3D Scene Recognition From a Single Image

Altaf Khan

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Eastern Mediterranean University
February 2021
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Prof. Dr. Hadi Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Prof. Dr. Hasan Demirel
Co-Supervisor

Assoc. Prof. Dr. Alexander Chefranov
Supervisor

Examining Committee

1. Prof. Dr. Abdullah Aydın Alatan
2. Prof. Dr. Gözde Bozdağı Akar
3. Prof. Dr. Hasan Demirel
4. Prof. Dr. Marifi Güler
5. Assoc. Prof. Dr. Adnan Acan
6. Assoc. Prof. Dr. Alexander Chefranov
7. Assoc. Prof. Dr. Önsen Toygar

ABSTRACT

Human eyes capture the world around us and effortlessly derive an impression of scene depth from a single image. However, developing an artificial system that can identify the impression of the 3D scene with the same performance and robustness as humans, still is a challenge for researchers from such fields as physiology, computer science, and artificial intelligence. The 3D scene recognition from a single image is an important problem for many applications of computer vision such as autonomous vehicle control, scene understanding, and 3D TV. The contributions of the thesis are explored in three different ways. First, the segmentation-based feature extraction method is introduced to classify the relatively clear geometry structure images, in which the image features are extracted by exploiting predefined templates, each associated with an individual classifier. Each of the individual classifiers learns a discriminative model and their outcome are fused together using sum-rule for recognizing the 3D scene geometry of an input image. It achieves 86.25% recognition accuracy on ‘stage dataset 1’, which is higher than the state-of-the-art methods.

In the second contribution, a new method of 3D scene recognition-based on the fusion of deep convolutional neural network (CNN) features and texture gradient features is presented. Meanwhile, as the 3D scene geometry dataset is not publically given, thus, a medium scale, ‘stage dataset 2’, is introduced. Experimental results exhibit that the proposed method reaches 86.29% recognition accuracy, which achieves higher accuracy and faster than the baseline methods.

Finally, in the third contribution, the handcrafted features are integrated with multi-layer features at different intermediate blocks of CNN, and each block is connected with an individual classifier and then scores of these classifiers are combined while using sum and product-rule to recognize the scene geometry type. The introduced approach is validated on two benchmark datasets and it achieves 95.17% and 97.68% recognition accuracy on ‘stage 2 dataset’ and ‘15-scene’, which is superior to the state-of-the-art methods.

Keywords: CNN, Ensemble of classifiers, Handcrafted feature, Multi-layer features, Predefined templates, Stages, 3D scene recognition

ÖZ

İnsan gözleri çevremizdeki dünyayı yakalar ve tek bir görüntüden zahmetsizce sahne derinliği izlenimi çıkarır. Bununla birlikte, 3B sahnenin izlenimini insanlarla aynı performans ve sağlamlıkla tanımlayabilen yapay bir sistem geliştirmek, fizyoloji, bilgisayar bilimi ve yapay zeka gibi alanlardan araştırmacılar için hala bir zorluktur. Tek bir görüntüden 3B sahne tanıma, otonom araç kontrolü, sahne anlama ve 3B TV gibi birçok bilgisayar görüşü uygulaması için önemli bir sorundur. Tezin katkıları üç farklı şekilde incelenmiştir. İlk olarak, her biri ayrı bir sınıflandırıcıyla ilişkilendirilmiş önceden tanımlanmış şablonlardan yararlanılarak görüntü özniteliklerinin çıkarıldığı nispeten net geometri yapı görüntülerini sınıflandırmak için segmentasyon tabanlı öznitelik çıkarma yöntemi tanıtılmıştır. Her bir sınıflandırıcı, ayırt edici bir modeli öğrenir ve sonuçları, bir giriş görüntüsünün 3B sahne geometrisini tanımak için toplama kuralı kullanılarak bir araya getirilir. Son teknoloji yöntemlerden daha yüksek olan "aşama veri kümesi 1" de %86.25 tanınma doğruluğuna ulaşır.

İkinci katkıda, derin evrişimli sinir ağı (CNN) özelliklerinin ve doku gradyan özelliklerinin birleşimine dayalı yeni bir 3B sahne tanıma yöntemi sunulmuştur. Bu arada, 3B sahne geometrisi veri kümesi halka açık olarak verilmediğinden, orta ölçekli bir "aşama veri kümesi 2" tanıtıldı. Deneysel sonuçlar, önerilen yöntemin, temel yöntemlerden daha yüksek doğruluk ve daha hızlı olan %86.29 tanıma doğruluğuna ulaştığını göstermektedir.

Son olarak, üçüncü katkı olarak, el yapımı özellikler, farklı CNN ara bloklarında çok

katmanlı özniteliklerle entegre edilir ve her blok ayrı bir sınıflandırıcıyla birleştirilir ve ardından bu sınıflandırıcıların puanları, toplam ve çarpım kuralı kullanılarak birleştirilir. Sunulan yaklaşım, iki kıyaslama veri kümesinde doğrulanmıştır ve son teknoloji yöntemlerden daha üstün olan "aşama veri kümesi 2" ve "15 sahnesi" veri kümesinde %95.17 ve %97.68 tanıma doğruluğuna ulaşır.

Anahtar Kelimeler: CNN, Grup sınıflandırıcılar, El işi öznitelikleri, Çok katmanlı öznitelikler, Önceden tanımlanmış şablonlar, Sahneler, 3B sahne tanıma

DEDICATION

To My Family

ACKNOWLEDGEMENT

First and foremost I would like to show my high gratitude and appreciation to my supervisor, Assoc. Prof. Dr. ALEXANDER CHEFRANOV and my Co-supervisor Prof. Dr. HASAN DEMIREL for their endless effort and great support along with valuable information that had benefit value on understanding this interesting topic with more details during their supervision of my thesis. I must show my sincere respect to their passion that they had for image processing & computer vision, as without their keen support, valuable comments and suggestions, I would not have been able to complete this work.

I would like to thankful to my follow-up jury committee members Assoc. Prof. Dr. ADNAN ACAN and Assoc. Prof. Dr. ÖNSEN TOYGAR for their constructive advices and suggestions.

I am truly thankful to my parents, brothers and my sisters for their love, trust, and support in all means. I dedicate this study to my family who was and will always be my idol.

Finally, I would like to thank all my friends for supporting and encouraging me during my thesis study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
B	Bines
BoW	Bag of Words
BP	Back Propagation
CMY	Cyan, Magenta, Yellow
CNN	Convolutional Neural Networks
CWCH	Concentric Weighted Circles Histogram
DFF-ADML	Deep Feature Fusion through Adaptive Discriminative Metric Learning
E	Entropy
ELM	Extreme Learning Machine
FC	Fully Connected
FK	Fisher Kernel
F-s	F-score
FTOTLM	Fine-Tuned On the proposed Transfer Learning Model
G(x, y)	Gaussian function
GAP	Global Average Pooling
GB	Giga Bytes
G-MS2F	GoogLeNet based Multi-Stage Feature Fusion
HF-MSF	Handcrafted Features with CNN Multi-Stages Features
HGSIR	Hybrid Geometric Spatial Image Representation
HOG	Histogram of Oriented Gradients
HSV	Hue, Saturation, Value

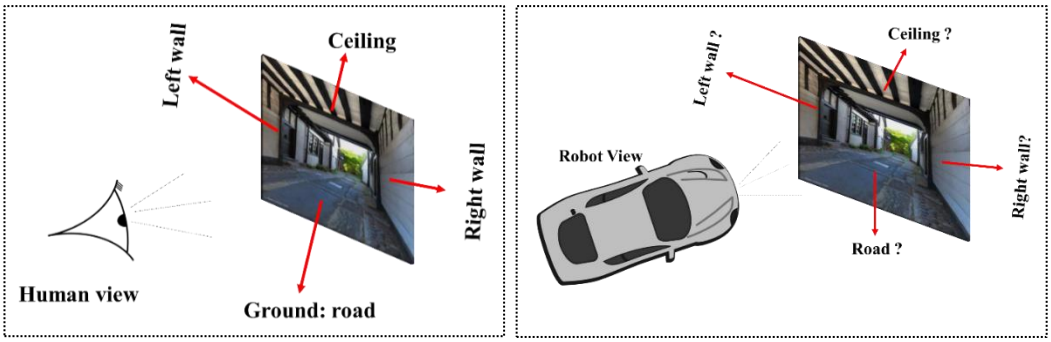
I	Image
LBP	Local Binary Pattern
LBP-E	Local Binary Pattern & Entropy value
LVFC-HSF	Local Visual Feature Coding based on Heterogeneous Structure Fusion
MLA	Machine Learning Algorithm
Model-HFF	Model of Handcrafted and multi-layer features using Feature-level Fusion
Model-HSF	Model of Handcrafted and multi-layer features using Score-level Fusion
OVH	Orthogonal Vectors Histogram
Pr	Precision
R-CNN	Region based CNN
Re	Recall
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
Seg	Segment
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SVM	Support Vector Machine
T	Template
TGF-DeepFF	Texture Gradient Features and Deep CNN Feature Fusion
TV	TeleVision
V	Vector

VGG16	Visual Geometric Group and 16 indicates number of layers
YTR(N)	Training labels
YTS(M)	Testing labels
Ω	Set of categories
Acc	Accuracy
CL_t	tth classifier
$H \times W$	Image size with height and width
α	Shape parameter
β	Scale parameter
σ	Standard deviation
${F_\mathrm{N}}^D$	Feature vectors with Dimension (D) for N images
$\|V\|_2$	L2-norm
ω_i	ith Category
P_{tj}^S	Probability of tth classifier of s number of categories for jth image
∇I	Gradient of Image I
1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional

Chapter 1

INTRODUCTION

Vision is the process of discovering from images what is present in the world that is captured in an image and where it is (Marr, 1982)? Human eyes capture 10 GB of information per second from the world around us (Anderson, Van Essen, & Olshausen, 2005) and effortlessly derive an impression of scene depth from a single image the depth is the number of bits used to represent each pixel in an image. This is because the world around us behaves regularly and structure regularities are directly reflected in the 2D image of a 3D scene (Richards, Jepson, & Feldman, 1996). However, developing an artificial system that can identify the impression of the scene with the same performance and robustness as humans, still is a challenge for researchers from such fields as physiology, engineering, computer science, and artificial intelligence. For instance, in Figure 1.1 (a), the human can understand the structure of the figure as a tunnel with two sides (right and left wall), ceiling, and ground surface. But it is a challenge for a robot (computer vision algorithm) to understand the structure of the image (Derek Hoiem, Efros, & Hebert, 2007; Nedovic, Smeulders, Redert, & Geusebroek, 2010). How does it understand that input image is a tunnel and its parts? Furthermore, how does robot understand that green bushes are away from the position of the camera (depth of the object)? Deriving the same impression of the real world as a human can understand from the single 2D image is a challenging task for the computer vision algorithms.



(a) (b)

Figure 1.1: A simple image of indoor scene, having ceiling, side walls, and road. (a) Human view and understand the scene types and its parts. (b) A robot view the same image and does he understand the scene and its subparts?

In computer vision, this impression is expressed in terms of 3D scene structure of a 2D image. This problem is particularly challenging because of the vast complexity and variation in appearance, we observe from our visual world. Existing computer systems have limited memory and resources, therefore, researchers have narrowed down this problem and split the indoor and outdoor natural scenes into finite set of categories. Indoor images are mostly related to the human-made objects, e.g., room, kitchen, office, tunnel, etc., whereas the outdoor images are natural scenes without borders, e.g., sky-mountain-ground, building and sky, etc. Examples of indoor and outdoor images are shown in Figure 1.2. These categories are designed on the base of image scene geometry. Image scene geometry represents the 3D rough structure, also called as 3D scene geometry, the examples are shown in the top right corners of Figure 1.2, (a), (b). The 3D scene geometry recognition is important for many computer vision applications, such as 3D TV, navigation system, video categorization (Lou, Gevers, & Hu, 2015; Nedovic et al., 2010).

(a) (b)

Figure 1.2: (a) an outdoor sky-background-ground image and (b) an indoor image corner of the room. There 3D scene structure are shown in top right corner, 3D scene geometry figures are given at (Nedovic et al., 2010).

It is important for a computer vision system to understand the scene geometry of an input image because it is easier to understand and extract the image layout when its geometry type is known (Nedovic et al., 2010). There are several 3D scene geometries with which it can be possible to handle both indoor and outdoor images. Scene datasets introduced in (J. Deng et al., 2009; Xiao, Hays, Ehinger, Oliva, & Torralba, 2010) used about 900 and 1000 categories, respectively. However, these datasets have large number of categories that cannot be easily manipulated for 3D scene geometry recognition. Thus, Nedovic at al. (Nedovic et al., 2010) proposed significantly more compact approach splitting the indoor and outdoor scenes into 12 categories. These 3D scene geometries reflect the rough structure of the images where the small objects are ignored and are called '*stages'*. In consequence, the 3D scene recognition problem can be solved by following way: train the 3D scene recognition model by using labeled images and then this model can be validated on testing image (s). For instance, Figure 1.3 represents the generic model of 3D scene recognition of a single image. In this model, an artificial intelligence (AI) based system which is well trained by using training images, it has ability to identify the 3D scene geometry type of input image. While, to date, computer vision researchers have made astonishing progress in many of the individual fields of vision, such as object classification (Jia Deng et al., 2014),

face recognition (Zarbakhsh & Demirel, 2018), structure from motion (Iglhaut et al., 2019), and matching and tracking (Luo, Sun, Chen, Ji, & Xia, 2015). Meanwhile, several researchers have paid attention to develop 3D scene recognition models that have ability to recognize 3D scene of a single input image (Lou et al., 2015; Nedovic et al., 2010).

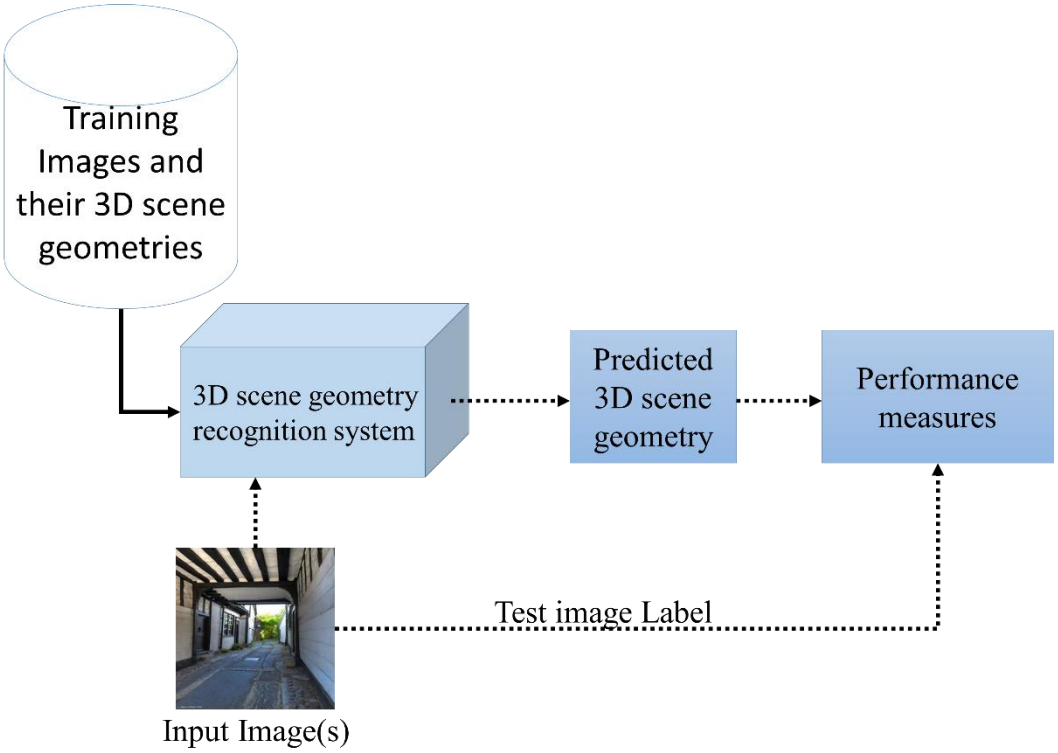


Figure 1.3: An artificial intelligence system that can recognize the 3D scene geometry of an image. Black arrows are indicating training images. Dotted arrows indicate data flow of test image (s).

The short introduction of these methods and their drawbacks are discussed here. Nedovic et al. (Nedovic et al., 2010) divide an image into 4×4 grid parts, which is called patches and extract the rich discriminative information from each patch and combine these information into a single feature vector. These $n \times n$ parts of an image has height $h = \frac{W}{n}$ and width $w = \frac{H}{n}$, where H and W are representing the height and width of an image measured in pixels. The discriminative information is a set of

features of an image. These features, including parameters of Weibull distribution (J.-M. Geusebroek & Smeulders, 2005), color, and perspective line features (J. Geusebroek, Smeulders, & Weijer, 2003) are combined for each input image and fed into a machine learning algorithm. Machine learning algorithms, such as Support vector machine (SVM) (Cortes & Vapnik, 1995), are programs that provide the ability to learn the system automatically and update from experience without being explicitly designed (Breiman, 2001). It achieves 38.0% recognition accuracy of 12 scene geometry dataset. However, this model did not show significant performance because it uses grid patches for features extraction which may result in large difference for the same category (Lou et al., 2015). For example, two images in Figure 4.1 belonged to the same category, which is sky-ground but their results may have a large difference. Such as, a square region which is labeled with red color, have different image content for both image.

(a) (b)

Figure 1.4: Two different images of sky-ground class. Same region of the two images generate different content information of the same class.

To solve above issue, Lou et al. (Lou et al., 2015) use predefined templates to segment the each image and then segments are used to extract the image features, namely, Histogram oriented gradients (HOG) (Dalal & Triggs, 2005), mean value of RGB and HSV components, parameters (α, β) of Weibull distribution (J.-M. Geusebroek & Smeulders, 2005) from each patch. The predefined template represents a 2D rough

structure of 3D scene geometry (Lou et al., 2015). Instead of dividing the whole image into gird patches, they divide each template-based and obtained segment into patches, and features from these patches for all the segments are fused into a one feature vector. They introduced a model to learn a representation from image features to a category by utilizing the latent variables representing model sub-parts, i.e., sky-background-ground. This model achieves 47.3% recognition accuracy of 3D scene geometry on the dataset used in (Nedovic et al., 2010). However, in this method, the template-based segmentation is used. For each image, it generates hundreds of segments (Lou et al., 2015) and finds the segment that has the largest overlap-to-union score for each component of the templates, is computationally expensive.

Other approaches that can be used for 3D scene recognition are based on Bag of words (BoW) model (Lazebnik, Schmid, & Ponce, 2006). For example, J. Sanchez et al. (Sanchez et al., 2013) use dense scale-invariant feature transform (SIFT)(Lowe, 1999) features in the Fisher Kernel (FK) (Sanchez et al., 2013) framework as an alternative patch encoding technique and it achieves 47.20% recognition accuracy by using 50% of SUN dataset (Xiao et al., 2010) samples for training (see Table 5, at last row (Sanchez et al., 2013)). The most recent approaches of image recognition are based on convolutional neural networks (CNN). These networks are typically trained on large image datasets, e.g., ImageNet (J. Deng et al., 2009), and have achieved sufficient accuracy in many applications, such as scene classification, face detection, and object localization (He, Zhang, Ren, & Sun, 2016; S. Liu & Deng, 2015; Szegedy et al., 2015) (B. Zhou, Lapedriza, Khosla, Oliva, & Torralba, 2018). These techniques show high recognition accuracy for large datasets (B. Zhou et al., 2018) and also applicable for the 3D scene recognition if a large dataset is given. On the contrary, these networks require the large labeled images dataset and need high performance hardware

resources for parameters optimization (Marculescu, Stamoulis, & Cai, 2018). It is a one of the CNN main challenges. This challenge also exists with 3D scene recognition as well, because the large dataset of 3D scene geometries does not available publically, which can be used to train the CNN networks. There are some common shortfalls with state-of-the-art approaches:

1) The effectiveness of the existing approaches (Lou et al., 2015; Nedovic et al., 2010) were measured on the limited images (about 2000 images), which did not reach the significant level of 3D scene recognition. Furthermore, the classification performance is not enough, distinguishing the scene images with complex structure, this is because the presence of humans in feature extraction greatly affects the representation of the scene image.
2) The datasets of 3D scene geometry are not publically available.
3) The goal of the thesis is to develop the novel method of 3D scene recognition for both indoor and outdoor images. To overcome the above shortfalls, nature of the topic dictates the necessity of both theoretical and experimental studies on 3D scene geometry recognition to measure the effectiveness of the known and developed methods.

1.1 Contributions

The key contributions of the thesis are as follows:

1) Introducing a method of 3D scene geometry recognition (Khan, Chefranov, & Demirel, 2020a)
    i. Utilizing different types of image features that provide rich information of scene depth.

ii. Each predefined template represents a different rough structure of an image, therefore, applying predefined templates to extract the features from subpart of the image.

iii. Combine these features for each template and feed to an individual machine learning algorithm, combine their yield to predict the final 3D scene geometry.

iv. Achieve the state-of-the-art recognition accuracy on two different datasets.

2) Introducing novel 3D scene geometry dataset (Khan, Chefranov, & Demirel, 2020b) and providing a deep CNN based model of 3D scene geometry recognition that yields state-of-the-art performance on the 3D scene geometry dataset (12000 scene images).

3) Introducing a novel 3D scene geometry recognition model by integration of handcrafted features with deep CNN multi-layer features by utilizing the feature fusion and score-level fusion techniques.

i. Extracting deep features from multiple layers of CNN model.

ii. Utilizing different types of handcrafted features that provide rich information of scenes geometries.

iii. Combine handcrafted features and deep features at multiple layers of CNN and then score-level fusion is used to predict the 3D scene geometry.

1.2 Outline

The rest of the thesis is structured as follows. Chapter 2 describes related work, background of 3D scene recognition, and problem definition. In Chapter 3, the segmentation-based 3D scene recognition system is proposed and evaluated on two benchmark datasets. Chapter 4 presents a novel 3D scene geometry dataset and also a deep CNN based model of 3D scene geometry recognition. Chapter 5 introduces a

novel model of 3D scene recognition using handcrafted features, which have discriminative information about 3D scene geometry, combine with multi-layer deep features at different layers of CNN model. The evaluation results on novel 3D scene geometry dataset are also given in this chapter. Chapter 6 concludes the study and discusses the future work.

Chapter 2

BACKGROUND, RELATED WORK, AND PROBLEM DEFINITION

In this chapter, Section 2.1 introduces background (notions, concepts, definitions) of 3D scene recognition. Section 2.2 describes the deep CNN architecture. Section 2.3 represents the existing methods and their results of scene recognition. Section 2.4 describes the problem definition of this thesis.

2.1 Notions and Techniques of 3D Scene Recognition

In this Subsection, firstly, the notion of scene, digital image are described. Then basic concept of 3D scene recognition is given in Subsection 2.1.1 and its steps, including inputs, pre-processing steps, features extractions and different types of features, evaluation of 3D scene recognition model, and performance metric are given in Subsections 2.1.2-7, respectively.

What is a Scene? According to Xiao et al. (Xiao et al., 2010), "a scene is a place in which a human can act within, or a place to which a human being could navigate". Oliva et al. (Oliva & Torralba, 2001b) , p. 146, define the scene as follows: "If an image represents an 'object' when the view subtends one to two meters around the observer, a 'view on a scene' begins when there is actually a more space or distance between the observer and the fixated point. Typically this distance is more than five meters. Thus, a scene is defined as place in which we can move". A complete scene on 2D image represents existing object(s), semantic relations between objects, and

contextual information with respect to the background (Xie, Lee, Liu, Kotani, & Chen, 2020). An example is shown in Figure 2.1, it mainly consisting of three parts: sky, background, and ground. Each part can contain multiple objects, such as background contains building, trees, and ground contains dry grass, rough road, etc. In order to understand the scene in digital computer, the scene image must be in digital.

Figure 2.1: An Outdoor scene.

Digital image can be represented in gray-scale and color. A gray-scale digital image is a 2D function, $f(i,j)$, $i = 0,1,2,\dots,M-1$, $j = 0,1,2,I,N-1$, M is number of rows and N is number of columns, where (i,j) are discrete coordinates, and $f(i,j)$ is intensity value, or gray-level of the image at that pixel (i,j). In 8-bit gray-scale image, the range of the pixel values is 0,1,2,…,255: 0 represents the black and 255 represents for white color (Gonzalez, Woods, & Eddins, 2003). An example is shown in Figure 2.2 (a), in which the value of the image at any coordinates (i,j) is shown $f(i,j)$, i and j are integers, $i = 0,1,2,\dots M-1, j = 0,1,2..N-1$. More detail of gray-scale image is given in (Gonzalez & Woods, 2006), p.78. Some programming languages (e.g., MATLAB) have starting index one instead of zero.

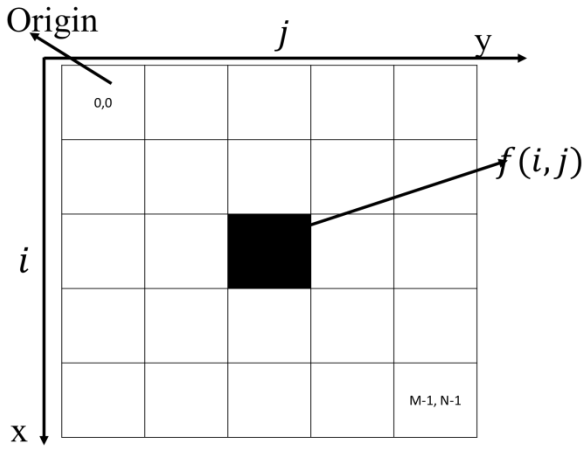
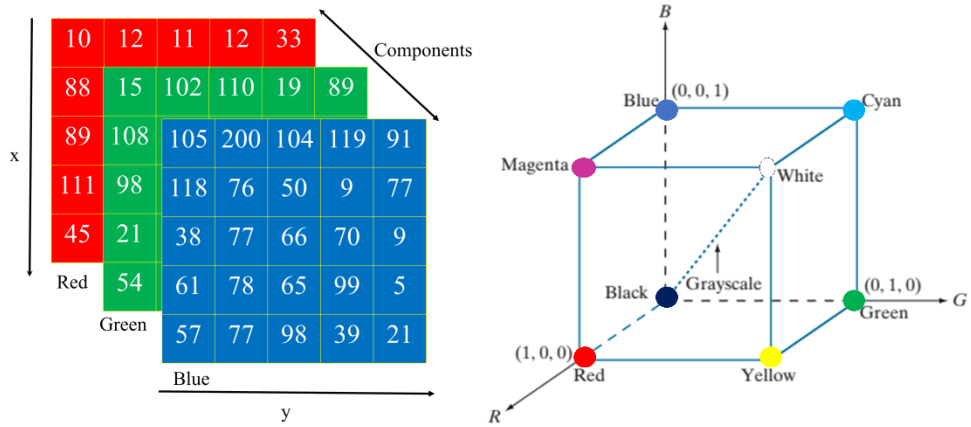


Figure 2.2: Gray-scale digital image of size 5×5 pixels. The positive x-axis increases downward and the positive y-axis increases to the right direction. M and N are row and column number of the image.

The digital color image is represented in different color spaces, such as RGB (Red, Green, Blue) (Figure 2.3(a)), HSV (hue, saturation, value), and CMY (Cyan, Magenta, Yellow) (see Figure 2.3 (b), etc., (Gonzalez et al., 2003). Color space is a "specification of a coordinate system and a subspace within that system where each color is represented by a single point" (Gonzalez et al., 2003), p.423). This study uses RGB and HSV color spaces only. The RGB image is represented in 3-dimensional array $I(M, N, 3)$ where M is number of rows, N is number of columns, and 3 is number of components of RGB. Each pixel $I_{i,j}$ of an RGB image has three components: red, green, and blue, which is represented by $I(i, j, \mathrm{k})$, where $i = 0,1,2, \dots, M-1, j = 0,1,2 \dots, N-1$, and k= 1,2,3, as Figure 2.3 (a) illustrates. Red component is represented by $I(i, j, 1)$, green component is represented by $I(i, j, 2)$, and blue component is represented by $I(i, j, 3)$, belonging to $0,1,2 \dots 255$.



(a) RGB image of size 5 × 5 pixels (b) Color cube model

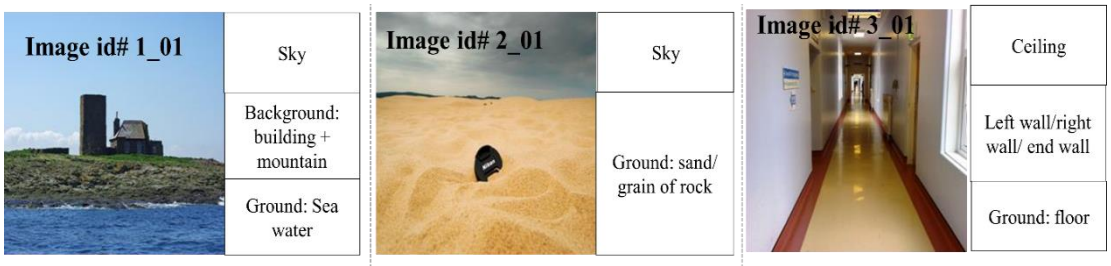
Figure 2.3: Color Image. (a) is RGB image of size 5 × 5 pixels. The color cube model in (b) represents different color components in the range of [0, 1] (Gonzalez et al., 2003).

The HSV (hue, saturation, value) color space separates *luma*, or the image intensity from *chroma* or the color information, which is very useful in many applications (Gonzalez et al., 2003). Hue is a color component that represent a pure color (pure yellow, orange, or red), whereas saturation (chroma) provides a measure of the degree to which a pure color is diluted by white light (Gonzalez & Woods, 2001). Value or brightness is an achromatic notion of intensity and is one of the key factors in describing the color sensation (Gonzalez & Woods, 2001). More detail about HSV is given in (Gonzalez et al., 2003), p.420, and HSV components are visualized in (Szeliski, 2011), p. 91. The HSV components can be calculated by MATLAB function, $[h, s, v] = rgb2hsv(I)$, where I is an RGB image and $h, s, and\ v$ are estimated HSV components.

2.1.1 3D Scene Recognition

When a human views the scene for a short time, he extracts enough visual information to accurately recognize its functional and categorical properties such as people in street, surrounded by tall building (Oliva & Torralba, 2001b), p. 146. So, in digital computer, scene recognition is one of the hallmark tasks, which allows definition of a

context for object recognition (Bolei Zhou, Lapedriza, Xiao, Torralba, & Oliva, 2014), p. 487. "Scene recognition not only describes existing objects but also the semantic relation between objects and the contextual information with respect to background" (Xie et al., 2020), p. 1, (Sect. 1, paragraph 2). It is widely used in many computer applications such as intelligent robotics, autonomous driving, and video surveillance. Consequently, 3D impression of these scenes as human can understand from the world can be categorized according to different image geometries (Nedovic et al., 2010; Oliva & Torralba, 2001b), which are called 3D scene geometries ('*stages*') (more detail about scene geometries are given in next section 2.1.1) and identification of images belonging to these scenes geometries is called 3D scene recognition. For instance, some examples of categories are shown in Figure 2.4, such as sky-background-ground in Figure 2.4(a), sky-ground in Figure 2.4(b), and corridor in Figure 2.4(c), and their names (image id#) are shown at the top of the image.



(a) Scene: sky-background-ground (b) Scene: sky-ground (c) Scene: corridor

Figure 2.4: Examples of scene recognition.

To recognize the 3D scene, a computer program requires the following fundamental steps as described in the flowchart, Figure 2.5. Each step is labeled with number. First we give a short overview of 3D scene recognition system and then detail of the each part is given in Subsections 2.1.2-7. Firstly, it inputs N training images, M testing images, 3D scene geometries (S(n)), n is the number of geometries, training labels

($YTR(N)$), testing labels ($YTS(M)$) as input to the computer program. These labels are names of the categories. Next, the training and testing images are pre-processed e.g. image segmentation (Lou et al., 2015) and then features are extracted for the both training and testing images. In feature extraction process, the particular discriminative information is extracted from each input image (G. Kumar & Bhatia, 2014). After extracting the features, the training images' features with their labels (3D scene geometries) are used for training the machine learning algorithm (MLA). MLA can be defined as a pattern recognition technique, which categorizes a huge numbers of data into limited classes (Faruk Ortes, Derya Karabulut, & Arslan, 2019). After training a machine learning model, the testing images' features are used to evaluate the trained model (more detail of training and testing steps are given in next Subsections). Thus, each testing image's features are given to trained model and this model predicts the 3D scene geometry of each input image. The predicted 3D scene geometry and given testing label of input image will be compared to calculate the performance. If both labels are same, it means system predicts an accurate 3D scene geometry of input image. Similarly, the predicted categories of all testing images are matched with given labels (3D scene geometries of testing images) and average accuracy and other metrics can be calculated over the M testing images. The formal description of 3D scene recognition in the form of pseudo code is given in Algorithm 2.1 and flowchart is given in Figure 2.5.

Algorithm 2.1: 3D Scene Geometry Recognition system

Input: *N* training images TRI(N), M testing images TSI(M), S(n) geometry classes, *YTR*(N) training images labels, *YTS* (M) testing images labels

Output: Performance Measures: Accuracy, Acc; Precision, Pr; Recall, Re; F-score

// Pre-processing and feature extraction steps

1: **for** j=1:N **do** // for each training image TRI_j

2: TRI'_j= Apply pre-process step of each input image TRI_j

3: F_j^{train}= Extract features for each input image TRI'_j

4: **end for**

5: **for** j=1:M **do** // for each testing image TSI

6: TSI'_j= Apply pre-process step of each input image TSI_j

7: F_j^{test}= Extract features for each input image TSI'_j

8: **end for**

// Training step

9: $X' = \{(F_j^{train}, YTR_j)\}_{j=1}^{N}$ // N is a number of training samples, YTR_j label of image I_j.

10: $Model = MLA(X')$, // X' is used to get trained model 'Model' by MLA

// Testing step

11: **for** j=1: M **do** // loop on test images TSI

12: $s_j = Classify(Model,\ F_j^{test})$ // s_j is predicted label (geometry) for the image TSI_j.

13: **end for**//j

//Performance measures

14: [Acc, Pr, Re, F-score]=Performance Measures $((s_j, YTS_j)_{j=1}^{M})$ //YTS_j is the true label of the test image TSI_j

End Algorithm

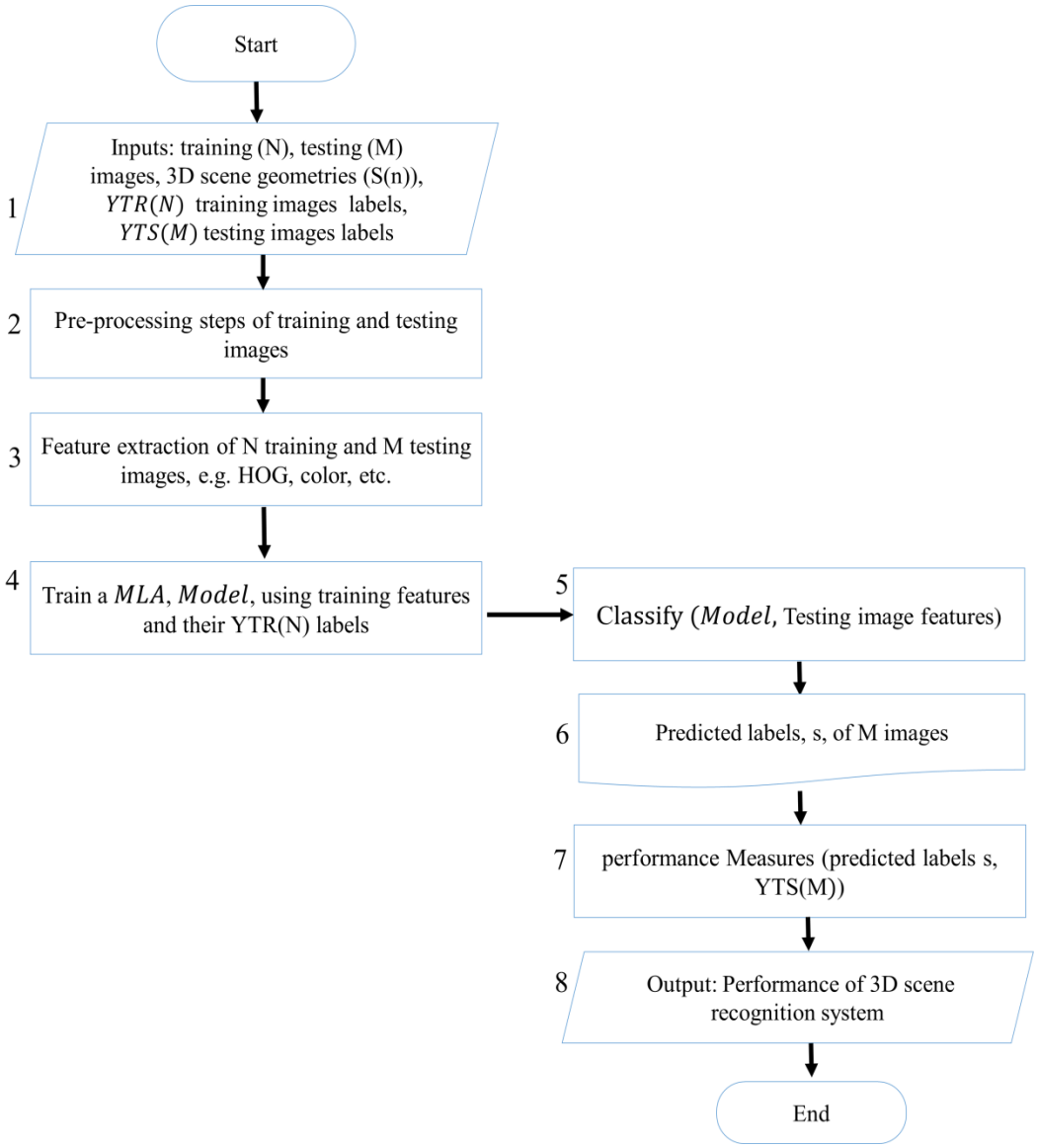


Figure 2.5: Flowchart of 3D scene recognition.

The steps of flowchart of 3D scene recognition are labeled with a number from 1 to 8. The pre-processing and feature extraction steps for both training and testing images are same. Therefore, they are labeled with same numbers. Meanwhile, Algorithm 2.1 is illustrating the pseudo code of the 3D scene recognition and each line of this pseudo code and flow chart steps are described in Subsections 2.1.2-7. The step 1 of flow chart is described in Subsection 2.1.2. Step 2 is discussed in Subsection 2.1.3 (lines 2 and 6 of Algorithm 2.1). Step 3 is described in Subsection 2.1.4 (lines 3 and line 7 of Algorithm 2.1). Step 4 is described in Subsection 2.1.5 (lines 9 and 10). Steps 5-6 are

described in Subsection 2.1.6 (lines 11-13). Steps 7, and 8 are explained in Subsection 2.1.7 (line 14).

2.1.2 Inputs of 3D Scene Recognition

According to Figure 2.5 and Algorithm 2.1, the 3D scene recognition system has inputs of N training images, M testing images, S(n) categories, n is the number of categories, *YTR(N)* training images labels, and *YTS(M)* testing images labels. For 3D scene recognition, the images are labeled according to 3D scene geometries. The scene geometry concerns depth, shape, and pose (D. Hoiem, Efros, & Hebert, 2006; Jung & Kim, 2012; Nedovic et al., 2010). Some researchers (D. Hoiem et al., 2006; Jung & Kim, 2012; Nedovic et al., 2010) represent world scene images as 3D scene geometries. Each 3D scene geometry covers several images due to structure regularities in the physical world (Nedovic et al., 2010). This is beneficial to narrow down the scene recognition task and reduce the computational complexity (Jung & Kim, 2012). For instance, Figure 2.6 shows capturing a 2D image from Physical 3D world and examples of scene images corresponding to 3D scene geometry are also given in Figure 2.6. Here, 3D world is represented as a rough 3D scene geometry. Nedovic et al. (Nedovic et al., 2010) introduce twelve 3D scene geometries also called as 'stages' to represent the indoor and outdoor scene images, namely, sky-background-ground (*skyBkgGnd*), sky-ground (*skyGnd*), background ground (*bkgGnd*), *ground*, one side wall (*sidewallRL*), *box*, diagonal background (*diagBkgRL*), ground-diagonal background (*groundDiag BkgRL*), *corner*, table-personbackground (*tabPersonBkg*), person-background (*personBkg*), and no depth (*noDepth*). Thus, all indoor and outdoor scene images are divided into these twelve categories. After that, each image is labeled with the name of its corresponding category, such as Figure 2.4, each image has category name given in the caption and name of each image is shown at the top of the

image. Thus, N number of training and M number of testing images are used and their category types are also given to the system. Mostly, 80% images of each category are used for training and 20% images for testing.

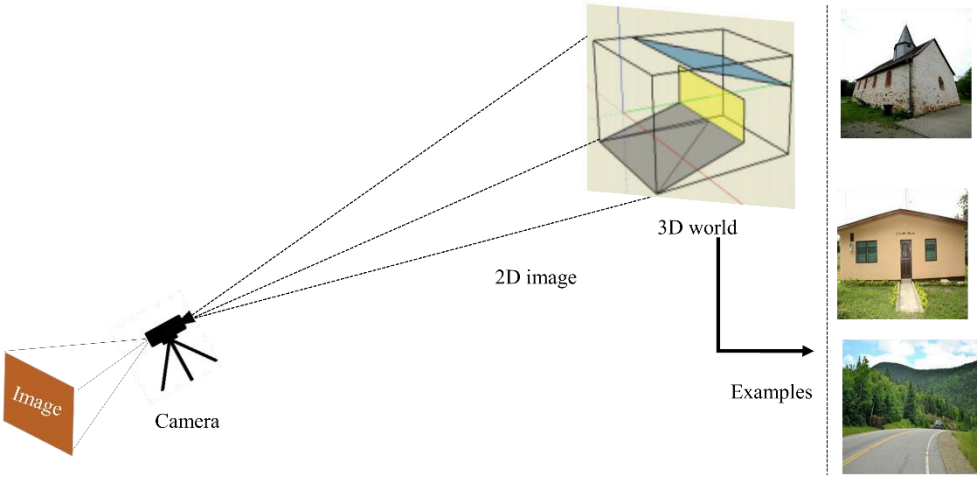


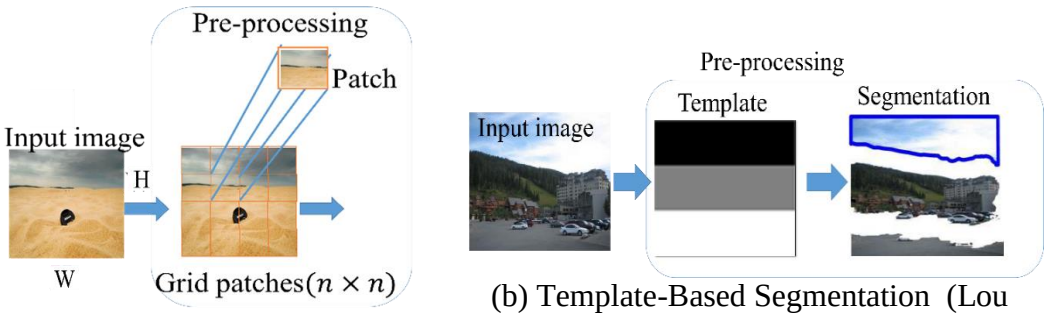
Figure 2.6: Generation of 2D image from 3D world.

2.1.3 Pre-Processing Steps of Feature Extraction

In Figure 2.5 (3rd step) and (lines 2, 6 of Algorithm 2.1), for each training TRI_j and testing images TSI_j ($j = 1,2..N$ for training and $j = 1,2..M$ for testing) is pre-processed by different ways, such as image segmentation (Lou et al., 2015), filtering (Paris, Hasinoff, & Kautz, 2011) or patches (Nedovic et al., 2010). And pre-processing images of training (TRI') and testing (TSI') are further used for feature extraction. Here, we will explore two existing pre-processing techniques of feature extraction. 1) Uniform grid based patches. 2) Template-based segmentation.

1) Some researchers (Nedovic et al., 2010; Oliva & Torralba, 2001b) divided the each input image into uniform grid patches. The patches are $n \times n$ parts of an image having height $h = \frac{W}{n}$ and width $w = \frac{H}{n}$, where H and W are the representing the height and

width of an image measured in pixels as it shown in Figure 2.7(a). The patch is shown with rectangle box corresponding to its position on the image 2.7 (a).



(a) Grid patches technique.

(b) Template-Based Segmentation (Lou et al., 2015).

Figure 2.7: Pre-processing steps of feature extraction.

2) Lou et al. (Lou et al., 2015) use predefined template to segment the image for pre-processing purpose. The template is a predefined 2D rough structure of 3D scene geometry (Lou et al., 2015). Each component of the template indicates corresponding subpart of 3D scene geometry. Therefore, these templates are beneficial to parse the image into subparts. For instance, in Figure 2.7(b), an input image is parsed into a template and its template-based segmentation are obtained: top is sky, middle is background, and bottom is ground as certain template has three components: top, middle, and bottom. After template-based segmentation, each segment available for feature extraction. The template can be defined as: template T, has the same size, $H \times W$ as the input image I. It is composed of ST segments, $s_k, k = 1,..,ST$. Element, $T_{ij} = k$, if pixel, $I(i, j)$ belongs to the segment s_k, e.g., in Figure 2.8 (a, b, c), templates have two components ($ST = 2$) and templates in Figure 2.8 (b, h, g), have three components ($ST = 3$). The template (f) is shown empty indicating that a whole image is considered as a one segment (more detail about template generation is given in Appendix A).

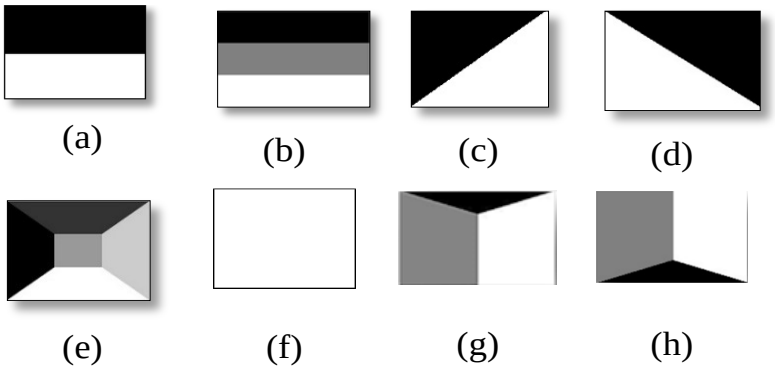


Figure 2.8: Examples of predefined templates (a) to (h).

Lou et al. (Lou et al., 2015) use soft and hard segmentation method to segment the input image. Hard segmentation follows a template components one-by-one, as it is shown in Figure 2.8. For example, a template in Figure 2.8(a) has two components. Thus, Image will be divided into two components and features will be extracted by following these segments. For soft segmentation, Lou et al. (Lou et al., 2015) use Carrira and Sminchinsescu (Carreira & Sminchisescu, 2012) segmentation method. In this method, the foreground seeds are placed on the image at uniformly grid points, and the background seeds are set at the boundary of the image. It generates several segments from each input image, the more detail is given at (Carreira & Sminchisescu, 2012). They select the segments which have the largest overlap-to-union score for feature extraction. However, this technique generates hundreds of segments for each image and to select the segment that has largest overlap-to-union score is an expensive task as it requires to compare each template component to each segment. In contrary, the active contours algorithm (Chan & Vese, 2001) can be used for template-based segmentation. It generates a single accurate segment if a template component is used as an initial contour. Active contours algorithm is a segmentation technique which uses the energy constraints and forces in the image for separation of the region of interest. It separates boundaries for the regions of target object for segmentation. The region of interest possesses a group of pixels such as circle, polygon or irregular shapes, the

more detail is given at (Chan & Vese, 2001). In MATLAB tool, the function *activecontour(I, mask)* indicates the method. It segments the image, *I*, into foreground and background regions using active contours. The mask indicates the initial contours (e.g. template component) for an image *I*. An example is shown in Figure 2.9, where a scene image is parsed with a predefined template 2.8(b) and, using active contours algorithm, the input image is segmented into three parts, namely, sky, background, and ground.

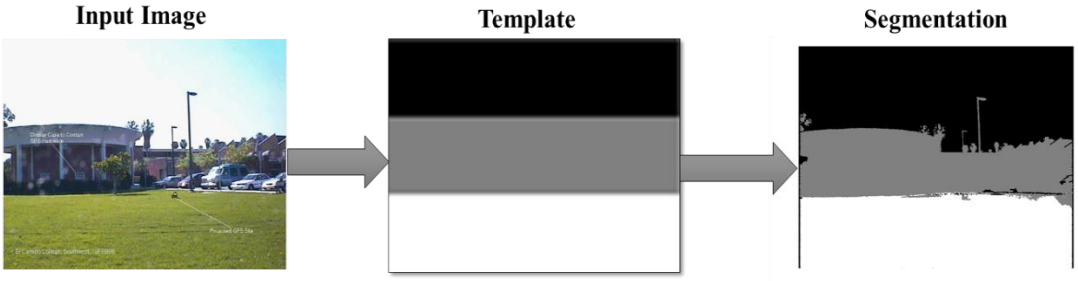


Figure 2.9: An Example of active contours algorithm's segmentation using predefined template.

2.1.4 Feature Extraction

Next, the 3rd step of the Figure 2.5 is the feature extraction (line 3 and 7 of Algorithm 2.1). The feature is the discriminative information of an image extracted, e.g., by using convolutional operation on the input image (Gonzalez & Woods, 2001). Convolutional operation is the mathematical operation on two functions (A and B); it yields the result that represents the effect of B on function A (Gonzalez & Woods, 2006). In Figure 2.10, a patch of an image having size 5x5 pixels, and 3x3 size of conv. function (filter) are applied at one pixel of an image. For example the red pixel (middle) has value '3', and after applying the 3x3 convolutional filter on this pixel, which gets value '5'. A

convolutional filtering is the process of moving a filter mask over the image and computing the sum of products at each location. It is defined as a filter $w(x, y)$ of size $m \times n$ with an image $I(x, y)$, denoted as $w(x, y) * I(x, y)$, is given by the equation (Gonzalez & Woods, 2006),

$$w(x, y) * I(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) \times I(x - s, y - t), \quad (2.1)$$

where a=(m-1)/2, b=(n-1)/2, and m and n are odd integers. '*' is convolutional operator. The detailed of convolutional operation is given in (Gonzalez & Woods, 2006), pp. 168-171.

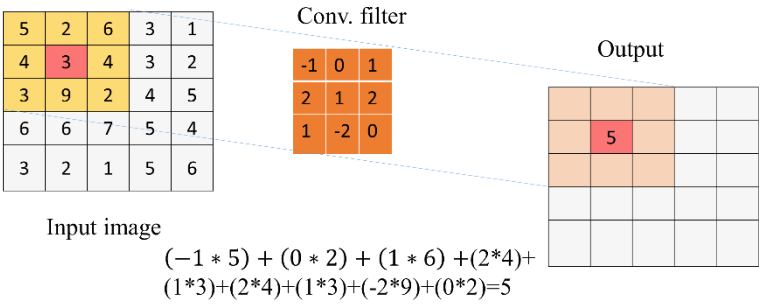


Figure 2.10: Convolutional operation on a 2D image of size 5x5 pixels.

In existing approaches of 3D scene recognition, such as (Lou et al., 2015; Nedovic et al., 2010), the features are extracted from each image patch, and, after that, they are concatenated to obtain a single feature vector. It is visualized in Figure 2.11: the small rectangles are connected into a single vector. Assume x_i^j is a feature vector for patch i of j-th input image. The concatenated feature vector for j-th image can be written as, $F^j = [x_1^j, x_2^j, \dots, x_{n\times n}^j]$, where $n \times n$ is the number of patches of j-th input image. For N training images, the feature vector, F_j^{train}, is extracted for each input train image, TRI'_j (see line 3 of Algorithm 2.1), $j = 1,2,..N$. Similarly, for M testing images, the

feature vector, F_j^{test}, is extracted for each input test image TSI'_j (see line 7 of Algorithm 2.1), $j = 1,2, \ldots M$.

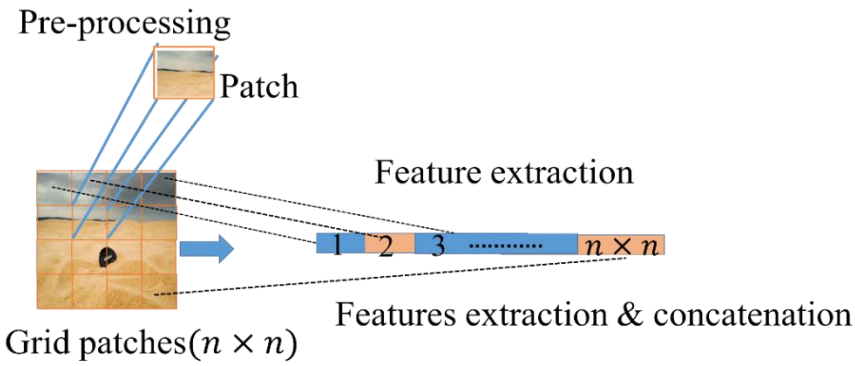


Figure 2.11: General representation of feature extraction and concatenation.

Different types of image features can be extracted for 3D scene recognition, which provide discriminating information to distinguish one image scene from other scenes (Lou et al., 2015; Nedovic et al., 2010), namely parameters of Weibull distribution (J.-M. Geusebroek & Smeulders, 2005), color (Nedovic et al., 2010), HOG (Dalal & Triggs, 2005), and Local binary pattern (Ojala, Pietik, & Maenpaa, 2002). The description of the features is given below in Subsections 2.1.4.1-5, respectively.

2.1.4.1 Parameters of Weibull Distribution

The relation between scene depth and image statistics is studied in (J.-M. Geusebroek & Smeulders, 2005; Nedovic et al., 2010; Torralba & Oliva, 2002). Nedovic et al. measure the depth of scene image based on *stage* types. "Nedovic et al. (Nedovic et al., 2010) show that parameters of the Weibull distribution are informative to capture local depth ordering" (Lou et al., 2015), p.3103 (background of Weibull distribution is given in Appendix B). Therefore, the parameters of the Weibull distribution can be used as scene features (Lou et al., 2015). They measure the parameters (α (shape), and β (scale)) of Weibull distribution for each image patch (four features). "Parameters of the distribution are derived using maximum likelihood

estimator (MLE)" (Duffy, 1997) (Nedovic et al., 2010), p. 1677. The MATLAB provides the default function of MLE, namely, $[parameters, \sim] = wblfit(h)$, where 'h' is a sample data and function, $wblfit(.)$ returns the estimates of Weibull parameters (α, β). For example, h = [0.05, 0.75,0.1, 1.0, 1.5, 2.0], then the estimates of Weibull parameters, $\alpha = 0.99, \beta = 0.89$. Nedovic et al. (Nedovic et al., 2010), p. 1677, describe that Gaussian derivative filters (x and y-directions with $\sigma = 3$) are used (as convolutional operators (equation (2.1))) to extract the texture information from each image patch. Gaussian function is defined with standard deviation, σ, as (Gonzalez & Woods, 2018), p.724,

$$G(x, y)=e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{2.2}$$

So, the derivatives, with respect to x and y: $\frac{\partial G(x,y)}{\partial x}$, $\frac{\partial G(x,y)}{\partial y}$, are calculated as (Gonzalez & Woods, 2018):

$$\frac{\partial}{\partial x}(G(x,y)) = -\frac{x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{2.3}$$

$$\frac{\partial}{\partial y}(G(x,y)) = = -\frac{y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{2.4}$$

The more detail of x and y-derivative filters are explained in (Gonzalez & Woods, 2018), pp. 716-731. Let $I(x, y)$ denote input image, then the $I_x(x, y)$ and $I_y(x, y)$ are generated by convolving I(x, y) with $\frac{\partial G(x,y)}{\partial x}$ and $\frac{\partial G(x,y)}{\partial y}$, respectively:

$$I_x(x, y) = \frac{\partial G(x,y)}{\partial x} * \mathrm{I}(\mathrm{x}, \mathrm{y}), \tag{2.5}$$

$$I_y(x, y) = \frac{\partial G(x,y)}{\partial y} * \mathrm{I}(\mathrm{x}, \mathrm{y}), \tag{2.6}$$

where, '*' is a convolutional operator defined in (2.1). After that, the histograms, hx and hy are calculated for $I_x(x, y)$ and $I_y(x, y)$, respectively. The histogram is generated by a complete set of non-overlapping intervals, called bins, and the number

of points in each bin is counted (Scott, 1992). The bins should all have the same width in order for the bin counts to be identical. Then, two parameters, the bin width, w, and the bin origin, t_0, are fully determined by the histogram. Often, the bin origin is chosen to be t_0=0 (Scott, 1992), p.52. More detail of histogram and its bins calculation is given in chapter 3 of (Scott, 1992). In implementation, it can be computed using Matlab function, *histogram (H, b)*, H is an input data and 'b' is the number of bins. The size of bin is based on input data and this function calculates the width of the each bin and then distributes the data over the number of bins accordingly. After obtaining the histogram, before giving it into $wblfit(.)$ function to estimate the Weibull parameters, its components are first normalized by dividing each of its components by the total number of pixels in the patch, denoted by the product '$hw = h \times w$', where h and w are the row and column dimensions of the patch. The normalized histogram is obtained by dividing each component of the histogram to number of elements. It is defined as (Gonzalez & Woods, 2018; Scott, 1992),

$$f(b) = \frac{b_k}{hw}, \tag{2.7}$$

where b_k is the number of data points falling in the k-th bin. For example, H = [2, 1, 3,14, 17,3,4,5,6,5,8,9,12,24,25], and assume bins, b, set to 6, then the components of the histogram are: [5,4,2,1,1,2], where the width of each bin is 4.20. After it, the normalized components of the histogram, [0.333, 0.267, 0.133, 0.067, 0.067, 0.133], are computed by using (2.7). The normalized components are used as input to Matlab function, $wblfit(.)$ to estimate Weibull α, and β parameters. The relation between histogram and parameters of Weibull distribution is shown in Appendix C. Thus, the 4 features (α and β for x and y derivatives) are obtained for each patch as a feature set. The change of α and β parameters with respect to depth are visualized in Appendix D.

2.1.4.2 Color Features

The properties of a light source and colors of scene objects can be used for stage classification (Lou et al., 2015; Nedovic et al., 2010). Two different color spaces, RGB and HSV, are used as feature sets for each image patch. Nedovic et al. (Nedovic et al., 2010) use the three features of the color correction coefficients of RGB estimated by a Gray-World algorithm (Weijer, Gevers, & Gijsenij, 2007) and the other three features, Hue, Saturation and Value (HSV), are included in a color feature set for each image patch. Color correction is an estimation of the illumination of the color, which encodes the properties of the light source (Nedovic et al., 2010), p. 1677. Nedovic et al. show that it improves the stage recognition accuracy. Gray-World algorithm is simplest color correction estimation method, "which assumes that the average reflectance of the surfaces in the world is achromatic" (Nedovic et al., 2010), p. 1677. The basic description and implementation is given in (Weijer et al., 2007), pp.2208-9, formula is given in (10). Matlab package of Gray-World algorithm is available in (Joost van de Weijer , Theo Gevers , & Gijsenij, 2007). The main function, $[W_R, W_G, W_B, \sim] = general\_cc(RGB\ image, 0,1,0)$, takes an RGB image with it default parameters ((0, 1, 0), which indicates differential order, L1-norm, and sigma, respectively) as inputs and generates its color coefficients, W_R for red, W_G for green, and W_B for blue. Here, we use color correction coefficient as features. The HSV components can be calculated by MATLAB function, $[h, s, v] = rgb2hsv(I)$, where I is an RGB image and $h, s, and\ v$ are estimated HSV components. Therefore, six color features are obtained for each image patch.

2.1.4.3 Histogram of Oriented Gradients Feature

The histogram of oriented gradients (HOG) is one of the most popular features and has been widely used in an object detection for representing the shape of objects (Dalal &

Triggs, 2005; Tomasi, 2012). It is available as a Matlab function, $[features] = extracthogfeature(I, parameters)$, where, I is an input image and $features$ representing the output of HOG feature vector. Parameters contain number of histogram bins, normalization type (i.e. L2-norm), and constant ϵ value. We modify the existing code by extracting the 9 features (setting bins=9) for each image patch as Lou et al. (Lou et al., 2015) used it. Instead of dividing the image into blocks, we use the patch as a single block for feature extraction. The HOG feature is useful for differentiating the image geometry such as the shape of the sky–ground and the shape of the corner are very different (Lou et al., 2015). The HOG feature can be calculated by following main steps as described in (Dalal & Triggs, 2005): gradient computation, orientation binning, and bin vector normalization. First step is gradient computation. The gradient at arbitrary location (x, y) of an image,I, denoted by ∇I and defined as the vector (Gonzalez & Woods, 2018), p. 716:

$$\nabla I(x,y) = grad[I(x,y)] = \begin{bmatrix} I_x(x,y) \\ I_y(x,y) \end{bmatrix} = \begin{bmatrix} \frac{\partial I(x,y)}{\partial x} \\ \frac{\partial I(x,y)}{\partial y} \end{bmatrix}, \tag{2.8}$$

where, $\frac{\partial I(x,y)}{\partial x}$ and $\frac{\partial I(x,y)}{\partial y}$ are partial derivatives at each pixel location in the image. The magnitude M(x, y) of this gradient vector at a point (x, y) is given by (Gonzalez & Woods, 2018),

$$M(x,y) = \|\nabla I(x,y)\| = \sqrt{(I_x(x,y))^2 + (I_y(x,y))^2}\,, \tag{2.9}$$

This is the value of the change rate at the point in the direction of the gradient vector (x, y). The direction of the gradient vector at a point (x, y) is calculated by (Gonzalez & Woods, 2018),

$$\theta(x,y) = \tan^{-1}(I_y(x,y)/I_x(x,y)), \tag{2.10}$$

Where the angle, θ, is determined with respect to the x-axis in the counterclockwise direction. More detail is given in (Gonzalez & Woods, 2018), p. 716. Instead of using

the derivatives computed in equations (2.5) and (2.6), typically forward or centered finite difference (Milton Abramowitz & Stegun, 1972) is used to calculate the derivative (Gonzalez & Woods, 2018), pp. 717-18) (Dalal & Triggs, 2005), formula is given in Appendix E. Using forward differences, we obtain (Gonzalez & Woods, 2018), p. 718):

$$I_x(x,y) = \frac{\partial I(x,y)}{\partial x} = I(x+1,y) - I(x,y) \tag{2.11}$$

$$I_y(x,y) = \frac{\partial I(x,y)}{\partial y} = I(x,y+1) - I(x,y) \tag{2.12}$$

For all values of x and y, these two equations can be applied by filtering $I(x,y)$ with one dimensional (1D) kernels ($[-1,1], [-1,1]'$, see Figure 10.13 in (Gonzalez & Woods, 2018)). The different types of kernels are studied in (Dalal & Triggs, 2005; Tomasi, 2012). However, the best performance is obtained by convolution of 1D kernel, [-1, 0, 1] (Dalal & Triggs, 2005), p. 889. Convolutional function is given in equation (2.1). For RGB images, the gradients of each color component are calculated separately, and the one with the largest value is taken as the pixel gradient (Dalal & Triggs, 2005). In the next step, the histogram with orientation bins (B) is calculated for each local region (patch). The standard definition of histogram is defined in previous Subsection 2.1.4.1. Each pixel within the patch cost a weighted vote for an orientation-based histogram on the value computed in the gradient calculation (Dalal & Triggs, 2005), p. 889. "The vote is a function of the gradient magnitude at the pixel, either the magnitude itself, its square, its square root, or a clipped form of the magnitude representing soft presence/absence of an edge at the pixel. In practice, using the magnitude itself gives the best results" (Dalal & Triggs, 2005), p. 889. "The orientation bins are evenly spaced over 0° – 180°($0 - \pi$, unsigned gradient) or 0º – 360º (0-2π, signed gradient)" (Dalal & Triggs, 2005), p. 889. The magnitude of each pixel is assigned to bin on the base of its orientation that in which bin it exist. In

implementation, (Dalal & Triggs, 2005) shows that unsigned gradient gives the best results. After obtaining a bin vector '*V*' (with a dimension *B*, *B* is set to 9 by (Dalal & Triggs, 2005)) for a single image patch. It is normalized by its 'L2- norm'. L2-norm is calculated as: Let V be the un-normalized descriptor vector, $\|V\|_2$, $\|V\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$, where $v_1, v_1, .., v_n$ are elements of vector with n length, be its L2-norm (more detail is given in (Weisstein & W., 2020)), and ϵ be a small constant (Dalal & Triggs, 2005), p. 891. It is defined as (Dalal & Triggs, 2005):

$$V = \frac{V}{\sqrt{\|V\|_2^2 + \epsilon^2}}. \tag{2.13}$$

In Matlab, it can be calculated by $V' = normalize(V, 'norm')$, where, norm is parameters which indicating the L2-norm calculation. Note: ϵ is used to prevent division by zero. It is equal to 0.01 to minimize its influence on HOG features. In this way, nine features are extracted for each image patch.

2.1.4.4 Local Binary Pattern and Entropy Value Features

One of the most successful approaches to texture description is Local Binary Pattern (LBP) and its variants (Ojala et al., 2002). It is explained in (Ojala et al., 2002). It obtains invariant, uniform representations, and rotation, important for many applications, such as face recognitions, texture recognition, and remote scene classification, etc. (Pietikäinen & Zhao, 2016), due to its discriminative power and computational simplicity (Pietikäinen, Hadid, Zhao, & Ahonen, 2011). The Matlab code of converting the RGB image into LBP code is available at (Nikisins, 2020), in a form of function, $LBP_{output} = LBP(I, R)$. It has two parameters, input RGB image, I, and radius R. The possible value of the R=1, 2, 3., etc. By default, R=1, is used. R=1 means eight neighbor pixels and R=2 means 16 neighbor pixels around the center pixel (more detail is given in (Ojala et al., 2002)). The LBP_{output} is a 2D array of same size

of input image *I*. Then the histogram is constructed for LBP_{output}. The histogram is defined in Subsection 2.1.4.1. Here, the Matlab function, $histogram(LBP_{output}, b)$ is used, b is number of bins. Then the components of the histogram are normalized as (2.7). Thus, the normalized components of dimension, b, are used as a feature vector for each image patch.

Entropy (E) is another approach to describe the texture content (Gonzalez et al., 2003). It is a function which measures the variability of data and it approaches to zero for a constant input data (Gonzalez et al., 2003). "Given a source of statistically independent random events from a discrete set of possible events $\{a_1, a_2, \dots, a_J\}$ with associated probabilities $\{p(a_1), p(a_2), \dots, p(a_J)\}$, the average information per source output, called the entropy of the source, is" (Gonzalez & Woods, 2018), p. 546,

$$E = -\sum_{j=1}^{J} p(a_j) \log_2 p(a_j), \tag{2.14}$$

where a_j is called as a source symbol. E is an entropy of the source. The more detail is explained in (Gonzalez & Woods, 2018), p. 546. Different researchers use it as features for scene classification, such as (Derek Hoiem et al., 2007) utilize it for texture features. In Matlab, entropy is available in a function, $E = entropy\ (x)$, where x is an sample data or gray-scale image and its output, E, is a real value. Each 3D scene geometry represents the different texture information of the image scene (Nedovic et al., 2010). Entropy measures the variability of data (Gonzalez et al., 2003), therefore, it can be used as a feature for 3D scene geometry classification. The LBP_{output} contains texture information, thus, it can be used for measuring the entropy value. For this, normalized histogram components are used as input of entropy function. Thus, we compute the LBP-E features for each patch. (As these features was not study for

3D scene geometry, we have shown the performance of these features in our experiment (see Chapter 3, Table 3.2)).

2.1.5 Training of Machine Learning Algorithm

After extracting the feature vector, the next point from Figure 2.5 is 'train a machine learning algorithm, 'MLA' (lines 9, 10 of Algorithm 2.1). Suppose that we have training dataset with N samples (F_j^{train}, TRY_j), where F_j^{train} is a feature vector for image TRI_j, and TRY_j is the class label. $TRY_j \in \varphi = \{\omega_1, \omega_2, \dots, \omega_n\}$, ω_i denotes the class such as sky-background-ground, $i = 1,2..n$, n is number of classes. Then, the feature vectors with labels for N samples can be given as,

$$X' = \{(F_j^{train}, TRY_j)\}_{j=1}^{N}. \quad (2.15)$$

Then the features with labels, X', are used as input to a machine learning algorithm, MLA (X') to train a model, $Model$ (line 10 of Algorithm 2.1). It generates a single trained model, 'Model' for N labeled data. The model can be used as, $classify\ (model, test\_image)$. Classify(.) is a function which predicts the class type of an input image, $test\_image$ (line 12 of Algorithm 2.1). The detail of testing images and prediction is given in the next Section 2.1.6.

However, the most useful MLA of scene classification is a SVM (Cortes & Vapnik, 1995; Nedovic et al., 2010). Recently, the extreme learning machine (ELM) also becomes a famous classifier because of its satisfactory generalization performance (Guang-Bin, Qin-Yu, & Chee-Kheong, 2004; G.-B. Huang, Zhu, & Siew, 2006; G. Huang et al., 2015b). The SVM (Cortes & Vapnik, 1995), ELM (G. Huang et al., 2015b), and ensembles of classifiers, are introduced in Subsections 2.1.5.1-3, respectively.

2.1.5.1 Support Vector Machine

Support vector machine (SVM) (Cortes & Vapnik, 1995) has been commonly employed in objects classification, image segmentations, and scene classification tasks (Cortes & Vapnik, 1995; Mohandes, Deriche, & Aliyu, 2018; Somvanshi, Chavan, Tambade, & Shinde, 2016). The SVM's key concept is to learn the maximal margin parameters of the hyperplane to distinguish two groups on a training set. In MATLAB-2019a, SVM is available as a function, *fitcecoc (.)*. The fitcecoc function takes feature vectors and class labels and returns a fully trained model using $S(S-1)/2$ binary SVM models, where *S* is the number of unique class labels. The interface of the function is *Model = fitcecoc(x, y, 'learners', t)*, where *x* is the input feature matrix of the training images, and *y* is a categorical vector of class labels. '*Learners*' indicates the SVM classifier kernel, such as 'linear,' 'polynomial', 'Gaussian', etc., detail of the SVM kernel is given in (Anguita, Boni, Ridella, Rivieccio, & Sterpi, 2005). The *t* is an optional parameters that specifies the properties of classifiers such as '*linear*', '*cross-validation*', etc. The '*Model*' indicates the trained model, which is used to classify the test data.

2.1.5.2 Extreme Learning Machine

The extreme learning machine (ELM) is "originally proposed for generalized single-hidden layer feedforward neural networks" (G. Huang et al., 2015b), p.18. Its learning speed can be thousands times faster than traditional feedforward network learning algorithms like back-propagation (BP) algorithm and obtains better generalization performance (Guang-Bin et al., 2004), p. 985. Guang-Bin et al, (Guang-Bin et al., 2004), p. 985, note that it reaches to the smallest training error, runs extremely fast, and in order to differentiate it from the other popular learning algorithms, it is called the "Extreme Learning Machine (ELM)". "A layer in the network is the set of nodes

(neurons) in a column of the network" (Gonzalez & Woods, 2018), p. 945. Each layer in the network can have a different number of nodes, but each node has a single output. A single neuron is a basic unit of a neural network, often called a node or unit. It receives input from other nodes or external source and compute an output (Gonzalez & Woods, 2018; Ujjwalkarn, 2016). Each input has an associated weight (w) (Ujjwalkarn, 2016). The node applies an activation function, h, to the weighted sum of its inputs as given in Figure 2.12. In this Figure, the x_1 and x_2 are numerical inputs and has weights w_1 and w_2 respectively. Additionally, it has another input 1 with weight b (called the bias) associated with it. Y is output of the neuron (Ujjwalkarn, 2016), in the range of [0,1] or [-1,1]. The output, Y is obtained by activation function.

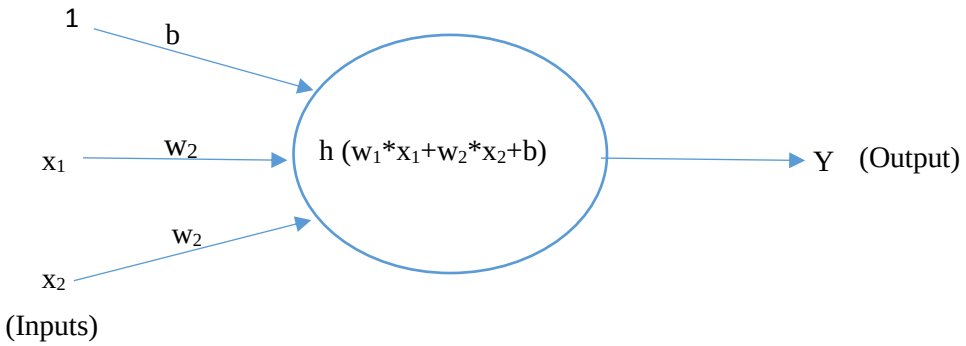


Figure 2.12: Sample of a single neuron.

The input vectors x transformation is given by (Gonzalez & Woods, 2018),

$$z(x) = w^T x + b \tag{2.16}$$

where w and x are n-dimensional column vectors and $w^T x$ is the dot product of the two vectors. The b is bias and z is result of the computation performance by the neuron. The equation is expressed in summation form as (Gonzalez & Woods, 2018),

$$z(x) = \sum_{i=1}^{n} w_i x_i + b \tag{2.17}$$

The component of a vector x having length n are: $x_1, x_2, \ldots, x_n, 1$ and component of

weight vector, w, are: $w_1, w_2, \ldots, w_n, w_{n+1}$, where w_{n+1} is the bias, represented by b in (2.16-17) (Gonzalez & Woods, 2018), pp. 935. The more detail is given in (Gonzalez & Woods, 2018), pp. 934-6. Activation function can be defined as: the output of the node denoted by a, is obtained by passing z through $h(.)$. $h(.)$ is the activation function, and refer to its output, $a = h(z)$, as the activation value of the node (Gonzalez & Woods, 2018), p. 545. The activation function is non-linear function which introduces non-linearity into the output of a neuron. This is important because most real world sample data is non-linear and neurons to learn these non-linear representations (Ujjwalkarn, 2016). The several activation functions are used in practice, such as sigmoid and Gaussian function. The sigmoid function takes real value, z, as input and generates the output in the range of [0, 1]. The sigmoid function, $h(.)$, is defined as (Gonzalez & Woods, 2018), p.944,

$$h(z) = \frac{1}{1+e^{-z}}, \tag{2.18}$$

where z is the results of the computation performed by the neuron. The e^{-z} is standard exponential function for input z. For example, x_1=1.2, x_2=1, w_1=0.2, w_2=0.5, and b=1, then the output of the (2.17) is become, $z = 1.2 * 0.2 + 1 * 0.5 + 1$, z=1.74. Then, the output, Y, of the neuron using (2.18) is, $\frac{1}{1+e^{-1.74}}$= 1/(1+0.175)= 0.850. Results of activation function are used to decide that input pattern is belonged to which category. The more detail is given in (Gonzalez & Woods, 2018), p.949.

The hidden layer/s is the layer between the input and output layers of the algorithm, in which the function applies weights to the inputs and direct them through the activation function as the output. Single hidden layer indicates only one layer between the input and output layers.

ELM provides efficient solutions for the applications of feature learning and image classification (G. Huang et al., 2015b), p.18. The ELM is described in (G.-B. Huang et al., 2006; G. Huang et al., 2015b). Implementation of the ELM is given at (G. Huang, Bai, Kasun, & Vong, 2015a). Generally, ELM algorithm requires the following parameters: '*training features', 'training labels', 'test features', 'testing labels', 'L', 'h',* and *'C*' value and returns the trained model, prediction of testing images, and accuracy (optional output). L is number of hidden neurons (neurons of hidden layers). The h indicates activation function. It uses 'sigmoid', 'tribas'(triangular basis), and 'Gaussian' (radial) as activation function, detail is given in (G.-B. Huang et al., 2006). C is the controlling parameter and its belong to (0.001,0.1,1,10,100) (G. Huang et al., 2015b).

2.1.5.3 Ensembles of Classifiers

Ensembles of classifiers increase the performance of pattern recognition applications (Kittler, Hatef, Duin, & Matas, 1998; Mohandes et al., 2018; Snelick, Uludag, Mink, Indovina, & Jain, 2005; Tulyakov, Jaeger, Govindaraju, & Doermann, 2008). Typical ensembles of classifiers take features vector (X') and generate score vectors. Then these outputs from individual classifiers are used to produce a combined output for each class, as it is shown in Figure 2.13.

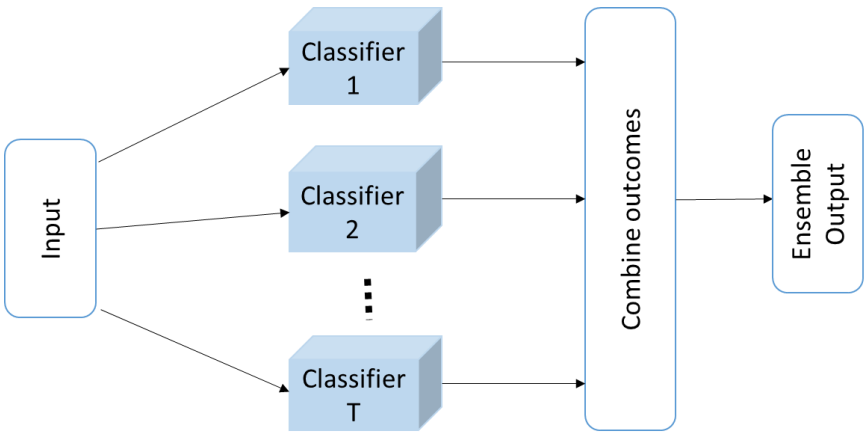


Figure 2.13: General model of ensemble of classifiers.

To fuse the outputs of classifiers, two kinds of fusion strategies can be used: soft-level or score-level and hard-level fusion (Mohandes et al., 2018). Hard-level fusion involves summing the predictions for each class label and predicting the class label with the most votes, which is called as majority voting. Soft-level fusion or score-level uses estimate of the aposteriori probability or scores of the categories. It involves summing the predicted probabilities or score for each class label and predicting the class label with largest probability. The different rules can be used for predicting the class label, such as sum, max, product, and min rules. "Product rule quantifies the likelihood of a hypothesis by combining the aposteriori probabilities generated by the individual classifiers by means of a product rule" (Mohandes et al., 2018), p. 19630. The sum rule simply adds the weight of scores or aposteriori probabilities provided by each classifier for each class, and derives the class label for input image having the maximal sum value (Mohandes et al., 2018). "The max rule is an approximation of the sum rule and takes the maximum of the aposteriori probabilities (Mohandes et al., 2018). In (Kittler et al., 1998; Mohandes et al., 2018; Snelick et al., 2005; Tulyakov et al., 2008), it is illustrated that the sum and product-rules are quite simple and have low error rate.

2.1.6 Testing Images and Classification

When a model is fully trained, the next step of 3D scene recognition is to evaluate or test the trained model (as shown in steps 5 and 6 of Figure 2.5). In testing model, suppose we have M input testing samples and their features, $F_j^{test}, j = 1,2,..,M,$ (line 7 of Algorithm 2.1) for each image TSI_j is given to the trained model, $Model$, (see lines 11-13 of Algorithm 2.1). The model predicts the class label, s_j, for each input image, TSI_j j=1,2..M (see line 12 of Algorithm 2.1 and step 5 of Figure 2.5), which is defined as,

$$s_j = classify\left(Model, F_j^{test}\right), j = 1,2,..M, \quad (2.19)$$

where s_j is predicted label of the input testing image TSI_j. M is test images. Predicted label s_j is denoting the 3D category type, such as sky-background-ground.

Meanwhile, model predict the aposteriori probability or score vector of the input image. The score vector or aposteriori probability distributed over the number of classes. The model can also predict $score_{jn}$ vector distributed over categories, n, for each image TSI_j. If the ensemble of classifiers is used then these scores from each classifier can be combined using soft-level fusion and then final 3D scene geometry can be predicted as it discussed in Subsection 2.1.5.3. For example, if T classifiers are used, then, the class label can be predicted by using sum rule,

$$s_j' = sum\_rule\left(score^t{}_{ji}\right)_{t=1}^{T}, i = 1,2..n, j = 1,2 \dots M, \quad (2.20)$$

where T is an ensemble of classifiers, $score_{ji}$ is predicted score of tth classifier for jth input image and for ith class. The n is number of classes. M is testing images, and $'sum\_rule(.)$' is indicating the sum rule to combine the outcome of ensemble of classifiers. The s_j' is a predicted label of classifiers combination for input image, TSI_j.

2.1.7 Performance Metrics

After testing, the last step (7) of Figure 2.5 is measuring the effectiveness of trained model. In order to evaluate the effectiveness, the different researchers use different calculation metrics. Most of the researchers calculate only accuracy of scene recognition, such as (Nedovic et al., 2010; Oliva & Torralba, 2001b) measure the accuracy of different classes. However, the state-of-the-art 3D scene recognition method (Lou et al., 2015) uses following metrics: confusion matrix (Ballabio, Grisoni, & Todeschini, 2018; Rosset, 2004), accuracy (Aghdam & Heravi, 2018; Lou et al., 2015), means precision (Rosset, 2004), means recall (Rosset, 2004), and means F-

score (Rosset, 2004). Let M samples are used in testing (line 14 of Algorithm 2.1). The n is the number of classes, M_c represents the number of samples truly belonging to the c-th class, while M'_c is the number of samples predicted belonging to the c-th class.

The classification results can be represented in the confusion matrix (Ballabio et al., 2018; Rosset, 2004). It is a square $n \times n$ matrix whose rows and columns represent true and predicted classes, respectively. Each its entry, G_{cg}, represents the number of samples belonging to c-th class and predicted as belonging to g-th class. The diagonal elements G_{cc} represent the number of correctly classified samples, while remaining elements represent the number of incorrectly classified samples. Confusion matrix contains all the information related to the distribution of samples within the class and the classification performance. The number of test samples M is (Ballabio et al., 2018; Rosset, 2004):

$$\mathrm{M} = \sum_{c=1}^{n} \sum_{g=1}^{n} G_{cg}. \tag{2.21}$$

The number of samples truly belonging to the c-th class is (Ballabio et al., 2018; Rosset, 2004):

$$\mathrm{M}_c = \sum_{g=1}^{n} G_{cg}. \tag{2.22}$$

The number of sample predicted in the c-th class (M'_c) is (Ballabio et al., 2018; Rosset, 2004):

$$\mathrm{M}'_c = \sum_{c=1}^{n} G_{cg}. \tag{2.23}$$

Accuracy, Acc, is defined as the total number of truly predicted samples over total number of samples in dataset (Aghdam & Heravi, 2018; Lou et al., 2015). The accuracy is defined as:

$$\mathrm{Acc} = \frac{1}{M}\sum_{c=1}^{n} G_{cc}. \tag{2.24}$$

Precision, Pr(c), of the c-th class is defined as (Rosset, 2004):

$$\mathrm{Pr}(c) = \frac{G_{cc}}{M_c'}. \tag{2.25}$$

then the average, Pr, is calculated by:

$$\mathrm{Pr} = \frac{1}{n}\sum_{c=1}^{n} \mathrm{Pr}(c). \tag{2.26}$$

Recall, Re (c), is calculated as (Rosset, 2004):

$$\mathrm{Re}(c) = \frac{G_{cc}}{M_c}, \tag{2.27}$$

the average, Re, is calculated as:

$$\mathrm{Re} = \frac{1}{n}\sum_{c=1}^{n} \mathrm{Re}(c). \tag{2.28}$$

Finally, F-score, F(c), is calculated as (Lou et al., 2015; Rosset, 2004):

$$\mathrm{F}(c) = 2\frac{Re(c)\times \mathrm{Pr}(c)}{Re(c)+\mathrm{Pr}(c)}. \tag{2.29}$$

The average F-Score is obtained by:

$$\mathrm{F\_Score} = \frac{1}{n}\sum_{c=1}^{n} \mathrm{F}(c). \tag{2.30}$$

2.2 Convolutional Neural Networks and Scene Recognition

In this section, the basic concept of convolutional neural networks (CNN) is discussed. The CNN architecture is explained in Subsection 2.2.1. The standard CNN architectures that are used in our research are described in Subsections 2.2.2-3.

2.2.1 Convolutional Neural Networks

Neural networks are systems inspired by parallel distributed processing in the brain (Zurada, 1992). They show high classification when they are trained on a labeled dataset (Brownlee, April 24, 2019). Recently, the deep learning approaches show high accuracy in many computer vision applications including scene recognition (B. Zhou et al., 2018), medical images (Hassantabar, Ahmadi, & Sharifi, 2020), object detection

(He et al., 2016), etc. A deep neural network is the name used for ‘stack neural networks’, which is composed of several layers. The most wide used types of deep network is the convolutional neural networks (CNN). It adopts special architecture that is particularly focus on images classification (Brownlee, April 24, 2019). In Figure 2.14, a standard CNN’s architecture is shown. It consists of convolutional layers, pooling layers, and fully connected (FC) layers. Input and output indicate input image and predicted class type, respectively. The detail of each layer is given below in Subsections 2.2.1.1-4, respectively.

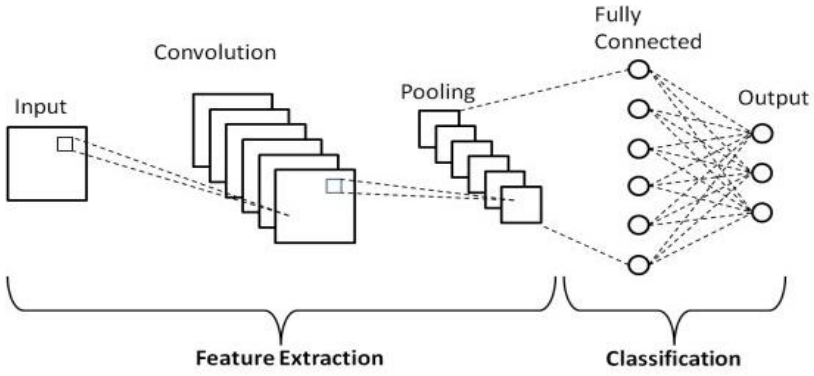


Figure 2.14: A standard deep CNN architecture (Phung & Rhee, 2019).

2.2.1.1 Convolutional Layers

It is a fundamental component of the CNN architecture that performs feature extraction, and it typically consists of combination of linear and nonlinear operations, i.e. convolutional operation and activation function (define in Section 2.1.5.2). Convolution operation is explained in Section 2.1.4 and formula is given in (2.1). Convolutional operation is applied on the input image, and yields a new value for each pixel, as Figure 2.10 illustrates. It is called as a feature map, an example is shown in Figure 2.15 in which 2D array size of 5×5 is given and 3×3 kernel is applied as convolutional operation. Next, the output of the convolution operation is then given to a nonlinear activation function. The activation function is deciding what value is to be

given to the next neuron. The most common nonlinear activation function, rectified linear unit (ReLU) is used, which simply computes the function:

$$f(x) = max(0, x), \tag{2.31}$$

where each x is numerical input which is taken from previous neuron (Krizhevsky, Sutskever, & Hinton, 2017; LeCun, Bengio, & Hinton, 2015). The output is in the range of $[0, \infty]$. The $max(.)$ is a function which returns largest value from the numbers provided.

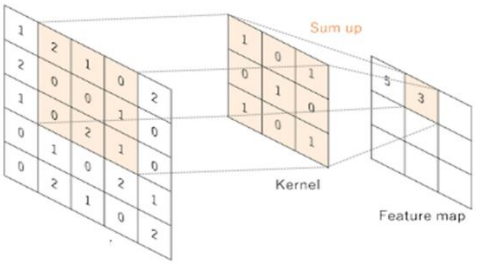


Figure 2.15: Example of convolutional operation (Yamashita, Nishio, Do, & Togashi, 2018).

2.2.1.2 Pooling Layers

The pooling layer captures an increasingly larger field of view and it reduces the features map (Yamashita et al., 2018). The most popular form of pooling operation is max-pooling. It extracts patches from the input image feature maps, outputs the maximum value in each patch, and discard all the rest values. A max-pooling with a filter of size 2×2 is commonly used in practice (Yamashita et al., 2018), p. 616. An example is shown in Figure 2.16 in which an input data is divided into four patches and a maximum value is selected from each of the patches. Another pooling operation is global average pooling(GAP) (Yamashita et al., 2018). In this operation, the feature map with size of $height \times width$ (see Figure 2.15), is down sampled into a 1×1 array by simple taking the average of all the elements in each feature map, whereas the depth of feature maps is unchanged (Yamashita et al., 2018). Depth indicates the length

of the feature vector that corresponding to the number of feature maps (feature vector with length n, where n is number of feature maps). Global pooling is applied only once before the FC layers.

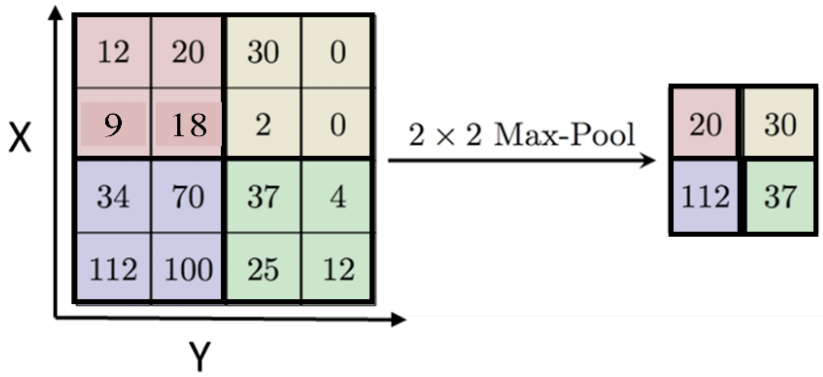


Figure 2.16: Example of max-pooling using 2×2 conv. filter.

2.2.1.3 Fully Connected (FC) Layer

The last layer of the CNN architecture is FC layer. The output feature maps of the final convolution or pooling layers is typically represented in a 1D array of vector and connected to a one or more FC layers, also called as dense layers (Yamashita et al., 2018). In FC layer, every input is connected to every output by a learnable weight (Yamashita et al., 2018), as illustrated in Figure 2.17. The $1,2,\ldots,n,$ denotes the number of input features to FC layer. The weights are generally learnt during training, therefore, it is called as learnable weight. FC layer has same number of outcome nodes as the number of categories. Each FC layer is followed by a non-linear function, such as ReLU, as given in (2.31). Furthermore, last/output layer is typically different from the input and middle FC layers. An activation function, "softmax function" is applied for classification problem (Yamashita et al., 2018), which outputs class probabilities or score vector, in which each value ranges between 0 and 1 and all values sum to 1.

The softmax function is special type of activation function, which is computed as (Brébisson & Vincent, 2015),

$$f(\boldsymbol{x}_i) = \frac{\exp(x_i)}{\sum_{j=1}^{S} \exp(x_j)}, i = 1,2..S, \quad (2.32)$$

Where $f(x_i)$ is softmax function, x_i input vector, $exp(x_i)$ is standard exponential function for input vector. S is number of classes. More detail of softmax function is given in (Brébisson & Vincent, 2015). Finally, the output of softmax function is measured in terms of accuracy or any metric as it depends on the classification problem. In this research, the accuracy is used, which can be measured using equation (2.24).

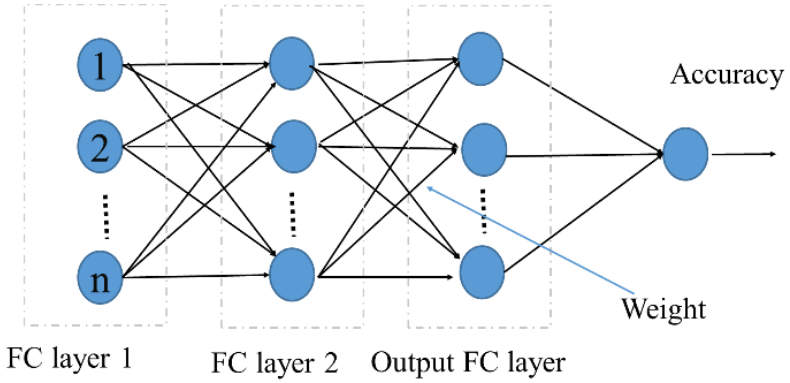


Figure 2.17: Basic structure of fully connected layer.

2.2.1.4 Training and Validation of a CNN Architecture

Network training is a method of identifying biases in convolution layers and weights in FC layers that reduce variations on a training dataset between performance predictions and given ground truth labels (Gonzalez & Woods, 2018), p. 953. "Backpropagation algorithm is the method commonly used for training neural networks where loss function and gradient descent optimization algorithm play essential roles" (Yamashita et al., 2018), p.617. Back propagation (BP) algorithm finding the value of the weights of FC layers and biases. It involves four basic steps

(Gonzalez & Woods, 2018), p. 953: (1) input the samples (training images), (2) a forward pass through the network to classify all the training samples and measure the classification error, (3) a backward (backpropagation) pass that feeds the performance error back through the network to measure the changes expected to adjust the parameters, and (4) updating the weights and biases in the network. "These steps are repeated until the error reaches an acceptable level" (Gonzalez & Woods, 2018), p.953. The detail of BP and training process is given in Chapter 12, (Gonzalez & Woods, 2018), pp 953. Loss function (L) is a cost function, which is calculated the classification error (i.e. accuracy) between output predictions of the network and given ground truth labels (accuracy defined in (2.24)). The parameters are updated by stochastic gradient descent (SGD) optimization algorithm that iteratively updates the parameters including weights, basis of the architecture to minimize the classification error. This process is given in MATLAB (version 2018b) as a function that can be applicable for new image dataset. The function is defined as: Model = trainNetwork (TrainData, TrainLabels, 'Pre-trained CNN', 'options'), where 'TrainData' indicates training images, and 'TrainLabels' is ground truth labels of training images. The 'Pre-trained CNN' model is already trained model on the similar large dataset. Instead of building a model from scratch to solve the classification problem which need a large dataset and huge effort, the pre-trained model can be used as a starting point (Tang, Wang, & Kwong, 2017a). It consists of multiple layers and it allows fixed size of input image (as details of the different layers are given above). The fixed size mean the size of receptive field such as GoogLeNet (Szegedy et al., 2015) accept 224x224 pixels size of image in the RGB color space. Every architecture has different number of layers (most useful architectures are given in Section 2.2.2-3). The standard deep CNN architectures with different number of layers, such as AlexNet (Alex, Sutskever, &

Hinton, 2012), GoogLeNet (Szegedy et al., 2015), ResNet (He et al., 2016), and VGG (Simonyan & Zisserman, 2015) are available in Matlab package, which can be easily installed and can be used. The last parameters, 'options', contain training parameters, such SGD, initial learning rate (e.g. 0.005), number of epoch (e.g. 20), and batch size (e.g. 16). The batch size indicates a subset of training dataset (detail is given in (Yamashita et al., 2018)). After completing the training process, it returns a trained model, '*Model*'. This model can be used to classify test images, and its performance can be calculated by metrics given in Subsection 2.1.7. The standard CNN architectures, namely GoogleNet (Szegedy et al., 2015) and ResNet (He et al., 2016) architectures are briefly described below.

2.2.2 GoogLeNet Architecture

The GoogLeNet (Szegedy et al., 2015) is the winner of the 2014 ILSVRC competition on ImageNet dataset (J. Deng et al., 2009). The main objective of the GoogLeNet architecture is to approach high accuracy with a reduced computation cost (Szegedy et al., 2015). It introduced the new concept of inception module in CNN, whereby it incorporates multi-scale convolutional transformation by splitting, transform, and merge idea, as shown in Figure 2.18. In inception module, the fixed size of convolutional filters ($1 \times 1, 3 \times 3, and\ 5 \times 5$) and 3x3 max-pooling operation are used in a parallel way on the input image and the output of these filters are stacked together to generate final output for next module (Szegedy et al., 2015). In Figure 2.18, inception module, 'Module 1', is shown with its conv., pooling filters, and number of stride (S). The filter moves from one position to the next position by number of pixels, which is called "stride". In this way, it captures spatial features at different scales. The standard GoogLeNet architecture consists of 9 inception modules and it contains three

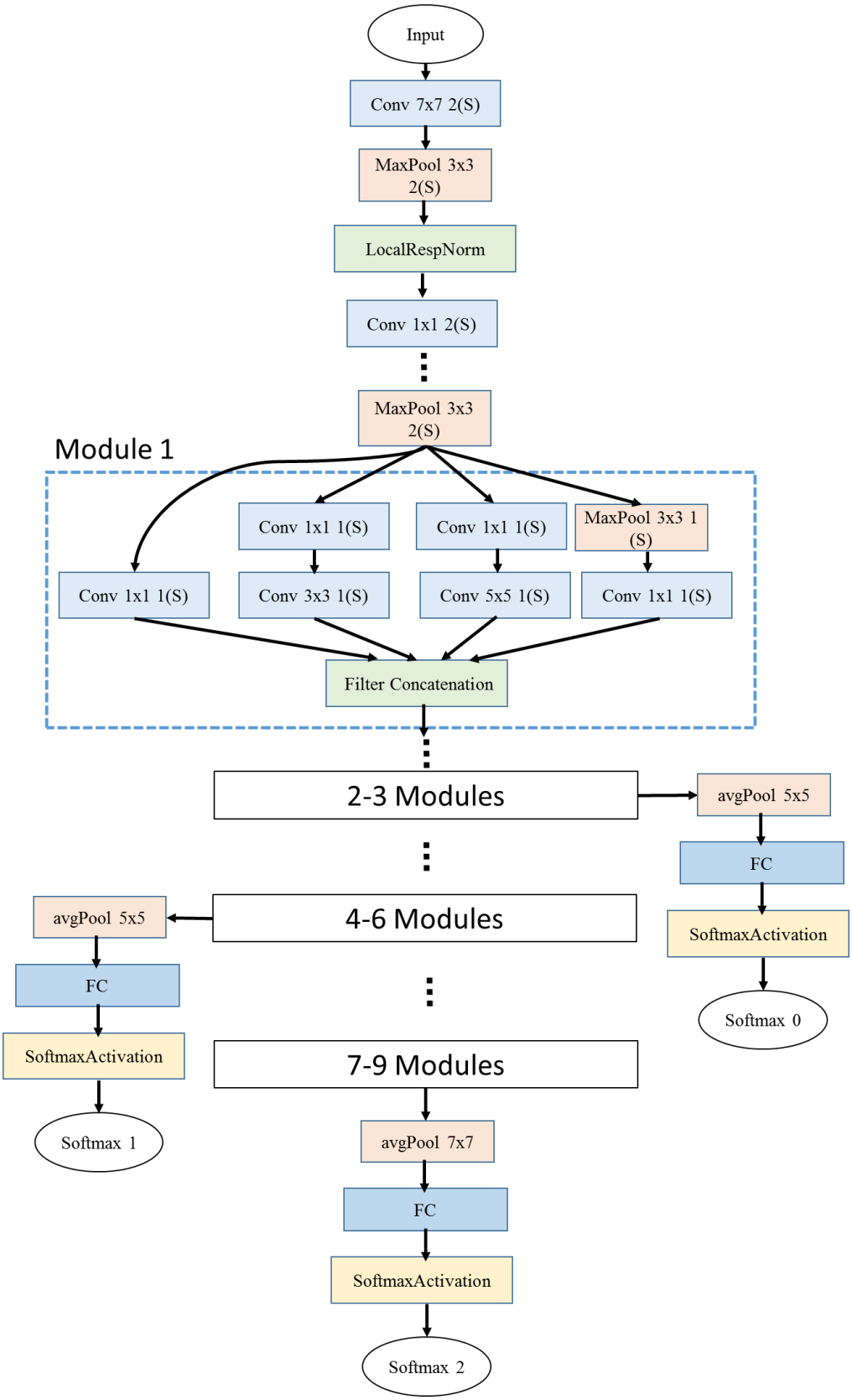


Figure 2.18: GoogleNet architecture (Szegedy et al., 2015).

auxiliary classifiers, softmax 0, softmax 1, and softmax 2, connected after each of the three modules, respectively, (see Figure 2.18). The softmax function is defined in equation (2.32). The GoogleNet architecture has an additional conv., max-pooling layers at the beginning and three GAP layers before FC layers, which are connected individually with three auxiliary classifiers. Authors show that performance of auxiliary classifiers was relatively minor (around 0.5%) when the ImageNet dataset is used. In this architecture, the parameters are reduced from 60 million (AlexNet (Alex et al., 2012)), to 4 million, and it achieves a top-5 error rate of 6.67% (Szegedy et al., 2015). The parameters are sum of all weights and biases in the network (Alom et al., 2018). The two type of error rates are used (Alex et al., 2012): "top-1 and top-5, where the top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model" (Alex et al., 2012), p.2. In Matlab the GoogLeNet is available as a pre-trained model on ImageNet dataset, $net = GoogLeNet$, which can be used as input to the trainNetwork(.) function to train the new model according to the given image dataset. Furthermore, the intermediate modules can be used for training the model. The features from these modules can be extracted using Matlab function, X_1 = activations (net, Training data, layer); X'_1= squeeze (mean(X_1,[1 2]))'. Then these features can further use for training the classifier. Activation(.) function takes pre-trained CNN model (net), training data (labels and images), and particular name of the layer (mostly max-pooling layer) as input and generates the feature maps, X_1. X_1 is high dimensional matrix and it has different dimensions at different layer. E.g., 7x7x1000 dimensional array (see Figure 2.18, (avg. pool 7x7) before softmax 2). The squeeze (.) function takes the X_1 with [1, 2] parameters and returns 1D array, feature vector, e.g., with the length of 1000, if

7x7x1000 array is given. Thus, these functions are used as GAP operation to extract the feature from any intermediate layer.

2.2.3 ResNet Architecture

The ResNet was introduced by He et al. (He et al., 2016) which is considered as continuation of deep network. ResNet wins the CNN architectural competition by introducing the concept of residual learning in deep CNN model and derived an efficient methodology for learning the deep network. The standard architecture is shown in Figure 2.19. The model consists of several blocks, each block is called as residual block or identity block. Residual block, as shown in the top left of Figure 2.19, is the basic structure of the ResNet architecture to learn the residual function of F(X), which is related to the standard function of H(X)=F(X)+X. The H(X) is learned by model which is closer to identifying function X than random. Thus, instead of having a network which learns H(X) from randomly initialized weights, the residual F(X) is learned. In this way, it saves the training time and also solve the problem of vanishing gradient by including the skip connections (He et al., 2016). The vanishing gradient problem is some cases, the gradient is vanishingly small, and in worst case it may completely stop the neural network from further training (Hochreiter, Bengio, Frasconi, & Schmidhuber, 2001). A skip connection allows the information of previous layer to flow more easily to the next layer, as shown in Figure 2.19. The ReLU is used as activation function (defined in (2.31)) in each residual block. Moreover, the downsampling is performed directly by conv. layers that have a stride 2. The ResNet model has an additional conv layer at the beginning and GAP layer (avg pool in Fig 2.19) at the end after the last layer. It has FC layer with 1000 neurons and a softmax (He et al., 2016). More detail of the ResNet architecture is given in (He et al., 2016). It introduces with three different versions mainly, 50, 101, and 152 layers

deeper. It achieves 3.57% of top-5 error rate on ImageNet dataset (J. Deng et al., 2009) when the 152-layers are used. He et al. (He et al., 2016) also applied the ResNet with

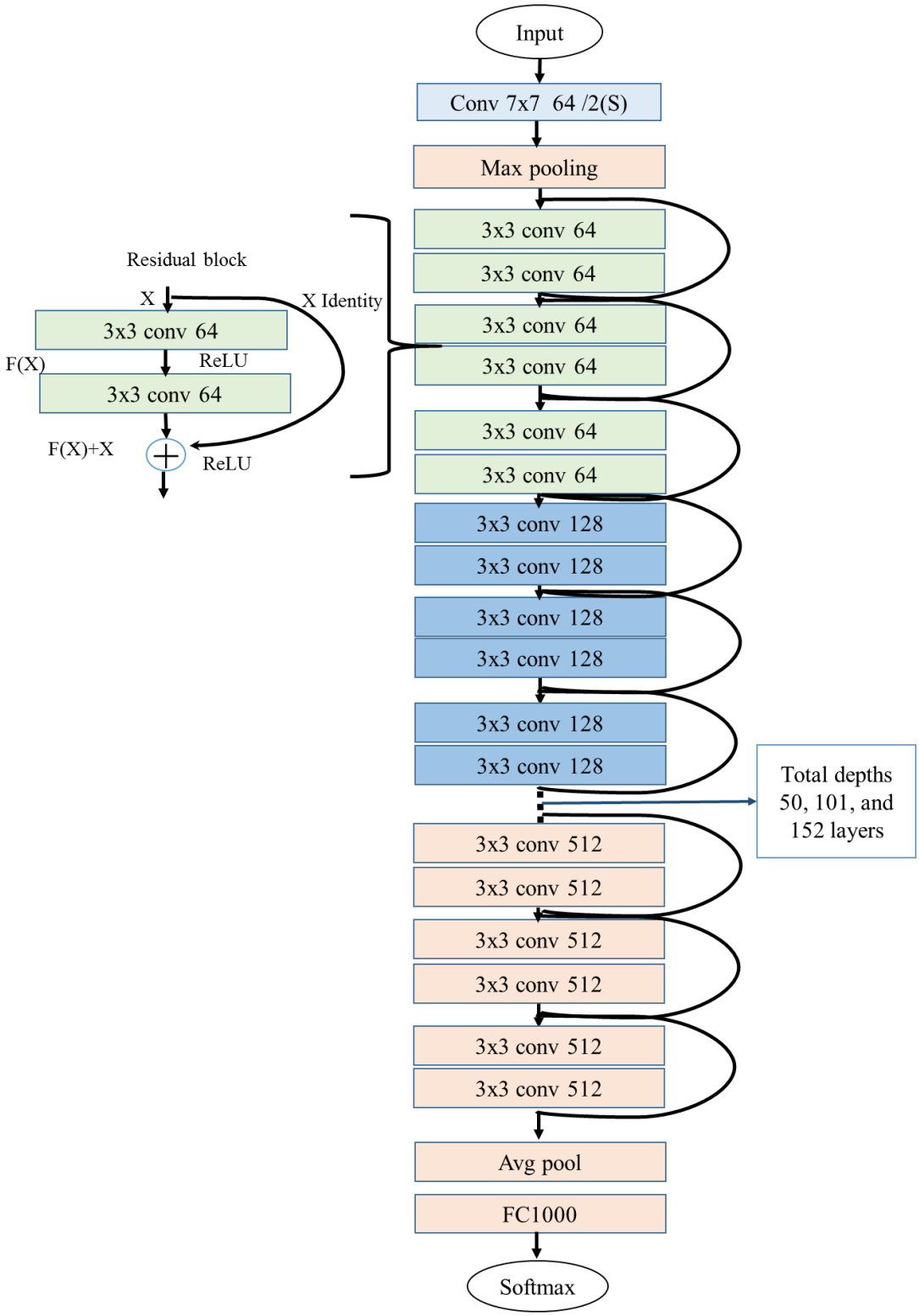


Figure 2.19: ResNet architecture (He et al., 2016).

50 and 101-layers deeper architectures on ImageNet dataset and achieve 5.25% and 4.60% top-5 error rate, respectively. The Matlab provides $net = ResNet50/101$, as pre-trained model on ImageNet dataset, which can be used as input to the trainNetwork(.) function to train the new model according to the given image dataset. The deep features can be extracted from intermediate residual blocks as well by using the Matlab functions as described in above Section 2.2.2.

2.3 Scene Recognition Methods and Results

In this section, the existing methods for scene recognition are described. The summary of these methods are given in Table 2.1, in which author name of each method, dataset, methodology, and results are elaborated. The datasets detail is summarized in Table 2.2, which are used by related methods. The dataset name, number of categories, scene types (indoor or outdoor), number of images, size of images, and images per category detail are given in this Table.

Oliva et al. (Oliva & Torralba, 2001b) proposed a computational model where in order to represent the spatial structure of the image scene, they use local and global image scene data. It is estimated the relation between spatial properties and scene categories by using Gist features. The model suggested that images have a common spatial structure in a scene class that can be derived without image segmentation. The mathematical description of the spatial structure of the scene is represented using Gist features. It catches the dominant perceptual features of a scene, such as naturalness, openness, roughness, expansion, and ruggedness of a scene (Oliva & Torralba, 2001b). It achieves 83.0% accuracy on eight different outdoor scene categories. Hoiem et al. (Derek Hoiem et al., 2007) introduce a framework of outdoor 3D scene recognition from a single image. The method was designed for 3 outdoor categories including, sky,

background (vertical), and ground (support). It uses superpixels and multi-segmentations and variety of cues such as color, texture, location, and perspective features to estimate the 3D surface of the outdoor images. However, the framework can only handle the outdoor images. It achieves 88.10% recognition accuracy on 'about 300' images dataset.

Table 2.1: Summary of previous work.

Rows No.	Name of Authors	Name of datasets	Methodology	Performance Metric (%)
1	Oliva et al. (Oliva & Torralba, 2001b)	8-scene categories	Use GIST descriptors at 4 × 4 uniform grid patches. 100 samples for training of each class and rest for testing	Accuracy (ACC): 83.70
2	Hoiem et al. (Derek Hoiem et al., 2007)	3 geometric classes	Superpixels and multi-segmentations utilization. Use different cues including color, texture, shape, and location.	ACC: 88.10
3	Nedovic et al. (Nedovic et al., 2010)	stage dataset	Use texture gradient, color, and perspective line features at 4 × 4 grid patches.	ACC: 38.0
4	Lou et al. (Lou et al., 2015)	stage dataset	Predefined template-based segmentation and structure SVM. 50% for training and 50% for testing	ACC: 47.30, precision(PR): 45.60, recall (RE):44.90, F-score: 44.20
5	J. Sánchez et al. (Sanchez et al., 2013)	SUN 397 dataset	BoW, fisher & pyramid based method, Gaussian Mixture models (GMM). 50% images for training and 50% for testing	ACC: 47.20
6	Zafar et al. (Zafar, Ashraf, Ali, Ahmed, Jabbar, & Chatzichristofis, 2018)	15-scene, UCM-21 datasets	BoW & orthogonal vector histogram (OVH). 15-scene: 100 images for training and rest for testing. UCM-21: 80% for training and 20% testing.	ACC 15-scene: 87.07, UCM-21: 100
7	Ali et al. (Ali et al., 2018)	15-scene	Hybrid geometric spatial image representation (HGSIR) method. 100	ACC: 90.41

			images for training and rest for testing	
8	Patalas et al. (Patalas & Halikowski, 2019)	Hand written image dataset	Use CNN replacing FC layer with SVM. 60000 images for training 10000 images for testing.	ACC: 99.04
9	Tang et al. (Tang et al., 2017a)	15-scene, MIT67, SUN397 datasets	Intermediate layer features of GoogLeNet model and score-level fusion. 15-scene: 100 images for training and rest for testing. MIT67: 80% for training 20% for testing. SUN397: 50% for training and 50% for testing	ACC 15-Scene: 92.90, MIT67: 79.63, SUN397:64.06
10	Liu et al. (Shaopeng Liu, Tian, & Xu, 2019)	15-scene, MIT67, SUN397 datasets	Intermediate layer features of ResNet model, softmax function. It uses same setting given in Tang et al. (Tang et al., 2017a)	ACC 15-scene: 94.04, MIT67:74.63, SUN397:65.46
11	Chen et al. (Wang, Peng, & Lin, 2021)	15-scene, UCM-21	Robust local metric learning via least square regression regularization. It uses same setting as used in (rows: 6,8,9)	ACC 15-scene: 93.50, UCM-21: 97.81
12	Chen et al. (Wang, Peng, & De Baets, 2020)	15-scene, MIT67, UCM-21 datasets	Intermediate layer features, adaptive discriminative metric learning. It uses same setting given in Tang et al. (Tang et al., 2017a) and for UCM-21: 80% for training and 20% testing.	ACC 15-scene: 96.39, MIT67:88.43, UCM-21: 99.14

Table 2.2: Summary of the datasets used in related methods.

Rows No.	Database name	Indoor/ Outdoor	No. of classes	No. of images	Size of images (pixels)	No. of image per category
1	SUN 397 (Xiao et al., 2010)	Both	397	130519	Different size	100 to 2361
2	MIT67 (Quattoni & Torralba, 2009)	Indoor	67	15620	Different size	At least 100
3	8-categories (Oliva & Torralba, 2001b)	Outdoor	8	2688	256×256	292 to 410

4	15-scene (Lazebnik et al., 2006)	Both	15	4486	different size	200 to 400
5	stage dataset (Nedovic et al., 2010)	Both	12	About 2000	Different size	Different no. of images
6	UCM-21 (Yang & Newsam, 2010)	Remote scene	21	2100	256×256	100
7	3 geometric classes (Derek Hoiem et al., 2007)	Outdoor	3	300	Different size	Different no. of images
8	Hand written digits dataset (Yann LeCun, Corinna Cortes, & Burges, 2010)	-	10	70000	28x28	Different no. of images

Some approaches of scene recognition are based on Bag of Words (BoW) model (Lazebnik et al., 2006), e.g., J. Sánchez et al. (Sanchez et al., 2013) use dense scale invariant feature transform (SIFT) features in the Fisher Kernel (FK) framework as an alternative patch encoding technique and it achieves reasonable accuracy of 47.2% by using 50% training samples of SUN397 dataset (Xiao et al., 2010). The SUN397 contains 397 scene categories with 130519 images in total (detail is given in Table 2.2). Similarly, Zafar et al. (Zafar, Ashraf, Ali, Ahmed, Jabbar, & Chatzichristofis, 2018) proposed a new model based on BoW by computing an orthogonal vectors histogram (OVH) for triplets of identical visual words. The histogram–based representation is computed by using magnitude of these orthogonal vectors. This model is based on the geometric relationships among visual words and computation complexity of these approach increases exponentially with increasing in the size of codebook (Ali et al., 2018; Zafar, Ashraf, Ali, Ahmed, Jabbar, Qureshi, et al., 2018). It achieves 87.07% recognition accuracy on 15-scene categories (Lazebnik et al., 2006)

using 100 images for training and remaining for testing. It also utilized the UMC-21 dataset (Yang & Newsam, 2010) as well and achieved 100% recognition accuracy when the 20% images were used for testing. UMC-21 is remote sensing categories dataset (summary is given in Table 2.2). Another recent approach, hybrid geometric spatial image representation (HGSIR) method (Ali et al., 2018) achieves the maximum recognition accuracy of 90.41% on 15-scene image dataset. The above methods classify images in different categories such as coast, beach, mountain, kitchen, bedroom, and office, etc. However, the number of scene categories is very large. E.g., the SUN dataset (Xiao et al., 2010) has 397 scene categories. On the other hand, UMC-21 is not suitable for recognizing the world in 3D scene geometries (stages).

Nedovic et al. (Nedovic et al., 2010) classified scenes into twelve stages, i.e., sky-background-ground, ground, and background-ground, etc. Nedovic et al. (Nedovic et al., 2010) extract features set including parameters of Weibull distribution (four features), color (five features), and Perspective line (eight) features from $n \times n$ patches of an image for geometry classification. This algorithm uses multi-class SVM for stage recognition and achieves 38.0% recognition rate on their proposed a stage dataset which contains about 2000 images (summary is given in Table 2.2). Lou et al. (Lou et al., 2015) utilized template-based segmentation and extract HOG (Dalal & Triggs, 2005) (nine features), color features (HSV and RGB, 6 features), parameters of Weibull distribution (J.-M. Geusebroek & Smeulders, 2005) (4 features) for each image patch and introduced a graphical technique to learn an image presentation from features to scene classes. In consequence, Lou et al. (Lou et al., 2015) achieves 47.30 % accuracy of stage recognition on 50% testing images by utilizing the stage dataset (Nedovic et al., 2010). However, this method is complex and generates more than 100

segments for each template and finding the best fitting segment over component of a template is required extra computation.

CNNs have recently revolutionized computer vision and demonstrate substantial output achieved in many applications, such as classification of image scenes (B. Zhou et al., 2018), texture recognition, and facial applications (Mei Wang & Deng, 2018), background of CNN is given in Subsection 2.2. The popular CNNs architectures, such as GoogLeNet (Szegedy et al., 2015), ResNet (He et al., 2016), AlexNet (Krizhevsky et al., 2017), and VGG-16 (Simonyan & Zisserman, 2015) are required the large amount of labelled data for training process (J. Deng et al., 2009) with a specified input size and achieve a high rate of accuracy. The classification performance of CNN based methods depends on an input dataset. Recent studies, such as (Patalas & Halikowski, 2019) having claimed that replacing the trainable classifier (conv. Softmax function) of a deep CNN model with SVM can enhance the recognition performance and 99.04 % accuracy on hand written dataset is achieved (Yann LeCun et al., 2010). The CNN based methods require a large labeled dataset for a particular issue which is one of the CNN main challenges. And CNN architecture suffers from the overfitting problem when small or medium-scale datasets are utilized and this problem is existing with 3D geometry recognition as well.

Some researchers utilize the intermediate and FC layers features for scene recognition. The standard CNN methods only take semantic information by activation of the FC layer, which are robust and show good performance of scene recognition. However, it loses the object description details during the multiple convolutional and pooling operations when the small or medium dataset is used (Shaopeng Liu et al., 2019; Tang et al., 2017a). To address this issue, Tang et al. (Tang et al., 2017a) introduced G-

MS2F model which investigates the intermediate layers features of GoogLeNet (Szegedy et al., 2015) model for scene recognition. The inception modules in GoogLeNet contribute to limiting the size of parameters and the difficulty of the model. These modules are divided into three sections and three objective functions (classifiers) are included after every three inception modules to solve the problems of over-fitting (detail is given in subsection 2.2.2). The output features of each of the parts are fed to a classifier and predicted scores which are fused by using product-rule for final decision. It achieves 92.90% recognition accuracy on 15-scene image dataset using standard setting (100 images for training and remaining for testing, see Table 2.2). Author also applied this method on MIT67 (Quattoni & Torralba, 2009), and SUN397 dataset and it achieved 79.63% and 64.06%, respectively. MIT67 is indoor 67 categories dataset which contains 15620 images in total (summary is given in Table 2.2). As ResNet model obtained higher performance than GoogLeNet model (see Subsection 2.2.3), Liu et al. (Shaopeng Liu et al., 2019) proposed a novel ResNet-based model by utilizing multi-layer features and taking advantage of these features and fused them for scene classification by softmax function (FTOTLM). Multi-layer features denote the intermediate layer features of CNN model. Liu et al. (Shaopeng Liu et al., 2019) use the pre-trained 18-layer ResNet contains 8 residual blocks. Each of the blocks can be used for feature extraction while it is not wise to extract features from each block because it is similar to the adjacent block and it increases the redundancy and complexity of the model. Therefore, Liu et al. (Shaopeng Liu et al., 2019) use 5 residual blocks to extract features. It increases the dimensionality which is reduced by using GAP. This model obtains 94.04% recognition accuracy on 15-scene dataset. Also it achieves 74.63% and 65.46% accuracy on MIT67 and SUN397 datasets, respectively. Chen et al. (Wang et al., 2021) introduced a novel method,

called “robust local metric learning via least square regression regularization” (RLML-LSR), to learn a more advance distance metric for scene categorization. This method achieves 98.33%, 97.81%, and 93.50% recognition accuracy on UCM-21, 8-categories sports image dataset, and 15-scene datasets, respectively. Chen et al. (Wang et al., 2020) also proposed a deep feature fusion method, called “deep feature fusion through adaptive discriminative metric learning (DFF-ADML)”, to determine the complementary and consistent information for scene recognition. It achieves 96.39%, 88.43%, 99.14% scene recognition accuracy on 15-scene, MIT67, and UCM-21 datasets, respectively. These methods solved the scene recognition problem by adapting the intermediate layer features.

In conclusion, the above datasets, such as 8-categories, MIT67, UCM-21, and SUN397 are very large or only contain indoor or outdoor scene images. In addition, the datasets are not annotated on the base of image scene geometry structure, therefore, these datasets are not sufficient for 3D scene recognition problem. However, 15-scene datasets can be used to test the 3D scene recognition model as this dataset categories are likely to the stages, but the dataset contains 4485 images in total which are not enough to efficiently train a deep CNN model. The Nedovic et al. (Nedovic et al., 2010) and Lou et al. (Lou et al., 2015) use the 3D scene geometry dataset (stage dataset, see Table 2.2), but this dataset is not publically available.

2.4 Problem Definition

In this thesis, the 3D scene recognition problem from a single image is considered, which is important for many applications of computer vision, such as robot navigation system, 3D TV, scene understanding. The problem of 3D scene recognition can be described in three following parts:

1) The recent approaches of 3D scene recognition, (Lou et al., 2015; Nedovic et al., 2010), were applied on the limited images (about 2000 images), which did not reach the significant level of 3D scene recognition as results which were shown in literature study. And their classification performance are not enough to well distinguishing the scene images with complex structure. Therefore, it needs a deep investigation of 3D scene recognition and requires a novel approach, which should achieve sufficient performance of 3D scene recognition for medium scale datasets.
2) The recent studies of scene recognition are based on deep CNN, which achieve significantly high performance on large image datasets as compared to traditional approaches. However, particular 3D scene geometry recognition using CNN is not well investigated. Meanwhile, the CNN based methods require a large labeled dataset for a particular scene classification problem, which is one of the CNN main challenges. The datasets of 3D scene geometry are not publically available and it is big obstacle to evaluate the new idea of 3D scene recognition. Therefore, it is required to introduce a new 3D scene geometry dataset, which should be suitable for learn the recent machine learning algorithms, such as deep CNN.
3) Some researchers utilize the intermediate layer features of CNN for scene classification when the medium scale datasets are available. However, intermediate layer and higher layer features still show the weak performance because of losing the scene shape, color, texture information, and scene to object relationship when the images are passing from multiple convolutional and pooling operations. It needs more attention to investigate the complementary and consistent features of 3D scene recognition at intermediate layers.

The above problems are investigated and solutions are proposed in Chapters 3-5, respectively.

Chapter 3

3D SCENE RECOGNITION USING SEGMENTATION-BASED FEATURE EXTRACTION METHOD

In this chapter, we introduce a novel segmentation-based feature extraction method of 3D scene recognition. Literature study (Lou et al., 2015) shows that the predefined templates are rough 3D scene geometries, which can be used for segment the input image for feature extraction as it is discussed in Subsection 2.1.3. Inspired by this idea, we propose a novel method of segmentation-based feature extraction that utilizes the predefined templates and the ensemble of classifiers (see detail in Subsection 2.1.5.3). Each template represents unique rough structure and provides different segments. Features are extracted by following these template-based segments and then fed into individual classifier. Then classifiers outcome are combined using sum rule. Lou et al. (Lou et al., 2015) use Carrira and Sminchinsescu (Carreira & Sminchisescu, 2012) segmentation method, as described in Subsection 2.1.3. However, this technique generates hundreds of segments for each input image and to select the segment that has largest overlap-to-union score is an expensive task as it requires to compare each template component to each segment. In contrary, the active contours algorithm (Chan & Vese, 2001) can be used for template-based segmentation, which can generate one segment for one template-component, the description of active contours algorithm is given in Subsection 2.1.3. Next, the Segmentation-based feature extraction method uses ensemble of classifiers because of each template has unique structure and template generates different segments for feature extraction. Therefore, it is logical to

use an individual classifier for each template.

The design of the method is given in Section 3.1. Its implementation, testing, and experimental results are given in Section 3.2. Summary of this method is given in Section 3.3.

3.1 Design of Segmentation-Based Feature Extraction Method

In this section, the segmentation-based feature extraction method is defined by Algorithm 3.1 and illustrated by Figure 3.1. For more understanding, we include the Matlab codes in Appendices corresponding to each main function of this algorithm. Compared to Algorithm 2.1, the Algorithm 3.1 contributes segmentation process, ensemble of classifiers, which are associated with T number of templates, and the combination of scores of T classifiers. The Algorithm 3.1 takes T templates, N training images, $\mathcal{N}$ total number of images, Y_N training labels, and Y_M testing labels, S classes as inputs and generates accuracy (Acc), precision (Pr), recall (Re), and F-score for testing images. The method consisting of three main steps for the 3D scene recognition of an image by exploiting template-based segmentation and feature extraction, the training and testing of an ensemble of classifiers, and the fusion of the ensemble of classifiers. Step 1: the template-based segmentation and feature extraction procedures are discussed in Subsection 3.1.1 (lines 1-6 of Algorithm 3.1). Step 2: classifier training and the testing of the ensemble of classifiers are described in Subsection 3.1.2 (lines 7-17). Step 3: the fusion of the ensemble of classifiers are explained in Subsection 3.1.3 (lines 18-21).

3.1.1 Template-Based Segmentation and Feature Extraction Procedures

In this step of the 3D scene recognition method, each image is parsed to predefined templates T, as shown in Figure 3.1. Each template generates a different set of

Algorithm 3.1 Method of 3D Scene Recognition using segmentation based features extraction

Input: *T* templates, *N* training images, $\mathcal{N}$ total number of images, *S* classes, Y_N training labels, Y_M testing labels

Output: Acc, Pr, Re, F-score

// Template–based segmentation and features extraction

1: **for** j=1:$\mathcal{N}$ **do** // for each image I_j

2: **for** t=1:T **do**

3: $Seg^{tj} = Template\_based\_segmentation(t, I_j)$ // according to Algorithm 2

4: $X_{tj} = Features\ extraction(Seg^{tj}, I_j)$ // feature vector for image I_j, t is a certain template

5: **end for**

6: **end for**

// Training & testing

7: **for** t=1:T **do**

8: $\check{X}_t = \{(X_{tj}, Y_j)\}_{j=1}^{N}$ // N is a number of training samples.

9: **end for**

10: **for** t=1:T **do**

11: $TrainedCL_t = CL_t(\check{X}_t)$, //where $\check{X}_t$ is used to train *t*th classifier

12: **end for**

// Testing (prediction)

13: **for** j=N+1:$\mathcal{N}$ **do** // loop on M test images

14: **for** t=1:T **do** //loop on classifiers

15: $P_{tj}^{S} = Classify(TrainedCL_t, X_{tj})$ // P_{tj}^{S} is S-dimensional score vector for an image I_j.

16: **end for** //t

17: **end for**//j

// Classifiers combination

18: **for** j=N+1:$\mathcal{N}$ **do** // M test images

19: l'_j = Sum_rule ($\{P_{tj}^{S}\}_{t=1}^{T}$), fusion of T scores

20: **end for**//j

//Performance measures

21: [Acc, Pr, Re, F-score]=Calculate_Measures (***l'***, Y_M). // Y_M is a true labels

end Algorithm

segments and returning segments are used to extract the feature set, as steps are given on lines 1-6 in Algorithm 3.1. Line 3 derives segments, Seg^{tj}, from an image I_j by using template, $t, t = 1,2,..T$. On Line 4, the features are extracted from Seg^{tj}, and are assigned to a vector X_{tj}. The detail of template-based segmentation procedure is explained in Section 3.1.1.1 and the feature extraction procedure is described in

Section 3.1.1.2.

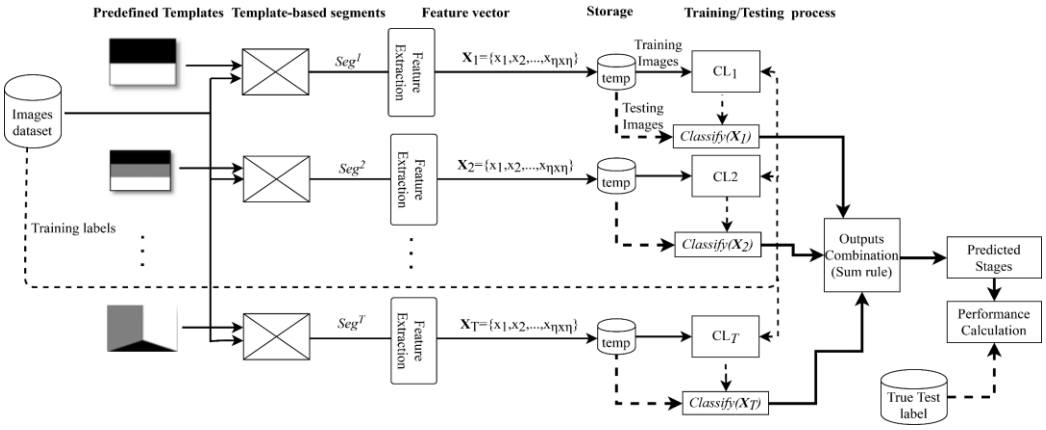


Figure 3.1: 3D scene recognition using segmentation-based features extraction. The predefined templates are following by Figure 2.8 (a)-(h)). Seg is segmentation, X_t is feature vector for each template t. CL_t is tth classifier. Sum-rule indicates summation of score of different classifiers.

3.1.1.1 Template-Based Segmentation Procedure Description

The template-based segmentation is described by $Algorithm$ 3.2 having two inputs: one is a template and second is an image I, and it generates a template-based segmentation, Seg. In Algorithm 3.2, line 1 shows the 's' number of components of an input template, TS. Next, each component, sk, $sk \in s$, is used as an initial contour in the active contours algorithm (description is given in Subsection 2.1.3). δ is a 2D array with a value of zero or sk. Lines 6 and 7 check whether the element of sk is equal to the element of $TS(i,j)$. Then it assigns $\delta(i,j)$. Next, lines 11 and 12 measure the center of the template component and these center values are used to find the minimum distance of the pixels $I(i,j)$ to segments; if they are not assigned to any segments then that pixel will be assigned to the nearest segment. After that, on line 13, the active contours algorithm is used to generate a segment M' of each sk component of the template. M′ is a 2D matrix contains the value [0, 1] where one shows that pixel

Algorithm 3.2 Template-based segmentation

Input: TS(H,W) template, I(H,W) image
Output: Seg(H,W) segmented image of size I
Initialization: Seg(H,W)=0, M(H,W)=0 where δ(H, W)=0 is temporary matrix

1: s=number of components(TS);//defines number of components in the template TS
2: **for** sk=1 to s **do**,
3: $\delta_{ij} = 0$; // loop on TS components
4: **for** i=1 to H **do** //loop on rows
5: **for** j=1 to W **do** //loop on columns
6: $\boldsymbol{if}(TS_{ij} = sk)$ **then** //check that
7: $\delta_{ij} = sk$ //if yes, store it in δ
8: **end if**
9: **end for** // j
10: **end for** // i
// find the x and y coordinate of the centre point of the sk-th component of TS
11: $C(sk,1) = (\min(i|\delta_{ij} = sk) + \max(i|\delta_{ij} = sk))/2$
12: $C(sk,2) = (\min(j|\delta_{ij} = sk) + \max(j|\delta_{ij} = sk))/2$
13: $M' = active\_contours(I, \delta)$
14: **for** i=1 to H **do** //loop on rows
15: **for** j=1 to W **do** //loop on columns
16: $\boldsymbol{if}(M'_{ij} == 1 \;\&\&\; Seg_{ij} == 0)$ **then** //$M'_{ij} = 1$ indicates part of the segment, 0 otherwise.
17: $Seg_{ij} = sk$
18: **end if**
19: **end for** // j
20: **end for** // i
21: **end for** // sk
22: **for** i=1 to H **do**,
23: **for** j=1 to W **do**,
24: **if** ($Seg_{ij} == 0$) **then** //if pixel is not a part of any segment
25: $c = find\_segment\_with\_centre\_nearest(i,j,C)$
// finds nearest segment, c, using Euclidean distance to out of s components.
26: $Seg_{ij} = c$; //c has value belonging to s.
27: **end if**
28: **end for** //end I
29: **end for** // end j
30: **Return Seg** // return the template-based segmentation

I(i,j) belongs to that certain segment. Thus, each component δ^{sk}, sk = 1, 2, …, s, is used as an initial contour and generates a segment M' of an image I. In the next step (at line 16–17), if output M′(i,j) is equal to one and $Sg(i,j)$ is equal to zero then the sk value will be assigned to $Sg(i,j)$. Thus, the set of generated segments composed in

Sg are equal to the number of components of a template. For example, in the second row of Figure 3.2, the template has three components, 'sky-background-ground' and it generates three segments: upper, Sg^1, middle, Sg^2, and bottom Sg^3. Each pixel of the image is assigned to one of the segments. Note that a pixel $I(i,j)$ that is not part of any segment is added to the segment Sg^{sk}, by using the center of components (C(sk,1) and C(sk,2)), which has minimal Euclidean distance to $I(i,j)$, (see lines 24–25 of Algorithm 2). The Matlab code of template-based segmentation procedure is shown in Appendix F. Line 25 illustrates the Euclidean distance function and returns the segment index c; later, it is assigned to the non-segmented pixel of image $I(i,j)$. The main purpose of this task is to fill the small holes among segments and generate proper segments for feature extractions.

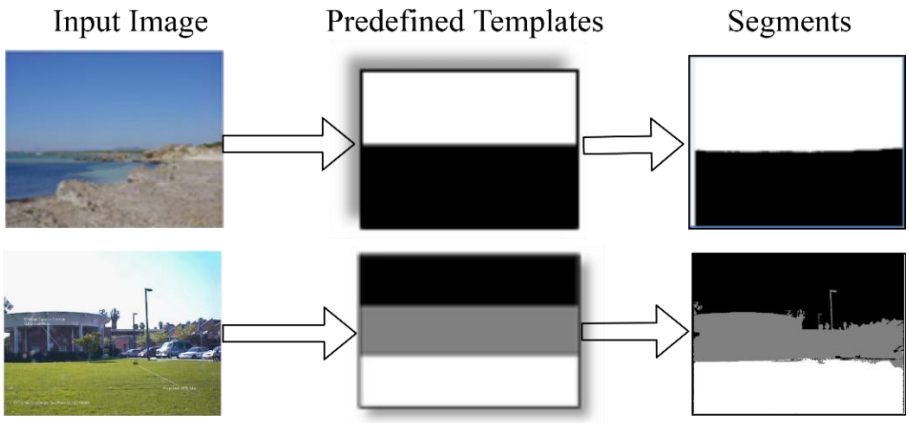


Figure 3.2: Example of template-based segmentation. Each image with size $H \times W$ is given to the Algorithm 3.2, with template, TS, which returns same size of segmented image as shown in the last column.

3.1.1.2 Segmentation-based Features Extraction Procedure

As in Figure 3.1, the next step after segmentation is feature extraction (Algorithm 3.1, line 4). The procedure, $Features\ extraction$, takes segmented image, Seg^{tj}, and RGB image, I_j, as input and returns the feature vector, X_{tj}. In this procedure, the input Seg^{tj}, is divided into η × η grid patches: $P_{i1,i2}, i1, i2 = 1,2,\ldots,\eta$, such that,

$$P_{i1,i2}(k,l) = Seg^{tj}((i1-1) \times M' + k, (i2-1) \times N' + l), \mathrm{k} = 1 \ldots \mathrm{M}',\ l = 1 \ldots \mathrm{N}', \quad (3.1)$$

where $M' = \left\lfloor \frac{H}{\eta} \right\rfloor$, $N' = \lfloor W/\eta \rfloor$. H and W are height and width of Seg^{tj}. If all elements of a patch have the same segment number, sk, then the patch belongs to the segment $Seg^{tj}(sk)$. The t indicates template and j indicates an input image. For example, Figure 3.3 shows that an image j is parsed by a template t and generates two segments, sky ($Seg^{tj}(1)$) and ground ($Seg^{tj}(2)$). Next, the patches are divided into groups by following these segments. Generally, a patch may overlap to several segments. In this scenario, it will be assigned to the segment group that has the largest overlap-to-union score. The corresponding function in Matlab is given in Appendix G. After that, the image features, F, that are explained in Subsection 2.1.4 are extracted for each image patch. From these features, the parameters of Weibull distribution (4 features) (see Subsection 2.1.4.1), mean of the color corrected coefficient of RGB and HSV (6 features) (see Subsection 2.1.4.2), HOG (9 features) (see Subsection 2.1.4.3), and LBP-E (6 features) (see Subsection 2.1.4.4) are normalized (in range [0, 1]) and then concatenated one after the other to obtain a single vector. The Matlab function of features extraction and features combination is shown in Appendix H. Next, these feature vectors are fused into a single vector x_{sk} for each sk-th segment, as shown in Figure 3.3. Then feature vectors for all the segments are combined into a single feature vector X_{tj} and stored into 'temp'. Thus, each feature vector X_{tj} for template 't' and image 'j' has same length of $\eta \times \eta \times \mathrm{F}$.

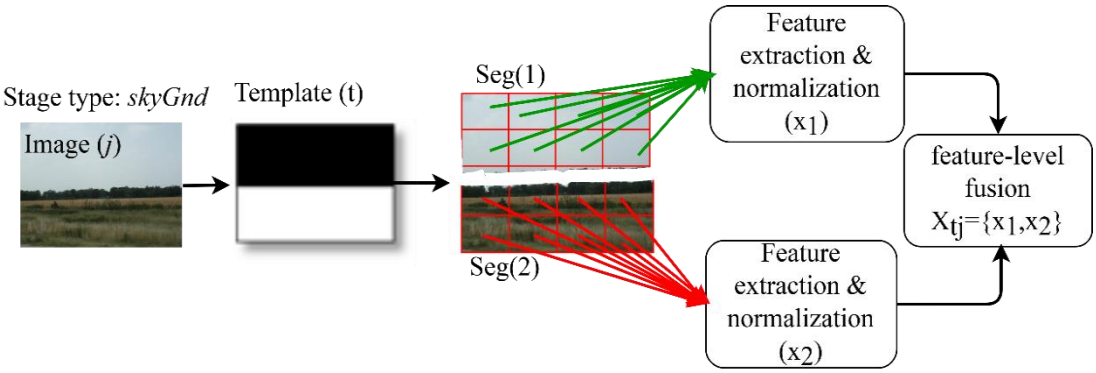


Figure 3.3: Procedure of feature extraction from each template-based segments and feature combination.

3.1.2 Classifiers Training and Testing for 3D Scene Recognition

Following Figure 3.1, the features vector from templates are ready to use in training set of classifiers. Suppose that we have training dataset with N records, (X_{tj}, y_i), where X_{tj} is a feature vector of image I_j for the template t and $y_j^i \in \Omega = \{\omega_1, \ldots, \omega_s\}$ is the *i*th category label, e.g., ω_i denotes the category 'skyBkgGnd', $i = 1,2, \ldots . S$, S is number of categories. $t = 1, \ldots, T$, is a template (see Algorithm 3.1, line 7-12). Then, the N-labeled feature vector can be represented by modifying (2.15),

$$\check{X}_t = \{(X_{tj}, y_j)\}_{j=1}^{N} t = 1,2.., T. \quad (3.2)$$

Therefore, we have the $\{\check{X}_1, \check{X}_2, \ldots \check{X}_T\}$ set of labeled feature vectors for N training samples. Each $\check{X}_t$ has a different order of feature concatenation and, for this reason, they are used as inputs to an individual classifier, $CL_t(\check{X}_t)$, to train a model $TrainedCL_t$, t=1, 2,…,T. Therefore, the method generates T-trained models for N-labeled data. As each template indicates a different pattern and features are extracted by following that pattern, the feature set of the testing data of each template is evaluated by using a corresponding trained model. Suppose that the feature vectors of the M testing samples are $\{X_{t1}, X_{t2}, \ldots, X_{tM}\}$ for template t, and are used as inputs to the $TrainedCL_t$ model, which predicts a score vector, P_{tj}^S, for the j-th image. P_{tj}^S is an

S dimensional vector, where S is a number of classes. The Matlab code of training classifier is shown in Appendix I. As classifiers are described in Subsections 2.1.5.2-3, the training images are used as input and it needs to set the classifiers kernel type, more detail is described in the comments of the code, see Appendix I. Similarly, for testing images, these classifiers return score matrix with size of $M \times S$, where M is number of testing images, and S is number of classes.

3.1.3 Fusion of the Ensemble of Classifiers and Performance Measures

This is the last step of the 3D scene recognition method, wherein the outputs of the ensemble of classifiers are fused together to obtain a class label. In Algorithm 3.1, lines 18–20 describe the decision of score combination using sum-rule (the score combination and sum rule is described in Subsection 2.1.5.3). Each classifier provides an individual score vector, P_{tj}^{S}, for each sample j, where t = 1, 2,…,T. Sum rule combines (2.20) the score vectors of T classifiers and generates a class label $l'_j, l'_j \in \Omega$ and j indicates an input sample. Thus, for testing data with a size M, it predicts l'_M, labels. The Matlab code of score combination is given in Appendix J. Finally, line 21 of Algorithm 3.1 calculates the Acc, Pr, Re, and F-score of the input testing data using equations (2.24-2.30). The metric calculation using Matlab function is given in Appendix K. This code requires predicted labels of M images, and ground truth labels of the M images and returns the performance metric.

3.2 Implementation, Testing of Segmentation-based Feature Extraction Method, and Experiments on Stage and 15-Scene Datasets

In this section, implementation details, testing of the method, description of datasets (including stage dataset and 15-scene dataset (summary is given in Table 2.2)), results of (our) segmentation-based feature extraction method, and state-of-the-art methods are discussed in Subsections 3.2.1-4.

3.2.1 Implementation Details

We conducted experiments with the segmentation-based feature extraction method to assess its effectiveness and performance on two different datasets. For both datasets, we followed the same sequence of steps, discussed in 3.1, to generate the features and their classification process. To evaluate this method, eight numbers of predefined templates ($T = 8$) are used in the experiments. Each template indicates particular image scene geometries as its association with scene geometries, which is shown in Figure 3.4. For indoor images, we were inspired by the work of (Lee, Badrinarayanan, Malisiewicz, & Rabinovich, 2017), where they defined a keypoint-based room layout that actually determines the indoor room shape. We use it as predefined template to improve the recognition rate of our indoor scene images. Thus, we add two predefined templates, which roughly show the scenes of corner images, as shown in Figure 2.8 (g, h). The 3D scene recognition using segmentation-based feature extraction method is implemented in MATLAB-2019a without using any parallel processing functionality. The MATLAB code is available in Supplementary Material (Khan et al., 2020a) and main functions are also given in Appendices as discussed in above Section 3.1. The SVM classifier is used to train and test our method. In MATLAB-2019a, it is available as a function, *fitcecoc (.),* description is given in Subsection 2.1.5.1 (see Appendix I). The kernel functions set to the linear, Gaussian or polynomial (Quadratic: with degree 2), selected kernel for each experiment are reported in tables. The cross-validation is set to 20-fold on the training dataset. We run it on a portable computer with Intel Core™ i5 CPU (M460), 2.53 GHz, 4 GB RAM, and Windows 7.

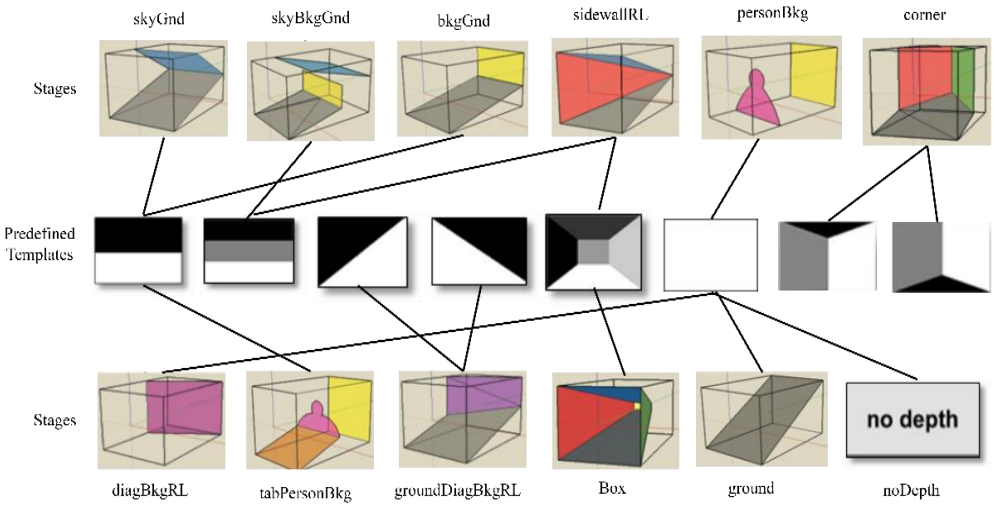
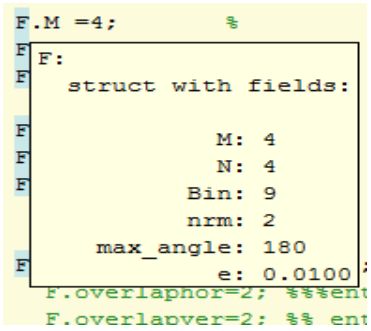


Figure 3.4: Association of predefined templates with stages. Stage models (3D scene) are defined in (Khan et al., 2020).

Before using the designed method on the images. We have tested our system to be sure that it is correctly worked. For this purpose, we have done testing of the segmentation-based feature extraction method which is given in Subsection 3.2.2. The evaluation of method on two different datasets is divided into two Subsections. Subsection 3.2.3 describes the performance of segmentation-based feature extraction method and the baseline methods on image dataset 1 (stage dataset). Subsection 3.2.4 describes the proposed method's performance on Dataset 2 (15-scene dataset) and the performance of the baseline methods that have used Dataset 2.

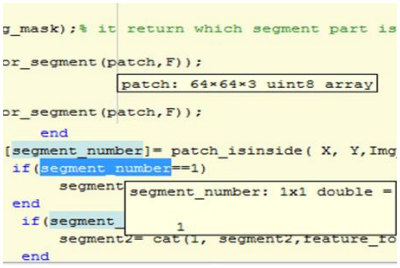
3.2.2 Testing of Segmentation-based Feature Extraction Method

In this Subsection, we tested segmentation-based feature extraction method discussed in Section 3.1. The method is implemented in the Matlab and code is verified it in following steps. The left side is indicating the expected value from the segmentation-based feature extraction method. Right side is indicating the output of the segmentation-based feature extraction method for a single input image. Horizontal lines differentiate the different steps.

Manually calculation and settings	Output of Segmentation-based feature extraction Method
Step1: Input RGB image Input RGB image: Size 256x256x3 pixels	 Image reference (B. Zhou et al., 2018).
Step2: parameters settings Parameters verification: We set the following parameters: 4x4 grid patches. HOG parameters: Bins=9, normalization =L_2-norm, angle= 180 degrees, ε constant = 0.01.	**Output:** Parameters verification 
Step 3: Template-based segmentation We apply templates (see Figure 2.8) and check that it is working properly. E.g. applying template (a) from Figure 2.8 with its two components. 	**Output:** Template–based segmentation of input image. Output segmented image has same size of input image and it has two segments (Seg1, Seg2) corresponding to an input template. 
Step 4: Patch size calculation	**Output:** Patch size calculation is shown here.

Input image size is 256x256x3 pixels. Then each patch size is 256x256/4= 64x64x3 pixels and e.g., first patch is belonging to top left corner, so it should be belonged to segment 1 (Seg 1).

And it is also shown that it belongs to Seg1.



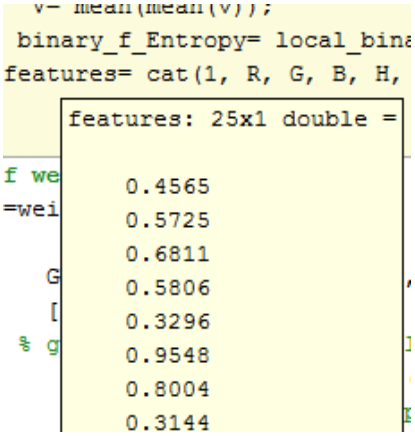
Segment_number =1, which indicates that it belongs to Seg1.

Step 5: Features extraction

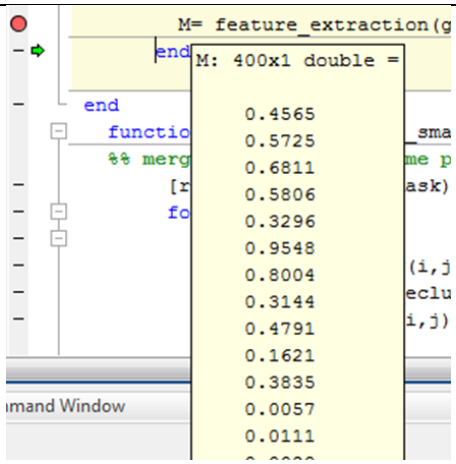
Each patch should has following 25 features: HOG: 9 features, Color: 6 features, Weibull distribution: 4 features, LBP-E: 6 features. And full image should have, 4x4x25= 400 features.

Output: Method obtains 25 features for each patch and it generates 400 features for whole image, see in screenshots below.

Each patch features: 25 features (data type: double)

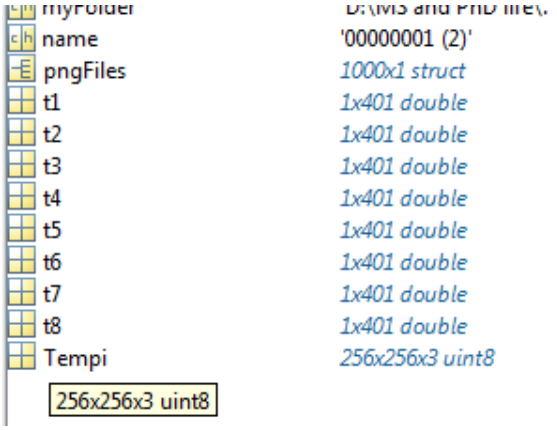


Whole image features: 400 features (datatype: double)



Step 6: As each template of Figure 2.8 is generating 400 feature vector. We have 8 different feature vectors in total. Each image has particular category. So, we keep the category information for each template. The total size of a feature vector becomes 401.

Output: Templates, t=1,2,..,8 are shown in screenshot with their feature vectors. However, size of each template-based features vector is 1x401 dimension double array in which 400 indicates total features of the image and one extra element of array is indicating the category type of the image.



Step 7: Training and testing of machine learning algorithm. As we test our method on a single image, thus, training and testing part are not feasible, although we test our method that it is properly work for training and testing process, when two medium size of the datasets are applied. Next, the sum-rule is also tested on their

prediction scores as the performance is shown in the Table 3.2 (last four rows).

3.2.3 Dataset 1: Stage Dataset

As the stage dataset is not publically available. A new stage dataset is constructed by combining the different datasets. The new stage dataset contains 1209 images in total. It consists of 300 images from 'Geometric Context' dataset (D. Hoiem, Efros, & Hebert, 2005), 481 images from 'Putting Objects in Perspective' dataset (D. Hoiem et al., 2006), 205 indoor images from the dataset (Hedau, Hoiem, & Forsyth, 2009), 132 images from 'Pixel-wise labeled image' dataset (Winn, Criminisi, & Minka, 2005), and 88 images from 'gettyimages' website and resized by 512×512 pixels . We have annotated them manually into the twelve 3D scene geometries, which are used to test the proposed method, followed (Nedovic et al., 2010). These geometries are explained in Subsection 2.1.2.

3.2.3.1 Experiments and Results for Stage Dataset

In first experiment, the different features of existing methods (Existing methods are summarized in Subsection 2.3, Table 2.1) are extracted for the stage dataset and their accuracy (see equation (2.24)) are calculated for linear, Gaussian, and quadratic SVM kernels. The SVM kernels can be easily selected from Matlab code (see Appendix I). Each image is divided by 4×4 patches and features, namely HOG (Dalal & Triggs, 2005), HSV, RGB, LBP-E, Weibull distribution (W) (J.-M. Geusebroek & Smeulders, 2005), Atmospheric scatting ('A') (Nedovic et al., 2010) are extracted from each image patch. These features are described in Subsections 2.1.4.1-4. To measure the texture information, the five feature of LBP per patch is calculated and then their entropy E value is measured (six features). It is because we are measuring the stage information where the small objects are ignored. The small number of feature per patch

provides more generic information. Therefore, we set it to 5 in our experiments. Nedovic et al. (Nedovic et al., 2010) use atmospheric scatting ('A') consists of mean and variance of saturation component (2 features) and color coefficient of RGB estimated by grey world algorithm (Weijer et al., 2007) (3 features per patch) (saturation and RGB correction coefficient are described in Subsection 2.1.4.2). We also applied the Gist (Oliva & Torralba, 2001b) on stage dataset to compare the feature effectiveness. Gist features implementation is publically given at (Oliva & Torralba, 2001a).

The dataset is divided into two parts: half for training and half for testing. The Table 3.1 shows the accuracy (see equation (2.24)) of each feature type. We combine the features into one set and perform different experiments. The best combination of different features and their accuracy are shown in Table 3.1. The feature set $L = \{HOG, HSV,\ RGB, W, LBP\text{-}E\}$ having length of 400 features for each image yields 69.50% accuracy, while by adding the Gist features (Oliva & Torralba, 2001b) with 32 features per image patch obtains 69.58% accuracy when the SVM with quadratic kernel is used. The improvement by adding Gist features is very limited. And it increases the length into 912 features per image. Therefore, we ignore the Gist features. The feature set L is optimal and can be used to learn the stages efficiently.

The second experiment is evaluated for measuring the effectiveness of segmentation-based feature extraction method using eight numbers of templates. Each template-based feature vector is individually fed to SVM classifier with the quadratic kernel to predict the class label for an input image and their average accuracy for each template is reported in Table 3.2.

Table 3.1: Accuracy of stage recognition for dataset 1 using different features.

No. of features/ image	Feature name and their different sets	Average accuracy of Stage recognition (%)		
		SVM Linear	SVM Quad.	SVM Gaussian
64	Weibul Dist. (W)	49.70	53.71	51.24
80	Atm. Scattering (A)	48.61	50.88	50.60
144	HOG	58.88	62.28	59.60
48	HSV	48.61	54.31	52.87
96	LBP-E	56.52	58.63	57.61
512	Gist	61.61	63.71	61.01
144	W+A	55.89	60.19	58.20
208	W+HOG	60.14	65.10	61.54
304	HOG+RGB+ HSV+W	62.73	66.72	64.43
336	HOG+HSV+ RGB +LBP-E	64.42	68.51	65.82
400	HOG+HSV+ RGB+W+LBP-E	65.01	69.50	66.43
912	HOG+HSV+ RGB+W+LBP-E+Gist	65.60	69.58	65.34

Raw data is given in Appendix S.

The Table 3.2 also shows the influence of different feature set with different number of templates where their associated classifiers outcome are combined by using sum rule (it is defined in Subsection 2.1.5.3, code is shown in Appendix J). For the features set (*HOG+HSV+ RGB+W*) used in Lou et al. (Lou et al., 2015) method are also tested on our method by using *T=6* templates without including indoor *corner* templates and with including *corner* templates (*T=8*). The average accuracy of stage recognition approaches to 77.49% by using *T=6* templates and it approaches 79.48% by using T=8 number of templates. The segmentation-based feature extraction method with full feature set (*HOG+HSV+RGB+W+LBP-E)* and with *T=6* templates achieves 80.40% accuracy of stage recognition. In addition, the segmentation-based feature extraction

method by using *T=8* templates yields 82.50% accuracy. Thus, using full feature set (*HOG+HSV+RGB+W+LBP-E*) and *T=8* templates allow outperforming by 5.01% accuracy of stage recognition. The confusion matrix for stage dataset is shown in Figure 3.5 (code is shown in Appendix K). The diagonal values show the precision for each class. It shows low performance in indoor categories, such as *box*, *corner*, and *DiagBkgRL*. This is due to less number of images for training and the larger variability of scene shape, amount of occlusions and diversity of objects present in these categories. E.g., some images of *corner* categories are confused with the similar category such as *Box*.

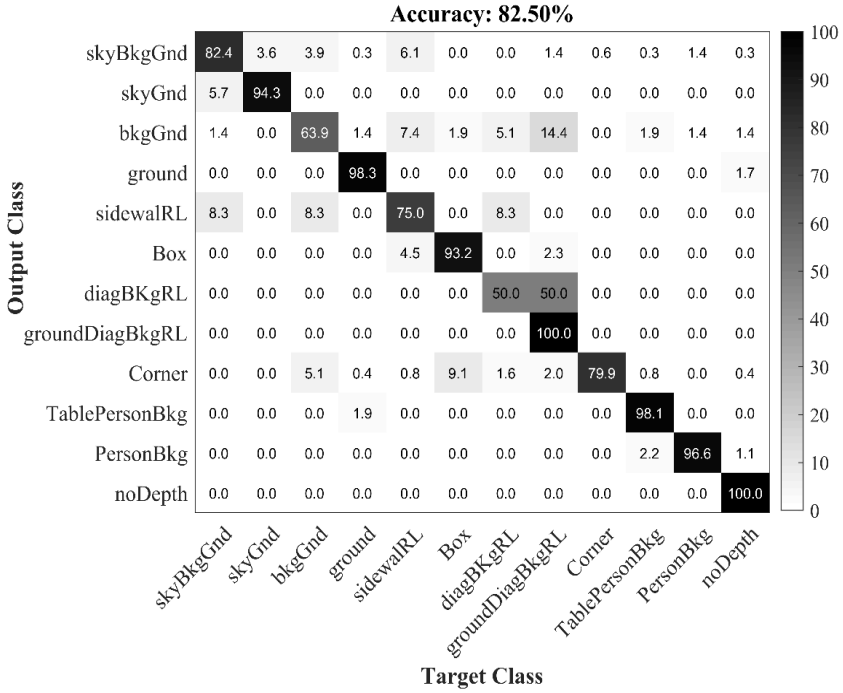


Figure 3.5: The confusion matrix of segmentation-based feature extraction method for dataset 1. Raw dataset is shown in Appendix T.

Table 3.2: Accuracy of stage recognition for dataset 1 using segmentation-based feature extraction method.

No. of features	Methods	Template(s)	Accuracy of stages (%)
400	HOG+HSV+RGB+W+LBP-E	(a)	68.59
		(b)	68.17
		(c)	66.25
		(d)	67.17
		(e)	66.58
		(f)	69.50
		(g)	68.59
		(h)	69.01
304	HOG+HSV+ RGB+W	(a)-(e)	77.49
304	HOG+HSV+ RGB+W	(a)-(h)	79.48
400	HOG+HSV+RGB+W+LBP-E	(a)-(e)	80.40
400	HOG+HSV+RGB+W+LBP-E	(a)-(h)	82.50

Raw data is shown in Appendices S and T.

3.2.3.2 Comparison With State-of-the-Art Methods

The state-of-the-art methods (Derek Hoiem et al., 2007; Nedovic et al., 2010; Oliva & Torralba, 2001b; Sanchez et al., 2013) are summarized in Table 2.1 and applied on the our stage ‘dataset 1’. We follow the Nedovic et al. (Nedovic et al., 2010) approach, where each image is divided into non-overlapped 4×4 local regions (patches). Then, the following methods are used to extract their features, namely Gist features (Oliva & Torralba, 2001b), Nedovic et al. (Nedovic et al., 2010) features set, and Hoiem proposed feature set (D. Hoiem et al., 2005; Derek Hoiem et al., 2007) features. Features of each existing method are extracted from each image patch and grouped

into a single feature vector. The Nedovic et al. (Nedovic et al., 2010) introduced features set namely, Atmospheric scattering (discussed above in 3.2.3.1), Perspective line, and parameters of Weibull distribution (described in Subsection 2.1.4.1) and these features are extracted from each image patch and concatenate into a single vector. The perspective line (P) (Nedovic et al., 2010) is another feature type where an anisotropic filter are applied at four different angles ($\theta = \{30°, 60°, 120°, 150°\}$) (see detail at (Nedovic et al., 2010), p.1678). Using maximum likelihood estimator (MLE), the parameters ($\alpha,\ \beta$) of Weibull distribution (see Subsection 2.1.4.1) are obtained for each angle (8 features). The source code is given at (Jan Mark Geusebroek, Arnold W. M. Smeulders, & Weijer, 2007). Next, the Hoiem proposed features (D. Hoiem et al., 2005; Derek Hoiem et al., 2007), namely color, texture, location and shape, and 3D scene geometry features are computed at each patch. The source code of Geometric Context features is available at (D. Hoiem, A.A. Efros, & Hebert, 2007). The Sánchez et al. (Sanchez et al., 2013) baseline method is also applied on stage dataset by using the MATLAB implementation that is given at (Mensink, 2012). The number of components for Gaussian Mixture Models (GMM) is set to 64. We run it on Ubuntu V. 18.04.3 by using same portable computer. After extracting the feature vectors of these methods, the SVM classifier with linear and quadratic kernel is applied. The 50% images are used for training and 50% for testing of stage recognition performance. The performance is measured in terms of accuracy (*Acc*), average precision (*Pr*), average recall (*Re*), and average *F-score* according to Subsection 2.1.7.

The experiment results of stage recognition are given in Table 3.3, where Hoiem proposed features (Derek Hoiem et al., 2007) has achieved 64.7% accuracy, and Gist features (Oliva & Torralba, 2001b) obtains 63.7% recognition accuracy. The next,

stage recognition accuracy of Nedovic et al. (Nedovic et al., 2010) method reaches to 59.8% with combining 'A' and 'W' features set, while it approaches to 60.29 % by combining 'A' and 'P' features set. The most effective performance from state-of-the-art methods is obtained by Sánchez et al. (Sanchez et al., 2013) that is reached to 72.40 %. In contrary, the segmentation-based feature extraction method achieves the recognition accuracy of 82.50%, which is outperform the Sánchez et al.(Sanchez et al., 2013) by 10.10%. On the other hand, the training and testing time were calculated for each method. The segmentation-based feature extraction method consumes more time compare to state-of-the-art methods because it uses the ensemble of classifiers but achieves superior accuracy of stage recognition.

Table 3.3: The Acc, Pr, Re, F-Score, training + testing time of state-of-the-art and segmentation-based feature extraction method are given for stage dataset 1.

Row #	No. of features	Method [Reference]	SVM (Kernel)	Acc %	Pr %	Re %	F-score %	Training + testing time (sec)
1	512	Gist (Oliva & Torralba, 2001b)	Quad.	63.71	69.41	52.02	55.09	16.03
2	992	Hoiem et al. (Derek Hoiem et al., 2007)	Quad.	64.70	66.81	51.63	55.74	76.75
3	208	Nedovic et al. (Nedovic et al., 2010) (P+A)	Gauss.	60.29	58.70	46.78	50.17	13.89
4	144	Nedovic et al. (Nedovic et al., 2010) (W+A)	Gauss.	59.80	59.11	45.85	49.47	13.44
5	3072	Sánchez et al. (Sanchez et al., 2013)	Quad.	72.40	73.4	59.90	63.61	68.45
6	400	Segmentation-based feature extraction method	Quad.	82.50	85.95	68.24	71.60	320.29

Raw data is given in Appendices S and T.

The stage recognition performance of the segmentation-based feature extraction method is also compared with pre-trained famous CNN frameworks, namely GoogLeNet (Szegedy et al., 2015), GoogLeNet365 (GoogLeNet trained on Places365 database (B. Zhou et al., 2018)), ResNet-50 (He et al., 2016), AlexNet (Krizhevsky et al., 2017), and VGG-16 (Simonyan & Zisserman, 2015). The Matlab code of deep CNN model is given Appendix L. We choose ResNet-50 model because of the stage dataset is small size dataset and deeper models, such as ResNet-101 may overfit on it. The stage dataset is randomly divided into two parts, with 80% used for training and 20% is used for testing. Each CNN model uses BP algorithm with stochastic gradient descent for its parameters updates (described in Subsection 2.2.1.4). The training parameters are defined as follows: the size of the batch is 10, the number of epochs is set 20, the momentum, and learning rate are set to 0.9 and 0.0003, for each epoch, respectively. All deep CNNs models are trained by using these parameters setting and their *Acc*, average *Pr*, average *Re*, and average *F-score* are calculated for testing dataset (see equations (2.24-30)). Additionally, we re-implemented CNN-SVM (Kim, Kavuri, & Lee, 2013; Patalas & Halikowski, 2019) by using ResNet-50 (He et al., 2016) replacing the FC layers with linear SVM classifier. The feature vector of ResNet-50 is normalized before feeding it into the linear SVM classifier (code is shown in Appendix M). Similarly, ELM is a new approach, which shows high performance for image classification when it uses with deep CNN model (G. Huang et al., 2015b), detail of ELM is given in Subsection 2.1.5.2. Instead of SVM, we apply CNN-ELM in which the ELM is used for classification of the input stream that can be obtained by CNN based method by replacing the FC layers with ELM classifier. We utilize pre-trained ResNet-50 (He et al., 2016) for feature extraction, and FC layers are replaced with ELM by setting neurons, $L = \{1000,\ 2000, 3000, 4000, 5000,\ 6000\}$,

Table 3.4: The performance of stage recognition for dataset 1. The Acc, avg. precision (Pr), avg. recall (Re), avg. F-score, training and testing time (sec), fusion time (sec) of classifiers are given.

Row#	Method	Acc %	Pr %	Re %	F-score %	Training time	Testing time	Fusion time	Total time
1	GoogLeNet (Szegedy et al., 2015)	78.93	81.39	72.75	74.88	11518	46.31	-	11564.31
2	GoogLeNet365 (Szegedy et al., 2015)	82.02	79.36	75.24	75.66	13324	47.32	-	13371.32
3	ResNet-50 (He et al., 2016)	82.23	79.49	77.22	76.14	30099	37.07	-	30136.07
4	AlexNet (Krizhevsky et al., 2017)	78.93	80.10	69.97	73.34	5760	38.70	-	5798.70
5	VGG-16 Conv. (Simonyan & Zisserman, 2015)	79.75	77.52	75.78	75.55	12934	42.54	-	12976.54
6	CNN+SVM (Patalas & Halikowski, 2019; Penatti, Nogueira, & Santos, 2015)	80.58	83.48	75.24	76.24	22.72	10.16	-	32.88
7	CNN+ELM (S. Liu & Deng, 2015)	76.86	70.32	81.43	73.45	22.41	0.51	-	22.92
8	Segment.-based feature extraction method	86.25	86.91	75.32	77.00	314.60	2.22	0.028	316.85

Raw data is shown in Appendix U.

$C = 2^5$, and activation function is set to 'sigmoid function' (Matlab code is shown in Appendix M). Finally, the maximum stage accuracy of CNN-ELM is obtained by using L=6000 neurons as shown in Table 3.4. To compare the recognition performance

with CNN models, the same set of 967 images (80%) for training and 242 (20%) for testing are used to evaluate the segmentation-based feature extraction method. The SVM is used with its polynomial (Quadratic) kernel. Time of training and testing of each experiment is individually measured. In Table 3.4, the ResNet-50 (He et al., 2016) achieves stage recognition accuracy of 82.23%, which is the highest over the deep CNN models, while the segmentation-based feature extraction method achieves 86.25% recognition accuracy. It seems that the segmentation-based feature extraction method outperform the ResNet-50 (He et al., 2016) by 4.02% when small scare stage dataset is used. In this scenario, the segmentation based feature extraction method performs well. To compare the computational cost of the segmentation-based feature extraction method in terms of time taken for training and testing, the segmentation-based feature extraction method took 316.85 sec in total versus ResNet-50 (He et al., 2016) which takes 30136.07 sec. On the other hand, CNN-ELM (S. Liu & Deng, 2015) method although shows highest recall rate (81.43%) and took shortest time of stage recognition but its maximum accuracy approaches to 76.86% by using 6000 neurons which is comparably very low as illustrated by the corresponding accuracy values.

3.2.4 Dataset 2: 15-Scene Images Dataset

The ’15-scene image dataset’ (Lazebnik et al., 2006) is a publically accessible dataset which consists of fifteen categories of indoor and outdoor images, there are total of 4,485 images with an average of 300×250 pixels (summary is shown in Table 2.2). The images corresponding to each category are shown in Figure 3.6. It is a commonly used dataset for the assessment of scene classification studies. The details about the class labels and number of images per class is referred to (Lazebnik et al., 2006). This dataset is utilized to evaluate our segmentation-based feature extraction method as categories representing the particular scene images such as Coast category is like as

sky-ground category. Therefore, the same set of templates are used for template-based segmentation. E.g., bedroom or street categories can be handle by *corner* or *box* templates.

Figure. 3.6: Examples of 15-scene images dataset. Each image represents a corresponding category type.

3.2.4.1 Experiments and Results for 15-Scene Images Dataset

For evaluation of the segmentation-based feature extraction method on this dataset, as stated in Section 3.2.1, we followed the same experimental setup. To ensure a fair comparison with existing research in terms of recognition accuracy (see Table 2.1), the 100 images are selected from each of the class for training and remaining for testing. By following the Subsection 3.2.3.1, the feature set $L = \{HOG, HSV, RGB, W, LBP\text{-}E\}$ is extracted for each image by using eight predefined templates (see Figure 2.8). Then for each template, an individual model is trained. Next, these models are used to classify the test data and their performance are given in Table 3.5. Later on, these classifiers output are combined using majority voting, max and sum-rule (described in Subsection 2.1.5.3) and *Acc*, average *Pr*, *Re* and *F-score* are measured (code is given in Appendix K). The results demonstrate that maximum recognition

accuracy is obtained by utilizing the sum-rule is reached to 92.58% (Matlab function is shown in Appendix J) and its testing and its fusion time for eight classifiers is reached to 110.48 sec.

Table 3.5: The performance of the segmentation-based feature extraction method for 15-scene dataset. Templates (a)-(h) are followed by figure 2.8. The majority vote, max and sum rules are used to evaluate the segmentation-based feature extraction method.

Pred. Templates	No. of features	Acc. %	Pr %	Re %	F-Score %	Avg. Training time	Avg. time of Testing + Fusion
(a)	400	80.85	79.96	79.61	79.69	486.61	-
(b)		79.40	79.10	77.94	78.36		-
(c)		78.80	78.41	77.33	77.72		-
(d)		79.55	78.33	78.33	78.53		13.71
(e)		79.30	77.90	77.99	77.84		-
(f)		82.94	82.62	82.07	82.28		-
(g)		80.07	79.28	79.01	79.05		-
(h)		78.77	77.36	77.16	77.20		-
Majority vote((a)-(h))		86.46	86.19	85.84	85.96	-	109.88
Max-rule((a)-(h))		89.36	89.32	95.41	90.74	-	110.40
Sum-rule((a)-(h))		92.58	92.59	95.96	93.19	-	110.48

Raw data is shown in Appendix V.

3.2.4.2 Comparison with State-of-the-Art Methods for 15-Scenes Images Dataset

We compared segmentation-based feature extraction method performance with the recent methods that use 15-scene image dataset and show state-of-the-art recognition performance (see Table 2.1). The Table 3.6 exhibited that the segmentation-based

feature extraction method representation gains the highest recognition accuracy (see equation (2.24)). It provides 5.51% higher accuracy as compared to Zafar et al. (Zafar, Ashraf, Ali, Ahmed, Jabbar, & Chatzichristofis, 2018) (OVH) method. Lin et al. (Lin, Fan, Zhu, Miu, & Kang, 2017) use "local visual feature coding based on heterogeneous structure fusion (LVFC-HSF)" and obtains 87.23% recognition accuracy. Zafar et al. (Zafar, Ashraf, Ali, Ahmed, Jabbar, Qureshi, et al., 2018) use "concentric weighted circles histogram (CWCH)" to obtain a robust performance for 15-scene images dataset and reached to 88.04% accuracy. The most recent approaches, "hybrid geometric spatial image representation (HGSIR) method" (Ali et al., 2018) achieves the maximum recognition accuracy of 90.41%. Alternative to these methods, the deep VGG-16 (Simonyan & Zisserman, 2015) method shows 88.65% recognition accuracy.

Table 3.6: Comparison with proposed and state-of-the-art methods in terms of recognition rate while using 15-scene image dataset.

Methods	Accuracy%
Zafar et al. (Zafar et al., 2018) (OVH)	87.07
Lin et al. (Lin et al., 2017) (LVFC-HSF)	87.23
Zafar et al. (Zafar et al., 2018) (CWCH)	88.04
Ali et al. (Ali et al., 2018)(HGSIR)	90.41
VGG-16 (Simonyan & Zisserman, 2015)	88.65
Segmentation-based feature extraction method	92.58

The segmentation-based feature extraction method uses low dimensional feature set and achieves the highest accuracy of scene recognition because the feature are extracted by following image geometry structure and fusion at decision-level by

utilizing sum rule is clear evidence from Table 3.5 and Table 3.6 that the segmentation-based feature extraction method gains the highest recognition rate.

3.3 Summary

In this research study, a novel segmentation-based feature extraction method is proposed. In a segmentation-based feature extraction method, the feature set such as HOG, color (RGB, HSV), parameters of the Weibull distribution, and local binary patterns and its entropy value are extracted for each local region and combined into a single vector based on the template-based segmentation. A set of templates that are associated with individual classifiers are used in the segmentation-based feature extraction method. Each template defines a particular rough structure of 3D scene geometry. Thus, for each template, the individual classifier is trained. Finally, the obtained results of the ensemble of classifiers are fused using sum-rule. The segmentation-based feature extraction method is evaluated on a two different datasets. First dataset is a new stage dataset having 1209 images. Compared to the state-of-the-art methods, our segmentation-based feature extraction method obtained significantly improvement in stage recognition accuracy on a new dataset. The segmentation-based feature extraction method's results illustrate that the information fusion of different features by following the template-based segmentation and fusion using sum-rule provides a higher accuracy of stage recognition than the state-of-the-art algorithms. E.g., compared to Sánchez et al. (Sanchez et al., 2013) method, the segmentation-based feature extraction method improves stage accuracy by 10.10% and compared to CNN based methods, it achieves the better performance in most scenarios. Next, the segmentation-based feature extraction is evaluated on the 15-scene image dataset. This dataset is categorized on the base of global image structure and classes are limited as well. Thus, we use it in our study and obtained significantly high accuracy compared

to recent state-of-the-art approaches, such as Ali et al. (Ali et al., 2018) obtain 90.41% recognition accuracy on 15-scene images dataset, while the segmentation-based feature extraction approach achieves 92.58% recognition accuracy.

The segmentation-based feature extraction method does not require the very high-performance hardware and a large dataset for training that are typically required for CNN-based methods. Moreover, the segmentation-based feature extraction method is mainly designed for 3D scene geometry recognition, which can be used as prior knowledge for pixel-level 3D layout extraction. A statistical evaluation of the experimental results illustrated that the recognition rate of this method for both datasets was higher than the state-of-the-art methods in terms of accuracy and F-score value. This is because features are extracted for each sub-region separately and then grouped together for that region, indicating that they have similar statistical values, which reduces the intra-class variation.

The segmentation-based feature extraction method can be applied on other scene datasets. If the categories of a dataset are based on scene geometry structure, then it may need to change the number of input templates. For example, if a dataset has only outdoor images, then it does not need to use indoor templates, e.g., box or corner. Furthermore, if a dataset contains objects such as a person or animals, etc., then the template structures can be adjusted according to the rough shape of that category.

Chapter 4

STAGE DATASET AND 3D SCENE RECOGNITION METHOD USING TEXTURE GRADIENT AND DEEP FEATURES FUSION

In this chapter, the two main tasks are considered. Firstly, it is investigated that a medium size 3D scene dataset is not available. A small scale stage dataset (1209 images) is introduced in our previous Chapter (Chapter 3). However, the number of images in a per category is very small, which is not enough for recent deep-learning based methods that need a large scale image dataset for training process (B. Zhou et al., 2018). Therefore, we introduced a medium scale dataset of 3D scene recognition, in which 1000 images in each class, with fixed size of 256×256 pixels. The 2nd task of this chapter is to verify the dataset by using baseline CNN models and to introduce a novel method based on the texture gradient features (TGF) and CNN features (also called as 'deep features') fusion into a single feature vector (TGF-DeepFF). The texture gradient features are measured by estimating the parameters of Weibull distribution for $n \times n$ local region of the input image (detail is given in Subsection 2.1.4.1). The feature combination methods, such as (Khoo, Goi, Chai, Lai, & Jin, 2018; Xin et al., 2018) show sufficient accuracy in biometric recognition models by fusing the biometrics trait features. We are inspired by the progress of these approaches and have implemented a new methodology focused on 3D scene recognition. The fused feature vector is classified by using SVM and ELM classifiers (described in

Subsections 2.1.5.1-2). ELM classifier is applied because it is simple to use, provides satisfactory performance, and learns thousands of times faster than traditional feedforward network learning algorithm like BP algorithm (Huang et al., 2006), p.490, (G. Huang et al., 2015b), p.19. Furthermore, (Huang et al., 2006) claim that it reaches to smallest training error with smallest norm of weights. Researchers take benefits of ELM in different fields such as medical diagnosis (Özyurt, 2020), tumor detection (Özyurt et al., 2020), and other engineering practice (Lei et al., 2021). Also, (Cheng et al., 2015; J. Tang, Deng, Huang, & Zhao, 2015; Wang et al., 2020; Wang, Peng, & Lin, 2021; Weng, Mao, Lin, & Guo, 2017) applied it to remote sensing images, and indoor and outdoor image scene (e.g. 15 scene dataset) classification. Inspired by their results of scene classification, we utilize ELM classifiers to train a discriminative model.

Design of 3D scene recognition method using TGF-DeepFF is given in Subsection 4.1. The new dataset detail, implementation detail, testing, and experimental results are given in Subsection 4.2. Summary of this Chapter is given in Subsection 4.3.

4.1 Design of 3D Scene Recognition Method using Texture Gradient Features and Deep Feature Fusion

In this section, we explain the TGF-DeepFF, two stream features with fusion method of 3D scene classification. The two-stream is indicating that the features are extracted in two different ways: deep CNN (1^{st} stream) and TGF (2^{nd} stream). The novelty of this method compared to Algorithm 2.1, are utilizing the deep features and texture gradient features in two different streams, and then combined them for each input image. Texture gradient contains rich information of 3D scene geometry and when it combines with deep features at FC layer, it improves the discriminative features of 3D

scene recognition. Next, process of training and testing are same as Algorithm 2.1. The TGF-DeepFF method is defined by Algorithm 4.1 and illustrated in Figure 4.1. It has following inputs: N training images, M testing images, $\mathcal{N}$ total number of images, CNN model, S classes, Y_N training labels, Y'_M testing labels. The output of the method is: performance metric in terms of Acc, Pr, Re, F-score. The detail of the method by following Algorithm 4.1 is given following main points:

1) In first stream, Figure 4.1 shows that the deep features are extracted by three different steps: i) input layer, ii) feature extraction layer, iii) output layer ,which is also called as final layer. The detail of these layers is given in Subsections 2.2.1-3. Input layer is indicating that input RGB images are given to the pre-trained CNN model. Next layer is indicating the feature extraction layer. It has several convolutional and pooling layers, as discussed in Subsection 2.2.1. Finally, the features are obtained at GAP layer. CNN model returns a feature vector, F_j^L, having length L, for each input image I_j, j=1,2,.. $\mathcal{N}$, $\mathcal{N}$ is number of the images of scene dataset (see line 1 of Algorithm 4.1). The function, $deep\ features\ extraction(.)$ is used which takes pre-trained CNN model, input images, $\mathcal{N}$, and returns features $F_{\mathcal{N}}^L$, in the Algorithm 4.1. L is length of features for each input image.
2) In second stream of feature extraction (see Figure 4.1, lines 2-4 of Algorithm 4.1) are extracted by using function, $Texture\ gradient\ features\ extraction(.)$. It takes image I_j, and η, and returns feature vector, $Tg_j^{L'}$. η is an input for $\eta \times \eta$ patches. In this function, each input image is partitioned into $\eta \times \eta$ patches and TGF (f_i, $i = 1,2, \dots, \eta \times \eta$) are extracted for each patch, as described in Subsection 2.1.4.1. It provides 4 features for each image patch. For whole image I_j, it generates

$L' = \eta \times \eta \times 4$ dimension vector, , $Tg_j^{L'}$, where $j = 1,2,..\mathcal{N}$. $\mathcal{N}$ is number of images of the scene dataset.

3) After extracting the both feature vectors for every input sample. The next step, then to normalize and combine both feature vectors into a single vector. Thus, z-score normalization is utilized on both feature vectors, individually. Then both feature vectors are combined into a single vector. Z-score normalized is available in Matlab function, $Fn_j = normalize(F_j)$, where F_j is a input feature vector of image I_j, and Fn_j is a normalized output vector of jth image. The both feature vectors of deep CNN (F^L) and texture gradient features (, $Tg_j^{L'}$) are normalized using normalization function (see lines 5-7 of Algorithm 4.1).

Algorithm 4.1 Method of 3D Scene Recognition using texture gradient and deep CNN feature fusion

---

Input: N training images, CNN model, $\mathcal{N}$ total number of images, S classes, Y_N training labels, Y'_M testing labels, η as input for $\eta \times \eta$ patches

Output: Acc, Pr, Re, F-score

1: $F_{\mathcal{N}}^{L} = deep\ features\ extraction(\text{CNN model}, \mathcal{N})$ // Features $F_{\mathcal{N}}^{L}$ for $\mathcal{N}$ images, L is length of feature vector, CNN model is pre-trained model as input

2: **for** j=1:$\mathcal{N}$ **do** // for each image I_j

3: $Tg_j^{L'} = Texture\ gradient\ features\ extraction(I_j, \eta)$ // feature vector for image I_j, L' is length of feature vector, η is input for $\eta \times \eta$ patches

4: **end for**

5: **For j=1:** $\mathcal{N}$ **do**

6: $\boldsymbol{F}_j^D = feature\ fusion(normalize(F_j^L), normalize(Tg_j^{L'}))$ // combine two different features vectors, $D = L + L'$.

7: **End For**

// Training & testing

8: $\boldsymbol{F}^{D'} = \{(\boldsymbol{F}_j^D, Y_j)\}_{j=1}^{N}$ // N is a number of training samples

9: $Trained\_CL = CL(\boldsymbol{F}^{D'})$, //where $\boldsymbol{F}^{D'}$ labeled features is used to train the classifier

// Testing (prediction)

10: **for** j=N+1: $\mathcal{N}$ **do** // loop on M test images

11: $l_j = Classify(TrainedCL, \boldsymbol{F}_j^D)$ // l_j is predicted label for image Ij, belongs to S classes.

12: **end for**//j

//Performance measures
13: [Acc, Pr, Re, F-score]=Calculate_Measures ($\boldsymbol{l}$, Y'_M). // Y'_M is a true testing labels, $\boldsymbol{l}$ is predicted labels for M images.
End Algorithm

4) Next, after normalization of both feature vectors, the results vectors are merged into a one vector. Thus, $feature\ fusion(.)$ function is used, which takes normalized input feature vectors, F_j^L , $Tg_j^{L'}$, and returns combined feature vector, $\boldsymbol{F}_j^D$ for each image, I_j, j=1,2..,ℕ. $\boldsymbol{F}_j^D$ is the fused feature vector, with dimension D. The length of the combined features, D, is equal to the sum of the length of the two feature vectors ($D = L' + L$). The corresponding Matlab code is shown in Appendix N. The code takes both features vectors, normalized them and then returns the fused feature vector for each image.
5) Following Figure 4.1, next point is to train the machine learning algorithm and then test to that trained model. The feature vectors are ready for training the machine learning algorithm. By following the description of the Subsection 2.1.5, we utilize N number of images features for training and rest of the M (M= ℕ –N) images features are used testing by following training Y_N and testing Y'_M labels. The feature vector, $\boldsymbol{F}_j^{D'} = (\boldsymbol{F}_j^D , Y_j), j = 1,2,..N$, with N labels are generated by using (2.15) (see line 8 of Algorithm 4.1), and used as input to classifier, CL, to train a model, $Trained\_CL$ (see line 9 Algorithm 4.1). ELM is used as classifier to train a model. The feature vectors of the M testing images are used to evaluate the learned model. Suppose that the feature vector of the M testing images are $\boldsymbol{F}_j^D , j = 1,2,..M$, and are used as inputs to the classify(.) function with trained model, $Trained\_CL$, (see lines 10-12 of Algorithm 4.1), which predicts the class label, l_j, of each input image

I_j (testing process is given in Subsection 2.1.6, (2.19)). The predicted label is belong to S, and S is categories of 3D scene geometries.

6) Finally, the method calculates the Acc, Pr, Re, and F-score of the M testing images using equations (2.24-2.30). The predicted labels, $\boldsymbol{l}$, and ground truth labels, Y'_M, are used to measure the performance metric. The metric calculation using Matlab function is given in Appendix K. The Matlab code takes true labels, and predicted labels and generates these metric.

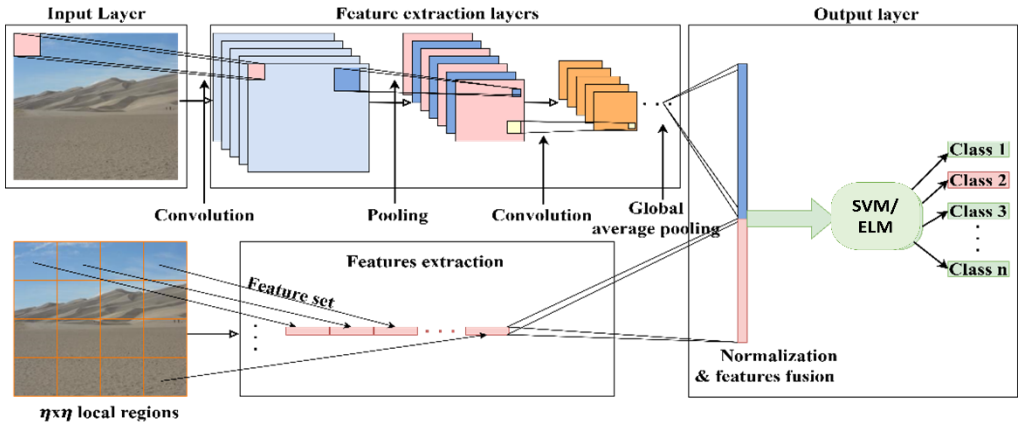


Figure 4.1: Texture gradient and deep features fusion based 3D scene recognition method.

4.2 Implementation, Testing of TGF-DeepFF Method, and Experiments on Stage Dataset 2

The MATLAB '2019a' is used to implement the TGF-DeepFF method and it is run on the same system used for segmentation-based feature extraction method (see Subsection 3.2.1). Detail is described in subsections below: the new dataset is given in Subsection 4.2.1. The implementation detail is described in Subsection 4.2.2. Testing of the TGF-DeepFF method is given in Subsection 4.2.3. Finally, experimental results are illustrated in Subsection 4.2.4.

4.2.1 Description of Dataset

We develop a novel stage dataset, which contains 12000 images in total. The source of the images are Places dataset (B. Zhou et al., 2018). The size of each image is defined by 256×256 pixels. All the images are manually labeled into the twelve different *stages* (3D scene geometries) (Nedovic et al., 2010), which are discussed in Subsection 2.1.2. We call it as 'Stage dataset 2'. We have annotated 1000 images for each category as following the structure of an image that which *stage* it belong to. Figure 4.2 illustrates the categories and their corresponding images. This dataset consists of 12000 images in total.

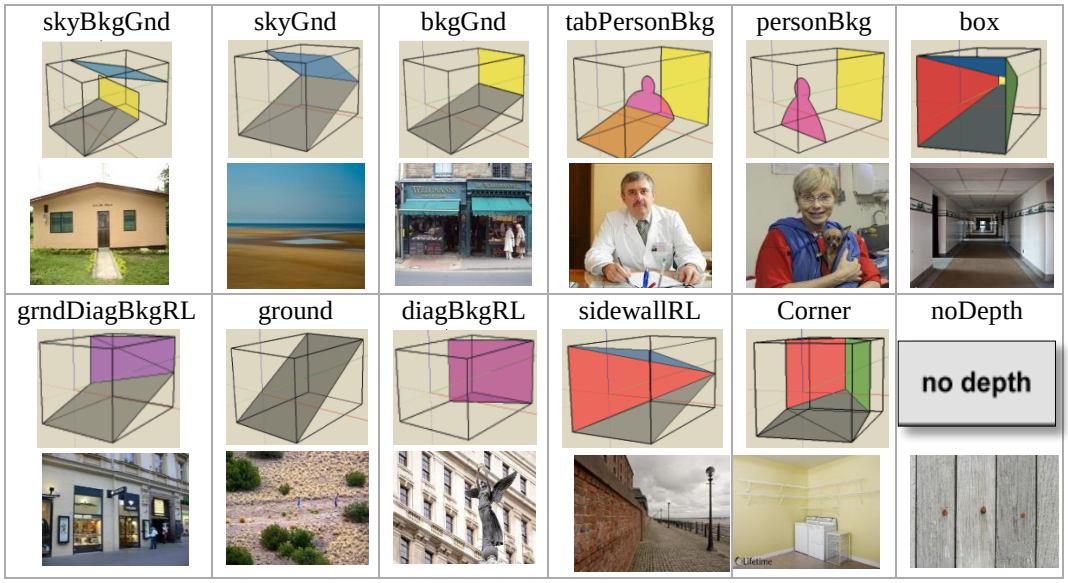


Figure 4.2: Examples of 'stage dataset 2' and their categories. Images are originated from Zhou et al. (B. Zhou et al., 2018). The reference of 3D scene geometries images is (Nedovic et al., 2010).

4.2.2 Implementation Detail

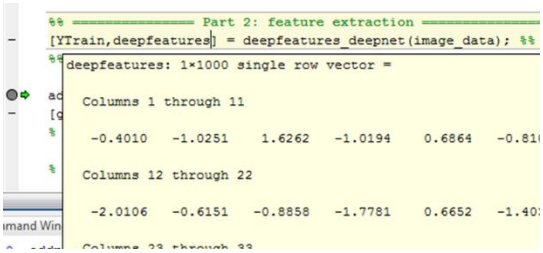
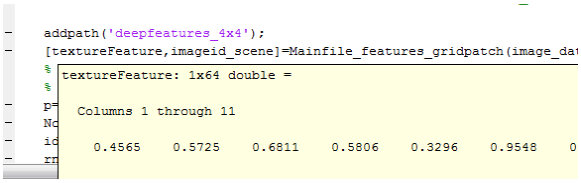
For feature extraction, in first stream, we utilize the pre-trained GoogLeNet (Szegedy et al., 2015) architecture for deep feature extraction because it is less deeper compared to ResNet (He et al., 2016) and VGG-16 (Simonyan & Zisserman, 2015) architectures and generates discriminative features for medium scale dataset. The detail is given in

Subsection 2.2.2. These CNN architectures generate 1000 deep features for each input image by using GAP. Matlab code of deep feature extraction is shown in Appendix M. In second stream, the parameters of Weibull distribution are calculated for 4×4 patches of input image. For each patch, these parameters (α, β) are calculated in both vertical and horizontal position as described in Subsection 2.1.4.1. In this way, the 64 features are measured for every input image. Matlab code is shown in Appendix O. After that, the normalization is utilized for both feature vectors and these features are combined into one vector. Matlab code of feature combination is shown in Appendix N. It generates, 1064=1000+64, length of the feature vector. For training the model, we adopt an ELM classifier to learn the 3D scene structure of train images. The 80% images are used for training and 20% images are used for testing. Then, for testing images, the performance is measured in accuracy, precision, recall and F-score (performance metrics are defined in Subsection 2.1.7). Matlab code is given in Appendix K. The method is tested many times for consistent comparison of results. Finally, results are reported by taking the average performance over the total number of tests.

4.2.3 Testing of Texture Gradient and Deep Features Fusion Based Method

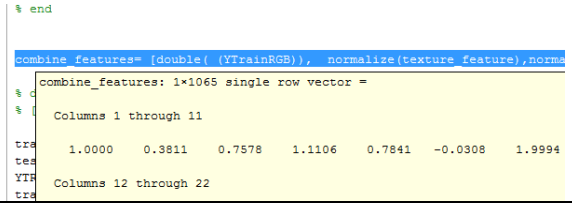
In this section, we test TGF-DeepFF Method implementation in different steps, which are given below.

The left side is indicating expected value from the TGF-DeepFF method. Right side is indicating the output of the TGF-DeepFF method for a single input image. Horizontal lines differentiate the different steps.

Manually calculation and settings	Output of TGF-DeepFF Method
Step1: Input RGB image Input RGB image: Size 256x256x3 pixels.	 Image reference (B. Zhou et al., 2018).
Step2: Extract deep CNN features We extract the deep CNN feature for an input image. It is not wise to use a single image for feature extraction. However, we are testing the system using a single image. The GoogleNet provides 1000 image features for each input image.	**Output:** Deep feature vector for single image is shown in screenshot below. It generated 1000 features for a single input image. 
Step 3: Extract Texture gradient feature extraction We extract parameters of Weibull distribution features from each image patch. 4x4 patches are used and, which generates 64 features for each input image.	**Output:** Texture gradient features are shown in screenshot. It is 64 features for a single image. 
Step 4: Features normalization and combination In this step, features are normalized first and then combined into single	**Output:** The screenshot shows that features are first normalized using Matlab function $normalize(.)$ and then the features are combined into a single vector, yielding 1065

vector. It length should be 1064 features. By adding the class label, 1, feature vector length becomes 1065 in total.

features. The output is 1064 features for a single image. It contains one more element which is indicating the class label of the image.



Step 5: Training and testing step

In this step, the extracted features are ready to use for training the machine learning algorithm. Here, in testing process, we use one image to test the system which is not feasible for training and testing the model. Therefore, the next Section 4.2.4 provides the training and testing results of TGF-DeepFF method.

4.2.4 Experimental Results

The experimental results are given in Table 4.1. We evaluate the performance of the deep features and then introduce the 2nd stream of TGF as parallel to the deep features. For this reason, we utilize well-known pretrained CCN architectures, including GoogLeNet (Szegedy et al., 2015), ResNet-50 (He et al., 2016), AlexNet (Alex et al., 2012), and VGG-16 (Simonyan & Zisserman, 2015) to find the well-fitting architecture of medium scale scene geometry dataset. Each architecture is separately applied on the same dataset and deep features are extracted at FC layers (given in Table 4.1) and then both linear SVM and ELM classifiers are applied for scene geometry recognition. The ELM parameters are set as follows: activation function set to 'sigmoid function', hidden neurons, L=12000, and $C=2^{-5}$, while the SVM kernel is set to linear with C=1. Beside this, these deep CNN architectures are applied with BP

algorithm. The training parameters of CNN models are defined as follows: the minimum batch size is set to 10, total number of epoch are set to 20, learning rate is set to (0.0003, 0.0005), and momentum is set to 0.9. The Matlab code is shown in Appendix L. The results demonstrate that GoogLeNet architecture achieves 79.33% and 79.92% recognition accuracy when SVM and ELM classifiers are applied, respectively, and it reaches 82.04% with BP algorithm. On the other hand, VGG-16 architecture obtains 72.21% and 72.83%, accuracy with SVM and ELM classifiers, respectively, and even with BP algorithms, it achieves 80.88% recognition accuracy, which is 1.16% and 1% lower than GoogLeNet and ResNet-50 architectures, respectively (see Table 4.1). In addition, VGG-16 took 2833.53 mins when used with BP algorithm. Meanwhile, ResNet-50 architecture did not perform well compared to GoogLeNet, because it may overfit on medium scale scene dataset. Similarly, AlexNet architecture obtains the lowest recognition accuracy of 70.54%, 72.46%, and 78.13%, for SVM, ELM, and BP algorithms, respectively, compared the GoogLeNet, ResNet, and VGG-16 architectures. In conclusion, GoogLeNet architecture achieves the highest accuracy of scene geometry recognition because it is less deep than VGG-16 and ResNet architecture. It takes advantage of inception module to get discriminative information of scene geometry for medium scale dataset. These experiments show (see Table 4.1) that ResNet and GoogLeNet architectures perform well compared to AlexNet and VGG-16 architectures for scene geometry structure recognition. As the GoogLeNet architecture achieves the best results of scene geometry recognition for all the three classification methods, therefore, we use it as backbone of our proposed method in which the GoogLeNet features are combined with Weibull distribution features. It reached the best performance of 85.54% and 86.29% for SVM and ELM

classifiers, respectively. It demonstrates that the two-stream architecture provides superior performance of 3D scene recognition than a single CNN architecture.

Table 4.1: Experimental results on 'stage dataset 2' using feature fusion method.

Methods	Training method	Features size	ACC%	PR%	RE%	FS%	Training + testing time/sec
Param. Weibull	SVM	64	37.33	36.20	37.28	36.52	182.99
Param. Weibull	ELM	64	40.52	39.28	40.52	39.60	138.75
Deep features of GoogLeNet Arch.	SVM	1000	79.33	79.70	79.66	79.63	55.59
Deep features of GoogLeNet Arch.	ELM	1000	79.92	79.55	80.26	79.78	179.86
Deep features of AlexNet Arch.	SVM	1000	70.54	70.35	70.75	70.49	60.41
Deep features of AlexNet Arch.	ELM	1000	72.46	71.49	72.72	71.37	154.03
Deep features of ResNet Arch.	SVM	1000	77.04	77.02	77.06	76.96	27.93
Deep features of ResNet Arch.	ELM	1000	76.62	75.75	76.66	75.78	349.44
Deep features of VGG-16 Arch.	SVM	1000	72.21	71.72	71.97	71.80	79.26
Deep features of VGG-16 Arch.	ELM	1000	72.83	71.09	72.49	71.25	353.23
TGF-DeepFF	SVM	1064	85.54	85.72	85.62	85.63	38.65
TGF-DeepFF	ELM	1064	**86.29**	**85.92**	**86.16**	**85.96**	278.37
GoogLeNet (Szegedy et al., 2015)	BP Algorm.	-	82.04	82.17	82.09	82.11	659.1min
AlexNet (Krizhevsky et al., 2017)	BP Algorm.	-	78.13	77.76	78.13	77.81	517.16 min
VGG-16 Net (S. Liu & Deng, 2015)	BP Algorm.	-	80.88	80.50	80.87	80.36	2833.53 min
ResNet-50 (He et al., 2016)	BP Algorm.	-	81.88	81.92	81.88	81.73	1458.33min

*Bold text indicates the best result. Raw Data is given in Appendix W.

The performance at each class-level can be observed in confusion matrix, given in Figure 4.3. However, results demonstrate that ELM classifier outperform the linear SVM in terms of Acc, Pr Re, and F-score for scene geometry recognition. On the other hand, ELM classifier took more time in training and testing compared to linear SVM classifier because of use of large number of hidden neuron. Experimental results in Table 4.1 show that our TGF-DeepFF method outperforms the baseline CNN models by using same number of training and testing images. The maximum accuracy among standard CNN models is obtained by GoogLeNet (Szegedy et al., 2015), which is 82.04%. On this other hand, ResNet-50 achieves 81.88% recognition accuracy.

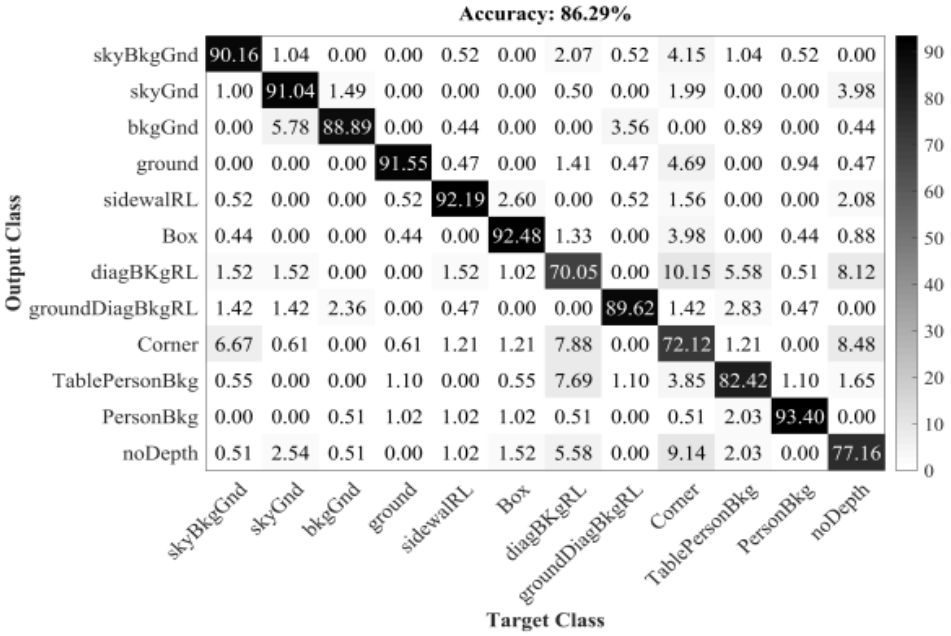


Figure 4.3: Confusion matrix of TGF-DeepFF method of ‘stage dataset 2’.

4.3 Summary

In this chapter, we have two contributions. First, introducing a new stage dataset, ‘stage dataset 2’, of 3D scene geometries, which was not available publically. It consists of 12000 scene images, equally distributed on 12 different stages. Each class contains 1000 images. Second contribution, introducing a novel 3D scene geometry

recognition method, TGF-DeepFF that is based on the combination of deep features of CNN model and texture gradient features for $n \times n$ patches of an image. Then, the output features are combined into a single vector that is used for learning the stage model. The two different classifiers, ELM and linear SVM, are utilized to test the TGF-DeepFF performance. TGF-DeepFF method is applied on the new 'stage 2 dataset'. The well-known CNN architectures, including AlexNet, VGG-16, GoogLeNet, ResNet architectures are studied for the purpose of learning the scene geometry representation at FC layers, and finally results exhibited that the TGF-DeepFF method obtains the maximum accuracy of 86.29% of 3D scene recognition, which is higher than the accuracy achieved by baseline architectures, including AlexNet, ResNet-50, GoogLeNet and VGG-16 architectures. In addition, results of these methods on 'stage dataset 2' indicate that the dataset is properly organized and suitable for testing the effectiveness of the novel 3D scene recognition methods.

Chapter 5

3D SCENE RECOGNITION MODEL USING HANDCRAFTED FEATURES AND MULTI-LAYER CNN FEATURES FUSION

In this chapter, a novel model is introduced in which the handcrafted features and multi-layer CNN features are fused at different layers. The handcrafted features are manually designed features (Nanni, Ghidoni, & Brahnam, 2017), which are used for classification problem in traditional approaches. The Weibull parameters (J.-M. Geusebroek & Smeulders, 2005), color, HOG (Dalal & Triggs, 2005) features (see description in Subsections 2.1.4.1-3). The multi-layer CNN features are indicating the CNN features that are extracted from different intermediate layers, as described in (Shaopeng Liu et al., 2019; Tang et al., 2017a). The handcrafted features possesses unique representative power of image scene geometry. We believe that it is remarkable to explore the integration of handcrafted features with CNN multi-stages features (HF-MSF) by utilizing the feature fusion and score-level fusion techniques. We call these multi-stages as '*blocks*' to avoid the confusion with the word class 'stages'. As CNN architecture contain different blocks (see Subsections 2.2.2-3). So, the features can be extracted after each block. The GAP operation is used to generate the feature vector from each intermediate CNN block, which is also called as multi-layer features. The standard CNN architecture and its layer detail is explained in Subsection 2.2. The design of the model is given in Section 5.1. The implementation detail and testing is

given in Section 5.2. Experiments and results are illustrated in Section 5.3. The summary is given in Section 5.4.

5.1 Design of HF-HSF Model of 3D Scene Recognition

In this section, we describe the HF-MSF Model of image scene geometry recognition. Novelty of the model compared to previous methods (Chapter 3, Chapter 4) and Algorithm 2.1, it is utilizing the multi-layer and learned features from different blocks of CNN, and combine them individually with handcrafted features at each block. Then these features are separately classified at each block and their outcome are combined to obtain the robust performance. Combination of features at each block-level, which contains discriminative features of scene geometry was not studied before for 3D scene recognition. Algorithm 2.1 indicates the traditional algorithm, which uses only handcrafted features, while Algorithm 3.1 is segmentation-based feature extraction method, which involves segmentation for feature extraction and shows improvement in the stage classification. In Algorithm 4.1, the deep features at final layer are combined with texture gradient features, for a 12000 images dataset. However, these algorithm lose object to scene relationship during feature extraction. Segmentation-based feature extraction involves human interaction, which performs well for small scale dataset (see Table 3.4), but may lose discriminative information when the large scale dataset is used. The Algorithm 4.1, uses deep features which may lose information at intermediate layers when a large number of convolutional and pooling operations have been done.

In this method, the score-level and feature-level fusion strategies are used and visualized in Figures 5.1-2, respectively. Figure 5.1 illustrates score-level fusion, we call it as ‘Model-HSF: Model of Handcrafted and multi-layer features using Score-

level Fusion' and Figure 5.2 represents the feature-level fusion, we call it as Model-HFF: Model of Handcrafted and multi-layer features using Feature-level Fusion. The handcrafted features are extracted from $\eta \times \eta$ grid patches, as it describes in Subsection 3.1.1.2. However, this method simply divide image into patches and features are extracted (same as when template 2.8(f) is used in Subsection 3.1.1.2). The handcrafted features, namely, Weibull parameters, HOG, and color features, given in Subsections 2.1.4.1-3, are used to enhance the discriminative information of image scenes. The detail of the both strategies is given in Subsections 5.2.1-2.

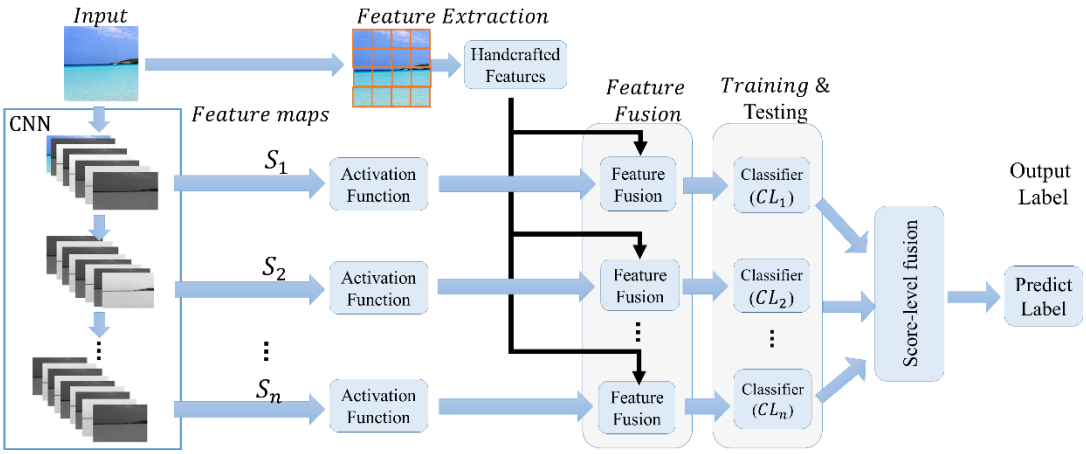


Figure 5.1: 3D scene recognition using Model-HSF (score-level fusion).

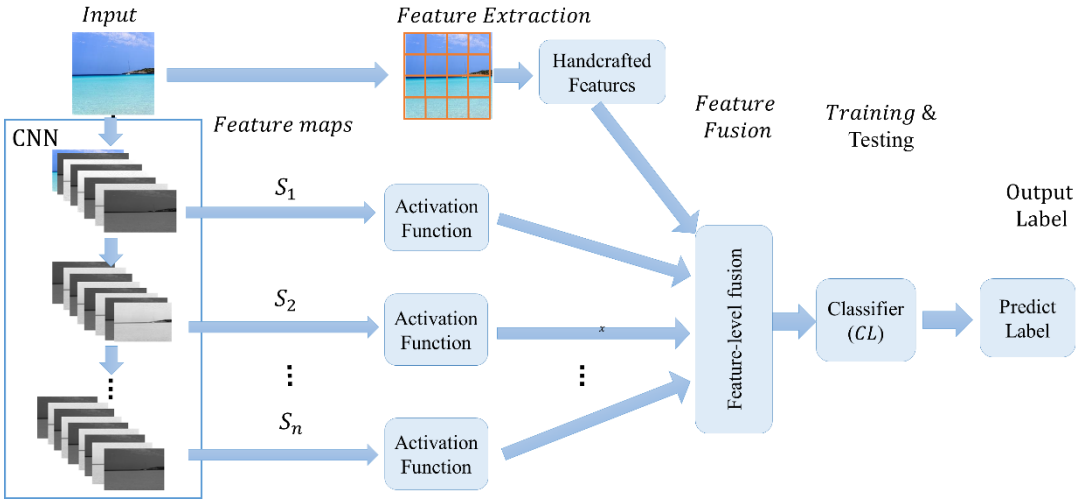


Figure 5.2: 3D scene recognition using Model-HFF (feature-level fusion).

5.1.1 Design of Model-HSF for 3D Scene Recognition

Algorithm 5.1 represents step by step process of Model-HSF, which is also illustrated in Figure 5.1. The Algorithm 5.1 takes N training images, Ɲ total number of images, Y_N training labels, and Y'_M testing labels, C classes, pre-trained CNN model, n number of blocks of CNN, η as input for η×η patches and score-level fusion method (e.g. sum or product rule) as inputs, and generates accuracy (Acc), precision (Pr), recall (Re), and F-score for testing images.

In this model, the deep features, $[f_{1,j}^{d1}, f_{2,j}^{d2}, \dots, f_{n,j}^{dn}]$, at multi-layers are extracted for each input image $I_j, j = 1,2,..Ɲ$ by using function '$deep\ features\ extraction(.)$' (see line 1 of Algorithm 5.1). This function takes CNN model, and number images, Ɲ, and generates multi-layer features as output. n is blocks or multi-stages of CNN ($S_{1,2,\dots,n}$) and di is a dimension of the ith block. The features extraction detail from intermediate blocks are shown in Subsection 2.2.2. Meanwhile, the handcrafted features,$hf_j^{d_{handcrafted}}$, are extracted from the each input image, $I_j, j = 1,2,..Ɲ$, following the model shown in Figure 5.1 (see lines 2-4 of Algorithm 5.1). Before combining the handcrafted features with deep multi-layer features, the both feature vectors are separately normalized by z-score normalization. Then combination of handcrafted features with multi-layer features is given in (see lines 5-9, Algorithm 5.1),

$$F_j^{Di} = [hf_j^{d_{handcrafted}} + f_{i,j}^{di}], i = 1,2,..n, j = 1,2,..Ɲ, \quad (5.1)$$

where $Di = (d_{handcrafted} + d_i)$, Ɲ is total images. '+' is indicating the feature concatenation. Next, the Ɲ number of features vectors ($F_Ɲ^{Di}$) is divided into N training and M testing parts by following the Y_N and Y'_M input labels. For training and testing,

we follow the description given in Subsections 3.1.2. The N-labeled feature vector can be represented by using (3.2), it become, $\boldsymbol{F}_i^{D'} = \left\{(F_{ji}{}^{Di}, y_j)\right\}_{j=1}^{N}$ $i = 1,2..,n.$ Therefore, we have the $\{\boldsymbol{F}_1^{D'}, \boldsymbol{F}_2^{D'}, \ldots \boldsymbol{F}_n^{D'}\}$ set of labeled feature vectors for N training samples. $i = 1,2..,n$ are different CNN blocks. Then, these features vectors are used as inputs to an individual classifier, $CL_i(\boldsymbol{F}_i^{D'})$, i=1,2,..n, to train a model $TrainedCL_i$, i=1, 2,…,n (see lines 11-13 of Algorithm 5.1).

Algorithm 5.1 Method of 3D Scene Recognition using Model-HSF

Input: N training images, CNN model, $\mathcal{N}$ total number of images, C classes, Y_N training labels, Y_M testing labels, η as input for $\eta \times \eta$ $patches$, n is number of blocks of CNN, score-level fusion method

Output: Acc, Pr, Re, F-score

1: $[f_1^{d1}, f_2^{d2}, \ldots, f_n^{dn}] = deep\ features\ extraction$(CNN model, ℕ) // Features f_1^{d1} for ℕ images with length d1, at block, S_1.
2: **for** j=1:ℕ **do** // for each image I_j
3: $hf_j^{d_{handcrafted}} = handcrafted\ features\ extraction(I_j, \eta)$ // feature vector for image I_j, L is length of feature vector, η is input for $\eta \times \eta$ patches e.g. η =4.
4: **end for**
5: **For i=1:n**
6: **For j=1:** ℕ **do**
7: $\boldsymbol{F}_{ji}^{Di} = feature\ fusion\left(normalize\left(hf_j^{d_{handcrafted}}\right), normalize(f_{ji}^{di})\right)$ // combine two different features vectors, with length of $Di = L + di$.
8: **End For**/j
9: **End For** /i
// Training & testing
10: $\boldsymbol{F}_i^{D'} = \left\{(\boldsymbol{F}_{ji}^{Di}, Y_j)\right\}_{j=1}^{N}$ // N is a number of training samples with D' dimension, and ith block, i=1,2..n.
11: **for** i=1:n **do**
12: $TrainedCL_i = CL_i(\boldsymbol{F}_i^{D'})$, //where $\boldsymbol{F}_i^{D'}$ is used to train ith classifier
13: **end for**
// Testing (prediction)
14: **for** j=N+1: ℕ **do** // loop on M test images
15: **for** i=1:n **do** //loop on classifiers
16: $P_{ij}^{C} = Classify(TrainedCL_i, \boldsymbol{F}_{ji}^{D'})$ // P_{ij}^{C} is C-dimensional score vector for an image I_j.
17: **end for** //I

18: **end for**//j
// Classifiers combination
19: **for** j=N+1: $\mathcal{N}$ **do** // M test images
20: l'_j = score-level fusion ($\{P_{ij}^{C}\}_{i=1}^{n}$), fusion of n classifiers scores
21: **end for**//j
//Performance measures
22: [Acc, Pr, Re, F-score]=Calculate_Measures ($\boldsymbol{l'}$, Y_M). // Y_M is a true labels
end Algorithm

The Matlab code of training classifier is shown in Appendix I. In this code a classifier takes a set of labeled features and trains a model. Therefore, the method generates n-trained models for N-labeled data (detail is given Subsection 3.1.2). The feature set of the testing data at each block is evaluated by using a corresponding trained models. Suppose that the feature vectors of the M testing samples are $\{\boldsymbol{F}_{1i}^{D\,'}, \boldsymbol{F}_{2,i}^{D\,'}, \dots, \boldsymbol{F}_{M,i}^{D\,'}\}$ for *i*th block, and are used as inputs to the *TrainedCL*$_i$ model, which predicts a score vector, P_{ji}^{C}, for the *j*th image (see Algorithm 5.1, lines 14-18). P_{ji}^{C} is an *C*-dimensional vector, where C is classes. Next step of Model-HSF is score-level fusion, wherein the outputs of the n classifiers are fused together to obtain a class label (see lines 19-21, Algorithm 5.1). As it describes in Subsection 3.1.3, the predicted score of n classifiers combined using sum or product-rule (described in Subsection 2.1.5.3). Each classifier provides an individual score vector, P_{ji}^{C}, for each *j*th sample, where $i = 1, 2, \dots, n$. Sum or product-rule combines the score vectors of n classifiers and generates a class label l'_j and, j indicates an input sample. Thus, for testing data with a size M, it predicts l'_M, labels. The Matlab code of score combination is given in Appendix J. Finally, the Acc, Pr, Re, and F-score of the input testing data using equations (2.24-2.30) are calculated (see line 22 of Algorithm 5.1). The metric calculation while using Matlab function is given in Appendix K. It takes predicted and ground truth class labels of M images and generates performance metric.

5.1.2 Design of Model-HFF for 3D Scene Recognition

The Model-HFF (see Figure 5.2) fused the multi-layer features and handcrafted features into a single vector. Steps are described in Algorithm 5.2. It takes same input as given for Model-HSF and generates Acc, Pr, Re, and F-score as output. Similar to Model-HSF, the multi-layer features and handcrafted features are obtained from input samples as it describes in Algorithm 5.2, lines 1-4.

Algorithm 5.2 Method of 3D Scene Recognition using Model-HFF

Input: N training images, CNN model, $\mathcal{N}$ total number of images, C classes, Y_N training labels, Y_M testing labels, η as input for $\eta \times \eta$ $patches$, n is number of blocks of CNN

Output: Acc, Pr, Re, F-score

1: $[f_1^{d1}, f_2^{d2}, \dots, f_n^{dn}] = deep\ features\ extraction$(CNN model, $\mathcal{N}$) // Features f_1^{d1} for $\mathcal{N}$ images with length d1, at block, S_1, CNN model is pre-trained model

2: **for** j=1:$\mathcal{N}$ **do** // for each image I_j

3: $hf_j^{d_{handcrafted}} = handcrafted\ features\ extraction(I_j, \eta)$ // feature vector for image I_j, L is length of feature vector, η is input for $\eta \times \eta$ patches e.g. η =4.

4: **end for**

5: **For j=1:** $\mathcal{N}$ **do**

6: $\boldsymbol{F}_j^{Di} = feature\ fusion\left(normalize\left(hf_j^{d_{handcrafted}}\right), \{normalize(f_{ji}^{di})\}^{n}_{i=1}\right)$ // combine two different features vectors, with length of $Di = L + di, + \cdots, dn$.

7: **End For**/j

// Training & testing

8: $\boldsymbol{F}^{D'} = \{(\boldsymbol{F}_j^{Di}, Y_j)\}_{j=1}^{N}$ // N is a number of training samples with D' dimension

9: $TrainedCL = CL\ (\boldsymbol{F}^{D'})$, //where $\boldsymbol{F}^{D'}$ is used to train CL classifier

// Testing (prediction)

10: **for** j=N+1: $\mathcal{N}$ **do** // loop on M test images

11: $l'_j = Classify(TrainedCL, \boldsymbol{F}_j^{D'})$ //l_j predicted label of image, I_j.

12: **end for**//j

//Performance measures

13: [Acc, Pr, Re, F-score]=Calculate_Measures ($\boldsymbol{l'}$, Y_M). // Y_M is a true labels

end Algorithm

Then, these multi-layer features and handcrafted features are first normalized as shown in Algorithm 5.2, lines 6, and then are linearly concatenated to form a $\boldsymbol{F}^D$, that is given by,

$$\boldsymbol{F}_j^D = [f_j^{d_{handcrafted}} + \cup_{i=1}^{n}(f_{i,j}^{dj})],\ j = 1,2,..\text{N}, \tag{5.2}$$

where $f_i^{d_{handcrafted}}$ is a handcrafted features of the image $I_j, j = 1,2,..\text{N}$, $\cup$ is union operator, which indicating the combination of features of n sample, and '+' is a concatenation operator for handcrafted features and multi-layer features. $D = d_{handcrafted} + (d_1 + d_2 + \cdots + d_n)$ and $f_{i,j}^{di}$ is a multi-layer features from the block S_i for jth image. Next, the N number of features vectors ($F_{\text{N}}{}^D$) is divided into N training and M testing parts by following the Y_N and Y'_M input labels (see lines 8 and 9 of Algorithm 5.2). For training, the feature vector, $\boldsymbol{F}_j^{D\prime} = (\boldsymbol{F}_j^D, Y_j), j = 1,2,..N$, with N labels are generated by using (2.15), and used as input to a classifier, CL, to train a model, $Trained\_CL$ (see line 9 of Algorithm 5.2). The feature vectors of the M testing images are used to evaluate the trained model. Suppose that the feature vectors of the M testing images are $\boldsymbol{F}_j^D, j = 1,2,..M$, and are used as inputs to the $Trained\_CL$ model, which predicts the class label, l_j, of each input image I_j (testing process is given in Subsection 2.1.6, (2.19)) (see lines 10-12 of Algorithm 5.2). The predicted label belongs to C, and C is categories of 3D scene geometries. After predicting the class label, I_M, M is testing images, the Acc, Pr, Re, and F-score of the input testing data using Equations (2.24-2.30) are calculated (see line 13 of Algorithm 5.2). The metric calculation using Matlab function is given in Appendix K. It takes predicted and ground truth class labels of M images and generates performance metric.

5.2 Implementation, Testing of HF-MSF Model, and Results on 15-Scene and Stage Dataset 2

In this section, the implementation setting of our HF-MSF model is given. The multi-layer feature extraction setting is given in Subsection 5.2.1. The classifiers setting is given in Subsection 5.2.2. The HF-MSF model is validated on the two benchmark image scenes datasets: 15-scene and 'stage dataset 2', which are illustrated in Subsection 3.2.3, and Subsection 4.2.1, respectively. The MATLAB '2019a' is used to implement the HF-MSF model and executed on a Dell PC with 8 GB RAM, 2.53 GHz i5 processor, and Ubuntu OS.

5.2.1 Multi-Layer CNN Feature Extraction Setting

In order to measure the effectiveness of the HF-MSF model, we show in Section 4.2.3 (results are described in Table 4.1) that the GoogLeNet (Szegedy et al., 2015), and ResNet-50 (He et al., 2016) architectures perform well for image scene geometry recognition. Furthermore, the recent studies (Tang et al., 2017a) and (S. Liu et al., 2019) utilize the multi-layer information of GoogLeNet and ResNet architectures for scene categorization, respectively, and achieve state-of-the-art performance. Therefore, we utilize these architectures as a backbone for multi-layer features extraction. These features are extracted at early, intermediate, and last layers, where these layers output are connected to the GAP function to reduce their dimensionality, at these places, the activation function ($Relu$) (see equation (2.31)) can be used to activate the multi-layer features, as described in Subsection 2.2.2. The Matlab code with ResNet model is shown in Appendix P. Further, implementation detail for each CNN model is given below.

The GoogLeNet and ResNet models (described in Subsections 2.2.2-3) consists of several consecutive layers and due to the redundancy among these layers' features, it reduces the performance of scene recognition for medium and small scale datasets. To implement the HF-MSF model using pre-trained GoogLeNet architecture (pre-trained on Places365 (B. Zhou et al., 2018)), we follow (Tang et al., 2017a), where model is divided into three blocks (S_1, S_2, S_3), to extract three different feature vectors for each input image (Matlab code is shown in Appendix P, which is needed to set GoogleNet blocks given in comments). The features are extracted from GAP layer, before auxiliary classifiers, see Figure 2.18. In this way, we are getting branches where each block generating feature vector with 1000 dimensions. At each block (S_i), $i = 1,2..n$, the deep features (f_i^{1000}) and handcrafted features ($f_{handcrafted}^{320}$), are integrated by following proposed Model-HSF, which form a 1320-dimension fusion vector ($Di = (d_{handcrafted}^{320}, d_i^{1000})$), i indicates an intermediate ith block, and d is the feature dimension. Matlab code is shown in Appendix N. The handcrafted feature set is calculated by partitioning each image into 4×4 patches and then parameters of Weibull distribution (four features, see in Subsection 2.1.4.1), Color (seven features, including color corrected coefficient of RGB (3 features), mean of HSV (3 features), and variance of saturation component (1 feature)), and HOG (nine features, described in Subsection 2.1.4.3) features are extracted for each patch that yields 320-dimensional feature vector for a single image. As the experiments in Chapter 3 (see Table 2.3) show that the template (f), from Figure 2.8 has better results compared to each individual template, therefore, we adopt it with 4×4 grid patches for handcrafted feature extraction. For selecting the particular features of image scene geometry, we follow (Nedovic et al., 2010) and (Lou et al., 2015) methods, which particularly pay attention to the use of rich image scene geometry features. These methods utilize parameters of

Weibull distribution, color features (correction coefficient of RGB using gray word algorithm, mean of HSV, an average of the variance of saturation component), and HOG features. Besides this, we also use LBP-E features to represent the scene geometry structure. These features are applied on 'stage dataset 2' using linear SVM. The results of different features combinations are shown in Appendix S, in Table S4 and, Figure S1. We found that combining the above features (without LBP-E features) achieves 58.71% recognition accuracy for 'stage dataset 2' (12000 images), (see Appendix S, in Table S4), while by adding the LBP-E features, the accuracy reaches 56.46% (2.25% decreases) and the size of the feature vector reaches 400 when the 4x4 grid patches are used. Therefore, we ignore using of the LBP features for medium-scale dataset. On the other hand, the parameters of Weibull distribution capture the texture information (Nedovic et al., 2010), p.1677 and improve the scene geometry recognition accuracy when combined with deep features, as we observe in Chapter 4 (see Table 4.1). Thus, the HOG, color, and parameters of Weibull distribution are extracted as handcrafted features from each image patch to represent the image scene geometry. These features are normalized before integrating into a one single set. For Model-HFF, the deep features of each block (S_i) and handcrafted features are linearly integrated to allow 3320-dimensions fusion vector $D = [d_{handcrafted}^{320}, d_1^{1000}, d_2^{1000}, d_3^{1000}]$. It performs feature-level fusion with handcrafted features. The combination of these features are done by same function, shown in appendix N.

Next, the HF-MSF model has also utilized the pre-trained ResNet-50 architecture. The pre-trained ResNet-50 architecture contains 16 residual blocks. The basic structure of the residual block is shown in Figure 2.19. Each block contains several internal layers,

and at the end of the block, the output of current block is combined with the previous block by the residual connection. After this point, we can extract the features. "The feature maps created from the adjacent residual block have certain similarities. It is not wise to extract features from every block, which can increase both the network complexity and computation time" (S. Liu et al., 2019), p.194. Therefore, we extract each feature vector after two consecutive blocks. Five different residual blocks, namely blocks 10 ('activation_29_relu'), 12 ('activation_35_relu'), 14 ('activation_41_relu'), 16 ('activation_45_relu'), and at 'fc1000' layers ($S_1, S_2, \dots, S_5$) are selected for multi-layer feature extraction. Matlab code is given in Appendix P. The previous blocks are ignored because beginning layer contain more detailed information, but suffer the problem of background clutter and semantic ambiguity (Yu, Yang, Yao, Sun, & Xu, 2017). The model provides 5 different vectors having 256, 256, 512, 512, 1000 dimensions and each block outcome is separately concatenated with handcrafted features vector (320 features). And then it feeds to the five ensemble of classifiers. Matlab code is shown in Appendix Q. By using Model-HFF, the all feature vectors including handcrafted features which are linearly integrated to form a 2,856-dimensions fused vector $\left(F^D, \mathrm{D} = (d_{handcrafted}^{320}, d_1^{256}, d_2^{256}, d_3^{512}, d_4^{512}, d_5^{1000})\right)$. The combination of these features are done same function, shown in appendix N. We also apply the pre-trained AlexNet (Alex et al., 2012) architecture. We extracted the multi-layer features, 96, 256, 256, and 1000, at 'maxpool1' (S_1), 'maxpool2' (S_2), 'maxpool5'(S_3), and 'fc8' (S_4) layers, respectively, by applying the ReLU activation function with GAP operation and then these features are individually fed to the classifiers, as it is described above for GoogLeNet and ResNet architectures.

We adopt the pre-trained GoogleNet model on Places365 (B. Zhou et al., 2018) Dataset, which contains about 10 million scene images in total and more than 400 image categories. The ResNet is pre-trained on ImageNet dataset (J. Deng et al., 2009) which contains more than 1.2 million images with 1000 classes. After that, both models' parameters are fine tune on the two benchmark scene datasets: 15-scene and 'stage dataset 2', which are illustrated in Subsection 3.2.3, and Subsection 4.2.1, respectively.

5.2.2 Classifiers Setting

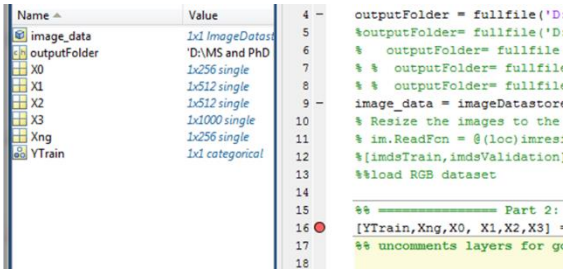
In HF-MSF model, we applied two well-known classifiers, namely ELM and Linear SVM (description is given in Subsections 2.1.5.1-2). SVM is a solid choice for scene geometry classification (Nedovic et al., 2010). The ELM also shows high performance for scene classification in several studies (Cheng et al., 2015; Lei et al., 2021; Özyurt, 2020; Özyurt et al., 2020; J. Tang et al., 2015; Wang et al., 2020; Wang et al., 2021; Weng et al., 2017), as it is described at the beginning of Chapter 4. Inspired by the high performance of (Wang et al., 2020) of scene recognition, we also utilize the ELM as a classifier in this study. We set following parameters: CL_i is set to $ELM, i = 1,2 \dots n$, n is blocks, neurons $L = 12{,}000$, the activation function is set as 'sigmoid function', and C= 10^{-7}. Matlab code of ELM classifier is given in Appendix I. It predicts the hard-label for each input image. Therefore, majority voting is used to obtain the final label when the Model-HSF is used. For Model-HFF, the fused feature vector is used to ELM classifier and class label is directly predicted.

Secondly, the linear SVM is considered due its effectiveness and efficiency for the CNN architectures (Kim et al., 2013; Tang et al., 2017a). We employed it with default parameters. The Matlab code is given in Appendix I. It generates aposteriori

probabilities or scores distributed over the number of classes and class label (class type) for each input image. In Model-HSF, these scores for $CL_i, i = 1,2,..n$ ($n = 3$ for GoogLeNet, and $n = 5$ for ResNet) are combined for test images by using majority, max, sum, and product rules. Matlab code is shown in Appendix R. In Model-HFF, it is directly used to predict the output label for test images.

5.2.3 Testing of the HF-MSF Model

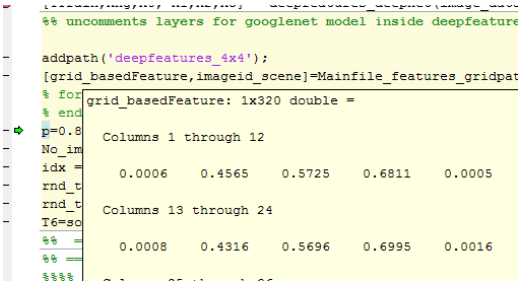
In this section, we test HF-MSF Method implementation in different steps, which are given below. The left side is indicating expected value from the HF-MSF method. Right side is indicating the output of the HF-MSF method for a single input image. The horizontal lines are differentiating multiple steps.

Manually calculation and settings	**Output of HF-HSF Method**
Step1: Input RGB image Input RGB image: Size 256x256x3 pixels.	 Image reference (B. Zhou et al., 2018).
Step2: Extract multi-layer CNN features We extract the multi-layer features from different blocks of the CNN models. Here, we show that features from 5 different blocks of ResNet-50 generates 256, 256, 512, 512, 1000 features for each input image.	**Output:** Multi-layer features for single image is shown in screenshot below. The 256, 256, 512, 512, 1000 features from 5 different blocks are shown below for each input single input image. 

Step 3: Handcrafted features extraction

We extract handcrafted features which are explained in Subsection 2.1.4.1-3. And these features are also used in Chapter 3. So, same code is used here with $n \times n$ grid patches. However, it generates 320 features for each input image are extracted.

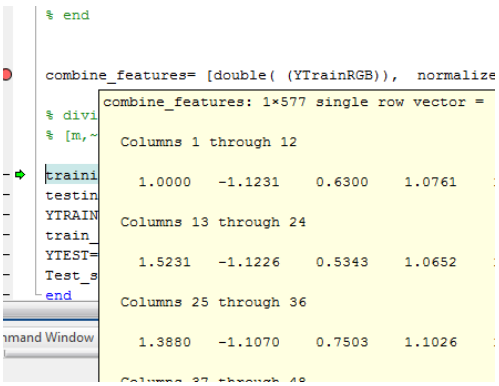
Output: The code generates 320 handcrafted features for each input image as shown in screenshot.



Step 4: Features normalization and combination

In this step, multi-layer and handcrafted features are normalized first and then combined into single feature vector. The Model-MSF generates 5 different feature vectors. Here, we show a single feature vector. The other features vector can be generated similarly. Suppose the length of the feature vector is 576= 256+320 at first block. By adding the class label, it reaches to 577. Similarly, Model-HFF features are combined.

Output: The screenshot shows that features are first normalized while using Matlab function $normalize(.)$ and then the features are combined into a single vector, yielding 576 features. It contains one more element which is indicating the class label of the image. It means that our implementation is accurate.



Step 5: Training and testing step

In this step, the fused feature vectors are ready to use for training the machine learning algorithm. We use 5 parallel classifiers and their output are combined at score-level. This process is given in the next section because here we verified that our implementation is correct for input testing image.

5.3 Experiments and Results

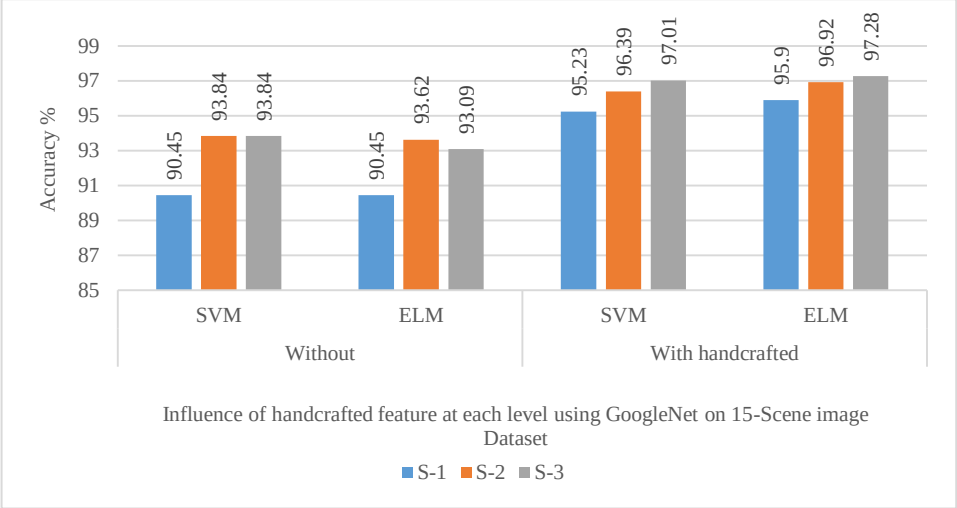
The image scene geometry recognition performance of the Model-HSF model, and comparison with the baseline methods are discussed in this section. The performance metrics is used which are described in Subsection 2.1.7, equations (2.24-30). The detail results, including Acc, Pr, Re, and F-score (F-s.) are given in Tables and accuracy is visualized in bar graphs (Figures) as well. The graphs are generated by excel tool. The best results are shown in bold font. The performance of HF-MSF model on 15-Scene and 'stage dataset 2' are given in Subsections 5.3.1-2, respectively. The 15-scene and 'stage dataset 2' are described in Subsections 3.2.4.1 and 4.2.1, respectively. For 15-scene dataset, the 100 images per category are used for training and rest for testing (as following standard setting, see Table 2.1). In 'stage dataset 2', the 80% images are used for training and 20% for testing as given in Subsection 4.2.1.

5.3.1 Performance of HF-MSF Model on 15-Scene Dataset

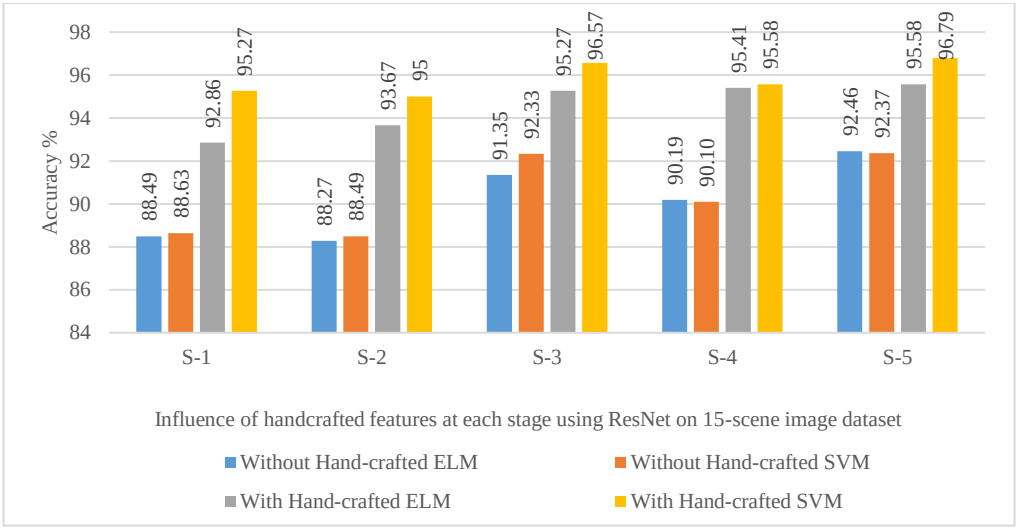
In the 15-scene dataset, there is a limited number of images in each category. We observe that the intermediate layer of both GoogLeNet and ResNet-50 architectures improve the performance by adding handcrafted features, as it given in Table 5.1. The score-level fusion (Model-HSF) uses different blocks of both ResNet and GoogLeNet architectures. In Table 5.1 and Figure 5.3, it is shown that adding handcrafted features increase accuracy of scene recognition at each block of CNN ($S_1, \dots, S_n$). Figure 5.3 (a) illustrates the GoogLeNet model with its three different blocks (S_1, S_2, S_3) and

Figure 5.3 (b) illustrates the ResNet-50 based model with its five different blocks, where SVM and ELM classifiers are applied at each block ($S_1, S_2, \ldots S_5$). 'with' and 'without handcrafted' indicates that handcrafted features are integrated at each steps or without integration and the results of each block for both SVM and ELM classifiers are given. Results in Figure 5.3 (a & b) and Table 5.1 describe that beginning blocks, S_1 and S_2, show lower results than the blocks S_3, S_4, and S_5 of ResNet-50 architecture and similar behavior is observed of GoogLeNet architecture (see Figure 5.3 (a)). Therefore, previous blocks are ignored for score-level fusion. Table 5.1 also describes the scene recognition results of four different layers, S_1, S_2, S_3, S_4, of AlexNet architecture. Results (see Table 5.1, AlexNet Arch.) indicate that the scene recognition accuracy increases from S_1 toward S_4 for both SVM and ELM classifiers and it reaches 69.23% to 86.73% for S_1 and S_4, respectively. Adding the handcrafted features to AlexNet architecture achieves maximum accuracy of 94.09% at the 'maxpool5' (S_3) layer when the SVM classifier is used. However, it is lower than the GoogLeNet and ResNet architectures by 2.92% and 2.70%, respectively (see Table 5.1). The results of the combination of different layers feature and score-level fusion are given in Appendix X (in Table X2 & X3). The Model-HSF performance is given in Table 5.2. In score-level fusion, the maximum accuracy is obtained by the sum and product rules which approaches to 97.55 % by using GoogLeNet architecture and the F-score is reached to 97.42% by using sum-rule. By adopting the ResNet-50 architecture (see Table 5.2, rows 6-10), the sum rule shows 97.86% and product-rule reaches to 97.77% using linear SVM. The maximum F-score is 97.79% when the sum-rule is used (8^{th} rows). Graphical representation of results is shown in Figure 5.4, where SVM and ELM classifiers are used and effects of 'with' and 'without' handcrafted features are visualized. The accuracy is calculated by majority voting, max, sum, and product rules

for each classifier. Figure 5.4(a) illustrates GoogLeNet based Model-HSF results and Figure 5.4(b) shows the results of the Model-HSF model using the ResNet-50 architecture. Similarly, the AlexNet architecture using sum-rule achieves 93.98% recognition accuracy, which is 3.88% and 3.57% lower than ResNet-50 and GoogLeNet architecture, respectively (see Table 5.2).



(a) Using GoogLeNet architecture. S-1, S-2,S-3 are indicating the S_1, S_2,S_3 blocks, respectively.



(b) Using ResNet architecture. S-1, S-2,..,S-5 are indicating the S_1, S_2,…,S_5 blocks, respectively.

Figure 5.3: 15-scene image dataset: influence of handcrafted features at each intermediate layers. Figure (a) and (b) are indicating GoogLeNet and ResNet architectures, respectively. Raw data is shown in Appendix X.

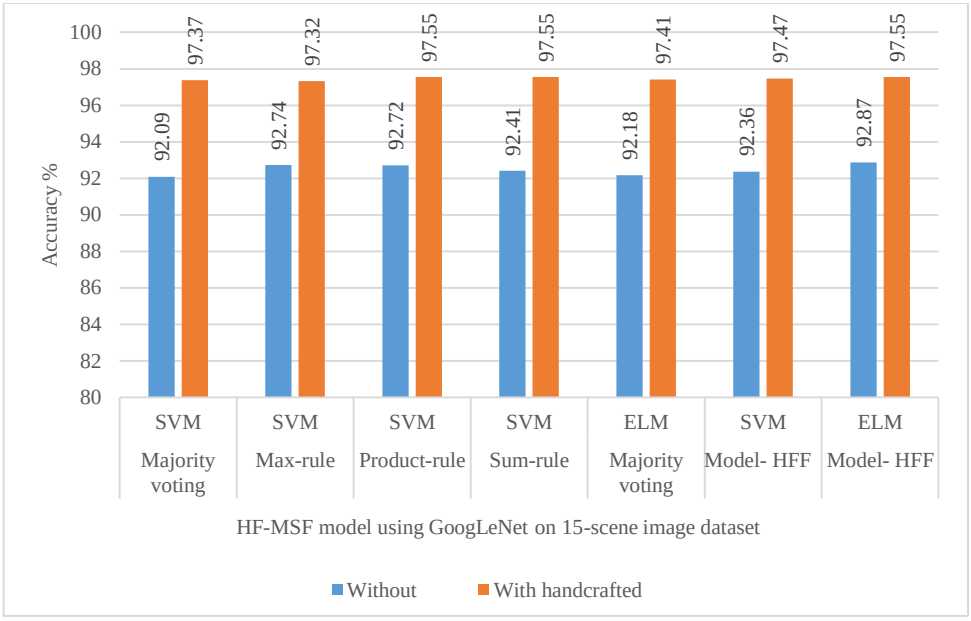
Table 5.1: 15-Scene images dataset: experiments on 'with' and 'without handcrafted' features fusion by using SVM and ELM classifiers.

CNN Architecture	Blocks	Without handcrafted (Accuracy %)		With handcrafted (Accuracy %)	
		ELM	SVM	ELM	SVM
ResNet Architecture	S_1	88.49	88.63	92.86	95.27
	S_2	88.27	88.49	93.67	95.00
	S_3	91.35	92.33	95.27	96.57
	S_4	90.19	90.1	95.41	95.58
	S_5	92.46	92.37	95.58	96.79
GoogLeNet Arch.	S_1	90.45	90.45	95.9	95.23
	S_2	93.62	93.84	96.92	96.39
	S_3	93.09	93.84	97.28	97.01
AlexNet Arch.	S_1	69.23	77.70	81.27	84.39
	S_2	82.94	83.84	91.19	91.42
	S_3	85.95	87.74	92.31	94.09
	S_4	86.73	85.73	93.53	93.87

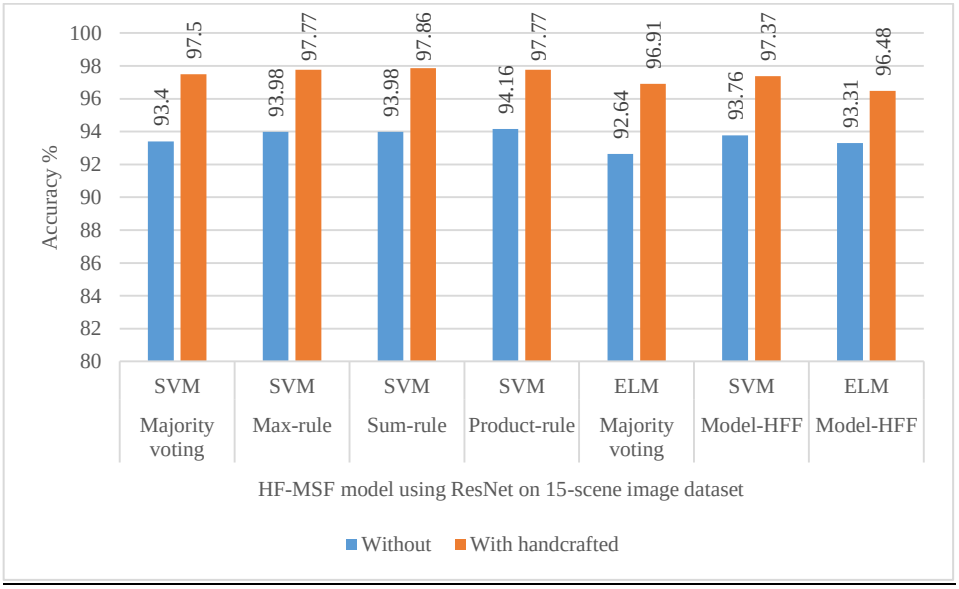
Raw Data is given in Appendix X.

Next, the feature-fusion strategy is applied (Model-HFF) on 15-scene image dataset. The features of different blocks and handcrafted features are fused into a one feature vector. Then, the SVM and ELM are applied and recognition results are given in Table 5.3. The maximum performance is obtained by ELM, which is reached to 97.41% recognition accuracy (2^{nd} row). While by employing the ResNet with SVM, the proposed Model-HFF achieves 96.48% recognition accuracy (3^{rd} row). On the other hand, AlexNet architecture reaches to 94.95% recognition accuracy when the SVM classifier is used. In conclusion, the GoogLeNet achieves the best performance (97.41%) when the Model-HFF is used. Generally, both proposed approaches (Model-HSF and Model-HFF) perform well by which it is clear evidence that handcrafted

features influence is positively improved the recognition of image scene classification. In conclusion, the Model-HSF shows the best performance in scene recognition compared to the Mode-HFF because of using score-level fusion of different classifiers.



(a) Using GoogLeNet architecture



(b) Using ResNet architecture

Figure 5.4: 15-Scene image dataset: performance of the model-HSF. Figure (a) and (b) are indicating GoogLeNet and ResNet architectures, respectively. Raw data is given in Appendix X.

Table 5.2: Experimental results of the Model-HSF on 15-scene images dataset.

CNN Arch.	Model-HSF	Without				With handcrafted features				
		Classifier	Acc.%	Pr. %	Re. %	F-s%	Acc.%	Pr. %	Re. %	F-s%
GoogLeNet Architecture	Majority voting	SVM	93.86	93.90	93.84	93.84	97.37	97.33	97.33	97.29
	Max-rule	SVM	93.94	93.75	93.84	93.78	97.32	97.27	97.37	97.26
	Product-rule	SVM	94.12	94.22	94.15	93.96	97.55	97.32	97.52	97.39
	Sum-rule	SVM	94.01	94.33	94.33	94.32	97.55	97.45	97.46	97.42
	Majority voting	ELM	93.74	93.70	93.87	93.72	97.41	97.38	97.51	97.42
ResNet Architecture	Majority voting	SVM	93.40	93.74	92.93	93.23	97.50	97.29	97.40	97.31
	Max-rule	SVM	93.98	93.94	93.61	93.73	97.77	97.68	97.65	97.63
	Sum-rule	SVM	93.98	94.33	93.44	93.74	**97.86**	**97.80**	**97.83**	**97.79**
	Product-rule	SVM	94.16	93.39	93.65	93.92	97.77	97.69	97.72	97.68
	Majority voting	ELM	92.64	92.43	92.39	92.36	96.91	96.83	96.94	96.83
AlexNet Arch.	Majority vote	SVM	87.51	86.53	86.06	86.09	92.31	91.63	91.49	91.43
	Max-rule	SVM	88.74	88.01	87.95	87.93	93.31	92.86	92.69	92.65
	Sum rule	SVM	90.64	90.00	89.61	89.73	93.98	93.63	93.35	93.34
	Product rule	SVM	89.68	89.07	89.05	89.01	93.53	92.98	92.91	92.84
	Majority voting	ELM	84.95	84.78	83.69	83.57	91.19	91.12	90.16	90.22

Raw data is given in Appendix X.

Table 5.3: Feature-level fusion: result of Model-HFF for 15-scene images dataset.

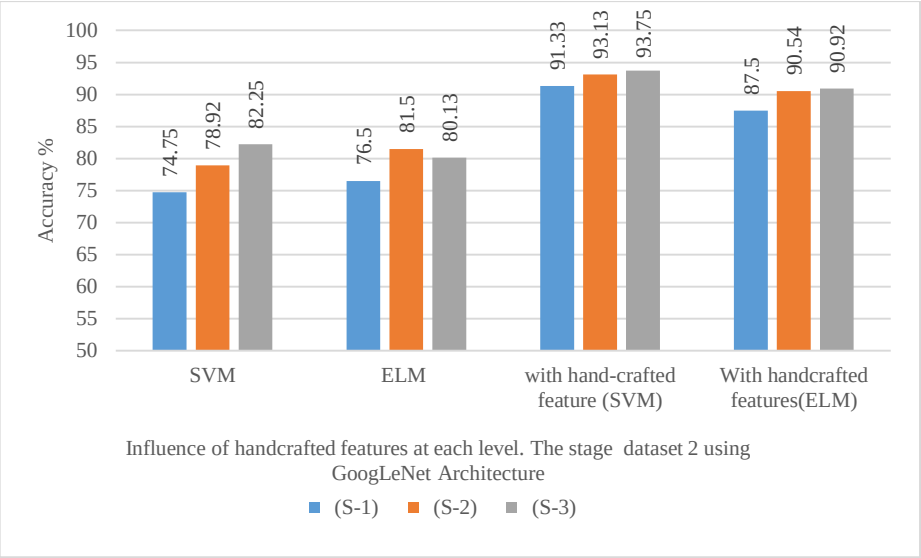
CNN Architecture	Classifier	Acc.%	Pr. %	Re. %	F-score%
GoogLeNet Architecture	SVM	97.32	97.45	97.34	97.25
	ELM	**97.41**	**97.33**	**97.45**	**97.45**
ResNet Architecture	SVM	97.37	97.24	97.41	97.30
	ELM	96.48	96.39	96.39	96.32
AlexNet Arch.	SVM	95.32	95.32	94.91	94.95
	ELM	93.98	94.29	93.50	93.61

Raw Data is given in Appendix X.

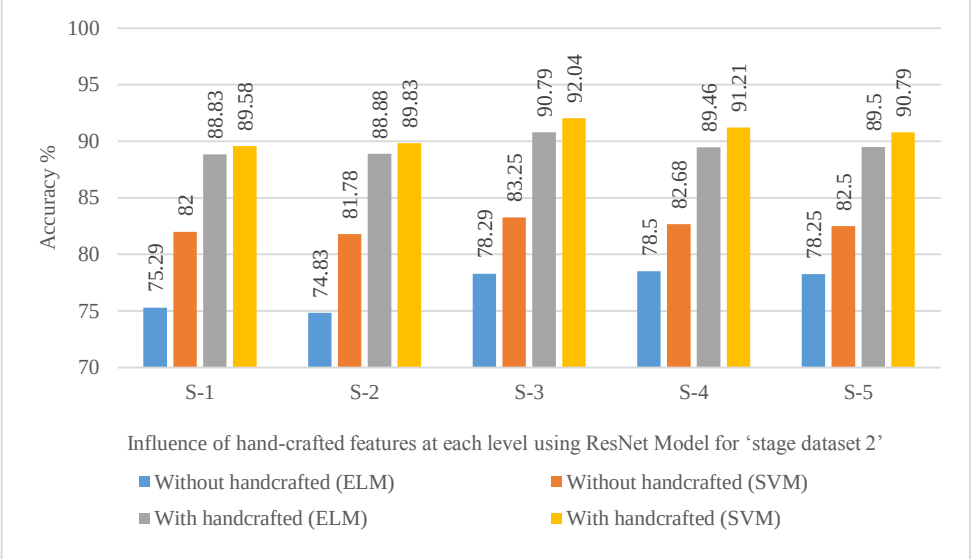
5.3.2 Performance of HF-MSF Model on Stage Dataset 2

In ‘stage dataset 2’, since the scenes are relatively classified according to image scene geometry structure and dataset size is larger than 15-scene dataset. Therefore, it is useful to employ the deeper architecture. The handcrafted features provide rich discriminative information of image scene geometry and fusion of this information at each deep CNN block, it improves the recognition rate effectively when we apply the Model-HSF. Figure 5.5, and Table 5.4 differentiate ‘with’ and ‘without’ handcrafted feature influence at each block of GoogLeNet and ResNet Architectures. In Figure 5.5(a), which illustrates the effectiveness of handcrafted features at different blocks of GoogLeNet model, shows that block (S_1) achieves 74.75% to 91.33% recognition accuracy when the handcrafted features are added and SVM classifier is applied. Similarly, block (S_1) reaches 76.50% to 87.50% recognition accuracy when the handcrafted features are added and ELM classifier is applied. The same behavior can be observed in Figure 5.5(b) and Table 5.4 (rows 1-5) by using the ResNet-50 Model. We also used the AlexNet architecture to ’stage dataset 2’. The results of each layer ‘with’ and ‘without handcrafted’ features are shown in Table 5.4. We observe that the recognition accuracy is increased toward the higher layers, such as it reaches 57.33% at S_1 and 70.49% at S_4 (see Table 5.4, with SVM). Adding the handcrafted features to AlexNet architecture, the accuracy reaches 77.17% for S_1 and it reaches 88.17% for S_4 when the linear SVM is used. It means that adding handcrafted features improves the recognition accuracy at each intermediate layer. However, the ResNet and GoogLeNet architectures achieve 3.87% and 5.58% higher accuracy, respectively, compared to AlexNet architecture when the handcrafted features are integrated (see Table 5.4). The performance of Model-HSF improves 81.83% to 95.17% (see Table 5.5 (row 4)) by using product-rule on test images when the GoogLeNet architecture is

applied. Meanwhile, its maximum F-score is achieved 95.06%. Detailed of the model-HSF is shown in Figure 5.6, where majority voting, sum, max, and product rules are calculated when an SVM classifier is applied and ELM performance is calculated by majority voting only. Figure 5.6(a) illustrates the effectiveness of handcrafted features for score-level fusion using GoogLeNet model. Product-rule achieves the highest performance and reaches to 95.17% when the handcrafted features are included. Figure 5.6(b) shows the effectiveness of Model-MSF using ResNet model. The handcrafted features influence improves 83.25% to 93.96% recognition accuracy by employing the product-rule when the SVM classifier is selected. It improves 10.71% recognition accuracy compared to the baseline ResNet model. By comparing among score-level fusion techniques (see Table 5.5), it is shown that the sum and product-rule always show better performance than majority voting and max rule, and provides highest score for both architectures (Table 5.5, rows:3,4 and 8, 9). The discriminative information for each class is exhibited in confusion matrix, Figure 5.7 (a, b), which clearly shows that simple combination of multi-layers features has confusion with different classes, such as *sidewallRL* confused with *groundDiagBkgRL* class up to 18.42%, while Model-HSF reduces this confusion to 3.62% (see Figure 5.7(b)). On the other hand, the AlexNet architecture reaches 74.38% recognition accuracy when the sum-rule is used for classifier-level aggregation without integrating the handcrafted features. It is 3.89% higher than the accuracy of the 'fc8' layer of AlexNet architecture. The results of intermediate layers feature combination and score-level combination are given in Appendix Y (Table Y3). However, the performance of AlexNet architecture is 6.58% and 7.79% lower than ResNet and GoogLeNet architectures, respectively, because these models take advantage of residual and inception blocks to avoid the overfitting problem.



(a) Using GoogLeNet architecture. S-1, S-2,S-3 are indicating the S_1, S_2,S_3 blocks, respectively.



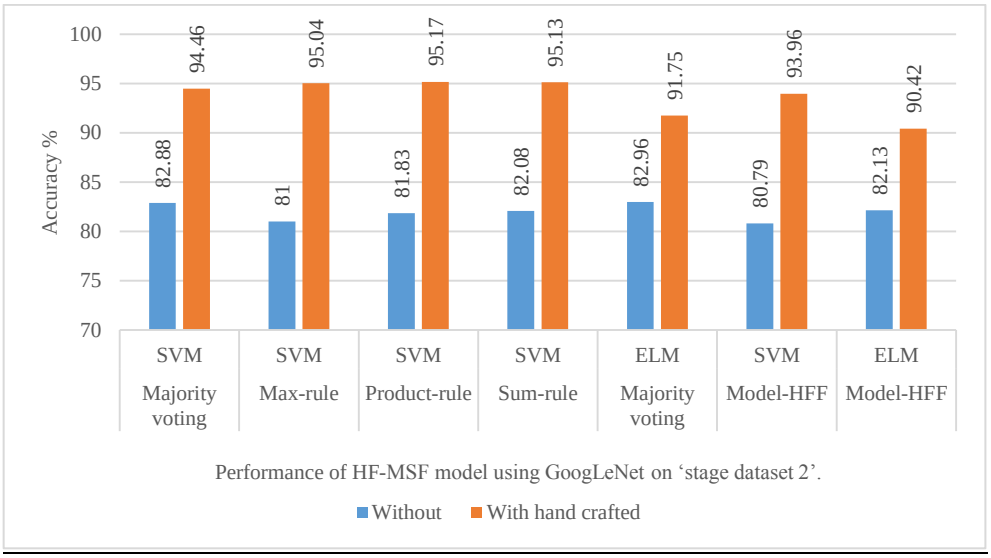
(b) Using ResNet architecture. S-1, S-2,..,S-5 are indicating the S_1, S_2,…,S_5 blocks, respectively.

Figure 5.5: 'Stage dataset 2': influence of handcrafted features at each intermediate block of GoogleNet (see in (a)) and ResNet architectures (see in (b)). Raw data is given in Appendix Y.

Table 5.4: Stage image dataset 2. Experiments of 'with' and 'without handcrafted' features fusion by using SVM and ELM classifiers.

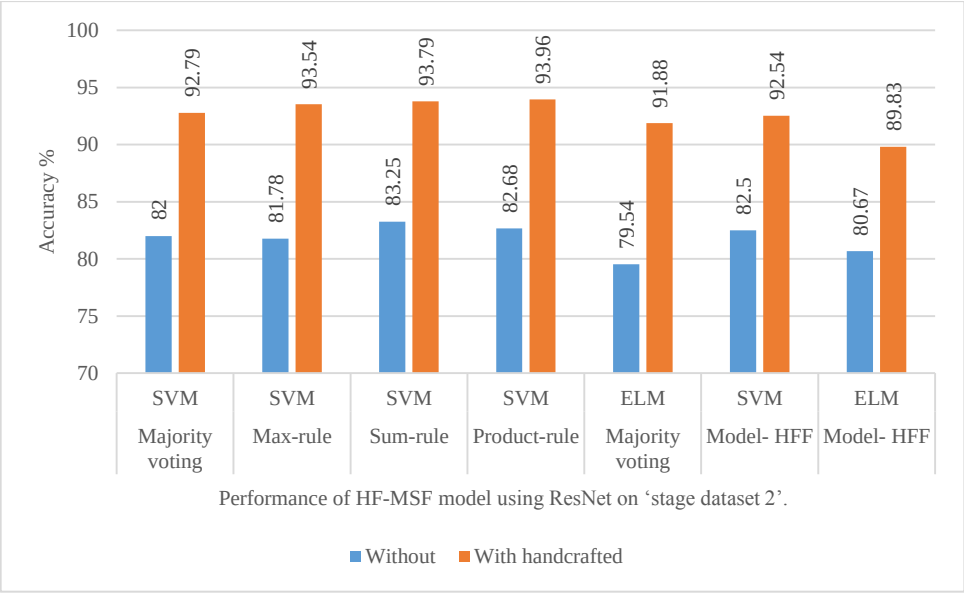
CNN Architecture	CNN Blocks	Without handcrafted (Acc. %)		With handcrafted (Acc. %)	
		ELM	SVM	ELM	SVM
ResNet Architecture	S_1	75.29	82.00	88.83	89.58
	S_2	74.83	81.78	88.88	89.83
	S_3	78.29	83.25	90.79	92.04
	S_4	78.50	82.68	89.46	91.21
	S_5	78.25	82.50	89.50	90.79
GoogLeNet Architecture	S_1	76.50	74.75	87.50	91.33
	S_2	81.50	78.92	90.54	93.13
	S_3	80.13	82.25	90.92	93.75
AlexNet Arch.	S_1	52.25	57.33	77.58	77.17
	S_2	64.42	64.38	81.58	84.21
	S_3	70.88	67.33	85.63	85.46
	S_4	72.46	70.49	85.92	88.17

Raw data is given in Appendix Y.



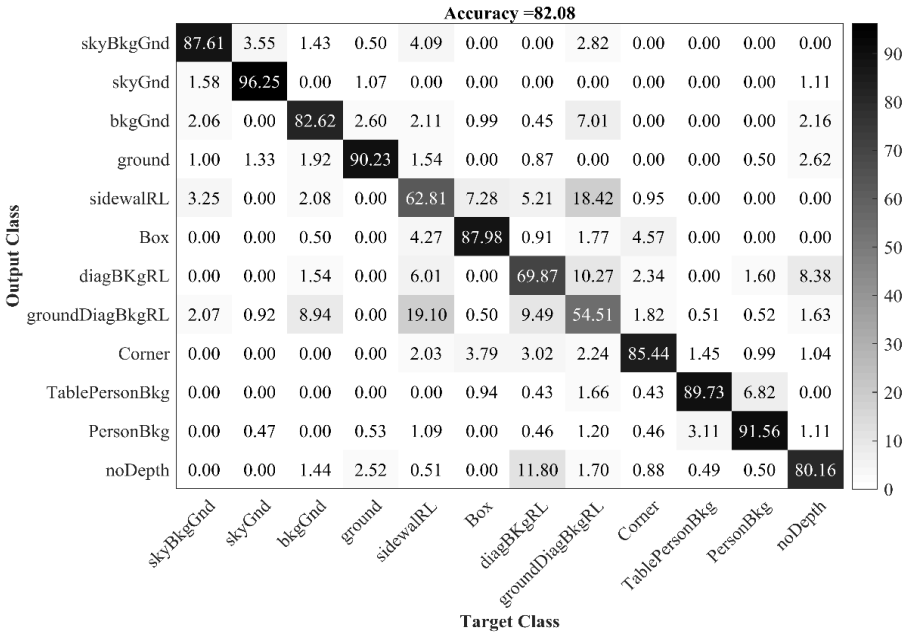
(a) Using GoogLeNet architecture

Figure 5.6: 'Stage dataset 2': performance of Model-MSF. Figure (a) and (b) are indicating GoogLeNet and ResNet architectures, respectively. Raw Data is given in Appendix Y.

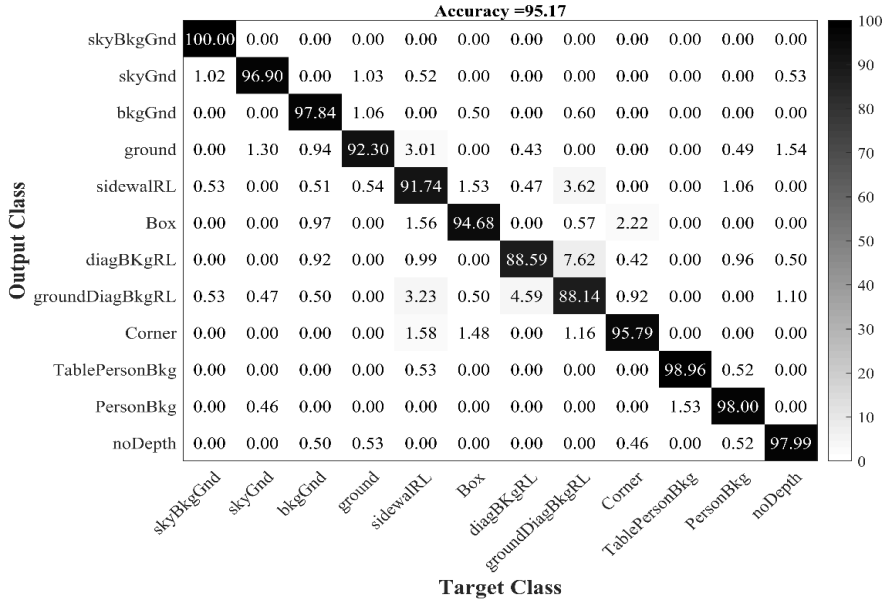


(b) Using ResNet architecture

In contrast, Table 5.6 shows that the Model-HFF achieves 93.92% recognition accuracy when the handcrafted features are integrated with multi-layer learned features of GoogLeNet model and selecting linear SVM as a classifier. It shows that handcrafted features influence greatly effects of feature-level fusion as well and it improves 11.42% recognition accuracy from 82.50% recognition accuracy which is achieved by using ResNet architecture with a linear SVM classifier. Meanwhile, the AlexNet architecture reaches 91.40% recognition accuracy of image scene geometry when the handcrafted features and multi-layer features are combined (see Table 5.6). The detailed results of the multi-layer combination are given in Y (Table Y4). However, the maximum accuracy is achieved by GoogLeNet architecture, which is reached 93.96% when it is integrated with handcrafted features. On the other hand, ResNet50 is reached 92.54% when the handcrafted features are integrated and SVM classifier is used. Both GoogLeNet and ResNet architectures achieve higher accuracy compared to AlexNet architecture because they are utilizing the concept of inception modules and residual blocks to avoid the overfitting problem, respectively.



(a) Confusion matrix of Model-HSF ‘without handcrafted’ features using sum-rule.



(b) Confusion matrix of HSF Model with handcrafted feature using product rule.

Figure 5.7: Confusion matrices of ‘stage dataset 2’ using GoogleNet architecture with multi-layer and Model-HSF given in Figure (a) & (b). Raw data is given in Appendix Y.

Table 5.5: 'Stage dataset 2': performance of Model-HSF using ResNet and GoogLeNet architecture.

CNN Architecture	Model-HSF	Classifier	Without				With handcrafted features			
			Acc.%	Pr. %	Re. %	F-s%	Acc.%	Pr. %	Re. %	F-s%
GoogLeNet Architecture	Majority voting	SVM	82.88	82.55	82.74	82.58	94.46	93.2	93.09	93.18
	Max-rule	SVM	81.00	80.78	81.05	80.89	95.04	94.99	94.87	94.91
	Sum-rule	SVM	82.08	81.61	81.71	81.62	95.13	95.09	94.99	95.02
	Product-rule	**SVM**	81.83	81.61	81.72	81.65	**95.17**	**95.13**	**95.03**	**95.06**
	Majority voting	ELM	82.96	82.13	82.82	82.27	91.75	91.69	91.83	91.65
ResNet Architecture	Majority voting	SVM	82.00	81.22	81.55	81.35	92.79	92.86	92.73	92.78
	Max-rule	SVM	81.78	81.61	81.5	81.53	93.54	93.47	93.36	93.40
	Sum-rule	SVM	83.25	82.69	82.8	82.70	93.79	93.77	93.64	93.69
	Product-rule	SVM	82.68	82.12	82.18	82.10	93.96	93.89	93.82	93.84
	Majority voting	ELM	79.54	78.91	79.65	78.87	91.88	91.87	92.01	91.83
AlexNet Architecture	Majority voting	SVM	68.92	68.28	69.10	68.40	85.75	85.89	85.88	85.86
	Max-rule	SVM	71.54	70.78	71.70	71.04	87.13	87.27	87.28	87.26
	Sum-rule	SVM	74.38	74.05	74.48	74.19	86.63	86.92	86.77	86.82
	Product-rule	SVM	72.88	72.34	73.01	72.45	87.38	87.52	87.51	87.50
	Majority voting	ELM	64.88	64.22	65.18	63.18	83.71	83.81	83.95	83.53

Raw data is given in Appendix Y.

Table 5.6: Model HFF results using ‘stage dataset 2’.

CNN Architecture	Classifier	Without				With handcrafted features			
		Acc.%	Pr. %	Re. %	F-s%	Acc.%	Pr. %	Re. %	F-s%
GoogLeNet Architecture	SVM	80.79	80.64	80.4	80.44	**93.96**	**93.92**	**92.87**	**93.88**
	ELM	82.13	81.32	82.00	81.54	90.42	90.24	90.49	90.25
ResNet Architecture	SVM	82.50	82.28	82.05	82.10	92.54	92.33	92.32	92.38
	ELM	80.67	80.16	80.74	80.16	89.83	89.81	89.92	89.73
AlexNet Arch.	SVM	74.04	73.90	74.24	94.03	91.33	91.48	91.43	91.45
	ELM	75.29	74.50	75.57	74.25	87.38	87.29	87.61	87.30

Raw data is given in Appendix Y.

5.3.3 Comparison with State-of-the-Art Methods

In this section, the HF-MSF model is compared with existing methods. The existing methods, for 15-scene dataset are given in Table 2.1. The G-MS2F (Tang et al., 2017a), FTOTLM (Shaopeng Liu et al., 2019), DFF-ADML (Wang et al., 2020), RLML-LSR Chen et al. (Wang et al., 2021) methods performance are also compared with HF-MSF model, and accuracy of these methods are given in Table 5.7. These methods achieve an accuracy of 92.90%, 94.01%, 96.39%, and 93.50%, respectively, on 15-scene dataset. On the other hand, Liu, et al.(B. Liu, Liu, & Lu, 2015), Lin et al. (Lin et al., 2017), Zafar et al. (Zafar, Ashraf, Ali, Ahmed, Jabbar, & Chatzichristofis, 2018) utilized BoW feature encoding techniques (see Table 2.1) and achieve 89.70%, 87.23%, 87.01%, respectively, on 15-scene dataset. Table 5.7 shows that CNN based methods (Shaopeng Liu et al., 2019; Tang et al., 2017a; Wang et al., 2020) achieve

Table 5.7: The comparison of HF-MSF Model with state-of-the-art methods for 15-scene dataset.

Methods	Accuracy %
Liu, et al. (B. Liu et al., 2015)	89.70
GoogLeNet (Places365) (B. Zhou, et al., 2018)	90.19
GoogLeNet (ImageNet) (J. Deng et al., 2009)	91.12
Zafar et al. (Zafar et al., 2018)	87.07
Lin et al. (Lin et al., 2017)	87.23
Segmentation-based feature extraction method	92.58
G-MS2F (Tang et al., 2017a)	92.90
FTOTLM (Shaopeng Liu et al., 2019)	94.01
DFF-ADML (Wang et al., 2020)	96.39
RLML-LSR (Wang et al., 2021)	93.50
HF-MSF model: (model-HSF using ResNet multi-layer features)	**97.86**
HF-MSF Model: (model- HFF using GoogLeNet multi-layer features)	**97.41**

higher recognition accuracy by taking the advantages of multi-layer features. HF-MSF model also utilized the advantages of multi-layers features of ResNet model and by adopting a score-level strategy, it reaches to 94.16% without adding external handcrafted features (see Figure 5.3(b)). At the end, full HF-MSF model with score-level strategy, including multi-layer features and handcrafted features, it achieves 97.86% recognition accuracy on 15-scene dataset, which is superior to the existing methods. The Segmentation-based feature extraction method (see Chapter 3) achieves 92.58% performance on 15-scene dataset. However, HF-MSF model improves the accuracy of 5.10% compared segmentation-based feature extraction method. It

demonstrates that combining rich handcrafted features with deep features of multi-layer improves the accuracy of the image scene recognition.

The existing methods are also evaluated on the ‘stage dataset 2’. For fair comparison to existing methods, we apply the Nedovic et al. (Nedovic et al., 2010) and J. Sánchez et al. (Sanchez et al., 2013) methods on the ‘stage dataset 2’ by utilizing 80% part training and 20% part for testing. Results are reported in Table 5.8. Nedovic et al. (Nedovic et al., 2010) features are extracted by partition the image into 4×4 patches and features are combined into a single vector for each image. In J. Sánchez et al. method (Sanchez et al., 2013), the Gaussian components in the GMM for encodes features is empirically set to be 64 (more detail of implementation is given in Subsection 3.2.3.2). We use SVM classifier for both methods and achieve maximum accuracy of 52.65% and 74.89%, respectively, as shown in Table 5.8. Next, the popular deep CNN architectures, which are the pre-trained on ImageNet dataset, namely GoogLeNet (Szegedy et al., 2015), ResNet-50 (He et al., 2016), AlexNet (Alex et al., 2012), and VGG-16 (Simonyan & Zisserman, 2015) are also employed on the ‘stage dataset 2’. These pre-trained weights are used as starting weights to learn the stage geometries. Each deep CNN parameters is fined tuned through the standard BP algorithms with a batch size 10. In order to obtain optimum performance of CNN models, the same learning parameters are used as described in Subsection 4.2.4. Training and testing time of each method is reported in Table 5.8. The results of deep CNN models are also reported same as given in Table 4.1. In standard CNN architectures, the GoogLeNet achieves maximum accuracy which reached to 82.25%.

Table 5.8: Comparison of HF-MSF model with state-of-the-art methods for the ‘stage dataset 2’.

Methods	Acc.%	Pr.%	Re.%	F-s%	Training +testing time/sec
Nedovic et al. (Nedovic et al., 2010)	52.65	51.95	52.37	52.04	58.0
Sánchez et al. (Sanchez et al., 2013)	74.89	71.57	68.23	69.11	180.0
GoogLeNet (Szegedy et al., 2015)	82.25	82.13	82.07	82.08	1009 min +43
AlexNet (Alex et al., 2012)	78.13	77.76	78.13	77.81	517min +16
VGG-16 Net (Simonyan & Zisserman, 2015)	80.88	80.50	80.87	80.36	2833 min +53
ResNet-50 (He et al., 2016)	81.88	81.92	81.88	81.73	1458min +33
TGF-DeepFF method	86.29	85.92	86.16	85.96	275.72
G-MS2F(Tang et al., 2017a)	82.96	82.13	82.82	82.27	47.29+4.35
HF-MSF Model: (Model-HSF, using GoogLeNet multi-layer features)	**95.17**	**95.13**	**95.03**	**95.06**	48.74+3.98
HF-MSF Model: (Model-HFF using GoogLeNet multi-layer features)	**93.96**	**93.92**	**92.87**	**93.88**	48.09+4.14

Raw data is given in Appendix W and Y.

Our HF-MSF model shows superior performance compared to existing methods for the 3D scene geometry recognition. The best performance of our method mainly benefits from the fusion of deep and handcrafted features, as our TGF-DeepFF method in Chapter 4 shows the positive influence of handcrafted feature fusion for image scene geometry recognition.

By involving the particular handcrafted features with multi-layer features CNN architecture, the accuracy significantly improved over the existing methods such as G-MS2F (Tang et al., 2017a) achieves 82.96% recognition accuracy when it applies on ‘stage dataset 2’. G-MS2F is implemented by using three auxiliary classifiers (explained in Subsection 2.2.2) and their outcome are combined using product-rule

(defined in Subsection 2.1.5.3). We use SVM with linear kernel, as author described in their study. The code is given at (Tang, Wang, & Kwong, 2017b). In contrast, the HF-MSF model approaches to 95.17% and 95.13% recognition accuracy when the score-level fusion using the product and sum rule is applied, respectively. It means that handcrafted features which are particularly contained the image scene geometry information has strong influence in image scene recognition of 'stage dataset 2' and it improves accuracy of 12.21% than existing G-MS2F (Tang et al., 2017a) method. It is shown that our HF-MSF model outperform by 12.92% of the G-MS2F (Tang et al., 2017a) method. On the other hand, it also improves 8.88% performance compared to TGF-DeepFF method. It is because of adding the discriminative information of scene geometry at each block of CNN and then applying the score-level fusion. The evaluation time of the HF-MSF model (training and testing) is approached to 52.72 sec and 52.23 sec for Model-HSF and Model-HHF, respectively, which is clear evidence that it is faster than BP techniques that takes a long time to fine tune it parameters.

5.4 Summary

In summary, we present a solution based on multi-layer features of CNN and handcrafted features fusion for the problem of 3D scene geometry recognition of a single image. The handcrafted features contain rich information of image scene geometry, including shape, depth, and color. Fusing the handcrafted features at different blocks of multi-layer CNN architecture improves the discriminative information of image geometry, which is important for 3D scene geometry recognition of a single image. Furthermore, HF-MSF model utilizes score-level and feature-fusion strategies to gain the highest recognition accuracy. In score-level fusion, the multi-layer features combined with handcrafted features at each blocks individually fused

and the classifier at each block which is used to learn the image representation, then their predicted scores at each block are combined using sum or the product rule to obtain final category type. In feature-level fusion, the handcrafted features and multi-layers features of CNN from different blocks are combined into a single feature vector and fed into a classifier to obtain final decision. Finally, to compare the effectiveness of the HF-MSF model, it is evaluated on two different datasets. One is a 3D scene geometry dataset, called the 'stage dataset 2', while the other is the '15-scene dataset', which is useful in image scene recognition. The HF-MSF model is built on well-known deep architectures, namely GoogLeNet and ResNet architectures. The deep features are extracted at three different blocks of GoogLeNet and five different residual blocks of ResNet model. Two classifiers, linear SVM and ELM are used with score-level fusion and feature-level fusion strategies. Analysis shows that score-level fusion using SVM classifier achieves best performance for both datasets. The HF-MSF model achieves maximum accuracy of 97.86% on 15-scene dataset and 95.17% on 'stage dataset 2' by using ResNet and GoogLeNet models, respectively. We also tested the AlexNet architecture with four different layers for feature extraction on 'stage dataset 2', while the accuracy was low compared to ResNet and GoogLeNet architectures, which shows that ResNet and GoogLeNet capture strong scene discriminative information. Compared to state-of-the-art methods, HF-HSF model achieves superior performance on the both datasets with 12.21% and 4.96% which increases accuracy versus the G-MS2F (Tang et al., 2017a) method and 11.92% and 3.85% increase accuracy versus FTOTLM (Shaopeng Liu et al., 2019) method on 'stage 2' and '15-scene' image datasets, respectively. On the other hand, compared to DFF-ADML (Wang et al., 2020) and RLML-LSR (Wang et al., 2021) methods, our HF-MSF model increases 1.47%, 4.36% accuracy on 15-scene dataset, respectively. Compared to

segmentation-based feature extraction method, HF-MSF model improves the accuracy of 5.10% when the 15-scene image dataset is used. Summarily, the results indicate that our contributions provide superior accuracy under two different CNN models with SVM and ELM classifiers. However, model is designed for medium scale 3D scene geometry datasets. Furthermore, it also involves the handcrafted features which needs to extract beside with deep CNN features that increase the computational time compared to standard CNN model.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we investigate the problem of 3D scene recognition from a single image, which is an important task for many computer vision applications, i.e., 3D TV, vehicle navigation system, video categorization. In this study, we explore different aspects of image scene, such as image features, segmentation techniques, dataset availability, deep CNN features, and handcrafted features integration at different blocks to obtain higher recognition rate. The most recent approaches of feature extraction, machine learning and image processing are adapted to tackle the 3D scene recognition problem.

In this thesis, we introduce three different methods of 3D scene recognition. In the first method, we explored the different features that describe the structure information of image scene, and predefined templates with respect to scene geometries, and the segmentation process is utilized to obtain high accuracy on small and medium-scale datasets. Thus, the segmentation-based feature extraction method of 3D scene recognition is implemented to tackle the problem of the 3D scene recognition for small and medium scale datasets and to achieve the significant recognition accuracy of 3D scene recognition. The handcrafted features, including HOG, color (RGB, HSV), parameters of the Weibull distribution, local binary patterns, and entropy value are studied and these features are fused for each patch and further these fused feature

vectors are combined into a single vector based on the template-based segmentation, which is individually fed into an SVM classifier. In this way, this method reduces the intra-class variation problem. Finally, the obtained results of these eight classifiers are integrated by using sum-rule. Compared with the state-of-the-art methods, our segmentation-based feature extraction method obtained the significant improvement in 3D scene geometry recognition accuracy on two different scene datasets. The results are motivating in the sense that applying the appropriate templates and particular image features improve the 3D scene recognition accuracy on small and medium-scale datasets when the images with relatively clear geometry structures are given.

Accordingly, in the second method, the problem of 3D scene recognition is tackled for the medium-scale dataset by using texture gradient features and deep CNN feature fusion method (TGF-DeepFF). It reduces the intra-class variation problem between the similar stages by using additional features of parameters of the Weibull distribution. As CNN models require a large dataset for training purpose and 3D scene geometry dataset is not publically available. Therefore, a novel 3D scene geometry dataset, 'stage dataset 2' is constructed to handle this issue. The key advantages of using texture gradient features are that rich image scene geometry knowledge is easy to extract. Finally, ELM classifier is applied to learn the 3D scene geometry model. The TGF-DeepFF method is evaluated on introduced dataset, 'stage dataset 2', and results exhibit that the TGF-DeepFF method obtains 86.29% recognition accuracy, which is higher and it requires less training time compared to standard CNN architectures, including AlexNet (Alex et al., 2012), GoogLeNet (Szegedy et al., 2015), VGG-16 (Simonyan & Zisserman, 2015) and ResNet-50 (He et al., 2016) architectures.

Finally, in the third method, a novel approach is presented based on multi-layer CNN's features and handcrafted features fusion to the problem of 3D scene geometry recognition of a single image (HF-MSF). Above proposed methods show reliable performance on small or medium-scale datasets, but losing scene to object relationship because using the predefined structure and human interaction in feature extraction does not represent well of image scene geometry structure when complex image scenes are given. And TGF-DeepFF method uses CNN architecture which do not well train on the small or medium scale datasets. Thus the HF-MSF approach is proposed to achieve high recognition accuracy of the medium-scale scene geometry dataset. The related works did not pay attention to use the handcrafted and deep feature fusion at the intermediate layer. In this system, the handcrafted features and multi-layer features are fused at different blocks. The handcrafted features contain rich information of image scene geometry, including shape, depth, and color. Fusing the handcrafted features at different blocks of multi-layer CNN architecture improves the discriminative information of image geometry, which is important for 3D scene geometry recognition of a single image when the medium scale datasets are given. Furthermore, the proposed HF-MSF model utilizes score-level and feature-fusion strategies to gain the highest recognition accuracy. The analysis shows that score-level fusion using the SVM classifier achieves the best performance for both datasets. The proposed HF-MSF approach achieves maximum accuracy of 97.86% on 15-scene dataset and 95.17% on 'stage dataset 2' by using ResNet and GoogLeNet models, respectively, which is superior to the state-of-the-art methods. Result indicates that the proposed approach achieves high accuracy on medium-scale scene datasets using additional handcrafted features with multi-layer CNN features.

Summarily, the 3D scene recognition problem is studied in this thesis and it can be recommended in three different aspects:

1) Template-based segmentation method (see Chapter 3) is used to achieve high recognition accuracy on small and medium-scale datasets with relatively clear geometric structures (see Tables 3.2, 3.3, and 3.5). We recommend this method for small-scale image scene geometry datasets with relatively clear geometric structure, see Table 6.1.

2) Texture Gradient Features and Deep CNN Feature Fusion (TGF-DeepFF) method is introduced for medium-scale image scene recognition. The proposed TGF-DeepFF method is useful for capturing the accurate depth and scene structure information as it shows better accuracy on 'stage dataset 2' (see Table 4.1) compared to standard CNN architecture. It takes benefits of texture gradient features (Nedovic et al., 2010), which provides rich information of scene depth and by applying the extreme learning machine (Huang et al., 2006), it fine-tunes the scene recognition model faster than standard a backpropagation algorithm (see Table 4.1). Recommendations are given in Table 6.1 (row 2) that it can be useful for medium scale scene geometry dataset. It is very fast in training compared to standard CNNs based backpropagation algorithm.

3) Finally, if the complex scene medium scale datasets are given, then the Handcrafted Features with CNN Multi-Stages Features (HF-MSF) approach can be applied. As it uses additional features with multi-layer CNN features and achieving 95.17% and 97.87% recognition accuracy on 'stage dataset 2' and '15-scene' dataset, it can be recommended (see Table 6.1, row 3) for medium-scale datasets when high accuracy is required. It is complex compared to segmentation-based feature extraction and texture gradient features and deep CNN feature fusion method, but it achieves high accuracy on medium-scale datasets.

Furthermore, the TGF-DeepFF and HF-MSF approaches are proposed for medium-scale datasets and they use additional handcrafted features when the large scale datasets are not given because the images are not enough to well train the CNN models. Therefore, the accuracy of both approaches are higher than baseline methods.

Table 6.1: Recommendations of introduced methods.

Chapter #	Methods	Recommendation
1	Segmentation-based feature extraction method	We recommend this method for small-scale image scene geometry datasets with relatively clear geometric structures, as it shows high accuracy on 'stage dataset 1' (see Tables 3.2 and 3.4).
2	Texture Gradient Features and Deep CNN Feature Fusion method	It is recommended for medium-scale scene dataset as it shows higher accuracy compared to standard CNN architectures (see Table 4.1). It is very fast in training compared to standard CNN architectures (see Table 4.1). It can be useful for real-time systems.
3	Handcrafted Features with CNN Multi-Stages Features model	It is applicable for medium-scale scene geometry datasets when the high accuracy of image scene recognition is required.

6.2 Future work

In future work, the 3D scene recognition may aggregate with object detection, 3D scene layout extraction by using advanced machine learning approaches. A recent study of deep CNN shows that the CNN can be used for objects detection and segmentation purpose, i.e., region based-CNN (R-CNN) and mask R-CNN (He, Gkioxari, Dollár, & Girshick, 2020). We expect that objects detection and scene layout extraction accuracy may improve when the 3D scene geometry is known of a single image. However, a deep study of CNN for object detection and scene layout extraction by following 3D scene geometry of a single image is required.

REFERENCES

Aghdam, H. H., & Heravi, E. J. (2018). Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification *Springer*, 1-303. doi:10.1007/978-3-319-57550-6

Alex, K., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. 1097--1105.

Ali, N., Zafar, B., Riaz, F., Hanif Dar, S., Iqbal Ratyal, N., Bashir Bajwa, K., Sajid, M. (2018). A Hybrid Geometric Spatial Image Representation for scene classification. *PLOS ONE, 13*(9), e0203339. doi:10.1371/journal.pone.0203339

Alom, M. Z., Taha, T., Yakopcic, C., Westberg, S., Hasan, M., Esesn, B., Asari, V. (2018). The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches.

Anderson, C. H., Van Essen, D. C., & Olshausen, B. A. (2005). CHAPTER 3 - Directed Visual Attention and the Dynamic Control of Information Flow. In L. Itti, G. Rees, & J. K. Tsotsos (Eds.), *Neurobiology of Attention* (pp. 11-17). Burlington: Academic Press.

Anguita, D., Boni, A., Ridella, S., Rivieccio, F., & Sterpi, D. (2005). Theoretical and Practical Model Selection Methods for Support Vector Classifiers. In L. Wang

(Ed.), *Support Vector Machines: Theory and Applications* (pp. 159-179). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ballabio, D., Grisoni, F., & Todeschini, R. (2018). Multivariate comparison of classification performance measures. *Chemometrics and Intelligent Laboratory Systems, 174*, 33-44.

Brébisson, A., & Vincent, P. (2015). An Exploration of Softmax Alternatives Belonging to the Spherical Loss Family.

Breiman, L. (2001). Random Forests. *Mach. Learn., 45*(1), 5–32. doi:10.1023/a:1010933404324

Brownlee, J. (April 24, 2019). Convolutional Neural Network Model Innovations for Image Classification. *Deep Learning for Computer Vision.* Retrieved from https://machinelearningmastery.com/review-of-architectural-innovations-for-convolutional-neural-networks-for-image-classification/, Feb 2020.

Carreira, J., & Sminchisescu, C. (2012). CPMC: Automatic Object Segmentation Using Constrained Parametric Min-Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 34*(7), 1312-1328. doi:10.1109/TPAMI.2011.231

Chan, T. F., & Vese, L. A. (2001). Active contours without edges. *IEEE Transactions on Image Processing, 10*(2), 266-277. doi:10.1109/83.902291

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning, 20*(3), 273-297. doi:10.1023/A:1022627411411

Cheng, D., Yu, W., He, X., Ni, S., Lv, J., Zeng, W., & Yuanlong, Y. (6-9 Dec. 2015). *Scene recognition based on extreme learning machine for digital video archive management.* Paper presented at the 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO).

D. Hoiem, A.A. Efros, & Hebert, M. (2007). Matlab code of Recovering Surface Layout from an Image. Retrieved from http://dhoiem.cs.illinois.edu/, Feb 2018.

Dalal, N., & Triggs, B. (20-25 June 2005). *Histograms of oriented gradients for human detection.* Paper presented at the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).

Deng, J., Ding, N., Jia, Y., Frome, A., Murphy, K., Bengio, S., Adam, H. (2014). *Large-Scale Object Classification Using Label Relation Graphs.* Paper presented at the Computer Vision – ECCV 2014, Cham.

Deng, J., Dong, W., Socher, R., Li, L., Kai, L., & Li, F.-F. (20-25 June 2009). *ImageNet: A large-scale hierarchical image database.* Paper presented at the 2009 IEEE Conference on Computer Vision and Pattern Recognition.

Duffy, S. F. (1997). Weibull Parameter Estimation *Theory and Background–Information, Connecticut Reserve Technologies, LLC, Cleaveland, Ohio 44114*, 1-22.

Faruk Ortes, Derya Karabulut, & Arslan, Y. Z. (2019). General Perspectives on Electromyography Signal Features and Classifiers Used for Control of Human Arm Prosthetics. *IGI Global*, 1-17. doi:http://doi:10.4018/978-1-5225-2255-3.ch043

G. Kumar, & Bhatia, P. K. (8-9 Feb. 2014). *A Detailed Review of Feature Extraction in Image Processing Systems.* Paper presented at the 2014 Fourth International Conference on Advanced Computing & Communication Technologies.

gettyimages. (accessed at March 2019). gettyimages data Retrieved from https://www.gettyimages.com/photos/, June 2018.

Geusebroek, J.-M., & Smeulders, A. W. M. (2005). A Six-Stimulus Theory for Stochastic Texture. *International Journal of Computer Vision, 62*(1), 7-16. doi:10.1007/s11263-005-4632-7

Geusebroek, J., Smeulders, A. W. M., & Weijer, J. v. d. (2003). Fast anisotropic Gauss filtering. *IEEE Transactions on Image Processing, 12*(8), 938-943. doi:10.1109/TIP.2003.812429

Gonzalez, R. C., & Woods, R. E. (2001). *Digital Image Processing*: Addison-Wesley Longman Publishing Co., Inc.

Gonzalez, R. C., & Woods, R. E. (2006). *Digital Image Processing (3rd Edition)*: Prentice-Hall, Inc.

Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing 4Th edition*. Pearson: Printed and bound in Malaysia.

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2003). *Digital Image Processing Using MATLAB*: Prentice-Hall, Inc.

Guang-Bin, H., Qin-Yu, Z., & Chee-Kheong, S. (25-29 July 2004). *Extreme learning machine: a new learning scheme of feedforward neural networks.* Paper presented at the 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541).

Hassantabar, S., Ahmadi, M., & Sharifi, A. (2020). Diagnosis and detection of infected tissue of COVID-19 patients based on lung x-ray image using convolutional neural network approaches. *Chaos, Solitons & Fractals, 140*, 110170. doi:https://doi.org/10.1016/j.chaos.2020.110170.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2020). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 42*(2), 386-397. doi:10.1109/TPAMI.2018.2844175

He, K., Zhang, X., Ren, S., & Sun, J. (27-30 June 2016). *Deep Residual Learning for Image Recognition.* Paper presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Hedau, V., Hoiem, D., & Forsyth, D. (29 Sept.-2 Oct. 2009). *Recovering the spatial layout of cluttered rooms.* Paper presented at the 2009 IEEE 12th International Conference on Computer Vision.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer & J. F. Kolen (Eds.), *A Field Guide to Dynamical Recurrent Neural Networks*: IEEE Press.

Hoiem, D., Efros, A. A., & Hebert, M. (17-21 Oct. 2005). *Geometric context from a single image.* Paper presented at the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1.

Hoiem, D., Efros, A. A., & Hebert, M. (17-22 June 2006). *Putting Objects in Perspective.* Paper presented at the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06).

Hoiem, D., Efros, A. A., & Hebert, M. (2007). Recovering Surface Layout from an Image. *International Journal of Computer Vision, 75*(1), 151-172. doi:10.1007/s11263-006-0031-y

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing, 70*(1), 489-501. doi:https://doi.org/10.1016/j.neucom.2005.12.126

Huang, G., Bai, Z., Kasun, L. L. C., & Vong, C. M. (2015a). Extreme learning machine, Matlab implementation. Retrieved from https://github.com/ExtremeLearningMachines/ELM-LRF, August 2019.

Huang, G., Bai, Z., Kasun, L. L. C., & Vong, C. M. (2015b). Local Receptive Fields Based Extreme Learning Machine. *IEEE Computational Intelligence Magazine, 10*(2), 18-29. doi:10.1109/MCI.2015.2405316

Iglhaut, J., Cabo, C., Puliti, S., Piermattei, L., O'Connor, J., & Rosette, J. (2019). Structure from Motion Photogrammetry in Forestry: a Review. *Current Forestry Reports, 5*(3), 155-168. doi:10.1007/s40725-019-00094-3

Jan Mark Geusebroek, Arnold W. M. Smeulders, & Weijer, J. v. d. (2007). Matlab code for anisotropic filter Retrieved from https://ivi.fnwi.uva.nl/isis/publications/bibtexbrowser.php?key=GeusebroekTIP2003&bib=all.bib, July 2018.

Joost van de Weijer , Theo Gevers , & Gijsenij, A. Retrieved from https://staff.fnwi.uva.nl/th.gevers/software.html, March 2019.

Jung, C., & Kim, C. (2012). Real-time estimation of 3D scene geometry from a single image. *Pattern Recognition, 45*(9), 3256-3269. doi:https://doi.org/10.1016/j.patcog.2012.02.028

Khan, A., Chefranov, A., & Demirel, H. (2020a). Image-Level Structure Recognition Using Image Features, Templates, and Ensemble of Classifiers. *Symmetry*, *12*(7), 1072. doi:doi.org/10.3390/sym12071072

Khan, A., Chefranov, A., & Demirel, H. (2020b, 10-12 Aug. 2020). *Texture Gradient and Deep Features Fusion-Based Image Scene Geometry Identification System Using Extreme Learning Machine.* Paper presented at the 2020 3rd International Conference of Intelligent Robotic and Control Engineering (IRCE), University of Oxford, UK.

Khoo, Y.-H., Goi, B.-M., Chai, T.-Y., Lai, Y.-L., & Jin, Z. (2018). *Multimodal Biometrics System Using Feature-Level Fusion of Iris and Fingerprint*. Paper presented at the Proceedings of the 2nd International Conference on Advances in Image Processing, Chengdu, China. https://doi.org/10.1145/3239576.3239599

Kim, S., Kavuri, S., & Lee, M. (2013). *Deep Network with Support Vector Machines.* Paper presented at the Neural Information Processing, Berlin, Heidelberg.

Kittler, J., Hatef, M., Duin, R. P. W., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(3), 226-239. doi:10.1109/34.667881

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Commun. ACM, 60*(6), 84-90. doi:10.1145/3065386

Lazebnik, S., Schmid, C., & Ponce, J. (17-22 June 2006). *Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories.* Paper presented at the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436-444. doi:10.1038/nature14539

Lee, C., Badrinarayanan, V., Malisiewicz, T., & Rabinovich, A. (22-29 Oct. 2017). *RoomNet: End-to-End Room Layout Estimation.* Paper presented at the 2017 IEEE International Conference on Computer Vision (ICCV).

Lei, Y., Karimi, H. R., Cen, L., Chen, X., & Xie, Y. (2021). Processes soft modeling based on stacked autoencoders and wavelet extreme learning machine for aluminum plant-wide application. *Control Engineering Practice, 108,* 104706. doi:https://doi.org/10.1016/j.conengprac.2020.104706

Lin, G., Fan, C., Zhu, H., Miu, Y., & Kang, X. (2017). Visual feature coding based on heterogeneous structure fusion for image classification. *Inf. Fusion, 36*(C), 275–283. doi:10.1016/j.inffus.2016.12.010

Linda G. Shapiro , & Stockman, G. C. (2001). *Computer Vision* (1st Edition ed.): Pearson, ISBN-10: 0130307963.

Liu, B., Liu, J., & Lu, H. (2015). Learning representative and discriminative image representation by deep appearance and spatial coding. *Computer Vision and*

Image Understanding, 136, 23-31. doi: https://doi.org/10.1016/j.cviu.2015.03.006

Liu, S., & Deng, W. (3-6 Nov. 2015). *Very deep convolutional neural network based image classification using small training sample size.* Paper presented at the 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR).

Liu, S., Tian, G., & Xu, Y. (2019). A novel scene classification model combining ResNet based transfer learning and data augmentation with a filter. *Neurocomputing, 338*, 191-206. doi: https://doi.org/10.1016/j.neucom.2019.01.090

Lou, Z., Gevers, T., & Hu, N. (2015). Extracting 3D Layout From a Single Image Using Global Image Structures. *IEEE Transactions on Image Processing, 24*(10), 3098-3108. doi:10.1109/TIP.2015.2431443

Lowe, D. G. (20-27 Sept. 1999). *Object recognition from local scale-invariant features.* Paper presented at the Proceedings of the Seventh IEEE International Conference on Computer Vision.

Luo, N., Sun, Q., Chen, Q., Ji, Z., & Xia, D. (2015). A Novel Tracking Algorithm via Feature Points Matching. *PLOS ONE, 10*(1), e0116315. doi:10.1371/journal.pone.0116315

Marculescu, D., Stamoulis, D., & Cai, E. (2018). *Hardware-aware machine learning: modeling and optimization*. Paper presented at the Proceedings of the

International Conference on Computer-Aided Design, San Diego, California. https://doi.org/10.1145/3240765.3243479

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*: Henry Holt and Co., Inc.

Mei Wang, & Deng, W. (2018). Deep Face Recognition: A Survey. *arXiv preprint arXiv:1804.06655*.

Mensink, T. (2012). Matlab code of Retrieved from https://github.com/tmensink/fvkit, July 2018.

Milton Abramowitz , & Stegun, I. A. (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (9th ed.).

Mohandes, M., Deriche, M., & Aliyu, S. O. (2018). Classifiers Combination Techniques: A Comprehensive Review. *IEEE Access, 6*, 19626-19639. doi:10.1109/ACCESS.2018.2813079

Nanni, L., Ghidoni, S., & Brahnam, S. (2017). Handcrafted vs. non-handcrafted features for computer vision classification. *Pattern Recognition, 71,* 158-172. doi:https://doi.org/10.1016/j.patcog.2017.05.025

Nedovic, V., Smeulders, A. W., Redert, A., & Geusebroek, J. M. (2010). Stages as models of scene geometry. *IEEE Trans Pattern Anal Mach Intell, 32*(9), 1673-1687. doi:10.1109/TPAMI.2009.174

Nikisins, O. (2020). Local binary patterns transformation of the input image. Retrieved from https://ch.mathworks.com/matlabcentral/fileexchange/37781-local-binary-patterns-transformation-of-the-input-image, June 2018.

Ojala, T., Pietik, M., & Maenpaa, T. (2002). Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Trans. Pattern Anal. Mach. Intell., 24*(7), 971-987. doi:10.1109/tpami.2002.1017623

Oliva, A., & Torralba, A. (2001a). Gist descriptor Matlab code Retrieved from https://people.csail.mit.edu/torralba/code/spatialenvelope/, June 2018.

Oliva, A., & Torralba, A. (2001b). Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *International Journal of Computer Vision, 42*(3), 145-175. doi:10.1023/A:1011139631724

Özyurt, F. (2020). A fused CNN model for WBC detection with MRMR feature selection and extreme learning machine. *Soft Computing*, *24*(11), 8163-8172. doi:10.1007/s00500-019-04383-8

Özyurt, F., Sert, E., & Avcı, D. (2020). An expert system for brain tumor detection: Fuzzy C-means with super resolution and convolutional neural network with extreme learning machine. *Medical Hypotheses*, *134*, 109433. doi:https://doi.org/10.1016/j.mehy.2019.109433

Paris, S., Hasinoff, S. W., & Kautz, J. (2011). Local Laplacian filters: edge-aware image processing with a Laplacian pyramid. *ACM Trans. Graph., 30*(4), 1-12. doi:10.1145/2010324.1964963

Patalas, M., & Halikowski. (2019). A Model for Generating Workplace Procedures Using a CNN-SVM Architecture. *Symmetry, 11*(9), 1151. doi:10.3390/sym11091151

Penatti, O. A. B., Nogueira, K., & Santos, J. A. d. (7-12 June 2015). *Do deep features generalize from everyday objects to remote sensing and aerial scenes domains?* Paper presented at the 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Boston, MA, USA.

Phung, & Rhee. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. *Applied Sciences, 9*, 4500. doi:10.3390/app9214500

Pietikäinen, M., Hadid, A., Zhao, G., & Ahonen, T. (2011). Computer Vision Using Local Binary Patterns. In M. Pietikäinen, A. Hadid, G. Zhao, & T. Ahonen (Eds.), *Computer Vision Using Local Binary Patterns* (pp. E1-E2). London: Springer London.

Pietikäinen, M., & Zhao, G. (2016). Two decades of local binary patterns: A survey. *CoRR, abs/1612.06795*.

Quattoni, A., & Torralba, A. (20-25 June 2009). *Recognizing indoor scenes.* Paper presented at the 2009 IEEE Conference on Computer Vision and Pattern Recognition.

Richards, W., Jepson, A., & Feldman, J. (1996). Priors, preferences and categorical percepts. In C. K. David & R. Whitman (Eds.), *Perception as Bayesian inference* (pp. 93-122): Cambridge University Press.

Rosset, S. (2004). *Model selection via the AUC*. Paper presented at the Proceedings of the twenty-first international conference on Machine learning, Banff, Alberta, Canada.

Sanchez, J., Perronnin, F., Mensink, T., & Verbeek, J. (2013). Image Classification with the Fisher Vector: Theory and Practice. *Int. J. Comput. Vision, 105*(3), 222-245. doi:10.1007/s11263-013-0636-x

Scott, D. W. (1992). Histograms: Theory and Practice, Chapter 3 *Multivariate Density Estimation* (pp. 47-94).

Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556* http://arxiv.org/abs/1409.1556. http://arxiv.org/abs/1409.1556

Snelick, R., Uludag, U., Mink, A., Indovina, M., & Jain, A. (2005). Large-scale evaluation of multimodal biometric authentication using state-of-the-art

systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(3), 450-455. doi:10.1109/TPAMI.2005.57

Somvanshi, M., Chavan, P., Tambade, S., & Shinde, S. V. (12-13 Aug. 2016). *A review of machine learning techniques using decision tree and support vector machine.* Paper presented at the 2016 International Conference on Computing Communication Control and automation (ICCUBEA).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Rabinovich, A. (2015). *Going deeper with convolutions*. https://doi.org/10.1109/CVPR.2015.7298594

Szeliski, R. (2011). Computer vision algorithms and applications. London; New York: Springer.

Tang, P., Wang, H., & Kwong, S. (2017a). G-MS2F: GoogLeNet based multi-stage feature fusion of deep CNN for scene recognition. *Neurocomputing, 225*, 188-197. doi:https://doi.org/10.1016/j.neucom.2016.11.023

Tang, P., Wang, H., & Kwong, S. (2017b). G-MS2F: GoogLeNet Based Multi-Stage Feature Fusion of Deep CNN for Scene Recognition. Retrieved from https://mic.tongji.edu.cn/51/46/c9778a86342/page.htm, August, 2020.

Tang, J., Deng, C., Huang, G., & Zhao, B. (2015). Compressed-Domain Ship Detection on Spaceborne Optical Image Using Deep Neural Network and

Extreme Learning Machine. *IEEE Transactions on Geoscience and Remote Sensing, 53*(3), 1174-1185. doi:10.1109/TGRS.2014.2335751

Tomasi, C. (2012). Histograms of oriented gradients. *Computer Vision Sampler* 1-6.

Torralba, A., & Oliva, A. (2002). Depth estimation from image structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(9), 1226-1238. doi:10.1109/TPAMI.2002.1033214

Tulyakov, S., Jaeger, S., Govindaraju, V., & Doermann, D. (2008). Review of Classifier Combination Methods. In S. Marinai & H. Fujisawa (Eds.), *Machine Learning in Document Analysis and Recognition* (pp. 361-386). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ujjwalkarn. (August 9, 2016). A Quick Introduction to Neural Networks. Retrieved from https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/, Dec 2020.

Wang, C., Peng, G., & De Baets, B. (2020). Deep feature fusion through adaptive discriminative metric learning for scene recognition. *Information Fusion, 63*, 1-12. doi:https://doi.org/10.1016/j.inffus.2020.05.005

Wang, C., Peng, G., & Lin, W. (2021). Robust local metric learning via least square regression regularization for scene recognition. *Neurocomputing, 423*, 179-189. doi:https://doi.org/10.1016/j.neucom.2020.08.077

Weng, Q., Mao, Z., Lin, J., & Guo, W. (2017). Land-Use Classification via Extreme Learning Classifier Based on Deep Convolutional Features. *IEEE Geoscience and Remote Sensing Letters, 14*(5), 704-708. doi:10.1109/LGRS.2017.2672643

Weihull, W. (1951). A statistical distribution function of wide applicability. *J Appl Mech, 18*, 290-293.

Weijer, J. v. d., Gevers, T., & Gijsenij, A. (2007). Edge-Based Color Constancy. *IEEE Transactions on Image Processing, 16*(9), 2207-2214. doi:10.1109/TIP.2007.901808

Weisstein, & W., E. (2020). Vector Norm. Retrieved from https://mathworld.wolfram.com/VectorNorm.html, Dec., 2020.

Winn, J., Criminisi, A., & Minka, T. (17-21 Oct. 2005). *Object categorization by learned universal visual dictionary.* Paper presented at the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, Beijing, China

Xiao, J., Hays, J., Ehinger, K. A., Oliva, A., & Torralba, A. (13-18 June 2010). *SUN database: Large-scale scene recognition from abbey to zoo.* Paper presented at the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA

Xie, L., Lee, F., Liu, L., Kotani, K., & Chen, Q. (2020). Scene recognition: A comprehensive survey. *Pattern Recognition, 102*, 107205. doi:https://doi.org/10.1016/j.patcog.2020.107205

Xin, Y., Kong, L., Liu, Z., Wang, C., Zhu, H., Gao, M., Xu, X. (2018). Multimodal Feature-Level Fusion for Biometrics Identification System on IoMT Platform. *IEEE Access, 6*, 21418-21426. doi:10.1109/ACCESS.2018.2815540

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging, 9*(4), 611-629. doi:10.1007/s13244-018-0639-9

Yang, Y., & Newsam, S. (2010). *Bag-of-visual-words and spatial extensions for land-use classification*. Paper presented at the Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, San Jose, California.

Yann LeCun, Corinna Cortes, & Burges, C. J. (2010). [online] MNIST hand-written digit database. AT&T Labs.

Yu, W., Yang, K., Yao, H., Sun, X., & Xu, P. (2017). Exploiting the complementary strengths of multi-layer CNN features for image retrieval. *Neurocomputing, 237*, 235-241. doi:https://doi.org/10.1016/j.neucom.2016.12.002

Zafar, B., Ashraf, R., Ali, N., Ahmed, M., Jabbar, S., & Chatzichristofis, S. A. (2018). Image classification by addition of spatial information based on histograms of

orthogonal vectors. *PLOS ONE, 13*(6), e0198175. doi:10.1371/journal.pone.0198175

Zafar, B., Ashraf, R., Ali, N., Ahmed, M., Jabbar, S., Qureshi, K. N., Jeon, G. (2018). Intelligent image classification-based on spatial weighted histograms of concentric circles. *Comput. Sci. Inf. Syst., 15*, 615-633.

Zarbakhsh, P., & Demirel, H. (2018). Low-rank sparse coding and region of interest pooling for dynamic 3D facial expression recognition. *Signal, Image and Video Processing, 12*(8), 1611-1618. doi:10.1007/s11760-018-1318-5

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., & Torralba, A. (2018). Places: A 10 Million Image Database for Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40*(6), 1452-1464. doi:10.1109/TPAMI.2017.2723009

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., & Oliva, A. (2014). *Learning deep features for scene recognition using places database*. Paper presented at the Proceedings of the 27th International Conference on Neural Information Processing Systems, Volume 1, Montreal, Canada. https://dl.acm.org/doi/10.5555/2968826.2968881

Zurada, J. (1992). *Introduction to artificial neural systems*: West Publishing Co. https://dl.acm.org/doi/book/10.5555/131393.

APPENDICES

Appendix A: Predefined Template Structure

Template,T, is represented by an array of $R \times C$ size same as input image I. Each element of an array is an integer number in a range of $SS = [0..S-1], S = 2 \ldots Nb$ is the number of segments/components, Ns, in the template, T. Nb is obtained by $= \lceil log_2 Ns \rceil$. Element $T_{ij} = k$, where k is from SS, if the pixel $p(i,j)$ in an image shall belong to the k-th segment. If $Ns = 2$, then $Nb = 1$ and $2^1 = 2$ (2 are segments in one template, e.g. shown in Figure A.1).

1,1							1,C
R,1							R,C

Figure A.1: Image I with R rows and C columns. Coordinates of four corners are shown. Each cell represents a pixel of 2D image I.

The some templates' example are given below:
Template T_1 from Figure 2.8 (a) can be represented by equation (A.1). Assume T_1 same size as input image I, then,

$$\begin{cases} T_1\left(1:\left\lceil\frac{R}{2}\right\rceil, 1:C\right) = 1, \\ \text{and} \\ T_1\left(\left\lceil\frac{R}{2}\right\rceil + 1:R, 1:C\right) = 0. \end{cases} \quad (A.1)$$

So, resultant T_1 can be visualized by Figure A.2.

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(a) Template with each pixel value

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b) Template with each pixels value and shadow.

Figure A.2: Implementation of template T_1 for $R = C = 8$.

The template T_2 (Figure 2.8 (b)) can be obtained by dividing the image into 3 horizontal components as it is shown in Figure A.3 (b). Thus,

$$\begin{cases} \mathrm{T}_2\left(1{:}\left\lceil\frac{\mathrm{R}}{3}\right\rceil, 1{:}\,\mathrm{C}\right) = 1, \\ \text{and} \\ \mathrm{T}_2\left(\frac{\mathrm{R}}{3}{:}\left\lceil\frac{\mathrm{R}}{3}\right\rceil + \left\lceil\frac{\mathrm{R}}{3}\right\rceil, 1{:}\,\mathrm{C}\right) = 2, \\ \text{and} \\ \mathrm{T}_2\left(\left\lceil\frac{\mathrm{R}}{3}\right\rceil + \left\lceil\frac{\mathrm{R}}{3}\right\rceil + 1{:}\,\mathrm{R}, 1{:}\,\mathrm{C}\right) = 3. \end{cases} \quad \text{(A.2)}$$

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3

(a)

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3

(b)

Figure A.3. Implementation of template T_2. (a) is showing the value of each pixel and (b) is a segments with boundaries by following these values.

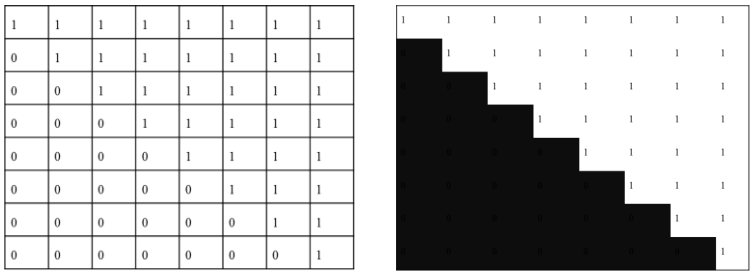
Template T_3 from Figure 2.8(c) is formed a triangle. Hence, we assume three points, $(p1, p2, p3)$ to generate mask of triangle using image coordinates.

$$p1 = T3\,(1,1),$$
$$p2 = T3\,(1,C),$$
$$P3 = T3\,(R,C),$$

$$t_3=\{\mathrm{T3}(i,j) = 0;\ i = 1,2,\ldots,R, j = 1,2,\ldots,C,$$

$$T_3=\ \{pixel\_inside\_Triangle\ (\mathrm{t3}, p1, p2, p3) = 1. \quad \text{(A.3)}$$

$Image = pixel\_inside\_Triangle(\ p1, p2, p3) = 1$, where the pixel lie inside the triangle, the value is considered to be 1. And remaining pixels value will be zero. Example is shown in Figure A.5.



(a) (b)

Figure A.5: Implementation of template Figure 2.8(c). (a) is showing the value of each pixel and (b) is a segments with boundaries by following these values.

Template T_4 from Figure 2.8 (d) can be defined by using equation (A.3). Only value of the parameters will be changed. Thus, the value of the parameter points $(p1, p2, p3)$ is:

$$T4 = size(I);$$
$$p1 = T4\,(R, 1)$$
$$p2 = T4\,(1, C)$$
$$P3 = T4\,(R, C)$$

0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

(a)

0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

(b)

Figure A.6: Implementation of template T_4. (a) is showing the value of each pixel and (b) is a segments with boundaries by following these values.

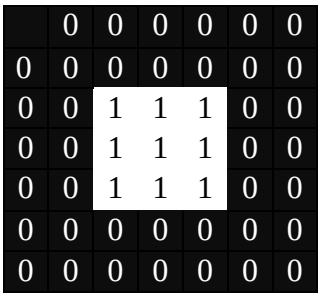
Template T_5 (Figure 2.8(e)) can be obtained by,

$$\begin{cases} \mathrm{T5}(i,j) = 0;\ i = 1,2,\dots,R;\ j = 1,2,\dots,C, \\ \mathrm{T5}\left(\frac{\mathrm{R}}{2} - \frac{\mathrm{R}}{8} : \left\lceil \frac{R}{2} + \frac{R}{8} \right\rceil, \frac{\mathrm{C}}{2} - \frac{\mathrm{C}}{8} : \left\lceil \frac{C}{2} + \frac{C}{8} \right\rceil\right) = 1. \end{cases} \quad \text{(A.4)}$$

The graphical representation of the template T_5 (e) is given in Figure A.7.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

(a)



(b)

Figure A.7: Implementation of template T_5. (a) is showing the value of each pixel and (b) is segments with boundaries by following these values.

Appendix B: Basic Description of Parameters of Weibull Distribution

Weibull distribution was introduced by the Swedish physicist Waloddi Weibull in 1939 (Weihull, 1951). The definition of Weibull distribution is, a random variable X is said to have a Weibull distribution with parameters α and β where α>0, β>0; if the pdf of X is,

$$f(x;\alpha,\beta) = \begin{cases} \frac{\alpha}{\beta^{\alpha}} x^{\alpha-1} e^{-(x/\beta)^{\alpha}} & x \geq 0 \\ 0 & x < 0 \end{cases} . \quad \text{(B.1)}$$

Where, α is shape and β is scale parameter and x is real number. Both α and β can be varied to get the number, as it looking in density curves. Figure B.1 illustrates the α is shape and β is scale parameter. Figure B.2 illustrates α is shape and β is scale parameter and it shows different scale of β for (0.5, 1, 2). In Figure B.2 (a-d) shows examples of the Weibull distribution for β=1 with varying the value of $\alpha = \{0.5, 1, 2, 4\}$. Increasing the value of α, it yields smoother curve.

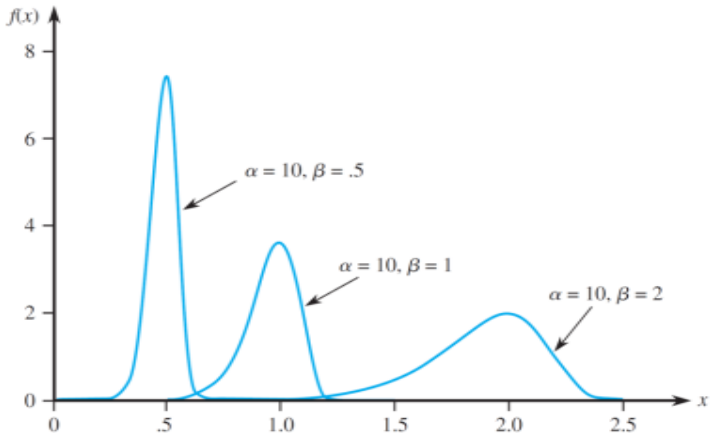
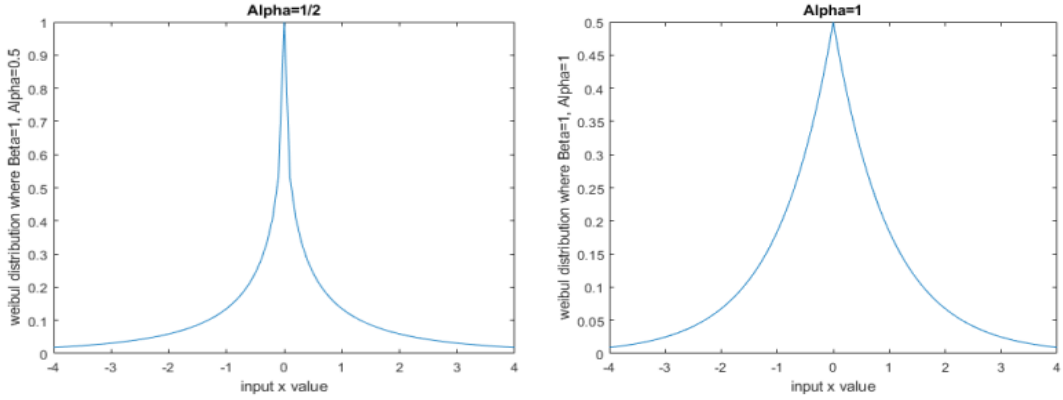
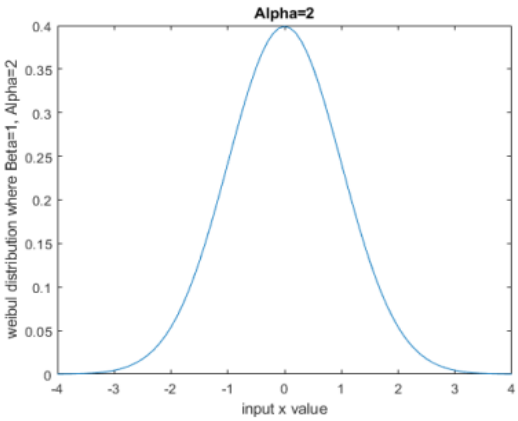


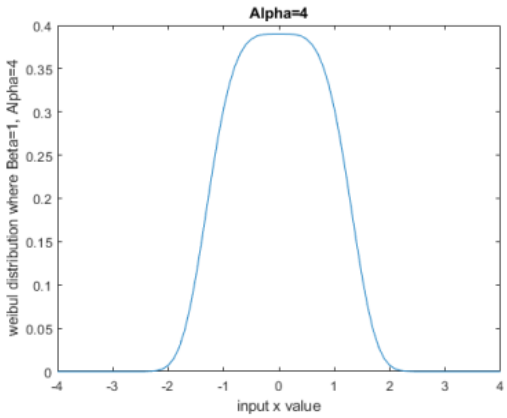
Figure B.1: Weibull distribution with different α and β values using equation (B.1).



(a) alpha =0.5 Beta =1 (b) Alpha =1, Beta =1



(c) Alpha=2, Beta=1



(d) Alpha=4, Beta =1

Figure B.2: Representation of Weibull distribution and its parameters.

Appendix C: Relation between Parameter of Weibull Distribution and Histogram

The relationship of histogram and weibull distribution can be visualized using Matlab package, as shown in Figure C.1. The $b = betarnd(A, B, m, r)$ is a Matlab function, geneates random numbers from the beta distribution with parameters specified by A and B. The m is a size of radom data. The b is an output of beta distriubtion with dimention of $r \times m$. Assume a data sample m=100 with paramters (3, 20), then beta distribution is: b = betarnd(3,20,100,1). It returns the beta values with size of 1×100. Then the histogram using n bins with beta distrubtion can be constructed using matlab funciton, $Histfit(b, n, 'beta')$, where n (we use 15 in Figure C.1) are bins of histogram and 'beta' is indicating beta distribution value.

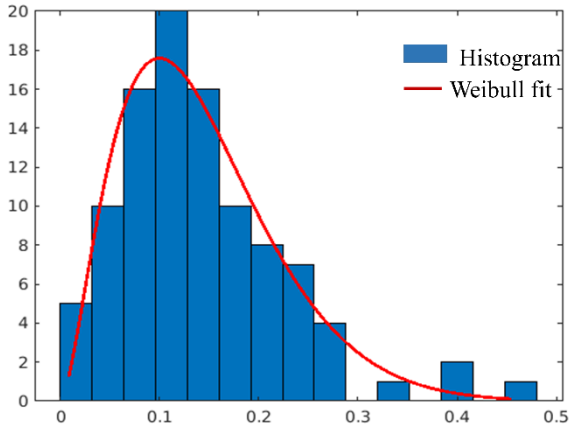


Figure C.1: Histogram and weibull fit (beta) of random data.

Figure C.1 indicates the histgoram gives the discrete representation of weibull parameters. Therefore, it is contructed for weibull parameters estimation.

Appendix D: Analysis of Change of Parameters of Weibull Distribution with Respect to Change of Image Depth

The change in Weibull parameters over depth is demonstrated in Figure D.1. We describe here the (Nedovic et al., 2010) experiments of parameters of Weibull distribution with respect to depth. Nedovic et al. direction (Nedovic et al., 2010) compute the α and β parameters of Weibull distribution for vertical position and horizontal position by dividing the input image into patches and individual average them along the direction perpendicular to change in depth. The Figure D.1(a) (see at next page) demonstrates the vertical image position, in which the Weibull parameters are averaged along x-direction (horizontal axis) over the n number of patches. The Figure D.1(d) demonstrates the horizontal image position, in which the Weibull parameters are average along y-direction (vertical axis) over the n patches, more description is given in (Nedovic et al., 2010), p. 1677. The the average along n patches can be defined as,

$$avg\_\alpha = \frac{1}{n}\sum_{i=1}^{n}\alpha_i, \tag{D.1}$$

$$avg\_\beta = \frac{1}{n}\sum_{i=1}^{n}\beta_i. \tag{D.2}$$

Similarly, in the horizontal image position, the Weibull parameters are averaged along y-direction (vertical axis) over the n patches. These average values of Weibull parameters in x and y-directions are plotted in Figure D.1 (b, c), and Figure D.1 (e, f) respectively, for two different surfaces, the implementation is followed by (Nedovic et al., 2010), p.1677. According to the nature of an image (see Figure D.1 (a)), α increases (see Figure D.1 (b, e)) with the depth of an image and β decreases slightly with the depth of the image (see Figure D.1 (c, f)).

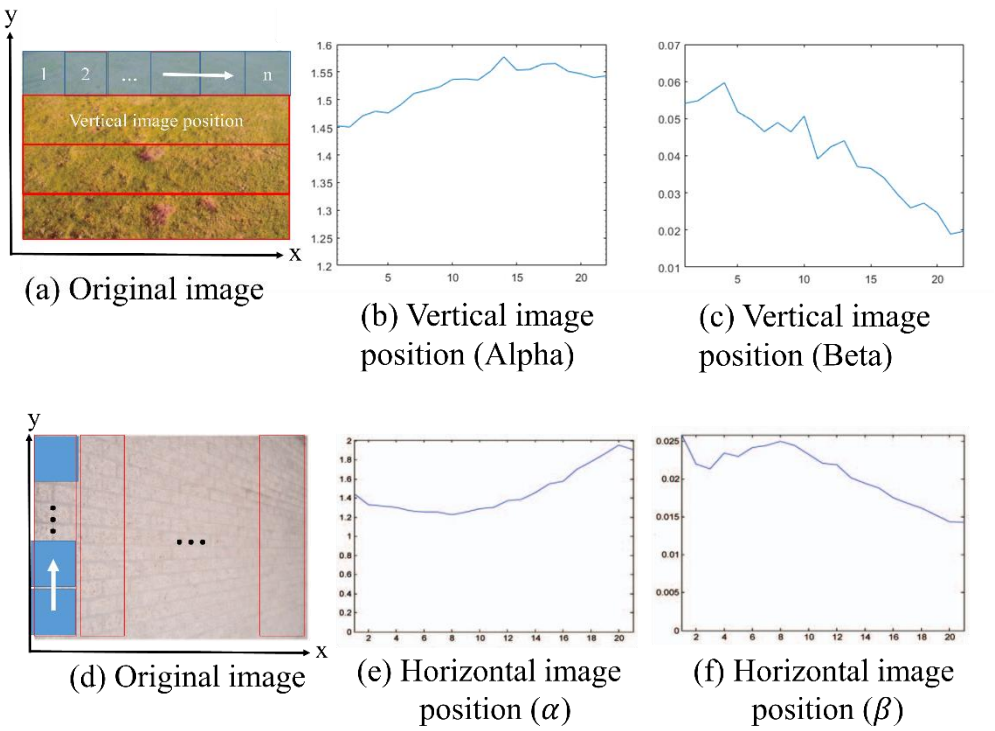


Figure D.1: Weibull paramters values (for y-derivative) as a function of depth for texture of grassy and brick surfaces (Nedovic et al., 2010). (a,d) are original input images. (b,c) and (e,f) are graphs of parameters of Weibull distribution for vertical and horizontal image positions. α increases with depth, whereas β decreases slightly with the depth of the image.

Appendix E: 1D and 2D Kernel Using First Derivative

We can obtain the 1D kernel by using first derivative along x or y-axis, (Taylor series expansion) (Milton Abramowitz & Stegun, 1972).

The forward difference of $f(x)$ is following: A function $f(x)$ can be evaluated at values that lie to right of x. In calculus, we have continuous valued function but with images we have discrete pixel values. The first derivative (un-centered difference) of discrete data is defined in equation (E.1). The un-centered difference can be obtained by h =1 as following (Dalal & Triggs, 2005) and output kernel can be used as convolutional filter with an input image.

$$\frac{df}{dx} = f'_x = \lim_{h->0} \frac{f(x+h) - f(x)}{h} \approx f(x+1) - f(x), h = 1 \rightarrow Mask : \frac{1}{2}[-1,1]. \quad \text{(E.1)}$$
$$(not - centered - at - x)$$

For central-difference, a function $f(x)$ can be evaluated at the values which exist left and right of the x. The formula will show that the abscissas that are selected symmetrically on both sides. It is defined as:

$$\frac{df}{dx} = \lim_{h->0} \frac{f(x+h) - f(x-h)}{2h} \approx \frac{f(x+1) - f(x-1)}{2}, h = 1, Mask : \frac{1}{2}[-1,0,1](centered - at - x) \quad \text{(E.2)}$$

For 2D kernel, it can be defined as:

$$\frac{df}{dx} = \lim_{h->0} \frac{f(x+h,y) - f(x-h,y)}{2h} \approx \frac{f(x+1,y) - f(x-1,y)}{2}, h = 1 \quad \text{(E.3)}$$

Note: Often the division by 2 is ignored to save the computation time, resultant matrix is scaled estimates (Gonzalez et al., 2003; Linda G. Shapiro & Stockman).

Appendix F: Matlab Code of Template-Based Segmentation

```
function Feature_Vector= Feature_extraction_from_each_template (Tempi, img, F)
%% input:
% tempi indicates input template having same size of img. e.g. template (a) from Figure 2.8
% input image: img RGB image for feature extraction
% F contains all the parameters, including nxn patch size (e.g. 4x4), parameters of features etc.
%% Output
% It return M that contains full set of features from an image which we
% obtain by follwoing this template, it is next step of our algorithm
Seg_set = unique(Tempi);    % find components of template
       check=size(Seg_set); %
    if(F.temp==6)          % if it is 6th template as it did not use segmentation see figure 2.8, we apply feature extraction directly
        Feature_Vector = feature_extraction(Tempi,img,F); % feature extraction step (next step of algorithm)
    else
        global_mask = zeros(size(Tempi));
       for k=1:size(Seg_set)
         Image_mask  = zeros(size(Tempi));
          Image_mask(Tempi  == Seg_set(k)) = 1;
          bw=activecontour(rgb2gray(img),Image_mask, 'Chan-vese'); % apply active contour algorithm
               for kk=1: numel(global_mask(:,1))
                  for ll=1: numel(global_mask(1,:))
                    if (global_mask(kk,ll)==0)&&(bw(kk,ll) == 1)
                       global_mask(kk,ll) =k;
                    end
                  end
               end
             [min_x,min_y, max_x, max_y]=find_area(global_mask); % it is finding center points(x,y)
              F.midx(k)=(min_x+max_x)/2;
              F.midy(k)=(min_y+max_y)/2;
       end
         %% call function for finding distance which near to it using Euclidian algorithm
         global_mask=merg_small_holes_into_neighbour(global_mask,F,k);
         Feature_Vector = feature_extraction(global_mask, img, F);%extract features for segmented image(next step)
    end
end
 function global_mask=merg_small_holes_into_neighbour(global_mask,F,k)
 %% merg small holes or some pixels into nearest segment  and return updated segment
     [r,c]= size(global_mask);
     for i=1:r
        for j=1:c
          if(global_mask(i,j)==0)
           index=  find_ecludean_max(F,i,j,k);%% apply Euclidean distance
            global_mask(i,j)=index;
         end
        end
    end
 end
function index= find_ecludean_max(F, i, j,k) % definition of Euclidean function
     temp=inf;
    for m=1:k
      dist = sqrt( (F.midx(m)-i)^2 + (F.midy(m)-j)^2 );
       if(dist<temp)
       temp=dist;
       index=m;
       end
   end
end
```

Appendix G: Matlab Code of Feature Combination from Segments

```
function combine_features= feature_extraction(Img_mask, img, F)
% input:
%take img_mask (segmented template), img (RGB image), and F structure which contains
%% patch size (NxN e.g. N=4) parameters of features (e.g., HOG parameters(e value, norm 2, No. of bins, degrees:180/360)
%%extraction. return one_segment_features, which has feature vector with
% Output: combine_features: set of features for whole image
%length 400 for each input 'img' image (color features 6,
  [Row,Col,K]=size(img); % row and column and K represent color channels
  % Possible set of segments from templates. We have at most 6 segments,
  % which are manually assigned.
  segment1= [];  segment2= [];segment3= [];segment4= [];segment5= [];segment6= [];
 Row_chunk=floor(Row/(F.M)); % nxn patches: x-axies
 col_chunk=floor(Col/(F.N)); % nxn patches: y-axies
 F.Row_chunk=Row_chunk;
 F.col_chunk=col_chunk;

 %% step_x and step_y are nxn patch to use for getting the features.
    for m=1:Row_chunk:(Row- (Row_chunk-1))    %%% row-m
       for n=1:col_chunk:(Col - (col_chunk-1))   %%col-n
          patch=img(m:m+(Row_chunk-1), n:n+(col_chunk-1),:);% generate patch
                  for cordi=1:Row_chunk
                    for cordj=1:col_chunk
                        Y(cordi,cordj)=m+cordi-1; % patch x coordinate yq(cordi,cordj)=m+cordi;
                        X(cordi,cordj)=n+cordj-1;%patch y cordinate
                     end
                  end
              [segment_number]= patch_isinside( X, Y,Img_mask);% it return which segment part is greater among the region
               if(segment_number==1)
                   segment1= cat(1, segment1,feature_for_segment(patch,F));
               end
               if(segment_number==2)
                   segment2= cat(1, segment2,feature_for_segment(patch,F));
               end
               if(segment_number==3)
                   segment3= cat(1, segment3,feature_for_segment(patch,F));
               end
               if(segment_number==4)
                   segment4= cat(1, segment4,feature_for_segment(patch,F));
               end
               if(segment_number==5)
                   segment5= cat(1, segment5,feature_for_segment(patch,F));
               end
               if(segment_number==6)
                   segment6= cat(1, segment6,feature_for_segment(patch,F));
               end
      end % end step colmn
    end % end steprows
combine_features=cat(1, segment1,segment2,segment3,segment4,segment5,segment6);  % combine features from segments
end % end function
```

Appendix H: Matlab Code of Feature Extraction

```
function [features]=feature_for_segment(patch, F)
% - Input: Image patch (3D array of size NxM),
% - F is structure same as previouse function, contains list of parameters of features extraction
% - It calculates HOG, Weibull parameters, COLOR:RGB and HSV, and LBP-E features
% - Output: features of 1D float_type array with size of 25 features)
%concatenate with: [R,G,B,H,S,V, Weilbul_parameters, LBP, hog features].
% - Where R, G, B and H,S,V contain mean value of R,G, B (3) and H,S,V(3) respectively.
% - Weilbul parameters contains 4 features and Hog features contains 9 features.
% - All these features are concatenated into 'features' array.
                    [hog_feature]=hog_custm(patch, F); % hog features (9 features)
                    [a1,b1,a2,b2] = weibul_fit(patch);
                     weilbulparameters= [a1,b1,a2,b2]; %shape parameters: a1,b1,
 represent alpha and beta for x direction and a2, b2, represents value
 of y-direction Gaussian derivatives
                   [wR,wG,wB,out1]=general_cc(patch,0,1,0); % k.e^(0,1,0) general_cc(input_data,njet,mink_norm,sigma,mask_im) gray world algorithm
                   [h s v]=rgb2hsv(patch);  % hsv values
                   R=mean(mean(wR)); % means of R,G and B and H,S,V values.
                   G=mean(mean(wG));
                   B=mean(mean(wB));
                   H=mean(mean(h));
                   S= mean(mean(s));
                   V= mean(mean(v));
                  binary_f_Entropy= local_binary_pattern_Entropy(patch);  % generate
lbp features and entropy value
                 features= cat(1, R, G, B, H, S, V,weilbulparameters',hog_feature, binary_f_Entropy' );
% combine feature for each patch
end       % end function
```

Appendix I: Matlab Code of Classifier Training and Testing

```
% Default function of SVM classifier
% input: train features: featuresTrain float array of NxL, N is number training images and L is length of features.
% similarly, featuresTest are testing features are
 MxL, where M is number of testing images.
% Train_labels, and Test_labels are 1xN
and 1xM integer arrays.
% CL_t is trained model, and t.Ylabel is
 predicted labels of test images.
validationPredictions is predicted score matrix of MxS dimension, S is number of classes.
 The accuracy is calculated for predicted and true labels.
t = templateSVM('KernelFunction','linear');
% templateSVM is default function: set kernel: Gaussian, linear, other useable parameter
CL_t = fitcecoc(featuresTrain,Train_labels, 'learner', t);
[t.Ylabel,t.validationPredictions ]= predict(CL_t, featuresTest);
accuracy = mean(t.Ylabel == Test_label)
 addpath('./ELM\ELM-LRF-master/');
 train = normalize(featuresTrain);
 test = normalize(featuresTest);
 mytrain= [double(string(YTrain)),double(train)];
 mytest= [double(string(YTest)),double(test)];
 % call ELM function with its paramters ( e.g. 6000 neurons, C=0.03125).
[TrainingTime, TestingTime, Trainacc, TestACC, ACTUAL,PRED] =ELM(mytrain, mytest, 1,6000,'sig',0.03125) % set parameters
```

Appendix J: Matlab Code of Predict Score Fusion

```
%% load the prediction score of M testing images for each classifiers: t1-t8.
t1= param.t1.validationPredictions;
t2= param.t2.validationPredictions;
t3= param.t3.validationPredictions;
t4= param.t4.validationPredictions;
t5= param.t5.validationPredictions;
t6= param.t6.validationPredictions;
   t7= param.t7.validationPredictions;
  t8= param.t8.validationPredictions;
 imglist=numel(t1(:,1)); % get number of images
tic % start time
countNum=0;
  for i=1:imglist
  m_majr(i,:)= t1(i,:)+t2(i,:)+t3(i,:)+t4(i,:)+t5(i,:)+t6(i,:)+t7(i,:)+t8(i,:) ; % sum rule
   [value, index]= max( m_majr(i,:));
  predict(i)=index;
  if index ==param.label.label(i)
     countNum=countNum+1;
  end
 end
Accuracy_sumrule= mean(predict== param.label.label);
fprintf('Accuracy  of sum rule: %2f \n', accuracy_sumrule*100);
toc %% timer end
```

Appendix K: Matlab Code of Performance Metric Calculation

```
function [confmat, prec, recall, fscore] = prec_recall(groundtruth,prediction)
%% input: ground truth (true labels) and prediction labels of input images
% output: confusion matrix, precision, recall F-score values
if ~iscolumn(prediction) || ~iscolumn(groundtruth)
  error('input must be column vectors');
elseif length(prediction) ~= length(groundtruth)
  error('input vectors must have the same length');
end
M = confusionmat(groundtruth,prediction);
P = sum(M,1); %
N = sum(M,2); %
prec = nan(size(M,1),1);
recall = nan(size(M,1),1);
for i = 1:size(M,1)
  prec(i) = M(i,i) / P(i);
  recall(i) = M(i,i) / N(i);
end
fscore = 2 * prec .* recall ./ (prec + recall); % fscore
prec = mean(prec);   % average
recall = mean(recall);
fscore = mean(fscore);
confmat = bsxfun(@times, M, 1./(sum(M, 1)));

%% optional
  plotConfMat(confmat, {'skyBkgGnd','skyGnd',
'bkgGnd','ground','sidewalRL', 'Box','diagBKgRL',
'groundDiagBkgRL', 'Corner', 'TablePersonBkg',
'PersonBkg','noDepth', ''});
  xtickangle(45)
End
```

Appendix L: Matlab Code of Pre-Trained Deep CNN

```
 clear ;
close all;
clc
fprintf('Loading Data \n');
% % n is the number of types of images
outputFolder= fullfile('D:\PhD\Image-Classifier-by-resnet\image_new_dataset');  % load data
im = imageDatastore(outputFolder,'IncludeSubfolders',true,'LabelSource','foldernames');
% net=googlenet(); %% we can select the pretrained CNN model
% net = googlenet('Weights','places365');
%analyzeNetwork(net);
%net=vgg16();
%net= alexnet();
 net=resnet50();
net.Layers(1);
%% list end
inputSize = net.Layers(1).InputSize;
if isa(net,'SeriesNetwork')
  lgraph = layerGraph(net.Layers);
else
  lgraph = layerGraph(net);
end
[learnableLayer,classLayer]  = findLayersToReplace(lgraph);
[learnableLayer,classLayer]
numClasses = numel(categories(imdsTrain.Labels));

if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer  = fullyConnectedLayer(numClasses,  ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer  = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
end
lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);
newClassLayer  = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);
layers = lgraph.Layers;
connections = lgraph.Connections;
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain);
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
miniBatchSize =10; % set batchsize
valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);
options = trainingOptions('sgdm', ...     %% training options
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',20, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',valFrequency, ...
    'Verbose',false, ...
    'Plots','training-progress');
net = trainNetwork(augimdsTrain,lgraph,options);
[YPred,probs] = classify(net,augimdsValidation);
accuracy = mean(YPred == imdsValidation.Labels)  % accuracy and metric calculation
[confmat, prec, recall, fscore] = prec_recall(imdsValidation.Labels,YPred)
```

Appendix M: Matlab Code of CNN-SVM and ELM

```
clear ;
close all;
clc
fprintf('Loading Data \n');
  outputFolder= fullfile('D:\PJD\Image Classification by_RESNET50\RGB_New_dataset');
 im = imageDatastore(outputFolder,'IncludeSubfolders',true,'LabelSource','foldernames');
 net= resnet50;
 inputSize = net.Layers(1).InputSize;

augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain);
augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsValidation);
%layer = 'loss3-classifier';
layer = 'fc1000';
featuresTrain = activations(net,augimdsTrain,layer,'OutputAs','rows');
featuresTest = activations(net,augimdsTest,layer,'OutputAs','rows');
whos featuresTrain
% SVM implementaiton using linear kernel (as it default)
YTrain = imdsTrain.Labels;
YTest = imdsValidation.Labels;
tic,classifier = fitcecoc( normalize(featuresTrain),YTrain);  toc
tic,[YPred,prob ]= predict(classifier,normalize(featuresTest));  toc
accuracy = mean(YPred == YTest)
[confmat, prec, recall, fscore] = prec_recall(YTest,YPred)
 %% ELM implementation
 addpath('./ELM\ELM-LRF-master/');
 train = normalize(featuresTrain);
 test = normalize(featuresTest);
 mytrain= [double(string(YTrain)),double(train)];
 mytest= [double(string(YTest)),double(test)];
 % call ELM function with its paramters ( e.g. 6000 neurons, C=0.03125).
[TrainingTime, TestingTime, Trainacc, TestACC,
 ACTUAL,PRED] =ELM(mytrain, mytest, 1,6000,'sig',0.03125) % set parameters
```

Appendix N: Matlab Code of Features Fusion Method

```
function [combined_features]=combine_features(YTrainRGB,
deepfeaturesRGB,texture_feature)
 %% combine deep features +texture features
%% input: image labels:YTrainRGB, deep features, texture_features)
%%  Output: combine_features:
floating 2D array. Rows are no. of images
 and no. of columns is length of both feature vector.
combine_features= [double(YTrainRGB),
 normalize(deepfeaturesRGB), normalize(texture_feature)];
 end
```

Appendix O: Matlab Code of Weibull Feature Extraction

```
function [a1,b1,a2,b2]=weibul_fit(patch)
% input: image patch
% output: a1,b1, parameters for hx, horizontal direction
a2,b2 paramters for hy vertical direction.
 G1=fspecial('gauss',[3,3], 0.5); % sigma 0.5 with [3,3] filter
  [Gx,Gy] = gradient(G1);
% apply derivative y-direction
dy=imfilter( double(patch),Gy,'replicate','conv');
      % apply in x-direction
              dx=imfilter( double(patch),Gx,'replicate','conv');
                    dy=double(dy);
                   h=   (hist(dy, [0.15,0.5,10])+0.01)/(length(dy)+0.01);
%add 0.01 for avoiding the undefined number
                    derivativ_y=(abs(h(:)));
                   derivativ_y=  derivativ_y+0.01;
                  [parmhat, ~] = wblfit(double(derivativ_y), 0.05);
% alpha =0.05 according six stimulus (see in paper)
                    a1=parmhat(2);
                   b1= parmhat(1);
             h=  (hist(dx, [0.15,0.5,5])+0.01)/(length(dx)+0.01);
                  derivativ_y=(abs(h(:)));
                  derivativ_y= derivativ_y+0.01;
                  [parmhat, ~] = wblfit(double(derivativ_y), 0.05);
                    a2=parmhat(2);
                   b2= parmhat(1);
end
```

Appendix P: Matlab Code of Multi-layer Feature Extraction

```
function [YTrain,X0, X1,X2,X3, X4]= deepfeatures_deepnet(imdsTrain )
%input: all input images with labels //training or testing or both
% output: set of features, X1,X2,...Xn,
% n is number of different blocks or layers.
%  Xi is feature vector with dimension of Ytrain x feature length.
%e.g. 10x1000, 10 images with 1000 feature vectors.
% YTrain, labels of images corresponding to these features.
  net = resnet50();
%  net = vgg16;;
%  net = googlenet
  analyzeNetwork(net);
net.Layers(1);
inputSize = net.Layers(1).InputSize;
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain);
 augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsValidation);
%% vgg16
%   layer1='pool2';
%    layer2='pool3';
%    layer3= 'fc8';
%%googlenet
%   layer1='pool3-3x3_s2';
%    layer2='pool4-3x3_s2';
%    layer3= 'loss3-classifier';
%% resnet
layerneg = 'activation_29_relu';
layer0 = 'activation_35_relu';
layer1 = 'activation_41_relu';
layer2= 'activation_45_relu';
  layer3= 'fc1000';
X0 = activations(net,augimdsTrain,layer0);  % intermediate layer features
 X0= squeeze(mean(X0,[1  2]))'; %% generate a single vector (GAP) applying
 X1 = activations(net,augimdsTrain,layer1);
 X1= squeeze(mean(X1,[1  2]))';
 X2 = activations(net,augimdsTrain,layer2);
 X2= squeeze(mean(X2,[1  2]))';
 X3 = activations(net,augimdsTrain,layer3);
 X3= squeeze(mean(X3,[1  2]))';
 X4 = activations(net,augimdsTrain,layerneg);
 X4= squeeze(mean(X4,[1  2]))';
whos featuresTrain
YTrain = imdsTrain.Labels;
 end
```

Appendix Q: Matlab Code of Classifiers Training, Testing, and Score-Level Fusion

```
[YTrain,Xng,X0,  X1,X2,X3] = deepfeatures_deepnet(image_data);
 %% extract deep features at multi-layers CNN: RESNET,GoogLeNET
%% uncomments layers for googlenet model inside deepfeatures_deepnet  function,
extract three (pooling, FC layers) instead of 5 different layers.
addpath('deepfeatures_4x4');
[grid_basedFeature,imageid_scene]=Mainfile_features_gridpatch(image_data);
 %% extract handcrafted features
% for i=1:length(Xng)
% end
p=0.80; %%%% training and testing ratio
No_images=length(YTrain);  %% randomly  selection for training + testing
idx = randperm(No_images);
rnd_train=idx(1:round(p*No_images));
rnd_test= idx(round(p*No_images)+1: end);
T6=sortrows(localfeatureset,1) ; %% grid features
%%  ==========================using RESNET Model ==================================
%% ================ Part 3: Training + testing for googlenet ================
%%%% training and testing using SVM+ ELM,
%function: obtain_train_test_list_of_features: normalized(): takes two different features,
random index of training+testing,
% divided into train+test features while keeping  image index information into YTRAIN_1,  YTEST_1.
% function:
%% call features training for 1st layers
[train_set_1, Test_set_1, YTRAIN_1,YTEST_1]=obtain_train_test_list_of_features(YTrain,Xng,T6,rnd_train,rnd_test);
%% combine at each layer's outcome
[svmng,elmng ]=classification_processing_usingSVM_ELM(train_set_1,Test_set_1,YTRAIN_1,YTEST_1);
%% call features training for 2nd layers
[train_set_1, Test_set_1, YTRAIN_1,YTEST_1]=obtain_train_test_list_of_features(YTrain,X0,T6,rnd_train,rnd_test);
 %% combine at each layer's outcome
[svm0,elm0 ]=classification_processing_usingSVM_ELM(train_set_1,Test_set_1,YTRAIN_1,YTEST_1);
%% call features training for 3rd layers
[train_set_1, Test_set_1, YTRAIN_1,YTEST_1]=obtain_train_test_list_of_features(YTrain,X1,T6,rnd_train,rnd_test);
%% combine at each layer's outcome
[svm1,elm1 ]=classification_processing_usingSVM_ELM(train_set_1,Test_set_1,YTRAIN_1,YTEST_1);
%% call features training for 4rth layers
[train_set_2, Test_set_2, YTRAIN_2,YTEST_2]=obtain_train_test_list_of_features(YTrain,X2,T6,rnd_train,rnd_test);
 %% combine at each layer's outcome
[svm2,elm2 ]=classification_processing_usingSVM_ELM(train_set_2,Test_set_2,YTRAIN_2,YTEST_2);
%% call features training for 5th layers
[train_set_3, Test_set_3, YTRAIN_3,YTEST_3]=obtain_train_test_list_of_features(YTrain,X3,textureFeature,rnd_train,rnd_test);
%% combine at each layer's outcome
[svm3,elm3 ]=classification_processing_usingSVM_ELM(train_set_3,Test_set_3,YTRAIN_3,YTEST_3);
%% ================ Part 4: score-level fusion using Model: (a)  Model-HSF (score-level fusion)================
%% score level fusion  for svm and elm (elm only work for  majority voting )/// model (Model-HSF)
[Ssvm]=calculate_scorefusion_svmresnet(svmng,svm0,  svm1,svm2,svm3,YTEST_3);
 %% linear svm score , fusion using sum,product rule, and majority voting
%% Ssvm: return accuracy  of majorty voting,sum, product, max rules.
[Selm]=calculate_scorefusion_ELM_resnet(elmng,  elm0,elm1 ,elm2 ,elm3,YTEST_3);
 %% majority voting,  YTEST_3 has same testing images as like YTEST_1,.2...
%% Selm return accuracy  of majority voting
% Ssvm is struct contains acc, pre, recal,fscore,  confusion matrix for sum
% rule, product rule, max rule, and majorty voting. while Selm is only
% contain acc, prec, recall,  fscore and conf.matrix  for majority voting.
%% ================ Part 5: score-level fusion using Model: (b) Model HFF  (score-level fusion)================
%% feature-level fusion and elements application // Model (B)
combine all features: deep+handcrafted,
deepfeature_cmb=[Xng,  X0,X1 X2 X3];
[train_set_cmb, Test_set_cmb, YTRAIN_cmb,YTEST_cmb]=obtain_train_test_list_of_features(YTrain,deepfeature_cmb,T6,rnd_train,rnd_test);
[svmCMB,elmCMB ]=classification_processing_usingSVM_ELM(train_set_cmb,Test_set_cmb,YTRAIN_cmb,YTEST_cmb);
 %% return performance: acc, pre, recall, and f-score, Conf. matrix.
```

Appendix R: Matlab Code of Score-level Fusion Strategies

```
function [S]=calculate_scorefusion_svmresnet(svmng,svm0, svm1,svm2,svm3,actual_label)
%% input: indicating layer1, to layer 5 score respectively,
%% output: S is structure. contains sum, product, and majority voting, in terms of acc, pre, rec, fscore, and conf. matrix.
%% initialize
no_fusion_layers=5;
label=actual_label;
tng= (svmng.score);
t0= (svm0.score);
t1= (svm1.score);
t2= (svm2.score) ;
t3= (svm3.score);
imglist=numel(t1(:,1));
  predictng=[];
 predict0=[];
predict1=[];
 predict2=[];
 predict3=[];
c=0;
c1=0;
count=0;
%% max rule
for i=1:imglist
  % for j=1:clss
    [mng,ing] = max(tng(i,:));
    [m0,i0] = max(t0(i,:));
     [m1,i1] = max(t1(i,:));
     [m2,i2]= max(t2(i,:));
      [m3,i3]= max(t3(i,:));
           set_temp=-inf;
          if(m0>set_temp)
           set_temp=m0;
           index=i0;
          end
          if(mng>set_temp)
           set_temp=mng;
           index=ing;
          end

          if(m1>set_temp)
           set_temp=m1;
           index=i1;
          end
          if(m2>set_temp)
           set_temp=m2;
           index=i2;
          end
          if(m3>set_temp)
           set_temp=m3;
           index=i3;
          end
          predict1(i)=index;
   list=[svm3.labels(i), svm2.labels(i),svm1.labels(i),...
      svm0.labels(i),svmng.labels(i)];
    cap=unique(list);
   res= histc(list,cap);
    [v,indx]= max(res);
     c1(i) =cap(indx);
    if ( label(i)==c1(i))
       c=c+1;
    end
end
fprintf(' \t \t accuracy of max rule:%3f\n', mean(label==predict1'));
```

```
S.majority = mean(label==c1');
[S.major.confmat, S.major.prec, S.major.recall, S.major.fscore] = prec_recall(double(label),c1')
S.maxrule = mean(label==predict1');
[S.maxr.confmat, S.maxr.prec, S.maxr.recall, S.maxr.fscore] = prec_recall(double(label),predict1')
%% sum rule performance,
tic
count2=0;
 for i=1:imglist
      m_majr(i,:)= (tng(i,:)+t0(i,:)+t1(i,:)+t2(i,:)+t3(i,:))/no_fusion_layers   ;
    [value, index]= max(m_majr(i,:));
     predict2(i)=index;
  if index == label(i)
    count2=count2+1;
 end
 end
S.sumrule= mean(label==predict2');
fprintf('sum_rule: %2f \n', S.sumrule*100);
[S.sum.confmat, S.sum.prec, S.sum.recall, S.sum.fscore] = prec_recall(double(label),predict2')
toc
%% product rule
count3=0;
 for i=1:imglist
    m_majr(i,:)= (tng(i,:).*t0(i,:).*t1(i,:).*t2(i,:).*t3(i,:))/no_fusion_layers;
    [value, index]= max( m_majr(i,:));
     predict3(i)=index;
  if index == label(i)
    count3=count3+1;
 end
 end
S.productrule= mean(label==predict3');
fprintf('product_rule: %2f \n', S.productrule*100);
Toc
end % End of function
```

Appendix S: Raw Data of Different Experiments Generated by Matlab

Table S1: Scene geometry recognition results of different features descriptors by using 4x4 patches.

Feature Set/ Methods	**Stage Classification (12 Stages)** SVM Linear/M. Gauss/ Quad. Kernel, c=1 (Acc %).
1312f – Geom. context feature (Hoiem D. 2007)	63.1/60.3/64.7
128f – Presp. Line (P)	38.2/**42.6**/43.2
512f – Gist descriptor(Gist)	61.25/61.1/**62.8**
48f - HSV feature	49.3/52.6/**54.1**
144f – HOG features	58.5/59.2/**62.6**
32f – local binary pattern(LBP)	45.3/48.6/**49.5**
32f - Local binary pattern R=2(LBP2)	44.2/ 47.8/48.5
3072f -Fisher & pyramid (2013)	**69.4/66.2/72.4**
3379f – fisher and pyramid (2013)+ HOG +HSV+ color	**74.0**

(Continue)

Table S2: Scene geometry recognition results using different templates

Feature extraction using Lou method and applied on majority method (see Appendix R)	**Using segment parameter 20 for box hard seg.**
304f -T1 (HOG +HSV+RGB)x16	66.2/65.4/68.1
304f -T2	64.7/65/67
304f -T3	63.7/63.2/65.8
304f -T4	63.0/62.2/67.7
304f -T5	63.9/63/65.2
304f -T6	66.8/66.5/69.6
Majority voting/max rules	73.53/77.88
Max adaptive rule	78.64
Using Lou et al. metod (Lou et al. 2015) method with LBP	67.25
Implementation of Lou's feature extraction	Using hard and soft segments
608f -T1 (HOG+HSV+RGB)x16x2:32patches	**69.5**
608f -T2	**69.1**
608f -T3	**68.9**
608f -T4	**68.9**
608f -T5	**67.1**
608f -T6	**70.1**
Majority voting	**68.6/69.1**
3379f – fisher and pyramid (2013)+hog+hsv+color	**74.0**

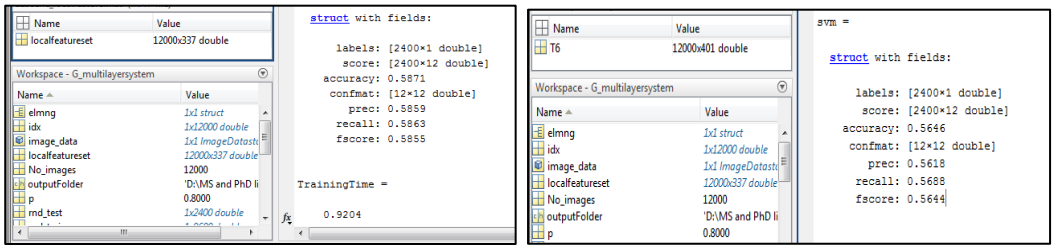
Table S3: Scene geometry recognition results of different number of features with 8x8 grid patches and template based features combination.

8x8 feature vector	Svm (kernel : linear/Gaussian/quadratic) Acc.%
256f- Weibull distribution (T)	51.5/53.3/53.9
128f- LBP	43.9/48.6/47.7
512f- Line_pres (P).	44.8/45.7/47.3
320f-Atm. Scattering. (A)	49.1/49.5/51.0
576f - HOG	57.6/57.9 /61.5
192f - HSV	49.7/53.3/53.6
4x4 version	Linear/Quadratic/Gaussian/cubic
64f - Weibul Dist. (T)	49.7/**53.7**/51.2
81f – Atm. Scattering (A)	48.6/**50.9**/50.6
128f – Pres. Line(P)	43.4/46.1/46.1/**46.8**
144f- HOG	58.9/**62.3**/59.6/**62.6**
48f- HSV	48.6/**54.3**/52.9
96f - LBP-E(5 LBP bins, 1 entropy)	45.5/**48.9**/**48.9**
160f- LBP+T	58.3/61.1/60.6
144f - T+A	55.9/**60.2**/58.2
209f - P+A	54.3/**57.2**/54.5
512f - Gist	61.6/ **63.7**/61.0
208f - T+HOG	60.1/**65.1**/61.5/65.5
288f - HOG+Color+T	62.7/**66.7**/64.4/**67**.0
800f – HOG+Color+T+Gist	63.9/**68.2**/ 64.8/**69.5**
848F- HOG+A+T+HSV +Gist	64.9/**69.1**/ 65.1/
336f – HOG+A+HSV+T	64.5/**68.1**/65.6/68.4
880F- HOG+COLOR+T+HSV+LBP + Gist	65.3/**69.8**/65.3
HOG+HSV+RGB+W+LBP-E features from each template T	SVM kernel: Linear/Q/Gauss/Cubic
400f- T(a)	66.6/68.3/66.2
400f- T(b)	65.2/67.3/66.6
400f- T(c)	63.4/65.7/62.3
400f- T(d)	64.1/66.0/63.3

400f- T(e)	64.4/65.7/64
400f- T(f)	66.7/69.5/66.4
400f- T(g)	65.2/67.5/ 65.4/67.8
400f- T(h)	65.5/**67.6**/65.7/**68.3**
Majority voting using up to T8 templates/max rules	76.63/81.18
Majority voting using up to T6 templates/max rules	73.45/77.72
Majority voting using up to T8 templates/sum-rules	81.15/82.495

Table S4: The experiments on 'stage dataset 2'. Linear SVM is used with 80% images for training and 20% for testing.

Feature (f) set extracted by 4x4 patches	**Acc./Pr/Re/F-score (%)**
320f – HOG+HSV+saturation var.+ RGB+Weibull features	58.71/58.59/ 58.68/58.55
400f – HOG+HSV+ RGB+Weibull features+LBP-E	56.46/56.18/56.88/56.44



(a) (b)

Figure S1. (a) indicates the results of 'HOG+HSV+Saturation variance+Weibull distr.' features. (b) represents the results of 'HOG+HSV+Weibull distr.+LBP-E' features.

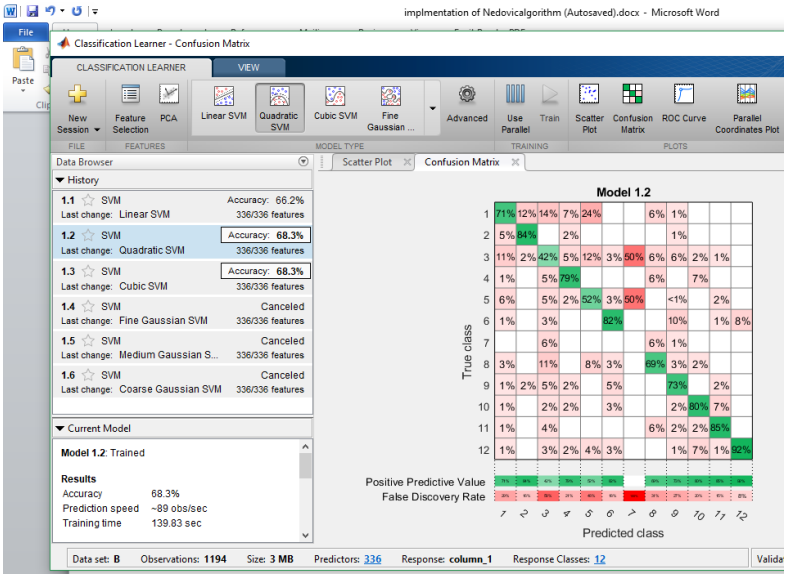


Figure S2, T1 confusion matrix and Accuracy.

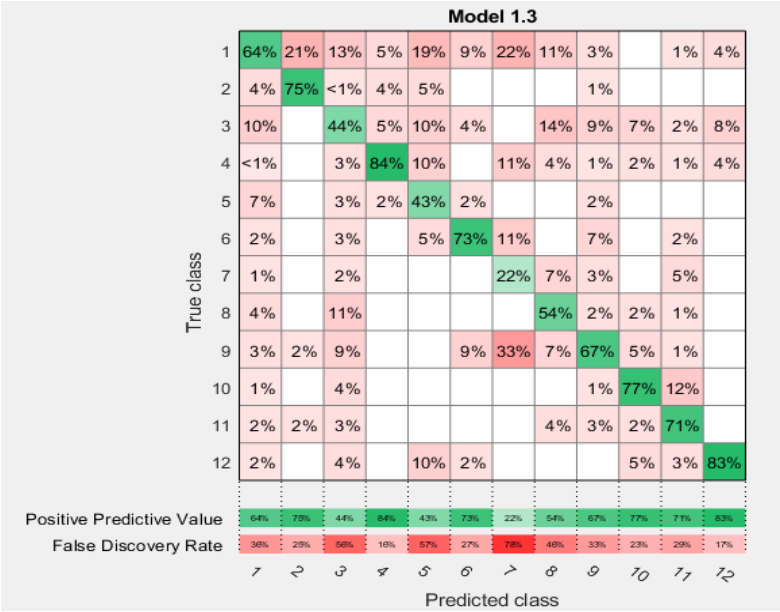


Figure S3. Confusion matrix of Gist feature on 1209 images.

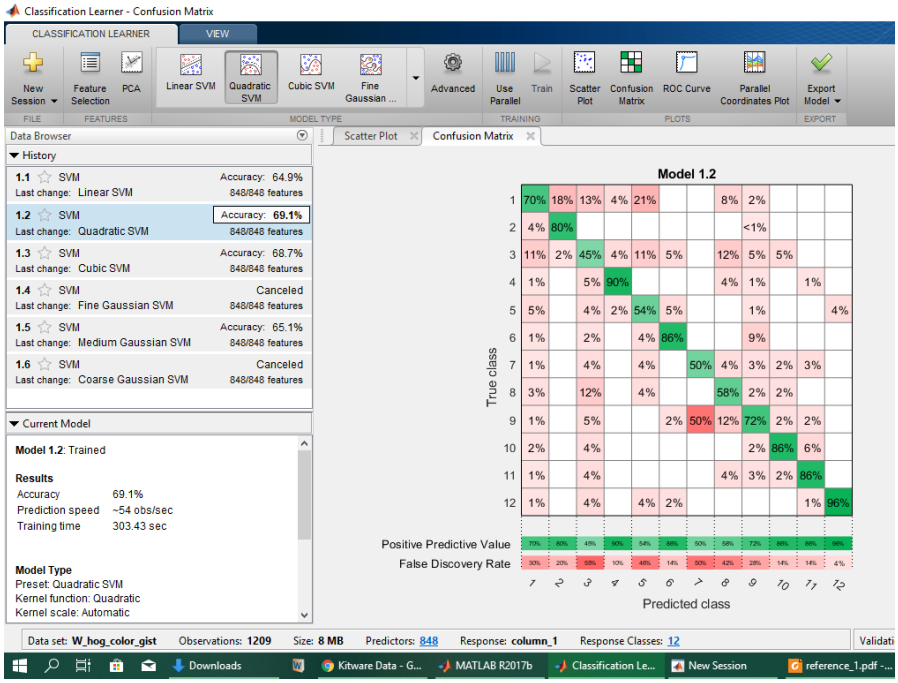


Figure S4: hog_T_Color_Hsv_gist feature accuracy and confusion matrix

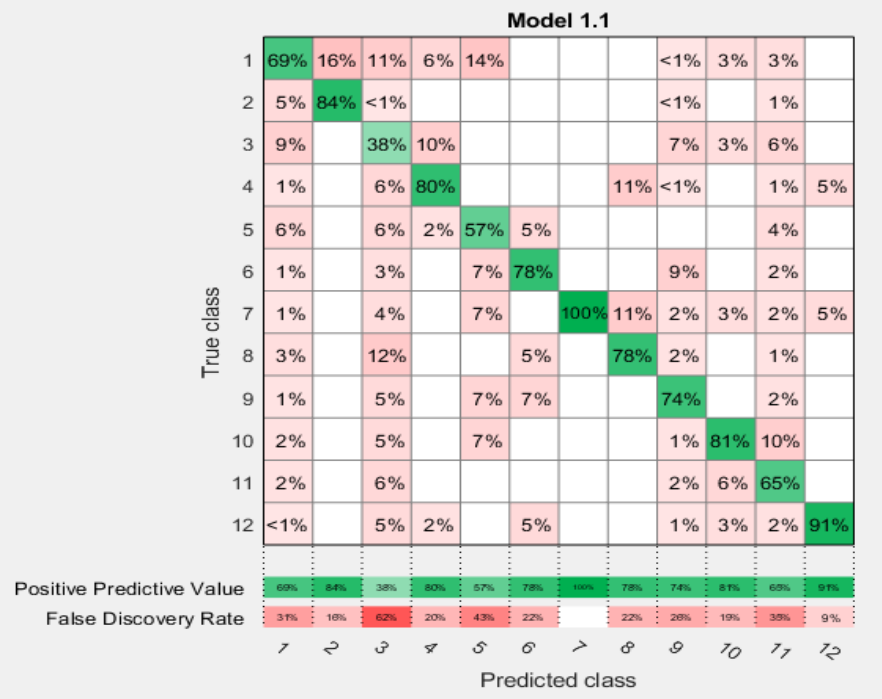


Figure S5: Confusion matrix of Hog+color+LBP feature set (304f) over 12 stages dataset 1

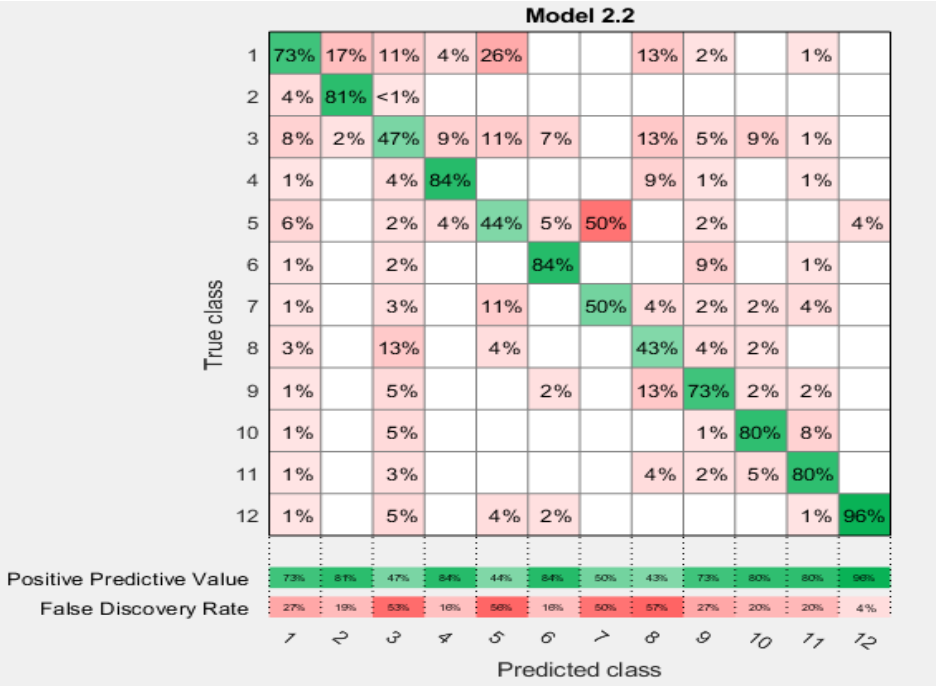


Figure S 6: hog+color+LBP+gist (816f) feature set and confusion matrix.

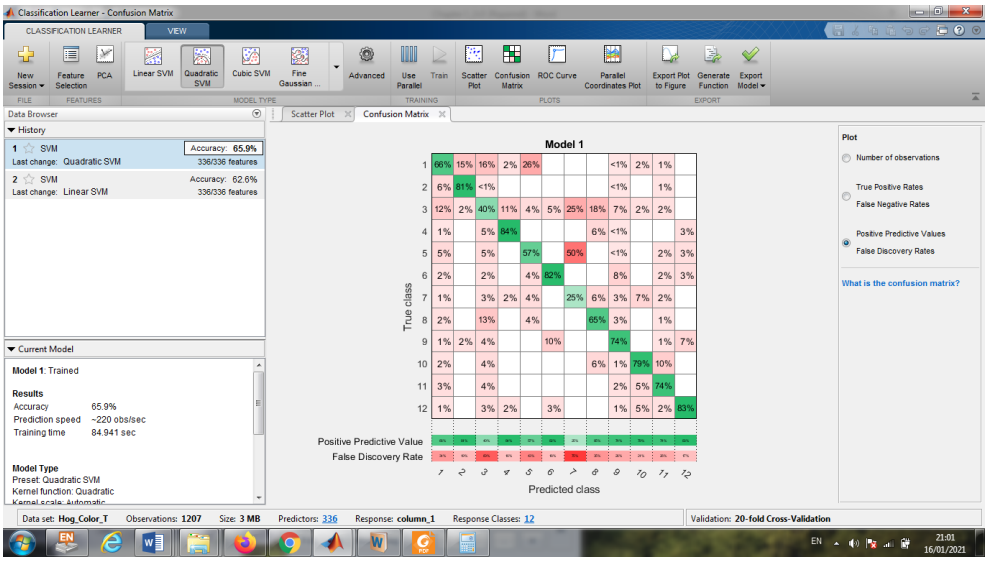


Figure S 7: hog+color+T confusion matrix and accuracy.

Appendix T: Predicting Accuracy Using Sum-rule

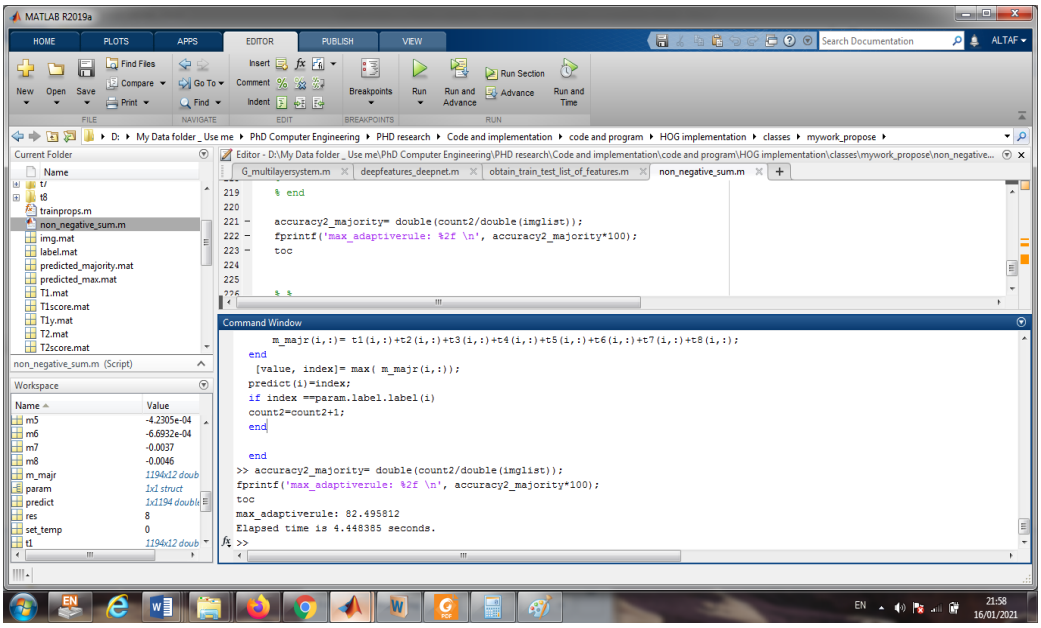
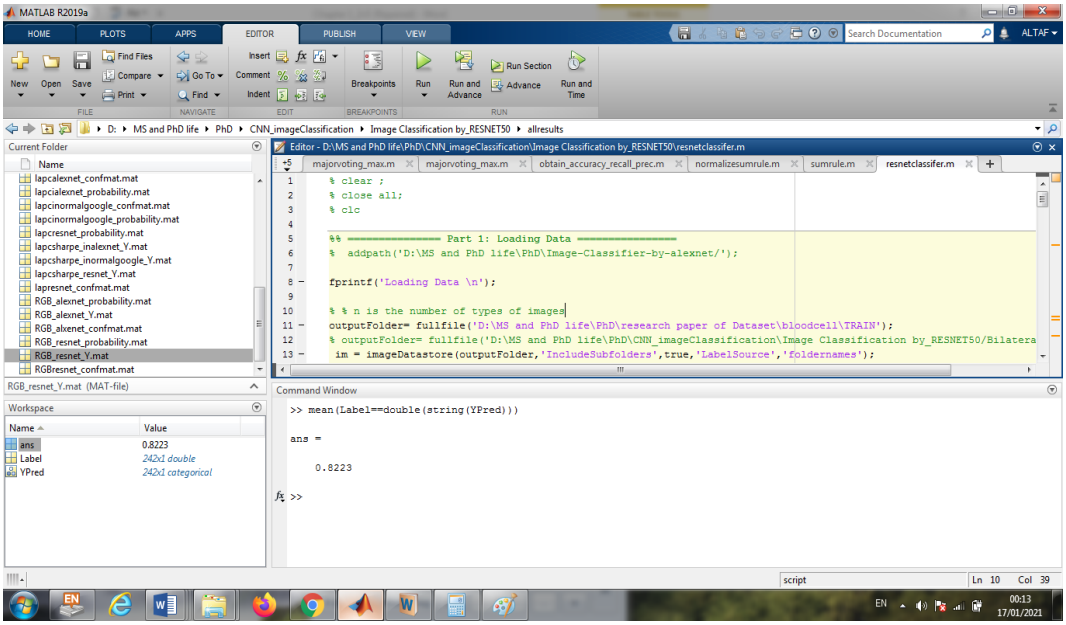


Figure T1: Predicting accuracy of Segmentation-based feature extraction method on stage dataset 1.

Appendix U: Predicting Accuracy Using CNN Models on Stage Dataset 1

The result of ResNet 50 is shown. Other methods results are calculated which are in form of structure, listed in left side of the window. We show these results in Table 3.4.



Appendix V: Predicting Accuracy Using Sum-rule on 15 Scene Dataset

Results are obtained from Matlab structure. The screenshot is shown for sum-rule method corresponding results are given in Table 3.5. Similarly, max rule is used here.

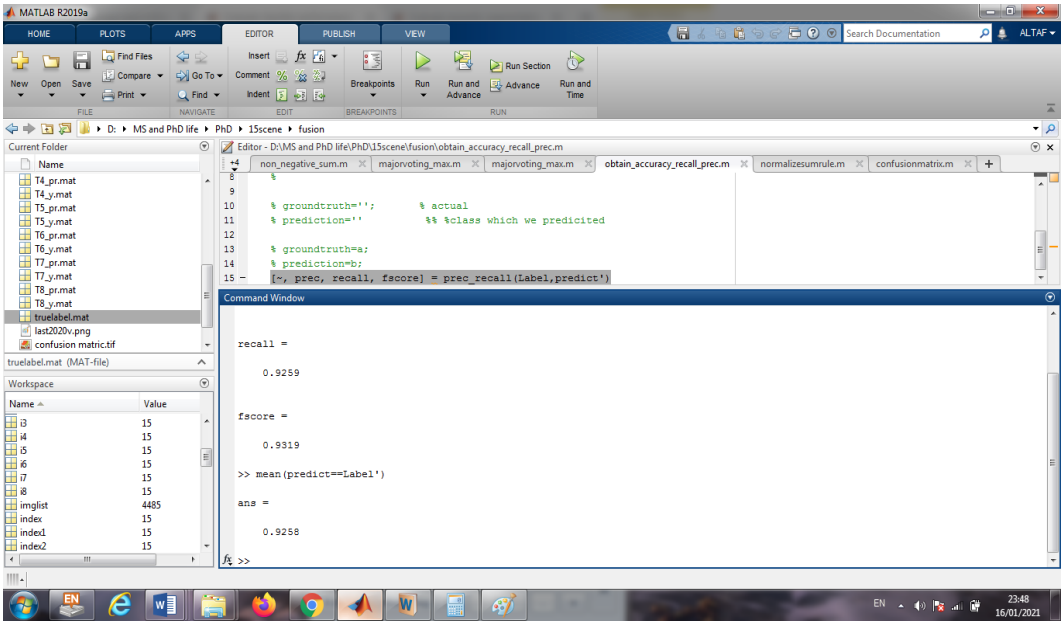


Figure V1: Predicted results using Sum-rule on 15 scene dataset.

Appendix W: Experiment on Stage Dataset 2

The deep features are extracted using Appendix M, and texture gradient features (Weibull parameters) are extracted using Appendix O, and are combined with deep feature using Function given in Appendix N. Output is shown below (Figure W1). The standard CNN models, AlexNet, ResNet, GoogLeNet, and VGG-16 are trained using Appendix L and their processing is shown below (Figure W2-W5). The perform metric is calculated by Appendix K. The 9600 images of stage dataset 2 are used for training and 2400 images for testing.

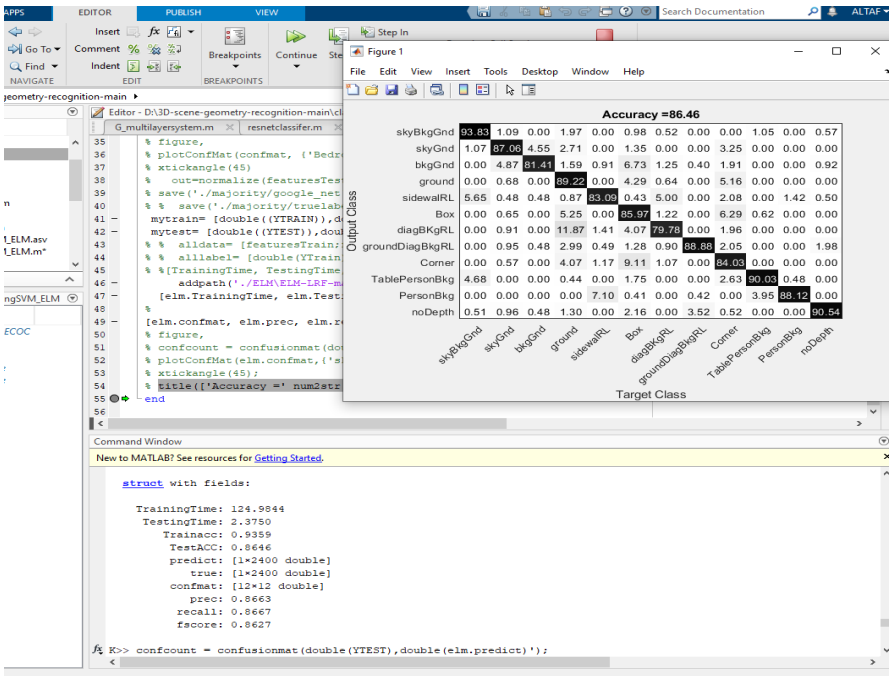


Figure W1: Combining the Texture gradient features and Deep CNN features. Results is shown the one of the iteration. The Accuracy, precision, recall, and F-score is calculated using Appendix K.

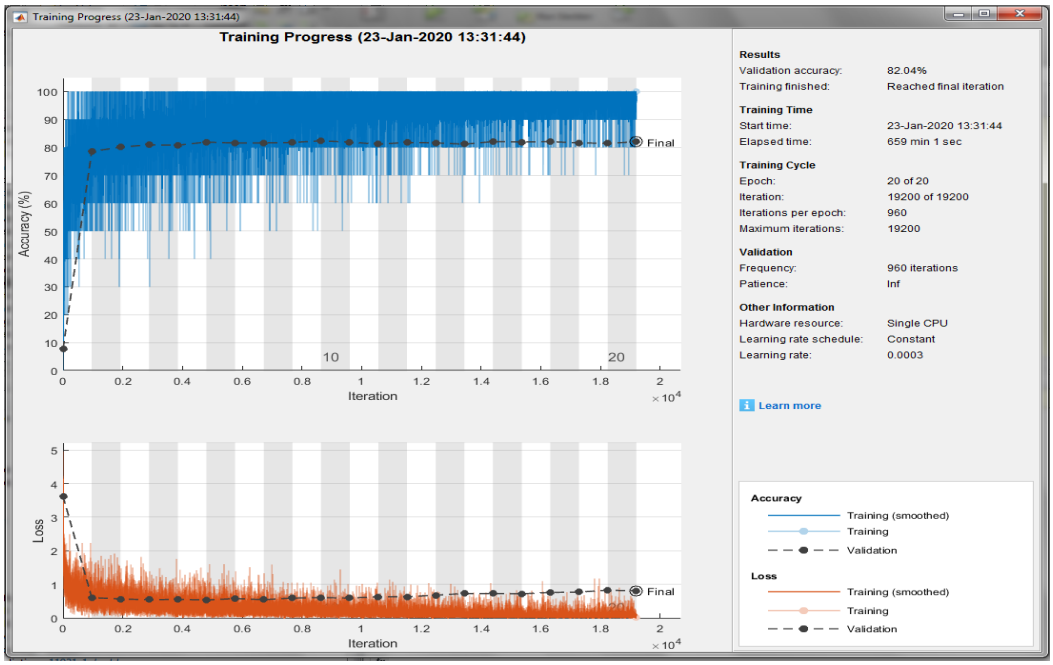


Figure W2: GoogLeNet architecture using back propagation with stochastic Gradient descent algorithm. Epoch=20.

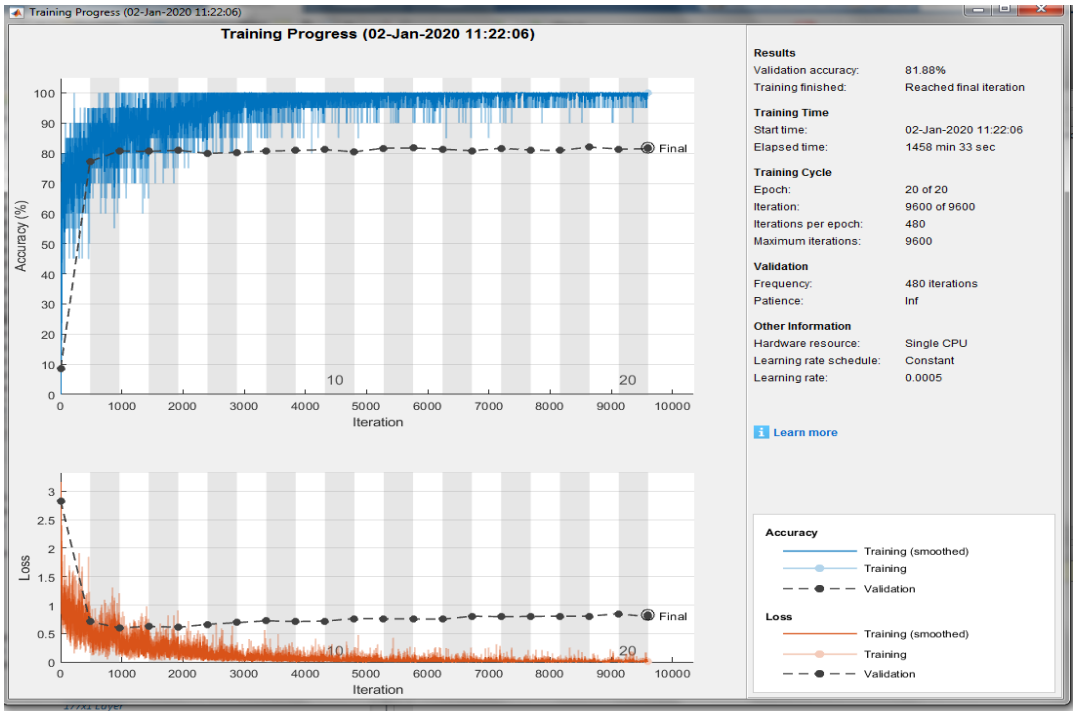


Figure W3: RESNET-50, Training and testing using back propagation with stochastic Gradient descent algorithm. Epoch=20.

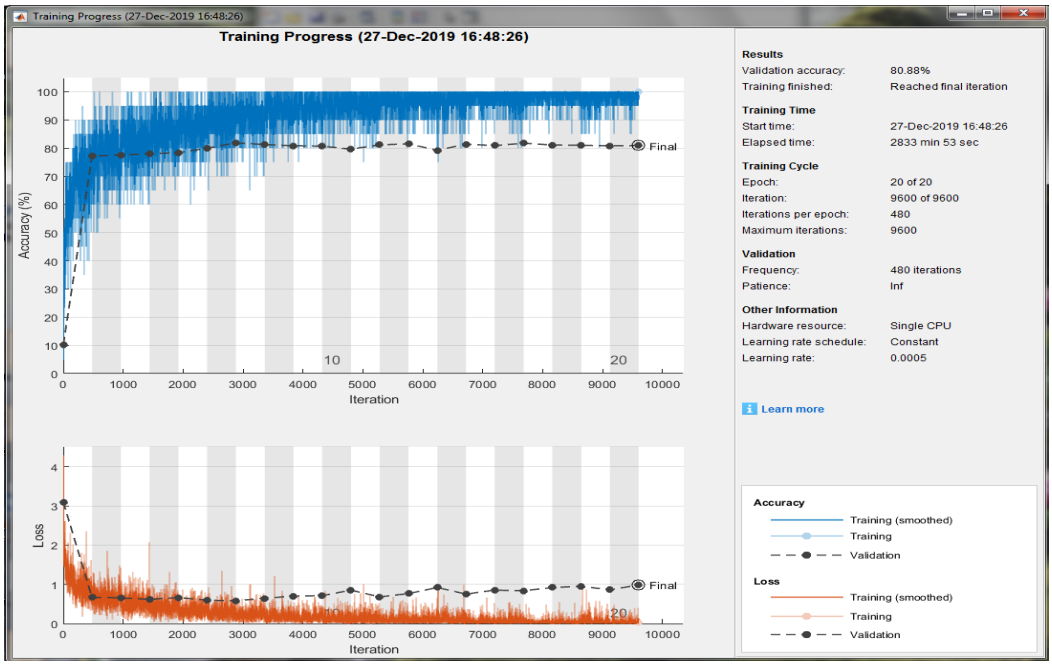


Figure S4: Vgg-16 architecture using back propagation with stochastic Gradient descent algorithm. Epoch=20

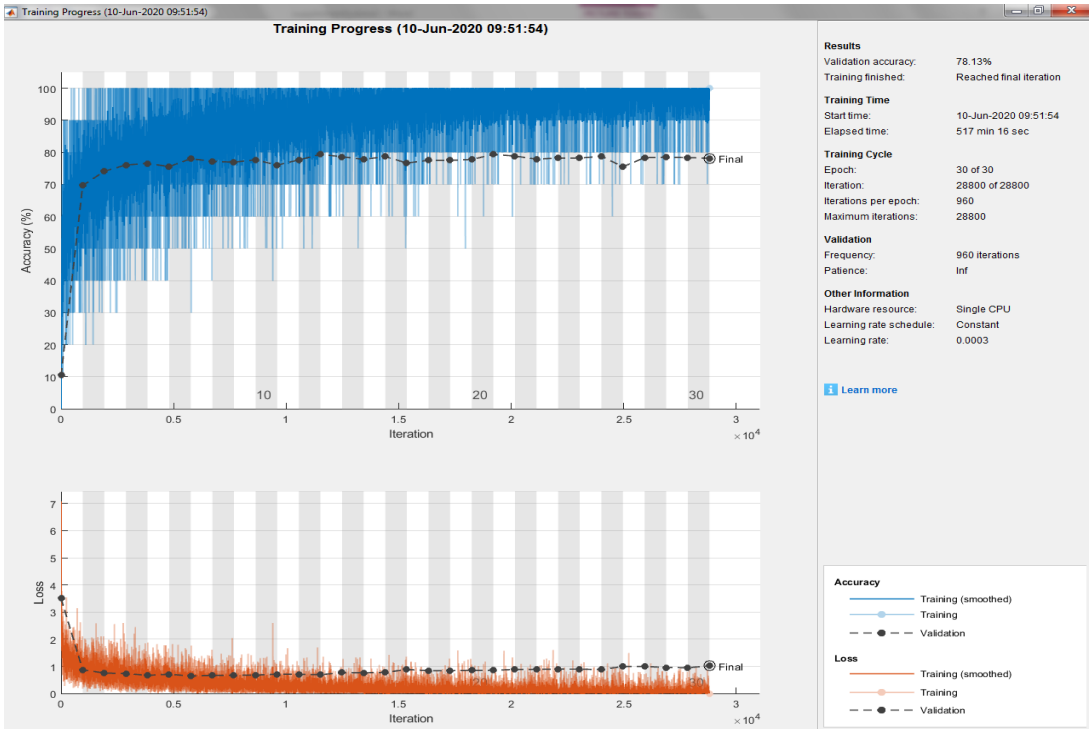


Figure S5: AlexNet architecture using back propagation with stochastic Gradient descent algorithm. Epoch=30

Appendix X: Experiment on 15-Scene Image Dataset: Influence of Handcrafted Features at Each Intermediate Layers

The results of Matlab code is saved in excel sheet. Here, we show one screenshot that contains Matlab structure, such as Ssvm and Selm are structures which contain score-level fusion results in majority voting, sum, max, and product rules. The Svm0, Svm1, Svm2, Svm3,Svm4 are contained the output of SVM classifier at each block 1-5 with their score values for ResNet and 1-3 blocks for GoogLeNet model. Similarly, Selm represents the output of ELM classifiers containing results in majority voting. For each block elm0-elm4 contain predicted values for test images. Meanwhile, feature-level fusion is also conducted and its output is shown in elmCMB, and svmCMB structures for ELM and SVM classifier, respectively.

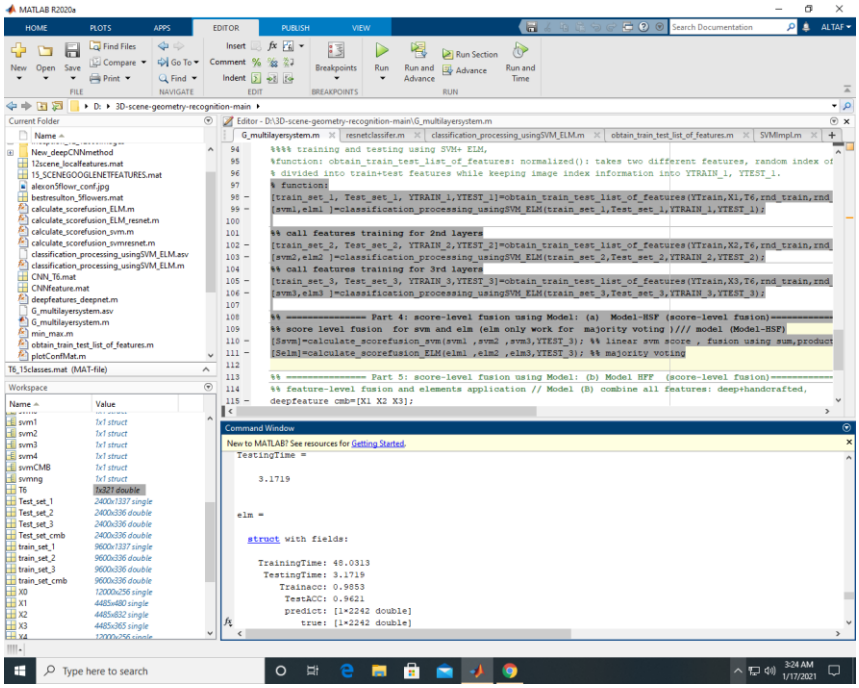


Figure X1: Screenshot of Matlab code that contain predicted data and final output.

	Without		With handcrafted	
	SVM	ELM	SVM	ELM
S-1	**90.45**	**90.45**	**95.23**	**95.9**
S-2	**93.84**	**93.62**	**96.39**	**96.92**
S-3	**93.84**	**93.09**	**97.01**	**97.28**

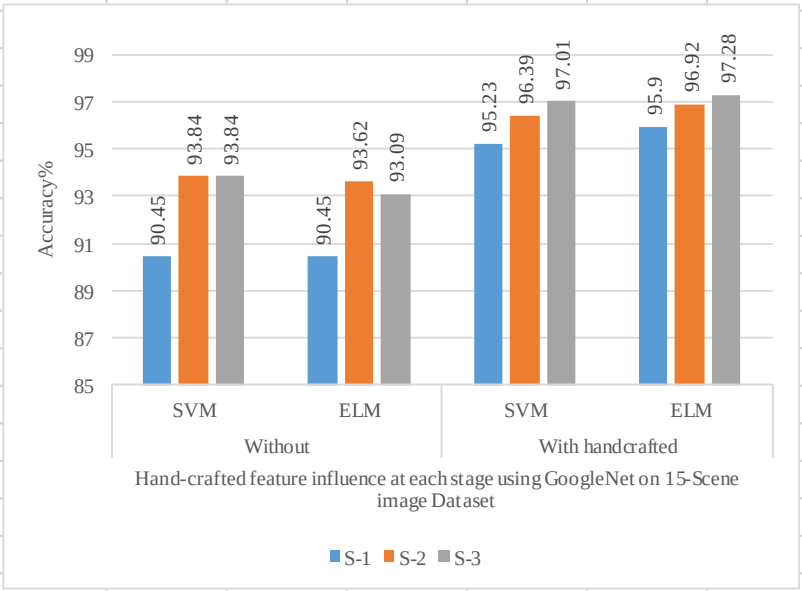


Figure X2: 15-scene Dataset result using GoogLeNet Model

Methods	Classifier	Without	With handcrafted
Majority voting	SVM	92.09	97.37
Max-rule	SVM	92.74	97.32
Product-rule	SVM	92.72	97.55
Sum-rule	SVM	92.41	97.55
Majority voting	ELM	92.18	97.41
Model-HFF	SVM	92.36	97.47
Model-HFF	ELM	92.87	97.55

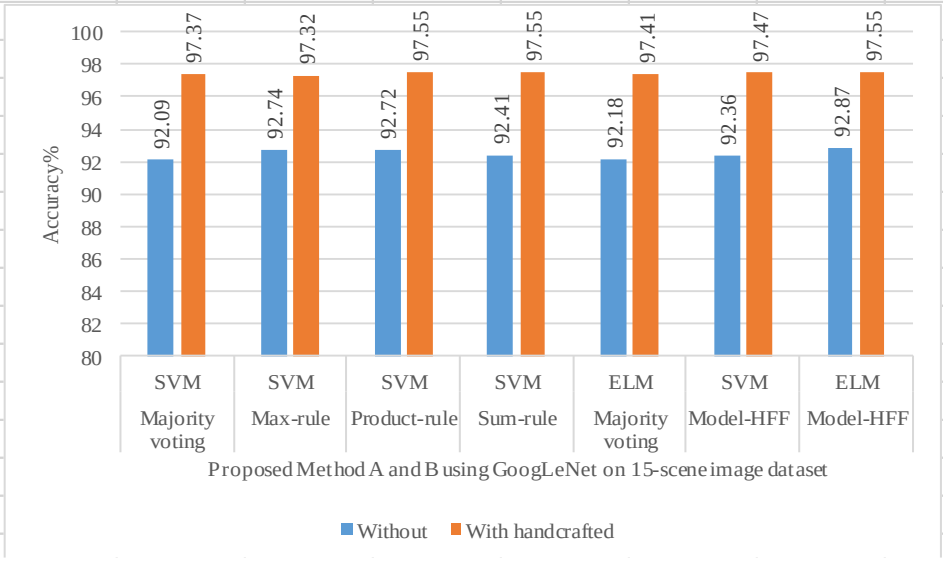


Figure X3: 15-scene dataset result using score-level (A) and feature level (B) methods shown in this Figure.

Table X1: 15 scene dataset using ResNet (with local features)ELM

S_1	92.86	92.39	92.46	92.36
S_2	93.67	93.24	93.38	93.25
S_3	95.27	94.93	94.95	94.9
S_4	95.41	95.26	95.26	95.22
S_5	95.58	95.49	95.5	95.45
Majority	96.91	96.83	96.94	96.83
Feature fusion ($S_1+..+S_5$).	96.48	96.39	96.39	96.32
15 scene dataset using ResNet (with localfeatures) with SVM classifier				
S_1	95.27	95.08	95.02	94.99
S_2	95	94.76	94.92	94.81
S_3	96.57	96.39	96.36	96.35
S_4	95.58	95.4	95.33	95.3
S_5	96.79	96.72	96.69	96.68
Majority	97.5	97.29	97.4	97.31
Max rule	97.77	97.68	97.65	97.63
Sum rule	97.86	97.8	97.83	97.79

Product rule	97.77	97.69	97.72	97.68
Feature fusion (S_1+..+S_5).	97.37	97.24	97.41	97.3
15 Without local texture features / ELM				
S_1	88.49	88.24	87.84	88.28
S_2	88.27	88.11	87.77	87.91
S_3	91.35	91.21	91.03	91.04
S_4	90.19	89.82	89.63	89.68
S_5	92.46	92.38	92.2	92.26
Majority	92.64	92.43	92.39	92.36
Feature fusion (S_1+..+S_5).	93.31	93.19	93.13	93.12
Without local texture features /SVM				
S_1	88.63	88.24	88.04	88.07
S_2	88.49	88.24	87.92	88.04
S_3	92.33	92.34	91.74	91.96
S_4	90.1	90.21	89.35	89.63
S_5	92.37	92.85	91.92	92.26
Majority	93.4	93.74	92.93	93.23
Max rule	93.98	93.94	93.61	93.73
Sum rule	93.98	94.33	93.44	93.74
Product rule	94.16	93.39	93.65	93.92
Feature fusion (S_1+..+S_5).	93.76	94.14	93.2	93.53

Table X2: 15 scene dataset using AlexNet (without handcrafted) features with Model-HSF.

Feature set	Deep features without handcrafted with Linear SVM			
	Majority voting	Max rule	Sum rule	Product rule
Combine maxpool 1 and maxpool 2	83.84	83.84	85.28	84.50
Maxpool1+maxpool2+maxpool5	87.85	87.07	88.85	87.85
Maxpool1+maxpool2+maxpool5+fc8	87.51	88.74	90.64	89.86

Table X3: 15 scene dataset using AlexNet with Model-HFF.

Feature set	SVM	ELM
maxpool1+maxpool2	84.50	83.05
Maxpooling+maxpooling2+maxpooling 5	88.96	87.96
Maxpool1+maxpool2+maxpool5+fc8	91.19	90.75
Combined all with handcrafted	95.32	93.98

Appendix Y: Experiment on Stage Dataset 2: Influence of Handcrafted Features at Each Intermediate Layers

The methods are run by Matlab and output are stored in structure. Thus, here we take those output in the tabular form. Some Screenshot are given. Data is given in tables. Detail about Matlab structure (Output of score-level and feature-level fusion) is given in Appendix X.

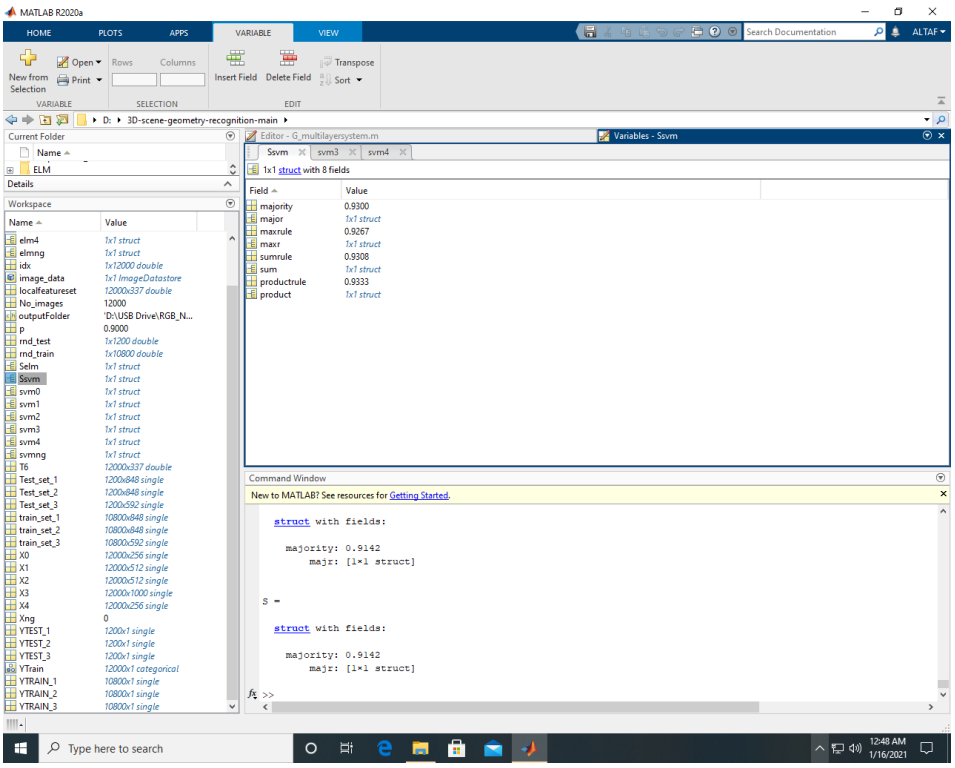


Figure Y1: An iteration of Matlab code. Ssvm: score-level combination for 5 different blocks using SVM classifier. 'stage dataset 2' is used. Left side of the window contains Matlab structure. We use them in excel sheet as our output.

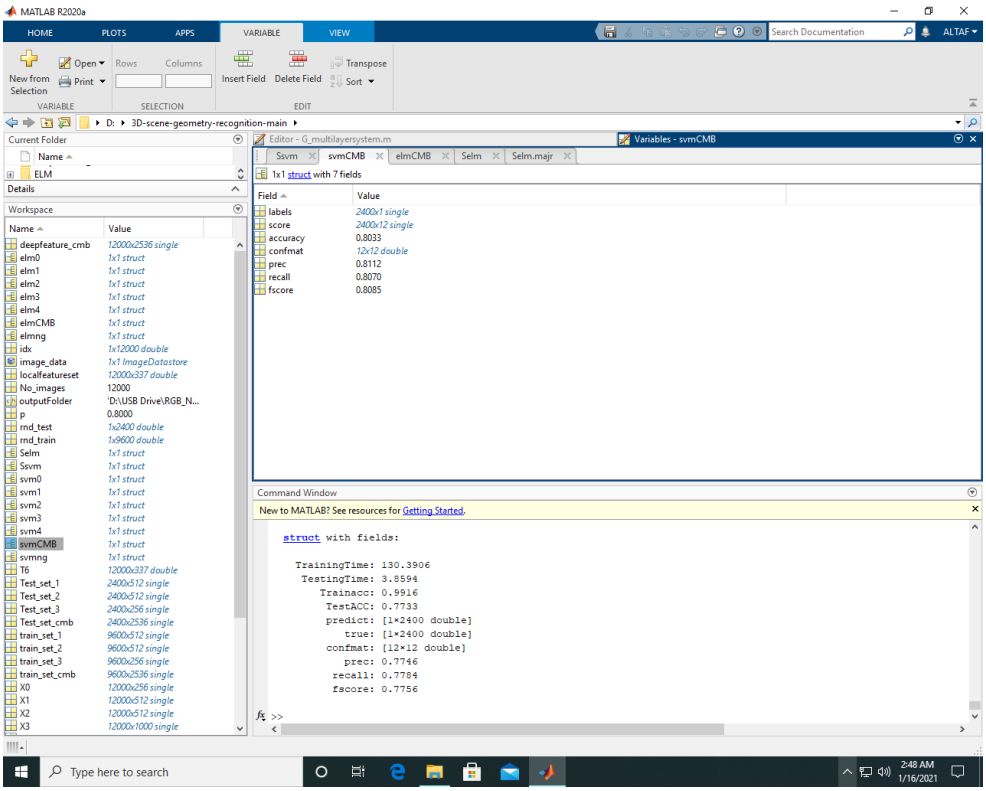


Figure Y2: An iteration for feature-level fusion using ResNet. SvmCMB: fused features given to the SVM. ‘stage dataset 2’ is used.

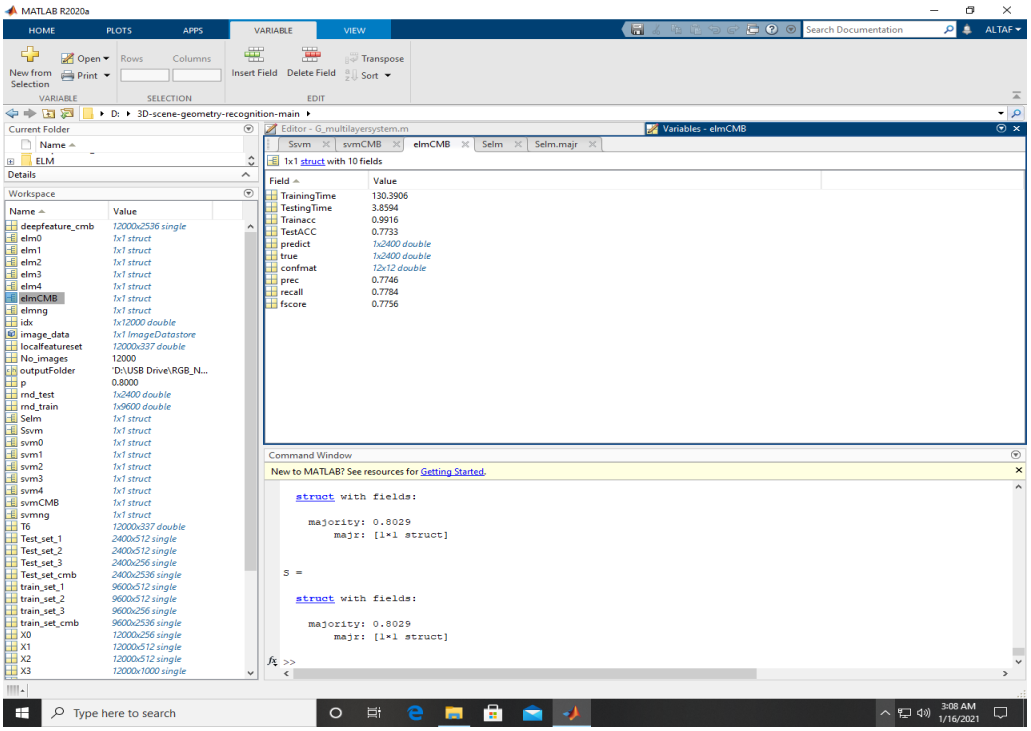


Figure Y3: ELM classifier perform using ResNet model when without handcrafted features are used of ‘stage dataset 2’.

Data in Tabular form is given below.

Table Y1: ResNet results on ‘stage dataset 2’ with and without handcrafted features influence at each block.

	ResNet result on ‘stage dataset 2’				
	For each stage with handcrafted features				
	Methods /SVM	Accuracy	Precision	Recall	F-score
	majority	92.79	92.86	92.73	92.78
	Max rule	93.54	93.47	93.36	93.4
	Sum rule	93.79	93.77	93.64	93.69
	Product rule	93.96	93.89	93.82	93.84
	Using all together	92.54	92.33	92.32	92.38
	Without handcrafted feature influence				
	majority	82	81.22	81.55	81.35
	Max rule	81.78	81.61	81.5	81.53
	Sum rule	83.25	82.69	82.8	82.7
	Product rule	82.68	82.12	82.18	82.1
	Using all together	82.5	82.28	82.05	82.1
	Without handcrafted feature				
	S_1	73.08	72.59	72.68	72.59
	S_2	74.29	74.25	73.85	73.89
	S_3	78.46	78.26	77.99	78.05
	S_4	76.25	75.93	75.77	75.81
	S_5	78.29	77.75	77.74	77.72
	Combined features with hand-crafted				
	S_1	89.58	89.49	89.37	89.41
	S_2	89.83	89.79	89.6	89.67
	S_3	92.04	92.02	91.83	91.89
	S_4	91.21	91.16	91	91.06
	S_5	90.79	90.72	90.6	90.64
	Hand-crafted features only. Using SVM and ELM				
	SVM	58.71	58.52	57.81	57.99
	ELM	58.75	59.55	58.33	59.30
ELM on ‘stage dataset 2’		with handcrafted			
	Methods	Acc	Precision	Recall	Fscore
	S_1	88.83	88.83	89	88.77
	S_2	88.88	88.94	889.05	88.887
	S_3	90.79	90.82	90.93	90.78
	S_4	89.46	89.36	89.6	89.36
	S_5	89.5	89.42	89.63	89.39
	Majority	91.88	91.87	92.01	91.83
	Using all together	89.83	89.81	89.92	89.73

	ELM without handcrafted features on ‘stage dataset 2’				
	S_1	75.29	74.71	74.42	74.72
	S_2	74.83	74.31	74.9	74.17
	S_3	78.29	77.62	78.38	77.66
	S_4	78.5	77.82	78.64	77.83
	S_5	78.25	77.56	78.35	77.6
	Majority	79.54	78.91	79.65	78.87
	Using all together	80.67	80.16	80.74	80.16

Table Y2: Results of GoogLeNet and ResNet on ‘stage dataset 2’ in tables. Model-HSF (Method A).

		Stage dataset perofrmance using score-level fusion method								
			Without				with hand-crafted features			
Architecture	Proposed Method (A)	Classifier	Acc.%	Pr. %	Re. %	F-s%	Acc.%	Pr. %	Re. %	F-s%
GoogLeNet Architecture	Majority voting	SVM	82.88	82.55	82.74	82.58	94.46	93.2	93.09	93.18
	Max-rule	SVM	81	80.78	81.05	80.89	95.04	94.99	94.87	94.91
	Sum-rule	SVM	82.08	81.61	81.71	81.62	95.13	95.09	94.99	95.02
	Product-rule	SVM	81.83	81.61	81.72	81.65	95.17	95.13	95.03	95.06
	Majority voting	ELM	82.96	82.13	82.82	82.27	91.75	91.69	91.83	91.65
ResNet Artichitecture	Majority voting	SVM	82	81.22	81.55	81.35	92.79	92.86	92.73	92.78
	Max-rule	SVM	81.78	81.61	81.5	81.53	93.54	93.47	93.36	93.4
	Sum-rule	SVM	83.25	82.69	82.8	82.7	93.79	93.77	93.64	93.69
	Product-rule	SVM	82.68	82.12	82.18	82.1	93.96	93.89	93.82	93.84
	Majority voting	ELM	79.54	78.91	79.65	78.87	91.88	91.87	92.01	91.83

all fusion into one set for googlent stage data

Method	Classifier	Accuracy%	Precision %	Recall%	F-score%
GoogLeNet (S_1+ S_1+ S_1)+hand-crafted	**SVM**	**93.96**	**93.92**	**92.87**	**93.88**
GoogLeNet (S_1+ S_1+ S_1)+hand-crafted	**ELM**	**90.42**	**90.24**	**90.49**	**90.25**

Table Y3: Results of AlexNet ‘stage dataset 2’ in Tables. Score-level fusion without handcrafted features of AlexNet model using Model-MSF with linear SVM.

Classifiers scores at intermediate stages	No. features	Majority voting	Max rule	Sum rule	Product rule
Combining ‘maxpooling1’, and ‘maxpooling2’ layers	96+256	64.38	64.75	65.75	66.58
Combining maxpooling1, maxpooling2, ‘maxpooling5’ layers	96+256+256	67.29	68.58	70.63	69.92

Combining maxpooling1, maxpooling2, maxpooling5, and Fc8	96+256+256 +1000	68.92	71.54	74.38	72.88

Table Y4: Performance of AlexNet model at each layer using Model-HFF. SVM and ELM took 61.05 sec and 123.65 sec average time, respectively.

Model-HFF	Accuracy %	
Features combination at different layers	SVM	ELM
'maxpooling1', + 'maxpooling2' layers	63.79	62.58
maxpooling1, + maxpooling2, + 'maxpooling5' layers	68.71	71.33
maxpooling1+ maxpooling2+maxpooling5+Fc8	74.04	75.29