a partially near duplicate image discovery technique to cluster visually and semantically consistent video frames together. These frame clusters can then be condensed to form an informative and compact summary of the video. The experimental results demonstrate the effectiveness of the semantic attributes in assisting the visual features in video summarization. The

content of this chapter is published in [7].

In **Chapter 5**, We introduce our work on traffic scene understanding by detecting, tracking and counting vehicles in road traffic videos. We first capture nearly an hour-long ultra high-resolution (4K) traffic video at 5 busy road intersections of a modern megacity by flying an UAV during the rush hours. We then develop a novel traffic analysis framework consisting of deep neural network based vehicle detection and localization, type (car, bus and truck) recognition, tracking and vehicle counting over time. In the experiments we show that our model outperforms other deep neural network based techniques and that deep learning techniques are more effective than traditional computer vision techniques in urban traffic analysis. The content of this chapter is published in [8].

In **Chapter 6**, we introduce our work on vehicle behavior recognition for city road traffic. Specifically, we first detect and track different types of road vehicles in traffic videos, and then we design a double spectral clustering method and a bidirectional long short-term memory method to identify different types of vehicle behaviors. By conducting comparative studies, we further demonstrate the powerfulness and versatility of our deep model for urban traffic scene understanding. The content of this chapter is published in [9] and [10].

In **Chapter 7**, we describe our work on character recognition in TV series videos using label-level supervision. We transform the problem to multi-label classification and design a novel semantic projection network (SPNet) to effectively recognize character in video frames. The proposed SPNet contains two stacked subnetworks with different constraints. The experiment results on three popular TV series video datasets demonstrate the effectiveness of the proposed model. The content of this chapter is published in [11].

Finally, in **Chapter 8** we summarize our work and discuss the path forward.

CHAPTER 2

Relevant Background

In computer vision, a long standing and open issue is how to enable computers or intelligent agents to automatically extract useful semantic information and make interpretations as human do. However it is too difficult to be solved by manually designed rules for computers to follow. Instead, various computer vision algorithms are designed for machines to learn the inherent patterns and powerful visual representations from a set of image/video data, and the learned feature representations can be then applied solve different computer vision problems. This practical approach is usually known as machine learning that enable computers to learn without being explicitly programmed. This chapter provides the necessary background on machine learning, deep learning, and natural language processing. For more details, we recommend the Pattern Recognition and Machine Learning book from CM Bishop et al. [12], the Deep Learning book from Goodfellow et al. [13] and the Foundations of Statistical Natural Language Processing book from CD Manning et al. [14].

2.1 Unsupervised Learning

Many machine learning problems can be formulated as training an artificial intelligence (AI) algorithm to perform a mapping function $f : X \rightarrow Y$, where $X = \{X_1, X_2, ..., X_p\}$ stands for variables in the input feature space, and Y is the outcome or response variable in the output space. The goal is then to predict Y using X, and this kind of machine learning is named as the supervised learning.

However, there are also many problems where we only have the input data X and no corresponding output variables exist. Instead of predicting something, the goal is to explore the underlying structure or distribution of data in the input space in order to learn more about the data themselves. Such kind of machine learning is called un-

supervised learning since no correct answer or guidance is available. Compared to supervised learning, there is no common metrics to evaluate how well the algorithm is doing: the performance is often subjective and domain-specific.

We will discuss two unsupervised learning techniques which we used in our experiments: clustering and dimensionality reduction.

2.1.1 Clustering

Clustering refers to the technique which is aiming to find subgroups or clusters in an unstructured and unlabelled dataset. In each distinct group, the data observations are quite similar to each other in some aspects, while they significantly differ in different groups. Anyway, the definition of being *similar* or *different* depends on the knowledge (domain-specific) of the data being studied. As we may not know what we're looking for, clustering is useful for discovery rather than prediction. It provides an insight into the inherent groups found within the dataset.

K-means Clustering

The K-means algorithm is probably the most popular clustering method, and it serves as the foundation of many sophisticated clustering algorithms. Assuming that we have a high dimensional feature space where each data observation is represented using a data point, then the K-means algorithm is aiming to cluster these points into k groups. A large value of k generates small groups with fine granularity, while a small value of k creates large groups with coarse granularity.

The K-means algorithm is formally described in Algorithm 1. Starting with some data points, the K-mean algorithms first randomly selects K point as the initial centroids, then iteratively assigns data points to its nearest centroid by some distance measure function and recompute all the centroids. The algorithm converges when the centroids do not change. Two commonly used distance functions are the Euclidean distance and the Cosine distance.

Spectral Clustering

Compared to the K-means algorithm which aims to condense data points into compact groups, the spectral clustering is aiming to inspect the intrinsic connectivity in the discrete data points. To achieve this, the algorithm represents data points as vertices V of a graph G, where vertices V are connected by edges E, and these edges have corresponding weights W. A large weight means that the adjacent vertices are quite similar, while a small weight implies certain dissimilarity between them.

Algorithm 1 The basic K-means algorithm.

Input: Initial data points.

Output: K data clusters with centroids $\{C_1, C_2, ..., C_K\}$.

1: Randomly select K data points as the initial centroids.
2: **repeat**
3: Calculate the distance between each data point and each centroid in the K clusters.
4: Assign data points to its closest centroid.
5: Recompute the centroid of each cluster.
6: **until** All centroids stop changing.

Hence, the goal of spectral clustering is straightforward: given data points $X = \{x_1, x_2, ..., x_n\}$ and the similarity matrix containing pairwise similarities $w(x_i, x_j)$, partitioning the data points into groups so that points within a group are similar and points in different groups are dissimilar. The similarity measure $w(x_i, x_j)$ can be some distance functions, such as the Euclidean distance, the Cosine distance and the Gaussian types. The Gaussian type is commonly used in spectral clustering and is defined as:

$$w(x_i, x_j) = exp\frac{-|x_i - x_j|^2}{\sigma^2} \tag{2.1.1}$$

where σ controls the width of the neighborhoods.

Apart from the distance measure, there are also different approaches to construct the data graph. For instance, in a fully connected graph, all weights $w(x_i, x_j)$ are not null; in a k-nearest neighbor graph, each vertex is only connected to its k nearest neighbors; and in a r-neighbor graph, each vertex is connected to other vertices where their weight w falls inside the value of r. Some other graphs are constructed using both k and r to control the local structure of the data points.

Here we provide an overview of two popular spectral clustering algorithms, the unnormalized spectral clustering and the normalized spectral clustering in Algorithm 2. Assuming that we have n data points $X = \{x_1, x_2, ..., x_n\}$, and we have computed their pairwise similarities $W(x_i, x_j)$ by some similarity function which is symmetric and non-negative, and we denote the similarity matrix by $W = (w_{ij})_{i,j=1,...,n}$. To perform the algorithm, we also need an essential matrix named the unnormalized graph Laplacian matrix L, which is defined as:

$$L_{ij} = D_{ij} - W_{ij} \tag{2.1.2}$$

where D is the diagonal degree matrix assigned to the graph vertices, and W refers to the similarity matrix. For a more systematic and complete explanation for Laplacian

matrix, we refer to the paper of Daniel A Spielma et al. [15].

Algorithm 2 The spectral clustering.

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.

1: Construct a similarity graph by using some distance measure. Denote W as the weighted adjacency matrix.
2: Compute the unnormalized Laplacian matrix L.
3: **if** perform the unnormalized spectral clustering **then**
4: **Compute the first** k **eigenvectors** $\{u_1, u_2, ..., u_3\}$ **of** L**.**
5: **else if** perform the normalized spectral clustering **then**
6: Compute the first k generalized eigenvectors $\{u_1, u_2, ..., u_3\}$ of the generalized eigenproblem $Lu = \lambda Du$.
7: **end if**
8: Denote $U \in \mathbb{R}^{n \times k}$ as the matrix which contains the eigenvectors $\{u_1, u_2, ..., u_3\}$ as its columns.
9: **for** $i = 1; i < n; i++$ **do**
10: let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
11: **end for**
12: Using the k-means algorithm to cluster the points $(y_i)_{i=1,...,n} \in \mathbb{R}^k$ into k clusters $\{C_1, C_2, ..., C_k\}$

Output: Clusters $M_1, M_2, ..., M_i$ with $M_i = \{j | y_j \in C_i\}$.

Compared to traditional clustering methods like k-means, spectral clustering only take the similarity matrix into consideration, which makes it very helpful to process sparse data. Besides, the built-in character of dimensionality reduction allows it to effectively tackle high-dimensional data.

2.1.2 Dimensionality Reduction

In machine learning, the term "dimensionality" simply refers to the number of features in the dataset. In the digital era, real-world data often comes with too many features which results in a very high dimensionality (see Chapter 1), and this poses significant challenges for machine learning algorithms to work on it. Sometimes, most of these features are redundant or even irrelevant, which introduces unnecessary computational cost and further leads to performance degradation. Hence, that's where we need dimensionality reduction techniques to fight against these problems.

The dimensionality reduction can be conducted in two ways: by only reserving the most relevant or useful features from the original dataset (this is called feature selec-

tion) or by transforming the original data from a high dimensional feature space to a new feature space with less number of dimensions (this is called feature extraction). In this work, we mainly introduce a very popular feature extraction technique named principal component analysis (PCA). For other dimensionality reduction techniques, please refer to the paper of C.O.S. Sorzano et al. [16].

Principal Component Analysis

The Principal Component Analysis (PCA) [17] is an unsupervised algorithm that creates principal components (new features) by linearly combining the original features. The newly created components are orthogonal, which means that they are uncorrelated. Besides, all components are ranked according to their "variance". The first principal component (PC_1) contains the most variance in your dataset, PC_2 holds the second-most variance, and so on.

Therefore, one can reduce dimensionality of the original dataset by limiting the number of principal components. For instance, we can decide to reserve as many principal components based as needed to keep a cumulative variance above 95%.

Assuming that we have the input image data matrix $X \in \mathbb{R}^{n \times d}$ where n denotes the number of images and d represents the original number of image feature dimensions, we can describe the working flow of PCA in Algorithm 3. Detailed explanations of some basic linear algebra concepts like eigenvalues, eigenvectors, and singular value decomposition could be found in the Introduction to Linear Algebra book from Gilbert Strang et al. [18].

Algorithm 3 The principal component analysis.

Input: Data matrix $X \in \mathbb{R}^{n \times d}$, number of reserved pricinpal components k.

Output: Transform matrix R.

1: Compute the mean value $\bar{x} = \frac{1}{n} \sum_{n}^{i=1} x_i$.
2: subtract $\bar{x}$ from each row x_i in X.
3: Compute the covariance matrix $M_{cov} = \frac{1}{n} X^T X$.
4: Compute eigenvalues $\{e_1, e_2, ..., e_d\}$ of M_{cov} and sort them.
5: Compute matrix V which satisfies $V^{-1} \times M_{cov} \times V = D$, where D is the diagonal matrix of the eigenvalues of M_{cov}.
6: Keep the first k column of V as the transform matrix R.

2.2 Supervised Learning

In supervised learning problems, we start with a data set containing data samples with associated ground-truth labels or values. For instance, when recognizing handwritten digits, a supervised learning algorithm accepts hundreds or thousands of handwritten digit pictures together with their labels indicating the number (from 0 to 9) each image represents. The goal of the algorithm is to learn a mapping function f to model the relationship ($Y = f(X)$) between images and their associated numbers. When the learning process completes, we apply the learned model to predict possible numbers represented by unseen images. The overall goal of the supervised learning algorithm is to generalize this function f so that it performs well on unknown samples.

Supervised learning problems can be grouped into the regression and the classification problems depend on the type of the output. The task of classification is to assign a label or category to a testing sample (e.g. the handwritten digit recognition), while the regression problem predicts a continuous numerical value such as "weight" or "price".

2.2.1 The Learning Objective

Assuming that we have a training data set T_n consisted of pairs of input and output points $\{(x_1, y_1), ..., (x_n, y_n)\}$ generated from some joint probability distribution function $P(x, y)$, and the data samples in T_n is independent and identically distributed (i.i.d). We then design appropriate learning function that ideally satisfy $y = f(x)$. More specifically, we consider some particular mapping functions $\mathbb{F}$ and choose a loss function $L(\hat{y}, y)$ which quantifies the differences between a predicted label $\hat{y}_i$ using $f \in \mathbb{F}$ and a ground-truth label y_i. The choices of loss function L could vary according to the practical need. In the case of pattern recognition where $y = \{-1, +1\}$, a common choice of L is the misclassification error:

$$L(f(x), y) = \frac{1}{2}|f(x) - y| \tag{2.2.1}$$

The loss function L leads to the definition of the risk (or *generalization error*) for the learning function f:

$$R(f) = \int L(f(x), y) dP(x, y) \tag{2.2.2}$$

Therefore, the learning objective is expressed as a minimization of R for any mapping function $f \in \mathbb{F}$. Unfortunately, it is impractical because the joint probability distribu-

tion $P(x, y)$ is unknown. However, under the i.i.d assumption, the problem of minimizing R could be approached using the available training set $T_n = \{(x_1, y_1), ..., (x_n, y_n)\}$ via:

$$R_{emp}(f, T_n) = \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) \tag{2.2.3}$$

Using Equation 2.2.3, We can then represent f as:

$$\tilde{f} = \arg\min_{f \in \mathbb{F}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) \tag{2.2.4}$$

2.2.2 The Generalization Problem

Although we hope the training data T_n is a good estimation for the actual distribution $P_{(x,y)}$, we cannot guarantee a low loss on all other data points generated using $P_{(x,y)}$, because our mapping function f may fit the training samples too well. Besides, there may be many different functions that achieve the same loss using Equation 2.2.4, but their abilities of generalization on other data points may significantly vary. A popular solution to this problem is to introduce a regularization term R(f) to the learning objective:

$$\tilde{f} = \arg\min_{f \in \mathbb{F}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + \lambda R(f) \tag{2.2.5}$$

where R serves as a penalty function that prefer some functions over others in despite of their fit to the training data samples. λ is a manually-defined parameter that controls the impact of regularization penalty. If $\lambda = 0$, the optimization problem minimizes empirical risk as Equation 2.2.4. In deep learning, the most commonly used function in the space of $\mathbb{F}$ is a neural network (see Section 2.3.1 for more details) with some parameters within the internal layers. The loss function L could be a Euclidean distance based loss in regression or a cross-entropy loss in classification, and the regularization term R generally takes the $L2$ norm (i.e. sum of squares of all parameter weights).

2.2.3 Optimization

Once the choices of these functions and terms are made, the solution of a supervised learning problem is formulated as an optimization problem with a general form $\hat{\sigma} = \arg\min_{\sigma} h(\sigma)$, where σ is the collection of parameters and $h(\sigma) = \frac{1}{n} \sum_{i=1}^{n} L(f_\sigma(x_i), y_i) +$

$\lambda R(f_\sigma)$. Note that σ only belongs to the mapping function f, and is irrelevant to the loss function L and regularization term R.

A greedy approach to choose the appropriate σ is to randomly draw a huge number of σ from some distribution, and then seek the one that minimizes $h(\sigma)$. This "guess-and-check" strategy might be helpful to find the global optimal solution, however, considering the millions or billions of parameters in a typical neural network (much more if it is deeper), such kind of approach is computationally intractable.

The first-order methods are good alternatives to the greedy search. Here "first-order" means these techniques only require first-order derivatives of the mapping function f. In this method, we use the gradient as the search direction. We can adjust σ to achieve lower $h(\sigma)$ by adding a small value to it, and this value is determined by the negative direction of the gradient (since the direction of gradient indicates the direction of increase of $h(\sigma)$). This is the core idea of the popular Gradient Descent (GD) algorithm. The GD algorithm iteratively execute these two steps: 1) compute the gradient, and then 2) update parameters by a small amount towards the direction of the negative gradient. A practical concern of the GD algorithm is it requires to compute the gradient using all training samples, however, the dataset we use in practice is often extremely large (e.g. the ImageNet has more than 10 million training images), which significantly limits its application. To fight this problem, we only estimate the gradient using a mini-batch of training samples (e.g. 32, 64 or more) at a time. This allows us to perform many small updates to approximate a large-scale update, and the corresponding algorithm is named Stochastic Gradient Descent (SGD) [19]. A popular variation of the SGD algorithm which is frequently used in deep learning is shown in Algorithm 4, where the *Momentum* [20] is introduced to improve the convergence speed.

Algorithm 4 The Stochastic Gradient Descent with Momentum.

Input: learning rate $\alpha > 0$, momentum weight $\eta \in (0, 1)$, mini-batch size $s \in \mathbb{N}$

Input: a starting point $\sigma \in (R)^n$

1: **repeat**

2: Sample a mini-batch of s samples $\{(x_1, y_1, ..., (x_s, y_s))\}$.

3: Compute the gradient $\nabla_\sigma h(\sigma) = \nabla_\sigma [\frac{1}{s} \sum_{i=1}^{s} L(f(x_i), y_i) + \lambda R(f)]$.

4: Compute the update direction $\Delta\sigma := \eta\Delta\sigma^* + (1 - \eta)\nabla_\sigma h(\sigma)$, $\eta\Delta\sigma^*$ is the previous update direction.

5: Update the parameter $\sigma := \sigma - \alpha\Delta\sigma$

6: **until** Convergence.

The learning rate α is critical in SGD because an inappropriately value of α may cause

the learning algorithm to malfunction: either to converge too soon (the α is very large) or too slow (the α is very small). Another important parameter is the momentum η which controls the speed of the gradient descent in order to provide small but consistent directions of the gradient. In recent years, some methods such as RMSProp [21] and Adam [22] are proposed to neutralize the mutual impact between α and η: parameters will take smaller learning rates if they see large gradients, and parameters that see low gradients will take relatively large learning rates.

2.2.4 Cross-validation

We have seen that the learning problem formulation and optimization requires setting proper values to various parameters, for example, the regularization term λ, the learning rate α, the momentum η, etc. However, these parameters are very difficult to set considering the huge searching cost in the parametric space. In practice, we often try several possibilities according to the specific dataset, then we optimize the learning model in each case, and ultimately evaluate the predicted results on an extra *validation* set which is isolated from the training samples. This is an effective approach to estimate the generalization error. In the case that there are not too many available data samples, one can split them into some number of folds, and then use one fold for validation and the rest for training. The average validation performance could be obtained by averaging the results on all possible combinations of training and validation folds. This process is named as *k-fold validation* (e.g. 5-fold validation in case of 5 folds).

2.3 Deep Learning

Most traditional machine learning methods work well because the data representation scheme and input features are manually designed in order to best fit the training data. When model learning is applied only to the input features, the learning process is mainly focusing on parameter optimization in order to achieve the optimal prediction results. However, a obvious drawback of this approach is that it relies on extra human guidance for specific dataset and exhibit poor performance in terms of generalization. Another problem is when the input feature space is linearly inseparable, traditional methods often fail to find an hyperplane to ideally divide the positive samples and the negative samples apart.

As an emerging subfield of machine learning, deep learning can be seen as the integration of feature representation and model learning. It attempts to jointly learn good

features, across multi-scale levels of feature representation, dimensionality reduction, and the final prediction. In stead of finding a hyperplane in the input space, deep learning algorithms use some non-linear mapping function to transform the original input space to another feature space, where the feature points might by linearly separable. In this section we mainly provides the neural network basics and introduce one of the most widely used deep learning model: the Convolutional Neural Network. The introduction to other popular networks such as the Recurrent Neural Network and the Generative Adverse Network could be found in the Deep Learning book from Goodfellow et al. [13].

2.3.1 Neural Networks

The fundamental component of a deep learning model is the the artificial neural network (ANN), where the term *neural* is derived from the functional unit *neuron* in animal brains. Similar to the biological neural network, the ANN also acquires knowledge through learning, and the learning knowledge is stored and updated within the inter-neuron connections known as the parameter weights.

From the mathematic view, neural networks are constructed by some repetitions of matrix multiplications and element-wise non-linear activations. Figure 2.1 illustrates a single neuron composed of inputs (including the bias), an activation function and the output. Denoting the inputs be some n-dimensional vector $x \in \mathbb{R}$, the output can be computed as:

$$a = f(w^T x + b) \tag{2.3.1}$$

where w refer to the parameter weights, b denotes the bias unit and f represents the activation function. f is also called the non-linearity and commonly used examples are the sigmoid function $\frac{1}{1+e^{-x}}$, the hyperbolic tangent function $tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$, and the rectified linear unit (ReLU) $max(0, x)$. The sigmoid activation function maps any real number to the interval of $[0, 1]$. With this unit, the activation process can be interpreted as the probability for the "neural unit" to be "on". The choice of nonlinearity often depends on the specific dataset and should be tested by cross-validation.

A neural network could be constructed by stacking single neurons horizontally and vertically. Vertically stacked neurons from layers in the network, and horizontally stacked layers show the internal structure of the network. The layers between the input and layer and the output layer are called hidden layers. In Figure 2.2, a 2-layer network could be implemented as $f(x) = W_2\varphi(W_1 x)$, where W_1, W_2 are weight matrices and φ

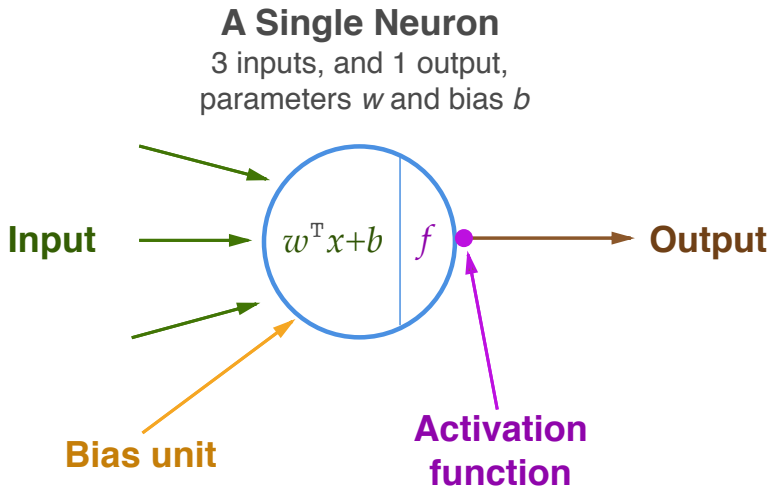


Figure 2.1: The definition of a single neuron with inputs, bias, activation function and the output.

is the element-wise non-linearity (e.g. sigmoid). Similarly, A 3-layer network would have the form $f(x) = W_3\varphi(W_2\varphi(W_1x))$, etc. Generally the last layer of the neural network does not contain the non-linearity, and a 1-layer neural network is just a linear transformation.

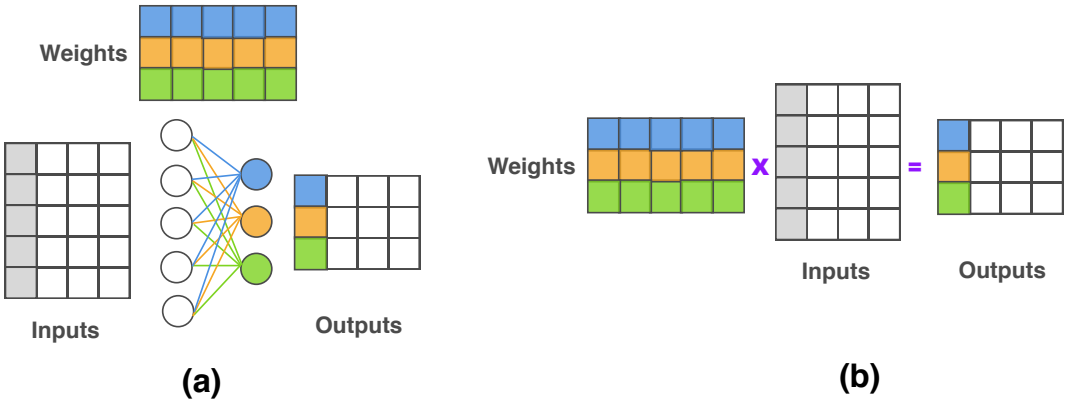


Figure 2.2: (a): An example arrangement of neurons in a 2-layer neural network. Neurons in one layer have connections to all neurons in the previous layer but are not connected to each other. The input layer was represented using a 6×4 matrix where 6 is the number of input neurons and 4 is the number of input samples. The output is a 3×4 matrix where 3 is the number of output neurons. **(b):** The matrix multiplication operation to compute the output.

2.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs / ConvNets) [23] are a category of neural networks that are have shown excellent performance in various tasks such as image/video

classification, speech recognition, text sequence prediction, etc. Similar to ordinary neural networks, CNNs are composed of neurons containing learnable parameter weights and biases. In the task of image classification, each layer in the designed CNN accepts some inputs, performs the matrix multiplication and optionally follows it with an activation function. The whole network still provide a scoring function to map the raw image pixels to the class scores in the output layer where a loss function is used.

The main difference between the CNN and the ordinary neural network is the inner structure of their hidden layers. We know that in the ordinary neural network, each hidden layer is made up neurons, where each of them is fully connected to all neurons in the previous layer, and neurons in the same layer do not have any connections. In other words, each neuron only stores its own parameter weight, and do not share it with all the others. A problem of this architecture is it do not scale well on large-scale input data. Assuming that we have an image with size of $48 \times 48 \times 3$ (48 width, 48 height, with 3 color channels), then a single fully-connect neuron in the first hidden layer of a ordinary neural network would have $48 \times 48 \times 3 = 6912$ weights. This amount seems not quite large, however, when the size of image becomes larger (e.g. $224 \times 224 \times 3$), the number of weights would be $224 \times 224 \times 3 = 150,528$. To construct a neural network, we often need some number of such neurons, and the amount of parameters could increase dramatically. It is obvious that the full connectivity makes the computation intractable and huge amount of parameters often lead to the overfitting problem.

As a alternative network structure, the convolutional neural network handles inputs in a square of neurons. For instance, an color image (3 channels) with the size of $48 \times 48 \times 3$ is stored as a multi-dimensional array (e.g. a *tensor*) in computers, and would be represented using 3 squares (48×48) of neurons as the input layer of a CNN. Besides, the neurons in the first hidden layer will only be connected to a small region of the input pixels (e.g. 3×3 region, corresponding to 9 input pixels), instead of all of the pixels in a fully-connected manner. Such a small region is called the *local receptive field* [13] for the hidden neuron. And each hidden neuron learns a overall bias for that region. This local-connectivity significantly reduce the dimensionality and the computation cost.

The Convolutional Layer

The *Convolutional layer* is the core building block of a CNN that slides the local receptive field (i.e. the convolutional operation) across the input tensor using a set of filters and then produces the output tensor. For example, assuming that the input tensor of an image is a $28 \times 28 \times 3$ tensor, if the size of local receptive field (or the filter size) is $5 \times 5 \times 3$, then each neuron in the following convolutional layer will have $5 \times 5 \times 3 = 75$

weights. We then "convolve" this filter by sliding it across the spatial positions of the input tensor and for each position, we compute the dot product between each small region of the input tensor and the filter. If we move the filter by one pixel at a time, the result will be a feature map represented by 24×24 neurons in the hidden layer (see Figure 2.3). Moreover, if we use a larger stride length, then we will get fewer number of neurons in the hidden layers. Sometimes it is useful to pad the input pixels with zeros around the border (i.e. zero-padding) to help limiting the output dimension. Since the convolutional layer often contains a set of filters, the final output tensor could be obtained by stacking all the outputs of the convolutional operation using each filter.

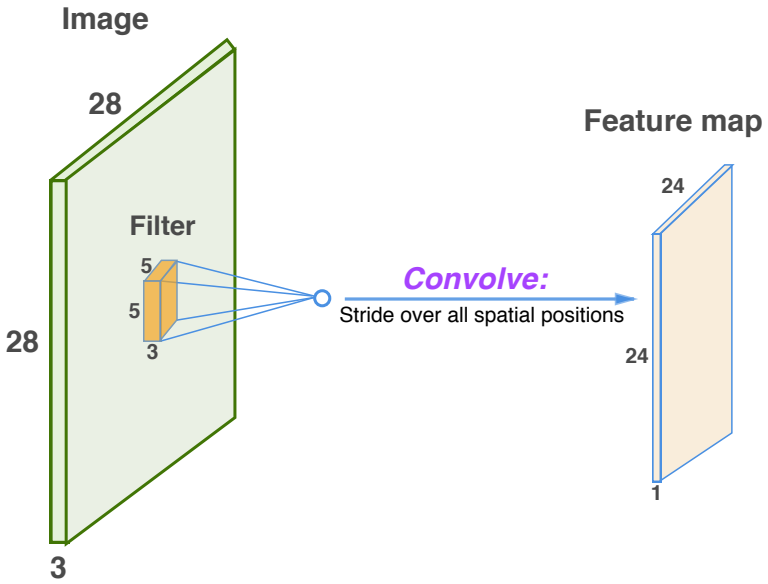


Figure 2.3: An illustration of convolving a $5 \times 5 \times 3$ filter over a $28 \times 28 \times 3$ input image with stride 1 and pad 0. The filers span a small area, but take the same depth as the input tensor (i.e. 3). In each depth, there are 24 unique spatial positions for a 5×5 filter in a 28×28 input tensor, so the convolutional operation produces a 24×24 feature map. A convolutional layer possess a set of different filters, each one is applied on the input tensor independently, generating different feature maps. The feature maps are stacked one-by-one to form the final output tensor.

To put it all together, with the input tensor size T, the filter size F, the stride length S, the zero padding size P on the border and the number of filters N, we can compute the dimension of the output tensor using a formula: $N((W - F + 2P)/S + 1)$. For instance, for a 7×7 input tensor, 30 filters with size of 3×3, stride 1 and pad 0, we would get a $5 \times 5 \times 30$ output tensor.

The Pooling Layer

The convolutional layers are often followed by pooling layers to further reduce the di-

mension of feature maps in order to alleviate the overfitting problem. Pooling (also called subsampling or downsampling) spatially simplify the information in each feature map but reserves the most important information. For example, in case of *max-pooling*, we could use a 2×2 filter with a stride of 2 to take the largest value of every spatial locations (2×2 or 4 numbers) in the feature map (see Figure 2.4). The result is an output tensor downscaled by a factor of 2 both vertically and horizontally. We then repeatedly apply the max-pooling on each feature map, and stack the results of them as the final output tensor. Instead of taking the max value in each filter, we could also take the average (*average-pooling*) or the sum of all values in that filter. Anyway, in practice, max-pooling is often shown to work better.

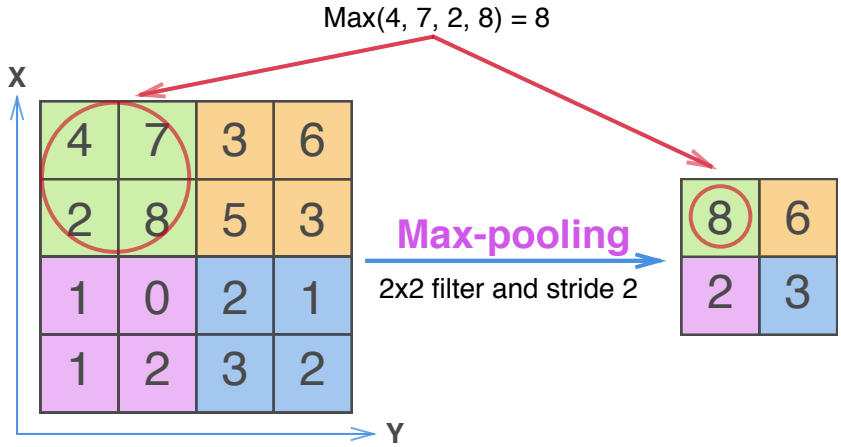


Figure 2.4: An illustration of max-pooling a 2×2 filter over a 4×4 feature map with stride 2. We slide the 2×2 filter by 2 and take the maximum value in each spatial location. This reduces the dimensionality of the feature map. The pooling results on each feature map are then stacked together to construct the final output.

The Architecture of the CNN

Generally a convolutional neural network is constructed by stacking a sequences of layers such as the convolutional layer (CONV), the pooling layer (POOL) and the fully-connected layer (FC). A simple CNN that process color images might have the structure $[INPUT, (CONV, CONV, CONV, POOL) \times 2, FC, FC]$. Here the $INPUT$ denotes a mini-batch of input image tensors (e.g. $[64 \times 28 \times 28 \times 3]$ for a mini-batch of 64 color images with the size of 28×28). The CONV computes the output of neurons that are connected to small regions in the input image pixels. The output would be a $24 \times 24 \times 30$ tensor if we use the 30 filters with size of 5×5, stride 1 and pad 0. The POOL summarizes the feature maps along the spatial dimensions and produce an output tensor with reduced dimension. The FC undertakes the same job as they do in ordinary neural networks: fully-connect the neurons in the previous layers and the

output layer. The last FC could be a $1 \times 1 \times 10$ tensor holding 10 class scores if the CNN is designed for the classification task. In such a manner, CNNs transform the original image pixels layer-by-layer to the final class scores. The parameters within the network could be trained using some optimizer function (e.g. SGD) and the backpropagation algorithm.

2.4 Natural Language Processing

Natural Language Processing (NLP) refers to the ability of a computer program to understand and manipulate human language as it is written or spoken. NLP has been studied for decades and now is seen as a critical component of the artificial intelligence (AI). Using different NLP techniques, researchers and developers can perform various tasks such as the automatic document summarization, machine translation (e.g. Google translation), named entity recognition (e.g. name or place recognition), question answering, sentiment analysis (e.g. online product review analysis), speech recognition (e.g. Apple Siri and Microsoft Cortana), topic classification, etc.

NLP is not easy. As great language users, human can communicate with each other freely and do not have to formally understand and describe the inherent rules that govern languages. However, human language is highly ambiguous, and it is also changing and evolving at all times [24]. Hence, to accurately understanding human language, computers need to learn the usage of these rules, this is what NLP algorithms is aiming to do.

Thanks to recent advances in the realm of machine learning (especially deep learning) techniques, NLP is able to automatically learn the intrinsic rules by analyzing a large set of examples (e.g books, news reports, human speeches, etc.), and then make inferences based on statistical theories. In general, increasing the scale of training data can more or less improve the accuracy of the model.

NLP tasks can be divided into many sub-groups, and in each group, a set of techniques are proposed to tackle specific problem. In this section, we mainly introduce two key NLP methods we used to aid our work of visual content recognition: the part-of-speech tagging and the word vector representation. To know more about NLP techniques, we recommend the book from CD Manning et al. [14].

2.4.1 Part-of-speech Tagging

Part-of-speech (POS) measures how a word functions in a sentence, and the process of assigning a POS to a word in a sentence is called POS tagging. The tag sets of POS includes nouns (NN), verbs (VB), adverbs (ADV), adjectives (ADJ), pronouns (PRON), prepositions (PREP), conjunctions (CONJ) and their fine-grained sub-categories. POS tagging is a critical problem in many NLP tasks due to the ambiguity of human language in different context. For instance, the word *chair* is noun (NN) in the sentence *"He is sitting on a brown chair"*, while it functions as a verb (VB) in the sentence *"She chairs the faculty for a few years"*. Besides, POS tagging is also useful for finding named entities like people, places and organizations in the text or speeches.

An example of POS tagging and sentence analysis (parsing) is shown in Figure 2.5. We use the Stanford CoreNLP toolkit [25] to analyze a sentence *A small dog is lying with a blue ball of yarn on the brown floor*. Firstly we can get each word's POS tag, and then we start to look into the relationships between different POS tags (e.g. noun phrases, verb phrases, etc.) in the sentence.

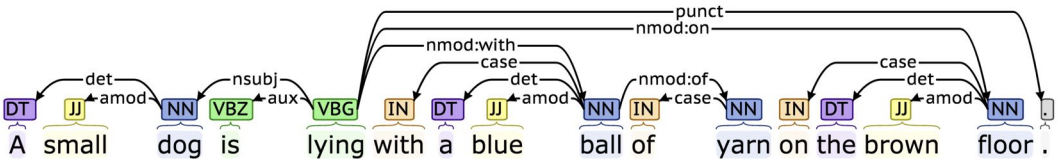


Figure 2.5: An illustration of POS tagging and sentence parsing for the sentence *A small dog is lying with a blue ball of yarn on the brown floor*. The capital letters in colorful boxes are the POS tags of the words in the sentence, and the abbreviations in the middle of bi-directional arrows indicate the relationships between different POS tags.

To give an intuitive impression how POS tagging could benefit computer vision (CV) tasks, we provide another example in Figure 2.6. We can point out the "dog" in the image with just a glance. This is also what early CV algorithms do: passively output a class label by analyzing the input image. Then we may notice more details like the "brown dog" (noun phrase), "blue ball of yarn" (noun phrase) and "dog holding a ball" (verb phrase), or we can simply describe the whole scene using "A small dog is lying with a blue ball." Note that we do this naturally without explicitly specifying the functionality (POS tags) of these words or pointing out the regions (bounding boxes) which correspond to these words in the image. However, computers need to do all of these things in order to make accurate detection and prediction. For a complete example of using the POS tagging, we refer to our work on automatic video summarization which is described in **Chapter 4**.

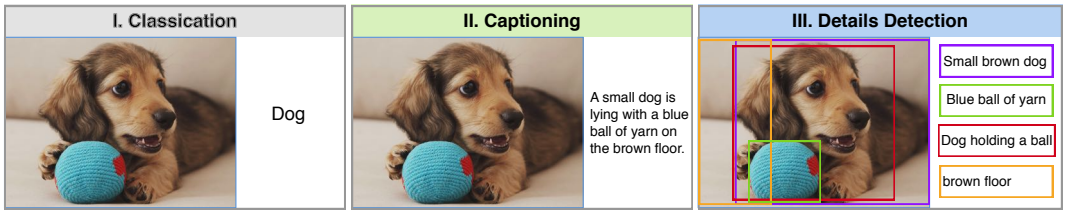


Figure 2.6: The pyramid of semantic scene understanding. **Left:** Mapping pixels to a semantic class label. **Middle:** Image captioning that captures more semantic meanings in the visual content. **Right:** Jointly detect and describe the relationships between semantic objects in the visual scene.

2.4.2 Word Vector Representation

From last section we know that computer vision algorithms can learn the meaning of words via POS tagging, however, they also need some way to represent words in order to learn these semantic meanings in the context when processing visual contents.

One-hot Representation

Many statistical NLP algorithms treat a single word as an atomic unit, and each unique word in the whole corpus (e.g. a book or a novel) is used to build the vocabulary. By this means, the word vector is represented as a vector of weights where "1" indicates the index location of current word in the vocabulary, and all other elements in the vector are zero. A demo is illustrated in Figure 2.7, where the size of vocabulary consists of five words: (*Python*, *Ruby*, *Caffe*, *Tensorflow*, *network*). Then the vector of *caffe* is $[0, 0, 1, 0, 0]$. This so-called "one-hot" representation has the problem that it does not describe any type of similarity between each pair of words. Unfortunately, there are more than 10 million words in English language, and many of these are related, for instance, *spouse* to *parter* and *hotel* to *motel*. Besides, the one-hot representation is impractical when the vocabulary is extremely large (e.g. a Google News document collection contains more than 3 million unique words.).

Distributed Representation

The distributed representation (a.k.a word embedding) represents a word as a fixed-dimensional and real-valued vector. Typically the dimension is low, like 100, 300 or 1000. For instance, the words (*python*, *ruby*, *caffe*, *tensorflow*, *network*) might be expressed as follows:

From Figure 2.8 we can observe that the word *Caffe* is similar to *Tensorflow*, and *Python* is similar to *Ruby*. This is meaningful to us because both Python and Ruby are programming languages, while Caffe and Tensorflow are deep learning frameworks. Hence, these dimensions in word vectors are believed to capture the semantic properties of

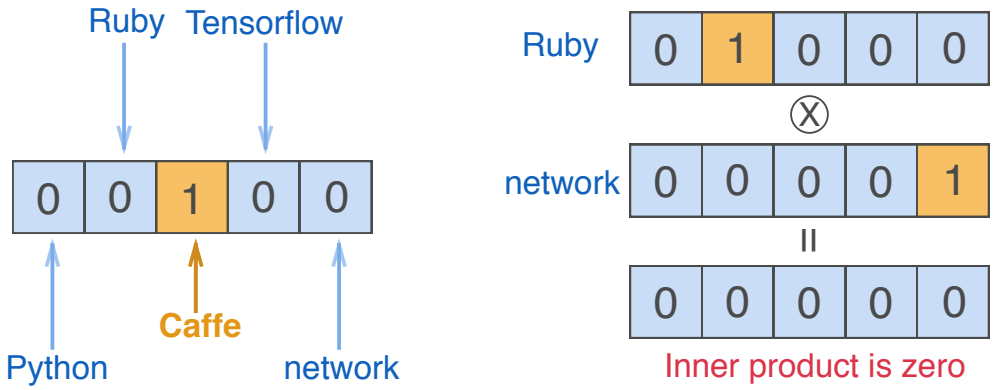


Figure 2.7: Left: The example of one-hot representation of the vocabulary (*Python*, *Ruby*, *Caffe*, *Tensorflow*, *network*). **Right:** The element-wise product is a zero vector, which means the relationship between different words cannot be measured by arithmetic operations.

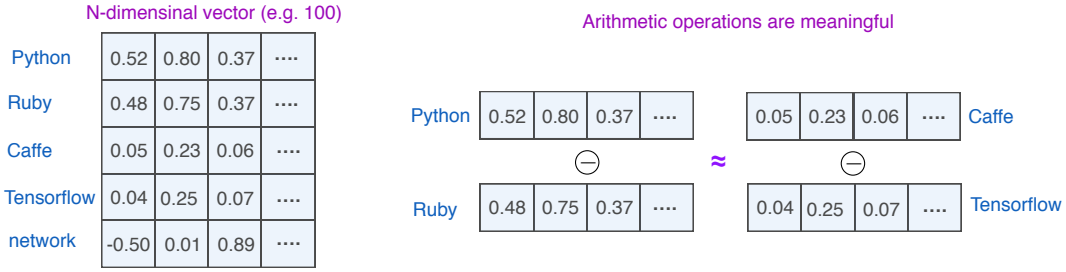


Figure 2.8: Left: Distributed representation of words (*Python*, *Ruby*, *Caffe*, *Tensorflow*, *network*). **Right:** The similarity of different words could be measured by arithmetic operations.

the words. Note that the values of these vectors in the example are chosen arbitrarily and do not describe an actual representation. The actual distributed word vectors can be obtained by using a large corpus (e.g. Google News, Wikipedia, etc.) to train a neural network with special architectures: the Skip-Gram model and the Continuous Bag-of-Words (CBOW) model [26].

In the Skip-gram model, we take a word as the central point and try to predict some of its neighboring words (typically within a window size). More specifically, the model is aiming to define a probability distribution representing the existence of some words as neighbors of the central word. Hence, the training objective is to adjust the values in the word vectors so as to maximize the corresponding probability. All values in the word vectors are randomly initialized before training. Conversely, in CBOW model, we try to predict the central word by evaluating the vectors of its neighbor words. For detailed introduction of these two models and their training procedure, we refer to the paper of Tomas Mikolov, et al. [26] and the Word2Vec toolkit [27]. Our work of using distributed word representation to improve the tag-based image annotation is

summarized in Chapter 3.

2.5 Applications

In this section, we will briefly introduce the applications of deep learning. Though deep learning are widely used in various research field such as computer vision, natural language processing and signal processing, we would like to focus on image and language related tasks.

The ultimate goal of computer vision is to sense the visual world and interact with us using our own languages. Most deep learning techniques for computer vision are used to object-oriented tasks, from image classification [28–31] which maps pixels to a semantic class label, to image captioning [32–34] which describes the image using natural language sentences and object detection [35–37] which localize and label various objects in a visual scene. Other applications include image segmentation [38, 39], which means labeling image pixels in order to localize specific objects and identify their shapes; style transfer [40–42], which means re-arranging image pixels in order to stylize it according to the style of other images; image synthesis [43, 44], which means generating new images that do not exist before.

CHAPTER 3

Visual Themes: More Comprehensive Annotation for Images

In this chapter we develop techniques to discover Visual Theme (VT) as a better (more compact, efficient and effective) alternative for tag-based image annotations. Given a joint image and tag corpora, We start by examining each tag's ability for visual content description, then eliminate tags whose descriptive abilities are relatively low. Next we measure the pairwise semantic and visual similarity amongst the remaining tags, then merge them into a joint similarity matrix. Visual similarity measures how tags are visually connected to describe similar visual contents, and semantic similarity measures how close they are in natural language understanding. Finally we cluster the tags into a collection of VTs according to the joint similarity matrix. We conduct user study and computer vision tasks to demonstrate the effectiveness and versatility of the discovered VTs. The workflow of the proposed framework is illustrated in Figure 3.1.

3.1 Related Work

Our definition of visual theme is partly inspired by the naming of the visual concept [45]. A visual concept is denoted as a subset of human language vocabularies that refer to particular visual entities (e.g. fireman, policeman). Visual concepts have long been collected and used by computer vision researchers in multiple domains [46–49]. An example in image analysis is the ImageNet [2], where visual concepts (only nouns) are selected and organized hierarchically on the basis of the WordNet [50]. A drawback of the visual concepts is, they are often manually defined, and sometimes they may fail to

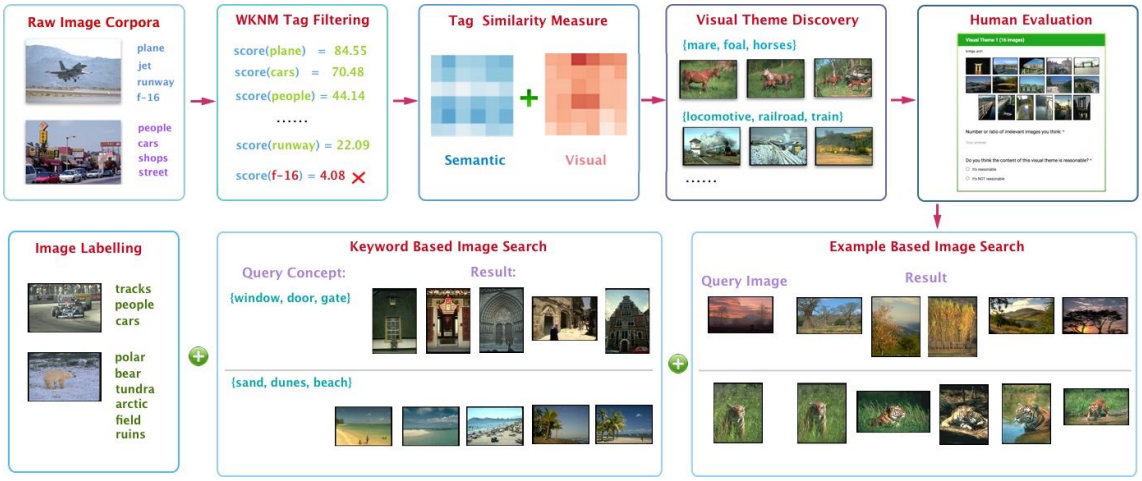


Figure 3.1: Overview of the visual theme discovery framework and its applications. Given images and associated tags, we first eliminate less qualified tags using the WNKM tag filtering method, then cluster tags into visual themes according to their semantic and visual similarities. Next we ask human evaluators to evaluate the quality of the discovered visual themes. The applications of the visual themes are shown in the bottom row.

capture complex information within the visual world. This makes them less applicable in multiple domains.

As discussed before, the subjectivity of visual concept definition hinders its extension to be used on different joint image and text databases. This motivates us to explore objective visual theme directly from raw images and associated tags. Our work on visual theme discovery is related to previous work on concept discovery [45, 51, 52]. In particular, LEVAN [52] starts with a group of general concepts and gradually divide them into sub-concepts according to massive resources of online books. VisKE [51] focuses on validating relationships between pairs of concept entities from the semantic and visual aspects. [45] builds a large amount of classifiers for terms filtering and similarity computation, then cluster selected terms into concepts.

A significant difference between our work and previous work is that we are not trying to build large amount of general visual concepts so as to describe as many images as possible, instead, we put forward an unsupervised and efficient framework to allow different image databases to have their own collection of visual themes for visual content description. Considering the quantity and diversity of images, dividing large image collections into visual theme based categories can facilitate various tasks such as image management, indexing and retrieval.

3.2 Methodology

This section elaborates the workflow of the visual theme discovery. Recall that a visual theme is constructed by a subset of tags which are capable of representing similar visual contents. To make it practical, we argue that a VT should show strong connection to certain visual contents that can be easily processed by computer vision algorithms. Besides, tags (including synonyms) describing same or similar visual content should be grouped into the same visual theme in order to maintain compactness. Starting from an image corpus and associated tags, we first pick tags which show high-level visual content descriptive ability, then cluster them into a set of VTs based on visual and semantic similarity.

3.2.1 Tag Filtering

As we mentioned before, not all tags show strong connection with their associated visual contents. Before discovering the visual themes, we need to examine each tag's ability of visual content description, and filter out ones who are not qualified. The idea to achieve this is simple: if a tag is good at depicting particular visual content, the majority of its associated images should also share similar visual contents. Hence, the visual similarities between images under a tag can reflect the tag's ability towards visual content description. For implementation, we represent images as feature activations extracted from a pre-trained convolution neural network (CNN) model due to its excellent performance in image classification tasks [53]. We then define the Weighted K-Nearest Measure (WKNM) as the measurement of tags' ability towards visual content description.

The procedure of WKNM is illustrated in Figure 3.2. Given tag t_i and its associated image set $F_i = set\left\{f_{i1}, f_{i2}, ..., f_{ij}, ..., f_{in}\right\}$, for every related image f_{ij}, we find its K nearest neighbors based on the cosine distance of their visual features. Thus, the similarity score between image f_{ij} and other images in F_i could be computed as:

$$Sim(f_{ij}, F_i) = \sum_{k=1}^{K}(1 - \frac{k-1}{K})\delta(t_i, f_{ijk}) \tag{3.2.1}$$

where $\delta(t_i, f_{ijk})$ is an indicator function which equals to 1 if image f_{ijk} contains tag t_i, otherwise it is set to 0. K is the number of nearest neighbors of image f_{ij}. It could be noticed that, $\delta(t_i, f_{ijk})$ is penalized by multiplying a weight according to the sequence in K neighbors (a closer neighbor has a smaller sequence index). Hence, $Sim(f_{ij}, F_i)$ quantifies tag t_i's ability towards visual content description based on image f_{ij}.

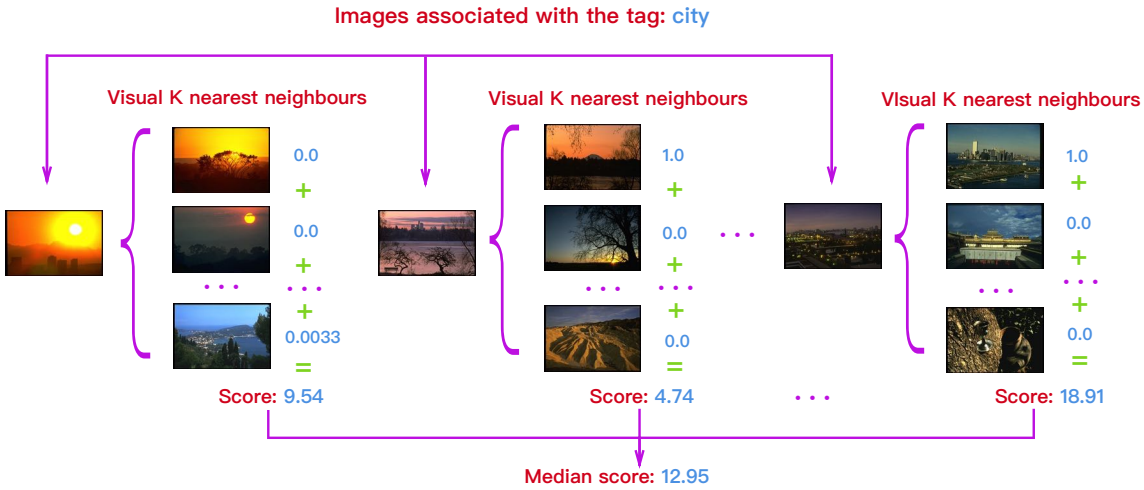


Figure 3.2: The workflow of Weighted K-Nearest Measure. Given a tag and its associated images, for each image, we find its visual K-nearest neighbors and examine if other images under the same tag frequently appear in its K neighbors. We compute a score (higher is better) of each associated image of the given tag, then take the median to quantify the tag's ability towards visual content description.

We successively compute all similarity scores based on each image in tag t_i's associated image set F_i, then take the median score to quantify tag t_i's ability of visual content description. We call such a median the Visual Content Descriptive Level (VCDL). A large VCDL of a tag indicates it is good at describing certain visual contents. We choose the median because it is a robust statistic, even if dataset is biased, the median is unlikely to offer an arbitrarily large or small value. We design the WKNM method to perform tag filtering because traditional method like tf-idf cannot offer much help on this task. More specifically, in each image's annotation, the tf-idf method only uses 1 to represent the existence of a certain tag, and 0 if it does not exist. So the tf in the tf-idf method would always be 1 or 0. Besides, some tags like *people*, *car* often appear in many image annotations, and that will make its idf scores extremely large. Consequently, those important tags will be allocated with small tf-idf scores, and would be potentially removed by thresholding.

Based on WKNM, we compute VCDLs for all tags and eliminate those whose scores fall below a certain threshold. The threshold is often empirically set since the distribution of tag data is often biased and unbalanced. Note that we do not need to examine each tag's frequency of occurrence since the WKNM method has inherently done this.

Table 3.1 gives a few examples of filtered tags from the Corel5K [54] dataset. Here we use "too specific", "too abstract" and "too generic" to subjectively describe some filtered tags. Note that The tag filtering algorithm is not aware of what is "too specific",

Table 3.1: Examples of filtered tags on Corel5K dataset.

Filtered tags	Evaluation
{f-16, kauai, oahu}	too specific
{whited-tailed, close-up}	too abstract
{art, festival}	too generic

"too abstract" or "too generic", and these terms are manually defined to show that the tag filtering algorithm is able to eliminate inappropriate tags (consistent with human perception) for building the visual themes. However, when we take a look at the remaining tags, we found some of them are synonyms e.g. *jet* and *plane*. It is necessary to group them together since they are likely to confuse computer vision (CV) algorithms and introduce extra computational cost. Moreover, we notice some tags are often used together to describe particular visual content. For instance, in the Corel5K dataset, *grizzly* only appears together with *bears* in images containing bears. This motivates us to measure tag similarity both semantically and visually.

3.2.2 Tag Visual Similarity Measure

We measure tag visual similarity by examining their distances in the visual space. In this space, each tag could be represented by its associated images which takes up a small region in the visual space. We use holistic visual features to represent images in the visual space. These visual features are extracted from a convolutional neural network named VGG-16 [55] trained on ImageNet [2] (a large dataset for image classification). VGG-16 is able to produce good image features and is widely used in many computer vision tasks in recent years. In the VGG-16 model, we take the output of fully-connected layer 'fc7' (4,096 dimensions) as the image-level holistic visual features. Hence, the well-known Hausdorff distance (HD) is quite appropriate to measure visual distance between two different tags.

The Hausdorff distance is defined as the maximum distance of a set to the nearest point in the other set [56]. In our case, the Hausdorff distance from tag A to tag B in the visual space would be:

$$h(A,B) = \max_{a \epsilon A}\{\min_{b \epsilon B}\{dist(a,b)\}\} \tag{3.2.2}$$

where a and b are image feature points of tags A and B in the high-dimensional visual space, $dist(a,b)$ denotes some distance measure between these points. For simplicity, we take $dist(a,b)$ as the Euclidean distance between a and b.

Since HD measures the relative position of points in visual space, it's more robust to position variations than other methods. However, HD method is quite sensitive to outliers, which makes it inappropriate to tackle noisy data. A modified version of HD is proposed in [57]:

$$h_{mod}(A,B) = \frac{1}{|A|}\sum_{a\epsilon A}\min_{b\epsilon B}\{dist(a,b)\} \tag{3.2.3}$$

where $|A|$ is the number of images associated with tag A. A problem of this modified version of HD is that it contains points whose pairwise distances are zero. Considering that an image's annotation often contains more than 1 tag, points a and b in $dist(a,b)$ could refer to the same image. Hence, we revise the formula in (3.2.3) to remove this negative impact:

$$h^{'}_{mod}(A,B) = \frac{1}{|A'|}\sum_{a\epsilon A}\min_{b\epsilon B}\{dist(a,b)\} \tag{3.2.4}$$

where $A^{'} = \min_{b\epsilon B}\{dist(a,b) \neq 0\}$. We use (3.2.4) to measure visual distance between tag A and tag B. Since associated images of different tags also differ, we modify the final Hausdorff distance between two tags as:

$$F^{'}(A,B) = \max\{h^{'}_{mod}(A,B), h^{'}_{mod}(B,A)\} \tag{3.2.5}$$

Ultimately we can obtain a distance matrix M_{vdist} where each entry is the visual distance between a pair of tags. It's easy to switch distance to similarity: just rescale all values in M_{vdist} to the range from 0 to 1, then replace each entry value with the difference between 1 and original value. We denoted the tag visual similarity matrix as M_{vsim}. Larger values in M_{vsim} indicates stronger visual similarity between two corresponding tags.

3.2.3 Tag Semantic Similarity Measure

We measure semantic similarity between two tags by evaluating their word embeddings [26] [58] in an unsupervised manner. In the embedding space, each distinct word is represented as an N-dimensional vector. The embedding algorithm first assigns each word vector with random values, then recursively adjust the value of these vectors according to some objective function. More specifically, we train a Skip-gram neural network language model [26] on the latest dump of English Wikipedia using the Word2Vec [27] toolset.

To elaborate, the training set is a large collection of English Wikipedia articles, and each articles contains lots of sentences. Since the length of the input of the word-embedding algorithm is fixed, sentences with different lengths cannot be directly passed to the algorithm. Hence, in the training phase, each time a short sequence of words are extracted from an article using a sliding window with fixed width (similar as 1-D convolution). Then the corresponding word vectors (random values at first) are extracted and fed into the skip-gram model. The training objective is to enable a word to effectively predict its nearby words, so words enjoy higher semantic similarity lie closer in the semantic space.

Once the training process is completed, we extract word vectors from the trained model according to the content of tags, then evaluate the semantic similarity of each pair of tags by computing cosine distance between their corresponding word vectors. Similarly, we build the semantic similarity matrix M_{ssim}. Again, we replace each entry value in M_{ssim} with the difference between 1 and original value. Larger values in M_{ssim} indicates stronger semantic similarity between two corresponding tags.

3.2.4 Clustering Tags into Visual Themes

With two similarity matrices M_{vsim} and M_{ssim}, we linearly merge them into a joint similarity matrix M_{join} via a parameter α (from 0 to 1). We can control the proportion of visual and semantic components by tuning α.

$$M_{join} = \alpha \times M_{vsim} + (1 - \alpha) \times M_{ssim} \tag{3.2.6}$$

Based on M_{join}, we use the spectral clustering [59] to cluster tags into a collection of visual themes. Table 3.2 describes a few themes discovered on the Corel5K dataset with α fixed to 0.12. We have done some experiments on alpha values ranges from 0.0 to 1.0 with an interval of 0.02. We found that a large value of alpha would group tags with similar semantic meanings into different visual themes, which makes results not quite reasonable. This is due to the way of computation of tag visual similarity. The tag visual similarity evaluates the similarity between two tags in the visual space. More specifically, if these two tags are often used to describe visually similar contents, they are then close to each other in the visual space and vice versa. However, although the image features are extracted from a large-scale image dataset, the number of tags participate in the computation is relatively small (260, 291 and 268 in three datasets) in this project. Hence, the visual similarity M_{vsim} computed on a relatively small image and tag dataset may introduce bias to the clustering algorithm. Consequently, tags

with similar semantic meanings are likely to be grouped into different visual themes because they are used to describe visually different images. While in the semantic space, tags are represented using word vectors and these vectors were trained on a very large natural language corpus (Wikipedia, over 6 billion word tokens). Hence, a large portion of tag semantic similarity Mssim (i.e. setting α to a small value) is able to balance the bias brought by visual similarity M_{vsim} in the joint similarity matrix M_{join}.

In this regard, we empirically set alpha to a small value of no more than 0.3 which we found worked well. This is not ideal and a systematic way to set this parameter would be better. However, as in many real applications, this is impossible. Fortunately, we have data and it is possible to use cross validation method to empirically choose a reasonable value.

Table 3.2: Examples of visual themes discovered on the Corel5K dataset.

Concept Type	Concept Content
scene	{sunrise, sunset}
object	{mare, foals, horses}
mixed	{cloud, sky, mist, horizon}
mixed	{jet, flight, runway, plane}

3.3 Human Evaluation

After the clustering phase, each visual theme is represented as a set of tags with associated images. As we mentioned in Section 3, a visual theme should show strong connection to certain visual contents. Besides, tags (including synonyms) describing the same or similar visual content should be grouped into the same visual theme. Hence, we design a human based test to evaluate the quality of discovered visual themes. As visual themes are subjective, human evaluation is probably the only way to assess the soundness of them.

We work on the Corel5K dataset and discover 100 visual themes from 4500 training images and associated tags. We feed training images into the VGG-16 [55] model and take the output of fully-connected layer 'fc7' (4,096 dimensions) as the image-level holistic visual features. Then we choose 499 testing images as the evaluation set, and replace tag based annotation with corresponding visual themes. Hence, testing images are categorized into a collection of visual themes. Next we remove themes whose frequencies of occurrence are less than 3 times across all testing images, and keep 66 visual themes

for evaluation.

We design a two-step procedure for human evaluation. An example of the evaluation interface is shown in Figure 3.3. For each visual theme, we first display its tags and associated images to human evaluators, then asked them to examine whether the visual content described by this visual theme appears in all the associated images. If not, they need to give the number of images which they think are irrelevant to the given theme. Thus we can easily compute the ratio of relevant images for each visual theme, and we name such a ratio as the accuracy of visual content description (AVCD) of a visual theme. The AVCD for each visual theme is obtained by averaging all evaluators' responses on that theme.

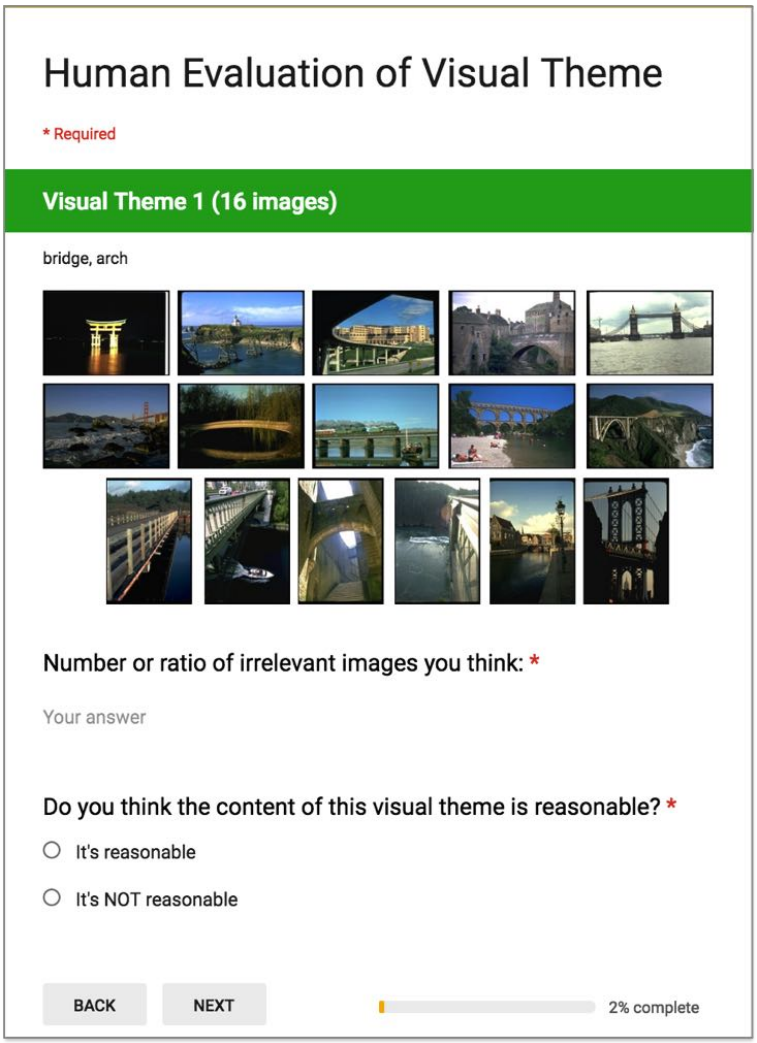


Figure 3.3: An example of the human evaluation interface. Human evaluators need to give the numbers of images which are irrelevant to current displayed visual theme, and also vote for the rationality of this visual theme.

In the following step we asked evaluators to examine tags contained in visual themes.

They need to check if all tags within a visual theme are semantically connected and refer to similar visual content. If so, the corresponding visual theme is regarded as rational and vice versa. The final decision of rationality for each visual theme was obtained by taking the majority of votes of the human evaluators.

17 human subjects participated in the evaluation experiment. Ten of them are students studying in the University of Nottingham or the Shenzhen University, and all of them have some knowledge of image annotation. The other seven participants are industrial practitioners and all of them were properly trained with basic concepts of image annotation. The evaluation result is summarized in Figure 3.4. In (a) we can clearly see that more than half of discovered visual themes achieve an AVCD over 0.9 on the visual content description, and only 4% of them did not perform well on this task. In terms of rationality, 92% of visual themes are voted as rational while the remaining 8% are not. The experiment result demonstrates the effectiveness of the discovered visual themes towards visual content description.

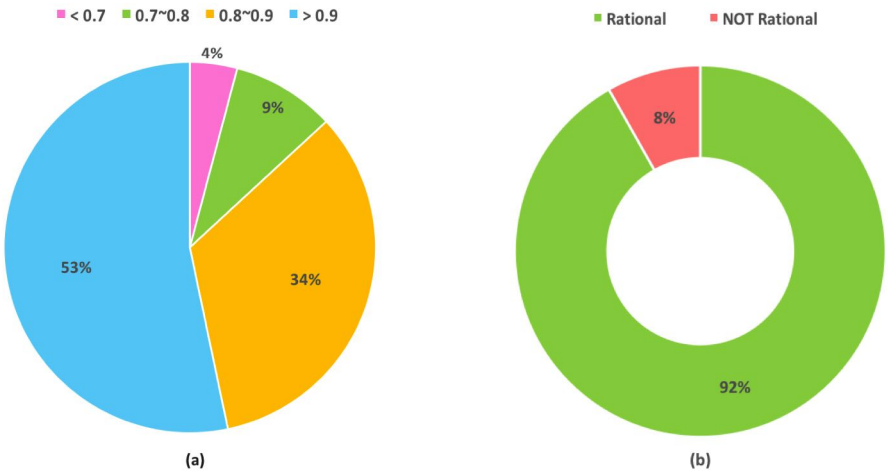


Figure 3.4: Results of the human evaluation of the discovered visual themes on the Corel5K dataset. (a): The result on the accuracy of visual content description of visual themes. (b): The evaluators' responses on rationality of visual themes.

3.4 Machine Evaluation

Besides human evaluation, we also evaluate the discovered VTs via three common CV tasks: example based image search, keyword based image search and image labeling. We work on four popular benchmarks, namely, Corel5K [54], NUS-Wide-Lite [60], IAPR-TC12 [61] and a subset of ESP-game [62]. The first two are used for example

based search and keyword based search respectively. The last two and Corel5K are chosen as the testbeds of image labeling.

To conduct the image retrieval and labeling experiments, we need to construct an appropriate framework. To achieve this, a vital issue needs to be taken into consideration: how to design an effective data structure in terms of storage and speedy retrieval. Inspired by [63], we build a random forest using image features and the discovered visual themes. The image features are holistic visual features extracted from a convolutional neural network named VGG-16 [55] model. Denoting the image feature vector as $f^{'}$,the split function is defined as:

$$\begin{cases} f_i^{'} \geq thres, \text{ go to the left child} \\ f_i^{'} < thres, \text{ go to the right child} \end{cases} \tag{3.4.1}$$

In the traditional random forest, we can generate multiple splits with different feature dimensions and thresholds. In this project, we plan to use the distribution of visual themes to guide the generation of trees. More specifically, in each random tree, we do binary split on visual features. After splitting image samples to the left child node or right child node, we compute the histogram of their corresponding visual themes. A good split means the visual theme histogram at the left child node should be quite different with from the visual theme histogram at the right child node. In other words, the visual theme histograms in the right or left child node can be viewed as the probabilities that a child node contains the specific visual theme. Hence, the well-known information gain [64] could be used as the objective function. The number of trees can vary and are empirically set based on the specific dataset. We do not explicitly set the maximum depth of trees and allow each single random tree to grow until no new split is made.

The architecture of our random forest is illustrated in Figure 3.5. Given a test image, we feed its visual feature into a random tree, and it keeps falling until it reaches a leaf node. Consequently, training examples under the same leaf node share similar or same visual themes with the test image. Here we name a related training example as a Hybrid Neighbor (HN). We successively feed the test image to all random trees and obtain the Hybrid Neighbor Set (HNS) which is formed by all HNs. Additionally, the frequency of occurrence for a single HN in HNS is defined as Hybrid Neighbor Vote (HNV). Apparently, a larger HNV indicates stronger similarity between a training image and a testing image, and vice versa.

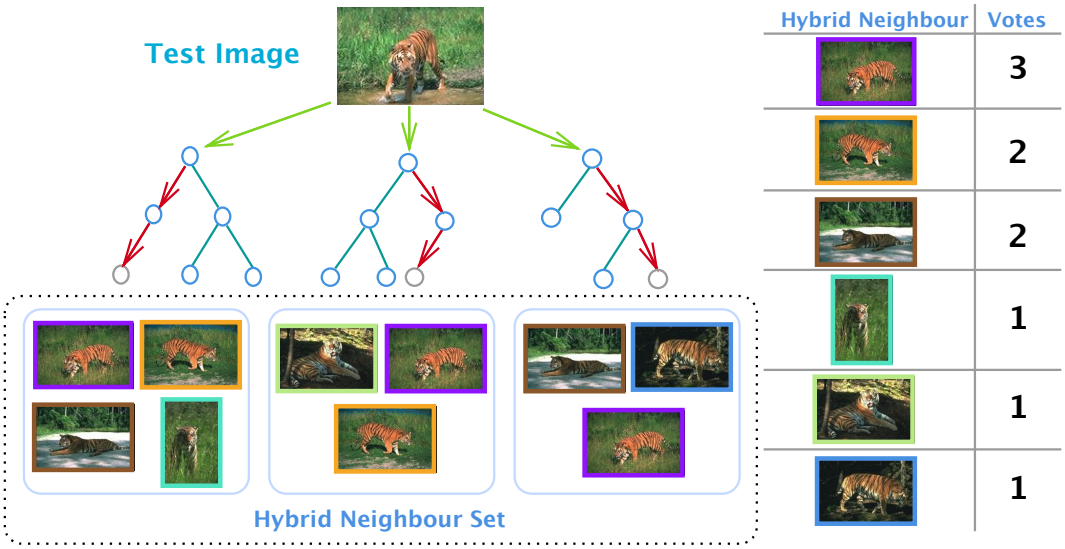


Figure 3.5: The architecture of our random forest for image retrieval and labeling. The visual feature of the testing image is put into the forest and the similar images in training set will be found. Training images with higher frequency of occurrence will enjoy a higher rank in the returned result.

3.4.1 Example Based Image Search

Scenario. The retrieval system accepts an image as the input and then returns a list of ranked images according to some similarity measure. In our case, we just put the testing image into the random forest and obtain its HNS and corresponding HNVs. The returned images are then ranked by their HNVs following a descending order. Usually the top K results will be returned by the retrieval system.

Data. We work on the popular Corel5K [54] benchmark which contains 4999 images. It is commonly split into 4500 images for training and the remaining 499 for testing, and 260 tags appear in both of these two sets.

Evaluation metric. Since the Corel5K dataset does not have ground truth results for this task, we use the K-Nearest Semantic Measure (KNSM) defined in [63] as the evaluation metric:

$$KNSM = \sum_{q=1}^{Q} \sum_{t=1}^{T} \sum_{k=1}^{K} \delta(H_{qk}, t) \tag{3.4.2}$$

where Q is the number of queries, T denotes the number of tags contained in the query image and K represents the top K retrieved Hybrid Neighbors. $\delta(H_{qk}, t) = 1$ if the query image q's tag t appears in its k^{th} HN, and $\delta(H_{qk}, t) = 0$ if not. Hence a larger KNSM indicates stronger similarity between the query image and its HNs since they share more visual themes.

Parameter setting. We eliminate tags whose visual content description levels (VCDLs) fall below 1.5, which removes 25 tags from the original tag set. Next, 100 visual themes are obtained by clustering 235 remaining tags. We initially created 600 random trees and start with different number of trees to carry out the image retrieval experiments. We found we got the best result when the tree number is set to 400. We also reproduce the result in [63] to justify the superiority of VTs over tags. In terms of the baseline method, we select the Joint Equal Contribution (JEC) [65] where various types of features are equally weighted for visual distance measurement, and is shown to perform well in image retrieval and annotation.

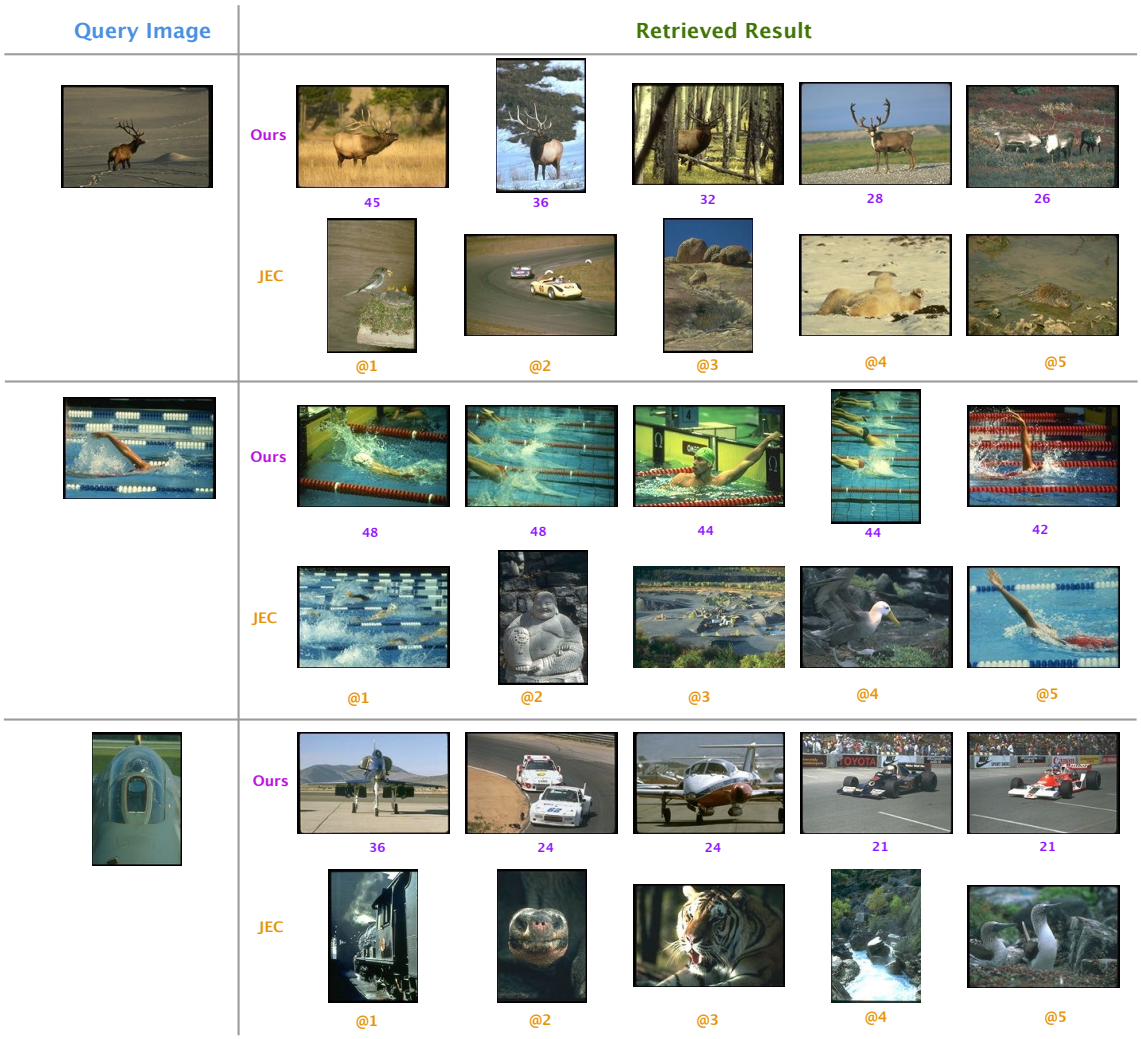


Figure 3.6: The qualitative result of the example based image search

Result. Figure 3.6 shows some qualitative results of these three methods: random forest on visual themes (RFoVT), random forest on tags (RFoT) [63] and JEC. Purple numbers under the result images denote their corresponding HNVs, the orange number is similar to HNV, but it's computed based on the tags in stead of the visual themes. The purple numbers in the third row mean the rankings of returned images using the

JEC method. Apparently RFoVT and RFoT greatly outperforms their JEC counterpart. Moreover, our RFoVT performs better than RFoT both in good (see first example) and bad cases (see the last example).

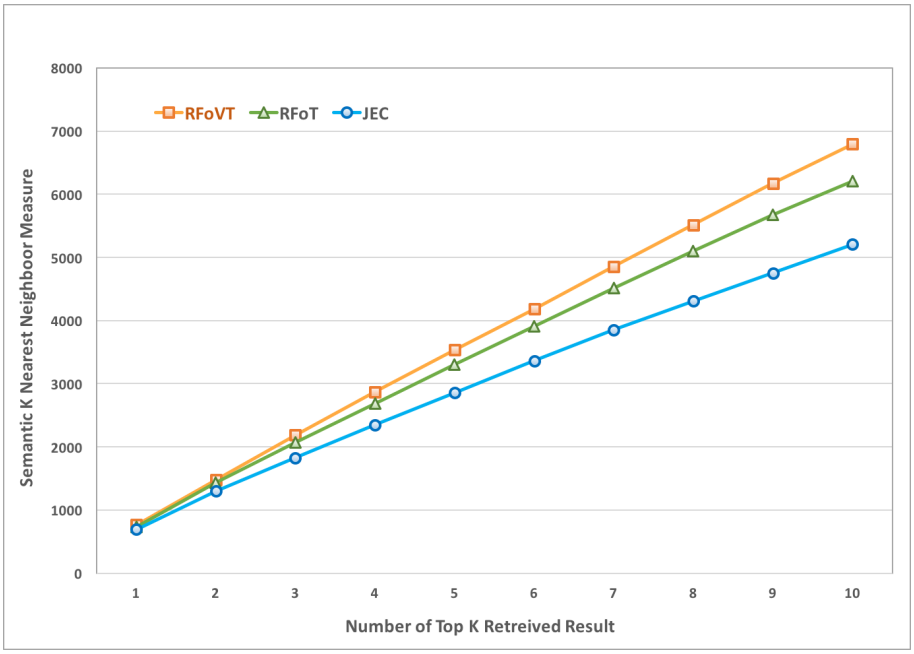


Figure 3.7: The KNSM measure of example based image search.

We also provide quantitative analysis using KNSM. We perform retrieval using all 499 testing images and result is illustrated in Figure 3.7. Clearly our method finds images with higher semantic similarity than the other two. Our success on this task demonstrates that visual themes are better than tags in terms of visual content description.

3.4.2 Keyword Based Image Search

Scenario. Given a query keyword, the retrieval system returns a collection of images that are most likely to contain that word. On this task, we tend to use a large image repository where the training instances are annotated with tags while the testing instances are not.

Data. We consider the NUS-Wide-Lite dataset which contains 55,615 images, half of them (27,807) are used for training and the other half (27,808) for testing. We directly use 1,000 tags provided by the author for visual theme discovery. There are also 81 manually defined concepts available in dataset, each concept is represented with a single word.

Parameter setting and evaluation metric. We first remove tags whose VCDLs are lower than 2.5, then cluster 904 remaining tags into 300 visual themes. We build 400

random trees and evaluate the proximity between a test instance T and a visual theme c as:

$$p(T,c) = \frac{\sum_{n=1}^{N} \delta(h_n, c) v_n}{\sum_{n=1}^{N} v_n} \tag{3.4.3}$$

where N is the size of hybrid neighbor set (HNV) of instance T, h_n denotes a hybrid neighbor (HN) in HNV, and v_n denotes the hybrid neighbor votes (HNV) of h_n. $\delta(h_n, c)$ is an indicator function which equals to 1 if visual theme c exists in h_n, and is equal to 0 otherwise.

In the experiment, we treat each visual theme as a whole keyword, which means searching with any tags within the same visual theme will obtain the same results. We compare the Mean Average Precision (MAP) achieved on visual themes (RFoVT) with five previous methods on 81 manually defined concepts, namely, K Nearest Neighbor (KNN), Support Vector Machine (SVM) [66], Entropic Graph Semi-Supervised Classification (EGSSC) [67], Label Exclusive Linear Representation (LELR) [68], and Feature Analysis and Multi-Modality Fusion (CFA-MMF) [69]. Additionally we repeat the work in [63] and construct another random forest using 81 manually defined concepts (RFoMC), then perform the same task.

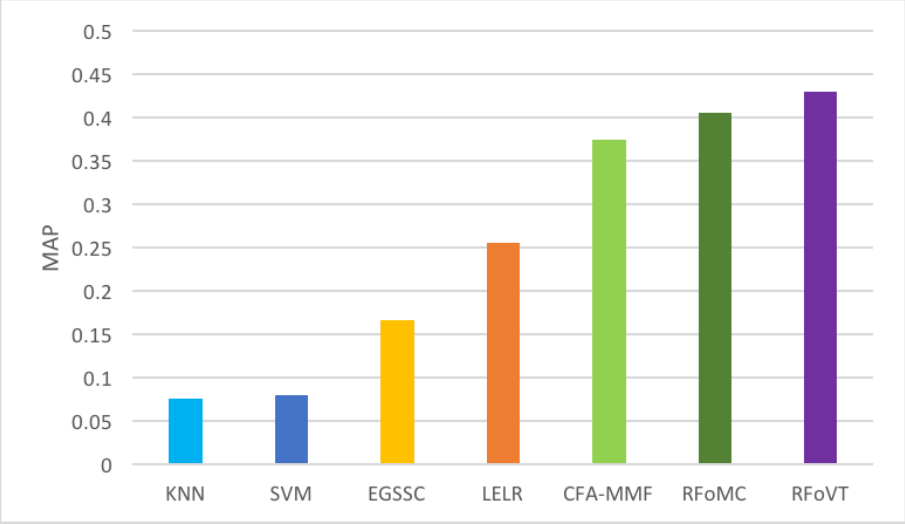


Figure 3.8: The MAP of the keyword based image search on NUS-WIDE-Lite.

Result. The overall results are shown in Figure 3.8. We can clearly see that some of previous methods have achieved much higher MAP than the KNN baseline on 81 manually selected concepts, but they still fail to achieve a MAP over 40%. While our random forest on visual themes (RFoVT) could obtain a MAP of 42.96%. This result demonstrates automatically discovered visual themes could do better than manually selected concepts in terms of visual content representation.

3.4.3 Image Labeling

In order to further explore the potential of visual themes, we perform the image labeling experiment on three well-known benchmarks: Corel5K [54], IAPR-TC12 [61] and a subset of ESP-game [62]. Table 3.3 provides details of three datasets and parameter settings of this task.

Table 3.3: Details of three image datasets and experimental parameters.

Dataset	Corel5K	IAPR-TC12	ESP Game
Training samples	4500	17665	18689
Testing samples	499	1962	2081
Tags	260	291	268
α	0.15	0.3	0.2
Random trees	400	400	400
Top voted HNs	3	3	3
Tags returned	up to 5	up to 5	up to 5

In this task, we do not perform tag filtering and only calculate the VCDLs for all tags. Given a testing image, we put it into the random forest and obtain its HNs, and retain top voted m HNs according to their HNVs. Then we collect all tags within these HNs and keep at most n tags with the highest VCDLs as the final results. It's a natural approach since selected tags are visually and semantically connected to the testing image.

We report the average precision and the average recall of image labeling with comparison to previous works in Table 3.4. From the table we can see that our method (RFoVT) outperforms all previous methods on Corel5K dataset, but its performance falls behind the TagProp [62] and RF_optimize [70] on the other two datasets. However, the success of TagProp largely depends on its tedious optimization for each image and tag, which hinders its extension to large scale dataset. While RF_optimize treats each tag as an independent unit and ignore their visual and semantic connection, which makes it less competent in dealing with noisy data. Note that web images in real world often come with considerable amount of redundant and unnecessary information. On the contrary, our image labeling method can be easily extended to large scale dataset, and can effectively eliminate the majority of noisy dataset by tag filtering. Although RFoVT does not perform very well on all datasets, it is quite simple yet efficient considering the intrinsic architecture of random forest. The result could be improved by adopting more sophisticated tag selection algorithm.

Table 3.4: Image annotation results on three datasets.

Dataset	Corel5K		IAPR-TC12		ESP Game	
Method	Precision	Recall	Precision	Recall	Precision	Recall
MBRM [71]	0.24	0.25	0.24	0.23	0.18	0.19
JEC [65]	0.27	0.32	0.28	0.29	0.22	0.25
TagProp [62]	0.33	**0.42**	**0.46**	**0.35**	0.39	**0.27**
GS [72]	0.30	0.33	0.32	0.29	-	-
SML+RF [73]	0.36	0.33	0.27	0.30	-	-
RF_optimize [70]	0.29	0.40	0.45	0.31	**0.41**	0.26
RFoVT	**0.40**	0.35	0.31	0.23	0.29	0.20

3.5 Summary

In this chapter, we have described an unsupervised framework to automatically discover visual themes which effectively depict various visual contents. Our idea of the unsupervised image tag clustering is very original and novel, which has been further demonstrated to be useful in common computer vision tasks via human evaluation and three experiments. Considering the huge amount of (noisy) tags in online image sources (e.g. Facebook, Instagram), our new image tag clustering method can potentially be used for knowledge discovery and image categorization in large online repositories.

CHAPTER 4

Semantic Attribute Based User Video Summarization

In this chapter, we build a novel offline video summarization framework to jointly use visual and semantic features to represent and summarize online user videos. More concretely, we build semantic features by introducing a set of semantic attributes. Each semantic attribute is in the form of a unique word discovered from a database consisting of web images and associated text captions. We then train a deep convolution neural network for extracting visual features as well as predicting the semantic attributes of the video segments. Based on the observation that adjacent video frames almost inevitably contain partially duplicate objects or regions, we adapt the partially duplicate image discovery technique, bundling center clustering method [74], for generating the final video summary. The experimental results show the effectiveness of the semantic attributes in assisting the visual features in offline video summarization and our new technique achieves state-of-the-art performance.

4.1 Related Work

In recent years, the rapid proliferation of video contents in the internet has created a demand for methods to perform effective video management and retrieval. However, the user-defined crowd-sourcing data such as titles, annotations and thumbnails often fail to provide specific semantic representation of the abundant visual content, which could lead to unsatisfactory retrieval performance.

A possible solution to improve the performance of video retrieval performance is to adopt automatic video summarization, which provides a synopsis for the given video content, saving time and money cost both for users and enterprises that provide video

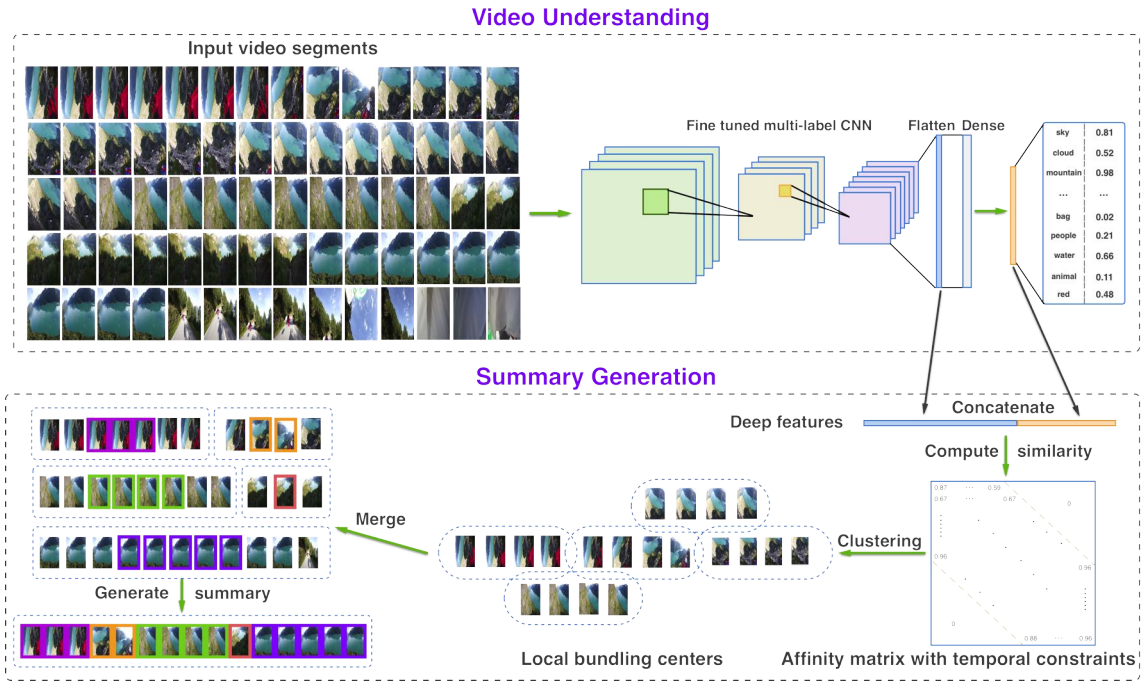


Figure 4.1: An overview of our offline video summarization framework. For video understanding, we train a deep neural network for predicting a set of semantic attributes. We then compute deep features of the input video segments and construct the affinity matrix based on the pairwise similarity and temporal constraints. For summary generation, we cluster the whole sequence of segments into several continuous groups and then concatenate the central part of some segments in each group to obtain the final video summary.

based services. Early work on automatic video summarization mainly focuses on certain domains such as sports [75] and news [76] videos, and generates summaries by leveraging domain-specific knowledge during the analysis process. However, most of these approaches only consider visual features, while the high-level semantics are often ignored.

To address this problem, some recent approaches attempt to introduce manually defined semantics to help generate video summaries, for example, interestingness [77], categoric knowledge from web images [78] and titles of user video [79], etc. An implicit assumption under this line of work is that these crowd-sourcing data, such as tags, categories, user titles are correctly given. But due to the subjective nature of the problem, different tags and titles are often used to describe the same or similar video contents, and some of them are even irrelevant. Such imperfections make these methods less applicable without proper human supervision.

In stead of using these subjective data, our work approach the video summarization

problem by automatically obtaining joint (visual and semantic) feature representation from the video themselves without explicit human supervision. We introduce a set of automatically discovered semantic attributes. Each semantic attribute is in the form of a unique word obtained from online images and associated text captions.

4.2 Methodology

Our video summarization framework is illustrated in Figure 4.1. Given an input video segment, we first use a trained deep neural network to extract the joint deep features, and then apply a clustering method to divide the whole sequence of segments into several continuous groups. Finally, we select a few segments from each group and concatenate them following the temporal order to form the final video summary.

Image	Captions	Attributes
	A plane emitting smoke stream flying over a mountain. A plane darts across a bright blue sky behind a mountain covered in snow. A plane leaves a contrail above the snowy mountain top. A mountain that has a plane flying overheard in the distance. A mountain view with a plume of smoke in the background.	blue snow sky plane mountain snowy
	The stoplight is much larger than the people standing around. A picture of a downtown intersection with a large street light. A stop light that is flashing green in front of a group of people. A traffic light turns green in a crowded city. people and buildings in a city with a streetlight	street group building green city picture light stop traffic
	The baby elephants follow the momma elephant through the field. Group of baby elephants walking behind a mother elephant in the woods. A herd of elephant standing on top of a grass covered field. A large elephant is leading two smaller elephants. An elephant and her young proceed through the forest.	group field grass elephant wood herd

Figure 4.2: The examples of attributes extracted from the original image captions (the right column).

4.2.1 Learning Semantic Attributes

As mentioned before, we want to use semantic attributes to capture the semantics in videos. In stead of using hand-labeled data, our semantic attributes are extracted from image captions, including object classes, appearance & properties and motions. The question is how to discover such attributes. We do it by examining a joint image and text corpora, where images come with human-annotated captions. Different from [80]

who directly extracts semantic features from the raw image captions, we use Stanford Corenlp toolkit [25] to automatically extract a set of attributes (words) from the captions of the training images in the Microsoft COCO [81] dataset. We then retain T most frequent words as our semantic attributes. In order to avoid overgrowing the dimension of our attribute vocabulary, we only consider the "lemma" form of a word, for example, "spots", "spotted" and "spotting" are all treated as "spot".

Some examples of extracted attributes are shown in Figure 4.2. We also examine the frequency of each attribute across all training image captions, and the result is illustrated in Figure 4.3. We notice that some attributes like "person", "street" and "white" appear with very high frequencies across the training set, which indicates that many images are annotated with such words by human annotators. It is reasonable since most of the images in the COCO database are about daily lives.

Given that attribute vocabulary, we can easily replace the original caption of an image with a small set of attributes. We then wish to train a predictor to predict the attributes of a given video segment. Considering that each segment may contain multiple attributes and some attributes may only apply to sub-regions of the segment, we treat the prediction task as a region-based multi-label classification problem [82].

Figure 4.4 summarizes our deep neural network for attribute prediction. We adopt the powerful ResNet [1] pre-trained on ImageNet [2] as our base model. We modify the structure of ResNet by inserting a fully-connected layer with 1024 neurons before the output layer and then change the target output for multi-label prediction. In the fine-tuning phase, the output of the fully-connected layer are passed to a T-way softmax function. For objective function, we use the popular binary cross-entropy loss. Assuming that we have N training samples and corresponding attribute based annotations, we then use $y_n = [y_{n1}, y_{n2}, ...y_{nt}]$ to denote the attribute vector of the n^{th} training image, where $y_{nt} = 1$ if the image contains the t^{th} attribute, and $y_{nt} = 0$ otherwise. If we use $\hat{y_n} = [\hat{y_{n1}}, \hat{y_{n2}}, ...\hat{y_{nt}}]$ to represent the predicted attribute probabilities of the n^{th} training image, then the training objective is to minimize:

$$J = -\frac{1}{N}\frac{1}{T}\sum_{n=1}^{N}\sum_{t=1}^{T}[y_{nt}\log\hat{y_{nt}} + (1-y_{nt})\log(1-\hat{y_{nt}})] \tag{4.2.1}$$

For implementation, we use the 82,783 training images from the MS COCO dataset [81] for the training purpose. We obtained 186 unique words and use them as our semantic attributes. We do not consider a large vocabulary since the distribution of discovered semantic attribute is unbalanced. The original caption of an image is then replaced with a small set of attributes as the ground-truth annotation. During the fine-tuning phase, the parameters of the new fully-connected layer and the output layer are

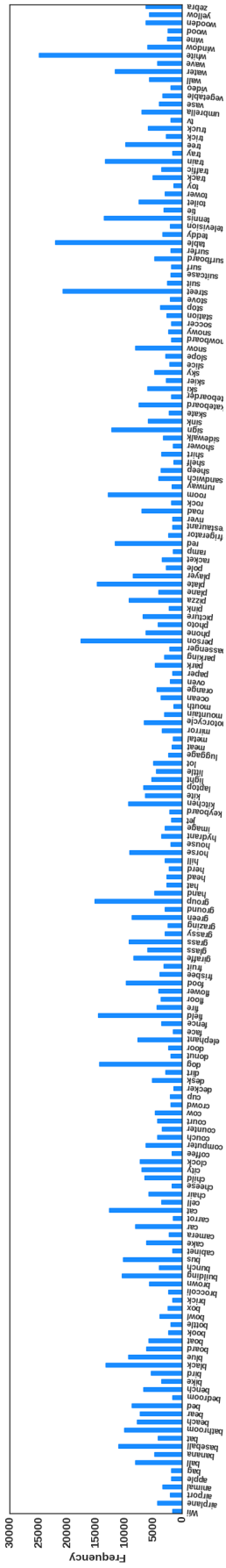


Figure 4.3: The frequency of each attribute. The blue bar denotes frequency of each attribute in the training set.

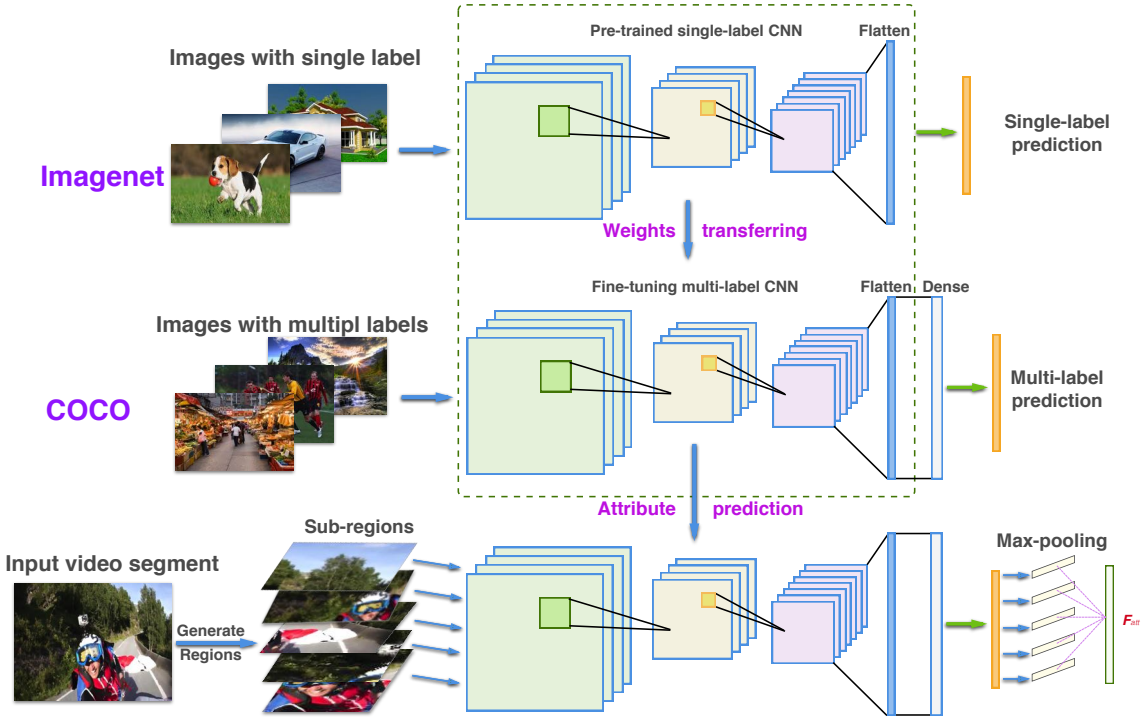


Figure 4.4: Our attribute prediction model. We build the CNN based on the ResNet [1] pre-trained on ImageNet [2]. We add a new dense layer after the last convolution block and then fine-tune the model for multi-label prediction purpose. Given a test video segment, a small set of proposed sub-regions are passed into the CNN and the output of each sub-region are then aggregated by max-pooling to generate the final attribute prediction result (denoted as F_{att}).

initialized with random values. The learning rates of these two layers are set to 0.001. All parameter values in other layers are fixed throughout the fine-tuning process. We employ the Stochastic Gradient Descent (SGD) as the optimizer and execute 20 epochs in total. The momentum is set to 0.9, and the dropout rate is set to 0.5.

After the fine-tuning process, we predict each video frame's attributes by selecting and feeding image regions into the trained model. Considering the efficiency of the deep network, we firstly employ Multi-scale Combinatorial Grouping (MCG) [83] to extract hundreds of sub-regions from the given image, we then follow a similar approach in [82] and adopt the normalized cut algorithm to cluster all region proposals into c clusters based on the IoU (Intersection-over-Union) affinity matrix. The top k proposals with highest predictive scores are then selected, which produces $c * k$ image proposals. Besides, we also add the original image to the proposal group, and then pass the $c * k + 1$ region proposals to the trained CNN for the attribute prediction. In the final stage, we use the simple max-pooling to aggregate the outputs of attribute prediction on all the proposals into a compact attribute prediction vector F_{att} (see Figure 4.4).

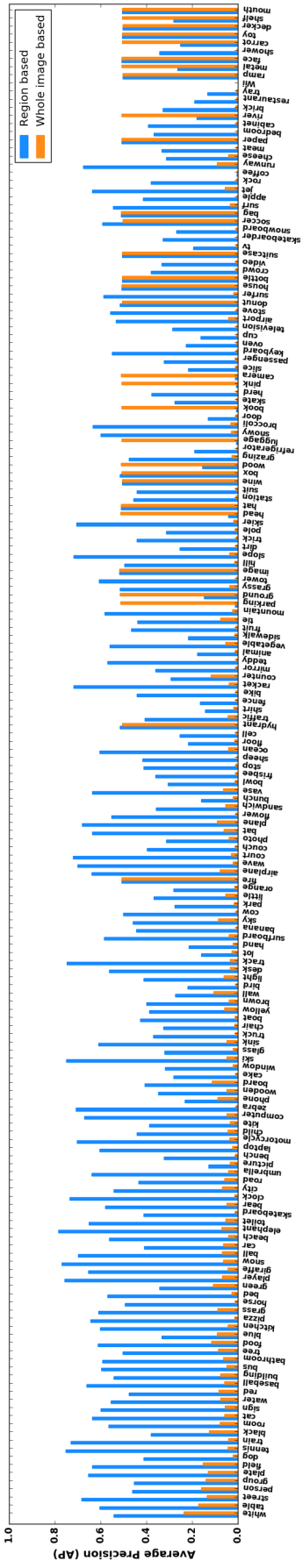


Figure 4.5: The results of the attribute prediction. The blue bar denotes the AP of region based attribute, and the orange bar represents the AP of whole image based prediction.

To evaluate the region-based attribute prediction approach, we design an experiment to predict attributes for a set of images using regions and the whole image respectively. Since no ground-truth captions of test images are provided in the MS COCO dataset, we randomly sample 5000 images from the validation set and use them for testing. The evaluation metric is the Average Precision (AP) and the mean of AP (mAP), complying with the protocols in [82].

We report APs on each attribute in Fig. 4.5. The results indicate that our region based prediction method significantly outperforms the whole image based method on more than 93% of all the attributes. The mAP on the 186 attributes is 0.439 for region-based method and 0.116 for whole image based. To provide more intuitive experience, we show some examples of attribute prediction in Fig.4.6. The experimental results demonstrate the effectiveness of our region-based attribute prediction strategy and strengthen our belief to apply the region-based method to video content analysis.

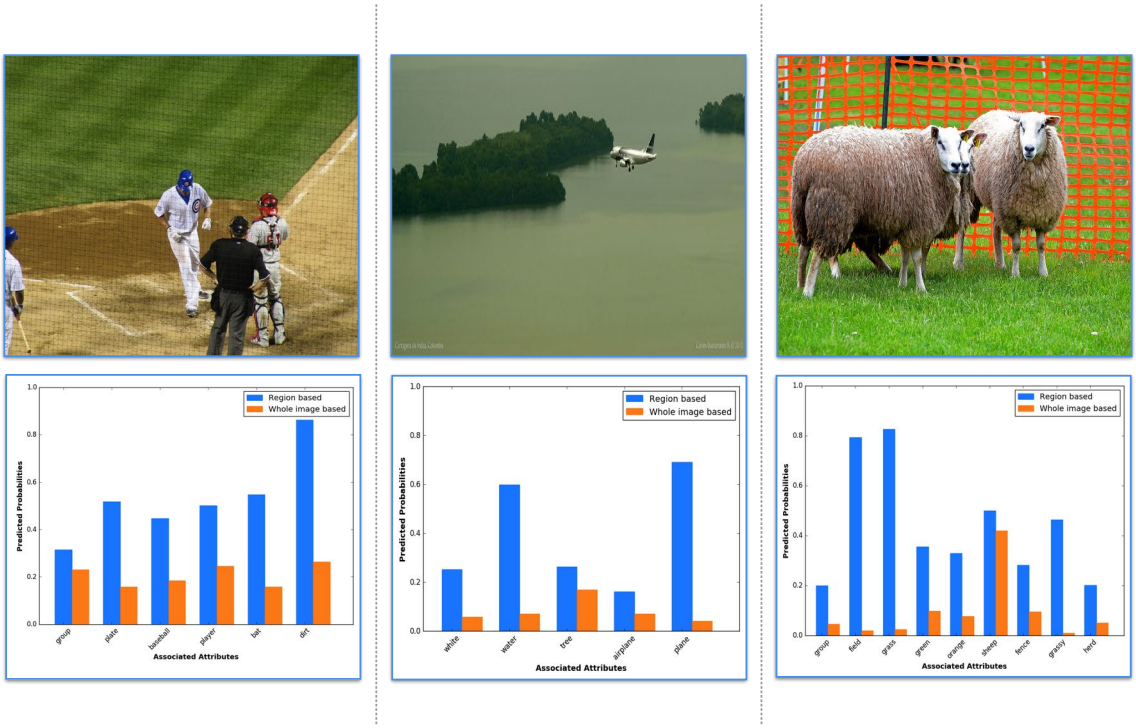


Figure 4.6: The examples of the attribute prediction. For each image, we show the predictive probabilities (PP) for the ground-truth attributes. The blue bar denotes PP of region based attribute prediction method, and the orange bar represents the PP of whole image based prediction.

4.2.2 Building Deep Features

As summarized in Figure 4.1, our deep features come with two parts. Given a single video frame, we pass it into the trained CNN and take the output of the last convolution block as the visual features. The result of attribute prediction is then treated as the semantic features. For feature fusion, we notice that the authors in [84] compute two affinity matrices on visual features and semantic features respectively, and introduce an external parameter γ (from 0 to 1) to fuse the two affinity matrices. This method is not applicable to our case, since their visual features and semantic features are extracted from two different models that are trained respectively on different image and text corpora. Moreover, the values of γ are often set empirically, and tuning the value of γ will introduce extra computational cost.

Considering that our visual and semantic features are computed using the same model, we fuse these two types of features by vector concatenation. Besides, we also investigate the impact of applying dimensionality reduction techniques to our deep features. Please see Section 4.3.1 for the experimental details.

4.2.3 Generating Video Summary

Given an input video, we divide it into small segments, and each small segment contains 10 consecutive frames. Considering the temporal redundancy in adjacent video frames, directly processing all the frames in each video segment would introduce unnecessary computational cost. Hence, for each segment, we only keep the first frame to represent the content of this video segment. For example, if a video contains 1000 frames, only 100 fames participate into computation. We then generate the video summary by finding an optimal subset of these video segments. To achieve this, our first step is to group all video segments into M continuous groups by evaluating pairwise similarity of the adjacent segments. We measure such similarity by computing the Cosine distance of their corresponding deep features extracted from our deep CNN. We then obtain a symmetric affinity matrix M_{sim} where each entry $M_{sim}(i, j)$ quantifies the similarity between segment i and segment j. Considering the temporal peculiarity of the video content, we only consider the similarity between a segment with its k temporal neighbors, so we set $M_{sim}(i, j) = 0$ if $|i - j| > k$, and $M_{sim}(i, i) = 0$.

We apply the Bundling Center Clustering (BCC) [74] to help cluster video segments into M groups. Based on the temporal constrained affinity matrix, dense neighbors [85] of each video segment from the matrix are generated, and local dense neighbor clusters with high average similarity score are then identified as the local bundling

centers. We follow [74] to grow and merge local bundling centers and finally obtain M merged large clusters, which we consider as the video segment groups. An obvious advantage of BCC is that the number of cluster M can be automatically obtained by its dynamic programming approach, which significantly reduce the time for parameter tuning.

After the clustering stage, for each group $G_i(i = 1, ..., N)$, we pick a certain length of continuous segments from its central part to avoid introducing noisy or redundant information near the group boundary (see Figure 4.1). Denote the video summary length as L_{sum}, the picked video length from each group G_i would be:

$$PickedLength(G_i) = \frac{length(G_i)}{\sum_{i=1}^{N} length(G_i)} * L_{sum} \tag{4.2.2}$$

Hence, a longer group will contribute relatively more video segments for the summary generation. The final summary is then obtained by concatenating all the selected video segments $S_m(m = 1, ..., M)$ in the temporal order.

4.3 Experiment and Discussion

To evaluate the proposed framework, we use the SumMe [77] dataset containing 25 videos as our testbed. This dataset features various types of user-generated videos, such as static, moving and egocentric, and most of them are either unedited or minimally edited. Each video in the SumMe dataset contains at least 15 manually-created video summaries, we treat all of them as the ground truth to evaluate our method. Two types of evaluation metric, namely, the maximun-based f-measure [86] and the average-based f-measure [77] are used. More specifically, the maximum-based f-measure is computed based on the most similar human generated summary, while the average-based f-measure is computed on all manually created summaries. We use both of them to compare our approach with other state-of-the-art methods.

We use the code provided by [77] to compute the average f-measure and maximum f-measure. For constructing the temporal affinity matrix M_{sim}, we set $M_{sim}(i, j) = 0$ if $|i - j| > 20$ (we have varied this number and obtained similar results). The summary length L is set to be approximately 15% of the input video's length following [77].

4.3.1 Evaluating Different Feature Construction Methods

To examine the performance of different feature construction approaches, we design the following five methods: 1) visual features only (VF, 2048-d); 2) semantic features only (SF, 186-d); 3) concatenate visual and semantic features (VSF, 2234-d); 4) concatenate visual and semantic features, then apply PCA to the joint features (PCA-VS, 442-d); 5) apply PCA to the visual features, then concatenate visual and semantic features (PCA-V+S, 256-d + 186-d).

We use these five types of deep features to summarize videos in the SumMe dataset and use the average f-measure as the evaluation metric. The results shown that the PCA-V+S approach outperforms all the other approaches in terms of the average f-measure (0.243) computed on all the 25 videos. We also notice that only using semantic features gives the worst performance in this experiment (0.176), this is probably caused by the lack of visual cues in the semantic space. Besides, we learn that directly concatenate visual and semantic features are not applicable, due to the dimensionality difference (2048 for visual part and 186 for semantic part). Hence, we adopt the PCA-V+S approach to compute our deep features for the summarization task. The complete result could be found in Table 4.2.

4.3.2 Maximum-based Evalution

We compare our SASUM with some recent approaches using the maximum f-measure: 1) **Interestingness** video summarization [77] is a supervised method which uses several manually defined objective function to help summarizing videos. 2) **Submodular** [86], in which a submodular function is learned to optimize the objective function for selecting video frames. 3) **DPP** [87] is a supervised approach which use Determinant Point Process (DPP) to help generating video summaries. 4) **dppLSTM** [88] is a supervised method which combines both of DPP and LSTM. 5) **Video MMR** [89] is an unsupervised approach which defines redundancy and representativeness to select video frames for summary generation.

As shown in Table 4.1, our approach achieves the highest overall score of 0.521 on the SumMe dataset (the previous state-of-the-art result published very recently was 0.429 [88]). The results demonstrate that the proposed approach is able to create video summaries closer to the human-level performances than other approaches.

Some examples of video summaries generated using our method is shown in Figure 4.7. The peaks of blue lines mean that the corresponding video segments enjoy high popularity for being selected by human subjects. From the figure we could observe

Table 4.1: The quantitative results with the maximum-based evaluation. We report the mean maximum f-measure computed on all videos in the SumMe dataset.

	Method	Mean Max F-measure
Supervised	Interestingness	0.394
	Submodular	0.397
	DPP	0.413
	dppLSTM	0.429
Unsupervised	Videov MMR	0.266
	SASUM (Ours)	**0.521**

that the segments selected by our method (orange blocks) show strong correlation to the blue lines. This demonstrate that our approach is consistent with human perception of the visual contents.

4.3.3 Average-based Evaluation

We compare our SASUM with several recent approaches using the average-based evaluation: 1) **Uniform sampling** is a popular baseline method. 2) **Interestingness** [77] manually defines and optimizes some objectives based on the SumMe dataset. 3) **Attention** model focuses on the visual attention [90] to generate video summary. 4) **Title-based** [79] is a semi-supervised method which leverages a set of visual concepts discovered using video titles in the SumMe dataset. 5) **Semantic** approach is an unsupervised technique that maps the visual content to the semantic space [80] for summary generation. 6) **WebPrior** [78] uses web-image based prior information to help generate video summaries. 7) **Quasi** [91] learns a dictionary from the given video using group sparse coding for summary generation.

We only report the mean average f-measure scores of the **WebPrior** [78] and **Quasi** [91] since the f-measure on each video are not available: [78] achieves 0.24 while [91] achieves 0.246. The remaining results of the average-based evaluation is shown in Figure 4.8. It could be noticed that our SASUM outperforms all the other video summarization approaches except the title-based. Note that this approach leverages the video titles in the SumMe dataset to find some groups of web images, and then a mapping function from images to videos is learned to generate visual concepts for the summarization task. Adding such domain specific knowledge would be helpful for video summarization. Our video summaries are generated using a relatively simple framework without prior knowledge from these videos. Nevertheless, our approach outperformed the title-based method for some videos. We believe the performance of our

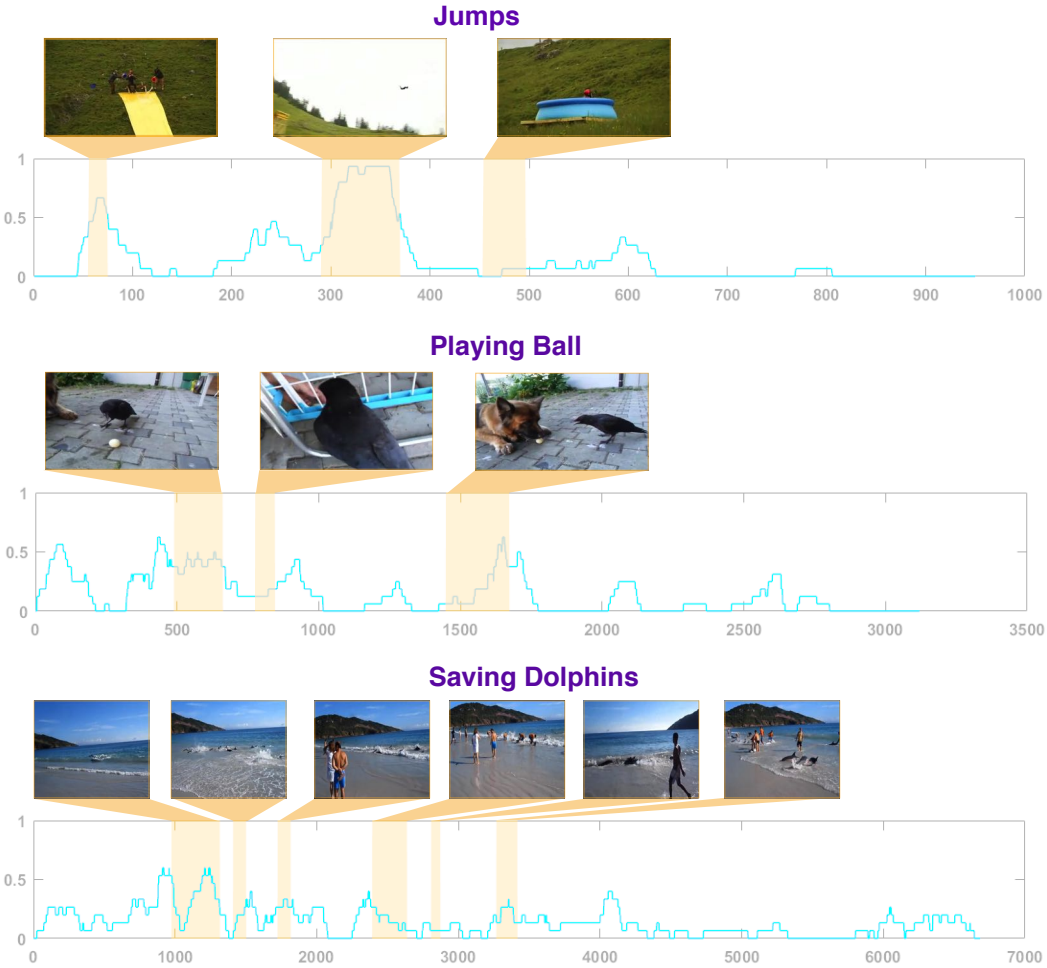


Figure 4.7: Some qualitative results of the video summarization. The orange blocks represent the video segments selected by our approach. For each segment, we show an image shot at its central part. The blue lines denotes the ratio of human annotators who agree to include each frame in their manually-created summaries.

approach could be enhanced by defining domain-specific semantic attributes and by adopting more sophisticated video segmentation techniques like [77].

4.4 Summary

In this chapter, we present the investigation into the value of automatically mining high-level semantic information for the offline video summarization problem. In the process, we design an algorithm to learn a set of semantic attributes that are automatically discovered from a joint image and text corpora. We then predict attributes on user videos and use the predicted output as an essential part of our deep features. We employ the bundling center clustering method to help generating the final video summary. By comparing our result with several recent computational approaches, we show the advantage of our joint deep features for the video summarization problem.

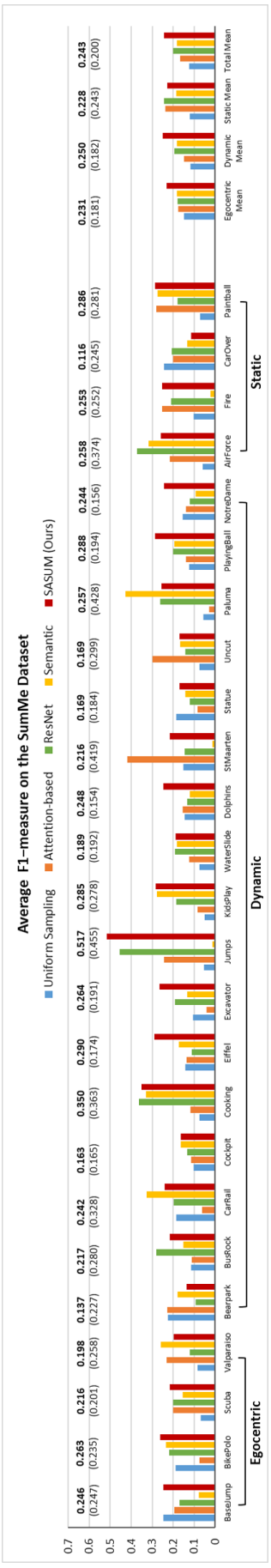


Figure 4.8: The quantitative results with the average-based evaluation. Bold-faced numbers represent our results; numbers in parentheses represent the highest score from any approach. The last column shows mean scores computed on all videos.

Table 4.2: The results of the video summarization using different types of deep features. We report the f-measure on each video and the average f-measure on all videos. The best scores among these approaches are highlighted.

Video Info		Different Types of Deep Features				
Type	Name	VF	SF	VSF	PCA-VS	PCA-V+S
Egocentric	Base Jumping	0.169	0.152	0.122	0.210	**0.246**
	Bike Polo	0.219	0.163	0.208	0.159	**0.263**
	Scuba	0.201	0.129	0.157	0.137	**0.216**
	Valparaiso Downhill	0.122	0.140	0.195	0.175	**0.198**
Dynamic	Bearpark Climbing	0.093	0.128	0.128	**0.140**	0.137
	Bus in Rock Tunnel	**0.28**	0.123	0.145	0.218	0.217
	Car Railcrossing	0.197	0.185	0.182	0.198	**0.242**
	Cockpit Landing	0.132	0.142	0.132	0.114	**0.163**
	Cooking	**0.363**	0.148	0.133	0.122	0.350
	Eiffel Tower	0.111	0.179	0.285	0.114	**0.290**
	Excavators River Crossing	0.191	0.204	**0.264**	0.221	**0.264**
	Jumps	0.455	0.335	0.350	0.374	**0.517**
	Kid Playing in Leaves	0.186	0.236	0.281	0.276	**0.285**
	Playing on Water Slide	**0.192**	0.144	0.189	0.135	0.189
	Saving Dophines	0.134	0.191	0.240	0.197	**0.248**
	St Maarten Landing	0.147	0.206	**0.216**	0.202	**0.216**
	Statue of Liberty	0.121	0.138	0.146	0.094	**0.169**
	Uncut Evening Flight	0.142	0.145	0.141	0.105	**0.169**
	Paluma Jump	**0.261**	0.219	0.227	0.228	0.257
	Playing Ball	0.201	0.215	0.214	**0.300**	0.286
	Notre Dame	0.121	0.174	0.177	0.113	**0.244**
Static	Air Force One	**0.374**	0.263	0.258	0.183	0.258
	Fire Domino	0.209	0.120	0.140	0.230	**0.253**
	Car Over Camera	**0.208**	0.093	0.100	0.110	0.116
	Paintball	0.18	0.225	0.158	0.100	**0.286**
	Mean	0.200	0.176	0.192	0.178	**0.243**

CHAPTER 5

Detecting, Tracking and Counting Vehicles for City Road Traffic

In this chapter, we present an advanced urban traffic analysis solution using the latest deep learning techniques to intelligently process 4K ultra high definition traffic videos taken from an Unmanned Aerial Vehicle (UAV). We first capture nearly an hour-long ultra high-resolution traffic video at 5 busy road intersections of a modern megacity by flying an UAV during the rush hours. We then randomly sampled over 17K 512x512 pixel image patches from the video frames and manually annotated over 64K vehicles to form a dataset for this research which will also be made available to the research community for research purposes. Our innovative urban traffic analysis solution consists of advanced deep neural network based vehicle detection and localization, type (car, bus and truck) recognition, tracking and vehicle counting over time. In the experiments, we show that our enhanced single shot multibox detector (Enhanced-SSD) outperforms other deep neural network based techniques and that deep learning techniques are more effective than traditional computer vision techniques in traffic video analysis. We also show that 4K ultra high-resolution video provides more information which enables more accurate vehicle detection and recognition than lower resolution contents.

5.1 Related Work

Urban traffic monitoring has long been a popular research topic among scholars and industrial practitioners. Considering the rapid growth of our metropolis road network and the booming of private cars in recent decades, it is indispensable to build a more comprehensive system to help understand the intricate transportation system in the

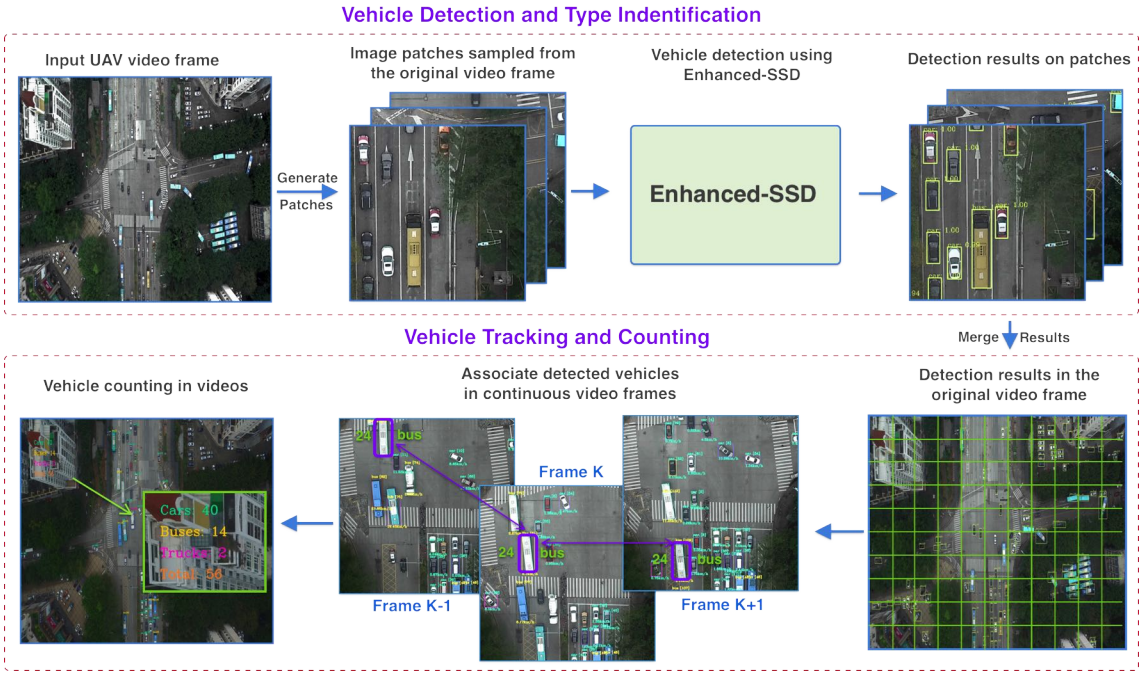


Figure 5.1: The working flow of the Deep Vehicle Counting Network. Detection is performed on image patches which are extracted from the original input video frame, and then the results are stitched back together to obtain the global result. In the tracking & counting phase, a set of trackers are built to capture unique vehicle identities across the whole video sequence. The numbers of vehicles could be obtained by counting the outputs of trackers.

urban area. Conventional traffic monitoring systems rely on thousands of detectors (e.g. cameras, induction loops, radar sensors) deployed on fixed locations with small detecting ranges to help capture various road conditions throughout the network [92–95]. Such kind of system has exhibited many limitations in terms of range and effectiveness. For instance, if the information is required beyond the scope of these fixed detectors (i.e. blind regions), human labors are then frequently deployed to assess these particular road conditions [96]. Besides, many monitoring tasks require to temporally detect detailed traffic conditions such as sources and destinations of the traffic flow, regions of incidents and queuing information at crossroads [97–99]. To achieve this, the visual information of multiple fixed detectors need to be aggregated in order to provide a relatively large view of the interested area, which could introduce extra noisy information and the overhead costs. Therefore, it is essential to develop a more effective approach for acquiring visual information.

To tackle these issues, some previous works attempt to exploit still satellite images for traffic monitoring [100–103]. Satellites allow observing wide areas, but they lack spatial resolution for specific ground locations. Additionally, the data acquisition and

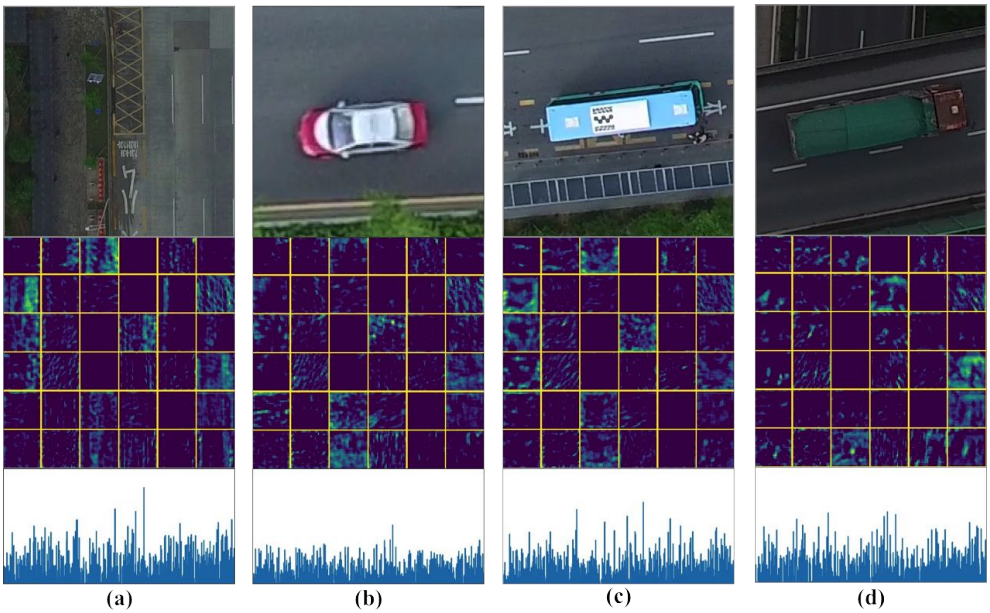


Figure 5.2: The examples of deep features computed on different types of vehicles including the background: (a) background, (b) car, (c) bus and (d) truck. In the top row, the original images are shown, while related deep features extracted from a convolution layer and a fully-connect layer are illustrated in the middle and bottom row respectively.

processing are complicated and time consuming, which hinders its application to real-time urban traffic monitoring tasks.

Thanks to the technological advances in electronics and remote sensing, Unmanned Aerial Vehicles (UAVs), initially invented for military purposes, are now widely available on the consumer market. Equipped with high-resolution video cameras, geo-positioning sensors and a set of communications hardwares, UAVs are capable of capturing a wide range of road situations by hanging in the air or by traveling through the road network without restrictions [104–107]. Traffic videos captured by UAVs contain important information for traffic surveillance and management, and play a vital role in multiple fields such as the transportation engineering, density estimation and disaster forecasting [108–110].

Among these tasks, traffic density estimation is of significant importance since it provides direct information about the traffic condition in various locations across the city road network, helping traffic authority better design traffic rules and manage the light signal system [107, 108, 110]. However, UAVs are not widely applied in the traffic density estimation system due to specific challenges for detecting and tracking vehicles in the UAV's images and videos.

On one hand, the equipped camera of a UAV may rotate and shift during the recod-

ing process. Moreover, sudden camera shakes could happen due to the unexpected airflows or electromagnetic interference, which would pose negative impacts on the data quality. On the other hand, compared with conventional monitoring systems, the UAV's video contains not only the ordinary data such as the global view of the traffic flow, but also each vehicle's own data like its moving trajectory, lane changing information and interaction with other vehicles [111, 112]. Therefore, the UAV's video needs to be recorded using a very high resolution and frame frequency so as to capture adequate ground details. This inevitably leads to a huge size of the UAV's video data and pose challenges to vehicle detection and tracking algorithms [113].

Detection Based Approaches

Many existing vehicle detection methods for aerial images mainly adopts sliding window based searching and hand-crafted feature matching techniques to identify and localize vehicles in an image (or a frame in videos) [96, 114–116], however, due to the lack of high-level semantic information in terms of vehicle types, all the detected objects are treated as vehicles. Note that identifying vehicle types in traffic density estimation is essential since the capacity of different vehicles contribute differently to the road traffic pressure. Some work adopts extra classifiers to recognize different types of vehicles [117], but their approaches introduce more overhead cost for computation and parameter optimization.

Motion Based Approaches

Several methods try to estimate traffic density using motion based vehicle tracking techniques (e.g. background subtraction and optical flow) [96, 118, 119]. These approaches could work well on simple traffic scene such as the expressway and roads in the rural area, but they tend to fail in the urban traffic scene due to the distraction of various background noises and intricate local ground conditions. Additionally, some vehicles appear in only a few frames and their trajectories cannot be accurately estimated.

Deep learning based Approaches

Recently a few deep learning based methods were proposed for object density estimation [120–123]. These methods attempt to predict the object density from a holistic view using deep neural networks (DNN). However, the original images have to be down-sampled in order to be processed by DNN, which would lead to the loss of local pixel-wise information. Besides, the problem of scale variation of moving objects is not well addressed.

To summarize, detection based and motion based approaches cannot work well on city

road traffic density estimation, because they are sensitive to video quality and road conditions. Moreover, many existing methods are unable to provide the accurate number of different types of vehicles. While our technique for traffic density estimation can work on high-resolution videos recorded under complex city road conditions, and the proposed deep vehicle counting framework (DVCF) can count different types of vehicles with accurate numbers. To the best of our knowledge, this is the first framework which integrates deep neural networks and traditional algorithms for analyzing 4K (3840×2178) UAV road traffic videos.

5.2 Methodology

In this section, we introduce the traffic data acquisition and pre-processing, then we elaborate the Deep Vehicle Counting Framework for traffic flow analysis. To be more specific, we handle the vehicle counting problem in two stages. The first stage is a modified and sliding window based Single Shot Multi-box Detector (Enhanced-SSD), which is able to produce bounding boxes of vehicles with type information in 4K videos. The second stage is a fast multi-object tracker applied on these bounding boxes, estimating each vehicle's trajectory, maintaining its unique identity and the corresponding type. Such a tracking-by-detecting framework is a natural approach to process very high resolution UAV videos since performing vehicle detecting and tracking simultaneously is almost impossible considering such high computational cost. The whole workflow is illustrated in Fig. 5.1.

5.2.1 Data Acquisition

A UAV traffic monitoring system has been set up to acquire the traffic data, which consists of a quadrocopter (Fig. 5.3a), a remote controller with built-in video transferring system (Fig. 5.3b) and a camera mount (Fig. 5.3c).

The quadrocopter used in the experiments is the Dajiang Innovations (DJI) Inspire 1 Pro. It contains motors, main controller, battery and the connection port for the camera mount. The UAV is designed to be lightweight, flexible and stable when recording high-quality videos. With the help of the built-in inertial measurement unit (IMU) which incorporates both a 6-axis gyroscope and an accelerometer for movement compensation, the camera mounted by the UAV is capable of stably recording road traffic at 4K resolution (30fps). The third part of the UAV is the remote controller which transmits real time video stream and UAVs' flight data back to the controller, such as the

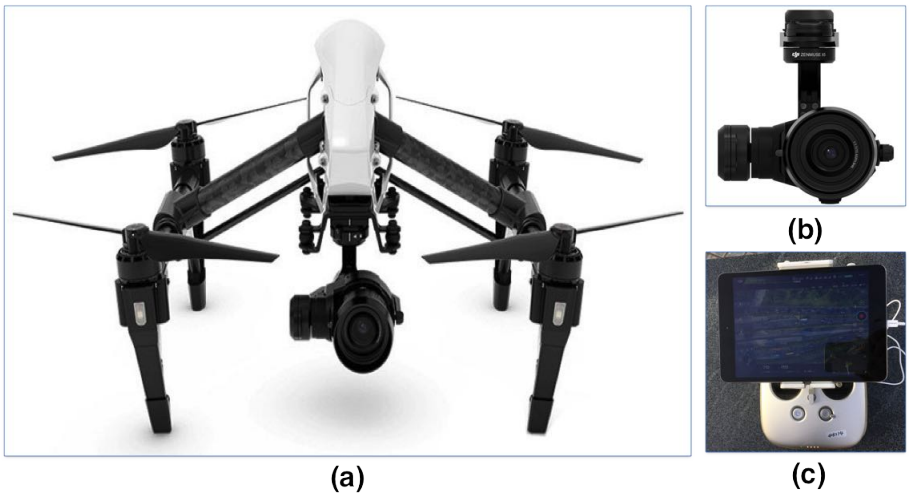


Figure 5.3: The UAV traffic monitoring system used in this paper: (a) the whole set, (b) the camera mount and (c) the remote controller .

distance between the aircraft and the remote controller, GPS location, flight velocity, etc. We collect traffic data in Shenzhen, a typical metropolis which undertakes significant traffic pressure in China. We pick 5 key road intersections in Shenzhen to acquire our traffic data. Considering the capability issues, the videos were firstly stored in the camera's SD card, and then transferred to the computer. Some parameter settings of the data collection are listed in Table 5.1.

Table 5.1: The parameter settings for data collection

Parameter	Range/Value
Time slot	Peak hours (7:00-9:00am, 5:00-7:00pm)
Weather condition	Sunny/cloudy
Operating temperature	-10°C∼50°C
Hovering altitude	126 meters above ground
Hovering accuracy (GPS Mode)	Vertical: 0.5 m, Horizontal: 2.5 m
Ground resolution	5.5 cm per pixel
Number of videos taken	10
Video length of each record	up to 10 minutes
Video resolution	3840×2178
FPS	30

We use this UAV toolkit to collect and then labeled a large-scale UAV city traffic dataset (UavCT) from 5 busy intersections of the city. In the UavCT, the video's physical resolution is 5.5 cm per pixel at the ground level. This makes cars range in size from 80 to 180 pixels. To build the training set, we first temporally subsample the original video frames by a factor of 150, then for each frame in the subset, we divide it into small

patches with a uniform size of 512 × 512. We allow an overlapping area of 200 pixels vertically and horizontally between these patches to ensure each vehicle appears as a complete object. The final training set contains 17,186 image patches. These patches are then annotated with the following information: (i): *Bounding box*: rectangle surrounding each vehicle. (ii): *Vehicle type*: three general types including car, bus, and truck. Note that we do not classify vehicles into very specific categories (e.g. private cars, taxi, etc.) because too many types would inevitably exacerbate the problem of unbalanced data, which would lead to sub-optimal performance of machine learning algorithms. An example of the data annotation is illustrated in Fig. 5.4.

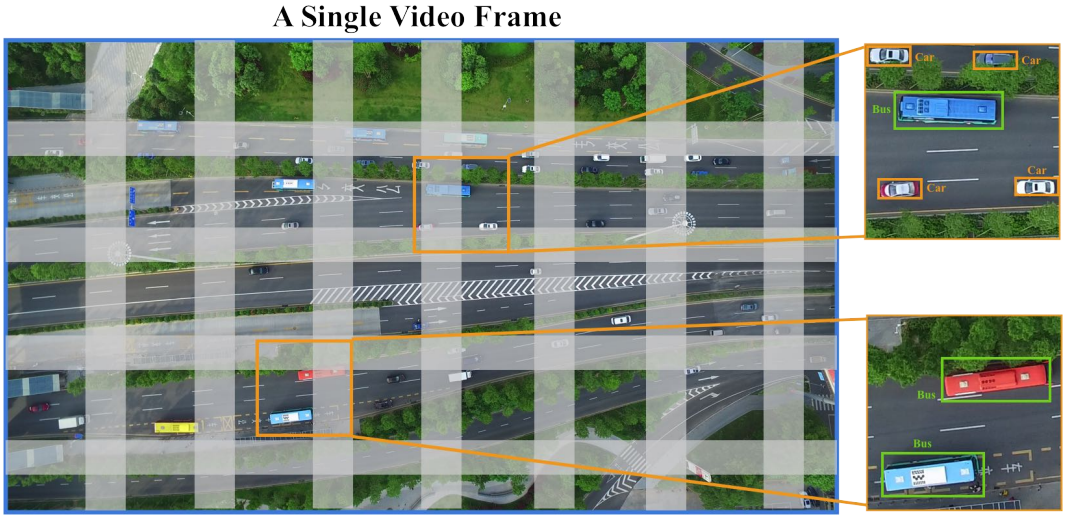


Figure 5.4: A example of the data annotation performed on the UacCT dataset. The overlapping areas are denoted by gray bars. The yellow boxes are image patches extracted from the original video frame. In each patch, vehicle are annotated with bounding boxes and corresponding types.

To challenge the robustness of the vehicle detection and tracking algorithms, we ask human annotators to avoid labeling similar vehicles for multiple times. In other words, most vehicles are annotated only once. But in the testing phase, they are required to be detected multiple times at different locations across the whole video sequence. More details about the training set are described in Table 5.2.

Table 5.2: The number of annotated vehicles in each training video of the UavCT dataset.

Type/Video	1	2	3	4	5	Total
Car	12190	9680	6281	8409	15531	52091
Bus	3082	940	704	181	1411	6318
Truck	330	284	143	3174	1727	5658
Total	15602	10904	7128	11764	18669	64067

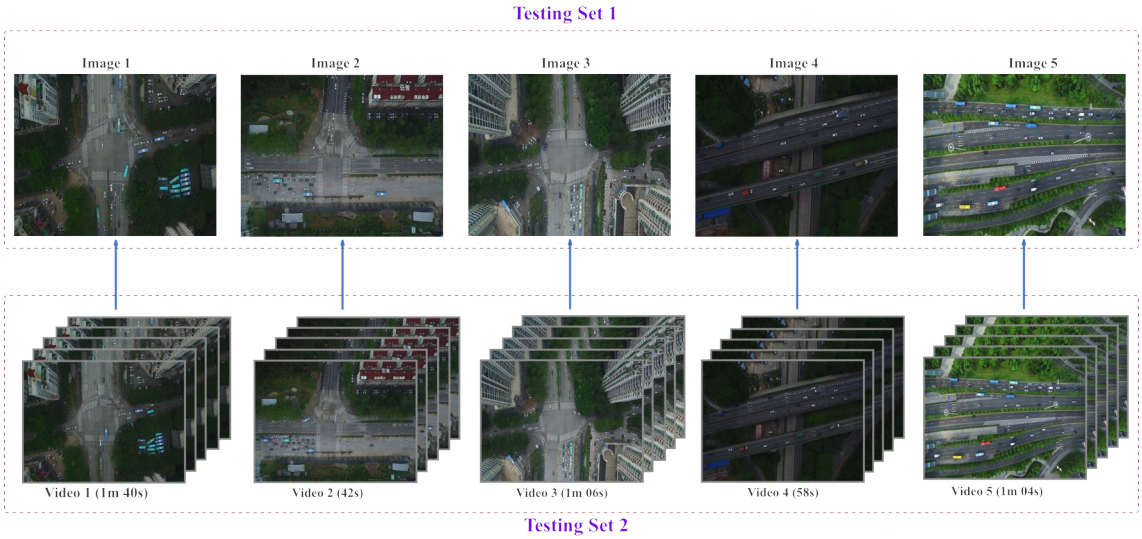


Figure 5.5: The overview of the two testing sets.

Table 5.3: The number of vehicles in each image in the testing set 1.

Type/Image	1	2	3	4	5	Total
Car	77	60	81	23	42	283
Bus	21	3	3	0	8	35
Truck	2	0	1	5	6	14
Total	100	63	85	28	56	332

For building the testing set, we collect traffic data from the five road intersections again but limit the length of each video to be no more than 100 seconds. That is to say, we take testing videos at different time slot from training videos. We then construct two testing sets to evaluate the proposed framework. The first one contains five images, each image is one full resolution frame (4K) which are randomly sampled from each testing video accordingly. The second testing set consists of the original five testing videos (see Fig. 5.5). The whole length of videos in testing set 2 is 5m 30s. To build the ground truth of the testing set 1, we ask human subjects to count numbers of different types of vehicles in each image in set 1. For the testing set 2, only vehicles within the road range are counted because other vehicles do not make contribution to the traffic pressure. More details of testing set 1 and set 2 could be found in Table 5.3 and Table 5.4 respectively.

5.2.2 Vehicle Detection

We propose a deep learning based vehicle detection method. Unlike previous works which perform vehicle detection and classification separately, we integrate these two parts in a whole deep neural network (DNN) named enhanced single shot multi-box

Table 5.4: The number of vehicles in each video in the testing set 2.

Type/Video	1	2	3	4	5	Total
Car	124	65	90	124	195	598
Bus	22	7	4	1	24	58
Truck	2	0	5	36	28	71
Total	148	72	99	161	247	727

detector (Enhanced-SSD).

The two main components of this approach are feature description and vehicle localization. To generate feature descriptors with class information, a set of convolutional layers in a DNN which is initially applied for image classification are used to construct our base network. In the vanilla single shot multi-box detector (SSD) [124], small convolutional filters are used to predict the object classes and coordinate offsets of the bounding boxes. Besides, a set of separate filters for detections with various aspect ratio are applied to multiple feature maps in order to detect objects at multiple scales.

For the proposed Enhanced-SSD, we follow similar approaches to build the detector as the vanilla single shot multi-box detector (SSD), however, the main difference between Enhanced-SSD and the vanilla SSD is that we employ the more powerful classification model called ResNet [4] as our based model. We do this because SSD has the limitation that very small objects (i.e. small vehicles in 4K videos) are not detected well, and replacing the original VGGNet [55] with ResNet would increase the number of layers and total number of channels, thus it can describe the object feature with more details than the original VGGNet model adopted in the vanilla SSD. Besides feature description, another key component is the vehicle localization. To achieve this, we add several auxiliary convolutional layers and a pooling layer to the base network for predicting the locations of vehicles. Then we feed the output of 6 layers (two from the reduced ResNet and four from the newly added layers) to the classification layer for generating each vehicle's location and its corresponding category. (see Fig. 5.6). Note that Pool6 is a Global Average Pooling [125] layer, and its size is determined by the number of categories (i.e. 4 classes in our case: background, car, bus and truck). Moreover, only layers in the multi-scale feature layer group are illustrated in Fig. 5.6. Other three layers, namely Conv1-1, Conv2-1 and Conv3-1 are all designed with 256 filters, the kernel size is 1×1 and the stride is 1.

Multi-scale feature layers for vehicle detection. Similar to the conventional single shot multi-box detector (SSD), in a convolutional manner, we initialize a set of detecting boxes with various scales and aspect ratios at each location in these multi-scale feature

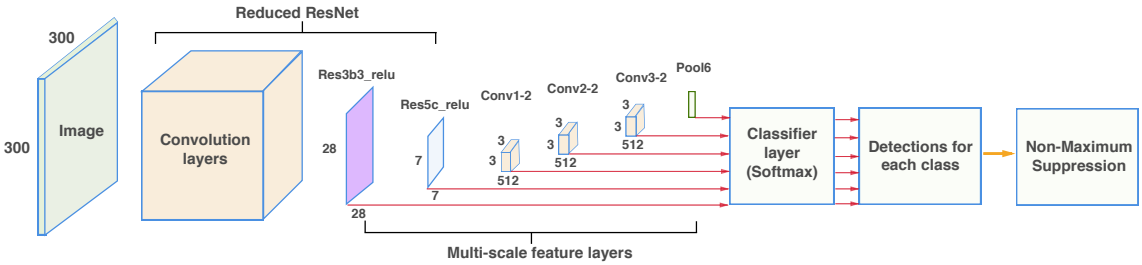
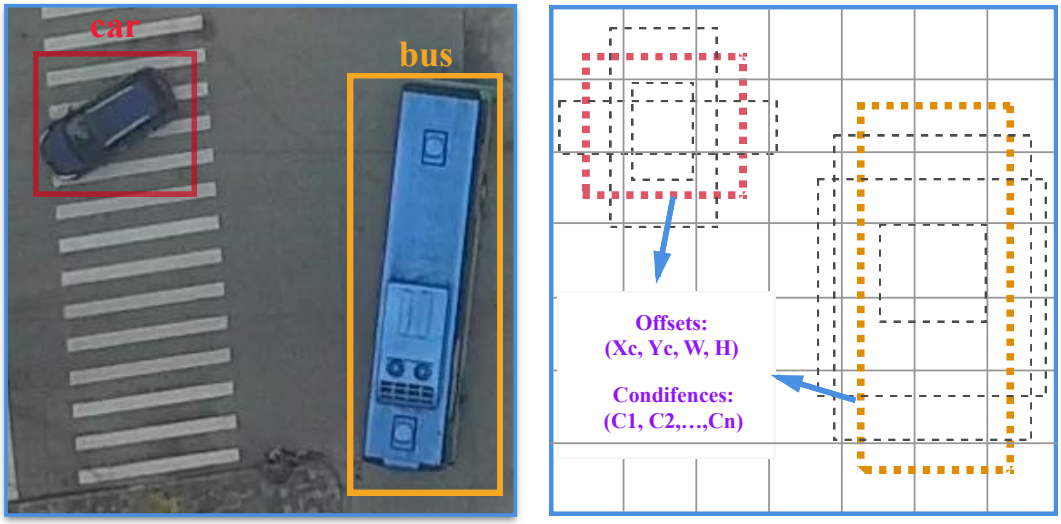


Figure 5.6: The overall structure of the Enhanced-SSD. The layers from Res3b3 to Pool6 which are fed into the classifier layer are responsible for predicting the locations of vehicles in different scales and aspect ratios. All outputs of the detected bounding boxes are filtered by thresholding in the Non-Maximum Suppression algorithm.

layers. For each detecting box, we calculate shape offsets [124] and confidences (class scores) for each vehicle's category $\{C_1, C_2, ..., Cn\}$. In a feature layer of size $i \times j$ with p channels, a small kernel (e.g. 4×4) is applied to predict either the class score or the shape offsets relative to the coordinates of the initial detecting box. More specifically, for each detecting box in m given locations, we compute n class scores and the 4 offsets relative to the original detecting boxes. This yields a total number of $(n+4)m$ filters which are applied on each location in the feature map, and resulting in $(n+4)mij$ outputs for a $m \times n$ feature map (see Fig. 5.7). Varying the shape of detecting boxes in several feature maps allows us to effectively capture different vehicle sizes in high resolution traffic videos.

Training. Training Enhanced-SSD is straightforward: we find which detecting boxes are close to a ground truth box and then tuning the parameters of the network accordingly. Specifically, for each ground truth box, we firstly pick a small set of detecting boxes with various locations, sizes and aspect ratios. And then we sequentially match each ground truth box to the detecting box according to the best Jaccard overlap [126]. Unlike the approach in [126] which reserves only one detecting box with the maximum overlap, we replace it with a threshold α (0.5 in our experiments) and allow matching detecting boxes to any ground truth box with the Jaccard overlap higher than α. This reduces the computational cost and avoids the risk of missing detecting boxes with high Jaccard overlap scores.

The training objective is extended from MultiBox objective [126] by adding support for multiple vehicle categories. It is consisted of two components: the localization loss (loca) and class confidence loss (conf). The former is responsible for localizing vehicle in input images and the latter is aiming to identify its corresponding type. Denoting $\delta(m, n, c) = \{0, 1\}$ as an indicator for matching the m^{th} detecting box to n^{th} ground truth box with category c, then according to the matching strategy, we could



(a) Input image with ground truth boxes **(b) Feature map (8 x 8 or 4 x 4)**

Figure 5.7: The structure of the multi-scale feature maps in Enhanced-SSD. Given an input image with ground truth bounding boxes (e.g. (a)), we first initialize a small set of detecting boxes with various aspect ratios at each location in several feature maps (e.g. 8×8 or 4×4 scales in (b)). Then for each detecting box, we predict the shape offsets of bounding boxes and the class scores for all vehicle types.

have $\sum_i \delta(m,n,c) \geq 1$. The overall objective function is:

$$L(\delta,c,p,g) = \frac{1}{N_{match}}(\gamma L_{loca}(\delta,p,g) + L_{conf}(\delta,c)) \tag{5.2.1}$$

$$L_{loca}(\delta,p,g) = \sum_{i\in Pos}^{N} \sum_{m\in\{cx,cy,w,h\}} \delta_{ij}^{k} \text{smooth}_{L1}\left(l_i^m - \hat{g}_j^m\right) \tag{5.2.2}$$

$$L_{conf}(\delta,c) = -\sum_{i\in Pos}^{N} \delta_{ij}^{p} log\left(\hat{c}_i^p\right) - \sum_{i\in Neg} log(\hat{c}_i^0) \tag{5.2.3}$$

where N_{match} is the number of matched detecting boxes, $L_{loca}(\delta,p,g)$ refers to the Smooth L_1 [127] localization loss between the ground truth box (g) and the predicted box (p). We represent the bounding box using its center coordinates (cx,cy), width (w) and height (h), and apply regression on the offsets of these parameters. The confidence loss $L_{conf}(\delta,c)$ is the softmax loss for multi-class classification. The weight term γ controls the proportion of localization loss and is set to 1 in our experiments (validated by cross validation). For pre-processing, we follow the conventional SSD to augment data and to determine shapes of the initial detecting boxes, please see [124] for more details.

Testing. In the testing phase, an input image is fed into the trained Enhanced-SSD, and couples of predicted boxes with class confidences are generated as the initial output.

For each unique vehicle, only a single prediction (bounding box and type) is reserved via thresholding using the Non-Maximum Suppression algorithm [128].

One important issue in testing is that the input scale of Enhanced-SSD is 300×300, so directly using the original 4K (3840×2178) traffic video frames is not applicable. To fill the scalability gap, we designed a region-based strategy by employing a sliding window to divide the original video frame into small patches with the size of 512×512. We allow an overlap of 200 pixels horizontally and vertically between patches in order to capture complete vehicles. We then perform detections on each image patch and stitch them back together to the initial scale (see Fig. 5.1).

Allowing overlaps between these patches could obtain complete detections, however, this also increases the numbers of repeated detections (i.e. a single vehicle is detected multiple times in different patches). To solve this issue, in our experiment, we find repeated boxes by evaluating: either their center distances are smaller than a threshold (T_{cd}) or their IoU (intersection over union) scores are above a threshold (T_{iou}). IoU is a popular evaluation criteria in the field of object detection [126, 127], which is used to measure the ratio of overlap between two bounding boxes. In our case, the IoU score of two predicted boxes B_i and B_j is:

$$IoU(B_i, B_j) = \frac{B_i \cap B_j}{B_i \cup B_j} \tag{5.2.4}$$

$IoU = 1$ represents a complete match between two bounding boxes. After we obtained all repeated boxes on a single vehicle, we reserve the one with the maximum scale.

5.2.3 Vehicle Tracking and Counting

As mentioned before, the proposed deep vehicle counting framework (DVCF) is a tracking-by-detection framework. Since we could obtain detection results in the whole video sequence, we simplify the problem of multiple object tracking (MOT) as a data association problem which is aiming to associate detections across different frames in a video sequence. Hence, we found that traditional algorithms are quite appropriate for this objective in term of accuracy and efficiency. Compared with previous works [129–131] which focus on a single variation of objects and low resolution videos, our approach is able to handle multiple types of objects simultaneously in high resolution (4K) videos.

In our approach, only the location coordinates of bounding boxes and corresponding vehicle types are considered for motion estimation and data association. Moreover, long-term occlusion is also ignored as it occurs infrequently in road traffic videos. De-

signing vehicle re-identification algorithms maybe helpful to fight this problem, however, it would introduce significant overhead cost to the whole framework, which potentially hinders its usage in real world applications.

Motion estimation. To estimate motions for each unique vehicle, we represent it using a linear model and propagate its identity into the next frame. And each modeled vehicle is independent of other vehicles and the camera motion. The state of each vehicle is represented using a column vector:

$$V = [x_c, y_c, s, a, c, \hat{x_c}, \hat{y_c}, \hat{s}]^T \tag{5.2.5}$$

where x_c and y_c represents the horizontal and vertical centers of the vehicle bounding box, while s and a refers to its scale and aspect ratio respectively. The vehicle category is denoted as c. $\hat{x_c}, \hat{y_c}, \hat{s}$ represent the estimated values of x_c, y_c and s. Note that the aspect ratio and the vehicle category is treated as constant during the tracking progress. Once a detection is assigned to a vehicle, its bounding box is used to update its state via the Kalman filter algorithm [132].

Data association. For assigning detections to vehicles over time, each vehicle's motion (bounding box coordinates) is estimated by computing its new location in the current frame. We then create a cost matrix M_{cost} by measuring the intersection-over-union (IoU) between each detection and predicted bounding boxes of the existing vehicles. Then our goal is finding an optimal assignment to maximize the numbers of matches in these two sets of bounding boxes. In our experiments, we solve it via the Hungarian algorithm [133]. Again, a threshold Th_{assign} is set to discard assignments with low IoU scores between detections and bounding boxes of existing vehicles.

Life cycle management of tracks. Track maintenance is an essential aspect of vehicle tracking. When vehicles enter or leave the traffic scene, unique trackers need to be created or deleted accordingly over time. In the first frame, a set of trackers are initialized by measuring locations (bounding box coordinates) of existing vehicles. Then in the following frames, the state of assigned trackers are updated using the matched detections, while any unassigned detection may begin a new track. For creating a new tracker, we treat any detection with an overlap (to existing trackers) lower than T_{assign} as an untracked vehicle.

Each track will keep count of a number of consecutive frames, where no new detections are assigned. If this number exceeds a threshold T_{miss}, the target is assumed to have left the field of traffic view and the track is terminated. This avoids overgrowing the number of trackers and reducing tracking errors caused by missing detections over

a long-term period. In our experiments, we empirically set T_{miss} to 10. We do this because trackers are initialized under the assumption that the velocity of moving targets is constant in short-term tracking, which means that it is a poor indicator to model the true dynamic movements in a long period. Besides, early deletion of missing targets improves efficiency.

Vehicle counting. Using the results of tracking, counting vehicles is simple. Each newly created tracker contains an unique ID, the vehicle type, and its bounding box coordinates. The numbers of different types of vehicles could be obtained by inspecting the number of trackers created with the specific type.

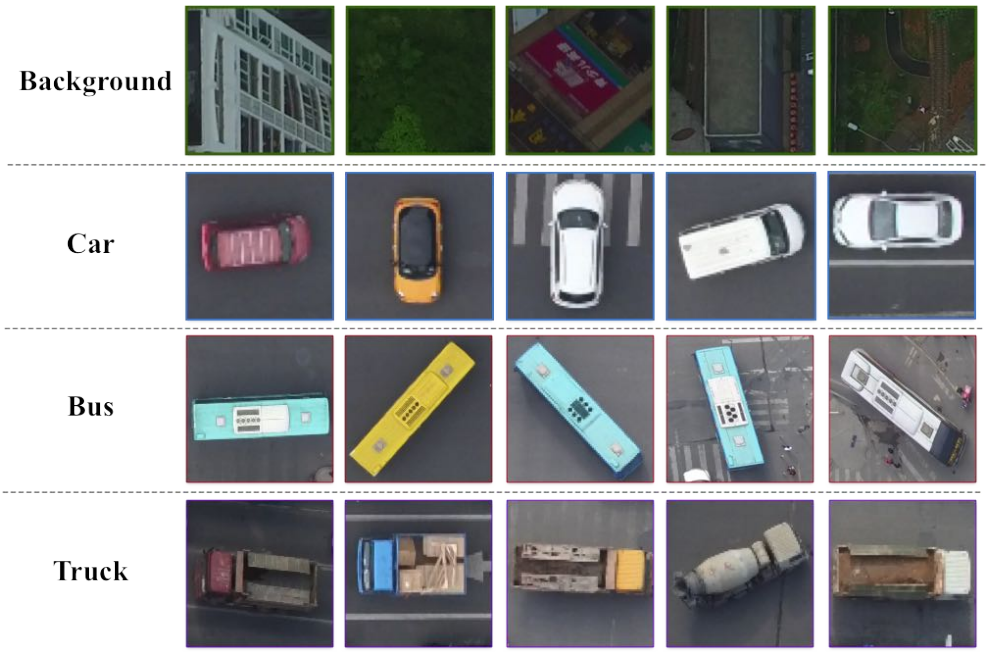


Figure 5.8: The examples of the sub-dataset for image classification. The *car* class contains private cars, taxis, SUVs and small vans, etc. The *bus* type refers to buses and coaches. While the *truck* category include various trucks and large-sized multi-functional vehicles.

5.2.4 Motivation of Deep Learning Based Approach

In this work, vehicle detection is done by feature matching, and good feature representation can help generating good detection results. We notice that deep feature representation which is proposed recently has shown significant superiority over conventional features in multiple fields of computer vision, such as image classification [4], image segmentation [134], object detection and tracking [135]. Hence we are interested that how well the deep features can do compared to conventional features in our experiments. To do this, we design a multi-label image classification test to distinguish

vehicles from the background as well as predicting corresponding vehicle types.

Settings. We train a linear Support Vector Machine (SVM) classifier to identify four categories: car, bus, truck and the background. To build the sub-dataset, we randomly subsample 4,000 images from the training set. Each image appears as a small block which contains at most one object: either a vehicle or just the background (see Fig. 5.8). We randomly select 3,200 images for training and remaining images for testing. For feature representation, we extract 3 types of deep features from different deep neural network models which are popular for image classification, namely AlexNet [136], VGGNet [55] and ResNet [4]. We then compute 3 famous conventional features, namely Scale Invariant Feature Transform (SIFT) [137], Speeded-up robust features (SURF) [138] and Histograms of Oriented Gradients (HOG) [139] features for comparison. For implementation, we use the Caffe [140] toolkit to extract deep features and use OpenCV [141] for conventional feature extraction and SVM based classification. More details of feature architecture are listed in Table 5.5.

Table 5.5: The architecture of features used in the classification experiment.

Feature	Type	Layer	Dimension
AlexNet	deep	fc7	4096
VGGNet	deep	fc7	1024
ResNet	deep	pool5	2048
SIFT	conventional	–	12800
SURF	conventional	–	6400
HOG	conventional	–	861840

We use the classification accuracy (CA) to evaluate the classification performance. CA is defined as either the fraction or the count of correct predictions. In multi-label classification, if the entire set of predicted labels completely match the ground truth labels, the CA would be 1.0, otherwise it is 0.0. Denote p_i as the predicted label of ith testing sample and g_i as the corresponding ground truth label, then the classification accuracy could be formulated as:

$$CA(p, g) = \frac{1}{N} \sum_{i=1}^{N} \varphi(p_i, g_i) \tag{5.2.6}$$

where $\varphi(p_i, g_i)$ is an indicator function which equals to 1 if $p_i = g_i$, otherwise it is 0. N represents the number of testing samples.

Results and discussion. We perform the classification test on the subset containing 4000 images and the quantitative results are shown in Fig. 5.9. The AC scores of deep features are significantly higher (more than 20%) than the conventional features, and features extracted from ResNet is ranked as the first place over the other five types of

features. These are reasonable results since ResNet (101 layers) is much deeper than AlexNet (7 layers) and VGGNet (16 layers), which means it is able to capture more in-depth information in the images, thus yielding relatively higher quality feature representation. This is also one reason why we use ResNet as our base network in the Enhanced-SSD. Since deep features perform overwhelmingly well in the classification test, we only consider deep learning based approaches in the following experiments.

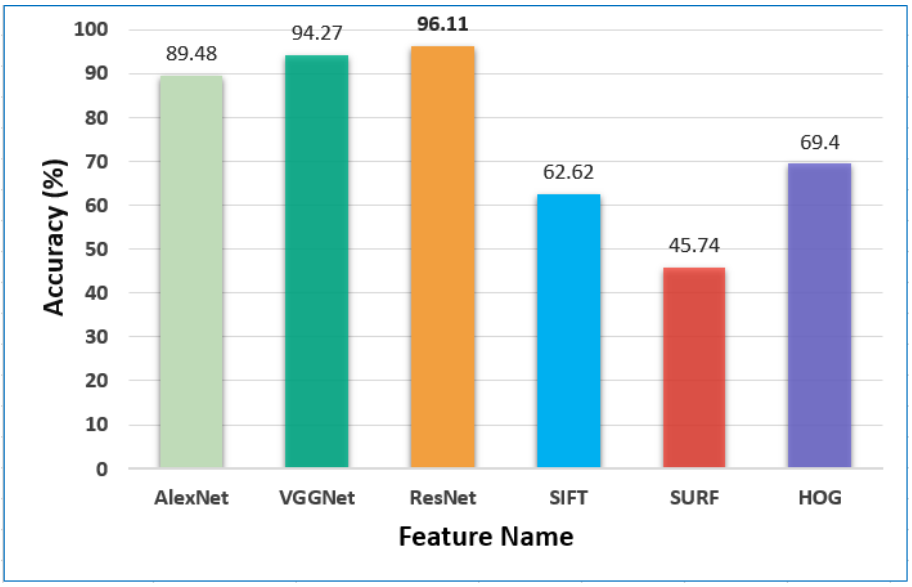


Figure 5.9: The results of the vehicle type classification using different types of features. The highest classification accuracy score is highlighted using the bold font.

5.3 Experiment and Discussion

We thoroughly evaluate our Enhanced-SSD on vehicle counting tasks using the UavCT dataset. We compare our method with three state-of-the-art deep learning based approaches on these two testing set to verify the effectiveness and robustness of the proposed framework.

5.3.1 Counting Vehicles in 4K Images (testing set 1)

In this section, we evaluate the proposed deep vehicle counting framework (DVCF) for vehicle counting in traffic images. Our objective is to count all types of vehicle in all the five images in the testing set 1.

Settings. The training dataset in the UavCT contains 17,168 image patches. We randomly select 85% of these patches for training and the remaining 15% for validation.

We compare our approach (Enhanced-SSD) with three recent deep learning based object detection methods trained on the same dataset, including the vanilla SSD [124], Faster RCNN (FRC) [142] and YOLO [143]. We train the four deep models using Caffe [140] toolkit on a GTX 1080Ti GPU with 11 GB video memory. The setting of main training parameters is shown in Table 5.6. We use smaller batch size to train the Enhanced-SSD to avoid the problem of insufficient video memory. The optimizer is set to stochastic gradient descent (SGD) for better performance in this experiment. We initialize the learning rate as 0.001 and it begins to decrease to the one tenth of current value after 20,000 epochs. The momentum is set to 0.9 by default according to these models.

Table 5.6: Training parameters of the four deep learning based approaches.

Model	SSD	Faster-RCNN	YOLO	Enhanced-SSD
Batch size	32	32	32	6
Optimizer	SGD	SGD	SGD	SGD
Learning rate	0.001	0.001	0.001	0.001
Momentum	0.9	0.9	0.9	0.9
Epoch	12,000	12,000	60,000	12,000

In the testing phase, a testing image is divided into small patches (512×512) with an overlap of 200 pixels, and these patches are then fed into the trained network to detect vehicles. The global result is obtained by aggregating detection results on all patches. We eliminate the repeated bounding boxes on each vehicle by setting center distance threshold T_{cd} as 0.3 and IoU threshold T_{iou} as 0.1 respectively (determined by cross validation).

To make vehicle counting more straightforward, the detection result is visualized by drawing vehicle locations and corresponding types on the input image. Then counting is done naturally by measuring the number of these bounding boxes. We quantitatively evaluate the counting result via correctness (Cor), completeness (Com) and quality (Qua), which are defined in [144]. The true positives (TP) means the number of correctly detected vehicles, false positives (FP) represents the number of invalid detections and false negatives (FN) denotes the number of missed vehicles. Among the three evaluation criteria, quality is most important since it considers both correctness and completeness of detection algorithms.

$$Correctness = \frac{TP}{TP + FP} \tag{5.3.1}$$

$$Completeness = \frac{TP}{TP + FN} \tag{5.3.2}$$

Table 5.7: The quantitative results of vehicle counting in 4K testing image. For each testing image, we show the TP, FP, FN, correctness, completeness and quality. The best values are highlighted by bold fonts. Up arrow means higher is better, and down arrow denotes lower is better.

Method	TP ↑	FP ↓	FN ↓	Correctness ↑	Completeness ↑	Quality ↑
Faster-RCNN	184	58	138	0.760	0.571	0.484
YOLO	158	**1**	174	**0.994**	0.476	0.474
SSD	287	8	45	0.973	0.864	0.844
Enhanced-SSD (ours)	**293**	7	**39**	0.977	**0.883**	**0.864**

$$Quality = \frac{TP}{TP + FP + FN} \tag{5.3.3}$$

Results and discussion. We report the overall counting result on testing set 1 (see Table 5.7) instead of each image because it better describes the overall performance and robustness of these detection algorithms. It is obvious that our Enhanced-SSD achieves the best performance in terms of quality on the testing set 1, followed by the conventional SSD. For correctness, YOLO earns the highest score. However, this method yields too many false negatives (missing vehicles) which leads to very low scores of completeness and quality. Faster-RCNN obtains similar results as YOLO, but with lower scores of correctness.

We also visualize the detection results on testing image 2 as an example to show the overall performance of deep learning based approaches (see Fig. 5.10). Cars, buses and trucks (if any) are automatically marked with light green, orange and light blue bounding boxes respectively. The small images in the middle are patches extracted from the original images which give more ground details for type-specific detection. We have noticed that all the four methods except YOLO generates a small number of false positives. That's no accident because in the training set, only regions containing vehicles are annotated by human annotators, while non-vehicle area (including pure background and empty road) are ignored. A few ignored regions may exhibit very similar appearance with particular vehicles (especially buses and trucks) which would consequently lead to a few wrong detections. YOLO does not give false positives probably because it takes a relatively conservative strategy by setting a high threshold to rejects many potentially correct detections in order to avoid generating wrong detections. This is why it misses a large number of vehicles (see orange boxes of YOLO in Fig. 5.10). Besides, both Enhanced-SSD and the conventional SSD performs very well on this task, but SSD gives more wrong detections and misses more vehicles than our model.

Figure 5.10: The counting results on the testing image 2 using four deep learning approaches. True positives (correctly detected vehicles) are marked using green boxes with corresponding class types and confidence scores, while false positives (erroneous detections) are denoted using red boxes and false negatives (missed vehicles) are surrounded by orange boxes.

The counting results of specific vehicle types are illustrated in Fig. 5.11. We show the counting quality measure here to evaluate the overall performance of these four deep learning approaches. It is obvious that the proposed Enhanced-SSD outperforms the other three methods on all vehicle types (especially on "truck"). This again demonstrates the effectiveness and versatility of our method.

5.3.2 Counting Vehicles in 4K Videos (testing set 2)

Since our deep vehicle counting framework (DVCF) is a tracking-by-detection approach, in this section, we investigate how detection performance will affect vehicle tracking and how well our tracking method could work collaboratively with the deep neural networks. Hence, our objective is to count all types of vehicles in 4K testing videos.

Settings. Given a testing video, we perform frame-by-frame detection using the trained network. This significantly challenges the generalization ability and robustness of the detection algorithms, because in the training set, most vehicles are annotated only once, and it is then required to be detected multiple times (at different locations) across the whole video sequence.

Once detection is done, A set of trackers are created to associate bounding boxes with different vehicles across the whole video sequence. We empirically set the threshold T_{assign} as 0.3 to start a new track, and set T_{miss} as 10 to terminate a track. However,

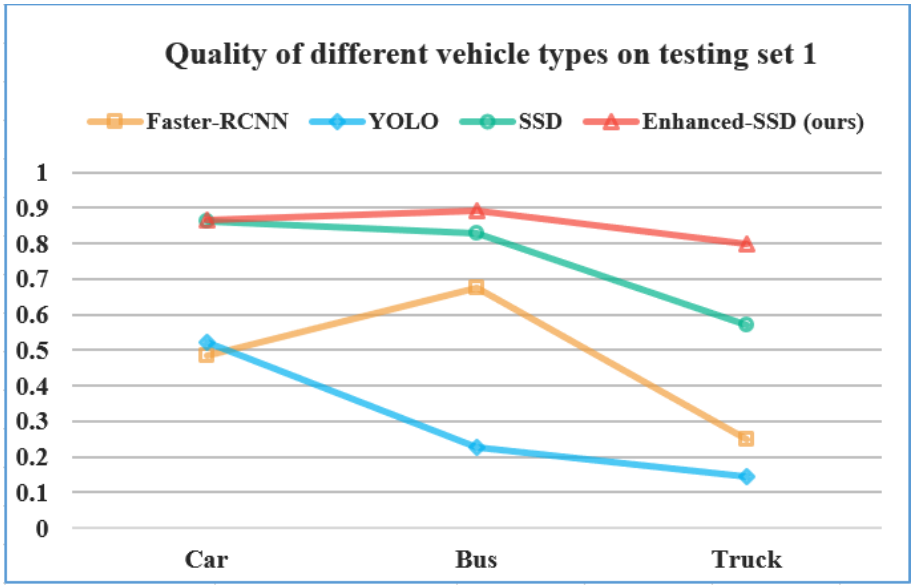


Figure 5.11: The counting results of specific vehicle types on the testing set 1 (images).

repeated tracker may still be created for the same vehicle because the algorithm loses track on it (exceeding the T_{miss}) and treat it as another new identity. To avoid this, we only initialize a set of trackers according to the detection results of the first frame in testing videos, then we only focus on vehicles who enter the traffic view in the testing videos and build new trackers for them. Another possible solution is to design vehicle re-identification algorithms which also consider the visual appearance to associate detections to vehicles, not just bounding boxes. However, it would inevitably introduce extra computational cost which leads to low efficiency. We will work on efficient and robust vehicle re-identification algorithms in our future work.

During the tracking phase, vehicles which are not within the range of roads (e.g. parking lots) are ignored in the counting phase since they contribute nothing to estimate the city traffic density. We manually define the road ranges since testing videos contain large-range and complex traffic scenes. For example, crossroads, T-junctions, slip roads and ring roads are often intertwined with buildings and planting, and they cannot be accurately identified by current automatic road-detection algorithms. Recent work on semantic segmentation may be helpful but it is beyond the scope of this work. For implementation, we run the tracking algorithm on the 5 testing videos using an Intel i7-6700K CPU with 32GB on-board memory.

Because our deep vehicle counting framework (DVCF) separates the vehicle detection and tracking into two different phases, the vehicle counting results should be evaluated separately. However, there's no common approach or standard to evaluate the result of vehicle counting in videos. In this work, we define the Counting Accuracy (CA)

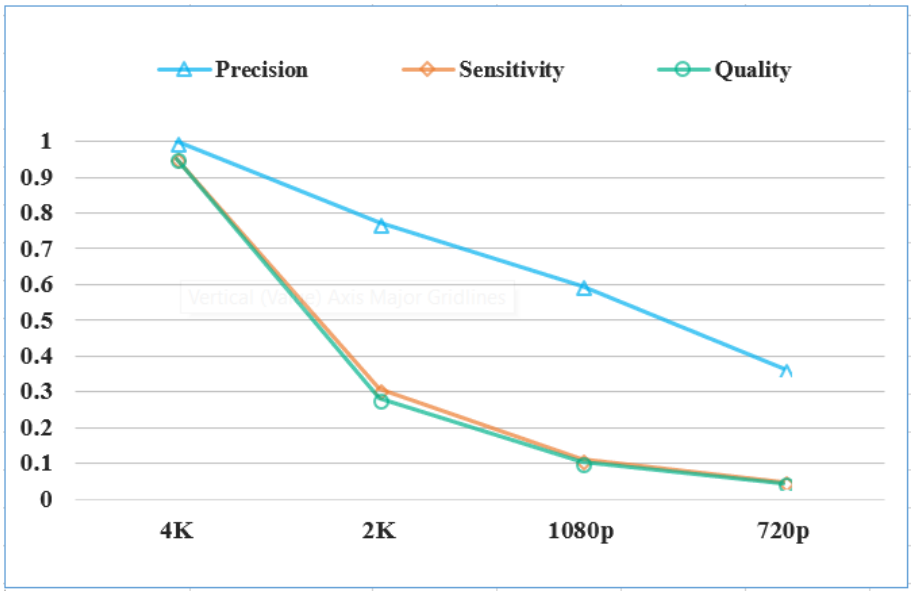


Figure 5.12: Detection performance in different resolutions.

which is based on the "tracking rate" defined in [145] as the evaluation criteria. The CA is formulated as $CA = \frac{N_{sc}}{N_t}$, where N_{sc} refers to the number of successfully counted vehicles, and N_t represents the total number of vehicles in the testing video. Counting of a vehicle is failed if the algorithm does not create a tracker for it during the tracking phase.

Table 5.8: The counting results on the testing set 2 (videos). We list the number of successfully counted vehicle (N_{sc}), the total number of vehicles (N_t) and the counting accuracy. The best values are highlighted using bold fonts.

Method	N_{sc} / N_t	Counting Accuracy (CA)
Faster-RCNN	470 / 727	0.646
YOLO	427 / 727	0.587
SSD	663 / 727	0.912
Enhanced-SSD (ours)	**681 / 727**	**0.937**

Results and discussion. The overall results of vehicle counting on testing set 2 are listed in Table 5.8, from which we can see that the proposed Enhanced-SSD achieves the best CA score on the testing set. This also indicates that good detection performance would lead to promising results in vehicle tracking and counting. Besides, Faster-RCNN and YOLO do not perform well on this task because they miss many vehicles in the detection phase (see Fig. 5.10), which undoubtedly poses negative impacts on the counting results. Similar results are obtained when we measure type-specific vehicle counting (see Fig. 5.13): on each vehicle type, our Enhanced-SSD outperforms other deep learning based approaches, especially for truck detection.

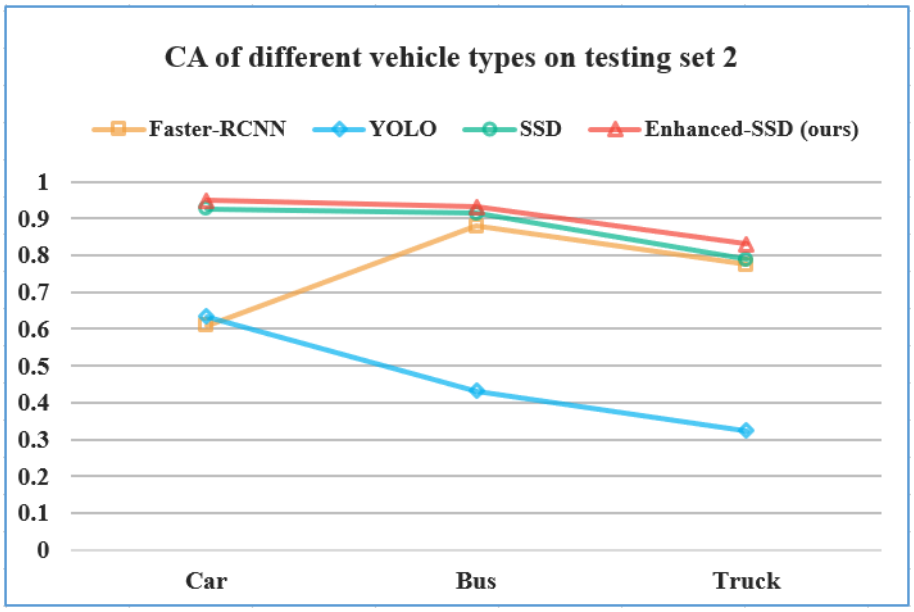


Figure 5.13: Counting results of specific vehicle types on testing set 2 (videos).

We also provide the training time and the testing time/speed in Table 5.9. It can be observed that training and testing of the Faster-RCNN method takes the longest time amongst the four approaches. The YOLO method achieves the lowest training time and the fastest testing speed on the testing set 2, which is due to its shallow network architecture compared with other models. However, its detection performance is significantly worse than other methods. The SSD and Enhanced-SSD (ours) approaches obtained similar results in terms of time consumption, however, the enhanced-SSD performs better than SSD when inspecting the performance (accuracy) of vehicle counting.

Table 5.9: The time consumption of vehicle counting in testing set 1 (images) and testing set 2 (videos). The up arrow means higher is better while the down arrow demotes lower is better. The best values are highlighted using a bold typeface.

Method	Faster-RCNN	YOLO	SSD	Enhanced-SSD
Training time ↓	87h 29m	**57h 31m**	64h 58 m	66h 42m
Testing time on testing set 1 ↓	52.35s	11.15s	**7.95s**	10.85s
Testing speed on testing set 2 ↑	35.3 fps	**72.3 fps**	44.1 fps	46.2 fps

5.3.3 The Impact of the Resolution

Although our data was recorded using ultra high resolution, we were interested to determine if a high resolution really helps detection results. To do this, we created an auxiliary set where we down-sampled all the testing images in the testing set 1 and

testing set 2. By adjusting the resolution of each image, we can determine performance changes of our detection algorithms. More specifically, we resize each original testing image to a resolution of 2K (2560), 1080p (1920 $\times$ 1080), and 720p (1280 $\times$ 720) respectively. We then count all types of vehicles in these low-resolution images/videos.

The results are shown in Table 5.10 and Table 5.11. We can see the counting performance degrades dramatically when image resolution goes down. It makes sense because in a high resolution image (both for training set and the testing set), a vehicle generally takes a few pixels. However, in a 720p image, it only takes one or two pixels. This makes these vehicles (especially small cars) totally unrecognizable (see Fig. 5.14) for vision-based algorithms. Hence, recording data in a high resolution is necessary since it provides enough ground details to help the detection algorithms accurately localizing different types of vehicles.

Figure 5.14: Example of an image patch in different resolution: (a) 4K, (b) 2K, (c) 1080p and (d) 720p.

Table 5.10: Counting results (quality measure) of specific vehicle types on testing set 1 (images) with different resolutions. The best values are highlighted using a bold typeface.

Type / Resolution	720P	1080P	2K	4K
Car	0.056	0.096	0.256	**0.886**
Bus	0.045	0.112	0.288	**0.892**
Truck	0.043	0.104	0.349	**0.80**

Table 5.11: Counting results (counting accuracy measure) of specific vehicle types on testing set 2 (videos) with different resolutions. The best values are highlighted using a bold typeface.

Type / Resolution	720P	1080P	2K	4K
Car	0.074	0.127	0.397	**0.950**
Bus	0.063	0.124	0.361	**0.931**
Truck	0.065	0.111	0.329	**0.831**

We show the testing time/speed of vehicle counting on the testing set 1 (images) and testing set 2 (videos) in different resolutions (see Table 5.12). It can be seen that the

time consumption decreases when the resolution goes down. This is because fewer vehicle are detected and tracked in low resolution images/videos, which requires relatively low computational cost. Anyway, we do not considering using a low resolution in practical applications since the counting performance degrades rapidly when the resolution goes down.

Table 5.12: The time consumption of vehicle counting on the testing set 1 (images) and testing set 2 (videos) in different resolutions. The up arrow means higher is better and the downarrow denotes lower is better. The best values are highlighted using a bold typeface.

Method	720P	1080P	2K	4K
Testing time on set 1 ↓	**2.95s**	7.5s	9.4s	10.85s
Testing speed on set 2 ↑	**125.3fps**	104.1fps	78.2fps	46.2fps

5.4 Summary

In this paper, a UAV and deep learning based vehicle detecting, tracking and counting system has been presented with a series of advantages in the traffic density estimation system. The proposed Deep Vehicle Count Framework (DVCF) effectively and efficiently extracts traffic density data from the high-resolution UAV videos at various geo-location with complex traffic view scopes. To summarize, three significant features of our approach have been demonstrated:

(i): A UAV city traffic video dataset is created to help estimate the real-world city traffic density and is also aiming to motivate research in vision based traffic flow analysis in intricate traffic views.

(ii): In this work, the deep vehicle counting framework (DVCF) is specifically designed for vehicle detection, classification, tracking and counting. However, it is easy to be extended to detect and track many other types of objects (e.g. people, bicycles etc.).

(iii): The proposed deep vehicle counting framework (DVCF) presents a successful attempt to integrate conventional vision based algorithms and deep learning based approaches. To be more specific, we design a deep neural network to accurately localize different types of vehicle in high-resolution images, and then employ traditional tracking approaches to associate the detection results in consecutive video frames in order to maintain the processing speed. Compared with recent methods, our approach considers both the accuracy and the efficiency, while exhibiting good robustness.

For future work, we would develop more effective vehicle detection and tracking al-

gorithms while achieving a high processing speed and robustness. Besides, another working direction is designing a method to automatically select wanted regions (city roads) to further reduce human supervision and improve the overall efficiency.

CHAPTER 6

Vehicle Behavior Recognition for City Road Traffic

In this chapter, we present an all-in-one behavior recognition framework for moving vehicles based on the latest deep learning techniques. Unlike traditional traffic analysis methods which rely on low-resolution videos captured by road cameras, we capture 4K (3840×2178) traffic videos at a busy road intersection of a modern megacity by flying a unmanned aerial vehicle (UAV) during the rush hours. We then manually annotate locations and types of road vehicles. The proposed method consists of the following three steps: (1) vehicle detection and type recognition based on deep neural networks; (2) vehicle tracking by data association and vehicle trajectory modeling; (3) vehicle behavior recognition by nearest neighbor search and by bidirectional long short-term memory network, respectively. This chapter also presents experimental results of the proposed framework in comparison with state-of-the-art approaches on the 4K testing traffic video, which demonstrated the effectiveness and superiority of the proposed method. We include all the necessary details to make the chapter self-contained.

6.1 Related Work

Behavior recognition of moving objects is a hot research topic in multiple fields, especially for surveillance and safety management purposes. In this paper, we focus on city road traffic where the basic road element is the vehicle object.

Studying the behavior of on-road vehicles at road intersections is a vital issue for building intelligent traffic monitoring systems and self-driving techniques. For example, in order to ensure safe driving, drivers need to know if the vehicles in front are going

straight through the intersection or are making left or right turns. However, due to the crossing of multiple roads, crashes generally occur at intersections [146]. In 2015, there were 5295 traffic crashes at four-way intersections with one or more pedestrian fatalities reported in the U.S. [147]. Hence, the intelligent transportation systems need to actively monitor and understanding the road conditions and give warnings of potential crashes or the occurrence of the traffic congestion. In this work, we focus on vehicle behavior recognition at intersections.

Behavior understanding could be treated as the classification of time series data, for example, matching an unknown sequence to some types of learned behaviors [148]. In other words, behavior understanding in traffic monitoring describes the type, location or speed changing of a vehicle in the traffic video sequence (e.g. running, turning, stopping, etc).

6.1.1 Behavior Recognition with Trajectory

Many existing traffic monitoring systems are based on motion trajectory analysis. A motion trajectory is generated by tracking an object frame-by-frame in the video sequence and then linking its locations across the consecutive frames. In recent decades, various approaches to handle trajectory of moving objects analysis based on city road traffic videos have been proposed. In [149], a self-organizing neural network is proposed to learn behavior patterns from the training trajectories, then activities of new vehicles are predicted based on partially observed trajectories. In [150], the vehicle trajectories are modeled by tracking the feature points through the video sequences with a set of customized templates. Then, the behavior understanding is conducted to detect abnormal events: illegal lane changing or stopping, sudden speeding up or slowing down, etc. In [151], the turning behaviors of road vehicles is detected by computing the yaw rate using the observed trajectories. Based on the yaw rate and modified Kalman filtering, the behavior recognition system is capable of effectively identify the turning behavior. In [152], the lane changing information of target vehicles is modeled using the dynamic Bayesian network, and the evaluation is performed using the real-world traffic video data.

6.1.2 Behavior Recognition Without Trajectory

Another way of behavior understanding is to inspect non-trajectory information such as the size, velocity, location, moving orientation, or the flow of traffic objects [153]. The main objective is to detect abnormal events according to these information of a moving

target if the values of these attributes exceed the pre-defined value ranges. In vision based road traffic analysis, speed is estimated by converting the image pixel based distances to the absolute distances by manual geo-location calibration. By extracting velocity data, the traffic monitoring system is able to quickly detect congestion, traffic accident or violation behaviors. For example, Huang et al. use velocity, moving direction and position of vehicles to detect vehicle activities including sudden breaking, lane changing and retrograde driving [154]. Pucher et al. employ video and audio sensors to detect accidents like static vehicles, wrong way driving behaviors and congestion on highways [155]. The authors in [156] adopt flow and velocity to detect the highway congestion, and they found that the traffic jam is caused by the weak traffic controlling, instead of the overload of the capacity.

To summarize, lots of work has been done on road vehicle behavior understanding, however, most published results are rely on small camera networks, which means their cameras only capture a small range of the traffic scene, and they focus on specific vehicle tracking and activity analysis. Besides, many approaches just treat road vehicles as "moving pixels" while the types of road vehicles are often ignored. For example, they cannot process a query like "find all illegally stopped cars in the Southwest road" or "find all trucks queueing for the red lights to cross the road". In our work, we use the UAV to capture a large area of the road traffic such as the whole crossroads, intersections or multi-lanes. Besides, our vehicle detection and tracking algorithms are able to recognize different types of vehicles and maintain these unique identities for all tracked vehicles.

6.2 Methodology

This section elaborates the deep vehicle behavior recognition (DVBR) framework. In the vehicle trajectory extraction part, we first detected road vehicles based on the Retina object detector [157] and then tracked vehicles by associated detections across whole video sequences. Next, we modeled and extracted the vehicle trajectories using the tracking results. In the behavior recognition part, we designed both semi-supervised and supervised approaches to classify vehicle trajectories in order to recognize their behaviors. Figure 6.1 illustrates an overview of our work and its relations to the intelligent transportation system (ITS).

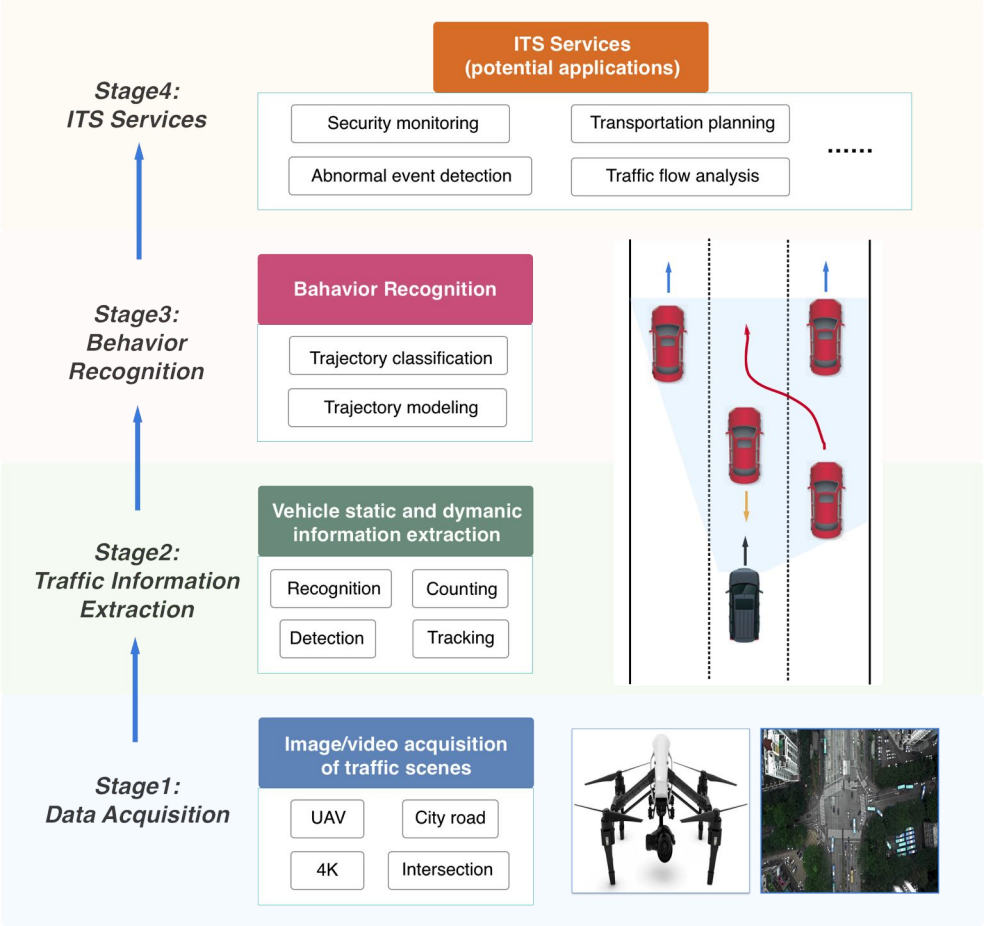


Figure 6.1: An illustration of the hierarchical structure of our work and its relationship to the intelligent transportation system (ITS). In the first stage, we used a UAV to collect high-resolution city traffic videos and, in Stage 2, extracted the static and dynamic information of road vehicles. In the third stage, we modeled and analyzed vehicle trajectory data, and observed vehicle behaviors. In our future work, all these achievements could contribute to the construction of comprehensive ITS services, such as traffic flow analysis, abnormal event detection, and security monitoring.

6.2.1 Vehicle Trajectory Extraction

Vehicle Detection

Network Architecture. We used RetinaNet [157] to detect vehicles in UAV videos. RetinaNet introduces a novel focal loss, which significantly improve the object detection accuracy than the SSD-based method (including Enhanced-SSD), especially for small objects. Besides, RetinaNet addresses the one-stage object detection problem in which the foreground and background classes are imbalanced. RetinaNet consists of a base network for multi-scale feature generation and two subnetworks for object detection

(see Figure 6.2). The base network uses a Feature Pyramid Network (FPN) [3] on top of a feedforward ResNet [4] initially designed for image classification. The FPN can be seen as a standard convolutional network with top–bottom and lateral connections in order to build multi-scale (feature pyramid) feature maps for a single input image. Each layer of the pyramid is responsible for detecting objects at a specific scale.

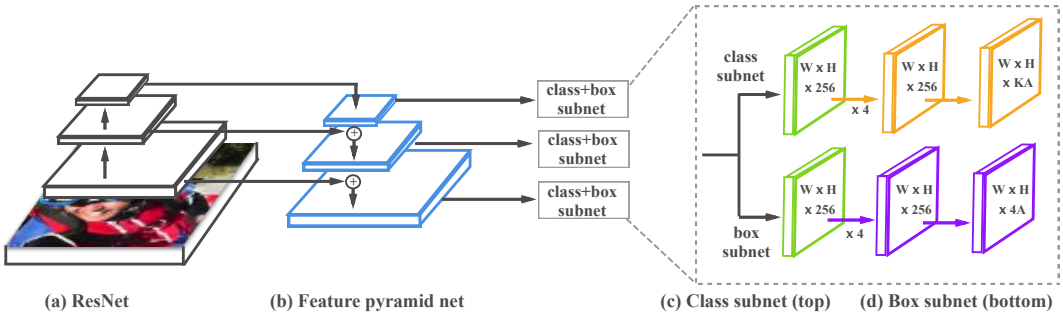


Figure 6.2: The RetinaNet network architecture uses a Feature Pyramid Network (FPN) [3] backbone on top of a feedforward ResNet architecture [4] (**a**) to generate a rich, multi-scale convolutional feature pyramid (**b**). To this backbone, RetinaNet attaches two subnetworks, one for classifying anchor boxes (**c**) and one for regressing from anchor boxes to ground-truth object boxes (**d**).

The other two subnetworks are used for object detection. The first one is the classification subnet which predicts the probability of object existence at each spatial position for each bounding box location and C object categories. The second one is the box regression subnet which regresses the offset from each predicted bounding box to a nearby ground-truth object bounding box.

Training and Testing. We used the training images extracted from the training video to train the vehicle detector, and in the testing phase, a testing image was fed into the trained detector. Couples of predicted boxes with class confidences were generated as the initial output. For each unique vehicle, using a Non-Maximum Suppression algorithm [128], only a single prediction (bounding box and type) was reserved via thresholding.

One important issue in testing is that the original 4K (3840×2178) traffic video frames are too large for the network input. To solve this, we designed a region-based strategy by employing a sliding window to divide the original video frame into small patches with a size of 512×512. We allowed an overlap of 200 pixel horizontally and vertically between patches in order to capture complete vehicles. We then performed detections on each image patch and stitched them back together to the initial scale.

Allowing overlaps between these patches sometimes yielded complete detections, but

this also increased the numbers of repeated detections (i.e., a single vehicle was detected multiple times in different patches). To solve this issue, in our experiment, we found repeated boxes by evaluating them: either their center distances were smaller than a threshold (T_{cd}) or their intersection-over-union (IoU) scores were above a threshold (T_{iou}). IoU is a popular evaluation criterion in the field of object detection [126, 127], and is used to measure the ratio of overlap between two bounding boxes. In our case, the IoU score of two predicted boxes B_i and B_j is

$$IoU(B_i, B_j) = \frac{B_i \cap B_j}{B_i \cup B_j}. \tag{6.2.1}$$

$IoU = 1$ represents a complete match between two bounding boxes. After we obtained all repeated boxes on a single vehicle, we reserved the one with the maximum scale.

Trajectory Modeling and Extraction

The proposed deep vehicle behavior recognition (DVBR) framework follows a tracking-by-detection strategy for trajectory modeling. Since we could obtain detection results in the whole video sequence, we simplified the problem of multiple object tracking (MOT) as a data association problem aiming to associate detections across different frames in a video sequence. In our approach, only the location coordinates of bounding boxes and corresponding vehicle types are considered for motion estimation and data association. Moreover, long-term occlusion is also ignored as it occurs infrequently in road traffic videos.

We adopt the same strategy for vehicle motion estimation, data association and track management, please see Section 5.2.3 for details.

The location of vehicles in video frames can be obtained from the tracking results, which are represented as their tracking ID and corresponding center points (cx and cy). We then extract the trajectory of a vehicle by linking its center points across the consecutive frames in the traffic video. More concretely, we represent the location of i^{th} vehicle in the video sequence as $L_i = [(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), ..., (x_{in}, y_{in})]$, where n refers to the number of frame where this vehicle is tracked. We can easily draw a vehicle's trajectory by linking all its center points stored in L. Compared to other trajectory modeling approaches where vehicle trajectories are only estimated data, our trajectory data are more accurate and reliable because they are obtained through frame-by-frame vehicle detection and tracking.

6.2.2 Vehicle Behavior Recognition

Behavior Recognition Based on Nearest Neighbor Search

We define three types of typical vehicle behavior, such as *go straight*, *right turn*, and *left turn*. We do not consider the *U-turn* because it occurs very rarely in traffic videos. This would lead to very few samples and cannot be processed by recognition algorithms.

In this section, we approach the vehicle behavior recognition by a semi-supervised nearest neighbor search. We first propose a double spectral clustering (DSC) method to cluster vehicle trajectories into three subgroups, and then in each subgroup, we determine its class label by inspecting the majority type of the trajectories in it. In the testing phase, we measure the distance between the testing image and each clustering center using the longest common sub-sequence similarity (LCSS), and assign a class label to it according to the label of the nearest clustering center. This is the basic idea of a nearest neighbor search.

The LCSS was proposed in [158] and is able to effectively handle trajectories with different lengths:

$$D_{LCSS}(F_p, F_q) = 1 - \frac{LCSS(F_p, F_q)}{min(T_p, T_q)} \tag{6.2.2}$$

where $LCSS(F_p, F_q)$ measures the longest overlapping length of the trajectory between F_p, F_q, and $T_p, and\ T_q$ refers to the length of these two trajectories. The LCSS is defined as

$$LCSS(F_p, F_q) = \begin{cases} 0, & T_p = 0 | T_q = 0 \\ 1 + LCSS(F_p^{T_p-1}, F_q^{T_q-1}), & d_E(f_{p,T_p}, f_{q,T_q}) < \varepsilon \\ max(LCSS(F_p^{T_p-1}, F_q^{T_q}), F_p^{T_p}, F_q^{T_q-1}), & otherwise \end{cases} \tag{6.2.3}$$

where ε denotes the threshold of the Euclidean distance, and f_p represents the sample point in the trajectory F_p.

The proposed double spectral clustering (DSC) method proceeds as follows. Given a vehicle trajectory, we first compute its curvature via the least square and polynomial fitting method and take the average values of the first N curvatures as the final result. We then treat the trajectory as a curve (i.e., a vehicle taking turns) if its curvature is larger than the threshold C_{th}, and treat it as a non-curve (i.e., vehicle going straight) otherwise. Note that the threshold C_{th} is empirically set, which means that a curve with a small curvature could be grouped into the non-curve subgroup. This is because the overtaking and lane changing behaviors can be frequently observed on these "go straight" vehicles, which make their trajectories frequently fluctuate. In some cases, if the magnitude of overtaking or lane changing is large enough, the LCSS [158] trajectory

similarity between "going straight" vehicles and "making turn" vehicles would be high. Hence, for non-curves, we firstly compute the similarity of dip angles of vehicle trajectories then perform the spectral clustering to distinguish "going straight" and "making turn" vehicles. In the second stage, the "making turn" vehicles are further clustering into "turning left" and "turning right" groups respectively based on the LCSS similarity. For curve trajectories, LCSS similarity can directly be used to perform spectral clustering since these trajectories only contain vehicles which make turns in the traffic video. Finally we merge the clustering results for curves and non-curves. The whole clustering workflow is illustrated in Figure 6.3.

The similarity between trajectory dig angles is defined as

$$Sim_{\theta(i,j)} = 1 - \frac{\theta_i - \theta_j}{d_{\theta max}}, 0 \leqslant i, j \leqslant n \tag{6.2.4}$$

where θ_i is the dig angle of the i^{th} trajectory and is computed as

$$\theta_i = \frac{arcsin(-\frac{k}{\sqrt{1+k^2}}) \times 180^\circ}{\pi} \tag{6.2.5}$$

$d_{\theta max} = max(|\theta_i - \theta_j|)$, k is the slope of the trajectory, and n is the number of trajectories.

Behavior Recognition by Classification

In this section, we approach the behavior recognition by supervised classification. Different from traditional approaches which incorporating Hidden Markov Modeling and other classification methods such as random forest and k nearest neighbor, we design a novel deep learning model based on Long Short-Term Memory (LSTM) [159]. As a special type of Recurrent Neural Networks (RNNs) [160], LSTM can effectively model the inherent structure of the sequential data and is proved to be powerful in many sequential classification problems [161–163].

Network Structure. The basic structure of LSTM is depicted in Figure 6.4. The LSTM has a memory named “cell” to store the state vector which summarizes the sequence of the past input data. The current state is updated according to the current input, output, and the previous state stored in that “cell”. LSTM has a gate control mechanism that allows the network to “forget” the past state stored in cells or to learn the time stamp to update its state according to the new state information. Denoting C_t as the state of the memory cell at the time step t, then C_t is updated by

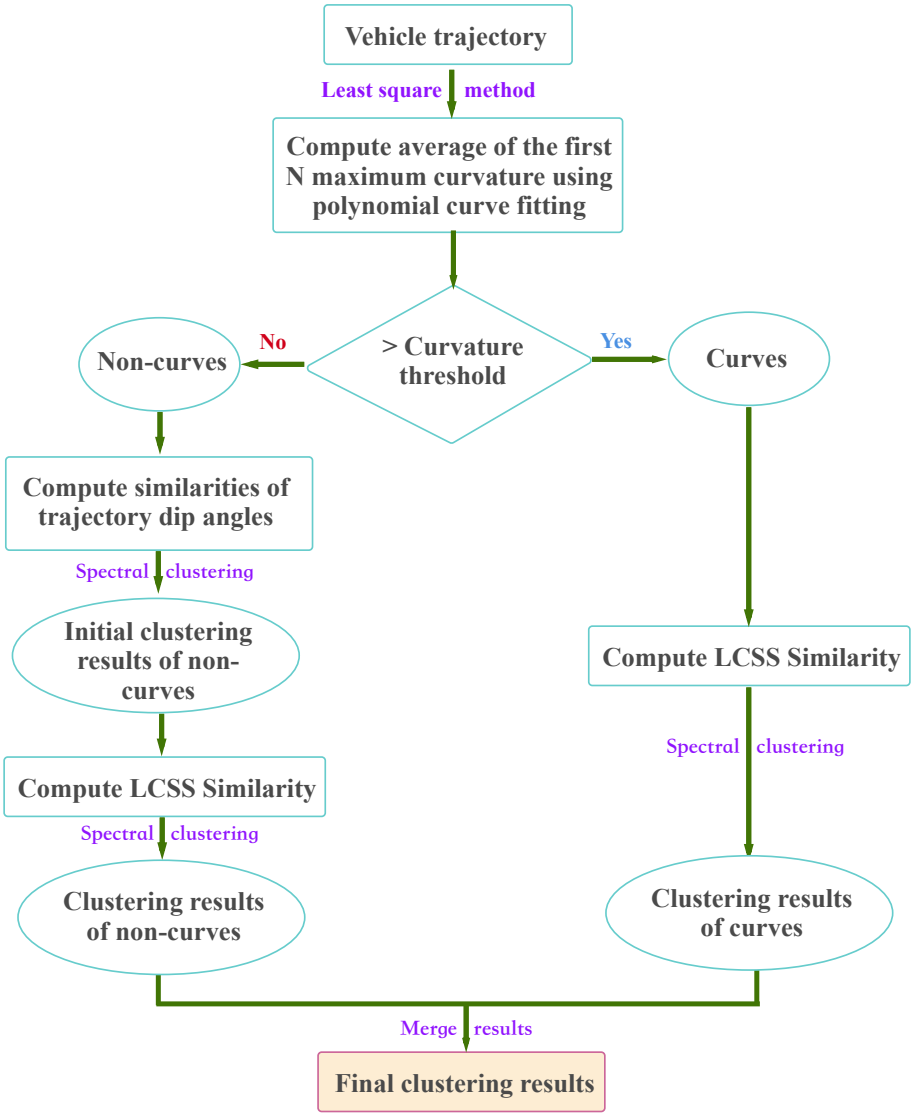


Figure 6.3: The working flow of the proposed double spectral clustering (DSC) method. For curve trajectories, we directly measure their pairwise similarities using the longest common sub-sequence similarity (LCSS) measure. For non-curve trajectories, we first cluster them according to the similarities of their dip angles, then use LCSS again to obtain the final results. Finally, we combine the clustering results on the two types of trajectories.

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
g_t &= tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{6.2.6}
$$

where σ is the sigmoid function, and $x \odot y$ means element-wise product. $W_{xi}, ..., W_{hc}$ are the weight matrices for linear transformation. b_i,b_f,b_o,b_c are the bias vectors. i_t is the input gate vector, f_t is the forget gate vector, o_t is the output gate vector, g_t is state update vector, and h_t is the output hidden state vector.

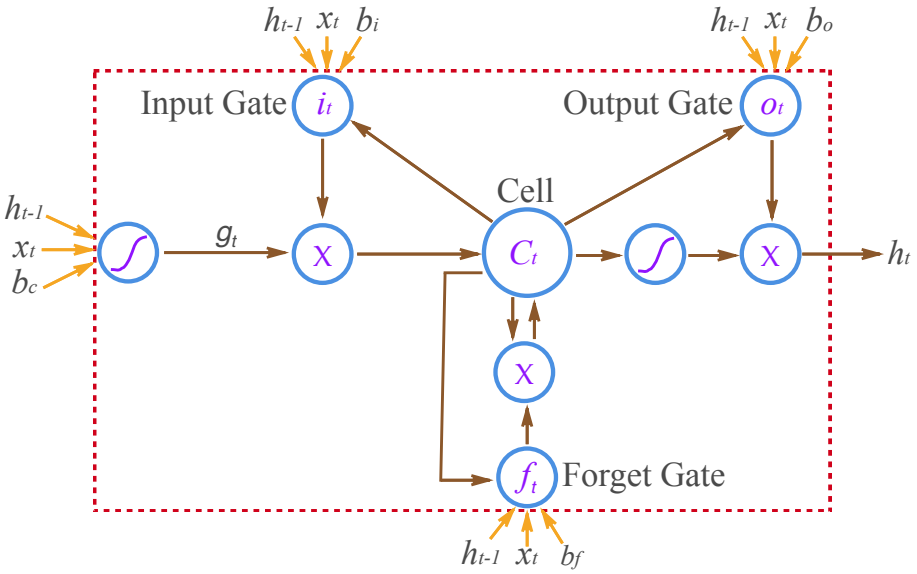


Figure 6.4: The basic Long Short-Term Memory (LSTM) structure.

The input gate i_t and the forget gate f_t can control the information flow from the input to the output, respectively. Note that the behavior of the gate control is learned from data as well. Due to their recurrent nature, even a single layer of LSTM nodes can be considered a "deep" neural network.

For many sequence classification tasks, it is beneficial to have access to future as well as past contexts. However, standard LSTM networks process sequences in temporal order and ignore past contexts. Bidirectional LSTM (BiLSTM) networks extend the standard LSTM networks by introducing a second layer where the hidden-to-hidden connections flow in opposite temporal order. The bidirectional model is therefore able to exploit information both from the past and the future. In our work, we built a trajectory-based bidirectional LSTM model (T-BiLSTM) to classify vehicle trajectories. We merged the output of the two directions by vector concatenation, which generates double the number of outputs to the next layer. In order to extract the information relevant to the class labels, we added an additional output network to the hidden state f_t. We used a fully connected(FC) layer and one softmax layer that contains the linear transformation of h_t followed by the softmax function. We show our network structure in Figure 6.5.

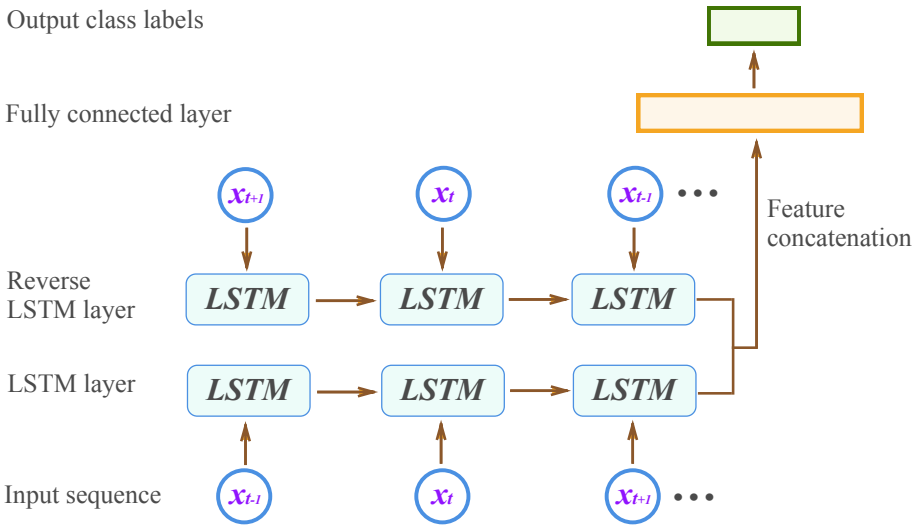


Figure 6.5: The network structure of trajectory classification using bidirectional LSTM (BiLSTM).

Feature representation. The BiLSTM accepts sequential vectors as inputs, so we need to transform the road plane position information stored in trajectories into the sequential features according to the temporal order. General features such as the location coordinates and vehicle speed often contain noisy information since they are sensitive to the motions of vehicles. In our work, we used angular changes to capture the trajectory variations due to its superior robustness compared to other types of features.

Let (x_t, y_t) be the trajectory coordinates of a vehicle at time step t and (x_{t+1}, y_{t+1}) be its coordinates at time step $t+1$. The direction angle θ can then be calculated by

$$\theta = \arctan \frac{(y_{t+1} - y_t)}{(x_{t+1} - x_t)}. \tag{6.2.7}$$

To build the trajectory features, first we resampled the trajectory to a unique length of N trajectory points. More specifically, we computed the overall length of trajectory M, then divided it into $N-1$ segments, each of which has length L. We then rechecked the distance between each adjacent point in M and linearly inserted a new point if their distance was larger than L. Each vehicle trajectory consisted of N points.

Second, we encoded and quantized the trajectory based on angular changes on 16 different directions with an interval of $\pi/8$, as depicted in Figure 6.6a. For example, the sequential angle changes of a vehicle going straight could be encoded as "3-3-3-3-3-3-3-3-3-3-3-2-2-2-2-3-3-3-3-3" (b), and a vehicle turning right could be encoded as "3-3-7-7-3-3-7-3-7-7-7-7-7-15-7-15-7-7-15" (c).

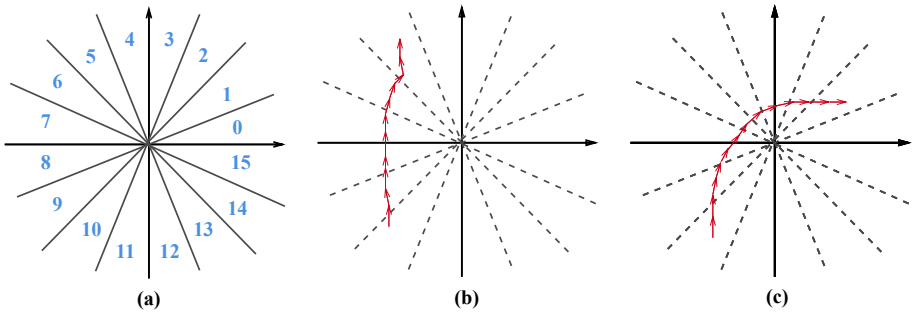


Figure 6.6: (**a**) The encoding and quantization of direction angles. (**b**) The changing angles of a vehicle going straight. (**c**) The changing angles of a vehicle turning right.

At last, we normalized the features values to $[0, 1]$ and used them as the input of the T-BiLSTM model.

Training. We formulated the behavior recognition problem as a multi-class classification problem where one class label was predicted given the sequential features of a testing vehicle's trajectory. To train the T-BiLSTM, we minimized the negative log-likelihood function:

$$L(w) = -\sum_{J}^{t=1}\sum_{m=1}^{M} c_{t,m} \ln z_{t,m} + (1 - c_{t,m}) \ln(1 - z_{t,m}) + \lambda\Phi(w) \tag{6.2.8}$$

where w refers to the parameter of the neural network, J is the number of training samples, $o_{t,m}$ is the m^{th} entry of c_t, $z_{t,m}$ denotes the m^{th} output of the softmax layer associated by class label $c_{t,m}$, and $\Phi(w)$ is the regularization term controlled by the parameter λ.

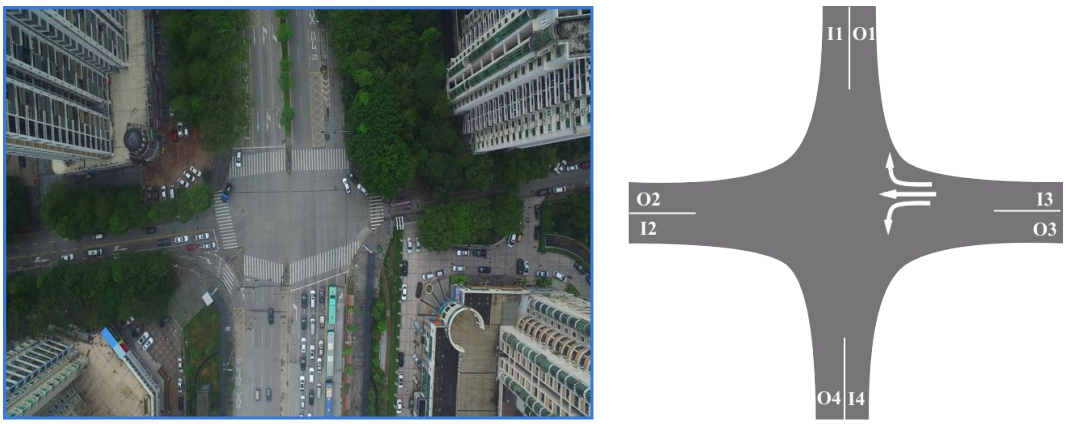
6.3 Experiment and Discussion

To evaluate the proposed framework, we captured a 14 m long traffic video with 4K resolution at a busy road intersections of a modern megacity by flying a UAV during the rush hours. The fps was 30 and the total number of frames was 25,200. The traffic scene at this intersection is shown in Figure 6.7.

To build the training set, we first temporally subsampled the original video frames by a factor of 150. For each frame in the subset, we then divided it into small patches with a uniform size of 512×512. We allowed an overlapping area of 200 pixel vertically and horizontally between these patches to ensure each vehicle appeared as a complete object. We then obtained 3400 training images. Next, we manually annotate vehicles with the following information: (a) bounding box: a rectangle surrounding each ve-

hicle; (b) vehicle type: three general types including car, bus, and truck. This yielded 10,904 annotated vehicles. We used this dataset to train the RetinaNet object detector.

For testing, we collected another short video at the same road intersection but at a different time. The length of the testing video is 2 m and 47 s, with a 3840 × 2178 resolution and 30 fps. The total number of frames was 5010.



(a) **(b)**

Figure 6.7: The traffic scene of the testing video. (**a**) The snapshot of the original video. (**b**) The sketch map of the intersection. The character "I" means "into the camera view" and the "O" means "out of the camera view".

6.3.1 Vehicle Trajectory Extraction

Vehicle Detection

We conducted the vehicle counting experiment to evaluate the effectiveness of the RetinaNet for vehicle detection. More concretely, we counted all types of vehicles in a randomly selected frame from the testing video.

Settings. In the training phase, we randomly selected 85% of these training images for training and the remaining 15% for validation. We compared RetineNet with another three recent deep-learning-based object detection methods: the you-only-look-once version 3 (YOLOv3) [164], the single shot multi-box detector (SSD) [124], and the faster regional convolutional neural network (Faster-RCNN) [142]. We trained the four deep models using Caffe [140] toolkit on a GTX 1080Ti GPU with 11 GB of video memory. The optimizer was set to stochastic gradient descent (SGD) for better performance. We initialized the learning rate at 0.001, and it began to decrease to one-tenth of the current value after 20,000 epochs. The total number of epochs was set to 120,000, and the momentum was set to 0.9 by default according to these models.

In the testing phase, the testing image was first divided into small patches (512 × 512)

with an overlap of 200 pixels, and these patches were then fed into the trained network to detect vehicles. The global result was obtained by aggregating detection results on all patches. We eliminated the repeated bounding boxes on each vehicle by setting the center distance threshold T_{cd} as 0.3 and the IoU threshold T_{iou} as 0.1, respectively (determined by cross validation).

Evaluation. To make vehicle counting more straightforward, the detection result was visualized by drawing vehicle locations and corresponding types on the input image. Counting was done naturally by measuring the number of these bounding boxes. We quantitatively evaluated the counting result via precision, sensitivity, and quality, which are defined in [144]. True positives (TPs) are correctly detected vehicles, false positives (FPs) are invalid detections, and false negatives (FNs) are missed vehicles. Among the three evaluation criteria, quality is most important since it considers both the precision and the sensitivity of detection algorithms.

$$Precision = \frac{TP}{TP + FP} \tag{6.3.1}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{6.3.2}$$

$$Quality = \frac{TP}{TP + FP + FN}. \tag{6.3.3}$$

Result and discussion. We report the counting result on the testing image (see Table 6.1). It can be seen that the RetinaNet achieves the best performance, followed by YOLOv3 and SSD. The Faster-RCNN method yields too many false negatives (missing vehicles), which leads to low sensitivity and quality scores.

Table 6.1: Quantitative results of vehicle counting in 4K testing image. For each testing image, we show the TP, FP, FN, precision, sensitivity, and quality. The best values are highlighted by bold typefaces. The up arrow means that higher is better, and the down arrow means that lower is better. The best values are highlighted using a bold typeface.

Method	**TP** ↑	**FP** ↓	**FN** ↓	**Precision** ↑	**Sensitivity** ↑	**Quality** ↑
Faster-RCNN	49	6	36	0.890	0.576	0.538
SSD	68	2	17	0.971	0.80	0.782
YOLOv3	71	**0**	14	**1.0**	0.835	0.835
RetinaNet	**81**	**0**	**4**	**1.0**	**0.953**	**0.953**

We visualize the results on the testing image (see Figure 6.8). Cars, buses, and trucks (if any) are automatically marked with light green, orange, and light blue bounding

boxes, respectively. The small images in the middle are patches extracted from the original images, which give clearer ground details for type-specific detection. We noticed that SSD and Faster-RCNN generate a small number of false positives. This is to be expected, because in the training set, only regions containing vehicles are annotated by human annotators, while non-vehicle areas (including pure background and empty road) are ignored. A few ignored regions may exhibit very similar appearances with particular vehicles (especially buses and trucks), which would consequently lead to a few wrong detections.

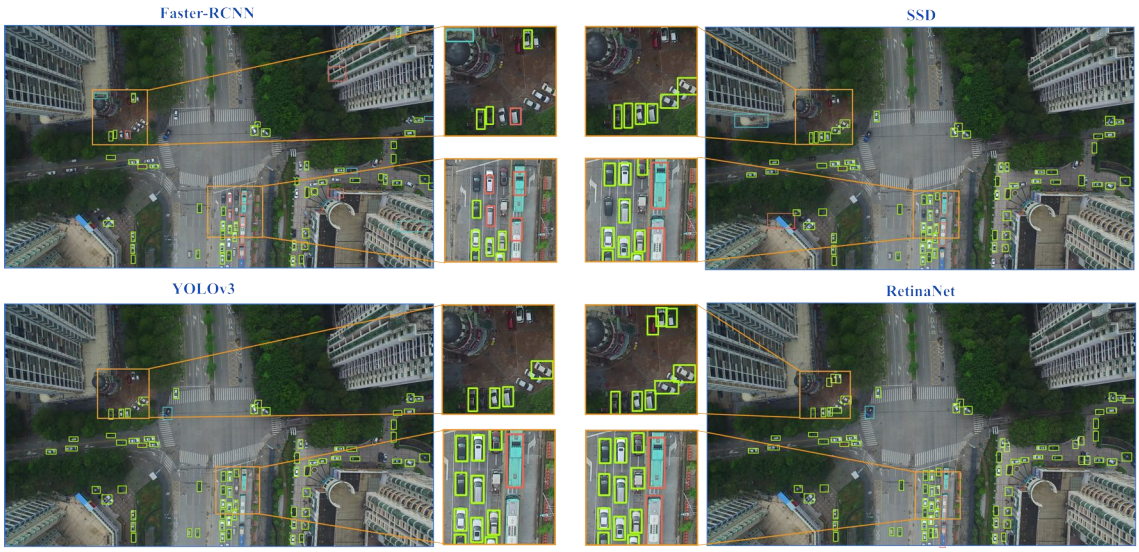


Figure 6.8: The qualitative result of vehicle detection on a testing image.

We also provide the training time and the testing speed (frame per second) in Table 6.2. It can be observed that training of the Faster-RCNN model takes the longest time, but the testing speed is the lowest. The YOLOv3 model achieves the lowest training time and the fastest testing speed, which is due to its shallow network architecture compared with other models. However, its detection performance is worse than the RetinaNet model.

Table 6.2: Training time and testing speed of vehicle detection methods. The up arrow means that higher is better, while the down arrow means that lower is better. The best values are highlighted using a bold typeface.

Model	Faster-RCNN	SSD	YOLOv3	RetinaNet
Training Time ↓	37 h 29 m	24 h 58 m	**18 h 29 m**	20 h 15 m
Testing Speed ↑	15.3	44.1 fps	**72.3** fps	60.2 fps

Vehicle Trajectory Modeling

Settings. We modeled vehicle trajectories according to the tracking results. Given the

testing video, we performed frame-by-frame vehicle detection using the trained network. Once detection was complete, a set of trackers were created to associate bounding boxes with different vehicles across the whole video sequence. We empirically set the threshold T_{assign} as 0.3 to start a new track, and set T_{miss} as 10 to terminate a track.

During the tracking phase, vehicles which were not within the range of roads (e.g., parking lots) were ignored in the counting phase since they contribute nothing to estimate the city traffic density. We manually defined the road ranges, since testing videos contained large-range and complex traffic scenes. For implementation, we ran the tracking algorithm on the testing videos using an Intel i7-6700K CPU with 32 GB on-board memory.

Evaluation. To evaluate the performance of the trajectory modeling approach, we tracked the target vehicle in consecutive frames, and extracted the tracked center point $\hat{C} = (\hat{x}, \hat{y})$ of its bounding box in each frame. For the i^{th}, we computed the trajectory modeling error between the tracked center point and the ground-truth center point (labeled by human annotators) $C_g = (x, y)$ using

$$E_i = \sqrt{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2}. \tag{6.3.4}$$

Based on Equation (6.3.4), we can compute the overall error by adding the modeling error in each frame:

$$E = \sum_{i=1}^{n} \sqrt{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2} \tag{6.3.5}$$

where n is the number of frame being tracked. This error E is quantified using the number of pixels, and it could be easily extend to the real value of centimeters by multiplying it by a factor of 10 (i.e., the ground resolution is 10 cm/pixel).

We tracked all vehicles in the testing video and extracted 238 complete trajectories. The trajectories with unknown types were ignored. We then randomly selected 50 vehicles and computed their trajectory modeling errors using Equation (6.3.4). We took the average value as the modeling error of this frame. The modeling error of the whole testing video was then computed using Equation (6.3.5).

Since our modeling method is based on vehicle tracking, we used three other recent tracking approaches to model the vehicle trajectory and evaluate their performance for comparison, namely, tracking-learning-detection (TLD) [165], tracklet confidence and online discriminative appearance learning (TC-ODAL) [166], and the Markov decision process (MDP) [167]. In other words, our objective was to model the vehicle trajectories using the four methods and then evaluate their performance by computing the modeling error.

Result and discussion. We illustrate the frame-based trajectory modeling error for the first 280 frames of the testing video in Figure 6.9 and report the overall error in Table 6.3. It can be seen that our method outperformed the other three approaches in terms of both frame-based error and the overall error. The TLD method performed worst in this experiment, probably due to the lack of tracking information, since this method does not perform frame-by-frame vehicle detection on the video sequence. Fluctuations of the modeling error could be observed from the results of all four methods, but the error fluctuation range of our method was the smallest compared to the other three ones.

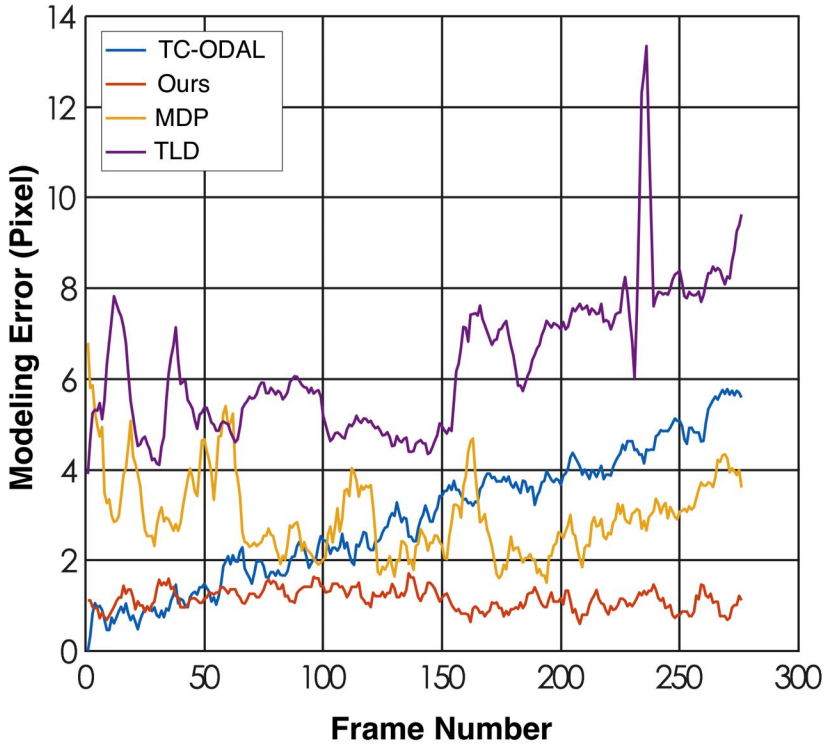


Figure 6.9: The frame-based trajectory modeling errors (lower is better) for four methods. The horizontal axe denotes the frame number, and the vertical axe represents the error value measured in pixels.

Table 6.3: The overall trajectory modeling errors (lower is better) for four methods: tracking-learning- detection (TLD), tracklet confidence and online discriminative appearance learning (TC-ODAL), the Markov decision process (MDP), and ours. The error value is measured in pixels. The best value is highlighted using a bold typeface.

	TLD	**TC-ODAL**	**MDP**	**Ours**
Overall Error (pixel) ↓	1743	796	632	**316**

We also provide the tracking speed (frame per second) of the aforementioned approaches

(see Table 6.4). Since we already have the frame-by-frame detection results, we can treat the tracking problem as the data association problem and do not need to train the algorithm. It can be seen that all the approaches achieve a relatively high tracking speed (i.e., above 55 fps), and the proposed method achieves the highest speed (i.e., 82.5 fps) as well as the lowest tracking error.

Table 6.4: The tracking speed (higher is better) of different methods. The best value is highlighted using a bold typeface.

Method	TC-ODAL	MDP	TLD	Ours
Tracking speed (fps)	78.9	65.1	58.4	**82.5**

6.3.2 Vehicle Behavior Recognition

Behavior Recognition by Nearest Neighbor Search

Settings. For training, we built a training set by extracting 973 complete vehicle trajectories from the training video. Five hundred forty-two of them were with the type "go straight", 204 of them were with the type "turn left", and the remaining 227 were with the type "turn right". For testing, we used 238 trajectories obtained from the testing video.

We applied the proposed double spectral clustering (DSC) on all the 973 trajectories and identified their types (i.e., *go straight*, *right turn*, and *left turn*). Given a testing trajectory, we determined its category based on a nearest neighbor search. We also performed two other clustering methods for comparison, one was a K-Means clustering based on LCSS similarity and the other one was normal spectral clustering based on LCSS similarity.

Evaluation. We employed the normalized accuracy metric considering the large variation in the number of samples in each trajectory type (most of them were of the type "going straight"). We first computed the accuracy within each class and then averaged them over all classes:

$$ACC_i = \frac{\text{Number of correct predictions of class i}}{\text{Number of samples from class i}} \tag{6.3.6}$$

$$ACC = \frac{1}{N}\sum_{N}^{i=1} ACC_i. \tag{6.3.7}$$

Result and discussion. The results are listed in Table 6.5. We noticed that our approach achieved an overall accuracy of 0.899, which is pretty high considering the complex road structure. In addition, the recognition accuracy of vehicle going straight is 0.910, followed by the the accuracy of "left turn" and "right turn", achieving 0.857 and 0.882,

respectively. On each class, our DSC method outperformed the other two methods, which demonstrates the effectiveness of our approach for unsupervised vehicle behavior recognition.

Table 6.5: The results of vehicle trajectory clustering for behavior recognition. For each type of behavior, we show the accuracy on each type and the average value (higher is better). The best values are highlighted using a bold typeface.

Bebavior	LCSS-KMeans	LCSS-Spectral	DSC (Ours)
Go straight	0.813	0.802	**0.910**
Right turn	0.727	0.764	**0.857**
Left turn	0.667	0.769	**0.882**
Average	0.756	0.782	**0.899**

The training time and testing speed of these three approaches are shown in Table 6.6. It can be seen that the LCSS-KMeans runs faster than other methods, this is due to the relatively simpler complexity of the KMeans algorithm (the time complexity of KMeans is $O(n,k)$, while the time complexity of spectral clustering is $O(n^3)$). However, the performance of LCSS-KMeans is worse than the proposed DSC method. In addition, the three methods achieve a very similar testing speed, because they have roughly the same sizes of searching space.

Table 6.6: The training time (in seconds, lower is better) and testing speed (trajectory per second, higher is better) of the three methods. The best values are highlighted using a bold typeface.

Method	LCSS-KMeans	LCSS-Spectral	DSC (Ours)
Training Time (s) ↓	**11.3**	25.8	43.2
Testing Speed (tps) ↑	**62.1**	57.5	55.6

Behavior Recognition by Bidirectional Long Short-Term Memory

In this test, we performed behavior recognition using the proposed T-BiLSTM model.

Settings. We used the same training and testing data as in the previous section. For feature representation, we resampled the length of each trajectory to 256 points, and computed the sequences of angular changes for them. We followed [159] and employed the "back propagation through time" (BPTT) algorithm with a mini-batch size of 32 to train the network. For implementation, we used the Keras [168] deep learning toolkit for Python using Tensorflow [169] as backend.

For comparison, we trained five other models using our training data, namely a Hid-

den Markov Model (HMM) [170], an Activity Hidden Markov Model (A-HMM) [171], a Hidden Markov Model with Support Vector Machine (HMM-SVM) [172], a Hidden Markov Model with Random Forest (HMM-RF) [172], and normal LSTM [173]. The first four algorithms were trained on the CPU (i7-6700K), while the normal LSTM and our method were trained on the GPU (GTX 1080Ti). For evaluation, we used the normalized accuracy metric again.

Results and discussion. Table 6.7 presents the classification accuracy of each method on the testing data. Our approach outperformed all other methods in terms of both single class performance and overall performance. To be more specific, our T-BiLSTM achieved an accuracy of 0.965 on the "go straight" types, probably because the structure of the trajectories under this type are relatively easy to be temporally modeled and recognized. The accuracies decrease on the other two trajectory types due to the increased structural complexity. The LSTM method achieves the second highest accuracy, followed by HMM-RF and A-HMM.

Table 6.7: The results of vehicle trajectory classification for behavior recognition. We show the accuracy on each type and the average value (higher is better). The best values are highlighted using a bold typeface.

Behavior	HMM	A-HMM	HMM-RF	HMM-SVM	LSTM	T-BiLSTM
Go straight	0.825	0.893	0.931	0.832	0.924	**0.965**
Right turn	0.703	0.803	0.856	0.752	0.896	**0.938**
Left turn	0.671	0.733	0.832	0.714	0.875	**0.916**
Average	0.733	0.810	0.873	0.766	0.898	**0.940**

We also show the training time and testing speed of these methods in Table 6.8. It can be seen that the training time of our method on the GPU is the lowest among the six trajectory classification approaches. For fair comparison, we also provide the training time of two deep-learning-based methods (LSTM [173] and ours) on the CPU side, which is much lower than on the GPU side. For testing speed, the HMM achieves the highest speed, but its classification performance is significantly worse than our method.

Table 6.8: The training time (in seconds, lower is better) and the testing speed (trajectory per second) of these methods. The best values are highlighted using a bold typeface.

Method	HMM	A-HMM	HMM-RF	HMM-SVM	LSTM (CPU/GPU)	T-BiLSTM (CPU/GPU)
Training ↓	296.4	331.1	395.7	388.5	1351.5/136.6	526.6/**47.7**
Testing ↑	**65.2**	62.1	59.9	58.1	37.1/40.3	38.2/59.2

6.4 Summary

A deep vehicle behavior recognition framework is proposed in this paper for urban vehicle behavior analysis in UAV videos. The improvements and contributions in this study mainly focus on four aspects: (1) we expand the vehicle behavior analysis area to the whole traffic network at road intersections, not individual road sections; (2) to recognize vehicle behaviors, we propose a nearest neighbor search based model and a deep BiLSTM-based architecture considering both forward and backward dependencies of network-wide traffic data; (3) multiple influential factors for the proposed model are carefully analyzed; (4) we combine deep-learning-based methods and traditional algorithms to effectively balance the speed and accuracy of the proposed framework.

In recent years, the rapid development of autonomous car technologies and driving safety support systems have attracted considerable attention as solutions for preventing car crashes. The implementation of technologies in the intelligent transportation system to assist drivers in recognizing driving behaviors around their own vehicles can be expected to decrease accident rates. Car crashes often occur when traffic participants attempt to change lanes or make turns. Hence, vehicle behavior recognition exhibits significant importance in our daily lives. In our work, we mainly use vehicle trajectory analysis to help recognize three types of vehicle behaviors, but vehicle trajectory analysis also has more applications which we will consider in future work: for example, illegal lane changes, violations of traffic lines, overtaking in prohibited places, and illegal retrograde. We will also implement an artificial-intelligence-based transportation analytical platform and integrate it into the existing intelligent transportation system in order to improve the driving experience and safety of drivers.

CHAPTER 7

Character Recognition via a Semantic Projection Network

In this chapter, we design a method to automatically recognize characters in popular TV series. In contrast to conventional approaches which rely on weak supervision afforded by transcripts, subtitles or character facial data, we formulate the problem as multi-label classification which requires only label-level supervision. We propose a novel semantic projection network consisting of two stacked subnetworks with specially designed constraints. The first subnetwork is a contractive autoencoder which focuses on reconstructing feature activations extracted from a pre-trained single-label CNN. The second subnetwork functions as a region-based multi-label classifier which produces character labels for the input video frame as well as reconstructing the input visual feature from the mapped semantic labels space. Extensive experiments show that the proposed model achieves state-of-the-art performance in comparison with recent approaches on three challenging TV series datasets (the Big Bang Theory, the Defenders and Nirvava in Fire).

7.1 Related Work

Recent years have witnessed increasing studies on character recognition in multimedia resources. Most previous approaches for automated character recognition are aided by transcripts aligned with the subtitles to provide strong supervision for the task [174–176], however, transcripts and subtitles for all films or the entire seasons of a TV series are tough to find from the IMDB or social media sites. Besides, they often come in various styles and formats, results in much work on pre-processing and re-formatting. In contrast, only labeling the existence of every targeted characters in video frames

are easy to achieve and does not introduces ambiguous information contained in transcripts and subtitles.

Some other transcript-free approaches tend to use face recognition algorithms to help capture the facial characteristics of actors in videos [177–182], and some also go beyond the frontal faces to also consider the profiles such as hair and poses [183–186]. Facial knowledge can benefit the character recognition algorithms, however, this requires extensive and tedious facial data annotation including the location and class labels of faces in order to train the algorithm. Moreover, the variations of lighting, posing and background also significantly challenge the face recognition algorithms, and would lead to dissatisfied performance if the training data is limited. In this work, we only use the label-level supervision to make predictions based on both holistic and regional information in the videos.

As the supervision is available from character labels, the problem of character recognition can be cast as the multi-label classification [187]. It is a long-standing problem and has been studied from multiple angles. One common method is the problem transformation. For example, Li et al. [188] transform the multi-label problem into a single-label problem by designing a binary coding strategy, while Nam et al. [189] treat each label independently and train a set of classifier to predict each label.

Many recent approaches tackle the multi-label classification problem using convolutional neural networks (CNNs). CNN has achieved promising results in many single label dataset [4, 55, 136] , such as the CIFAR10/100 [190] and ImageNet [2]. Many researchers have therefore adopted the CNN-based techniques to address multi-label classification problems. Wu et al. [191] design a weakly weighted pairwise ranking loss to tackle weakly labeled images and a triplet similarity loss to handle unlabeled images. Wang et al. [192] add a recurrent neural network (RNN) to the CNN backbone so as to predict multiple labels sequentially. Wei et al. [82] extend the single-label CNN to multi-label CNN which predicts all the labels at one time. All these methods can be formulated as a one-off mapping function which projects the visual (image) space to the semantic (label) space, however, such kind of projection strategy often suffers from the problem of imbalance data. This is, if some classes have very limited training samples, then the samples from these classes are likely to be classified as other classes which have many more training samples. In our work, we propose to solve this problem by encouraging mutual projection between the visual space and semantic space in order to learn more robust feature representation.

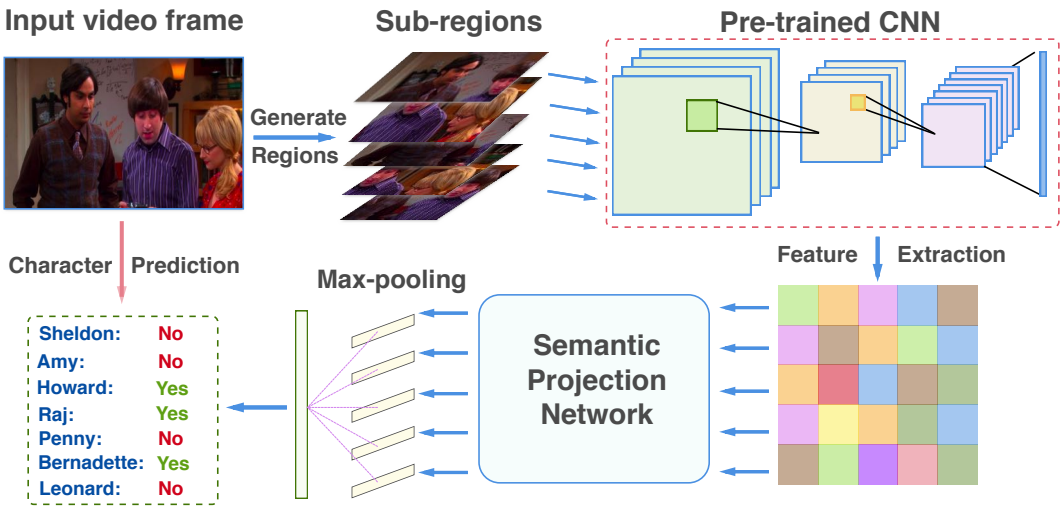


Figure 7.1: The workflow of the proposed character recognition framework. Given an input video frame, we first generate a set of region proposals and then extract their visual features using a pre-trained single-label CNN. Next we feed the features into the trained semantic projection network and predict the existence of each targeted character. To obtain the final predicted results, we employ the max-pooling strategy over all the region proposals to aggregate their predictions.

7.2 Methodology

In our work, we do not rely on transcript, subtitles or facial data to help recognize different characters, instead, we formulate this problem as the multi-label classification, where each character is treated as an independent label. Given a testing video frame, we first pass it into a pre-trained single-label convolutional neural network (CNN) to extract the visual feature activations from the last fully-connected layer, then take these features as the input of the trained semantic projection network (SPNet). The SPNet then produces the final character labels following a max-pooling strategy (see Fig. 7.1).

As depicted in Fig. 7.2, The semantic projection network (SPNet) consists of two stacked autoencoders with special constraints. Given the original visual features as the input (v), the first autoencoder (visual reconstruction subnet, VRNet) is used to learn robust visual embeddings (v_h) as well as reconstructing visual features from v_h. In the second autoencoder (semantic mapping subnet, SMNet), the learned visual embeddings (v_h) is employed to predict the character labels (s) while reconstructing the input v_h from s.

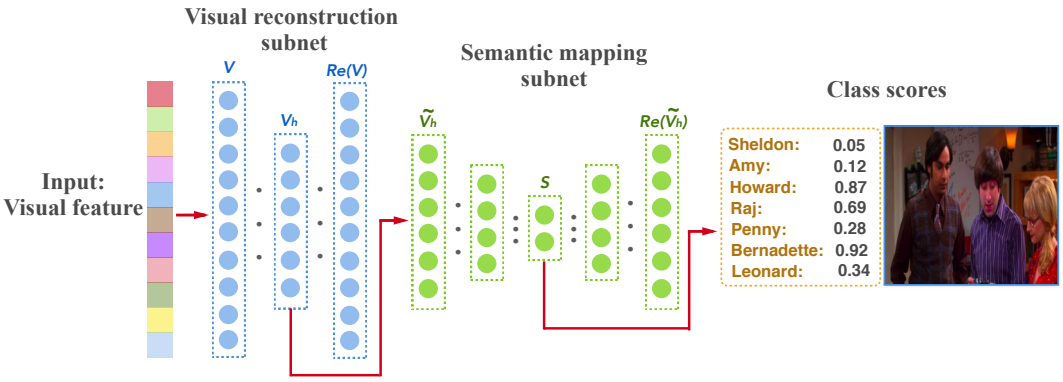


Figure 7.2: The architecture of the semantic projection network (SPNet). The input visual features are refined by the visual reconstruction subnet and then mapped to the semantic space (represented by n class labels) in the semantic mapping subnet. We encourage the mapping from semantic space to visual space so as to learn robust semantic embeddings for the input visual feature.

7.2.1 Visual Reconstruction Subnetwork

The first subnetwork is the visual reconstruction subnetwork (VRNet) which aims to learn robust visual embeddings from the visual features. We start by introducing the formulation of the linear autoencoder and then extend it to the proposed one. An autoencoder is a feed-forward neural network with the same input vector and the target output. In its simplest form, an autoencoder is linear and only one hidden layer is placed between the encoder and decoder layers, compressing the input data into a low-dimensional representation. Formally, given an input data matrix $D \in \mathbb{R}^{n\times M}$ consisting of M feature vectors with the feature dimension of n, the encoder projects it into a k-dimensional ($k < n$) latent space with an encoding matrix $W_{en} \in \mathbb{R}^{k\times n}$, resulting in a latent representation $H \in \mathbb{R}^{k\times M}$. The latent representation is then projected back to the input feature space via the decoding matrix $W_{de} \in \mathbb{R}^{n\times k}$ and becomes to the reconstructed data matrix $\hat{D} \in \mathbb{R}^{n\times M}$. For the learning objective, we minimize the reconstruction error, i.e. D and $\hat{D}$ should be as similar as possible. Hence, the objective function could be formulated as:

$$\min(W_{en}, W_{de}) = \|D - W_{de}W_{en}D\|_F^2 \tag{7.2.1}$$

The VRNet can be seen as a basic linear autoencoder with the contractive loss [193]. By adding such loss, it is aiming to learn more robust visual embeddings for the images of same class. To formulate, the VRNet projects the input visual feature vector v to the latent representation v_h, and then seeks to reconstruct v from v_h. Denoting recon-

structed vector as $\hat{v}$, the model parameters are learned by minimizing the regularized reconstruction error:

$$L_v = \frac{1}{N}\sum_{1}^{N} \|v - \hat{v}\|^2 + \alpha \|J(v)\|_F^2 \tag{7.2.2}$$

where N is the number of training samples, and the $J(\cdot)$ is the Jacobian matrix [193] and is computed by:

$$\|J(v)\|_F^2 = \sum_{ij}(\frac{\partial v_h(j)}{\partial v(i)}) \tag{7.2.3}$$

where ∂ denotes the differential operation, $v(i)$ means the i^{th} input visual feature vector, $v_h(j)$ denotes the j^{th} hidden vector. The Jacobian matrix contains partial derivatives of the feature activations of neurons with respect to the input values, and so it is possible to inspect the impact of variations of the activation values and penalizing the representation accordingly. The α is a hyper-parameter controlling the proportion of the contractive loss during training.

7.2.2 Semantic Mapping Subnetwork

The second subnetwork is the semantic mapping subnetwork (SMNet) which is a multi-layer autoencoder with semantic constrains. In the SMNet, the encoder projects the learned visual embeddings to the semantic label space, similar to a conventional multi-label classification model. However, we also consider the semantic label space as an input to a decode in order to reconstruct the input original visual feature representation. This extra reconstruction task introduces a new constraint to the learning of the projection function from the semantic space to the visual space.

To formulate, the input of the SMNet is the visual feature activations v_h extracted from the hidden layer of the trained VRNet. The objective of SMNet is to first encode v_h to the latent semantic label space s and then decode it to the input visual feature space $\hat{v_h}$. The number of hidden neural in the semantic label space equals to the number of class labels. Hence, we wish to minimize the visual reconstruction error combined with the multi-label classification error:

$$L_s = \beta\frac{1}{N}\sum_{1}^{N} \|v_h - \hat{v_h})\|^2 + \|\Phi(s)\|_F^2 \tag{7.2.4}$$

where N is the number of training samples and the parameter β controls the proportion of visual reconstruction loss in L_s. $\Phi(\cdot)$ denotes the multi-label soft margin error [194]:

$$\|\Phi(\hat{s}, s)\|_F^2 = -\sum_{i=1}^{N}\left[s_i \log \frac{e^{\hat{s_i}}}{1+e^{\hat{s_i}}} + (1-s_i)\log \frac{1}{1+e^{\hat{s_i}}}\right] \tag{7.2.5}$$

where $\hat{s_i}$ and s_i is the predicted label vector and ground-truth label vector for i^{th} testing sample respectively. Combining Eq. (7.2.2) and Eq. (7.2.4), we have:

$$L_{total} = L_v + L_s \tag{7.2.6}$$

To minimize L_{total}, we train the two subnetworks sequentially. In the first stage, we train the VRNet by minimizing L_v and then freeze the network parameters. In the second stage, we extract features from hidden layer (v_h) of the trained VRNet and use them as the input of the SMNet, then train the subnetwork by minimizing L_s.

7.2.3 Region-based Multi-label Classification

In our work, the predicted class scores can be directly obtained from the hidden layer of the SMNet (as depicted in Fig. 7.2) because we force its content to be as similar as possible to the ground-truth label annotations during training.

Considering that each video frame may contain multiple labels and some labels may only apply to sub-regions, we add a region-based strategy to predict the character labels. More specifically, we first employ multi-scale combinatorial grouping (MCG) [**?**] method to extract hundreds of sub-regions from the given image, we then adopt the normalized cut algorithm [195] to cluster all region proposals into c clusters based on the IoU (Intersection-over-Union) affinity matrix. In each cluster, we select k region proposals with the largest predictive scores defined by the MCG approach and feed them into the trained SPNet. We also add the original image to the proposal group, and obtain $ck + 1$ region proposals for that image. The final prediction result is then obtained by max-pooling the predicted output of all the proposals (as depicted in Fig. 7.1). With max-pooling, large predicted class scores corresponding to targeted characters will be reserved, while the values from the noisy proposals will be ignored.

7.3 Experiment and Discussion

We evaluate the proposed SPNet on three challenging TV series, namely *the Big Bang Theory, (BBT)* from the US, *the Defenders, (TD)* from the US and *Nirvana in fire, (NIF)* from China. We also examine the importance of region-based strategy for character recognition.

Datasets. Consider the temporal redundancy of videos, for each TV series, we first sample five consecutive episodes every 5 frames and annotate these sampled frames with character labels. We then use the first four annotated episodes for training and the last one for testing. Note that such splitting strategy is very challenging since the lighting, scenario, and costumes of characters might be totally different between the training samples and testing samples. More details about these datasets are shown in Table 7.1.

Table 7.1: Details of the three TV series video datasets.

Name	The Big Bang Theory	The Defenders	Nirvana in fire
Season No.	7	1	1
Training episodes No.	1∼4	1∼4	2∼5
Testing episodes No.	5	5	6
No. of training samples	54,985	56,616	53,349
No. of testing samples	5,481	14,661	13,325

Visual features. In our experiments, we use the ResNet (pre-trained on ImageNet) [4] features which is the 2048D activation of the final fully-connected layer. The input video frame is first resized to 224×224 and then fed into the ResNet model to extracted visual features. For fair comparison with published results, we uniformly use the ResNet features as the input of the compared methods.

Parameter settings. The length of layers in the VRNet (first subnetwork of SPNet) is $2048 \rightarrow 1024 \rightarrow 2048$, and the length of layers in the SMNet (second subnetwork of SPNet) is $1024 \rightarrow 512 \rightarrow n \rightarrow 512 \rightarrow 1024$, where n denotes the number of character labels.

Besides, the SPNet has two hyper-parameters: α in (see Eq. 7.2.2) and β (see Eq. 7.2.4). They are trade-off parameters for different loss components. As in [196], their values are set by class-wise cross-validation using the training data.

We train the two subnetworks in the SPNet separately. We employ the Adam algorithm [197] as the optimizer, the momentum is set to 0.9, the batch size is set to 128, the initial learning rate is set to 0.0001. We decrease the learning rate to one-tenth of its current value every 10 epochs. We execute 25 epochs to train the VRNet (first subnetwork) and 30 epochs to train the SMNet (second subnetwork).

Evaluation metric. We use the f1 scores to throughly evaluate the performance of the proposed model. This score can be interpreted as a weighted average of the precision and recall, where an f1 score reaches its best value at 1 and worst value at 0. The formula for the f1 score is:

$$f1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{7.3.1}$$

Competitors. We compare our method (SPNet-RP) with several recent multi-label classification approaches as follows.

CNN-SVM [198] and ML-KNN [199] serve as baseline methods which uses support vector machines and and k-nearest neighbor search to tackle the multi-label classification problem. Visual features are feature activations extracted from the pre-trained CNN.

HCP [200] is a novel CNN infrastructure, named hypotheses CNN pooling. In HCP, object segments hypotheses are taken as the input of the shared CNN, and the final predictions are obtained by max-pooling the results on all these hypotheses.

DeepBE [188] transforms the multi-label classification problem to single-label classification using the specially designed binary coding scheme. The transformed data can be learned by powerful CNNs which are initially designed for single-label classification.

LGC [201] is a flexible deep CNN framework for multi-label classification. LGC consists of a local level multi-label classifier which takes object segment hypotheses as inputs to a local CNN, and a global CNN that is trained by multi-label images to directly predict the multiple labels from the input. The predictions of local and global level classifiers are fused together to obtain the final predicted results.

Besides, we also predict the character labels without the region-proposals in the SPNet to spot the differences.

Implementations. We implementate all the models using the Python programing language with the support of the PyTorch[202] deep learning toolkit. Codes were running on the GTX1080ti GPU with 11GB display memory.

Results and discussion.

The results of character recognition on the three TV series are shown in Table 7.2, 7.3 and 7.4 respectively. From results, we can see that the proposed models (SPNet and SPNet-RP) outperform all the recent approaches and achieve a significant improvement over those two baseline methods (over ML-KNN [199] and CNN-SVM [198]). Only one observed exception is that the HCP [200] method achieves the best f1 score (0.791) on the *Bernadette* character in the *BBT* dataset.

We also notice that the best f1 scores obtained on the *BBT* and *TD* datasets (0.627 and 0.658 respectively) are lower than the one obtained on the *NIF* (0.788). This is because the video content in *NIF* contains many close-up views of individual characters which

provide more detailed information in the corresponding visual features.

Besides, it can bee seen that the SPNet with region proposals (SPNet-RP) achieved very similar results as the vanilla SPNet on the *BBT* and *TD* datasets, however, the former exhibits significantly better performance than the latter on the *NIF* dataset. This is because its video content contains many big scenes like palaces and the battleground, in which the characters only appear in small regions. This demonstrates the effectiveness of region-based strategy for character recognition.

Table 7.2: The result of character recognition on *the Big Bang Theory, (BBT)*. We show f1 scores computed on each individual character and the average of them. The best values are highlighted using bold fonts.

Method	ML-KNN	CNN-SVM	HCP	DeepBE	LGC	SPNet	SPNet-RP
Sheldon	0.663	0.697	0.641	0.767	0.665	0.778	**0.795**
Amy	0.303	0.311	0.273	0.390	0.474	**0.587**	0.546
Howard	0.535	0.437	0.611	0.466	0.520	**0.660**	0.618
Raj	0.364	0.356	0.407	0.407	0.452	0.663	**0.670**
Penny	0.463	0.571	0.497	0.348	0.411	**0.628**	0.540
Bernadette	0.571	0.544	**0.791**	0.586	0.557	0.506	0.553
Leonard	0.348	0.380	0.440	0.549	0.403	0.566	**0.635**
Average	0.464	0.471	0.523	0.502	0.497	**0.627**	0.622

Table 7.3: The result of character recognition on *the Defenders, (TD)*. We show f1 scores computed on each individual character and the average of them. The best values are highlighted using bold fonts.

Method	ML-KNN	CNN-SVM	HCP	DeepBE	LGC	SPNet	SPNet-RP
Dare Devil	0.511	0.614	0.633	0.609	0.620	0.629	**0.716**
Jessica Jones	0.364	0.378	0.427	0.529	0.441	**0.531**	0.497
Luke Cage	0.486	0.476	0.561	0.531	0.570	**0.596**	0.591
Iron Fist	0.543	0.522	0.592	0.520	0.613	0.573	**0.649**
Alexandra	0.461	0.611	0.712	0.771	0.831	**0.858**	0.836
Average	0.473	0.520	0.585	0.592	0.615	0.637	**0.658**

7.4 Summary

In this work we propose a novel semantic projection network (SPNet) to address the problem of character recognition in TV series. The SPNet consists of two stacked sub-networks with specially designed constraints for different purposes. More specifically,

Table 7.4: The result of character recognition on *Nirvana in fire, (NIF)*. We show f1 scores computed on each individual character and the average of them. The best values are highlighted using bold fonts.

Method	ML-KNN	CNN-SVM	HCP	DeepBE	LGC	SPNet	SPNet-RP
Changsu Mei	0.759	0.669	0.734	0.770	0.813	**0.834**	0.831
Jingyan Xiao	0.604	0.622	0.591	0.667	0.469	0.604	**0.725**
Nihuang Mu	0.596	0.579	0.579	0.617	0.585	0.683	**0.686**
Jinghuan Xiao	0.761	0.753	0.653	0.686	0.625	0.659	**0.839**
Emperor of Liang	0.610	0.605	0.705	0.751	0.773	0.775	**0.857**
Average	0.666	0.646	0.652	0.698	0.653	0.711	**0.788**

the first subnetwork is a contractive autoencoder which focuses on reconstructing visual feature activations extracted from a pre-trained CNN, while the second subnetwork functions as a multi-label classifier with additional constraints which require to reconstruct input visual features from the projected semantic space. Considering that some character labels may only apply to the subregions of the video frames, we introduce the region-based strategy to further improve the classification performance. Experimental results on three challenging TV series show that the proposed method achieves state-of-the-art performance.

CHAPTER 8

Concluding Remarks

8.1 Summary of Thesis and Contribution

In this thesis, we develop models and techniques for visual content understanding by connecting vision and language modalities. More specifically, we build both unsupervised and supervised learning models to bridge the gap of vision and natural language, and train the model parameters on datasets of images/videos with corresponding natural language annotations.

In Chapter 2, we provide necessary background knowledge of machine learning, including clustering, dimensionality reduction, deep neural networks and describe popular network architectures for processing images/videos. Besides, we also introduce some typical natural language processing techniques such as the part-of-speech tagging and the word vector representation.

In Chapter 3, we build a data-driven visual theme discovery framework which is able to automatically reorganize images with tag-based annotations into a set of compact and comprehensive visual themes. We do it by measuring the visual similarity and semantic similarity respectively for each pair of tags, and then apply a spectrum clustering technique to divide the tags and their corresponding images into different subgroups (visual themes). We conduct user study and common computer vision tasks to validate the effectiveness of the proposed framework and have obtained promising results.

In Chapter 4, we attempt to mine inherent semantic information from user videos by predicting their semantic attributes in each frame, and then inspect the impact of these semantic attributes in assisting the the visual features to effectively describe the visual content in videos. In particular, we first discover a set of semantic attribute from a joint image and text corpora, and then use these attributes as a supplementary compo-

nent of visual feature to summarize user videos. In the experiment, our unsupervised approach outperforms several supervised methods using the maximum-based evaluation, and achieve the second place using the average-based evaluation.

In Chapter 5, we focus on vehicle detection, tracking and counting in high resolution (4K) UAV traffic videos, and develop a deep neural network to accurately localize road vehicles as well as giving them semantic labels (i.e. car, bus and truck). We then design a fast online tracking technique to associate the vehicle detection results across consecutive video frames. We finally conduct the vehicle counting experiments on both 4K images and videos, and have obtained very promising results.

In Chapter 6, we extend our work in the previous chapter and perform vehicle behavior recognition in traffic videos based on trajectory modeling. We first model the vehicle trajectory by connecting the center points of each track vehicle in consecutive video frames, then design a double spectral clustering method to group their trajectories into a set of sub-groups. We also develop a Bidirectional LSTM model to address this problem from a supervised manner. In the experiment, we have achieved a high accuracy of vehicle behavior recognition both on the unsupervised side and the supervised side.

In Chapter 7, we approach the character recognition problem from the angle of multi-label classification. We propose a novel semantic projection network (SPNet) which consists of two stacked autoencoders with special constraints. The first autoencoder reconstructs the input visual feature activations from a pre-trained multi-label CNN in order to learn robust visual embeddings, and the second autoencoder reconstructs the learned visual features from the mapped semantic label space so as to learn robust semantic embeddings. The experiments on the three popular TV series video datasets demonstrate the effectiveness of the proposed model in comparison of state-of-the-art methods. We also show that adding region-based strategy can effectively improve the performance of character recognition.

8.2 Future Work

In this thesis, we have showed that combining convolutional deep neural networks with natural language processing techniques can effectively solve multiple vision recognition task. We hope these techniques can inspire novel ideas and solutions for the joint modeling of image, videos and natural languages.

However, despite the rapid progress in visual content recognition, it is clear that many challenging problems still remain unsolved. One major disadvantage that all the CNN

models for computer vision is that the joint learning of semantic and visual features are mainly based on the training of images with carefully annotated captions or tags, which means that they may not fully describe the real-world user data. Besides, the detection and reasoning of the interconnectivity of abstract concepts and knowledge which is hidden in the visual world is still beyond the scope of current visual recognition algorithms. This is because human can see much more from the visual scene than computers by thinking and reasoning. Hence, the major direction of our future work is to enable the computers to think beyond the vision, modeling abstract concepts and theories, and finally making predictions and reasoning like the human do. Our ultimate goal is to realize Turing's vision of artificial intelligence that computers can sense the world and interact with us using our own languages.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[3] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Jitendra Malik, Pablo Arbeláez, João Carreira, Katerina Fragkiadaki, Ross Girshick, Georgia Gkioxari, Saurabh Gupta, Bharath Hariharan, Abhishek Kar, and Shubham Tulsiani. The three r's of computer vision: Recognition, reconstruction and reorganization. *Pattern Recognition Letters*, 72:4–14, 2016.

[6] Ke Sun, Xianxu Hou, Qian Zhang, and Guoping Qiu. Automatic visual theme discovery from joint image and text corpora. In *2017 2nd International Conference on Multimedia and Image Processing (ICMIP)*, pages 220–224, March 2017.

[7] Ke Sun, Jiasong Zhu, Zhuo Lei, Xianxu Hou, Qian Zhang, Jiang Duan, and Guoping Qiu. Learning deep semantic attributes for user video summarization. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 643–648, July 2017.

[8] Jiasong Zhu, Ke Sun, Sen Jia, Qingquan Li, Xianxu Hou, Weidong Lin, Bozhi Liu, and Guoping Qiu. Urban traffic density estimation based on ultrahigh-resolution

uav video and deep neural network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(12):4968–4981, 2018.

[9] Jiasong Zhu, Ke Sun, Sen Jia, Weidong Lin, Xianxu Hou, Bozhi Liu, and Guoping Qiu. Bidirectional long short-term memory network for vehicle behavior recognition. *Remote Sensing*, 10(6), June 2018.

[10] Jiasong Zhu, Weidong Lin, Ke Sun, Xianxu Hou, Bozhi Liu, and Guoping Qiu. Behavior recognition of moving objects using deep neural networks. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 45–52. IEEE, 2018.

[11] Ke Sun, Zhuo Lei, Jiasong Zhu, Xianxu Hou, Bozhi Liu, and Guoping Qiu. Character prediction in tv series via a semantic projection network. In *International Conference on Multimedia Modeling*, pages 300–311. Springer, 2019.

[12] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[14] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[15] Daniel A Spielmat and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 96–105. IEEE, 1996.

[16] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[17] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2 (11):559–572, 1901.

[18] Gilbert Strang, Gilbert Strang, Gilbert Strang, and Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

[19] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[20] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[21] G Hinton, N Srivastava, and K Swersky. Rmsprop: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning, Coursera lecture 6e*, 2012.

[22] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computer Science*, 2014.

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

[24] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[25] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. word2vec, 2014.

[28] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Convolutional neural networks for large-scale remote-sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):645–657, 2017.

[29] Wei Li, Guodong Wu, Fan Zhang, and Qian Du. Hyperspectral image classification using deep pixel-pair features. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):844–853, 2017.

[30] Zhimin Gao, Lei Wang, Luping Zhou, and Jianjia Zhang. Hep-2 cell image classification with deep convolutional neural networks. *IEEE journal of biomedical and health informatics*, 21(2):416–428, 2017.

[31] Lin Zhu, Yushi Chen, Pedram Ghamisi, and Jón Atli Benediktsson. Generative adversarial networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2018.

[32] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4651–4659, 2016.

[33] Marco Pedersoli, Thomas Lucas, Cordelia Schmid, and Jakob Verbeek. Areas of attention for image captioning. In *ICCV-International Conference on Computer Vision*, 2017.

[34] Chenxi Liu, Junhua Mao, Fei Sha, and Alan L Yuille. Attention correctness in neural image captioning. In *AAAI*, pages 4176–4182, 2017.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, Xiangyu Zhang, and Jian Sun. Object detection networks on convolutional feature maps. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1476–1481, 2017.

[36] Houwen Peng, Bing Li, Haibin Ling, Weiming Hu, Weihua Xiong, and Stephen J Maybank. Salient object detection via structured matrix decomposition. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):818–832, 2017.

[37] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip Torr. Deeply supervised salient object detection with short connections. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5300–5309. IEEE, 2017.

[38] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[39] Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Pablo Arbeláez, and Luc Van Gool. Convolutional oriented boundaries: From image segmentation to high-level tasks. *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[40] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2414–2423. IEEE, 2016.

[41] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[42] Leon A Gatys, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Preserving color in neural artistic style transfer. *arXiv preprint arXiv:1606.05897*, 2016.

[43] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

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

[45] Chen Sun, Chuang Gan, and Ram Nevatia. Automatic concept discovery from parallel text and visual corpora. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2596–2604, 2015.

[46] Sreemanananth Sadanand and Jason J Corso. Action bank: A high-level representation of activity in video. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1234–1241. IEEE, 2012.

[47] Bolei Zhou, Vignesh Jagadeesh, and Robinson Piramuthu. Conceptlearner: Discovering visual concepts from weakly labeled image collections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2015.

[48] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C Platt, et al. From captions to visual concepts and back. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1473–1482, 2015.

[49] Bharat Singh, Xintong Han, Zhe Wu, Vlad I Morariu, and Larry S Davis. Selecting relevant web trained concepts for automated event retrieval. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4561–4569, 2015.

[50] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[51] Fereshteh Sadeghi, Santosh K Divvala, and Ali Farhadi. Viske: Visual knowledge extraction and question answering by visual verification of relation phrases. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 1456–1464. IEEE, 2015.

[52] Santosh Divvala, Ali Farhadi, and Carlos Guestrin. Learning everything about anything: Webly-supervised visual concept learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3270–3277, 2014.

[53] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *Computer Vision–ECCV 2014*, pages 584–599. Springer, 2014.

[54] Pinar Duygulu, Kobus Barnard, Joao FG de Freitas, and David A Forsyth. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *Computer Vision—ECCV 2002*, pages 97–112. Springer, 2002.

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

[56] Ron Shonkwiler. Computing the hausdorff set distance in linear time for any lp point distance. *Information Processing Letters*, 38(4):201–207, 1991.

[57] Marie-Pierre Dubuisson and Anil K Jain. A modified hausdorff distance for object matching. In *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 566–568. IEEE, 1994.

[58] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer, 2006.

[59] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17 (4):395–416, 2007.

[60] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, page 48. ACM, 2009.

[61] Hugo Jair Escalante, Carlos A Hernández, Jesus A Gonzalez, Aurelio López-López, Manuel Montes, Eduardo F Morales, L Enrique Sucar, Luis Villaseñor, and Michael Grubinger. The segmented and annotated iapr tc-12 benchmark. *Computer vision and image understanding*, 114(4):419–428, 2010.

[62] Matthieu Guillaumin, Thomas Mensink, Jakob Verbeek, and Cordelia Schmid. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *2009 IEEE 12th international conference on computer vision*, pages 309–316. IEEE, 2009.

[63] Hao Fu and Guoping Qiu. Fast semantic image retrieval based on random forest. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 909–912. ACM, 2012.

[64] Xiangyu Chen, Yadong Mu, Shuicheng Yan, and Tat-Seng Chua. Efficient large-scale image annotation by probabilistic collaborative multi-label propagation. In *Proceedings of the international conference on Multimedia*, pages 35–44. ACM, 2010.

[65] Ameesh Makadia, Vladimir Pavlovic, and Sanjiv Kumar. Baselines for image annotation. *International Journal of Computer Vision*, 90(1):88–105, 2010.

[66] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[67] Amarnag Subramanya and Jeff A Bilmes. Entropic graph regularization in non-parametric semi-supervised classification. In *Advances in Neural Information Processing Systems*, pages 1803–1811, 2009.

[68] Xiangyu Chen, Xiao-Tong Yuan, Qiang Chen, Shuicheng Yan, and Tat-Seng Chua. Multi-label visual classification with label exclusive context. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 834–841. IEEE, 2011.

[69] Hsin-Yu Ha, Yimin Yang, Fausto C Fleites, and Shu-Ching Chen. Correlation-based feature analysis and multi-modality fusion framework for multimedia semantic retrieval. In *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.

[70] Hao Fu, Qian Zhang, and Guoping Qiu. Random forest for image annotation. *European Conference on Computer Vision*, 2012.

[71] S L Feng, R Manmatha, and Victor Lavrenko. Multiple bernoulli relevance models for image and video annotation. 2004.

[72] Shaoting Zhang, Junzhou Huang andYuchi Huang andYang Yu, Hongsheng Li, and Dimitris Metaxas. Automatic image annotation using group sparsity. 2010.

[73] Motofumi Fukui, Noriji Kato, Wenyuan Qi, and Fuji Xerox. Multi-class labeling improved by random forest for automatic image annotation. 2011.

[74] Qian Zhang and Guoping Qiu. Bundling centre for landmark image discovery. *International Journal of Multimedia Information Retrieval*, 5(1):35–50, 2016.

[75] Engin Mendi, Hélio B Clemente, and Coskun Bayrak. Sports video summarization based on motion analysis. *Computers & Electrical Engineering*, 39(3):790–796, 2013.

[76] Marcus J Pickering, Lawrence Wong, and Stefan M Rüger. Anses: Summarisation of news video. In *International Conference on Image and Video Retrieval*, pages 425–434. Springer, 2003.

[77] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool. Creating summaries from user videos. In *European conference on computer vision*, pages 505–520. Springer, 2014.

[78] Aditya Khosla, Raffay Hamid, Chih-Jen Lin, and Neel Sundaresan. Large-scale video summarization using web-image priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2698–2705, 2013.

[79] Yale Song, Jordi Vallmitjana, Amanda Stent, and Alejandro Jaimes. Tvsum: Summarizing web videos using titles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5179–5187, 2015.

[80] Mayu Otani, Yuta Nakashima, Esa Rahtu, Janne Heikkilä, and Naokazu Yokoya. Video summarization using deep semantic features. *arXiv preprint arXiv:1609.08758*, 2016.

[81] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[82] Yunchao Wei, Wei Xia, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao, and Shuicheng Yan. Cnn: Single-label to multi-label. *arXiv preprint arXiv:1406.5726*, 2014.

[83] Jordi Pont-Tuset, Pablo Arbeláez, Jonathan T. Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping for image segmentation and object proposal generation. *IEEE Trans Pattern Anal Mach Intell*, 39(1):128–140, 2015.

[84] Zhuo Lei, Ke Sun, Qian Zhang, and Guoping Qiu. User video summarization based on joint visual and semantic affinity graph. In *Proceedings of the 2016 ACM workshop on Vision and Language Integration Meets Multimedia Fusion*, pages 45–52. ACM, 2016.

[85] Hairong Liu, Longin J Latecki, and Shuicheng Yan. Robust clustering as ensembles of affinity relations. In *Advances in neural information processing systems*, pages 1414–1422, 2010.

[86] Michael Gygli, Helmut Grabner, and Luc Van Gool. Video summarization by learning submodular mixtures of objectives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3090–3098, 2015.

[87] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Summary transfer: Exemplar-based subset selection for video summarization. *arXiv preprint arXiv:1603.03369*, 2016.

[88] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory. *arXiv preprint arXiv:1605.08110*, 2016.

[89] Yingbo Li and Bernard Merialdo. Multi-video summarization based on videommr. In *11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10*, pages 1–4. IEEE, 2010.

[90] Naveed Ejaz, Irfan Mehmood, and Sung Wook Baik. Efficient visual attention based framework for extracting key frames from videos. *Signal Processing: Image Communication*, 28(1):34–44, 2013.

[91] Bin Zhao and Eric P Xing. Quasi real-time summarization for consumer videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2513–2520, 2014.

[92] ZuWhan Kim and Jitendra Malik. Fast vehicle detection with probabilistic feature grouping and its application to vehicle tracking. In *IEEE International Conference on Computer Vision*, page 524. IEEE, 2003.

[93] Brendan Tran Morris and Mohan Manubhai Trivedi. A survey of vision-based trajectory learning and analysis for surveillance. *IEEE transactions on circuits and systems for video technology*, 18(8):1114–1127, 2008.

[94] Rita Cucchiara, Massimo Piccardi, and Paola Mello. Image analysis and rule-based reasoning for a traffic monitoring system. *IEEE Transactions on Intelligent Transportation Systems*, 1(2):119–130, 2000.

[95] Akio Yoneyama, Chia-Hung Yeh, and C-C Jay Kuo. Robust vehicle and traffic information extraction for highway surveillance. *EURASIP Journal on Applied Signal Processing*, 2005:2305–2321, 2005.

[96] Liang Wang, Fangliang Chen, and Huiming Yin. Detecting and tracking vehicles in traffic by unmanned aerial vehicles. *Automation in construction*, 72:294–308, 2016.

[97] Colin Brooks, R Dobson, D Banach, David Dean, Thomas Oommen, R Wolf, T Havens, T Ahlborn, and Ben Hart. Evaluating the use of unmanned aerial vehicles for transportation purposes. *MDOT, Houghton, Michigan, MTRI-MDOTUAV-FR-2014*, 2015.

[98] Hailing Zhou, Hui Kong, Lei Wei, Douglas C Creighton, and Saeid Nahavandi. Efficient road detection and tracking for unmanned aerial vehicle. *IEEE Transactions on Intelligent Transportation Systems*, 16(1):297–309, 2015.

[99] Hyondong Oh, Seungkeun Kim, Hyo-Sang Shin, Antonios Tsourdos, and Brian A White. Behaviour recognition of ground vehicle using airborne monitoring of unmanned aerial vehicles. *International Journal of Systems Science*, 45(12):2499–2514, 2014.

[100] Jens Leitloff, Stefan Hinz, and Uwe Stilla. Vehicle detection in very high resolution satellite images of city areas. *IEEE transactions on Geoscience and remote sensing*, 48(7):2795–2806, 2010.

[101] Stefan Hinz. Detection and counting of cars in aerial images. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 3, pages III–997. IEEE, 2003.

[102] Wei Yao and Uwe Stilla. Comparison of two methods for vehicle extraction from airborne lidar data toward motion analysis. *IEEE Geoscience and remote sensing letters*, 8(4):607–611, 2011.

[103] T Nathan Mundhenk, Goran Konjevod, Wesam A Sakla, and Kofi Boakye. A large contextual dataset for classification, detection and counting of cars with deep learning. In *European Conference on Computer Vision*, pages 785–800. Springer, 2016.

[104] Benjamin Coifman, Mark McCord, Rabi G Mishalani, and Keith Redmill. Surface transportation surveillance from unmanned aerial vehicles. In *Proc. of the 83rd Annual Meeting of the Transportation Research Board*, 2004.

[105] Anuj Puri. A survey of unmanned aerial vehicles (uav) for traffic surveillance. *Department of computer science and engineering, University of South Florida*, pages 1–29, 2005.

[106] Thomas Moranduzzo and Farid Melgani. Automatic car counting method for unmanned aerial vehicle images. *IEEE Transactions on Geoscience and Remote Sensing*, 52(3):1635–1647, 2014.

[107] Giuseppe Salvo, Luigi Caruso, and Alessandro Scordo. Urban traffic analysis through an uav. *Procedia-Social and Behavioral Sciences*, 111:1083–1091, 2014.

[108] Konstantinos Kanistras, Goncalo Martins, Matthew J Rutherford, and Kimon P Valavanis. Survey of unmanned aerial vehicles (uavs) for traffic monitoring. In *Handbook of unmanned aerial vehicles*, pages 2643–2666. Springer, 2015.

[109] Karen Gallagher and Patrick Lawrence. Unmanned systems and managing from above: the practical implications of uavs for research applications addressing urban sustainability. In *Urban Sustainability: Policy and Praxis*, pages 217–232. Springer, 2016.

[110] Muhammad Arsalan Khan, Wim Ectors, Tom Bellemans, Davy Janssens, and Geert Wets. Uav-based traffic analysis: A universal guiding framework based on literature survey. *Transportation Research Procedia*, 22:541–550, 2017.

[111] Benjamin Coifman, Mark McCord, Rabi G Mishalani, Michael Iswalt, and Yuxiong Ji. Roadway traffic monitoring from an unmanned aerial vehicle. In *IEE Proceedings-Intelligent Transport Systems*, volume 153, pages 11–20. IET, 2006.

[112] Per Skoglar, Umut Orguner, David Törnqvist, and Fredrik Gustafsson. Road target search and tracking with gimballed vision sensor on an unmanned aerial vehicle. *Remote sensing*, 4(7):2076–2111, 2012.

[113] Thomas Moranduzzo and Farid Melgani. Detecting cars in uav images with a catalog-based approach. *IEEE Transactions on Geoscience and Remote Sensing*, 52 (10):6356–6367, 2014.

[114] Hsu-Yung Cheng, Chih-Chia Weng, and Yi-Ying Chen. Vehicle detection in aerial surveillance using dynamic bayesian networks. *IEEE Transactions on Image Processing*, 21(4):2152–2159, 2012.

[115] Xueyun Chen, Shiming Xiang, Cheng-Lin Liu, and Chun-Hong Pan. Vehicle detection in satellite images by hybrid deep convolutional neural networks. *IEEE Geoscience and remote sensing letters*, 11(10):1797–1801, 2014.

[116] Ziyi Chen, Cheng Wang, Chenglu Wen, Xiuhua Teng, Yiping Chen, Haiyan Guan, Huan Luo, Liujuan Cao, and Jonathan Li. Vehicle detection in high-

resolution aerial images via sparse representation and superpixels. *IEEE Transactions on Geoscience and Remote Sensing*, 54(1):103–116, 2016.

[117] Kang Liu and Gellert Mattyus. Fast multiclass vehicle detection on aerial images. *IEEE Geoscience and Remote Sensing Letters*, 12(9):1938–1942, 2015.

[118] Guoliang Mo and Sanyuan Zhang. Vehicles detection in traffic flow. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 2, pages 751–754. IEEE, 2010.

[119] Zezhi Chen, Tim Ellis, and Sergio A Velastin. Vehicle detection, tracking and classification in urban traffic. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 951–956. IEEE, 2012.

[120] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 833–841, 2015.

[121] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 589–597, 2016.

[122] Zhuoyi Zhao, Hongsheng Li, Rui Zhao, and Xiaogang Wang. Crossing-line crowd counting with two-phase deep neural networks. In *European Conference on Computer Vision*, pages 712–726. Springer, 2016.

[123] Carlo Migel Bautista, Clifford Austin Dy, Miguel Iñigo Mañalac, Raphael Angelo Orbe, and Macario Cordel. Convolutional neural network for vehicle detection in low resolution traffic videos. In *Region 10 Symposium (TENSYMP), 2016 IEEE*, pages 277–281. IEEE, 2016.

[124] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[125] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *Computer Science*, 2013.

[126] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2154, 2014.

[127] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[128] Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 850–855. IEEE, 2006.

[129] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 3464–3468. IEEE, 2016.

[130] Ju Hong Yoon, Ming-Hsuan Yang, Jongwoo Lim, and Kuk-Jin Yoon. Bayesian multi-object tracking using motion context from multiple objects. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 33–40. IEEE, 2015.

[131] Caglayan Dicle, Octavia I Camps, and Mario Sznaier. The way they move: Tracking multiple targets with similar appearance. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2304–2311, 2013.

[132] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[133] G Ayorkor Mills-Tettey, Anthony Stentz, and M Bernardine Dias. The dynamic hungarian algorithm for the assignment problem with changing costs. 2007.

[134] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[135] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, pages 850–865. Springer, 2016.

[136] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[137] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[138] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[139] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[140] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[141] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015.

[142] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[143] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[144] Jens Leitloff, Dominik Rosenbaum, Franz Kurz, Oliver Meynberg, and Peter Reinartz. An operational system for estimating road traffic information from aerial images. *Remote Sensing*, 6(11):11315–11341, 2014.

[145] Xianbin Cao, Changxia Wu, Jinhe Lan, Pingkun Yan, and Xuelong Li. Vehicle detection and motion analysis in low-altitude airborne video under urban environment. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10): 1522–1533, 2011.

[146] Eun-Ha Choi. Crash factors in intersection-related crashes: An on-scene perspective. Technical report, 2010.

[147] Miller T., Kolosh K., Fearn K., and Porretta K. Injury facts, 2015 edition. *Technical Report, National Safety Council*, 2015.

[148] Aaron F. Bobick and James W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on pattern analysis and machine intelligence*, 23(3):257–267, 2001.

[149] Weiming Hu, Xuejuan Xiao, Dan Xie, and Tieniu Tan. Traffic accident prediction using vehicle tracking and trajectory analysis. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, volume 1, pages 220–225. IEEE, 2003.

[150] Huan-Sheng Song, Sheng-Nan Lu, Xiang Ma, Yuan Yang, Xue-Qin Liu, and Peng Zhang. Vehicle behavior analysis using target motion trajectories. *IEEE Transactions on Vehicular Technology*, 63(8):3580–3591, 2014.

[151] Alexander Barth and Uwe Franke. Estimating the driving state of oncoming vehicles from a moving platform using stereo vision. *IEEE Transactions on Intelligent Transportation Systems*, 10(4):560–571, 2009.

[152] Dietmar Kasper, Galia Weidl, Thao Dang, Gabi Breuel, Andreas Tamke, Andreas Wedel, and Wolfgang Rosenstiel. Object-oriented bayesian networks for detection of lane change maneuvers. *IEEE Intelligent Transportation Systems Magazine*, 4(3):19–31, 2012.

[153] Venkatesh Saligrama, Janusz Konrad, and Pierre-Marc Jodoin. Video anomaly identification. *IEEE Signal Processing Magazine*, 27(5):18–33, 2010.

[154] Han Huang, Zhaoquan Cai, Shixu Shi, Xianheng Ma, and Yifan Zhu. Automatic detection of vehicle activities based on particle filter tracking. *Proc. ISCSCT*, 2(2): 2, 2009.

[155] Michael Pucher, Dietmar Schabus, Peter Schallauer, Yuriy Lypetskyy, Franz Graf, Harald Rainer, Michael Stadtschnitzer, Sabine Sternig, Josef Birchbauer, Wolfgang Schneider, et al. Multimodal highway monitoring for robust incident detection. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 837–842. IEEE, 2010.

[156] Shunsuke Kamijo, Howard Koo, Xiaolu Liu, Kenji Fujihira, and Masao Sakauchi. Development and evaluation of real-time video surveillance system on highway based on semantic hierarchy and decision surface. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 840–846. IEEE, 2005.

[157] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[158] Stefan Atev, Grant Miller, and Nikolaos P Papanikolopoulos. Clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):647–657, 2010.

[159] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[160] Ken-ichi Kamijo and Tetsuji Tanigawa. Stock price pattern recognition-a recurrent neural network approach. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 215–221. IEEE, 1990.

[161] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[162] Jack Hanson, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Improving protein disorder prediction by deep bidirectional long short-term memory recurrent neural networks. *Bioinformatics*, 33(5):685–692, 2016.

[163] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 207–212, 2016.

[164] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[165] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, 2012.

[166] Seung-Hwan Bae and Kuk-Jin Yoon. Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1218–1225, 2014.

[167] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *2015 IEEE international conference on computer vision (ICCV)*, number EPFL-CONF-230283, pages 4705–4713. IEEE, 2015.

[168] François Chollet et al. Keras, 2015.

[169] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[170] Alexandre Hervieu, Patrick Bouthemy, and Jean-Pierre Le Cadre. A hmm-based method for recognizing dynamic video contents from trajectories. In *Image Pro-*

cessing, 2007. ICIP 2007. IEEE International Conference on, volume 4, pages IV–533. IEEE, 2007.

[171] Brendan Tran Morris and Mohan M Trivedi. Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach. *IEEE transactions on pattern analysis and machine intelligence*, 33(11):2287–2301, 2011.

[172] Qing fan, Tihong Ruan, Jiamin Wu, and Tianyang Dong. Vehicle behavior recognition method based on quadratic spectral clustering and hmm-rf hybrid model. *Computer Science*, 43(5):288–293, 2016.

[173] Florent Altché and Arnaud De La Fortelle. An lstm network for highway trajectory prediction. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 353–359. IEEE, 2017.

[174] Timothee Cour, Benjamin Sapp, Akash Nagle, and Ben Taskar. Talking pictures: Temporal grouping and dialog-supervised person recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1014–1021. IEEE, 2010.

[175] Josef Sivic, Mark Everingham, and Andrew Zisserman. "who are you?"-learning person specific classifiers from video. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1145–1152. IEEE, 2009.

[176] Makarand Tapaswi, Martin Bäuml, and Rainer Stiefelhagen. Story-based video retrieval in tv series using plot synopses. In *Proceedings of International Conference on Multimedia Retrieval*, page 137. ACM, 2014.

[177] Makarand Tapaswi, Martin Bauml, and Rainer Stiefelhagen. Storygraphs: visualizing character interactions as a timeline. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 827–834, 2014.

[178] Motoi Iwata, Atsushi Ito, and Koichi Kise. A study to achieve manga character retrieval method for manga images. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 309–313. IEEE, 2014.

[179] Zhen Dong, Su Jia, Tianfu Wu, and Mingtao Pei. Face video retrieval via deep learning of binary hash representations. In *AAAI*, pages 3471–3477, 2016.

[180] Yan Li, Ruiping Wang, Zhen Cui, Shiguang Shan, and Xilin Chen. Compact video code and its application to robust face retrieval in tv-series. In *BMVC*, 2014.

[181] Arsha Nagrani and Andrew Zisserman. From benedict cumberbatch to sherlock holmes: Character identification in tv series without a script. *CoRR*, abs/1801.10442, 2017.

[182] Yan Li, Ruiping Wang, Shiguang Shan, and Xilin Chen. Hierarchical hybrid statistic based video binary code and its application to face retrieval in tv-series. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*, volume 1, pages 1–8. IEEE, 2015.

[183] Vignesh Ramanathan, Armand Joulin, Percy Liang, and Li Fei-Fei. Linking people in videos with their names using coreference resolution. In *European Conference on Computer Vision*, pages 95–110. Springer, 2014.

[184] Makarand Tapaswi, Martin Bäuml, and Rainer Stiefelhagen. "knock! knock! who is it?" probabilistic person identification in tv-series. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2658–2665. IEEE, 2012.

[185] Piotr Bojanowski, Francis Bach, Ivan Laptev, Jean Ponce, Cordelia Schmid, and Josef Sivic. Finding actors and actions in movies. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2280–2287. IEEE, 2013.

[186] Omkar M Parkhi, Esa Rahtu, and Andrew Zisserman. It's in the bag: Stronger supervision for automated face labelling. In *ICCV Workshop*, volume 2, page 6, 2015.

[187] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3): 1–13, 2007.

[188] Chenghua Li, Qi Kang, Guojing Ge, Qiang Song, Hanqing Lu, and Jian Cheng. Deepbe: Learning deep binary encoding for multi-label classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 39–46, 2016.

[189] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. Large-scale multi-label text classification: revisiting neural networks. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer, 2014.

[190] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *MSc Thesis, University Toronto*, 2009.

[191] Fei Wu, Zhuhao Wang, Zhongfei Zhang, Yi Yang, Jiebo Luo, Wenwu Zhu, and Yueting Zhuang. Weakly semi-supervised deep learning for multi-label image annotation. *IEEE Transactions on Big Data*, 1(3):109–122, 2015.

[192] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2285–2294. IEEE, 2016.

[193] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, 2011.

[194] Zhanying He, Chun Chen, Jiajun Bu, Ping Li, and Deng Cai. Multi-view based multi-label propagation for image annotation. *Neurocomputing*, 168(C):853–860, 2015.

[195] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[196] Ziming Zhang and Venkatesh Saligrama. Zero-shot learning via joint latent similarity embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6034–6042, 2016.

[197] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[198] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 512–519, 2014.

[199] Minling Zhang and Zhihua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

[200] Yunchao Wei, Wei Xia, Min Lin, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao, and Shuicheng Yan. Hcp: A flexible cnn framework for multi-label image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9): 1901–1907, 2016.

[201] Qinghua Yu, Jinjun Wang, Shizhou Zhang, Yihong Gong, and Jizhong Zhao. Combining local and global hypotheses in deep neural network for multi-label image classification. *Neurocomputing*, 235:38–45, 2017.

[202] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.