Smart Library: Recognizing Books in a Library using Deep Learning Techniques

A B.Tech Dissertation report submitted in fulfilment of the requirements for the degree of Bachelor of Technology

by

ADARSH KUMAR JAIN (2015UCP1547),
RAVI KUMAR VERMA (2015UCP1536),
AMIT KUMAR MEENA (2015UCP1734)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY JAIPUR

May 2019

Certificate

We,

ADARSH KUMAR JAIN (2015UCP1547),

RAVI KUMAR VERMA (2015UCP1536),

AMIT KUMAR MEENA (2015UCP1734),

Declare that this thesis titled, "Smart Library: Recognizing Books in a Library using Deep Learning Techniques" and the work presented in it are our own. I confirm that:

- This project work was done wholly or mainly while in candidature for a B.Tech. degree in the department of computer science and engineering at Malaviya National Institute of Technology Jaipur (MNIT).
- Where any part of this thesis has previously been submitted for a degree or any other qualification at MNIT or any other institution, this has been clearly stated. Where we have consulted the published work of others, this is always clearly attributed.Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this Dissertation is entirely our own work.
- We have acknowledged all main sources of help.

Signed:

Date:

DR. NEETA NAIN

Associate Professor

Department of Computer Science and Engineering

Malaviya National Institute of Technology Jaipur

May 2019

Abstract

Name of the student:

ADARSH KUMAR JAIN (2015UCP1547),

RAVI KUMAR VERMA (2015UCP1536),

AMIT KUMAR MEENA (2015UCP1734)

Degree for which submitted: **B.Tech.**

Department: **Computer Science and Engineering**

Thesis title: **Smart Library: Recognizing Books in a Library using Deep Learning Techniques**

Thesis supervisor: **DR. NEETA NAIN**

Month and year of thesis submission: **May 2019**

Physical library collections are good resources for knowledge and learning. However, searching and managing books in a library often leads to tedious manual work and this process is very time consuming. Books might be missing, so the search can also be unfruitful.

Recently, deep neural architectures have been successfully applied to fields including computer vision, natural language processing, speech recognition etc, where they have produced very good results. Motivated by these, we aim to find their viability in searching and managing book.

Based on this, we investigate use of deep learning for finding books in library. We present a book retrieval and library inventory generation system. Our proposed system has the potential to greatly reduce the manual work and time required in searching and managing book inventories.

Acknowledgements

It is a matter of great pleasure and privilege for us to present our B. Tech Project report on "Smart Library: Recognizing Books in a Library using Deep Learning Techniques". First of all, we would like to express our deep sense of respect and gratitude towards our supervisor, Dr. Neeta Nain, Associate Professor, Department of Computer Science and Engineering, Malaviya National Institute of Technology Jaipur, who has been the guiding force behind this work. We want to thank her for introducing us to the field of Machine Learning and Deep Learning and giving us the opportunity to work under her. Her undivided faith in this topic and ability to bring out the best of analytical and practical skills in people has been invaluable in tough periods. Without her invaluable guidance and cooperation throughout the period, it would not have been possible for us to complete this thesis.

We also express our heartfelt gratitude towards all the teachers of Department of Computer Science and Engineering and technical staff for all their guidance and sympathetic throughout our stay at MNIT Jaipur.

We would also like to thank Prof. Udaykumar R Yaragatti, Director, Malaviya National Institute of Technology Jaipur, for providing all the institutional facilities.

Contents

List of Figures

List of Tables

Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
RFID	**R**adio **F**requency **I**dentification
RNN	**R**ecurrent **N**eural **N**etwork
WiFi	**W**ireless **F**idelity
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
SURF	**S**peeded **U**p **R**obust **F**eatures
CRNN	**C**onvolutional **R**ecurrent **N**eural **N**etwork
OCR	**O**ptical **C**haracter **R**ecognition
BLAS	**B**asic **L**inear **A**lgebra **S**ubprograms
FET	**F**ield **E**ffect **T**ransistor
ODE	**O**rdinary **D**ifferential **E**quations
API	**A**pplication **P**rogramming **I**nterface
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
FC	**F**ully **C**onnected
ReLU	**Re**ctified **L**inear **U**nit
LSTM	**L**ong **S**hort **T**erm **M**emory
PCA	**P**rincipal **C**omponent **A**nalysis
tSNE	**T**-distributed **S**tochastic **N**eighbor **E**mbedding
FCN	**F**ully **C**onvolutional **N**etwork
NMS	**N**on **M**aximum **S**uppression

Chapter 1

Introduction

Generating and maintaining an inventory of a large set of books in library is a time consuming and difficult task for both individuals and institutions. How can a person automatically generate an inventory of all the books he/she has in an efficient way, without going through the tedious process of typing in the title, authors and publisher into a computer? Similarly how can we generate an inventory of all the books currently on the bookshelves in a library without much manual work? Manual generation of the inventory is very expensive in terms of human labor and more error-prone.

Apart from generating inventory, manually searching bookshelves is also time-consuming and often not fruitful even though the book is present there. When searching books manually, we don't see many books on bookshelves and that can contain the book that we are finding. It wastes both the hard-work and the time that we spent on finding book. To solve this issue, research groups have developed many automated management systems to recognizing books on the shelves [1, 2, 3]. One way to recognize the books is to attach a physical marker on each book with an identifier such as an RFID or barcode and read the marker using a specialized reader. But this approach is very expensive and requires physically attaching a marker to each and every book.

A more cost-effective and convenient method for building and updating book inventories is desired. Modern smartphones are becoming are equipped with high-resolution cameras, various location sensors, and these phones have access to high-speed networks like WiFi. Also, there is lot of progress in deep neural networks such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). They have been applied to fields including computer vision, audio recognition, social network filtering, speech recognition, natural language processing, machine translation, bio-informatics, material inspection,

drug design, medical image analysis, and board game programs, where they have produced very good results.

To solve the problem, we propose a system using smartphones and deep neural networks to automatically segment book spines, detect and recognize text and then search the book from online databases. Our work has two goals:

1. Generating book inventory from only the images of bookshelves.
2. Help users in quickly finding the book in library.

We use smartphone to take picture of a bookshelf. The query bookshelf image is sent over a network to a server where the image is processed. Using different image processing techniques, the individual book spines are extracted and passed to the text-based and image feature-based recognition. Text-based system extracts text from the individual spines and the extracted text is used as keywords for searching book from an online database of books. Image feature-based system uses deep learning models for extracting features from the image and these features are matched to recognize book spine. Once the book is identified, additional stored information could also be retrieved, e.g. a short description of the book.

Our image features-based system outperforms the other image feature-based system [3] which uses SIFT, SURF [4] features. Our results show that text-based system achieves a performance substantially better than the image feature-based systems.

The main contributions of our work are as follows:

1. We design and implement a cost-effective system for book inventory management and searching books using smartphones.
2. Design and implementation of a book spine recognition system using image features extracted from a deep neural network which achieves great recognition accuracy compared to hand-crafted image feature-based recognition system.
3. Using convolutional neural network for detecting text on book spines.
4. For text recognition, we adopt a deep sequential labeling model based on Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) known as Convolutional Recurrent Neural Network (CRNN). We compared the performance of CRNN with the OCR text recognition.

1.1 Objective

The objective of this work is to make what was previously a tedious and time consuming experience (i.e., finding books in bookshelves) much more user-friendly. In addition, to efficiently build a large database of book collections using a cost-effective and convenient method.

Deep learning has achieved great success in both research and application. Today Deep Learning is used in autonomous vehicles, retail, personal assistance, language processing and many more. Our objective is to use the Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) in deep learning to solve the problem of manually generating book inventories in libraries and searching of a book in library.

1.2 Report Structure

In the remaining portion of the report, Chapter-2 covers the literature survey of methods related to the book recognition system and the related work done in this field. Chapter-3 explains our proposed work in detail. Chapter-4 shows the results and discussion. Chapter-5 summarizes the report and gives possible future work that can be done.

Chapter 2

Literature Survey

The literature survey is divided into five sections: Libraries and Frameworks, Image Processing, Deep Learning, Feature Extraction and the Related Work. This consist of all the topics that were looked into to get a thorough understanding of the different approaches present in the respective domains. The observations made regarding the advantages and disadvantages of different methods and comparing them with our use case provided us the key decision making points to select the best approach.

2.1 Libraries and Frameworks

A library is a collection of non-volatile resources which are used by computer programs, often for developing software. These may include help data, message templates, configuration data, documentation, pre-written code and subroutines, values, classes or type specifications.

A framework or software framework is a software application development platform. It provides a basis on which developers of software can build programs for a particular platform. For example, a framework may include predefined classes and functions for processing input, managing hardware devices, and interacting with system software.

2.1.1 NumPy

NumPy is a Python programming language library that adds support for large, multi-dimensional arrays and matrices as well as a large collection of high-level mathematical functions to work on these arrays.

Using NumPy in Python gives MATLAB-like functionality as both are interpreted and both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. Internally, for efficient linear algebra computations, both MATLAB and NumPy rely on BLAS and LAPACK.

2.1.2 SciPy

SciPy is a free, open-source Python library that is used for scientific and technical computing. SciPy includes modules for optimization, interpolation, special functions, linear algebra, integration, signal and image processing, FFT, ODE solvers, and other tasks common in science and engineering. A multidimensional array provided by the NumPy module is the basic data structure used by SciPy.

2.1.3 OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions primarily focused on real-time computer vision. OpenCV supports TensorFlow, Torch / PyTorch and Caffe's deep learning frameworks. The library has over 2,500 optimized algorithms, including a comprehensive set of computer vision and machine learning algorithms.

2.1.4 Keras

Keras is a Python-written high-level neural network API that can run on top of CNTK, TensorFlow or Theano. It has been designed to allow fast experimentation with deep neural networks, focusing on being modular, user-friendly and extensible. Keras includes numerous implementations of commonly used neural network building blocks such as layers, activation functions, objectives, optimizers and a host of tools to make it easier to work with image and text data. We can use Keras if we need a deep learning library that:

1. Allows for fast and easy prototyping (through modularity, user friendliness and extensibility).
2. Supports both convolutional neural networks and recurrent neural networks and combinations of the two.
3. Runs seamlessly on CPU and GPU.

2.2 Image Processing

Image processing is a method of converting an image into a digital form and performing certain operations on it to obtain an enhanced image or extract useful information from it. Image processing enables the use of much more complex algorithms and can therefore offer both the implementation of methods that would be impossible by analog means and more sophisticated performance in simple tasks.

2.2.1 Canny Edge Detection

Canny Edge Detection is a popular edge detection algorithm, developed by John F. Canny in 1986 [5]. The algorithm contains multiple stages. The stages are as follows:

1. **Noise Reduction**

   Since there can be noise in the image which would effect the edge detection, first step is to remove the noise in the image. One way to get rid of the noise on the image, is by applying Gaussian filter to smooth the image. To do this, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7 etc...). The size of the kernel depends on the expected effect of blurring.

2. **Finding Intensity Gradient of the Image**

   The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators.

   Smoothened image is filtered in both horizontal and vertical direction with a *Sobel* kernel to obtain the first derivative in horizontal direction and vertical direction. We can then find edge gradient and direction for each pixel from these two images as follows:

$$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

3. **Non-maximum Suppression**

   After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.

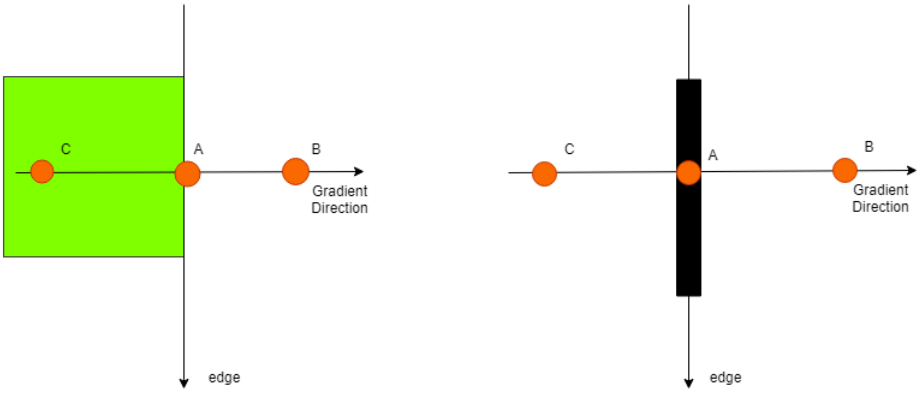

FIGURE 2.1: Non-maximum suppression

As shown in Fig. 2.1, point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero). The result is a binary image with "thin edges".

4. **Hysteresis Thresholding**

   This stage decides which edges are dominant edges and which are not. For this, we need two threshold values, $minVal$ and $maxVal$. Any edges with intensity gradient more than $maxVal$ are sure to be edges and those below $minVal$ are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

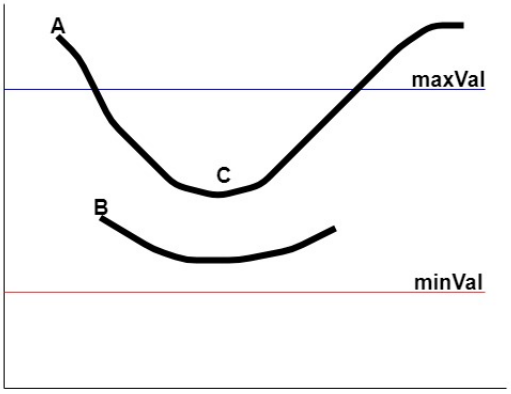

FIGURE 2.2: Hysteresis thresholding

As shown in the Fig. 2.2, edge A is above the $maxVal$, so considered as "sure-edge". Although edge C is below $maxVal$, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above $minVal$ and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select $minVal$ and $maxVal$ accordingly to get the correct result.

This stage also removes small pixels noises on the assumption that edges are long lines. So what we finally get is strong edges in the image.

2.2.2 Hough Transform

The Hough transform is a feature extraction technique used in computer vision, image analysis and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes using a voting mechanism. The voting is carried out in a parameter space, from which object candidates are obtained as local maxima in a accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

How It Works

The Hough technique is particularly useful for computing a global description of a feature(s), given (possibly noisy) local measurements. The motivating idea behind the Hough

technique for line detection is that each input measurement (e.g. coordinate point) indicates its contribution to a globally consistent solution (e.g. the physical line which gave rise to that image point).

As a simple example, consider the common problem of fitting a set of line segments to a set of discrete image points (e.g. pixel locations output from an edge detector). Figure 2.3 shows some possible solutions to this problem.

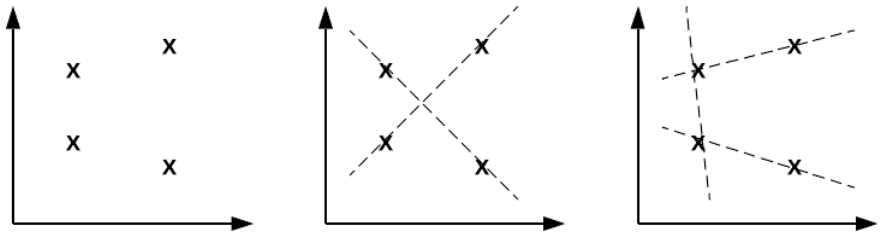

FIGURE 2.3: Coordinate points and possible straight line fittings

We can analytically describe a line segment in a number of forms. However, a convenient equation for describing a set of lines uses parametric or normal form:

$$x\ cos\theta + y\ sin\theta = r$$

where r is the length of a normal from the origin to this line and θ is the orientation of r with respect to the X-axis. This is shown in Fig. 2.4. For any point (x, y) on this line, r and θ are constant.

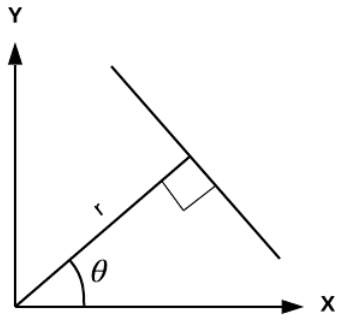

FIGURE 2.4: Parametric description of a straight line

In an image analysis context, the coordinates of the point(s) of edge segments (i.e. (x_i, y_i)) in the image are known and therefore serve as constants in the parametric line equation, while r and θ are the unknown variables we seek. If we plot the possible (r, θ) values defined by each (x_i, y_i), points in cartesian image space map to curves (i.e. sinusoids) in the polar Hough parameter space. This point-to-curve transformation is the Hough transformation for straight lines. When viewed in Hough parameter space, points which are collinear in the cartesian image space become readily apparent as they yield curves which intersect at a common (r, θ) point.

The transform is implemented by quantizing the Hough parameter space into finite intervals or accumulator cells. As the algorithm runs, each (x_i, y_i) is transformed into a discretized (r, θ) curve and the accumulator cells which lie along this curve are incremented. Resulting peaks in the accumulator array represent strong evidence that a corresponding straight line exists in the image.

2.3 Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

1. Deep learning requires large amounts of labeled data. For example, driverless car development requires millions of images and thousands of hours of video.

2. Deep learning requires substantial computing power. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

2.3.1 Neural Networks

A neural network uses network of neurons like in a brain, involving a layered structure. These are called fully connected layers. There is one input layer, one or more hidden layer and an output layer. A neural network to recognize patterns, classify data, and forecast future events because it can learn from data. Fig. 2.5 shows a neural network, which is organized in layers consisting of a set of interconnected nodes. Networks can have tens or hundreds of hidden layers.

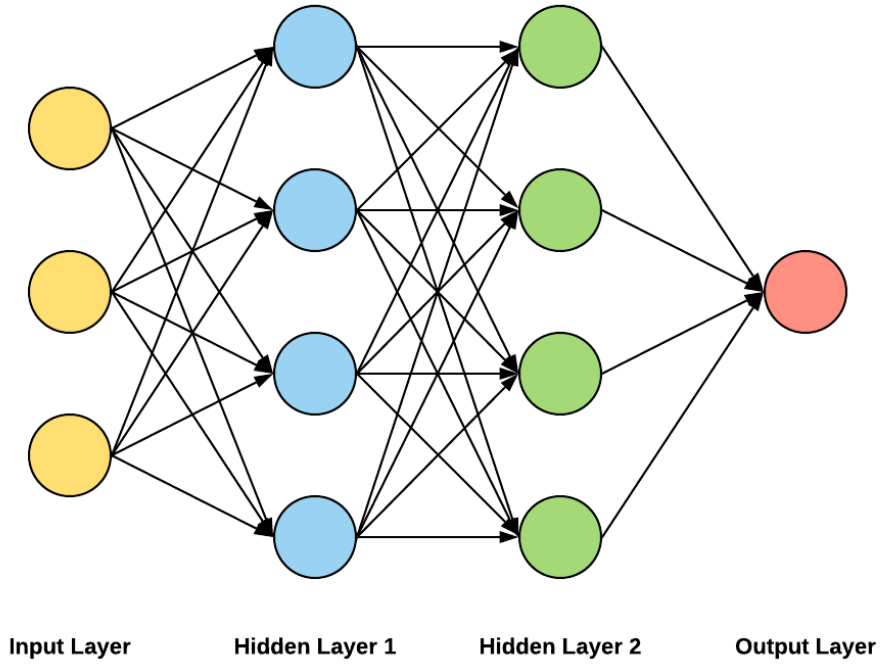

FIGURE 2.5: Artificial neural network

A neural network breaks down input into layers of abstraction. It can be trained over many data samples to recognize patterns in images, for example, just like the human brain does. Its behavior is defined by the way the individual elements are connected and by weights, of those connections. During training, these weights are automatically updated according to a specific learning rate/rule until the neural network performs the desired task correctly.

Neural networks are well suited to perform identify and classify objects or signals in speech, vision, and control systems and pattern recognition to . They can also be used for performing time-series prediction and modeling.

Neural networks that operate on two or three layers of connected neuron layers are known as shallow neural networks. Deep learning networks can have many layers, even hundreds. Both are machine learning techniques that learn directly from input data. Deep learning is especially well suited to complex identification applications such as face recognition, text translation, and voice recognition. It's also a key technology used in advanced driver assistance systems and tasks including lane classification and traffic sign recognition.

2.3.2 Convolutional Neural Networks (CNN)

One of the most popular types of deep neural networks is known as convolutional neural networks (CNN or ConvNet). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images.

CNNs eliminate the need for manual feature extraction, so you do not need to identify features used to classify images. The CNN works by extracting features directly from images. The relevant features are not pretrained; they are learned while the network trains on a collection of images. This automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification.

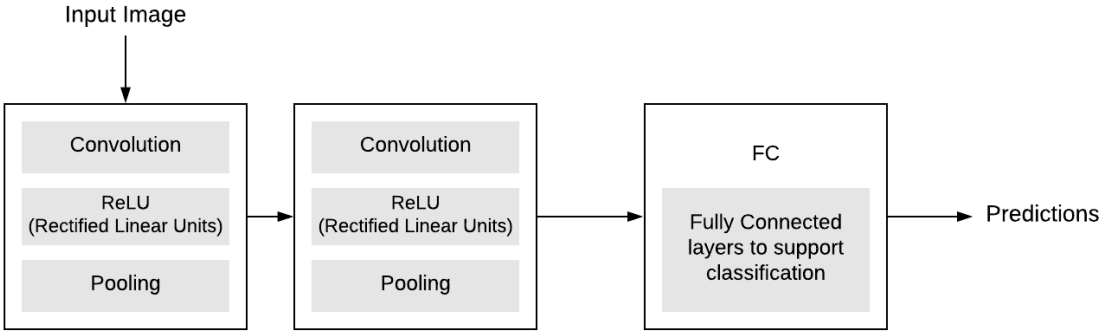

FIGURE 2.6: Layers in a convolutional neural network

Fig. 2.6 shows an example of a network. It shows three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. Convolutional layers are comprised of filters and feature maps. The pooling layers down-sample the previous layers feature map. Fully connected layers are the normal flat feed-forward neural network layer. These layers may have a non-linear activation function or a softmax activation in order to output probabilities of class predictions.

Filters are applied to each training image at different resolutions, and the output of each convolved image serves as the input to the next layer. CNNs learn to detect different features of an image using tens or hundreds of hidden layers. Every hidden layer increases the complexity of the learned image features. For example, the first hidden layer could learn how to detect edges, and the last learns how to detect more complex shapes specifically catered to the shape of the object we are trying to recognize.

2.3.3 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks and belong to the most promising algorithms out there at the moment because they are the only ones with an internal memory. Because of their internal memory, RNN's are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next.

This is the reason why they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more because they can form a much deeper understanding of a sequence and its context, compared to other algorithms. Recurrent Neural Networks produce predictive results in sequential data that other algorithms can't.

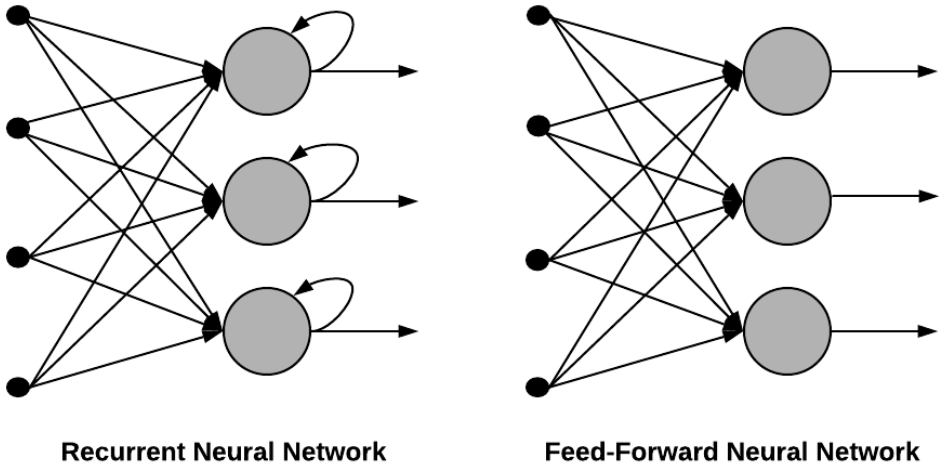

FIGURE 2.7: Comparison of RNN with other networks

In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received

previously. Fig. 2.7 illustrate the difference in the information flow between a RNN and a Feed-Forward Neural Network.

2.3.3.1 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between.

LSTM's enable RNN's to remember their inputs over a long period of time. This is because LSTM's contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory.

This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (e.g if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not.

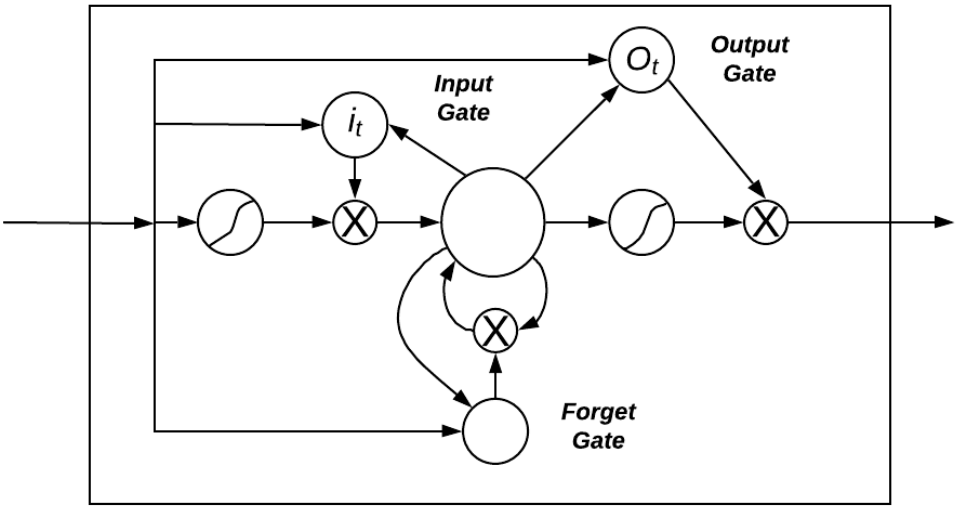

FIGURE 2.8: RNN with three gates

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it is not important (forget gate) or to let it impact the output at the current time step (output gate). Fig. 2.8 shows an illustration of a RNN with its three gates.

The gates in a LSTM are analog, in the form of sigmoids, meaning that they range from 0 to 1. The fact that they are analog, enables them to do backpropagation with it. The problematic issues of vanishing gradients is solved through LSTM because it keeps the gradients steep enough and therefore the training relatively short and the accuracy high.

2.4 Feature Extraction

A common and highly effective approach to deep learning on small image datasets is to leverage a pre-trained network. A pre-trained network is simply a saved network previously trained on a large dataset, typically on a large-scale image classification task. If this original dataset is large enough and general enough, then the spatial feature hierarchy learned by the pre-trained network can effectively act as a generic model of our visual world, and hence its features can prove useful for many different computer vision problems, even though these new problems might involve completely different classes from those of the original task.

For instance, one might train a network on ImageNet [6] (where classes are mostly animals and everyday objects) and then reuse this trained network. Such portability of learned features across different problems is a key advantage of deep learning compared to many older shallow learning approaches, and it makes deep learning very effective for small-data problems. There are two ways to leverage a pre-trained network: feature extraction and fine-tuning.

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. These features are then run through a new classifier, which is trained from scratch.

Convolutional networks used for image classification comprise two parts: they start with a series of pooling and convolution layers, and they end with a fully-connected classifier. The first part is called the "convolutional base" of the model. In the case of convolutional network, "feature extraction" will simply consist of taking the convolutional base of a previously-trained network, running the new data through it, and training a new classifier on top of the output.

The representations learned by the convolutional base are likely to be more generic and therefore more reusable. On the other end, the representations learned by the classifier will necessarily be very specific to the set of classes that the model was trained on – they

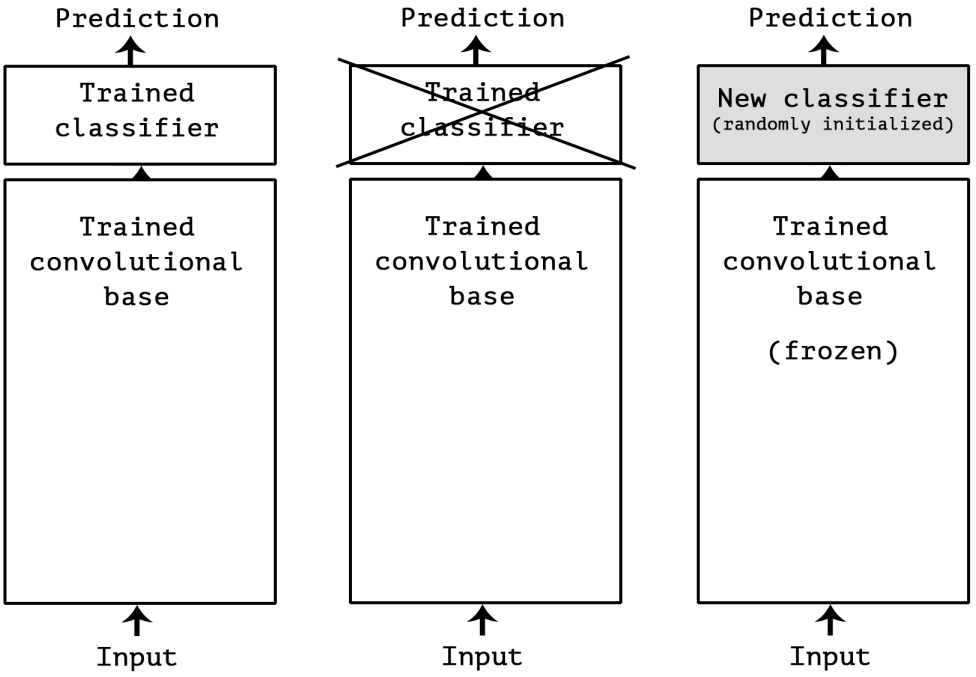

FIGURE 2.9: Using convolutional base for feature extraction

will only contain information about the presence probability of this or that class in the entire picture.

The level of generality (and therefore reusability) of the representations extracted by specific convolution layers depends on the depth of the layer in the model. Layers that come earlier in the model extract local, highly generic feature maps (such as visual edges, colors, and textures), while layers higher-up extract more abstract concepts.

Here we use four different models for feature extraction:

1. VGG16
2. VGG19
3. ResNet50
4. InceptionV3

2.4.1 InceptionV3

Inception v3 [7] is a widely-used image recognition model that has been shown to attain greater than 78.1 percent accuracy on the ImageNet dataset [6]. The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy, et. al [7].

The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Loss is computed via Softmax. Batchnorm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax.

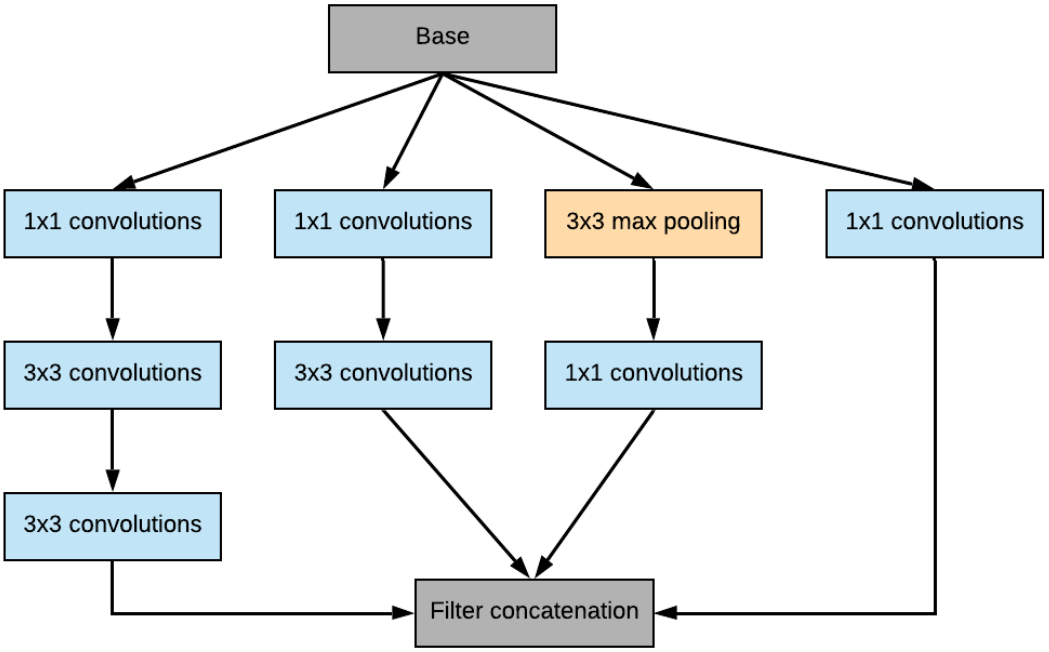

FIGURE 2.10: Inception module architecture

Fig. 2.10 shows a inception module. It performs convolution, with different sizes of filters (1x1, 3x3). Additionally, max pooling is also performed. The outputs are concatenated and sent to the next inception module. The authors limit the number of input channels by adding an extra 1x1 convolution before the 3x3 convolution.

This model has sparsely connected network architectures instead of fully connected network architectures, especially inside convolutional layers. Using this approach very deep architectures, increasing the depth and width of the network, can be maintained easily.

The inception layer is the core concept of a sparsely connected architecture. Inception Layer is a combination of multiple layers (namely, 11 Convolutional layer, 33 Convolutional layer, 55 Convolutional layer) with their output filter banks concatenated into a single output vector forming the input of the next stage.

2.4.2 ResNet50

ResNet-50 [8] is a convolutional neural network that is trained on more than a million images from the ImageNet database [6]. The network is 50 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images.

2.4.3 VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [9]. The model achieves 92.7 percent top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 33 kernel-sized filters one after another.

Fig. 2.11 shows the architecture of VGG16 model. The input to conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 33. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 22 pixel window, with stride 2. Three Fully-Connected (FC) layers follow a stack of convolutional layers. All hidden layers are equipped with the rectification (ReLU) non-linearity.

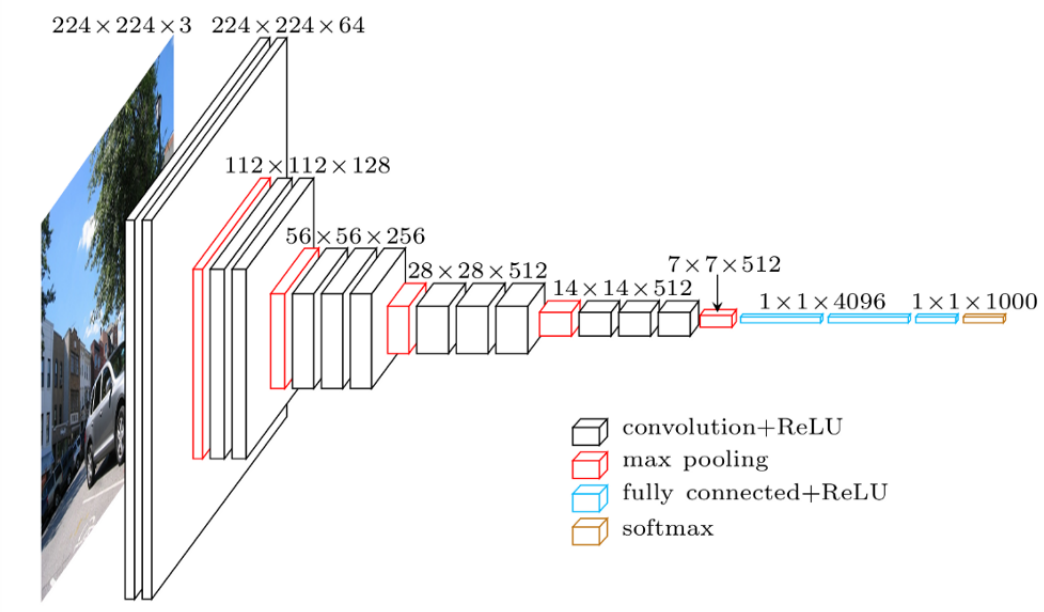

FIGURE 2.11: VGG16 architecture

2.4.4 VGG19

VGG-19 [9] is a convolutional neural network that is trained on more than a million images from the ImageNet database [6]. The network is 19 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images.

2.5 Related Work

Previous work in book inventory management has typically focused on book spine detection for which many different algorithms have been proposed. A general framework of book spine extraction was proposed by Quoc and Choi [10] and several challenges were addressed, such as lighting condition and distortion of images. Systems have been also designed for reading book spines including that of Chen et al. [3] and Tsai et al. [1].

Chen et al. [3] designed an efficient Hough Transform based book boundary detector to be used with SURF [4] feature image matching in order to retrieve books in an inventory. Tsai et al. [1] proposed a hybrid method that combined a text-reading-based method with an image matching-based method.

It is important to note that the performance of most existing approaches is limited by book spine recognition and off-the-shelf OCR systems. Hand-craft features based book spine segmentation suffers from common image distortion and low contrast between books. Off-the-shelf OCR system, such as Tesseract [11], perform poorly on image taken of natural scenes. Recently, scene text reading has gained a great deal of attention in the computer vision community.

In this paper, we present a deep neural network based system for text detection and recognition and demonstrate that this system can be effectively applied to book information retrieval and book inventories management. Also, image-feature based system can be improved using deep learning models instead of using hand-crafted features. Combined with other image processing techniques, such as the Canny Edge Detection [5] and Hough Transform, our system achieves robust performance on book information retrieval.

Chapter 3

Proposed Work

The proposed work mainly contains three steps:

1. Segmenting book spines from the bookshelf image.
2. Apply text detection procedure individually on the extracted spines.
3. Recognize all the text regions detected by text detection procedure using a hybrid deep learning model that combines a CNN with an RNN.

We also evaluate the performance of image feature-based system. Previous work [2, 3] has been done using image features but that is mainly based on low-level feature extraction method such as SURF [4]. We propose a image-feature extraction method using neural networks.

After text recognition, the recognized keywords are searched through an online database of books to get the information about every query spine image. The returned results are matched with the recognized one using similarity measures and then the books are marked as correctly identified or incorrectly identified.

3.1 Book Spine Segmentation

Book spine segmentation is a critical component of our system since each book is expected to be recognized, stored, and queried independently. To segment out individual book spines

in the query image, we exploit the fact that often books are densely packed together on a bookshelf and have roughly the same orientation.

Fig. 3.1 shows a block diagram of the operations required for the book spine segmentation. Two query images having multiple book spines in nearly vertical and slanted orientation are shown in Fig. 3.2. First, a Canny edge map is extracted from the query images [5]. In the Canny edge map, the separating lines between book spines appear as straight edges but the edges extracted by Canny edge map are not continuous. Fig 3.3 shows the edge map of the query images.

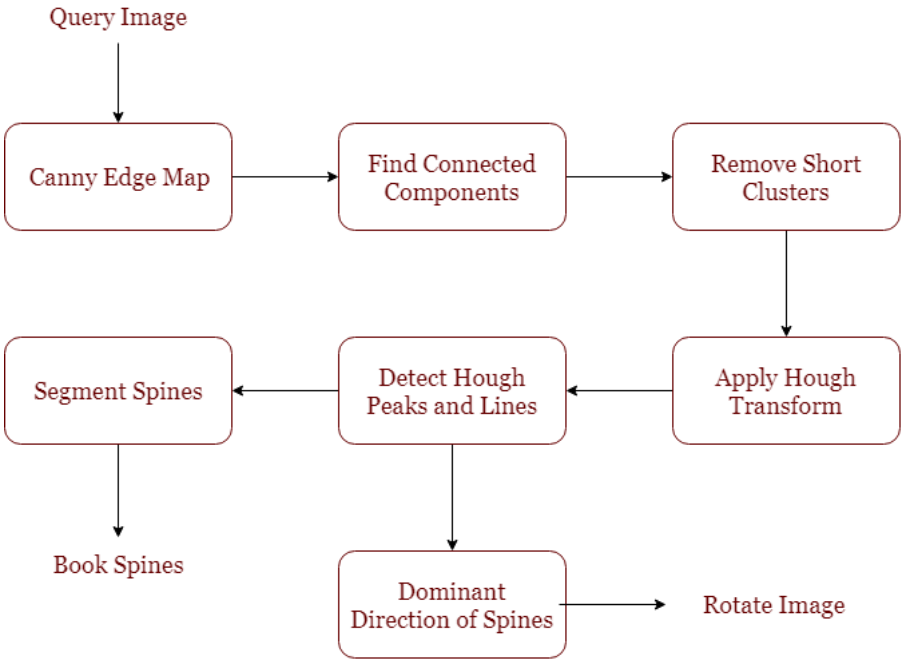

FIGURE 3.1: Operations for book spine segmentation

The book title, author names and any other graphics on the book spines appear as multiple short and curved edges. These short edges are mainly the noise and not required for detecting lines between the book spines. It create problems in finding the peaks in the Hough transform. So to remove short and curved edges, connected components are detected in the Canny edge map. 8-connectivity is used for finding connected components because the segmenting lines can be in any direction. 4-connectivity would check only for vertically and horizontally connected components. All components containing fewer pixels than a lower threshold are removed. Our experimental results show that threshold of 200 connected pixels works better. Fig. 3.4 shows the edge map after removing short and curved edges.

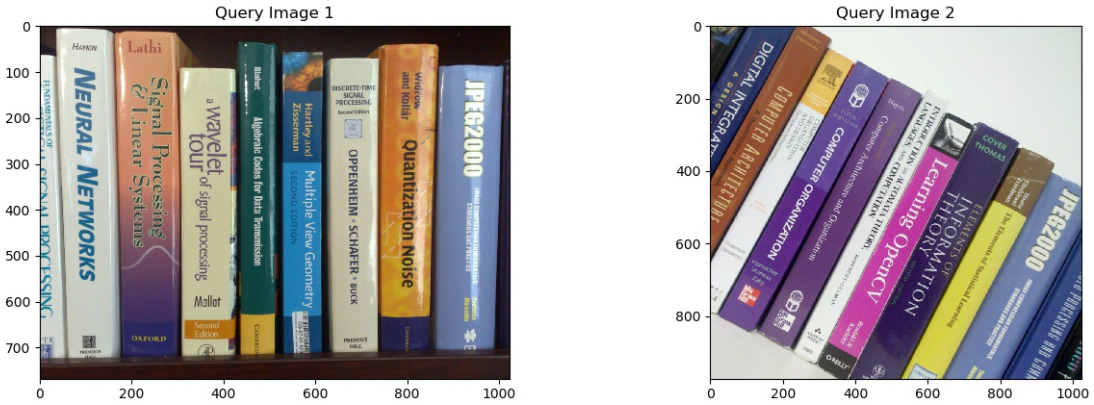

FIGURE 3.2: Query image showing book spines

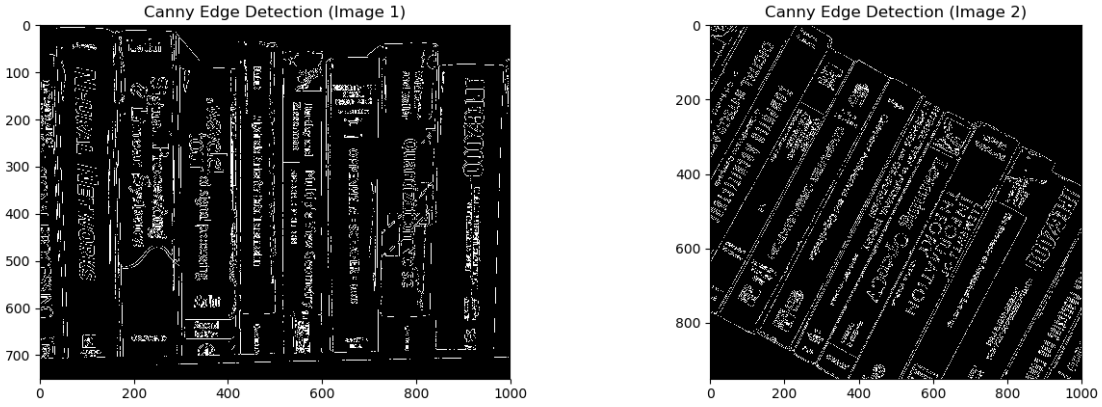

FIGURE 3.3: Extracted Canny edge map

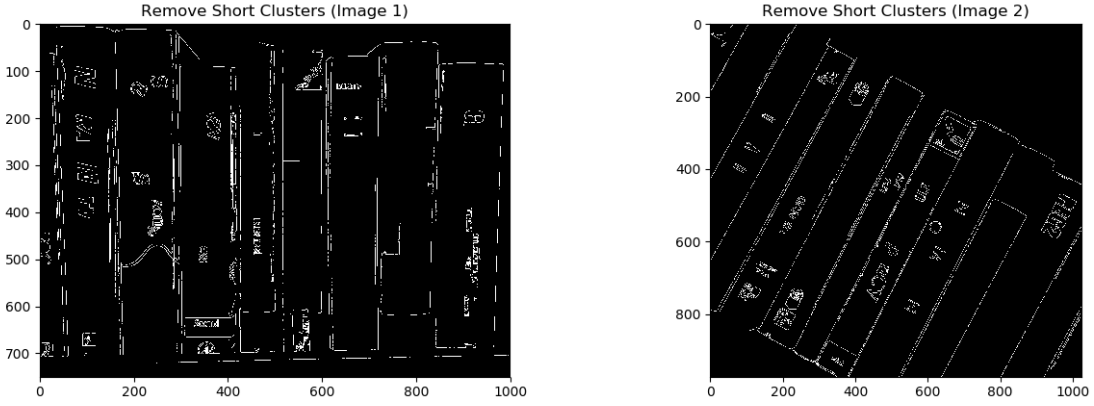

FIGURE 3.4: Short and curved edges removed

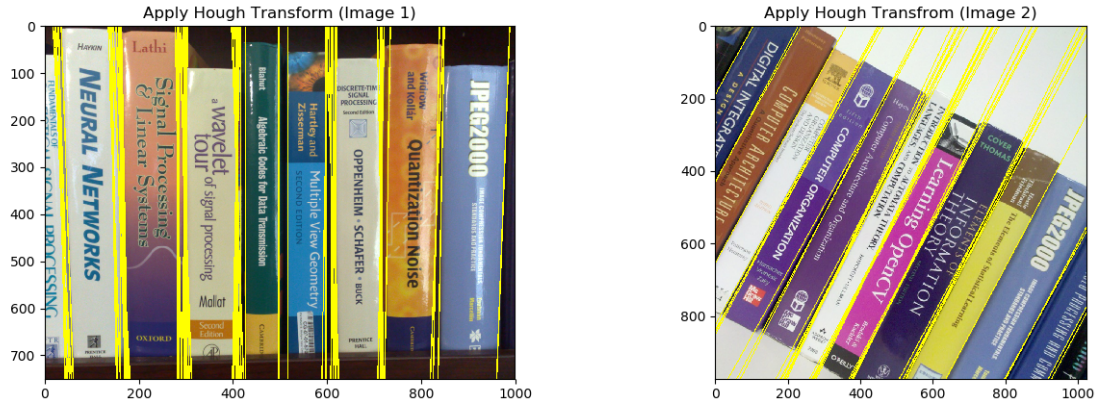

FIGURE 3.5: Detecting lines using Hough transform

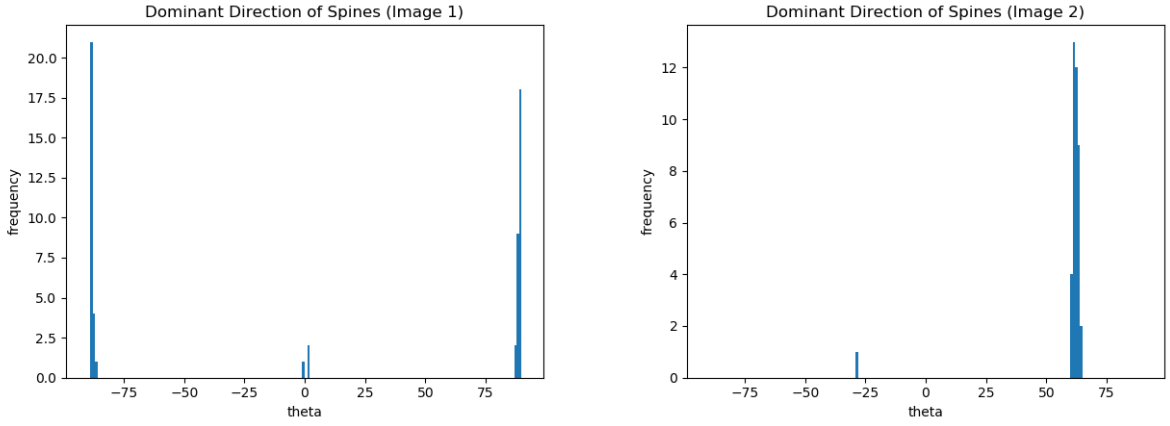

FIGURE 3.6: Dominant direction of spines in bookshelf

Next, to detect the long and straight lines in the Canny edge map with short edges removed, a Hough transform is calculated. Fig 3.5 shows the Hough transform calculated for the two query bookshelf images. To find the dominant angle of the book spines, frequency is calculated for every angle in the Hough peaks. The angle with the highest count yields the dominant angle θ_{spines} of spines. A histogram of θ values is shown in Fig. 3.6.

Hough transform gives multiple lines between two book spines. So the lines which are closer, are merged into a single one. This gives the final segmenting boundaries for bookshelf images, as shown in Fig. 3.7.

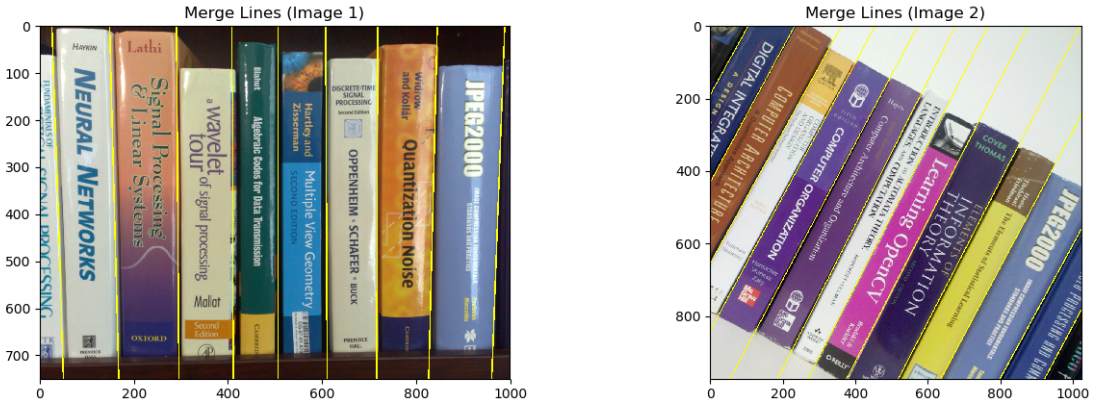

FIGURE 3.7: Segmentation boundaries between book spines

3.2 Book Spine Recognition

We propose two methods for book spine recognition:

1. **Image feature-based recognition**: Image features are extracted from the book spine images using convolutional neural network and matched to a already extracted book spine image features.
2. **Text-based recognition**: Texts from each book spine image, are detected and recognized and those are used as keywords to search from a database of book spine text.

We compare the performance of both the text and image based method.

3.2.1 Recognizing Spines with Image Features

In the image feature-based book spine recognition method, image features are used to match the query book spine to a database of book spine images. Every segmented spine is individually matched against a database of extracted book spine features.

Fig. 3.8 illustrates the steps used in the recognition process. We created a database of 497 book spines extracted from 59 bookshelf images. For each book spine image in database, features are extracted using a pre-trained convolutional neural network. All the extracted features are saved in a database. Now, for each query book spine image, features are

extracted and the Manhattan distance is calculated between the extracted features and the database of features. Minimum distance vector is selected and the corresponding book image then can be extracted.

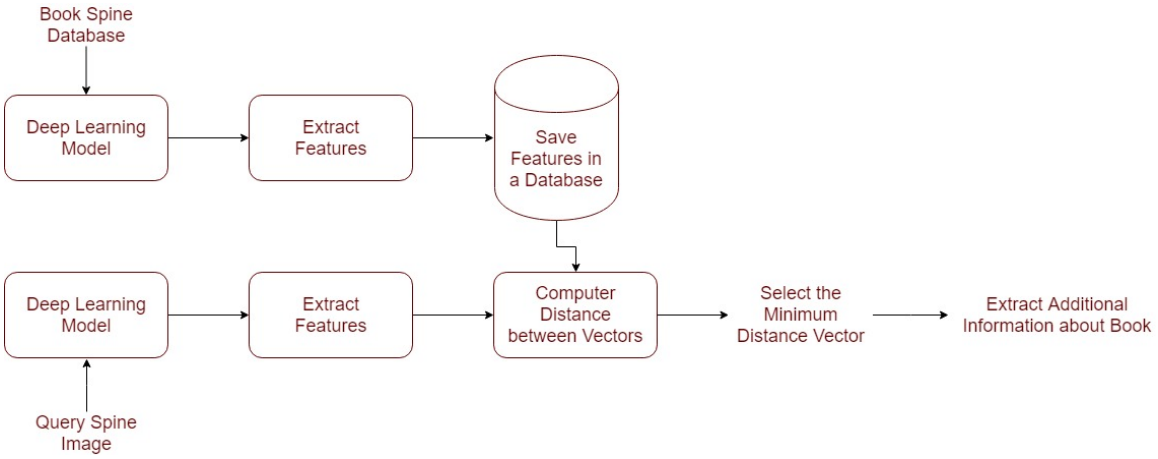

FIGURE 3.8: Operations for book spine recognition

We compared four different models for extracting feature vector: VGG16, VGG19, ResNet50 and InceptionV3. The extracted feature vector has dimension of 512 in VGG16 and VGG19 and 2048 in ResNet50 and InceptionV3.

We used a unsupervised machine learning technique, dimensionality reduction, to take the number of items in our feature vector (dimensions) down to 2. By reducing the dimensions down this way we can then easily visualize the relationships between each vector using a scatter plot to decide whether or not the features extracted by the models can distinguish the images of book spines for good recognition.

For this we used t-Distributed Stochastic Neighbor Embedding (t-SNE) and Principal Component Analysis (PCA) algorithm. PCA plot in Fig. 3.9 shows that most of feature vectors in InceptionV3 forms a cluster and the feature vectors have very less distance. Same clusters have been formed in VGG16 and VGG19. So in these cases, when the feature vectors are very similar, it creates problem in book spine recognition because multiple book spine images have similar features. ResNet50 also forms clusters but cluster size is small compared to other models. It helps in finding the best match feature vector and hence in recognizing book spines.

PCA is a linear dimensionality reduction algorithm. It will not be able to interpret complex polynomial relationship between features. t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimensionality reduction algorithm. It is based on probability

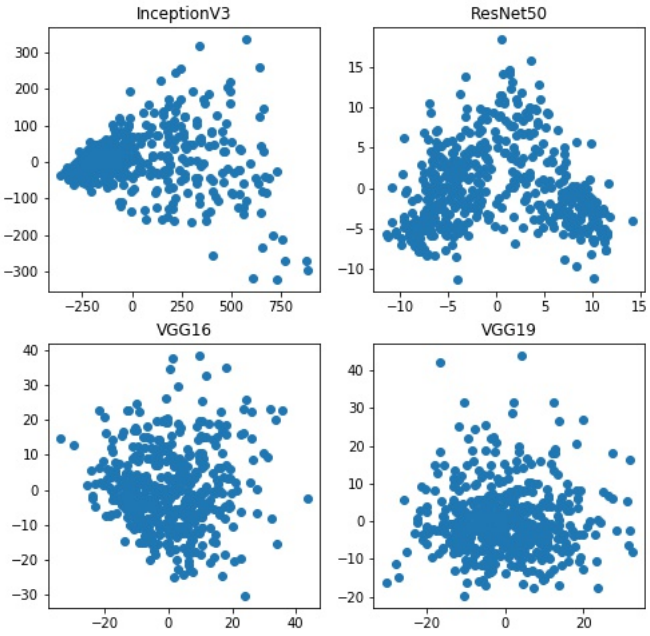

FIGURE 3.9: PCA algorithm applied to feature vectors

distributions with random walk on neighborhood graphs to find the structure within the data.

A major problem with, linear dimensionality reduction algorithms is that they concentrate on placing dissimilar data points far apart in a lower dimension representation. But in order to represent high dimension data on low dimension, non-linear manifold, it is important that similar data points must be represented close together, which is not what linear dimensionality reduction algorithms do.

Fig. 3.10 shows the results of tSNE dimensionality technique applied to the features extracted by different models. According to the plot, VGG19 has the most dissimilar feature vectors. InceptionV3 shows negative correlation in the features data. ResNet50 is showing three different clusters formed. Of all these models, VGG19 should perform better in spine recognition process.

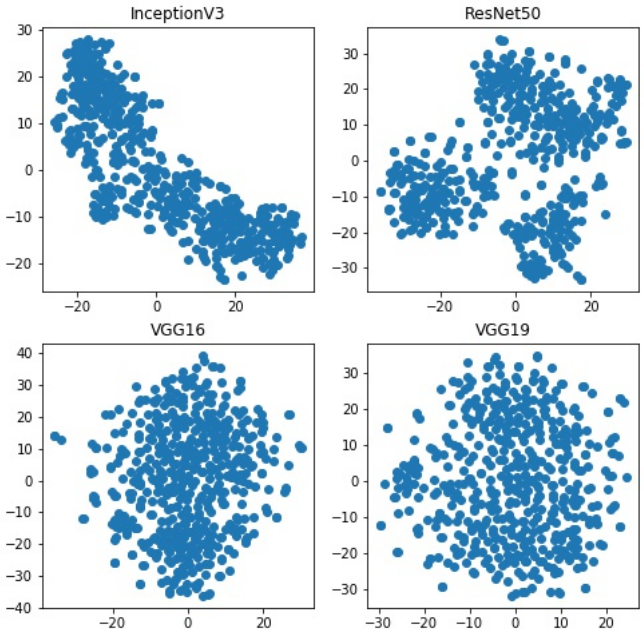

FIGURE 3.10: tSNE algorithm applied to feature vectors

3.2.2 Recognizing Spines with Text

The text on the book spines typically contains the title and the author names, which can provide effective keywords to search for the book. To use the text on the book spines, the text within the extracted book spine image has to be automatically recognized.

3.2.2.1 Text Detection

In this text detection method, a neural network model is trained such that it can directly predict the existence of text instances and their geometries from the images. The model utilizes a fully-convolutional network model adapted for text detection that produces word or text-line level predictions, excluding slow intermediate and redundant steps such as word partition and text region formation. The produced text predictions are sent to Non-Maximum Suppression (NMS).

Fig. 3.11 shows a schematic view of text detection model. The algorithm follows the design of DenseBox. In DenseBox, an image is fed into the fully-convolutional network and multiple channels of score map and geometry are generated. Score map is one of

the predicted channels and it's pixel values are in the range of [0,1]. Remaining channels represent geometries that encloses the word. Thresholding is then applied to each predicted region. The geometries whose scores are greater than the threshold is considered valid and then Non-Maximum Suppression (NMS) is applied to get the final output.

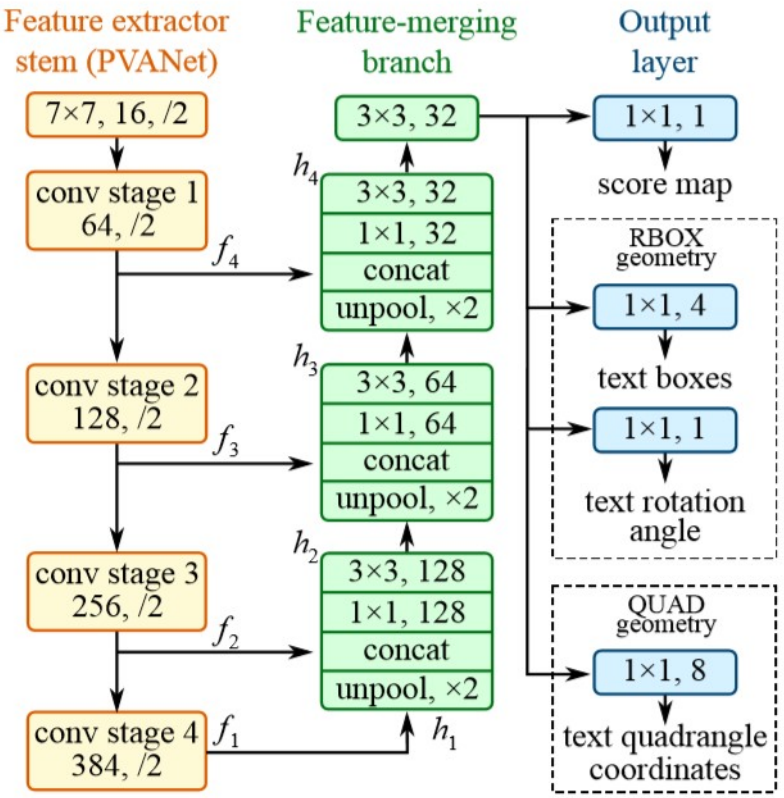

FIGURE 3.11: Architecture of the text detection model

The sizes of word regions in case of books can be very small or large. So determining the existence of large words would require features from late-stage of neural network, while small word regions need low level information in early stages of a network. To fulfill these requirements, the network must use features from different levels of the network. HyperNet [12] meets these conditions on features maps.

The model has three parts: feature extractor, feature merging and then output layer. The feature extractor is a convolutional neural network pre-trained on ImageNet dataset [6], with interleaving convolution and pooling layers. In the feature-merging process, the features are merged. In each merging step, features from the last stage is fed to an unpooling layer to double its size and then they are concatenated with the current feature map. The final output layer project 32 channels of feature maps into channels of score map and geometry map.

The text on the book spine can be in any orientation, so we need to decide whether a text line is upside down or not. To solve this problem, we train a CNN classifier on 32 X 64 image patches taken from the IIIT 5K-word dataset [13]. The binary classifier tells whether we need to rotate the text lines by 180° in order to provide the correct sequence to text recognition module. During testing, we first resize the height of all the extracted text regions to 32 pixels. Then passed it through the trained network. If the probability is higher than 0.7, we will rotate the text region.

3.2.2.2 Text Recognition

After the text detection process, texts are recognized. The recognized texts are used as keywords to search from a book database. In the text recognition step, a commonly used method is to first segment each character and recognize them, then output a word level prediction. These approaches are highly sensitive to various distortions in images, making character level segmentation less accurate. To bypass the character segmentation, our model is trained to recognize a sequence of characters simultaneously. We use a hybrid approach that combines a CNN with an RNN similar to He et al. and Shi, Bai, and Yao [14]. A CNN first learns feature sequences from the image which are subsequently fed into an RNN for sequential labeling.

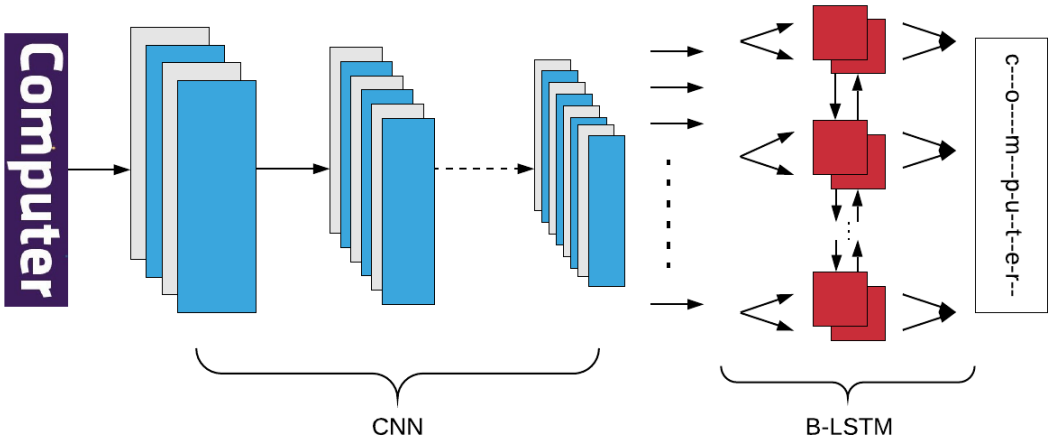

FIGURE 3.12: Architecture of the text recognition model

As shown in Fig. 3.12, the network architecture of CRNN consists of three components:

1. Convolutional layers
2. Recurrent layers

3. A transcription layer

The convolutional layers first extract a feature sequence $F = \{f_1, f_2, ..., f_N\}$ from each input image. After that, a Bidirectional Long Short-Term Memory (B-LSTM) model [15] is built for making prediction for each frame of the feature sequence. The transcription layer further translate the per-frame predictions by the recurrent layers into a label sequence, yielding sequence $X = \{x_1, x_2, ..., x_N\}$ as final outputs. Each one of the x_i is normalized through a softmax function. The target word Y can be viewed as a sequence of characters, $Y = \{y_1, y_2, ..., y_T\}$.

The predicted sequence X and target word Y have different lengths N and T respectively. In the absence of a per-timestep groundtruth label, it makes it difficult to train our model. So we adopt CTC loss [16] to allow an RNN to be trained for sequence labeling without exact alignment. CTC loss is the negative log likelihood of outputting a target word Y given an input sequence X.

$$CTC(X) = -\log P(Y|X) \tag{3.1}$$

Suppose there is an alignment a which provides a per-timestep prediction (whether blank or non-blank labels) for X, and a mapping function B which would remove repeated labels and blanks from a. For instance, $(-, a, a, -, -, b)$ would be mapped as (a, b) (using $-$ to denote blank label). Then, $P(Y|X)$ can be calculated by summing all possible alignments a that can be mapped to Y:

$$P(Y|X) = \sum_{a \in B^{-1}(Y)} P(a|X) \tag{3.2}$$

3.3 Text Searching

Once the text is recognized, the keywords are used to query an online database of books. First, using the keywords, we make a query. That query is searched on Google and the returned results are scrapped using python to find the links for amazon books. Once the amazon link is found, Amazon Products API is used for querying the Amazon book database. We also query Google Books API using the recognized keywords. The returned results from both the queries are compared with the recognized keywords to find the best match.

Chapter 4

Results and Discussion

We have implemented a library book recognition system, employing the techniques discussed in this paper. Each time the user takes a photo of a bookshelf, our system automatically segments and recognizes each spine in the photo. According to the requirement of the user, he can either search a book in library or can create an inventory of the books. In case of inventory generation, the recognized books are added to the inventory database.

To evaluate the performance of our proposed system, we constructed a database of 497 book spine images extracted from the 59 bookshelf images. The bookshelf images are collected from the Central Library at Malaviya National Institute of Technology Jaipur.

4.1 Segmentation Accuracy

To test the segmentation accuracy, we applied our segmentation process on each 59 query images. This process gives us the segmenting lines between book spines. Then, we manually checked each query image to find the correctly segmented spines, incorrect segmented spines, spines not segmented or the spines which has two or more books after segmenting. These results are shown in table 4.1.

Total spines in query images	503
Total spines segmented by our method	520
Correctly segmented	464
Incorrect spines	49
Segmented spine has 2 or more books	7
Spines not segmented	39

TABLE 4.1: Book spine segmentation parameters

Table 4.2 shows the confusion matrix for segmenting book spines. It reports the number of *false positives*, *false negatives*, *true positives* and *true negatives*. This allows more detailed analysis than mere proportion of correct classifications (accuracy).

		Prediction outcome		
		p	**n**	**total**
Actual value	**p′**	464 TP	39 FN	503
	n′	56 FP	0 TN	56
	total	520	39	

TABLE 4.2: Confusion matrix for segmentation process

The *precision* or positive predictive value (PPV) 4.1 is defined as the proportion of correctly segmented book spines to the declared correct segments, while the *recall* or true positive rate (TPR) 4.2 is defined as the proportion of correctly segmented book spines to all book spines. F_1 *Score* 4.3 is the harmonic mean of *precision* and *recall*.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$F_1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.3}$$

Using the above equations, we calculate the values for *precision*, *recall* and F_1 *Score*. Values are shown in Table 4.3. A *precision* score of 0.89 shows that most of the spines segmented by our method were correct. A *recall* score of 0.92 shows that most of the spines were correctly segmented by our process.

Precision	0.89
Recall	0.92
F1 Score	0.90

TABLE 4.3: Segmentation process evaluation

4.2 Book Spine Recognition Accuracy

To test the book spine recognition accuracy, we extract the features from each image of the book spine database of 497 images. During testing these features are matched with the extracted one. Once the book is correctly identified, extra information about book can also be retrieved.

Deep Learning Model	VGG16	VGG19	ResNet50	InceptionV3
Total Spines	497	497	497	497
Recognized Spines	403	373	424	286
Accuracy	81.08 %	75.05 %	85.31 %	57.54 %

TABLE 4.4: Book spine recognition accuracy

Table 4.4 shows the accuracy for four different model. Among all the models, ResNet50 correctly identifies 403 spines out of 497 spines getting an accuracy of 85.31 %.

Chen et al. (2010) [3] reports an accuracy of 74.5 % using SURF features [4]. Our method achieves 10.8 % higher accuracy compared to SURF feature extraction method.

Chapter 5

Conclusion and Future Scope

We have presented a recognition system for identifying books on a bookshelf for use in book management systems. The proposed system helps in finding a required book without manually searching each and every bookshelf and also it can be used for generating book inventories without going through a tedious manual work which is very time consuming.

A user takes a photo of the bookshelf using his smartphone. The query image first goes to the server. From the query image, system automatically extracts the individual book spine images by analyzing the line structures. Text is detected and extracted from the book spine images and used as keywords to search through a online book spine text database. After the book is identified, additional information about the book such as author, publisher, edition, short description of book can be extracted. User can also search for the related books matching to the recognized book.

Through experiments, we demonstrated that the proposed scheme achieves a significantly higher accuracy when using deep learning models for extracting features.

5.1 Future Scope

1. Develop a fully functional Android app using augmented reality to interact with books in real time.

2. Use the digital compass and accelerometer on the smartphone to estimate location of the books. The digital compass tells the direction the phone is facing when a book is photographed, while the trace of the accelerometer tells how far vertically and horizontally the phone has moved from the last anchor point.

Bibliography

[1] S. S. Tsai, D. M. Chen, H. Chen, C.-H. Hsu, K.-H. Kim, J. P. Singh, and B. Girod, "Combining image and text features: a hybrid approach to mobile book spine recognition," in *ACM Multimedia*, 2011.

[2] D. L. Chen, S. S. Tsai, K.-H. Kim, C.-H. Hsu, J. P. Singh, and B. Girod, "Low-cost asset tracking using location-aware camera phones," 2010.

[3] D. M. Chen, S. S. Tsai, B. Girod, C.-H. Hsu, K.-H. Kim, and J. P. Singh, "Building book inventories using smartphones," in *ACM Multimedia*, 2010.

[4] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features.," vol. 110, pp. 404–417, 01 2006.

[5] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679–698, Nov 1986.

[6] J. Deng, W. Dong, R. Socher, L. jia Li, K. Li, and L. Fei-fei, "Imagenet: A large-scale hierarchical image database," in *In CVPR*, 2009.

[7] S. I. J. S. Z. W. Christian Szegedy, Vincent Vanhoucke, "Rethinking the inception architecture for computer vision," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[10] N.-H. Quoc and W.-H. Choi, "A framework for recognition books on bookshelves," in *ICIC 2009*, 2009.

[11] R. Smith, "An overview of the tesseract ocr engine," vol. 2, pp. 629 – 633, 2007.

[12] T. Kong, A. Yao, Y. Chen, and F. Sun, "Hypernet: Towards accurate region proposal generation and joint object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 845–853, 2016.

[13] A. Mishra, K. Alahari, and C. V. Jawahar, "Scene text recognition using higher order language priors," in *BMVC*, 2012.

[14] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 2298–2304, 2017.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 1997.

[16] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006.

[17] X. Yang, D. He, W. Huang, Z. Zhou, A. Ororbia, D. Kifer, and C. L. Giles, "Smart library: Identifying books in a library using richly supervised deep scene text reading," *CoRR*, vol. abs/1611.07385, 2016.

[18] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: An efficient and accurate scene text detector," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2642–2651, 2017.