

CONTENT BASED IMAGE RETRIEVAL (CBIR) FOR
BRAND LOGOS

by
Enjal Parajuli



A thesis
submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Boise State University

May 2020

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Enjal Parajuli

Thesis Title: Content Based Image Retrieval (CBIR) for Brand Logos

Date of Final Oral Examination: 17 March 2020

The following individuals read and discussed the thesis submitted by student Enjal Parajuli, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Timothy Andersen, Ph.D.

Chair, Supervisory Committee

Edoardo Serra, Ph.D.

Member, Supervisory Committee

Casey Kennington, Ph.D.

Member, Supervisory Committee

The final reading approval of the thesis was granted by Timothy Andersen, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

This thesis is dedicated to my family. I have come this far only because of their love and support.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Timothy Andersen, for his guidance and support. I would also like to thank him for believing in me, helping me in choosing the path of my interest, and helping me in figuring out my thesis topic. I have learned a lot from him, and for that, I will always be grateful.

I am grateful to my committee members Dr. Edoardo and Dr. Kennington for being in my committee and for their valuable suggestions.

I would also like to thank Boise State University, Computer Science department, for offering full funding throughout my Master's program. I would also like to thank Benjamin Peterson for his technical support, and for helping me in building my first computer ever and to my advisor for giving me that opportunity. I would also like to thank administrative staffs for all their help.

I would also like to thank staff members of Idaho National Laboratory, especially Mr. Eric Whiting, for sponsoring me for using their GPU clusters.

Finally, but not least, I would like to thank my family for their love and support. I would also like to thank my friends, colleagues, and to any individual, I ever talked to about my thesis.

This research made use of the resources of the High Performance Computing Center at Idaho National Laboratory, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517.

Abstract

This thesis explores the problem of automatically detecting the presence of logos in general images. Brand logos carry the goodwill of a company and are considered to be of high value in the corporate world, and thus automatically determining whether or not a logo is present in an image can be of interest for companies that wish to protect their brand. The problem of automated logo detection is inherently complex, but is further complicated through intentional obfuscation of logo images, for example by color shifting or other slight image modifications that leave the logo intact and easily recognizable by a human, but difficult to determine through automated techniques.

For our use case, we are interested in leveraging the basic Content-based image retrieval (CBIR) approach to determine whether or not a brand logo is present within a larger image. CBIR systems retrieve images based on their similarity to a given query image/images. For example, this enables a user to retrieve images that contain dogs by entering an image with a dog in it.

Of the multitude works done in this field, most use supervised and only a few use self-supervised machine learning techniques. For our approach presented in this thesis, we utilize an Autoencoder, which is one of the self-supervised techniques. It is able to generate compressed representations of the original image and its variant exposed to simple transformations such as Gaussian noise and near to zero degree rotations. It, however, is not able to do so for complex transformations/noises. This thesis addresses this problem by introducing a novel autoencoder based architecture

that takes the encoding technique of an autoencoder one step further and produces a semantic rich compressed latent representation for the transformed logo that is similar to its original logo. This research work has achieved significant improvement over the general autoencoder architecture, and over current CBIR-based approaches for logo recognition.

Contents

Abstract	vi
List of Tables	xii
List of Figures	xiii
LIST OF ABBREVIATIONS	xvii
1 Introduction	1
1.1 Overview	1
1.2 Research Questions	4
1.3 Thesis Outline	5
2 Preliminary Work & Motivation	6
2.1 Existing Systems	6
2.1.1 Google Logo Detection	6
2.1.2 InVid	7
2.1.3 LIRe	7
2.2 Preliminary Work	8
2.2.1 With News Logos	9
2.2.2 With General Logos	10
2.3 Motivation	14

3	Background	15
3.1	Algorithms and terminology	15
3.1.1	Convolution Networks	15
3.1.2	Activation Functions	16
3.1.3	Max-Pooling Layers	17
3.1.4	Auto-encoders	18
3.1.5	Training Set, Validation Set, and Test Set	19
3.1.6	Loss Functions	19
3.1.7	Training Loss & Validation Loss	20
3.1.8	Overfitting & Underfitting	21
3.1.9	k-means Clustering	21
3.2	Related work	21
4	Methods	27
4.1	General System Architecture	27
4.1.1	Feature Extractor	28
4.1.2	Matching Process	30
4.1.3	Multilevel Searching	34
4.1.4	Architecture for more compression	35
4.2	Training Architecture	36
4.3	Datasets	38
4.4	Transformations	38
4.4.1	Rotations	39
4.4.2	Translation	40
4.4.3	Added Gaussian Noise	40

4.4.4	Stretching	40
4.4.5	Flipping and flopping	40
4.4.6	Negate	41
4.4.7	Cropping	41
4.4.8	Inserting logo into another picture	41
4.4.9	Add border	42
4.4.10	Darken and Lighten	43
4.4.11	Append	43
4.5	Creating ground truth	43
4.6	Creating Training Set, Validation Set, and Test Set	44
4.7	Evaluation	45
4.8	Creating Baselines	46
4.8.1	LIRe with individual descriptors	47
4.8.2	LIRe with Multilayer Perceptron	49
4.8.3	Full Autoencoder	51
4.8.4	Encoder Only	53
4.8.5	Google Logo Detection	55
5	Experiment & Results	56
5.1	Training on smaller datasets	56
5.2	Training on bigger dataset	57
5.3	Training for more compression	58
5.4	Manual combining of smaller models	61
5.5	Multilevel retrieval and charts	62
5.6	Observation	66

6	Conclusions	70
6.1	What has been done so far?	70
6.2	Limitations & Future Work	71
	Bibliography	72

List of Tables

4.1	Number of Original Images, Training Images, Test Images, and Validation Images in different datasets used in experiments	44
4.2	AUC value for each of the LIRe descriptors for the dataSets of 500 Images	46
4.3	AUC value for each of the LIRe descriptors for the dataSets of 1000 Images	47
4.4	Accuracy comparison using individual LIRe descriptors and MLP on datasets containing 500 images	49
4.5	Accuracy comparison using individual LIRe descriptors and MLP on datasets containing 1000 images	49
4.6	Full-Autoencoder & Encoder-Only accuracy on 500 images datasets . . .	52
4.7	Full-Autoencoder & Encoder-Only accuracy on 1000 images datasets . .	53
5.1	Accuracy of the proposed model at different compression on smaller datasets	57
5.2	Accuracy of the proposed model at different compression on bigger datasets when trained from scratch	58
5.3	Accuracy on bigger datasets by manual combining of smaller models . .	62
5.4	Accuracy Comparision of Proposed Model with baselines	67

List of Figures

2.1	InVid User Interface	7
2.2	Different transformations of Instagram Logo	9
2.3	News logos	10
2.4	Overall Performance on News Logos	10
2.5	Logo images without text on it	11
2.6	Percentage detection of brand logos without text on it	11
2.7	Logo images with text on it	13
2.8	Percentage detection of brand logos with text on it	13
3.1	General working mechanism of convolution networks at a given layer . .	17
3.2	Working mechanism of max-pooling layers	18
3.3	Overfitting & Underfitting	20
4.1	Architecture of the proposed CBIR system	27
4.2	Internal Architecture of the AutoEncoder	29
4.3	clusters formed using K-Means clustering	31
4.4	Multilevel clustering up to Level 3 and branching factor of 2	33
4.5	Architecture for training autoencoder	36
4.6	Working Mechanism of Autoencoder	37
4.7	Rotated Images formed from the original Image	39
4.8	Translations	40

4.9	Gaussian noise of various levels	40
4.10	Stretching in X and Y directions	41
4.11	Flipped and flopped images	41
4.12	Negate	41
4.13	Cropped Images	41
4.14	Logo Images of size 56X56 inserted into colorful background Image . . .	42
4.15	Logo Images of size 112X112 inserted into colorful background Image . .	42
4.16	Added Border	42
4.17	Darken and Lighten	42
4.18	Various images appended to original logo Images	43
4.19	Process of creating Training, Validation, and Test Set	45
4.20	Average categorical accuracy of MLP750 on datasets containing 500 and 1000 original images	51
4.21	Average categorical accuracy of Full Autoencoder on datasets contain- ing 500 and 1000 original images	53
4.22	Average categorical accuracy of Encoder Only on datasets containing 500 and 1000 original images	54
5.1	Effect of training epochs on Mean Square Error (MSE)	60
5.2	Average accuracy on smaller datasets at a compression factor of 1 with varying branching factor at different levels of multilevel clusters	63
5.3	Average accuracy on bigger datasets at a compression factor of 1 with varying branching factor at different levels of multilevel clusters	63
5.4	Average accuracy on smaller datasets at a compression factor of 8 with varying branching factor at different levels of multilevel clusters	63

5.5	Average accuracy on bigger datasets at a compression factor of 8 with varying branching factor at different levels of multilevel clusters	63
5.6	Average accuracy on smaller datasets at a compression factor of 16 with varying branching factor at different levels of multilevel clusters . .	64
5.7	Average accuracy on bigger datasets at a compression factor of 16 with varying branching factor at different levels of multilevel clusters	64
5.8	Average accuracy on smaller datasets at a compression factor of 64 with varying branching factor at different levels of multilevel clusters . .	64
5.9	Average accuracy on bigger datasets at a compression factor of 64 with varying branching factor at different levels of multilevel clusters	64
5.10	Average retrieval time per query at different levels of multilevel clustering with Compression Factor of 1, threshold at 10% and branching factor of 2	65
5.11	Average retrieval time per query at different levels of multilevel clustering with Compression Factor of 8, threshold at 10% and branching factor of 2	65
5.12	Average retrieval time per query at different levels of multilevel clustering with Compression Factor of 16, threshold at 10% and branching factor of 2	65
5.13	Average retrieval time per query at different levels of multilevel clustering with Compression Factor of 64, threshold at 10% and branching factor of 2	65
5.14	Retrieval time per query at different levels of multilevel clustering with Compression Factor of 128, threshold at 10% and branching factor of 2	66

5.15	Average categorical accuracy comparison of the proposed model at a compression factor of 16 with baselines for smaller datasets	68
5.16	Average categorical accuracy comparison of the proposed model at a compression factor of 128 with baselines for bigger datasets	69

LIST OF ABBREVIATIONS

ACC – Auto Color Correlogram

Acc@1 – Accuracy at top-1

Acc@10 – Accuracy at top-10

BGAN – Binary Generative Adversarial Network

CBIR – Content based Image Retrieval

CEDD – Color and Edge Directivity Descriptor

CNNs – Convolution Neural Networks

DCNN – Deep Convolution Neural Network

FCTH – Fuzzy Color and Texture Histogram

FIRE – Flexible Image Retrieval Engine

FPID – Fribourg Product Image Database

JCD – Joint Composite Descriptor

KNN – K-nearest Neighbour

LIRe – Lucene Image Retrieval

LLQ – Lower Left Quadrant

MSE – Mean Square Error

LRQ – Lower Right Quadrant

SIFT – Scale Invariant Feature Transform

SURF – Speeded-Up Robust Features

ULQ – Upper Left Quadrant

URQ – Upper Right Quadrant

Chapter 1

INTRODUCTION

1.1 Overview

This thesis work is particularly focused on creating an image retrieval system that can be used to assist in brand protection using a logo detection strategy. Brand logo detection and protection is important because brands carry the goodwill of a company and are considered of high value in the corporate world. Brand protection is not only about protecting the reputation of a company, but also about preserving trustworthiness among customers and partners.

Identifying and retrieving brand logos is difficult because there are no predefined rules for designing logos. A logo may consist of numbers, letters, combinations of both, or may just be a representation of an individual's thought. Sometimes companies present their logos differently by changing colors, fonts, backgrounds, etc. which further increases the challenges of brand identification, and complicates the brand protection task.

There are two popular types of image identification and retrieval techniques: annotation-based image retrieval, and content-based image retrieval (CBIR). Annotation-based image retrieval[1] searches and retrieves images based on the tags or names similarly associated with the image. Annotating images and performing searches on them may seem to give more promising results and make the retrieval system more

robust. However, it is impossible to exhaustively annotate an image with all the ways a user might view the information and want to be able to retrieve it. Furthermore, the process of annotating becomes more complex as the number of images increase. A recent boom in the production of various multimedia devices has created millions of digital images. Moreover, with the multimedia and storage devices getting cheaper every day, it is only going to increase the number of images produced and stored. This ever-increasing volume of images makes manual annotation impractical and unscalable.

CBIR[2] systems retrieve images that are visually similar to a given query image presented by a user. The similarity is based on the content of the image such as colors, textures, shape, and other features that are solely derived from the image itself rather than names, tags or annotations associated with the image. For example, if a user wants to find the picture(s) containing a Nike logo in a database, they will enter the picture of the Nike logo as the query, and the system will (hopefully) retrieve pictures containing Nike logo.

Even though the CBIR system is scalable compared to its annotation-based counterpart, it is not without problems. One of the most difficult problems of any CBIR system is the *semantic gap* problem. A machine or system may be able to perceive low-level feature representations, but generally has difficulty to form a hierarchy of concepts from images the way human beings do, which is popularly known as the semantic gap problem. The semantic gap problem can be avoided by representing an image properly with the meaningful features [4] and using proper distance metrics on those features [5] to calculate the similarity between images, which is still an area of active research.

One of the most popular emerging techniques for overcoming the semantic gap

problem is through the use of deep learning. Deep learning is a member of the machine learning family which consists of networks of neurons and tries to approximate the human brain to generate features and abstractions of the underlying data [11, 12, 13]. Like the neurons in the human brain communicate with one another, the neurons present in each layer of a deep learning model communicate with one another, learn the pattern, and transfer the knowledge to the neurons present in the next layer in a feed-forward scheme. A lot of work [6, 7, 8, 9] has been done in the field of image recognition and retrieval using machine learning and deep learning techniques. The supervised [23] method trains a system by mapping images to an integer value (also known as a class), rather than mapping to the actual information present in the image.

Autoencoders, introduced by Hinton et al. [24], is a self-supervised deep learning method that allows a network of neurons to learn encodings (generally compressed) of arbitrary data in a self-supervised manner. Denoising autoencoders [25] are a kind of autoencoder that maps images to the contents present in the image by a reconstruction process that generates a semantically rich compressed latent representation. For example, given a transformed image as input, an autoencoder takes the original non-transformed image as the label and trains the system to reconstruct the non-transformed image from the transformed input image. In the process of doing so, it generates a compressed representation at the encoder layer. An autoencoder has two parts - an encoder that generates the compressed representation of the input image, and a decoder that reconstructs the image from the compressed representation.

This thesis is an effort to create a robust CBIR system for brand logo retrieval that can be used for brand protection. In addition to using the system to detect infringing logos or images, the proposed CBIR system for logo retrieval can also be

used by people who want to design new logos for their companies. This can be done by feeding the new logo just created as a query to the CBIR system to see if other logos look similar for avoiding unwanted brand infringement. This thesis work has developed an autoencoder based novel architecture for the CBIR task that is capable of generating compressed representations of images that allow for fast image retrieval. This research work lays the foundation necessary to create a CBIR system that is fast, accurate, and can scale well to datasets containing large numbers of images.

1.2 Research Questions

Q1. *How does the compression level of the encoding learned by the autoencoder affect performance (speed, accuracy)?* We are interested in training the proposed system with varying compression factors and observing whether the system can be trained. If it is trainable, how is the retrieval accuracy affected for each of the compression factors, and how does it affect the speed of retrieval.

Q2. *How does branching factor affect accuracy and speed of retrieval for a multilevel clustering problem?* One of the techniques of boosting the retrieval speed is through the use of multilevel clustering for search and retrieval. Multilevel clustering divides the data at each level into n related sub-clusters (if the branching factor is n) until a stopping criterion is reached. We are interested in observing the effect of branching factors on retrieval accuracy and also interested in understanding the change in retrieval time with the increase in levels of the multilevel clustering.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: chapter 2 introduces the preliminary works that were done to study the performance of the existing logo detection systems and also the motivation to carry out further research, chapter 3 introduces important terminologies and gives an overview of some of the closely related works, chapter 4 covers the architectural details and gives an overview of the methods that constitute the proposed system, chapter 5 walks through the different experiments that are conducted and their results, and chapter 6 concludes this thesis.

Chapter 2

PRELIMINARY WORK & MOTIVATION

Section 2.1 introduces some of the existing logo detection systems. Section 2.2 provides the preliminary work on existing logo detection systems that laid the groundwork for this thesis. Section 2.3 explains the motivation to carry out further research.

2.1 Existing Systems

At the time of writing this thesis, some of the existing logo detecting systems are Google logo detection, InVid [15], and Lucene Image Retrieval (LIRe) [37]. Their respective details are given in the following sections.

2.1.1 Google Logo Detection

Google logo detection identifies popular brand logos present in an image. It is a cloud-based API for logo detection which supports various programming languages: `c#`, Go, Java, Node.js, PHP, Python, and Ruby. It also supports other options: `gcloud` command, `curl`, and power shell. To identify logos in a given image, a user has to send a query image using the underlying API. The Google logo detection then returns a JSON response containing various attributes: the logo name present in the image, the score representing the confidence of detection, and coordinates of a bounding box to represent the area where the logo was detected.

2.1.2 InVid

InVid is capable of identifying news logos in videos as well as in images. It is also capable of identifying logos of a large number of media organizations and militant organizations participating in the Syrian war. Moreover, it also provides a Wikipedia link for the detected logo. InVid is a very easy to use web application as it does not require any programming experience. As shown in figure 2.1, a user can upload an image or drag and drop an image, or can enter the link of the image or video from the web for logo detection. The URL to InVid is <http://logos.iti.gr/logos/>.

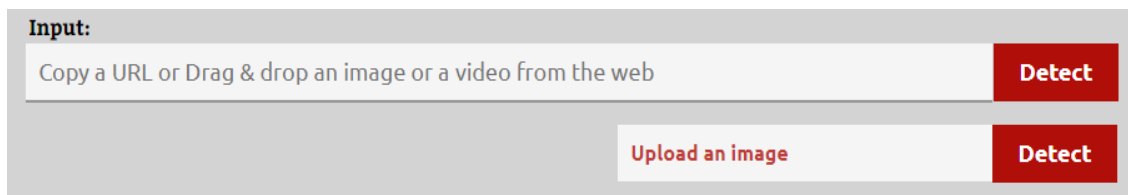


Figure 2.1: InVid User Interface

2.1.3 LIRe

LIRe is an open-source library, written in Java, that provides support for multiple image descriptors to extract features from images. LIRe uses LUCENE [16] indexing for the storage of the extracted image features and for faster retrieval of images. It is important to note that LIRe is designed for a CBIR system but not particularly designed to work for detecting logos. In our previous experiments, four LIRe image descriptors, FCTH (Fuzzy Color and Texture Histogram) [17], CEDD (Color and Edge Directivity Descriptor) [18], JCD (Joint Composite Descriptor) [19], and ACC (Auto Color Correlogram) [20] showed superior performance, based on the area under the curve values, at the identification of transformed logos. They will be referred to

as the top 4 LIRe descriptors, moving forward. Thus, this thesis work treats LIRe as an existing logo detecting system.

The overview of the top 4 LIRe descriptors are given in this paragraph. FCTH and CEDD descriptors combine colors and edge descriptors, are suitable for a large image database, and also maintain high accuracy even in the presence of deformations, noise, and smoothing. Even though FCTH and CEDD both use the same color information, FCTH uses more extensive edge descriptors and hence is less compact than CEDD (CEDD has 6 edge descriptors, whereas FCTH has 8). JCD is a descriptor that combines CEDD and FCTH. ACC counts how often a color appears in its immediate neighborhood instead of counting the number of pixels associated with each color. As ACC is based on colors, it is robust to rotations.

2.2 Preliminary Work

Preliminary work was done to determine the performance characteristics of the existing systems from section 2.1. To carry out the experiments, each of the images was modified to produce 35 additional images using the transformations mentioned in section 4.4 with two exceptions: the images for this experiment were only rotated every 45 degrees, and none of the logo images were shrunk to insert into a colorful background. Figure 2.2 shows transformed images generated from a single Instagram image. These transformed images were then used as query images to determine the ability of these systems to retrieve the correct logo image in the presence of transformations. For this, the data set was divided into two categories: With News Logos, and With General Logos. These categories are described in detail in the next two sections.
































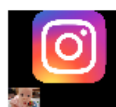




Original	Rotate 45°	Rotate 90°	Rotate 135°	Rotate 180°	Rotate 225°	Rotate 270°
						
Rotate 315°	LLQ	LRQ	ULQ	URQ	Enlarge 1.5x	Enlarge 2x
						
Stretch x	Stretch x	Stretch y	Stretch y	Flip	Flop	Negate
						
Darken	Darken	Lighten	Lighten	Noise	Noise	Noise
						
Crop	Crop	Append Image	Append Image	Append Image	Shrink	Shrink
						
border						
						

Figure 2.2: Different transformations of Instagram Logo

2.2.1 With News Logos

Figure 2.3 shows the non-transformed (or original) news logos that were downloaded from the InVid website. To carry out the performance comparison of LIRe image descriptors, InVid, and Google logo detection, they were tested on transformed news

logos. Figure 2.4 shows the overall performance comparison of LIRe, InVid, and Google Logo detection API on the news logos. The best performing image descriptor was ACC with LIRe performance ranging between 45.139% and 51.667%. This compares to Google’s performance of 51.11%. The descriptors of LIRe and Google logo detection API out-performed InVid. InVid was able to get an accuracy of only 14.306%. Even though InVid was specially designed for news channel logo detection, it performed poorly compared to the other two. The accuracy using the LIRe ACC descriptor and Google logo detection were very close to each other. Thus, it was decided to perform further experiments on a different dataset using LIRe and Google logo detection.



Figure 2.3: News logos

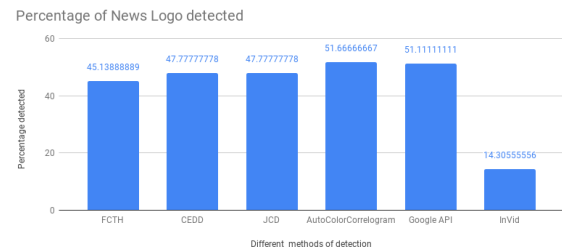


Figure 2.4: Overall Performance on News Logos

2.2.2 With General Logos

Due to the poor performance of InVid on news logos, even though it is designed to work with news logos, it was decided not to carry out further experiments with InVid. Further experiments were necessary to compare LIRe and Google logo detection on a

different dataset. For this experiment, a performance comparison of four LIRe image descriptors and the Google logo detection was done on brand logo images. For this, the data set was divided into two categories, logos without text and logos with text. These categories are described in detail in the next two sections.

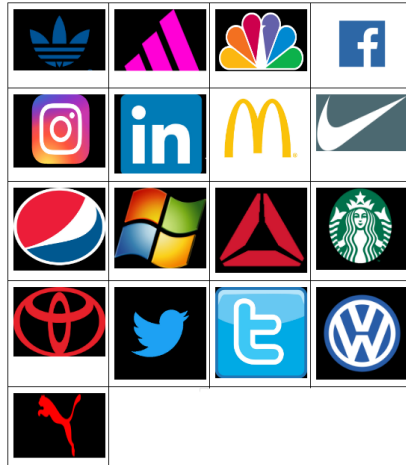


Figure 2.5: Logo images without text on it

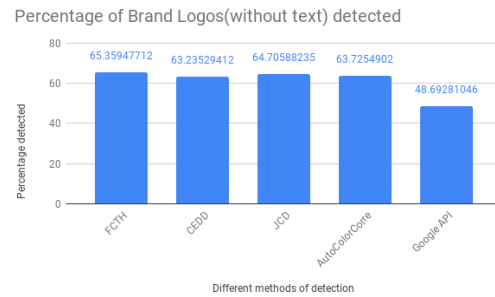


Figure 2.6: Percentage detection of brand logos without text on it

Logos without text

Figure 2.5 shows 17 original images without text on them that were transformed to form 612 query images. Some of those images were obtained by manual cropping of logos with text on them. To make the identification of a logo system a little harder for LIRe, an additional 6984 images were added to the database to make a database containing 7,001 images and were indexed using the top 4 LIRe image descriptors. Among the 7,001 images, 17 images were the original images, and the rest of the images were non-transformed but different from the original images. For testing the performance of LIRe image descriptors, the query images were searched by the LIRe searcher, and a query image was considered detected if it returned the original logo

image in the first position. For testing the performance of the Google logo detection API, a query image was considered detected if the first identified brand name matches the brand name of the query image. Manual verification was necessary for the Google logo detection API results because it had different textual representations for the same brand name. For example, some of the Nike logos were identified as Swoosh, which is also Nike.

Figure 2.6 shows the overall performance of the top four LIRe image descriptors and Google logo detection on all query images. All of the LIRe image descriptors out-performed Google logo detection. The accuracy of the top 4 LIRe descriptors ranges from 63.235% to 65.36%, while that of Google logo detection is 48.693%. The best performing LIRe descriptor is FCTH, which outperforms Google logo detection by 16.667%.

Logos with text

Figure 2.7 shows the 17 original images with text that were transformed to form 612 query images. For testing the performance of the LIRe, a similar database of 7,001 images were created. However, this time, the 17 images from figure 2.5 were replaced by the images from figure 2.7 in the data set mentioned in section 2.2.2. The LIRe image descriptor and Google logo detection API were evaluated using the same method as mentioned in the section 2.2.2.

Figure 2.8 shows the overall performance of the LIRe image descriptors and Google logo detection API on all query images. This time, Google logo detection outperformed all the top 4 LIRe image descriptors. The top 4 LIRe descriptors showed the accuracy decrease in the range 1% to 8% compared to their accuracy on logos without text. The highest accuracy drop, among the top 4 LIRe descriptors,

was seen in FCTH, which heavily relies on the textures/edges present in the image. As texts were added to the images, its texture content increased and also may add more colors. In addition to that, feature extractors that rely on texture involving edge directions usually cannot tolerate rotations greater than 15 degrees. These are the reasons for the big drop in the accuracy of FCTH as it relies on colors and heavily on edge directions.

It can easily be observed from the figure 2.8 that the Google logo detection accuracy increased by over 23% compared to its accuracy on logos without text (as shown in the figure 2.6). Thus, it became clear that Google logo detection was using textual information available on the image along with optical character recognition to aid in logo retrieval. This is because when the text was removed, Google's logo detection performance decreased drastically.



Figure 2.7: Logo images with text on it

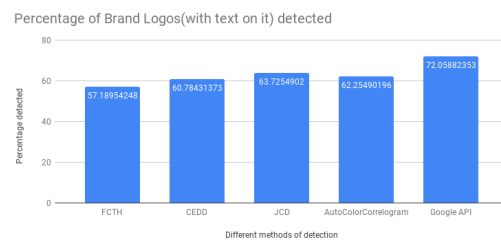


Figure 2.8: Percentage detection of brand logos with text on it

2.3 Motivation

From the preliminary analysis, it was also observed that even though the original images were detected correctly, most of the transformed images were undetected or detected incorrectly. In addition to this, it became obvious from the experimental results that the current existing logo detection systems rely mostly on textual features present in an image. Even though the top four LIRe detected rotated images properly, it was unable to do so for color shifted and translated images. This was the motivation to carry out this thesis work using machine learning techniques.

Chapter 3

BACKGROUND

Section 3.1 introduces the terminology and some of the core algorithms and approaches that are used in this thesis. Section 3.2 gives an overview of some of the closely related works.

3.1 Algorithms and terminology

In solving the problem of logo recognition in this thesis, I utilized convolution neural networks [26] and auto-encoder architectures to construct compressed representations of images containing logos. A brief description of the algorithms and terminology that is used is given in this section.

3.1.1 Convolution Networks

Convolution networks are feed-forward neural networks where the input of each convolution layer is the output from the previous layer. Figure 3.1 shows the general working mechanism of the convolution layers. It consists of a kernel (also called filter) that slides over the input feature matrix (i.e. pixel value of the image in case of this thesis) and calculates the sum of the products to output a new feature matrix. Boxes with arrows indicate the formation of the upper-left element of the output matrix by applying the kernel to the corresponding upper-left region of the input matrix.

The values inside the kernels are the weights of the neuron. These are the weights that are learned during the training process by the technique of backpropagation using gradient descent. The learning process is affected by various parameters, which are also known as hyperparameters. Choosing the best hyperparameters setting, however, is an open problem and one of the challenging aspects of the machine learning field. Some of the well-known hyperparameters are learning rate, weight decay, and momentum. The learning rate controls how large of a step to take in the direction of the negative gradient or towards a minimum loss value [27]. The parameters weight decay and momentum determine how the learning rate should change during the learning process.

For an input of size $N \times N$, and a convolution layer with filter of size $F \times F$, padding \mathbf{P} and stride \mathbf{S} , the size of the output is given by the following:

$$\left\lfloor \frac{N + 2P - F}{S} + 1 \right\rfloor \times \left\lfloor \frac{N + 2P - F}{S} + 1 \right\rfloor \quad (3.1)$$

For a stride of size \mathbf{S} , \mathbf{S} pixels are skipped after every convolution operation in both horizontal and vertical directions.

3.1.2 Activation Functions

Activation functions are usually present(or added) after each convolution layer. The purpose of adding an activation function is to introduce non-linearity to the training process. There are various types of activation functions. ReLu [28] is one of the popular activation function because it helps to avoid the problem of vanishing gradients [29], and also helps the network to train faster.

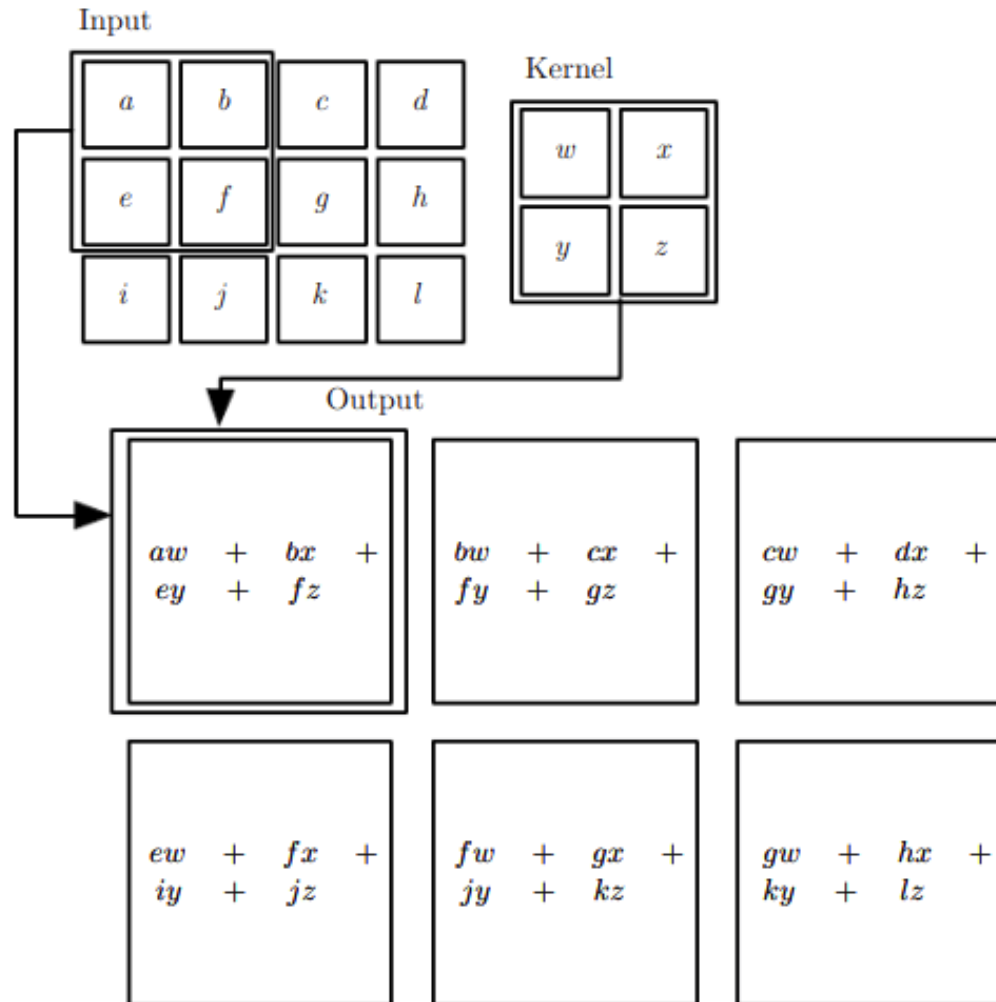


Figure 3.1: General working mechanism of convolution networks at a given layer

3.1.3 Max-Pooling Layers

Max-Pooling is one of the popular techniques for dimension reduction. This technique divides the feature representation matrix into non-overlapping rectangular regions and then takes the maximum value in each of those regions.

They are optional non-linear down-sampling layers that are usually added after an activation function is applied to the convolution layer. It reduces the dimension

of the input features (i.e. feature representation for the input image in the context of this thesis), the number of parameters to be learned, and the computation time. It can also make the model more robust by reducing the possibility of overfitting.

Figure 3.2 shows how the max-pooling layers works. The matrix on the left-hand side of the figure is the output from the previous layer (or some input layer), and the downsampled matrix on the right side is the output after applying max-pooling layers with a filter size of 2x2 and a stride of 2. The elements in the output matrix represent the maximum value from each of the 2x2 disjoint cells from the previous layers, and hence the name max-pooling.

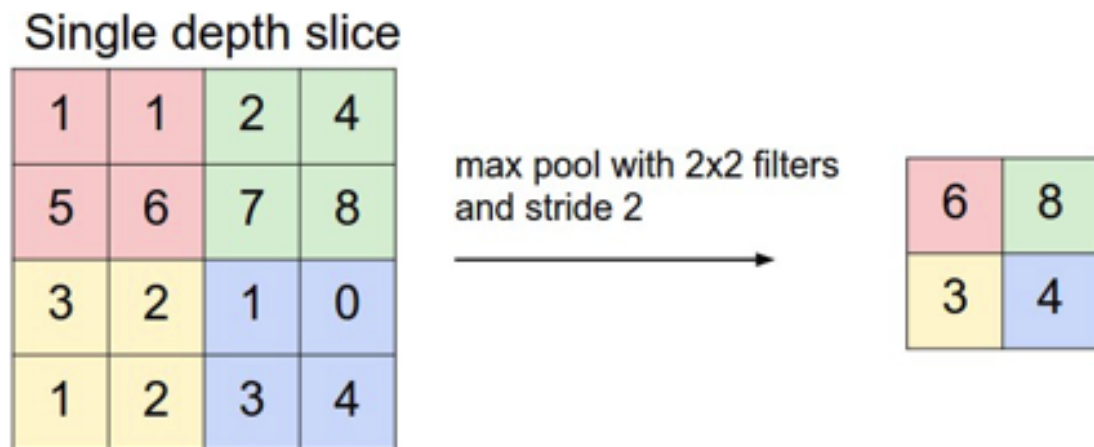


Figure 3.2: Working mechanism of max-pooling layers

3.1.4 Auto-encoders

An auto-encoder is a type of deep learning model architecture that reduces the dimension of the input feature space and facilitates self-supervised feature learning by a reconstruction process. It removes noise present in the input data, if any, and reconstructs features that are exact or very close to the actual input representation. In the process of reconstruction, it generates compressed low dimensional features

at the middle layer. Ideally, compressed features are representative of the input features. Figure 4.6 shows the general working mechanism of an autoencoder. It takes a distorted input image and reconstructs the original image. In the process of reconstruction, it learns to generate latent feature representations \mathbf{z} , as shown in the figure, such that the original and the distorted image have similar representations.

3.1.5 Training Set, Validation Set, and Test Set

Training data is a part of the total dataset used in training a machine learning model. A model trains better if the training set is a good representative of the total data set.

A validation set is useful for the verification of the model under training after every epoch. When the model has seen all the examples in the training data set, an epoch is considered complete. The statistics from processing the validation set with the model are used to estimate the generalization performance of the model. If the validation set performance is not increasing or is not meeting some user-defined expectations, the training process may be restarted, which may also involve the manipulation and tuning of hyperparameters.

Finally, after the model is trained, a test set is used to estimate the performance of the trained model on unseen data. In other words, it is intended to provide an unbiased estimate of the generalization power of the model of interest.

3.1.6 Loss Functions

The loss function is a predictive indicator of the training process. At any point in the training process, the deep learning model predicts an output for the given input such that the difference between the predicted and expected output can be calculated using one of the several loss functions or their combinations. The prediction at any point is

based on the weights assigned to each of the neurons in the network. The amount of error at the current iteration is used to update those weights by a backpropagation technique that uses gradient descent (i.e. decrease the slope of the error) to reduce the error in the next iteration.

3.1.7 Training Loss & Validation Loss

Training loss measures how well the machine learning model or the learning algorithm learned from the training data while the test error measures its generalization power. Lower the training and the test error, better the learning. An ideal model must have low training error and low test error.

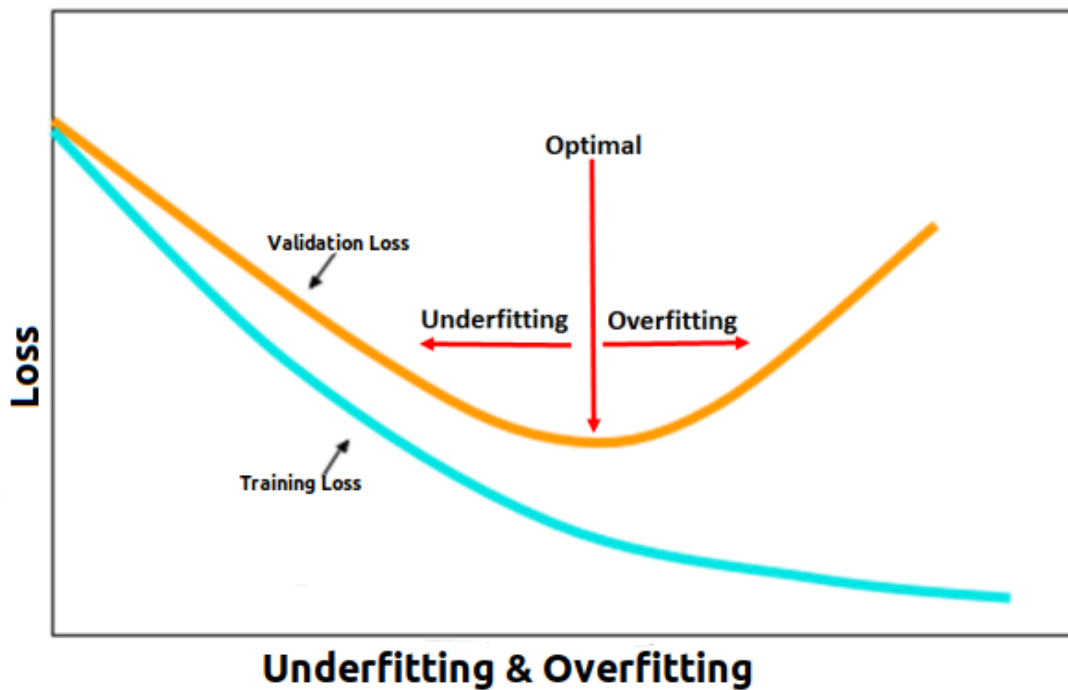


Figure 3.3: Overfitting & Underfitting

3.1.8 Overfitting & Underfitting

Overfitting is a situation when a model fits to the training data to such a large extent that it negatively impacts the generalization performance of the model on unseen data. Underfitting is a situation when a model does not fit properly to the training data as well as is unable to generalize to unseen data. A model is said to have properly learned if there is no underfitting and overfitting. Figure 3.3 gives a general overview of the overfitting, optimal, and underfitting conditions.

3.1.9 k-means Clustering

The K-means clustering algorithm [30] divides a given data set into k nonoverlapping clusters where each cluster consists of similar data. The similarity between data is calculated based on some distance metrics. Each of the clusters is representative of the centroid of all the data points in the cluster.

The centroid value of the cluster updates every time a new datum gets assigned to the cluster. The use of K-means clustering makes the retrieval system faster because the necessity of finding similar data for a given datum by computing the similarity scores with every other data in the data set is replaced by computing the similarity score of the data with the centroid of the k -clusters. In theory, the retrieval process should be n/k times faster, given that the data is equally divided among k clusters where n is the number of data in the data set.

3.2 Related work

Extracting meaning features from images plays a very important role in the image retrieval process. Over the years, various feature extractors have been developed. Some

of the most popular feature extractors for image retrieval are SIFT [32], Speeded-Up Robust Features (SURF) [33], and Bag of Words (BoW) [34, 35, 36] which have been used in various image retrieval tasks [31, 34]. Flexible Image Retrieval Engine (FIRE) [41], which combines text and color features, reported performance improvement of a CBIR system in medical images compared to their previous work [42], which only used color features. Lucene Image Retrieval (LIRe) [37], introduced in 2008, provides support for multiple feature extractors for images and uses LUCENE indexing for faster retrieval of images. The work by Markonis et. al. [43] uses MapReduce for speeding up the indexing and retrieval process. Several works have been done using LIRe. The work by Chen et. al. [40] combines LIRe and SURF for a product identification task of a CBIR system on the Fribourg Product Image Database (FPID) that contains more than 3,000 pictures of consumer products. In addition to that, LIRe is also integrated with Apache Solr in the research work [38] to support large scale visual data. The research work [39] by de Oliveira et. al. uses LIRe to support content-based video indexing and retrieval. Even though a lot of works have been done in increasing the accuracy and speed of image retrieval using the hand-crafted features, none of them has been able to solve the semantic gap problem.

Deep learning has emerged as a promising technique for overcoming the semantic gap problem. The research works like ImageNet [44], VGGNet [46], and GoogleNet [45] which uses deep learning architecture have shown that narrow deeper CNNs achieve significant improvement in image classification, detection, and localization task as compared to wider shallow CNNs. VGGNet has simple architecture and yet outperforms ImageNet, and GoogleNet in terms of single-network classification accuracy. VGGNet consists of a stack of convolutional layers followed by three Fully-

Connected (FC) layers. The first two have 4096 channels each, while the third has 1,000 channels for classification of 1000 classes. VGGNet uses two 3x3 convolution layer instead of one 5x5 convolution layer and three 3x3 convolution layers instead of one 7x7 convolution layer to make the decision function more discriminative by introducing more non-linearity. It also helps to decrease the number of parameters. For example, using three 3x3 convolution layers instead of one 7x7 convolution layer decreases the parameters to be learned by 81%. This inspired the use of VGGNet architecture in creating an autoencoder for this thesis work.

In the thesis work by Singh [47], annotated images are segmented and trained on eight classes using supervised [14] deep learning to generate an 8-bit binary representation of the image where each bit corresponds to the class present on that image. This method of representing each type of image by one bit is not scalable because it needs n number of bits to represent n different types of class. The research work [8] investigates the use of transfer learning using ImageNet for generating hash codes such that similar images produce similar hash codes while the research work [48] uses AlexNet for transfer learning, and represents each of the class labels using the binary representation for supervised learning of binary hash codes. As both of them learn binary hash codes in a point-wise approach, they are highly scalable with data size. Even though both of the research works [8, 48] works well with other deep standard architectures, and also scales well to the data size, they fail to do the same if the number of classes increases. Thus, they are not suitable for a task where the number of classes may increase. Similarly, the research work [52] uses VGGNet architectures to help prevent the registration of trademarks similar to the already registered trademarks. It makes use of transfer learning using two different architectures of VGGNet: VGG19v, and VGG19c. The VGG19v is trained

on conceptual similarities of trademarks, and VGG19c is trained on visual similarity organized according to experts from the IP office. The results from both the networks are combined using the Inverse Rank Position [53] to generate the final predictions. These works are in the context of supervised learning and are not feasible when the dataset does not have labels.

Deep Learning also finds its application in self-supervised learning of the feature representation of the images. The research work [55] introduces that training on patches of raw images using deep belief networks improves retrieval accuracy than training on image features. As it trains on raw images, there is a high probability of retrieving images with similar edge patterns, whether or not they are semantically similar. It uses deep autoencoders formed by stacking Restricted Boltzmann Machines (RBMs) to generate binary hash codes. However, using RBMs for autoencoders are complicated because they need to be trained with a contrastive divergence learning algorithm. It has also been found that using a deep autoencoder built by stacking convolutional neural networks has comparable performance for generating binary hash codes [21].

The research work [56] uses autoencoder to detect irrelevant image blocks, in medical images, by analyzing an error histogram. It makes use of shallow architectures and only works on images that are free from noise. Hence, it cannot be used for detecting logos from noisy (or transformed) images. The research work [57] applies radon transformation to an image to generate projections at some angle. It then trains autoencoder on those projections to generate barcodes and shows that deep autoencoder achieves better accuracy than shallow networks. The research work [50] uses autoencoders to generate sparse binary features by using a binary relaxation at the encoder layer rather than thresholding to produce 0s and 1s. While the

aforementioned works using autoencoders are mostly focused on generating binary hash code or barcodes, this thesis work focuses on improving the quality of the latent representation using an autoencoder.

The research work [51] uses convolutional autoencoder for detecting defective logos from the manufacturer on the surface of mobile phones. The defective logos have a slight variation from the production standard logos: varying intensity of illumination, a different angle of rotations, and some Gaussian noise. Even though it is just a slight variation, and also the number of defective logos are few, detecting them is very important because they can jeopardize the reputation of a company. They use an autoencoder to reconstruct the input image, and find the abnormal regions by comparing with the company standard input image based on threshold generated using an OSTU [51] algorithm. It, however, is only applicable to simple transformations: Gaussian noise, slight variation in rotations, and variation of illumination intensity. This thesis work uses convolutional autoencoder to detect logos in the presence of complex transformations.

Deep neural networks require a large amount of training data to learn properly. Gathering training data is tedious work, and in some cases, it is very difficult to obtain sufficient training data. Data augmentation is one of the methods to overcome such a shortage of training datasets, and it also helps to avoid the problem of overfitting. The paper [58] introduces basic transformations to create a dataset of 1 million images from just 15,200 images. It was noted in research work by [60] that rotating training images increased CNN performance. The research work by [59] [61] reports that introducing cropped images in the training dataset increases accuracy.

Convolutional autoencoders are excellent in extracting meaningful features from unlabeled data [49]. Most of the work using autoencoders has been to generate binary

hashcodes from images, find irrelevant image patches, and generate barcodes from images. All these works have achieved state of the art accuracy. However, there have been no works done to enhance the quality of the compressed representation generated using an autoencoder. Motivated by the simple architecture and high performance of VGGNet in supervised learning, and effectiveness of autoencoders in extracting meaningful features from unlabeled data, this thesis takes advantage of both worlds to improve the quality of the compressed feature generated by the autoencoder.

Chapter 4

METHODS

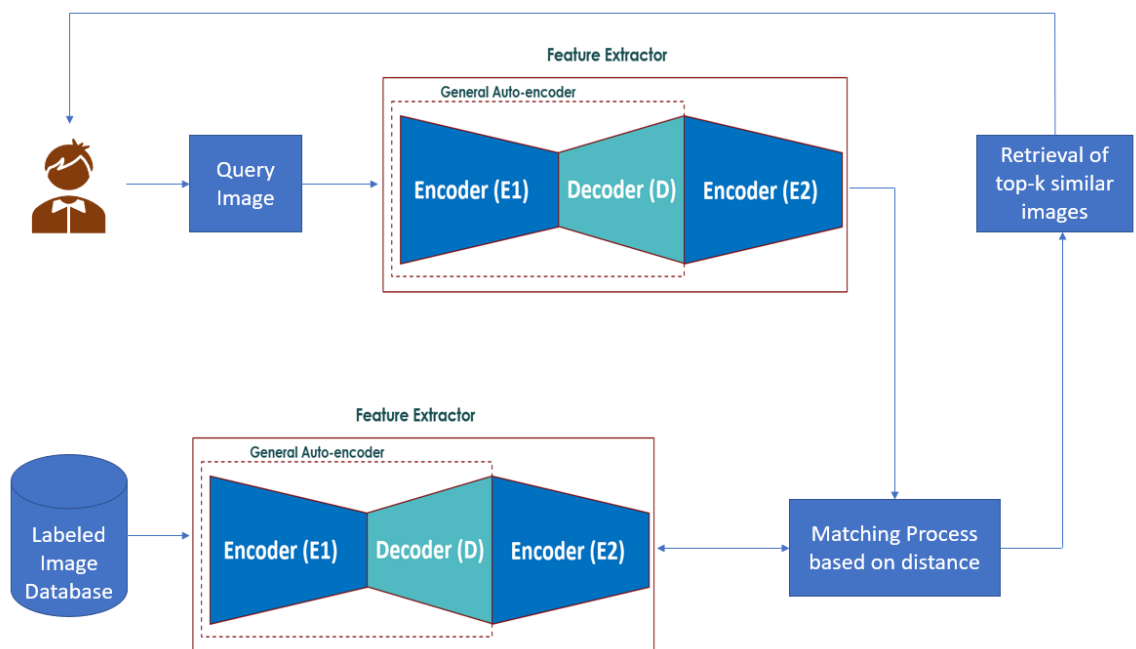


Figure 4.1: Architecture of the proposed CBIR system

4.1 General System Architecture

Figure 4.1 shows the overall architecture of the novel CBIR system. The autoencoder is not learning just a simple mapping of input - input. It is learning to map transformed input image, not just limited to the Gaussian noise, to the non-transformed or original input image. The other innovation is that we reuse the encoder **E1**

by applying it to the output of the autoencoder to reduce the output image to a searchable vector. It consists of two main parts - a **feature extractor** and a **matching process**. The feature extractor block extracts features from the query logo image inserted by the user and also from labeled original (i.e. non-transformed) images in the database. The matching process computes a similarity score between the extracted features. The system then retrieves the top-k closest relevant results based on the similarity score based on some distance metrics. The details about the feature extractor and the matching process are given in the following subsections.

4.1.1 Feature Extractor

The novel feature extractor consists of two parts - a general denoising auto-encoder and an encoder. The encoder, **E1**, produces a compressed representation from the input image and the decoder, **D**, tries to reconstruct the original logo image from the compressed representation of an arbitrary image containing the logo. The reconstructed image may or may not look similar to the original image. The next encoder **E2** takes the encoding technique of a general denoising auto-encoder one step further. **E2** takes the output of **D** and generates a compressed latent representation of the reconstructed image. The following subsections give architectural details about each component of the feature extractor module.

General Denoising AutoEncoder

The general denoising auto-encoder, as shown in the dotted square box in figure 4.1, has two parts- **E1** and **D**.

The architecture of the **E1** is inspired by the VGGNet[46] architecture. The red box in figure 4.2 shows the internal architecture of the **E1**. There are 10 convolution

layers. Each of the convolution layers has a kernel of size 3 x 3 with relu as the activation function and 64, 64, 128, 128, 256, 256, 512, and 512 filters respectively. There are 5 Max-Pooling layers and each of them have a kernel of size 2 x 2 and strides of size 2 x 2. The **E1** takes the input image and generates a feature at the Compressed Representation layer as mentioned in the red box of figure 4.2.

The **D** is shallower than the **E1**. The blue box in the figure 4.2 show the internal architecture of the **D**. It has 5 convolutional layers, each having a kernel size of 5 x 5 with relu as activation function except the last convolution layer which uses sigmoid as an activation function, and the number of filters in each of the convolution layers are 512, 256, 128, 64 and 64 respectively. There are 5 Max-Pooling layers and each of them have a kernel of size 2 x 2 and strides of size 2 x 2. The **D** takes the compressed representation generated by the **E**. The **D** then generates a reconstructed image at the Conv5-3 layer of the Decoder part as shown in figure 4.2.

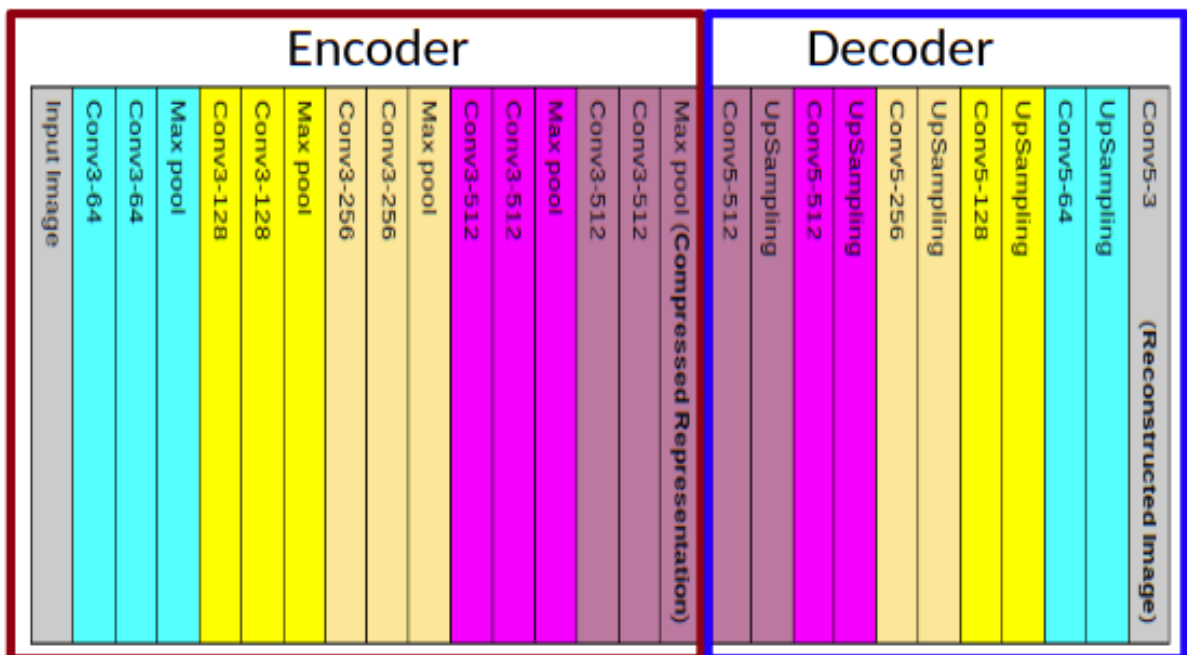


Figure 4.2: Internal Architecture of the AutoEncoder

Encoder (E2)

The encoder **E2** has the same architecture as that of the **E1**, and same weights. The only difference between them is that **E2** encodes reconstructed images whereas **E1** encodes input images.

4.1.2 Matching Process

The *Matching Process* block computes the similarity between the query image and images in the database and finds the top-k closest images from the database as relevant images. One way of finding the relevant images to the query image is by exhaustively computing the similarity score of the query image with each of the images in the database and returning images with the highest similarity score to the given query image. Given n images in the database, and the computing the similarity between a query image and an image from the database takes unit time, the total time to find k relevant images using exhaustive search takes n units of time, where $1 \leq k \leq n$. This may not be a problem with a smaller number of images in the dataset but will cause severe performance issues as the size of the database increases. For example, consider there are only 5 images in the database, and finding a similarity score between an image from the database with the query image takes 1 second. Then the total time to find the closest or relevant images takes 5 seconds. This might not be a problem as the user can wait 5 seconds. If the dataset contains 10,000 images, waiting for 10,000 seconds is not feasible. This makes the exhaustive search not a good option for datasets with a large number of images as the time complexity to find the relevant images increases linearly.

The other way of finding relevant images is through the use of an approximate

nearest neighbor finding techniques. K-Means[62] is one of such techniques that divide n images (or samples) into k sets or clusters such that the mean square error between the sample to the nearest cluster centroid is minimized. It is based on the hypothesis that given a sample, its nearest neighbors reside in the same cluster. Figure 4.3 shows k clusters formed from n samples where the clusters may or may not be of the same size. The cluster formation using optimized k-Means can run in linear time [63]. Once the cluster is formed, there is no need to re-run the cluster formation again unless a new image has been added to the dataset. During the time of searching the relevant images for a given query image, it is only required to find the closest clusters to the given query image using the distance metrics between the query image and the k clusters centroid. Given that images are equally divided into clusters, the time complexity to find the closest cluster will be reduced by a factor of k (i.e. n/k). Then, the time to find the k relevant images to the query image from the closest cluster takes $(n/k) + k$ units of time. This is an improvement over exhaustively searching for the closest images.

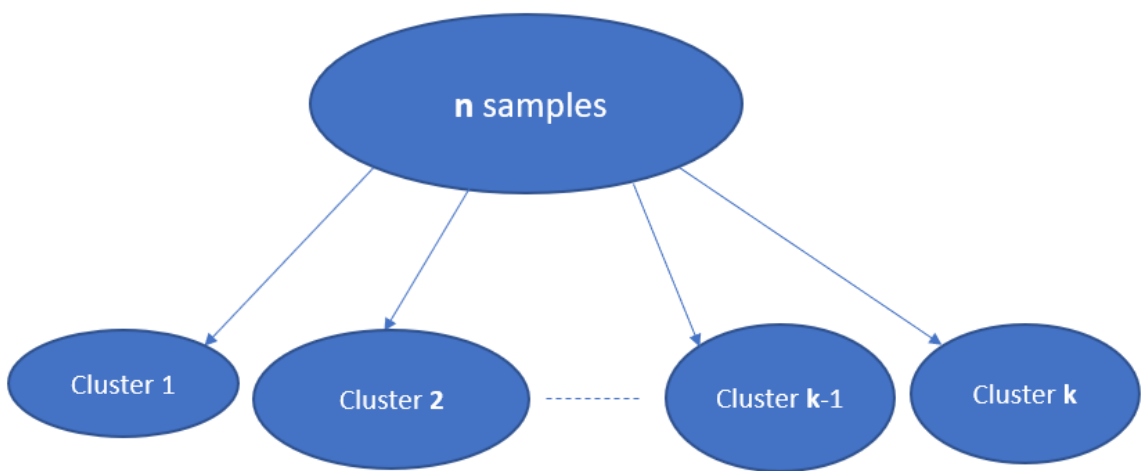


Figure 4.3: clusters formed using K-Means clustering

The proposed architecture is equipped with multilevel clustering and searching, and both of them help to make the retrieval systems faster than the previous two methods. It reduces the problem of finding similar images from the database to the given query image to the log of the total number of images in the database. They are discussed in detail in the following sections.

Multilevel Clustering

The multilevel clustering treats each of the clusters generated using the K-Means clustering scheme as separate data sets, and recursively generates sub-clusters based on the branching factor until a certain desired level is attained or a minimum threshold is reached.

Algorithm 1 shows how the multilevel clustering works. It takes in the parameter `desired_level` that indicates the level of the tree to form, the database of the original images, and the root of the tree. As long as the value of the `desired_level` is greater than 1, and the size of the current cluster is greater than some user-defined threshold value, it finds the optimum number of clusters at the current level, generates the optimum number of clusters using KMeans clustering algorithm, and adds those clusters to the current level of the tree. The process is recursive and is applied to each of the clusters at every level until it satisfies the desired level and the threshold values.

Figure 4.4 shows the multilevel clustering with a branching factor of 2 up to level 3. In the given figure, the multilevel clustering starts at level 1 with a cluster named A. At level 2, two clusters A1 and A2 are formed from the data in cluster A. The sum of the number of data in cluster A1 and A2 is equal to the number of data in cluster A. Similarly 4 clusters are formed at level 3. Clusters A11 and A12 are formed

Algorithm 1 MultiLevel Clustering Algorithm

```

1: procedure MultilevelClustering(desired_level, data, root_of_tree)
2:   root_of_tree.data  $\leftarrow$  data
3:   if desired_level > 1 then
4:     num_of_clusters  $\leftarrow$  findOptimumNumberOfClusters(data)            $\triangleright$ 
       Generates # of clusters
5:     clusters  $\leftarrow$  KMeans(num_of_clusters, data)                  $\triangleright$  Form clusters
6:     Add clusters to tree at this level
7:     for cluster in clusters do
8:       if desired_level > 1 & size_of(cluster) > THRESHOLD then
9:         MultiLevelClustering(desired_level - 1, data, reference_to_cluster)
10:      end if
11:    end for
12:  end if
13:  return root_of_tree
14: end procedure

```

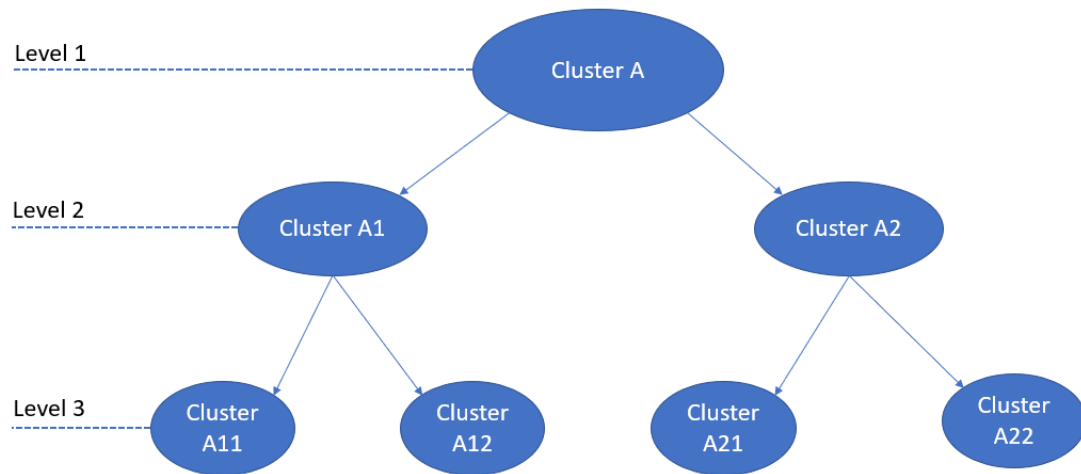


Figure 4.4: Multilevel clustering up to Level 3 and branching factor of 2

from cluster A1 and clusters A21 and A22 are formed from cluster A2. The sum of the number of data in cluster A11 and A12 is equal to the number of data in cluster A1. Similarly, the sum of the number of data in cluster A21 and A22 is equal to the

number of data in cluster A2.

4.1.3 Multilevel Searching

The *Matching Process* block is also equipped with multilevel searching to search faster for the retrieval. It is faster because the number of comparisons to find the closest images to the given query image is reduced by a factor of the branching factor at each level. This makes the retrieval time to the log-normal to the total number of images in the dataset. The multilevel searching can only be performed after the clusters have been formed using the multilevel clustering. For example, if a multilevel searching is to be performed on the clusters formed by the multilevel clustering as shown in figure 4.4, the searching first starts from Cluster A at level 1 which is the root for this tree. Then, it computes the distance of the query image with the centroid of all the clusters at level 2 (i.e. Cluster A1 and Cluster A2) and selects the cluster having the smallest distance with the query image. Now, if the user-defined the desired level of search up to level 2, and the Cluster A1 had the smallest distance with the query image at level 2, the top-k relevant images from the cluster A1 will be returned. If the user-defined desired level of search is greater than level 2, a similar step is repeated recursively until the desired level has been achieved.

Multilevel searching is also a recursive algorithm as shown in the algorithm 2 . Once the multilevel clusters of level L have been formed, the multilevel searching find the cluster based on the user defined parameters. The parameters consists of `desired_level` that indicates the level up to which the search is to be performed, the database of the original images, the level from which the search is to be started (it is always 1 at the start of searching process), the query data, and the root of the tree formed using the multilevel clustering. The algorithm recursively finds the clusters

with the centroid value close to that of the query image at each level based on a distance metric until a desired level has been achieved and returns the top-k relevant images from the closest cluster. Even though the desire level has not been achieved but the current closest cluster to the query image does not have sub-clusters, the algorithm stops and returns the relevant images from the current closest cluster.

Algorithm 2 MultiLevel Searching Algorithm

```

1: procedure MultiLevelSearching(desired_level, data, start_level, query_data, root)
2:   if desired_level == 1 or root.getNumberOfChildren() == 0 then
3:     return root
4:   end if
5:   for cluster in start_level.get_clusters() do           ▷ start_level is 1 at start
6:     resulting_cluster ← distance_min(query_data, cluster) ▷ get cluster with
    centroid having min distance with query data
7:   end for
8:   if desired_level > 1 and root.getNumberOfChildren() > 0 then
9:     root ← root.child[indexOfChildWithMinDistance]
10:    return MultiLevelSearching(desired_level-1, resulting_cluster.getDatas,
    start_level+1, query_data, root)
11:  end if
12:  return root
13: end procedure

```

4.1.4 Architecture for more compression

The architecture of the autoencoder with more compression is similar to the one shown in figure 4.2. The only difference is that it has three additional layers between the last layer of the encoder and the first layer of the decoder of figure 4.2. The additional layers consist of two dense layers and a dropout layer. The dropout layer sits between the dense layers. The first dense layer, following the encoder, compresses the feature to the desired length by varying the number of neurons. The second dense layer makes sure that the output from the dropout layer is converted to the

proper feature-length to feed as input to the decoder. The dense layer consists of a user-defined compression factor to further compress the feature coming from the last layer of the encoder.

4.2 Training Architecture

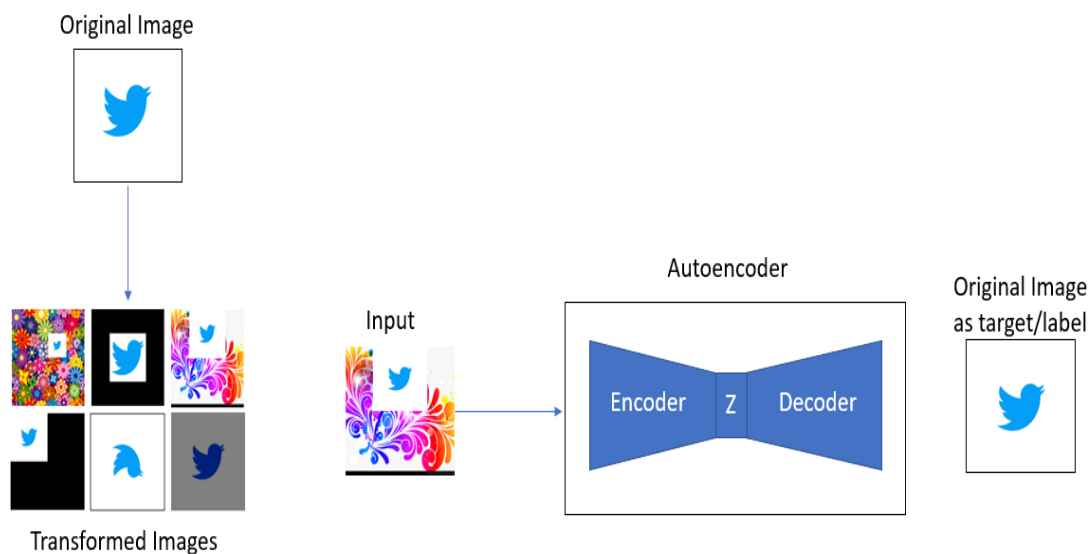


Figure 4.5: Architecture for training autoencoder

The training architecture uses transformed images as the input and original images as targets. Figure 4.5 shows the training architecture for the proposed model using a Twitter logo (the original image). Transformed images consist of rotated, translated, stretched, flipped, flopped, negated, cropped, added border, darken, lighten images, and images with added Gaussian noise. It also consists of logo images inserted into bigger images, and random images appended to the logo images. Details about transformations for generating transformed images from the original images can be found in section 4.4.

As shown in figure 4.5, for training, transformed images are generated from an original image. The transformed images are then selected randomly and fed to the autoencoder as input, and the non-transformed (i.e. original logos) images are set as labels (or target values) rather than using annotations (i.e. Twitter) as labels. For example, if the image being fed to the model is *X.png_TEST_rotate90.png*, the image named *X.png* acts as a label. For example, if the image being fed to the model is *X.png_TEST_rotate90.png* then the image named *X.png* acts as a label.

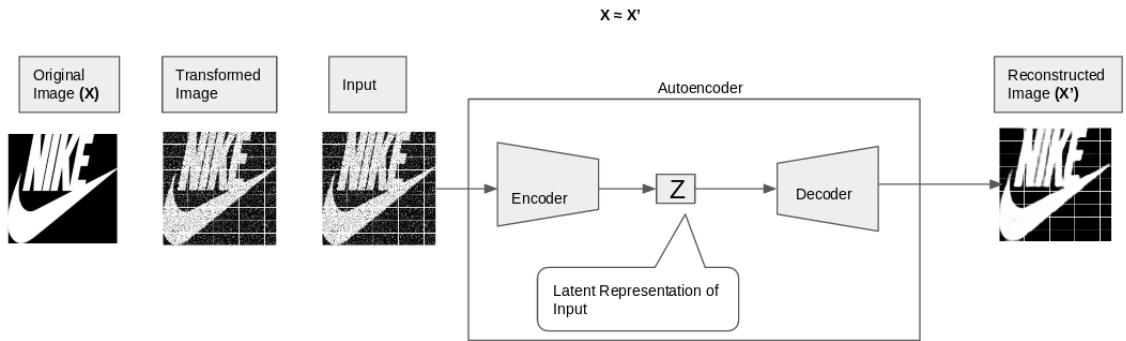


Figure 4.6: Working Mechanism of Autoencoder

As shown in the figure 4.6, during the training process, the autoencoder takes in the transformed image of the original image x , and generates a latent representation z at the encoder layer. The decoder then tries to generate the reconstructed image x' from z . The training process continues as long as the quality of the reconstructed image keeps on improving.

The quality of the reconstructed image is calculated using the mean square error (MSE) loss function. It calculates the pixel wise difference between the original image I^O and the reconstructed image I^R . The equation 4.1 shows the formula for calculating the MSE loss where H and W are the heights and widths of an image respectively.

$$Loss_{mse} = \frac{1}{HW} \sum_{i=1}^H \sum_{j=1}^W \left(I_{ij}^O - I_{ij}^R \right)^2 \quad (4.1)$$

4.3 Datasets

Most of the brand detection software available detects only popular brand logos. This thesis is an effort in creating a detection and retrieval system that can be easily trained and made to work on logos of interest. Thus the data are collected by crawling online sources, primarily mobile app stores, which may or may not contain popular logos. The collected dataset has approximately 13 million images. From the collected dataset, 1900 images were selected randomly and divided into four disjoint subsets: **base500**, **set2**, **set3**, and **set4**. The **base500** dataset consists of 400 crawled images with additional 100 popular logos, while the rest of the datasets consist of 500 crawled images each. To form bigger datasets containing 1000 Images, three subsets; set2, set3, and set4 are combined to produce three sets of 1000 images: **combined_set2_3**, **combined_set2_4**, and **combined_set3_4**. The **combined_set2_3** implies a set formed by combining set2 and set3. The datasets containing 500 original images are called smaller datasets, while those containing 1000 original images are called bigger datasets.

4.4 Transformations

ImageMagick, a free software, is used to generate transformed images. All of the transformed images are still recognizable to the human eye. A total of 101 images, including the original image, are created from a single image. The performance of any deep learning model depends on the data set and may not perform to its potential if

the dataset is not sufficiently large enough to train it. To create a data set that is bigger and has all the important representation of an image, the following mentioned transformations are applied to each of the data sets mentioned in section 4.3.

4.4.1 Rotations

The original image is rotated every 9° to generate transformed images having 9° , 18° , 27° , 36° , 45° , 54° , 63° , 72° , 81° , 90° , 99° , 108° , 117° , 126° , 135° , 144° , 153° , 162° , 171° , 180° , 189° , 198° , 207° , 216° , 225° , 234° , 243° , 252° , 261° , 270° , 279° , 288° , 297° , 306° , 315° , 324° , 333° , 342° and 351° rotations. These transformations creates 38 additional images from single original image. The Figure 4.7 shows images that are generated using rotations.

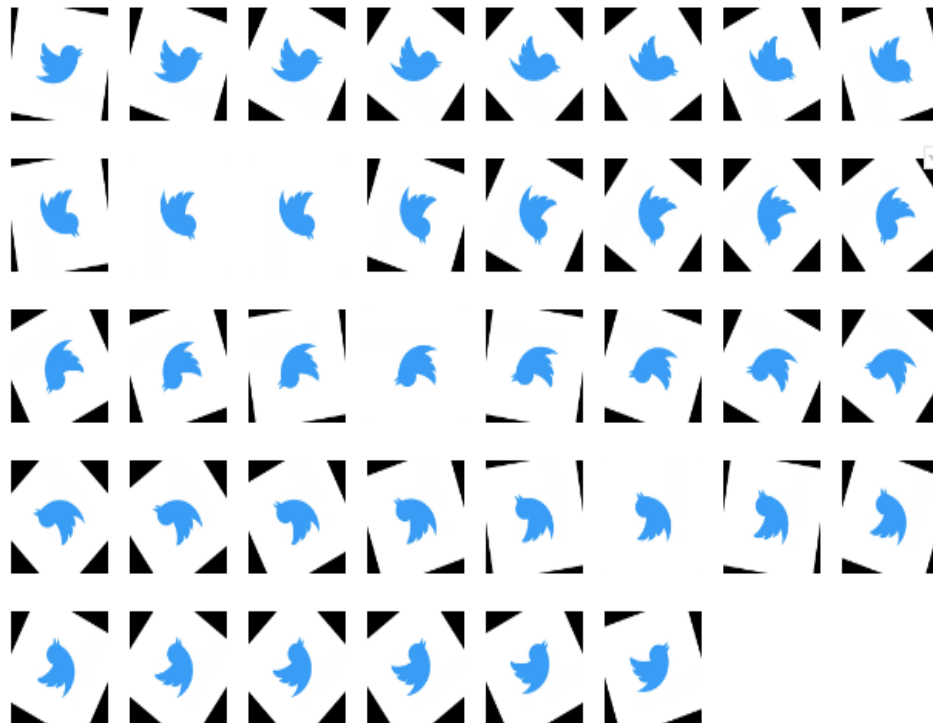


Figure 4.7: Rotated Images formed from the original Image

4.4.2 Translation

This transformation puts the image icon at various locations in the image; the lower left quadrant (LLQ), lower right quadrant (LRQ), upper left quadrant (ULQ), and upper right quadrant (URQ). The figure 4.8 shows images that are translated to LRQ, ULQ, LLQ, and URQ respectively.

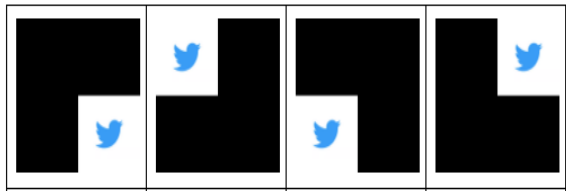


Figure 4.8: Translations

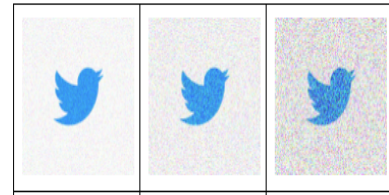


Figure 4.9: Gaussian noise of various levels

4.4.3 Added Gaussian Noise

This transformation adds various level of noise uniformly to the original image. Transformed images generated using various levels of gaussian noises are shown in the figure 4.9.

4.4.4 Stretching

This transformations stretches images by various amounts in x and y directions. The left two transformed logos and the right two transformed logs of the figure 4.10 shows images that are generated using stretching in x and y directions respectively.

4.4.5 Flipping and flopping

Flipping generates a mirror image of the original image while flopping generates a mirror reversal of the original image. The left hand side and the right hand side of

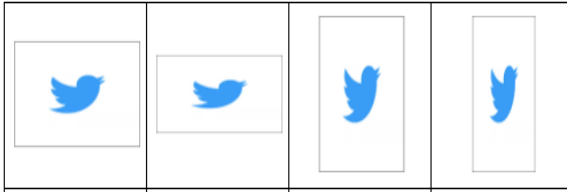


Figure 4.10: Stretching in X and Y directions

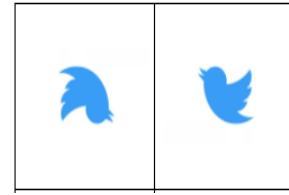


Figure 4.11: Flipped and flopped images

the figure 4.11 shows the flipped and flopped images respectively.

4.4.6 Negate

Negate transformation converts the color present in the image to its respective complementary color. The figure 4.12 shows the negated image that is formed from the original image.



Figure 4.12: Negate

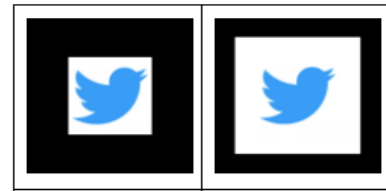


Figure 4.13: Cropped Images

4.4.7 Cropping

It crops the logo images by various amounts. The images generated by using this transformation are shown in the figure 4.13.

4.4.8 Inserting logo into another picture

For inserting an image into colorful background picture, the original image of size 224 x 224 is first reduced to size 56 x 56 and 112 x 112 respectively. Then they are placed



Figure 4.14: Logo Images of size 56X56 inserted into colorful background Image



Figure 4.15: Logo Images of size 112X112 inserted into colorful background Image

at various location of colorful background pictures of size 224 x 224. This creates 35 additional transformed images. The figure 4.14 and 4.15 shows the images of size 56 x 56 and 112 x 112 inserted into the colorful background images.

4.4.9 Add border

It adds various amounts of border to the image. The image generated by using this transformation is shown in the figure 4.16.



Figure 4.16: Added Border

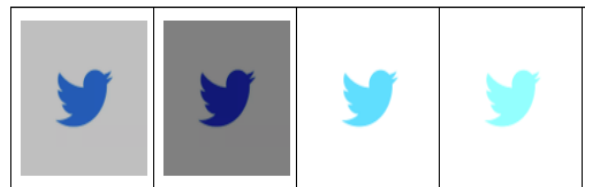


Figure 4.17: Darken and Lighten

4.4.10 Darken and Lighten

These transformations changes the color intensities of the input image. The transformed images generated using the darkening and lightening of the images are shown in the figure 4.17.

4.4.11 Append

Appending inserts random images to the original logo image. The images generated by appending are shown in the figure 4.18.

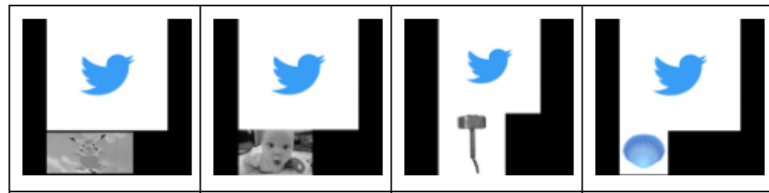


Figure 4.18: Various images appended to original logo Images

4.5 Creating ground truth

The ground truth is created by following a special naming convention - a transformed image name must have the original image name followed by a postfix `_TEST_` and, the type of transformation. For example, if the original image name is *X.png* then the name of transformed image generated using 90° rotation will be *X.png_TEST_rotate90.png*. This naming convention makes the verification of the correctness of the novel retrieval system easy. This is because the verification system only needs to check whether or not the retrieved original image name is contained in the query image name.

4.6 Creating Training Set, Validation Set, and Test Set

The dataset of interest is divided into training, test, and validation set. Training, validation, and test set constitute 60%, 20%, and 20% of the overall dataset respectively.

For making each of these datasets to be as balanced as possible, a special technique as shown in the figure 4.19 is applied. As shown in the figure, this process first takes all the images and converts them into folders. This process makes sure that each of these folders have equal proportions of all the applied transformations. The number of folders is equal to the number of the original images. Then based on the percentage, the images in folders are assigned to training, validation, and test sets. This technique makes sure that each of these sets have equal proportion of all the transformations, and also made sure that they have equal proportions of all the images. The non-transformed images are then added to the training set to prevent accidental addition of these images to the validation and the test set. Table 4.1 shows the number of original, training, test, and validation images for each of the datasets.

Table 4.1: Number of Original Images, Training Images, Test Images, and Validation Images in different datasets used in experiments

Datasets	Original Images	Training Images	Test Images	Validation Images
base500	500	30,500	10,000	10000
set2	500	30,500	10,000	10000
set3	500	30500	10,000	10000
set4	500	30,500	10,000	10,000
combined_set2_3	1000	61,000	20,000	20,000
combined_set2_4	1000	61,000	20,000	20,000
combined_set3_4	1000	61,000	20,000	20,000

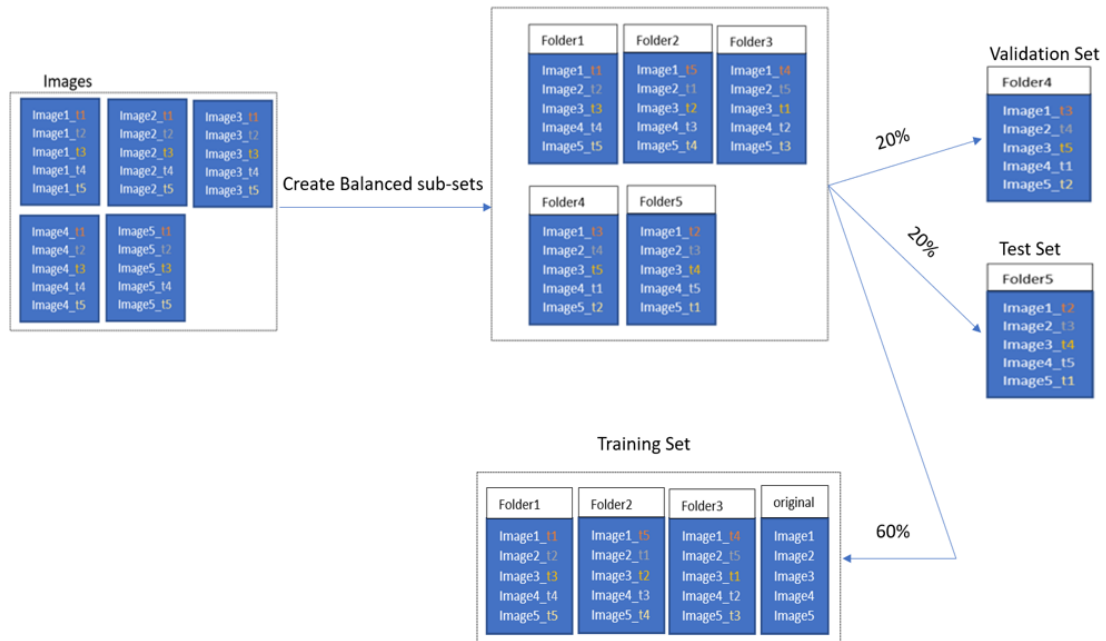


Figure 4.19: Process of creating Training, Validation, and Test Set

4.7 Evaluation

The proposed CBIR system is evaluated based on the percentage of the total query images detected correctly. The query images are mutated forms of an original image. The hypothesis is that, because the system is trained to reconstruct the original image for each of the mutated images, the CBIR system should be able to retrieve the corresponding original image as the first retrieved image for a given query image. Even if the system retrieves the corresponding original image within the top-k position, it is still considered to be relevant. The label matching is performed based on the special naming convention aforementioned in section 4.5. This metric has been chosen because the number of relevant images in the database for a given query image is always one. The accuracy of the system is calculated using the equation 4.2.

$$Accuracy_{@k} = \frac{\text{Number of queries detected at top k}}{\text{Total number of queries}} \quad (4.2)$$

4.8 Creating Baselines

Baselines are necessary to understand the impact of a research work. The baselines that uses the datasets used in this research work are not available. In order to understand the feasibility of the proposed system, 2 baselines using LIRe, 1 baseline using google logo detection, 1 baseline using autoencoder, and 1 using encoder only are created and explained below.

Table 4.2: AUC value for each of the LIRe descriptors for the dataSets of 500 Images

Descriptors	base500	set2	set3	set4	Average
ACC	0.889	0.905	0.909	0.905	0.905
BPP	0.680	0.678	0.680	0.683	0.68
CEDD	0.872	0.883	0.889	0.882	0.882
CL	0.790	0.792	0.799	0.792	0.793
EH	0.692	0.694	0.698	0.695	0.695
FCH	0.763	0.782	0.781	0.777	0.776
FCTH	0.859	0.876	0.881	0.873	0.872
Gabor	0.692	0.702	0.707	0.7	0.7
JCD	0.870	0.884	0.890	0.882	0.882
JCH	0.691	0.692	0.696	0.697	0.694
JH	0.813	0.815	0.820	0.820	0.817
LBP	0.659	0.639	0.640	0.638	0.644
LL	0.623	0.610	0.613	0.610	0.614
OH	0.830	0.850	0.856	0.849	0.846
PHOG	0.684	0.697	0.694	0.692	0.692
RILBP	0.689	0.697	0.699	0.694	0.695
SC	0.767	0.788	0.788	0.789	0.783
SCH	0.831	0.851	0.855	0.849	0.847
Tamura	0.690	0.661	0.664	0.664	0.67

Table 4.3: AUC value for each of the LIRe descriptors for the dataSets of 1000 Images

Descriptors	combined_set2_3	combined_set2_4	combined_set3_4	Average
ACC	0.907	0.905	0.907	0.906
BPP	0.697	0.680	0.681	0.686
CEDD	0.885	0.882	0.885	0.884
CL	0.795	0.791	0.795	0.794
EH	0.696	0.695	0.697	0.696
FCH	0.781	0.779	0.779	0.780
FCTH	0.878	0.874	0.877	0.876
Gabor	0.705	0.701	0.704	0.703
JCD	0.886	0.882	0.885	0.884
JCH	0.694	0.694	0.696	0.695
JH	0.817	0.817	0.820	0.818
LBP	0.639	0.638	0.639	0.639
LL	0.611	0.610	0.611	0.611
OH	0.853	0.849	0.852	0.851
PHOG	0.696	0.695	0.693	0.695
RILBP	0.698	0.695	0.696	0.696
SC	0.788	0.788	0.788	0.788
SCH	0.853	0.850	0.852	0.852
Tamura	0.662	0.662	0.664	0.663

4.8.1 LIRe with individual descriptors

The baseline using individual LIRe descriptors is created in two steps. The first step is to figure out 4 (out of 19) important LIRe image descriptors that perform best on the available datasets. The importance of each of the LIRe descriptors is calculated by using the area under the curve (AUC) score. To find out the AUC score, the problem is treated as a binary classification problem i.e. whether the logo of interest is present in the query image or not. To create the feature vector for the classification purpose, the distance between each of the original image and all possible transformed images is calculated using each of the image descriptors and labeled as ‘1’ if the original image is present in the transformed image and ‘0’ otherwise. The total length of the feature

vector for each pair of an original image and a transformed image is 19 because there are 19 distance values for 19 image descriptors. The AUC scores for each of the datasets containing 500 images are given in the table 4.2. Similarly, the AUC scores for each of the datasets containing 1000 images are given in table 4.3. Observing the results in tables 4.2, and 4.3, it can be seen that the top 4 LIRe descriptors with highest AUC scores are ACC, CEDD, FCTH, and JCD.

The second step is to find the best among the four top-performing LIRe descriptors by actually using them in image identification and retrieval problem. For this purpose, the database of the original images is indexed using each of the 4 best descriptors. After that, for each of the user inserted query image, the distance to the original image is calculated using LIRe's descriptor, and the original images are sorted in descending order of the distance to the query image. Finally, the top-k images are returned as retrieved images using each of the individual top 4 descriptors. The retrieval is considered accurate if the relevant original image is found within the top-k retrieved images.

The accuracy results for each of the datasets containing 500 images are given in the table 4.4. Out of the top 4 performing descriptors, as shown in the table 4.4, ACC outperformed all other image descriptors except for **base500** image dataset at top-10 accuracy. The average accuracy results for datasets containing 500 images, as presented in table 4.4, show that ACC outperformed the other three descriptors. The accuracy results for each of the datasets containing 1000 images are given in table 4.5, which shows that ACC outperformed the other three descriptors for each of the datasets containing 1000 original images. Thus, ACC turned out to be the best among the top 4 performing LIRe descriptors.

Table 4.4: Accuracy comparison using individual LIRE descriptors and MLP on datasets containing 500 images

Modes	base500		set2		set3		set4		Average	
	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10
ACC	0.447	0.552	0.538	0.641	0.541	0.66	0.555	0.652	0.52	0.626
CEDD	0.3959	0.581	0.426	0.602	0.456	0.615	0.437	0.603	0.429	0.6
FCTH	0.362	0.504	0.429	0.561	0.449	0.580	0.427	0.565	0.417	0.553
JCD	0.420	0.571	0.47	0.603	0.482	0.613	0.468	0.608	0.46	0.599
MLP750	0.593	0.725	0.651	0.772	0.652	0.762	0.640	0.763	0.634	0.755

Table 4.5: Accuracy comparison using individual LIRE descriptors and MLP on datasets containing 1000 images

Modes	combined_set2_3		combined_set2_4		combined_set3_4		Average	
	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10
ACC	0.524	0.614	0.533	0.614	0.531	0.623	0.529	0.617
CEDD	0.401	0.563	0.396	0.558	0.408	0.565	0.402	0.562
FCTH	0.408	0.532	0.401	0.524	0.409	0.535	0.406	0.53
JCD	0.447	0.572	0.44	0.572	0.448	0.573	0.445	0.572
MLP750	0.633	0.743	0.626	0.737	0.637	0.744	0.632	0.741

4.8.2 LIRE with Multilayer Perceptron

Until now, each of the best four LIRE descriptors was used individually, and no methods were used to combine the LIRE descriptors for retrieval. Multilayer perceptron (MLP) [64] is a feed-forward artificial neural network with one or more hidden layers. LIRE with MLP makes use of the MLP classifier with one hidden layer to combine 19 LIRE descriptors. The feature vector is formed by combining the distance value of a query image with all the original images using all the descriptors. As the total number descriptors used are 19, the feature vector length is 19. The labels are set as 1 if the query image is the transformed version of the original image and 0 otherwise.

Once the feature vectors are formed, it is still treated as a classification problem,

and the MLP is trained using variations of several neurons in the hidden layers: 5, 10, 20, 50, 100, 250, 500, 750, and, 1000. It is trained using a constant learning rate of 0.001, Relu as an activation function, and adam as a solver or weight optimization, which is a stochastic gradient-based optimizer.

The best accuracy is achieved when using a hidden layer with 750 neurons. The accuracy results for datasets containing 500 and 1000 original images are reported in tables 4.4, and 4.5 respectively. It can be observed from tables 4.4, and 4.5 that using MLP to combine LIRe descriptors significantly increased the retrieval accuracy over just using individual LIRe descriptors for all datasets. This is because of the following reasons.

- Most of the descriptors are robust to lossless rotations: ACC, SC, SCH, Gabor, and Tamura. ACC is also highly robust to noises or large blobs of same color.
- Descriptors CEDD, FCTH, and JCD are robust to scaling and rotation.
- Descriptors color layout and edge histogram are robust to scaling.

Combining all of the feature descriptors using the MLP helped to overcome the defects of LIRe descriptors and making it more robust to rotations, noise, scaling, flips, and flops. The reasoning above is also supported by the categorical accuracy plot for various transformations on datasets containing 500 and 1000 original images as shown in the figure 4.20. The figure provides the necessary evidence that the systems are robust to rotations, flips, flops, shrink, stretch, and appending of images. The worst performance was seen for transformations involving the inserting of logo images on the colorful background (i.e. T112x112 and T56x56). Combining LIRe descriptors using MLP resulted in an average increase in the accuracy by over 12% than just using CEDD, for datasets containing 500 and 1000 original images, which is easily

observable from tables 4.4, and 4.5 respectively. Thus, combining LIRe descriptors using MLP outperformed the technique of using LIRe individual descriptors by a greater margin.

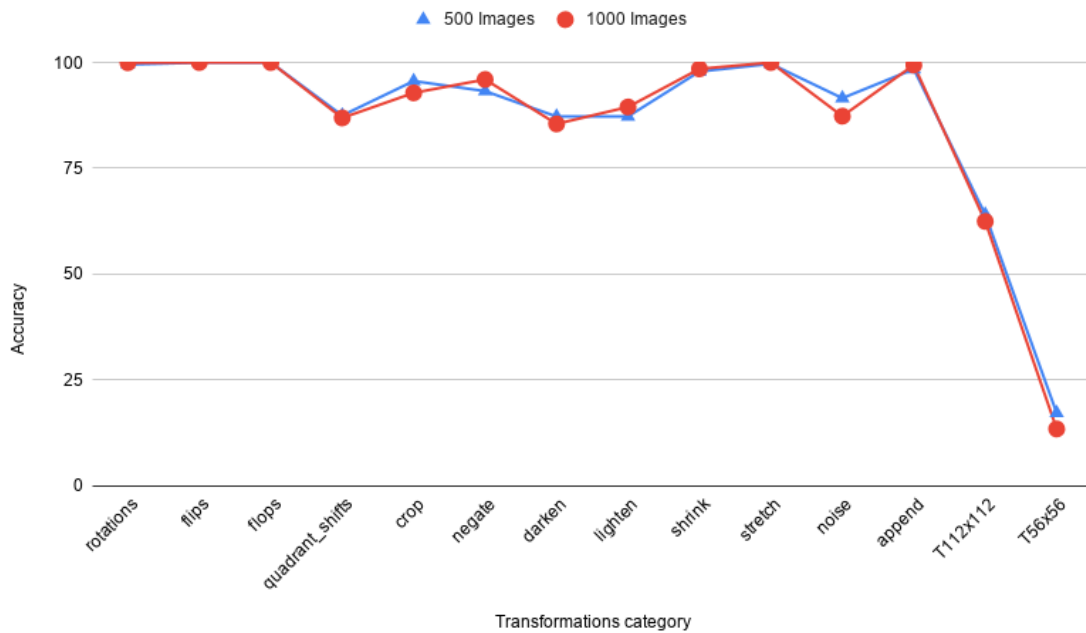


Figure 4.20: Average categorical accuracy of MLP750 on datasets containing 500 and 1000 original images

4.8.3 Full Autoencoder

This procedure uses the decoder layer, \mathbf{D} , of subsection 4.1.1 to generate features from each of the images in datasets under consideration and also for query images. Then it finds the top-k images from the database that are close to the query image using the KNN and cosine similarity.

The details about the accuracy results using full autoencoder on datasets containing 500, and 1000 original images are reported in the table 4.6, and 4.7 respectively.

Comparing the results from tables 4.6 and 4.7 with the results while using individual LIRe descriptors, it can be observed that the baseline using full autoencoder generates better average accuracy and also generates better accuracy for all the datasets except **set2**. Comparing the accuracy results using full autoencoder with the technique of using MLP to combine LIRe descriptors (as mentioned in section 4.8.2), it can be observed that the later not only generates better average accuracy but also better accuracy for all the datasets except for **combined.set3_4** and **base500**. Combining LIRe descriptors with MLP generates about 4% to 7% more accuracy than using full autoencoder. This is because of the superior performance of MLP750 on rotated, color shifted, flipped, flopped, and scaled images than the full autoencoder. It can be easily observed by comparing the categorical accuracy of MLP750 with full autoencoder from figure 4.20 and figure 4.21. Similarly, comparing the accuracy results using full autoencoder with the technique of simply using individual LIRe descriptors, it can be observed that the full autoencoder generates around 6% better accuracy than the best performing LIRe descriptors for datasets containing 500 and 1000 original images. Thus, it can be said that the full autoencoder stands between the techniques mentioned in section 4.8.1 and section 4.8.2 with the techniques mentioned in section 4.8.2 being the top-performing technique.

Table 4.6: Full-Autoencoder & Encoder-Only accuracy on 500 images datasets

Modes	base500		set2		set3		set4		Average	
	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10
Full-Autoencoder	0.602	0.730	0.388	0.576	0.606	0.739	0.523	0.681	0.530	0.682
Encoder Only	0.358	0.554	0.137	0.278	0.377	0.576	0.358	0.558	0.308	0.517

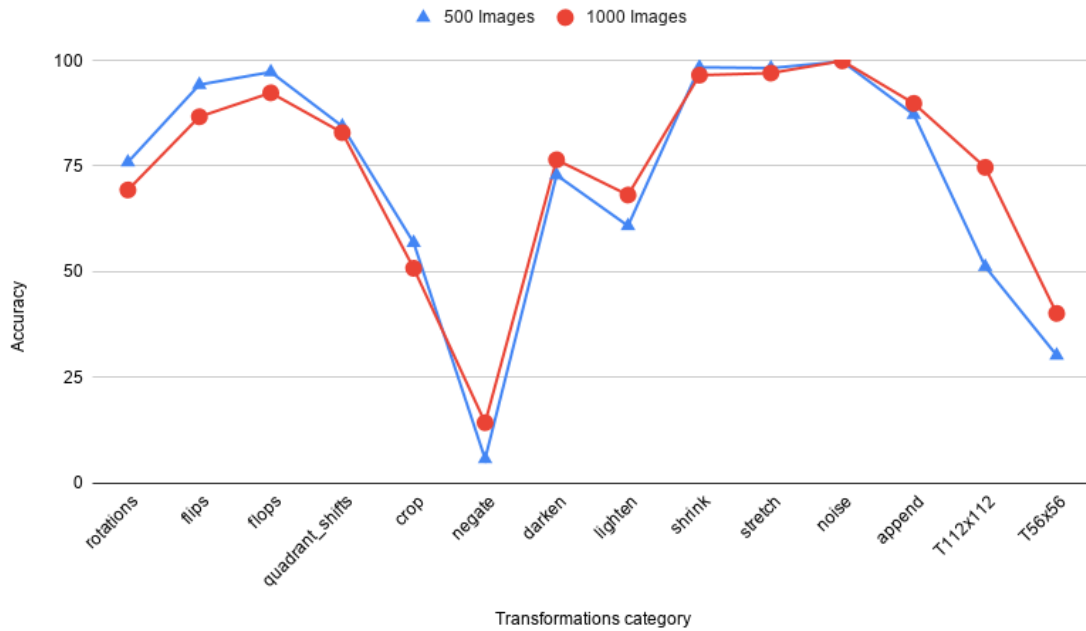


Figure 4.21: Average categorical accuracy of Full Autoencoder on datasets containing 500 and 1000 original images

Table 4.7: Full-Autoencoder & Encoder-Only accuracy on 1000 images datasets

Modes	combined_set2_3		combined_set2_4		combined_set3_4		Average	
	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10
Full AutoEncoder	0.498	0.696	0.456	0.614	0.657	0.772	0.537	0.694
Encoder Only	0.195	0.363	0.306	0.485	0.356	0.527	0.286	0.459

4.8.4 Encoder Only

The encoder only baseline takes the output from the encoding layer, **E1** of subsection 4.1.1, of the autoencoder, generates features and retrieves the top-k results using the KNN and cosine similarity. The accuracy results for datasets containing 500, and 1000 original images are reported in the table 4.6, and 4.7 respectively. Comparing the accuracy results with all the above-mentioned baselines, the encoder performs terribly. Figure 4.22 shows the categorical accuracy of encoder only on various categories of

transformations. It can be seen from the figure that the encoder only generates results comparable to the full autoencoder for the color shift transformations except for the negation. It can also be seen from the figure that the encoder only model maintains high accuracy on the Gaussian noise. It performed terribly on rotated images. After a careful examination of the accuracy of rotated images, it was found that the encoder only shows high accuracy on near to zero degree rotations. Thus, it can be claimed that even though the encoder only can generate a compressed representation that is similar to the original image and its variant when it is exposed to simple transformations such as Gaussian noise and near to zero degree rotations, it is unable to do so for complex transformations. This might have been one of the reasons for its poor performance.



Figure 4.22: Average categorical accuracy of Encoder Only on datasets containing 500 and 1000 original images

4.8.5 Google Logo Detection

Google logo detection claims to detect only popular logos. As the logo datasets used in this thesis may not contain popular logos, creating a baseline using Google logo detection using the logo datasets used in this thesis was not an option. To create this baseline, 100 popular logos were downloaded manually, and the transformations were applied. 20% of the overall transformed images (i.e. 2000 transformed images) were taken randomly and fed to the Google logo detection. The dataset named **base500** also contains these 100 popular logos. These logos were added to the base500 dataset to make a fair comparison between the Google logo detection and the proposed architecture trained on the base500 dataset. The details of the training process are given in the next chapter. Google logo detection achieved an accuracy of 0.6935.

Chapter 5

EXPERIMENT & RESULTS

All of the experiments were carried out using Keras framework, with Tensorflow as backend, on a machine having V100 GPU and Intel processor with 40 cores made available from Idaho National Laboratory Galena node.

Section 5.1 and 5.2 explains how the training process were performed on datasets containing 500 original images (referred to as smaller datasets moving forward) and 1000 original images (referred to as bigger datasets moving forward), respectively and tabulates the accuracy results. Section 5.3 explains the process that has been used to compress feature vectors, by using various compression factors (referred to as CF moving forward), generated by training in section 5.1 and 5.2 further. Section 5.4 talks about combining models trained on smaller datasets to support bigger datasets without retraining. Section 5.5 gives an overview of how the multilevel clustering experiments were performed. Section 5.6 sheds some insights on the results obtained from experiments.

5.1 Training on smaller datasets

It contained 30,500 training images, 10,000 validation images, and 10,000 test images. The model used for training was initialized using the Glorot normal [22] as an initializer with an initial learning rate of 0.01. Instead of loading the whole dataset

in the memory at once, it was loaded in chunks using the ImageGenerator function from Keras. The model was trained for 1000 epochs with a batch size of 80 using an early stopping feature from Keras. The learning rate was decreased by a factor of 10 if the validation loss did not improve for 5 consecutive epochs. The training for this model was stopped when the validation loss did not improve for 15 epochs. For training, fit_generator function from Keras was used. This training was carried out for all the smaller datasets: base500, set2, set3, and set4. The details about these datasets were already given in section 4.3. The first record in the table 5.1 shows the accuracy obtained on smaller datasets and the average accuracy.

Table 5.1: Accuracy of the proposed model at different compression on smaller datasets

CF	base500		set2		set3		set4		Average	
	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10
1	0.681	0.8866	0.4338	0.7255	0.674	0.8577	0.5946	0.8323	0.596	0.826
4	0.6202	0.8296	0.525	0.7303	0.5839	0.8038	0.5979	0.7871	0.582	0.788
8	0.5962	0.8247	0.6724	0.8579	0.575	0.7889	0.6308	0.8059	0.619	0.819
16	0.6733	0.8588	0.6585	0.8265	0.6019	0.8103	0.7016	0.8405	0.659	0.834
32	0.6056	0.8249	0.6538	0.829	0.534	0.7684	0.5829	0.773	0.594	0.799
64	0.4099	0.7343	0.6187	0.8149	0.6321	0.8176	0.7006	0.8365	0.59	0.801
128	0.5996	0.7905	0.5581	0.7431	0.687	0.8369	0.593	0.782	0.609	0.788
256	0.2851	0.6585	0.4674	0.7166	0.5335	0.785	0.485	0.7308	0.443	0.723
512	0.2214	0.5936	0.1617	0.5055	0.1953	0.5635	0.2227	0.5714	0.2	0.559

5.2 Training on bigger dataset

For training the model on bigger datasets, containing 1000 original images, the same architecture that was used for training on the smaller dataset was used. The only difference was that this uses a batch size of 160. The batch size of 160 was chosen because the network did not train with a batch size of 80. It contained 61,000 training

Table 5.2: Accuracy of the proposed model at different compression on bigger datasets when trained from scratch

CF	combined_set2_3		combined_set2_4		combined_set3_4		Average	
	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10
1	0.6409	0.8249	0.5343	0.755	0.6911	0.8559	0.622	0.812
4	0.6629	0.8049	0.5324	0.7312	0.6124	0.7728	0.603	0.77
8	0.6793	0.8172	0.5829	0.7358	0.6065	0.7793	0.623	0.777
16	0.6926	0.8269	0.409	0.6627	0.5996	0.7739	0.567	0.754
32	0.7776	0.8659	0.6785	0.8201	0.428	0.7331	0.628	0.806
64	0.5981	0.7709	0.5308	0.7334	0.5527	0.7585	0.561	0.754
128	0.698	0.8033	0.6635	0.7726	0.7414	0.831	0.701	0.802
256	0.4958	0.7168	0.1483	0.429	0.165	0.4728	0.27	0.54
512	0.2611	0.58	0.1229	0.398	0.3092	0.6106	0.231	0.529

images, 20,000 validation images, and 20,000 test images. The first record in the table 5.2 shows the accuracy obtained on bigger datasets and the average accuracy. The average accuracy of bigger datasets with the proposed model is comparable to the average accuracy on smaller datasets.

5.3 Training for more compression

The training process explained in sections 5.1 and 5.2 compressed an image size of 224 x 224 x 3 to generate a compressed feature of length 25088. These compressed features were obtained from the encoder that had been attached to the decoder using the novel approach after the training of the autoencoder. The feature vector of length 25,088, represented by CF of 1, was still long. Hence, it was decided to reduce the feature of length 25088 to something smaller.

For this, a training process was carried out by adding two more dense layers to the original architecture, as explained in section 4.1.4. A dropout layer with a dropout value of 0.5 was placed between the two dense layers. The first dense layer was used

to vary the number of neurons to specify the length of the further compressed feature. The number of neurons in the added dense layer was varied to specify the length of the further compressed feature to be generated. The number of neurons used in the dense layer is 25088, 6272, 3136, 1568, 784, 392, 196, 98, and 49. The second dense layer was used to convert the output coming from the dropout layer back to length 25088 to feed to the decoder. That is why the number of neurons in the second dense layer was always 25088. The weights of both the newly added dense layers were initialized using the Glorot normal initializer.

The weights for all rest of the convolutional layers were initialized by copying the weights from the corresponding model trained in either section 5.1 or 5.2. The training parameters - learning rate, early stopping criteria, the number of training epochs, method of changing the learning rate based on the validation loss remained the same as that mentioned in sections 5.1 and 5.2. Once the model was trained, the more compressed feature was obtained from the added dense layer using the novel approach. The term CF of 1, 4, 8, 16, 32, 64, 128, 256, and 512 represents feature of length 25088, 6272, 3136, 1568, 784, 392, 196, 98, and 49 respectively. The CF of 4, 8, 16, 32, 64, 128, 256, and 512 represents further compression.

For all of training methods explained in sections 5.1, 5.2, and 5.3, the retrieval process remained the same. For a model trained on any dataset, the predictions for the corresponding test set was first generated. After that, K-Means clustering was used to form clusters based on cosine similarity, and the top-k closest images were retrieved. If the expected images were retrieved within top-k results, the retrieval was considered relevant. The accuracy reports for training the smaller and bigger datasets with more compression are given in the table 5.1 and table 5.2, respectively. It can be seen from tables 5.1 and table 5.2 that the average accuracy does follow

any standard pattern of increase or decrease with the change in CF values. Similarly, the accuracy of the individual datasets does not show any pattern. It, however, can also be seen from tables that the best accuracy results after applying further compression on smaller datasets are achieved at a compression factor of 16, while for bigger datasets is best-achieved at a compression factor of 128. The accuracy for bigger datasets at a compression factor of 128 is considered best based on both its accuracy Acc@1 and Acc@10.

Even though the smaller and bigger datasets were trained using the same initial weights, the later achieved better accuracy at higher compression. This was unexpected. After careful observation of the training stats, it was clear that there were no consistencies in the number of epochs the datasets were trained, because of the early stopping conditions. To understand this better, a scatter plot between the MSE and the corresponding average number of epochs was plotted, as shown in figure 5.1, which shows that the number of training epochs is inversely related to the MSE.

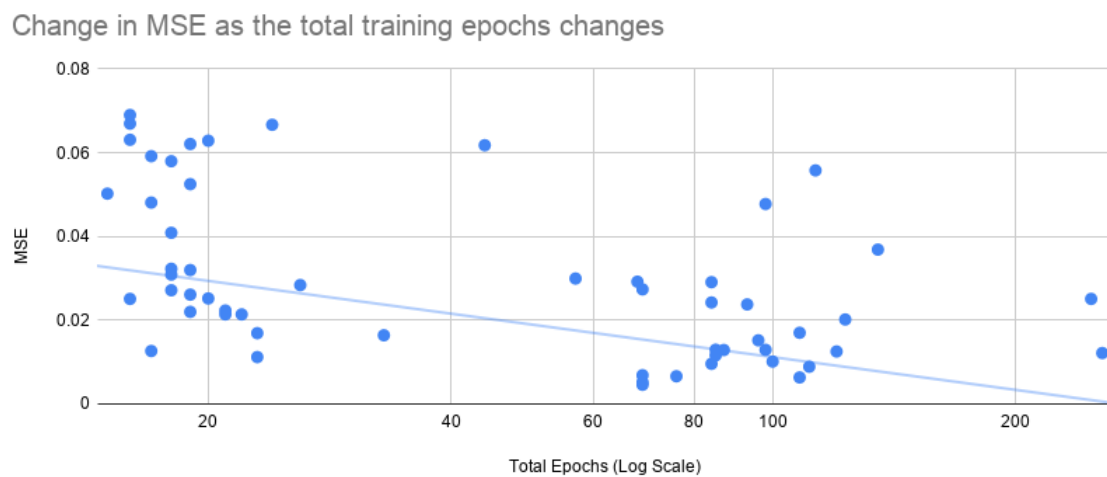


Figure 5.1: Effect of training epochs on Mean Square Error (MSE)

Thus, the early stopping criteria may have been one of the reasons for this unusual

behavior. To verify this further, some of the experiments were run straight for 1000 epochs without the stopping conditions, but none showed improvements in neither accuracy nor MSE. It is still not very clear at this point on why this behavior was seen. The reason may have been that, when the deep learning model was trained on the bigger dataset with a compression factor 128, it did not have a large degree of freedom and was restricted to learn only the meaningful representations from the image. There is still no proof to support this hypothesis.

As opposed to the accuracy change, retrieval time was found to decrease with the increase in the compression. Greater the value of the compression factor, the more compressed the features are. The shorter features of vector comparison take less time compared to the longer vectors. This was the reason for the decrease in the retrieval time with the increase in the compression.

5.4 Manual combining of smaller models

The training process, as mentioned in section 5.2, took a lot of time because to train the system on bigger datasets, the training process needed to be started from scratch even though the subsets that formed the bigger dataset had already been trained. The manual combining of smaller models could be one possible solution to this as it does not require further training. To understand the feasibility of this method, the two models trained on smaller datasets were each allowed to make their predictions on the bigger dataset. For example, the smaller datasets set2 and set3 were each allowed to make their predictions on the combined_set2_3 dataset. The same was done for the datasets combined_set2_4 and combined_set3_4 using datasets that constitute them.

During the time of retrieval, each of the models generated features for the query

Table 5.3: Accuracy on bigger datasets by manual combining of smaller models

CF	combined_set2_3		combined_set2_4		combined_set3_4		Average	
	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10	Acc @1	Acc @10
1	0.5405	0.799	0.5260	0.7962	0.6371	0.8606	0.568	0.819
4	0.5451	0.7913	0.5416	0.7675	0.5752	0.8108	0.554	0.79
8	0.6107	0.845	0.6267	0.8399	0.5879	0.8205	0.608	0.835
16	0.6049	0.8338	0.6441	0.8317	0.6219	0.8357	0.624	0.834
32	0.5791	0.8238	0.5879	0.8131	0.5312	0.7883	0.566	0.808
64	0.5979	0.8358	0.6213	0.8212	0.6278	0.8339	0.616	0.83
128	0.5921	0.8105	0.5389	0.7768	0.6063	0.8265	0.579	0.805
256	0.4809	0.7725	0.3574	0.6916	0.4869	0.7802	0.442	0.748
512	0.1943	0.5684	0.1986	0.5626	0.2169	0.5842	0.203	0.572

image and listed closest images based on the ascending order of their distance to the given query image. If any model was able to retrieve the relevant images within top-k, the retrieval was considered relevant. The accuracy results are reported in the table 5.3. Comparing the average Acc@10 results in the table 5.3 with the Acc@10 results in the table 5.2, it can be seen that this method outperformed training from scratch for all feature lengths. It is also observable that this does not necessarily guarantee better accuracy on individual datasets as compared to training from scratch.

5.5 Multilevel retrieval and charts

Several experiments were run using branching factors of 2, 3, 4, and 5 to understand if the retrieval speed can be made faster using multilevel clustering and retrieval. Experiments were also run using Silhouette to generate an optimum branching factor in the range of 2 to 20. These experiments were run on the best performing compression factors. The threshold for the multilevel clustering was set at 10% of the original number of images.

Figures 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 below show the accuracy of the proposed CBIR system with varying branching factors, and varying levels. From figures 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, it can be seen that the accuracy of the retrieval system is best with a branching factor of 2, and also shows that the accuracy of the retrieval system decreases with the increase in the levels.

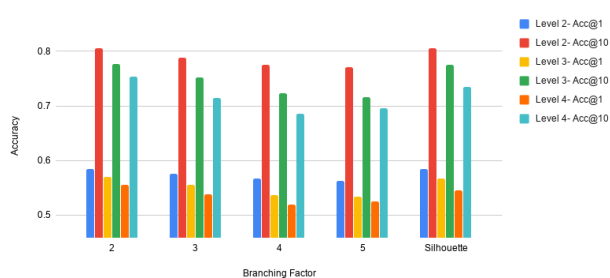


Figure 5.2: Average accuracy on smaller datasets at a compression factor of 1 with varying branching factor at different levels of multilevel clusters

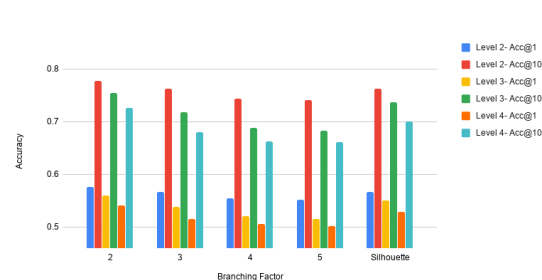


Figure 5.3: Average accuracy on bigger datasets at a compression factor of 1 with varying branching factor at different levels of multilevel clusters

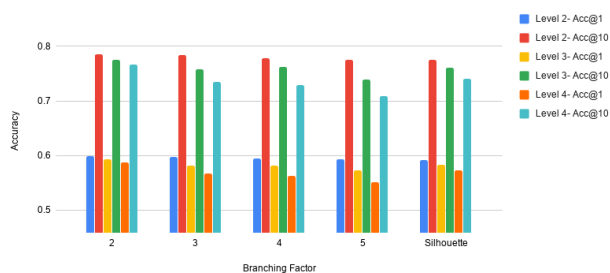


Figure 5.4: Average accuracy on smaller datasets at a compression factor of 8 with varying branching factor at different levels of multilevel clusters

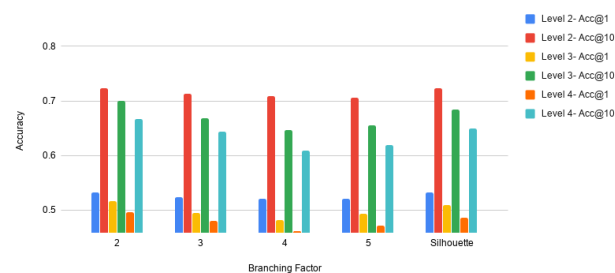


Figure 5.5: Average accuracy on bigger datasets at a compression factor of 8 with varying branching factor at different levels of multilevel clusters

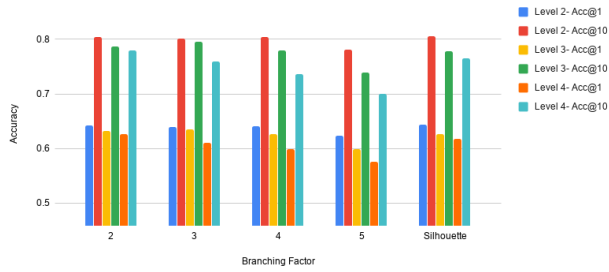


Figure 5.6: Average accuracy on smaller datasets at a compression factor of 16 with varying branching factor at different levels of multilevel clusters

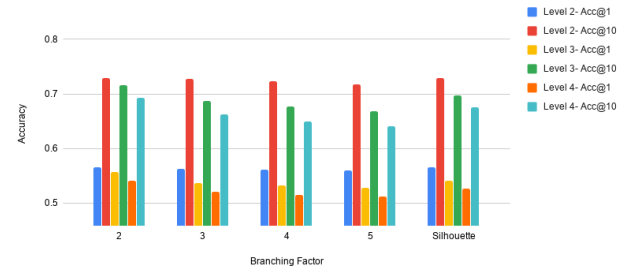


Figure 5.7: Average accuracy on bigger datasets at a compression factor of 16 with varying branching factor at different levels of multilevel clusters

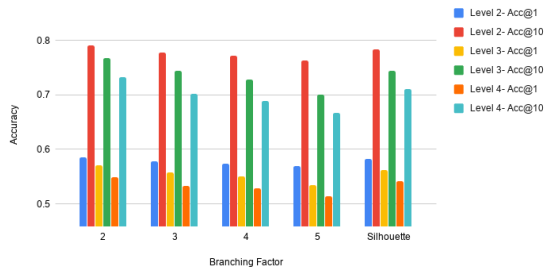


Figure 5.8: Average accuracy on smaller datasets at a compression factor of 64 with varying branching factor at different levels of multilevel clusters

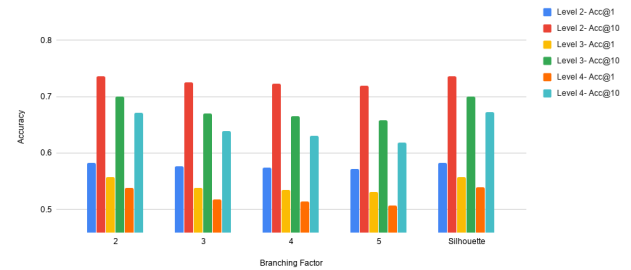


Figure 5.9: Average accuracy on bigger datasets at a compression factor of 64 with varying branching factor at different levels of multilevel clusters

To calculate the retrieval time, each of the experiments was run for 10 iterations, and the average time was calculated. For all of these experiments, the maximum level of the multilevel clustering was set to be 4. Based on the figure 5.10, 5.11, 5.12, and 5.13, it was also observed that the average retrieval time for each query followed a non-linear logarithmic decrease with the increase in level. As already mentioned before in section 5.3, the best average accuracy for smaller and bigger datasets was found at a compression factor of 16 and 128, respectively. The corresponding retrieval time per query, as seen from figures 5.12 and 5.14, for smaller and bigger datasets were found to be 4.729 and 1.587 milliseconds at level 1, respectively. This retrieval

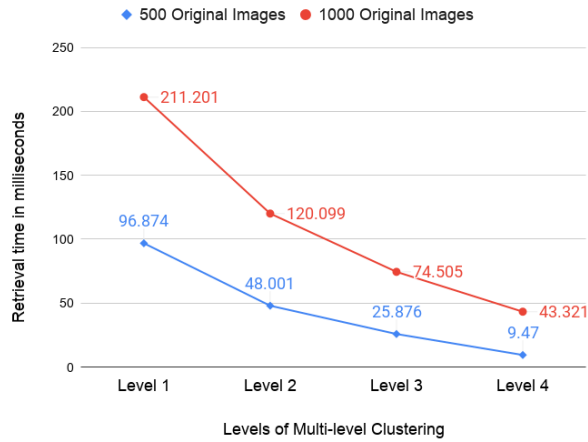


Figure 5.10: Average retrieval time per query at different levels of multi-level clustering with Compression Factor of 1, threshold at 10% and branching factor of 2

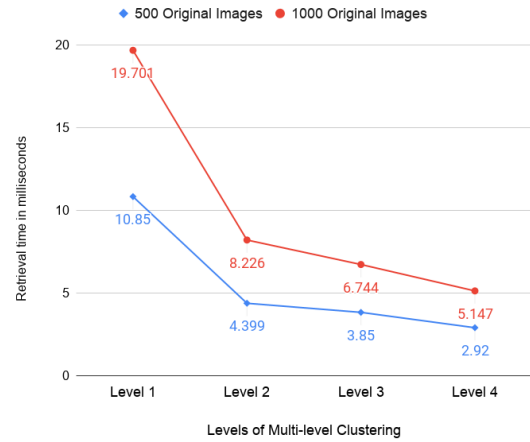


Figure 5.11: Average retrieval time per query at different levels of multi-level clustering with Compression Factor of 8, threshold at 10% and branching factor of 2



Figure 5.12: Average retrieval time per query at different levels of multi-level clustering with Compression Factor of 16, threshold at 10% and branching factor of 2



Figure 5.13: Average retrieval time per query at different levels of multi-level clustering with Compression Factor of 64, threshold at 10% and branching factor of 2

time is less than a blink of an eye (usually lasts 150 to 200 milliseconds [65]), and thus can be considered very fast. Anything at level 1 means without multilevel clustering. Thus, the best accuracy was achieved without multilevel clustering with significant speed for datasets used in this thesis.



Figure 5.14: Retrieval time per query at different levels of multilevel clustering with Compression Factor of 128, threshold at 10% and branching factor of 2

5.6 Observation

The table 5.4 shows the best accuracy of the proposed model and baselines on the test datasets. It can be seen from the table that the proposed model performed better than all the models; it exceeds the state-of-the-art for compression (i.e. full autoencoder)

Table 5.4: Accuracy Comparison of Proposed Model with baselines

Modes	Smaller datasets		Bigger Datasets	
	Acc @1	Acc @10	Acc @1	Acc @10
ACC	0.52	0.626	0.529	0.617
LIRe with MLP750	0.634	0.755	0.632	0.714
Full autoencoder	0.5297	0.6817	0.5369	0.6937
Encoder Only	0.308	0.517	0.286	0.459
Proposed Model @ Compression 16	0.659	0.834	0.567	0.754
Proposed Model @ Compression 128	0.609	0.788	0.701	0.802

by over 10% in terms of accuracy while still maintaining compressed representations. The high accuracy is because of the high-quality hash codes generated by the proposed approach. The feature vector length generated by the autoencoder for an image size of 224x224x3 is 150,528 long vector, whereas the feature vector generated by the proposed model is 25,088. Thus, it can be said that the proposed model generated greater accuracy while generating a feature vector that is 80% compressed than that generated by the autoencoder. This also made the proposed system to have faster retrieval speed than using the full autoencoder.

The feature vector generated by the proposed model, which already beat the state-of-the-art, was further compressed, as explained in section 5.3. The accuracy results for smaller and bigger datasets before further compression is represented by the first row of tables 5.1 and 5.2, respectively. It can also be seen from tables that the best accuracy results after applying further compression on smaller datasets are achieved at a compression factor of 16, while for bigger datasets is best achieved at a compression factor of 128. The proposed model still beat the state-of-the-art even with further compression. The feature vector length at a compression factor of 16 is 98.958% compressed than the input image feature, and at a compression factor of 128 are 99.958% compressed than the input image feature. Therefore, the innovative

approach of attaching the encoder next to the decoder still showed higher performance than the state-of-the-art, even at higher compression.

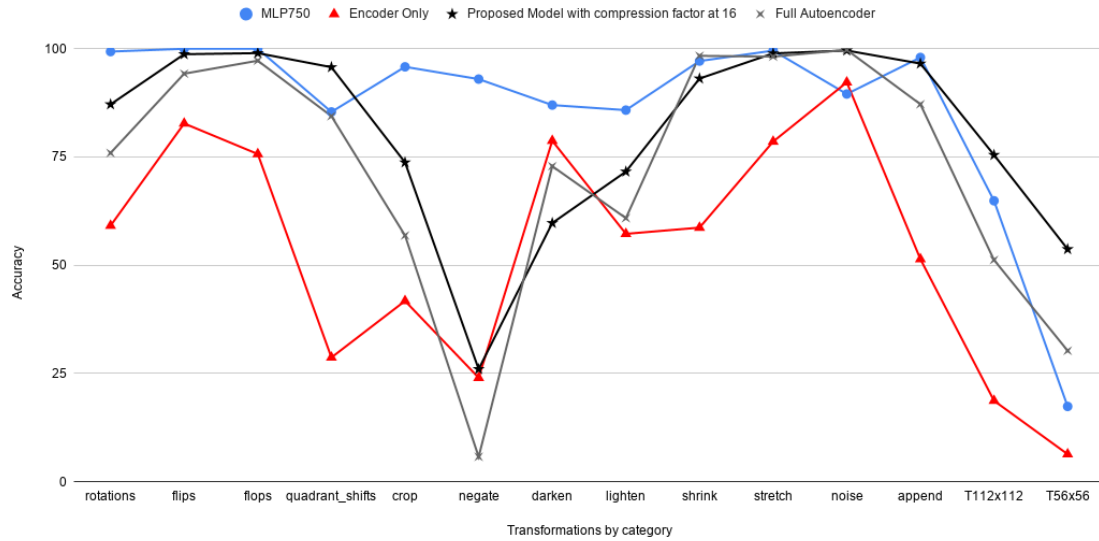


Figure 5.15: Average categorical accuracy comparison of the proposed model at a compression factor of 16 with baselines for smaller datasets

It can also be seen from table 5.4 that the second-best average accuracy for both smaller and bigger datasets is given by MLP750. Figures 5.15 and 5.16 shows the categorical accuracy comparison of the proposed system with all the baselines. It can be seen from the figure 5.15 that the proposed model outperforms the baselines full autoencoder and encoder only in all the transformations category except darken and shrink for datasets containing 500 original images. It is also observable that the MLP750 produces better accuracy than the proposed model, for all the transformations, except for transformations involving Gaussian noise (i.e. labeled as noise in figures) and complex transformations like quadrant shifts, inserting of logo images into colorful backgrounds (i.e. labeled as T112x112 and T56x56 in figures) on smaller datasets. The proposed model outperforms all the baselines by a large

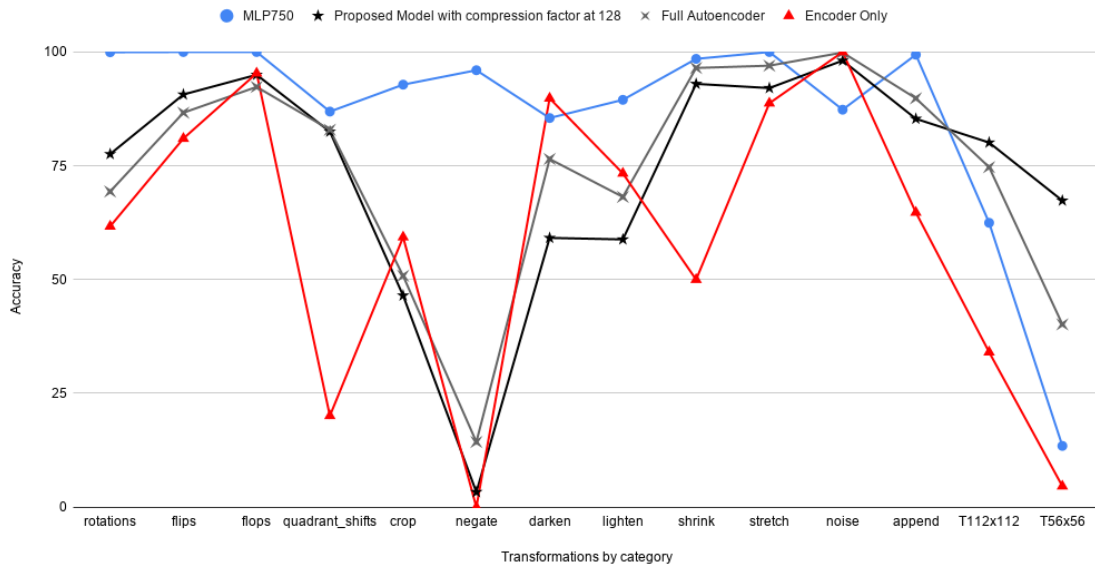


Figure 5.16: Average categorical accuracy comparison of the proposed model at a compression factor of 128 with baselines for bigger datasets

margin on complex transformations. As observable from figure 5.16, the proposed model outperforms all other baselines by a heavy margin for complex transformations even on bigger datasets. There has not been much research work that shows the improvement in retrieval accuracy by combining LIRe descriptors using MLP. This is also a finding of this research work.

The proposed model also surpasses Google logo detection on the base500 image dataset. The proposed model achieved an accuracy of 0.8295 (i.e. 82.95%), which is 13.6% greater than the accuracy generated by Google logo detection. Overall, the proposed model surpassed all the baselines in terms of accuracy. The table 5.3 shows the accuracy found by combining models trained on smaller datasets to work on bigger datasets without re-training. The average accuracy is greater than training on larger datasets from scratch. Hence, this also made the proposed system scalable to datasets.

Chapter 6

CONCLUSIONS

6.1 What has been done so far?

This thesis work has introduced a novel architecture that has surpassed the quality of the compressed feature vector generated by that of the state-of-art (i.e. autoencoder) for self-supervised learning. The proposed system has obtained a higher average accuracy than the state of the art, and also superior accuracy even on the complex transformations. The biggest breakthrough of this research work is that the quality of the compressed feature vector generated by the state of art can be improved without further training. This can be achieved by using the proposed novel approach of copying the architecture and weights of the encoder from the pre-trained autoencoder and attaching it next to the decoder.

The research work has also explored some of the approaches to improve the speed of retrieval: further compressing the generated features and using multilevel clustering. While the proposed system is still able to maintain high accuracy even for some higher level of compression, it shows monotonic decrease in the accuracy with the increase in the branching factor and increase in level of the multilevel clustering. The research work also introduces that combining models trained on smaller datasets can be made to work with bigger datasets formed using the smaller datasets. This makes the proposed system highly scalable.

Thus, the innovation of attaching encoder after the decoder in this research work has created a content-based image retrieval for brand protection that is robust, accurate, fast, and highly scalable.

6.2 Limitations & Future Work

The primary limitation of the proposed system is that it is only able to detect the presence of a single logo in a query image. It also expects unique logos in the dataset. In addition to that, the current work is mostly focused on enhancing the quality of the compressed features generated from the encoder of a denoising autoencoder such that the feature generated for different transformations of the same logo will have very close semantic representation. While it does a pretty good job of improving the quality of the compressed features, it does not generate binary hash codes from the compressed features.

Future work will enhance the proposed system to detect multiple logos in a query image, generate binary hash codes from compressed features for faster retrieval, study the use of LIRe in combination with the proposed system to see if it improves accuracy, and use various other standard architectures such as AlexNet, GoogleNet, and ResNet to create the autoencoder and compare their performances.

Bibliography

- [1] Rui, Y., Huang, T. S., Chang, S. F. (1999). Image retrieval: Past, present, and future. *Journal of Visual Communication and Image Representation*, 10(1), 1-23.
- [2] Long, F., Zhang, H., & Feng, D. D. (2003). Fundamentals of content-based image retrieval. In *Multimedia information retrieval and management* (pp. 1-26). Springer, Berlin, Heidelberg.
- [3] Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.
- [4] Liu, Y., Zhang, D., Lu, G., & Ma, W. Y. (2007). A survey of content-based image retrieval with high-level semantics. *Pattern recognition*, 40(1), 262-282.
- [5] Kokare, M., Chatterji, B. N., & Biswas, P. K. (2003, October). Comparison of similarity metrics for texture image retrieval. In *TENCON 2003. Conference on convergent technologies for Asia-Pacific region* (Vol. 2, pp. 571-575). IEEE.
- [6] Qayyum, A., Anwar, S. M., Awais, M., & Majid, M. (2017). Medical image retrieval using deep convolutional neural network. *Neurocomputing*, 266, 8-20.
- [7] Saritha, R. R., Paul, V., & Kumar, P. G. (2019). Content based image retrieval using deep learning process. *Cluster Computing*, 22(2), 4187-4200.
- [8] Lin, K., Yang, H. F., Hsiao, J. H., & Chen, C. S. (2015). Deep learning of binary hash codes for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 27-35).
- [9] Xu, H., Wang, J. Y., & Mao, L. (2017, June). Relevance feedback for Content-based Image Retrieval using deep learning. In *2017 2nd International Conference on Image, Vision and Computing (ICIVC)* (pp. 629-633). IEEE.
- [10] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- [11] Bengio, Y., & LeCun, Y. (2007). *Large-scale kernel machines. Scaling Learning Algorithms towards AI*. MIT Press, Cambridge.

- [12] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8), 1798-1828.
- [13] Arel, I., Rose, D. C., & Karnowski, T. P. (2010). Deep machine learning-a new frontier in artificial intelligence research [research frontier]. *IEEE computational intelligence magazine*, 5(4), 13-18.
- [14] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160, 3-24.
- [15] Zampoglou, M., Papadopoulos, S., Kordopatis-Zilos, G., Cozien, R., Mercier, G., & MAKINA, E. InVID-In Video Veritas: Verification of Social Media Video Content for the News Industry.
- [16] McCandless, M., Hatcher, E., Gospodnetić, O., & Gospodnetić, O. (2010). *Lucene in action (Vol. 2)*. Greenwich: Manning.
- [17] Chatzichristofis, S. A., & Boutalis, Y. S. (2008, May). Fcth: Fuzzy color and texture histogram-a low level feature for accurate image retrieval. In *2008 Ninth International Workshop on Image Analysis for Multimedia Interactive Services* (pp. 191-196). IEEE.
- [18] Chatzichristofis, S. A., & Boutalis, Y. S. (2008, May). CEDD: color and edge directivity descriptor: a compact descriptor for image indexing and retrieval. In *International Conference on Computer Vision Systems* (pp. 312-322). Springer, Berlin, Heidelberg.
- [19] Zagoris, K., Chatzichristofis, S. A., Papamarkos, N., & Boutalis, Y. S. (2010, September). Automatic image annotation and retrieval using the joint composite descriptor. In *2010 14th Panhellenic Conference on Informatics* (pp. 143-147). IEEE.
- [20] Tungkasthan, A., Intarasema, S., & Premchaiswadi, W. (2009, December). Spatial color indexing using ACC algorithm. In *2009 7th International Conference on ICT and Knowledge Engineering* (pp. 113-117). IEEE.
- [21] Sze-To, A., Tizhoosh, H. R., & Wong, A. K. (2016, July). Binary codes for tagging x-ray images via deep de-noising autoencoders. In *2016 International Joint Conference on Neural Networks (IJCNN)* (pp. 2864-2871). IEEE.
- [22] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256).

- [23] Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence-A Modern Approach* (3rd internat. edn.).
- [24] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation (No. ICS-8506). California Univ San Diego La Jolla Inst for Cognitive Science.
- [25] Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008, July). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning* (pp. 1096-1103).
- [26] Wu, J. (2017). *Introduction to convolutional neural networks*. National Key Lab for Novel Software Technology. Nanjing University. China, 5, 23.
- [27] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.
- [28] Agarap, A. F. (2018). Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375.
- [29] Ide, H., & Kurita, T. (2017, May). Improvement of learning for CNN with ReLU activation by sparse regularization. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 2684-2691). IEEE.
- [30] Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1), 100-108.
- [31] Mansoori, Z., & Jamzad, M. (2009, May). Content based image retrieval using the knowledge of texture, color and binary tree structure. In *2009 Canadian Conference on Electrical and Computer Engineering* (pp. 999-1003). IEEE.
- [32] Lowe, D. G. (1999, September). Object recognition from local scale-invariant features. In *iccv* (Vol. 99, No. 2, pp. 1150-1157).
- [33] Bay, H., Tuytelaars, T., & Van Gool, L. (2006, May). Surf: Speeded up robust features. In *European conference on computer vision* (pp. 404-417). Springer, Berlin, Heidelberg.
- [34] Yang, J., Jiang, Y. G., Hauptmann, A. G., & Ngo, C. W. (2007, September). Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval* (pp. 197-206). ACM.

- [35] Sivic, J., Russell, B. C., Efros, A. A., Zisserman, A., & Freeman, W. T. (2005, October). Discovering objects and their location in images. In Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 (Vol. 1, pp. 370-377). IEEE.
- [36] Wu, L., & Hoi, S. C. (2011). Enhancing bag-of-words models with semantics-preserving metric learning. *IEEE MultiMedia*, 18(1), 24-37.
- [37] Lux, M., & Chatzichristofis, S. A. (2008, October). Lire: lucene image retrieval: an extensible java cbir library. In Proceedings of the 16th ACM international conference on Multimedia (pp. 1085-1088). ACM.
- [38] Lux, M., & Macstravic, G. (2014, January). The LIRE request handler: A Solr plug-in for large scale content based image retrieval. In International Conference on Multimedia Modeling (pp. 374-377). Springer, Cham.
- [39] de Oliveira Barra, G., Lux, M., & Giro-i-Nieto, X. (2016, June). Large scale content-based video retrieval with LIVRE. In 2016 14th International Workshop on Content-Based Multimedia Indexing (CBMI) (pp. 1-4). IEEE.
- [40] Chen, K., & Hennebert, J. (2014, June). Content-based image retrieval with LIRE and SURF on a smartphone-based product image database. In Mexican Conference on Pattern Recognition (pp. 231-240). Springer, Cham.
- [41] Deselaers, T., Weyand, T., Keysers, D., Macherey, W., & Ney, H. (2005, September). FIRE in ImageCLEF 2005: Combining content-based image retrieval with textual information retrieval. In Workshop of the Cross-Language Evaluation Forum for European Languages (pp. 652-661). Springer, Berlin, Heidelberg.
- [42] Deselaers, T., Keysers, D., & Ney, H. (2004, September). FIRE—flexible image retrieval engine: ImageCLEF 2004 evaluation. In Workshop of the Cross-Language Evaluation Forum for European Languages (pp. 688-698). Springer, Berlin, Heidelberg.
- [43] Markonis, D., Schaer, R., Eggel, I., Müller, H., & Depeursinge, A. (2012, September). Using MapReduce for large-scale medical image analysis. In 2012 IEEE Second International Conference on Healthcare Informatics, Imaging and Systems Biology (pp. 1-1). IEEE.
- [44] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

- [45] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
- [46] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- [47] Singh, A. V. (2015). Content-based image retrieval using deep learning.
- [48] Yang, H. F., Lin, K., & Chen, C. S. (2017). Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 40(2), 437-451.
- [49] Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011, June). Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks* (pp. 52-59). Springer, Berlin, Heidelberg.
- [50] En, S., Crémilleux, B., & Jurie, F. (2017, September). Unsupervised deep hashing with stacked convolutional autoencoders. In *2017 IEEE International Conference on Image Processing (ICIP)* (pp. 3420-3424). IEEE.
- [51] Ke, M., Lin, C., & Huang, Q. (2017, November). Anomaly detection of Logo images in the mobile phone using convolutional autoencoder. In *2017 4th International Conference on Systems and Informatics (ICSAI)* (pp. 1163-1168). IEEE.
- [52] Perez, C. A., Estévez, P. A., Galdames, F. J., Schulz, D. A., Perez, J. P., Bastías, D., & Vilar, D. R. (2018, July). Trademark image retrieval using a combination of deep convolutional neural networks. In *2018 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-7). IEEE.
- [53] Jović, M., Hatakeyama, Y., Dong, F., & Hirota, K. (2006, September). Image retrieval based on similarity score fusion from feature similarity ranking lists. In *International conference on Fuzzy Systems and Knowledge Discovery* (pp. 461-470). Springer, Berlin, Heidelberg.
- [54] Krizhevsky, A., & Hinton, G. E. (2011, April). Using very deep autoencoders for content-based image retrieval. In *ESANN* (Vol. 1, p. 2).
- [55] Krizhevsky, A., & Hinton, G. E. (2011, April). Using very deep autoencoders for content-based image retrieval. In *ESANN* (Vol. 1, p. 2).
- [56] Camlica, Z., Tizhoosh, H. R., & Khalvati, F. (2015, November). Autoencoding the retrieval relevance of medical images. In *2015 International Conference on Image Processing Theory, Tools and Applications (IPTA)* (pp. 550-555). IEEE.

- [57] Tizhoosh, H. R., Mitcheltree, C., Zhu, S., & Dutta, S. (2016, December). Barcodes for medical image retrieval using autoencoded radon transform. In 2016 23rd International Conference on Pattern Recognition (ICPR) (pp. 3150-3155). IEEE.
- [58] Moëllic, Pierre-Alain, et al. "Evaluating Content Based Image Retrieval Techniques with the One Million Images CLIC TestBed." WEC (2). 2005.
- [59] Mash, R., Borghetti, B., & Pecarina, J. (2016, December). Improved aircraft recognition for aerial refueling through data augmentation in convolutional neural networks. In International Symposium on Visual Computing (pp. 113-122). Springer, Cham.
- [60] Dieleman, S., Willett, K. W., & Dambre, J. (2015). Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2), 1441-1459.
- [61] Taylor, Luke, and Geoff Nitschke. "Improving deep learning using generic data augmentation." arXiv preprint arXiv:1708.06020 (2017).
- [62] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7), 881-892.
- [63] Pakhira, M. K. (2014, November). A linear time-complexity k-means algorithm using cluster shifting. In 2014 International Conference on Computational Intelligence and Communication Networks (pp. 1047-1051). IEEE.
- [64] Gardner, M. W., & Dorling, S. R. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15), 2627-2636.
- [65] Burr, D. (2005). Vision: in the blink of an eye. *Current Biology*, 15(14), R554-R556.