Cairo University
Faculty of Engineering
System and Biomedical Engineering

Bachelor Thesis

July 18, 2017

Contend-Based Medical Image Retrieval using Deep Learning

Submitted by

Eslam Gamal Thabet

Mohammed Ahmed El-Marakbey

Mazen Fouad Al-Ahdal

Ahmed Gamal Othman

Supervised by

Prof. Ayman El-deeb

Assoc. Prof. Inas Yassin

Content based medical image retrieval using Deep Learning

By

Eslam Gamal Thabet
Mohammed Ahmed El-Marakbey
Mazen Fouad Al-Ahdal
Ahmed Gamal Othman

A thesis submitted in partial fulfilment of the requirements for the degree of

Bachelor of science in Systems and Biomedical Engineering

In the Graduate Academic Unit of (System and Biomedical Engineering)

Supervisors: *Prof. Ayman el Deeb*

Assoc. Prof. Inas Yassin

This thesis is accepted by the Dean of Graduate Studies

Cairo University

July, 2017

Abstract

Content-Based medical image retrieval (CBMIR) is been highly active and lush research area from past few years. In the past image annotation was proposed as the best possible system for CBIR which works on the principle of automatically assigning keywords to images that help image retrieval users to query images based on these keywords. The retrieval performance of a CBMIR system crucially depends on the learning of effective feature representation and similarity measures, which have been extensively studied by researchers for decades. Despite the presence of techniques that have been proposed, it remains one of the most robustly challenging problems in the current research of CBMIR, which is mainly due to the well-known semantic gap issue that exists between low-level image pixels captured by machines and high-level semantic concepts perceived by human. An important outstanding breakthrough technique in artificial intelligence known as Deep Learning, introduces the solution of such problem. Most current machine learning methods work well because of human-designed representations and inputs features. When machine learning is applied only to the input features it becomes merely about optimizing weights to make the best final prediction. Deep learning can be seen as putting back together representation learning with machine learning. It attempts to jointly learn good features, across multiple levels of increasing complexity and abstraction, and the final prediction. In our project we investigate the performance of retrieval task of the IRMA X-ray medical images using both models, the hand-crafted features and deep learning one.

Acknowledgments

We would like to take the opportunity to show our gratitude towards the people that without their support, this thesis would not have been possible.

First and foremost, we would like to thank our parents and families. We wouldn't be where we are today without their amazing support, love, presence and encouragement.

They upheld us with persistence, determination, endurance and hard work.

We also want to thank our mentor and advisor in this journey Dr. Inas Yassine for her constant guidance for us. With her cognizant efforts and sincerity, we could make it to successfully complete our thesis. She was really patient to us to took out time from her busy work load to help us whenever we were stuck and guided us to finish, explain mistakes we would make.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the past 10 years, content-based image retrieval has advanced remarkably in the field of computer vision such as medical imaging, geographical information, crime prevention, education and training, personal photos, and etc. [1]. The initial content-based image retrieval system was presented in the early 1990s to address the problem of retrieving relevant images from the image database [2].

Retrieval of images in the CBIR system is based on the various visual features. The first version of CBIR system was based on the query by image and measures the similarity for retrieving the images from database [3]. The main contribution of CBIR systems is the accurate retrieval of images from the large database based on specific features such as texture, shape and color.

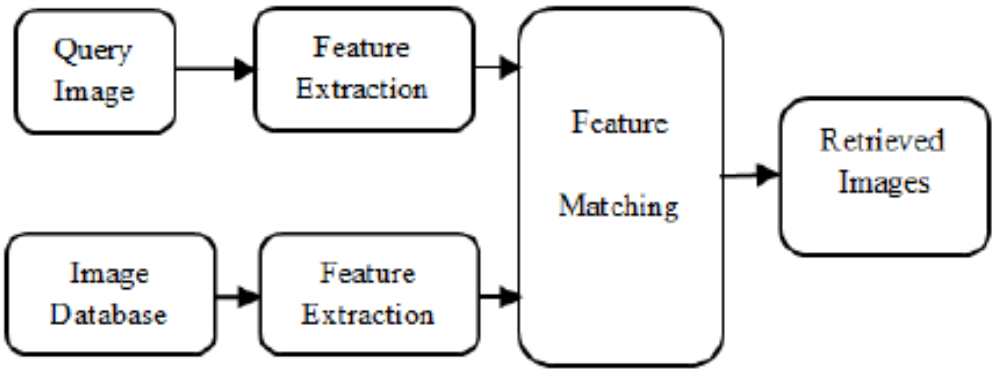


Figure 1: Example of a CBIR system

Many CBIR systems today have been developed for broad ranges of image types. Typical applications include finding images containing natural scenes with color or texture similarities, and presence of people or objects of interest having clearly defined visual features at macro scale. Commercial systems include IBM QBIC, Virage VIR, Google image search and many others. Every day large volumes of different types of medical images such as dental, endoscopy, skull, MRI, ultrasound, radiology are produced in various hospitals and medical centers.

Medical CBIR systems offer potential for clinical benefits [1]. Such systems need to cater for a wide range of image modalities [4] and specific features of importance [2]. These medical image CBIR systems generally offer two forms of functionality: they provide a ranking of a set of images, based on a similarity metric, or they allow the subset of images most similar to a given image to be identified [5].

A recent review of CBIR systems for medical application [6] reveals that the algorithms used for image similarity measures vary between those based on mapping the spatial distribution of overall image intensity properties, and those which focus on identifying the presence and locations of distinctive visual features. Essentially, the conventional approach to CBIR requires some structural modelling of the image content, with sophistication of the model varying from simplistic low level vision features to quite complex spatial mapping and object recognition at much higher levels of visual description.

Feature representation and similarity measurement are very crucial for the retrieval performance of a CBIR system and for decades researchers have studied them extensively. A variety of techniques have been proposed but even then it remains as one of the most challenging problems in the ongoing CBIR research, and the main reason for it is the semantic gap issue that exists between the low-level image pixels captured by machines and high level semantic concepts perceived by humans. Such a problem poses fundamental challenge of Artificial Intelligence from a high-level perspective that is how to build and train intelligent machines like human to tackle real-world tasks.

In chapter 2 we will present different types of hand crafted features for image retrieval such as color, texture, and shape features. In chapter 3 we will discuss deep learning, an important breakthrough technique, which includes a family of machine learning algorithms that attempt to model high-level abstractions in data by employing deep architectures composed of multiple non-linear transformations.

Finally, chapter 4 will contain performance and evaluation of both methods in detail.

Chapter 2

Hand-Crafted Features

Traditionally, visual features are heuristically designed and can be categorized into local features and global features. In early CBIR algorithms and systems, global features are commonly used to describe image content by color, texture and shape into a single holistic representation. Conventional CBIR approaches usually choose rigid distance functions on the extracted low-level features for multimedia similarity search, such as Euclidean distance or cosine similarity.

In this chapter, we will review the most common features used in our CBIR task, explain each feature pros and cons, evaluate the results from this approach to be compared with results of the state-of-art approach deep learning.

2.1 Color Features

Color histograms are widely used in image retrieval, it is one of the most basic approaches and to show performance improvement image retrieval systems. Color histograms give an estimation of the distribution of the colors in the image. The color space is partitioned and for each partition the pixels within its range are counted,

resulting in a representation of the relative frequencies of the occurring colors (0 - 255 grey level).

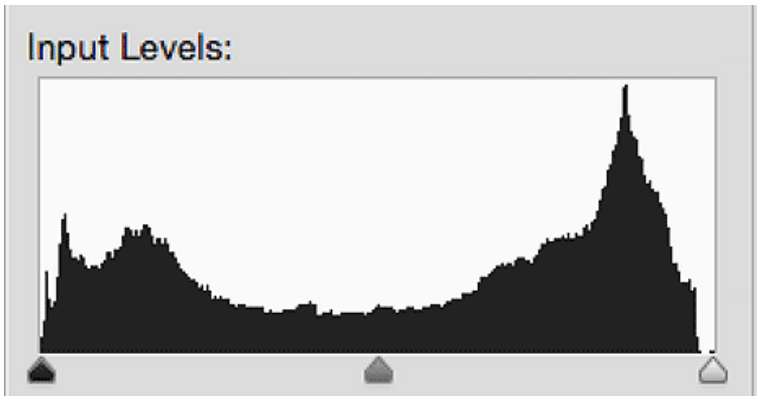


Figure 2: Histogram of an Image from 0-255

Most of content based image retrieval systems are using color histogram feature as a base line, as color histogram has very simple calculations where there is no math complexity or high computational power required. Another advantage of color histogram feature is translation invariance and scale invariance, as if the object in the image translated or scaled, the color distribution will remain the same, but the disadvantage of Color histogram feature that it's highly affected by illumination changes, which may change the color distribution graph.

2.2 Texture Features

A texture is a set of texture elements or texels occurring in some regular or repeated pattern.

2.2.1 Tamura's Textural Features

Textural features [7] corresponding to human visual perception are very useful for optimum feature selection and texture analyzer design. Approximating these features in computational form to be six basic textural features, namely, coarseness, contrast, directionality, line-likeness, regularity, and roughness. From experiments testing the significance of these features with respect to human perception, it was concluded that the first three features are very important. Thus in our experiments we use coarseness, contrast, and directionality.

1- Contrast:

Measures the way in which gray levels vary in the image, and to what extent their distribution is biased to black or white.

$C = \frac{\sigma}{(\alpha_4)^n}$, where n=0.25, σ^2 is variance, α_4 is kurtosis

2- Directionality:

Takes into account the edge strength and the directional angle. They are computed using pixel wise derivatives according to Prewitt edge detector.

$$Edge\ strength\ =\ 0.5(\ |G_x(x,y)|\ +\ |G_y(x,y)|\ )$$

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \qquad \begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}$$

$\boldsymbol{G_x}$ $\boldsymbol{G_y}$

$$\text{angle} = \tan^{-1}(\frac{\text{Gy}}{\text{Gx}})$$

Figure 3: Applying sobel horizontally and vertically on an IRMA image

2.2.2 Grey Level Co-occurrence Matrix

A statistical method of examining texture that considers the spatial relationship of pixels is the gray-level co-occurrence matrix (GLCM) [8], also known as the gray-level spatial dependence matrix. The GLCM functions characterize the texture of an image by calculating how often pairs of pixel with specific values and in a specified spatial relationship occur in an image, creating a GLCM, and then extracting statistical measures from this matrix. The gray-level co-occurrence matrix $P[i,j]$ is defined by first specifying a displacement vector $d = (dx, dy)$ and counting all pairs of pixels separated by d having gray levels i and j. For example, consider the simple 5 x 5 image having gray levels 0, 1, and 2 as shown in Figure 4. Since there are only three gray levels, $P[i,j]$ is a 3 x 3 matrix. Let the position operator be specified as (1, 1), which has the interpretation: one pixel to the right and one pixel below. In a 5 x 5 image there are 16 pairs of pixels which satisfy this spatial separation. We now count all pairs of pixels in which the first pixel has a value of *i* and its matching pair displaced from the first pixel by d has a value of *j*, and we enter this count in the *i*th row and *j*th column of the matrix $P[i,j]$ .For example, there are three pairs of pixels having values [2 , 1]

which are separated by the specified distance, and hence the entry $P[2,1]$ has a value of 3. The complete matrix $P[i,j]$ is shown below.

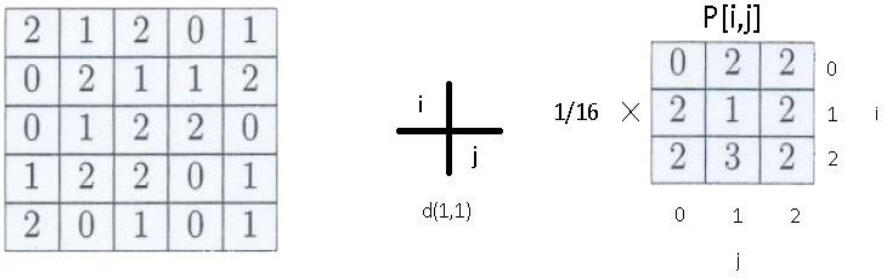


Figure 4: Left, 5x5 image with three gray levels 0, 1, 2. Right, the gray-level co-occurrence matrix for d = (1 , 1).

Note that $P[i,j]$ is not symmetric since the number of pairs of pixels having gray levels [i , j] does not necessarily equal the number of pixel pairs having gray levels [j, i]. The elements of $P[i,j]$ are normalized by dividing each entry by the total number of pixel pairs. In our example, each entry is divided by 16. This normalized $P[i,j]$ is then treated as a probability mass function since the entries now add up to 1.

After you create the GLCM, you can derive several statistical features which provide information about the texture of the image.
A feature which measures the randomness of gray-level distribution is the entropy, defined as:

$$Entropy = -\sum_i \sum_j P[i,j] log P[i,j].$$

Note that the entropy is highest when all entries in $P[i,j]$ are equal; such a matrix corresponds to an image in which there are no preferred gray-level pairs for the specified distance vector d.

The features of energy which provides the sum of squared elements in the GLCM, also known as uniformity or the angular second moment, Contrast which measures the

local variations in the gray-level co-occurrence matrix, and homogeneity that measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal are also defined using the gray-level co-occurrence matrix as given below:

$$Energy = \sum_i \sum_j P[i,j]^2$$

$$Contrast = \sum_i \sum_j (i-j)^2 P[i,j]$$

$$Homogeneity = \sum_i \sum_j \frac{P[i,j]}{1+|i-j|}$$

The choice of the displacement vector d is an important parameter in the definition of the gray-level co-occurrence matrix. Occasionally, the co-occurrence matrix is computed for several values of d and the one which maximizes a statistical measure computed from $P[i,j]$ is used. The gray-level co-occurrence matrix approach is particularly suitable for describing micro-textures. It is not suitable for textures comprising large area primitives since it does not capture shape properties.

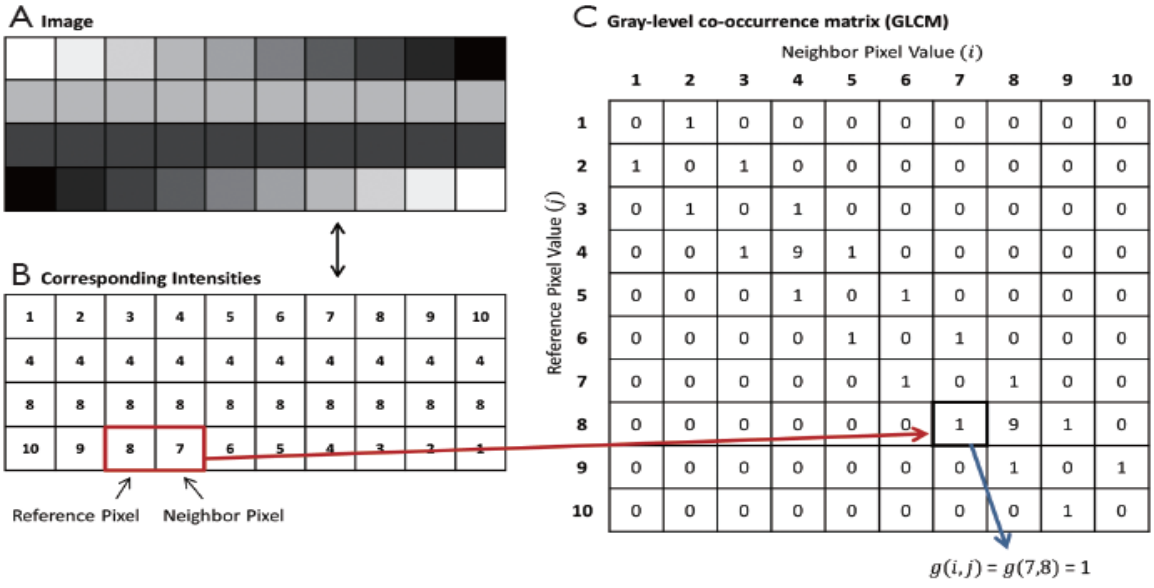


Figure 5: 4x8 image containing 10 gray levels and its GLCM

2.2.3 Gabor Filters

In image processing, a Gabor filter, named after Dennis Gabor, is a linear filter used for texture analysis [9], which means that it basically analyzes whether there is any specific frequency content in the image in specific directions in a localized region around the point or region of analysis. Frequency and orientation representations of Gabor filters are similar to those of the human visual system, and they have been found to be particularly appropriate for texture representation and discrimination. In the spatial domain, a 2D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave.

Because of the multiplication-convolution property (Convolution theorem), the Fourier transform of a Gabor filter's impulse response is the convolution of the Fourier transform of the harmonic function and the Fourier transform of the Gaussian function. The filter has a real and an imaginary component representing orthogonal directions. The two components may be formed into a complex number or used individually.

Complex:

$$g(x, y; \alpha, \theta, \varphi, \sigma, \gamma) = exp\left(-\frac{x'^2 + \alpha^2 y'^2}{2\sigma^2}\right).exp\left(i\left(2\pi\frac{x'}{\alpha} + \varphi\right)\right)$$

Real:

$$g(x, y; \alpha, \theta, \varphi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \alpha^2 y'^2}{2\sigma^2}\right).\cos\left(\left(2\pi\frac{x'}{\alpha} + \varphi\right)\right)$$

Where:

$$x' = x.cos(\theta) + y.\sin(\theta)$$

And

$$y' = -x.sin(\theta) + y.cos(\theta)$$

In this previous equation, $\boldsymbol{\alpha}$ represents the wavelength of the sinusoidal factor, $\boldsymbol{\theta}$ represents the orientation of the normal to the parallel stripes of a Gabor function $\boldsymbol{\varphi}$ is the phase offset, $\boldsymbol{\sigma}$ is the sigma/standard deviation of the Gaussian envelope and $\boldsymbol{\gamma}$ is the spatial aspect ratio, and specifies the ellipticity of the support of the Gabor function.

A set of Gabor filters with different frequencies and orientations may be helpful for extracting useful features from an image. In the discrete domain, two-dimensional Gabor filters are given by:

$$G_c[i,j] = Be^{-\left(\frac{i^2+j^2}{2\sigma^2}\right)}.cos(2\pi f\big(i.cos(\theta) + j.sin(\theta)\big))$$

$$G_s[i,j] = Ce^{-\left(\frac{i^2+j^2}{2\sigma^2}\right)}.sin(2\pi f\big(i.cos(\theta) + j.sin(\theta)\big))$$

Where **B** and **C** are normalizing factors to be determined. 2-D Gabor filters have rich applications in image processing, especially in feature extraction for texture analysis and segmentation. $\boldsymbol{f}$ defines the frequency being looked for in the texture. By varying $\boldsymbol{\theta}$, we can look for texture oriented in a particular direction. By varying $\boldsymbol{\sigma}$, we change the support of the basis or the size of the image region being analyzed.

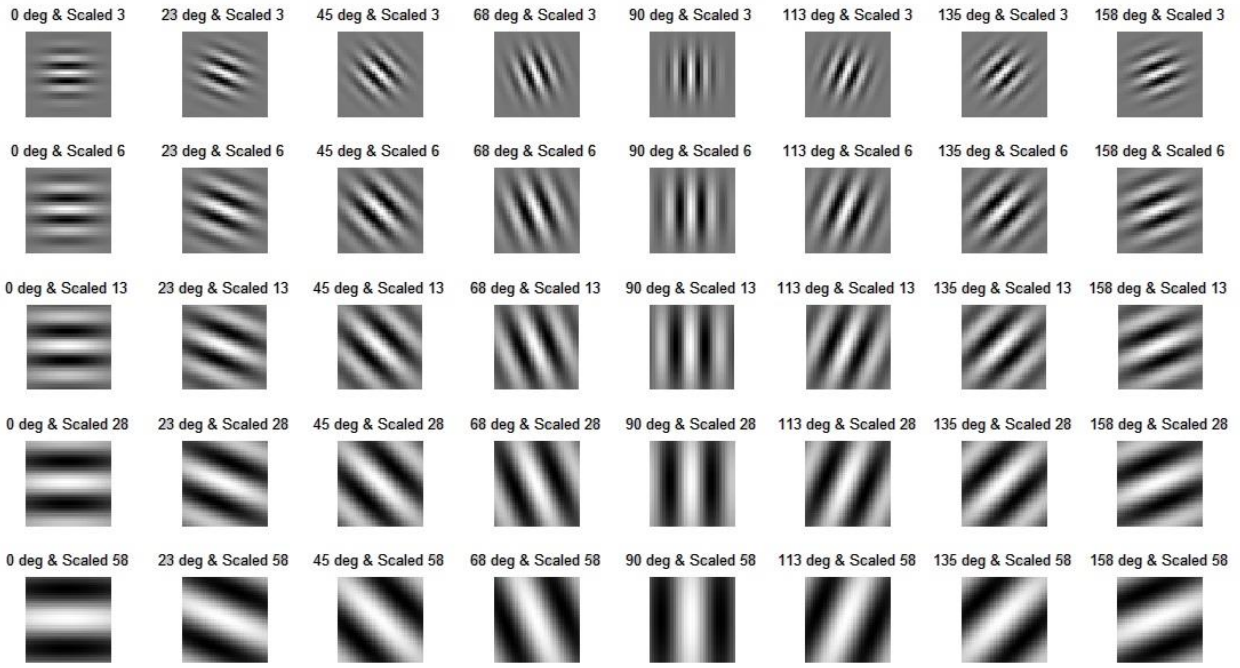


Figure 6: Gabor filter bank with different orientations and sizes

2.2.4 Local Binary Pattern (LBP)

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number [10]. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings. The original version of the local binary pattern operator works in a 3 × 3-pixel block of an image. The pixels in this block are thresholded by its center pixel value, multiplied by powers of two and then summed to obtain a label for the center pixel. As the neighborhood consists of 8

pixels, a total of 28 = 256 different labels can be obtained depending on the relative gray values of the center and the pixels in the neighborhood.

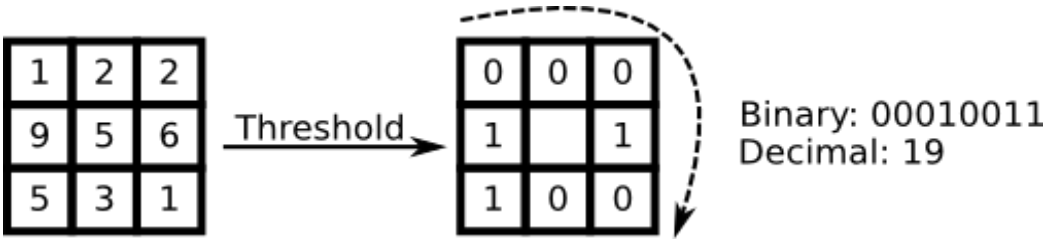


Figure 7: Example of obtaining the LBP micropattern

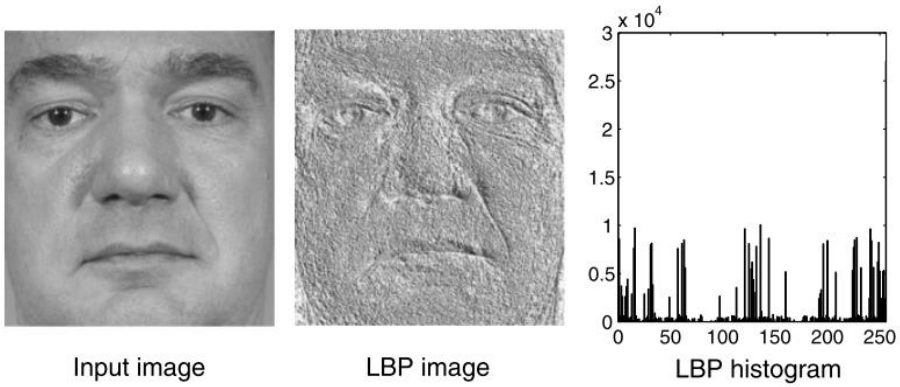


Figure 8: Example of applying LBP on an image

Several years after its original publication, the local binary pattern operator was extended to be presented in a more generic, this generic formulation of the operator puts no limitations to the size of the neighborhood or to the number of sampling points.

Using a circular neighborhood and bilinear interpolated values at non-integer pixel coordinates allow any radius and number of pixels in the neighborhood. The gray scale variance of the local neighborhood can be used as the complementary contrast measure. In the following, the notation (P, R) will be used for pixel neighborhoods which means P sampling points on a circle of radius of R.

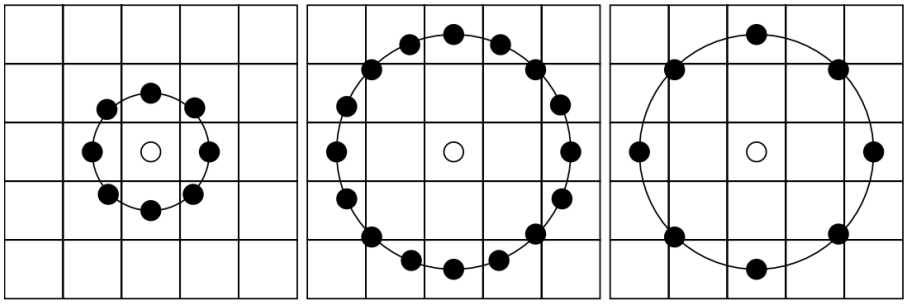


Figure 9: The circular (8,1), (16,2) and (8,2) neighborhoods. The pixel values are bilinearly interpolated whenever the sampling point is not in the center of a pixel

Consider a monochrome image $I(x, y)$ and let $\boldsymbol{g}$ denote the gray level of an arbitrary pixel (x, y), i.e. $\boldsymbol{g} = I(x, y)$. Moreover, let $\boldsymbol{g}_p$ denotes the gray value of a sampling point in an evenly spaced circular neighborhood of P sampling points and radius R around point (x, y):

$$g_p = I(x_p, y_p), \quad p = 0, \ldots., p-1$$

$$x_p = x + R\, cos(\frac{2\pi\rho}{P})$$

$$y_p = y - R\, sin(\frac{2\pi\rho}{P})$$

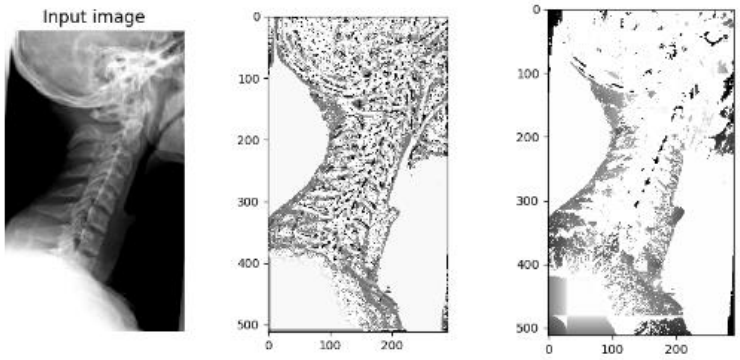


Figure 10: Experiments in IRMA dataset image on changing LBP parameters: Sampling points and Radius

2.3 Shape Features

2.3.1 Histogram Oriented Gradients (HOG)

Histogram of Oriented Gradient descriptors [11], or HOG descriptors, are feature descriptors used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. The essential thought behind the Histogram of Oriented Gradient descriptors is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The implementation of these descriptors can be achieved by dividing the image into small connected regions, called cells, and for each cell compiling a histogram of gradient directions or edge orientations for the pixels within the cell. The combination of these histograms then represents the descriptor. For improved performance, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination or shadowing.

The first step of calculation is the computation of the gradient values. To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels

$$D_x = [-1 \quad 0 \quad 1] \; and \; D_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

So the output of applying these two filter is:

Figure 11: Initial Image

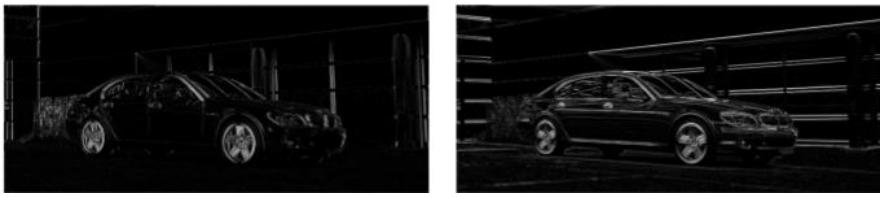


Figure 12: Left, X-derivative of the initial image; Right, Y-derivative of the initial image

So, being given an image I, we obtain the x and y derivatives using a convolution operation:

$$I_x = I * D_x \ \ and \ \ I_y = I * D_y$$

So the magnitude of the gradient is:

$$|G| = \sqrt{I_x^2 + I_y^2}$$

And the orientation of the gradient is given by:

$$\theta = arctan\frac{I_x}{I_y}$$

We notice that x-gradient fires on vertical lines and the y-gradient fires on horizontal lines. The magnitude of gradient fires wherever there is a sharp change in intensity, as the gradient image removed a lot of non-essential information (e.g. constant colored background), but highlighted outlines.

The next step is to calculate histogram of gradients, in this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells. Each pixel within the cell casts a weighted vote for an orientation-based histogram channel based on the values found in the gradient computation. The cells themselves

are rectangular and the histogram channels are evenly spread over 0 to 180 degrees or 0 to 360 degrees, depending on whether the gradient is "unsigned" or "signed". Dalal and Triggs found that unsigned gradients used in conjunction with 9 histogram channels performed best in their experiments. As for the vote weight, pixel contribution can be the gradient magnitude itself, or the square root or square of the gradient magnitude.

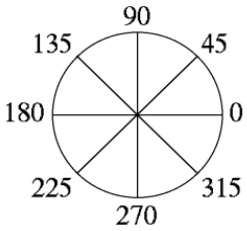


Figure 13: 0-360 gradient

The next step is to create a histogram of gradients in these 8×8 cells. The histogram contains 9 bins corresponding to angles 0, 20, 40 … 360.

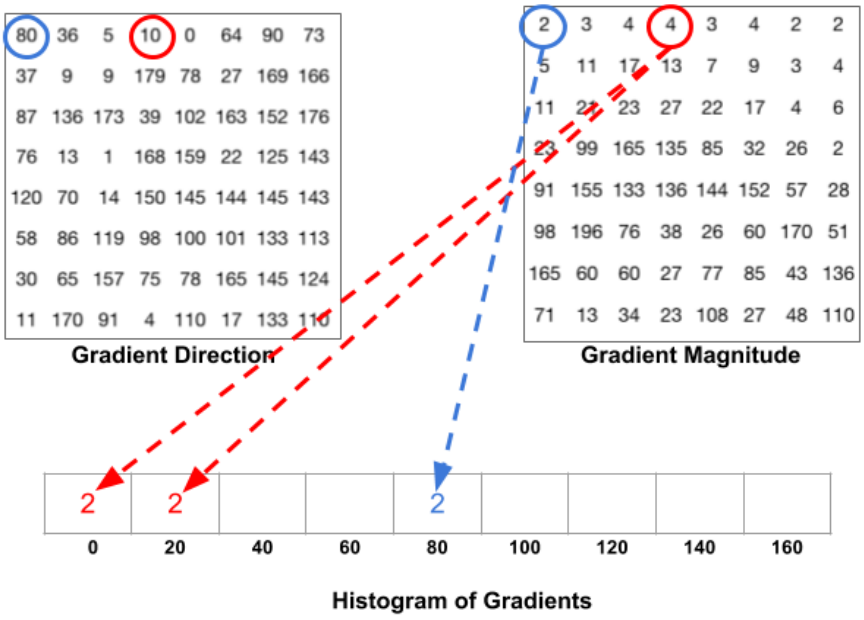


Figure 14: 8x8 cells example to obtain Histogram of Gradients

We are looking at magnitude and direction of the gradient of the same 8×8 patch as in the previous figure. A bin is selected based on the direction, and the vote (the value that goes into the bin) is selected based on the magnitude, and the contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram. For the patch above, it looks like this:

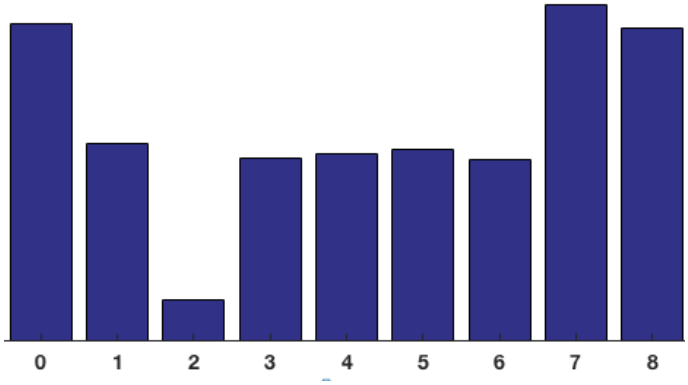


Figure 15: 9-bin histogram after obtaining the HoG

Next step is 16×16 Block Normalization, in order to account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spatially-connected blocks. The HOG descriptor is then the vector of the components of the normalized cell histograms from all of the block regions. These blocks typically overlap, meaning that each cell contributes more than once to the final descriptor.

There are different methods for block normalization. Let $\boldsymbol{V}$ be the non-normalized vector containing all histograms in a given block, $||V^k||$ be its k-norm for $k = 1, 2$ and $\boldsymbol{e}$ be some small constant (whose value will not influence the results). Then the normalization factor can be one of the following:

$$L2-norm:\ f = \frac{V}{\sqrt{||V||_2^2 + e^2}}$$

$$L1-norm:\ f = \frac{V}{\sqrt{||V||_1 + e}}$$

$$L1 - sqrt: f = \sqrt{\frac{v}{||V||_1 + e}}$$

Histogram of Oriented Gradient descriptor affected by two parameters:

1. Number of orientation bins
2. Cell size (mostly 8 x 8)

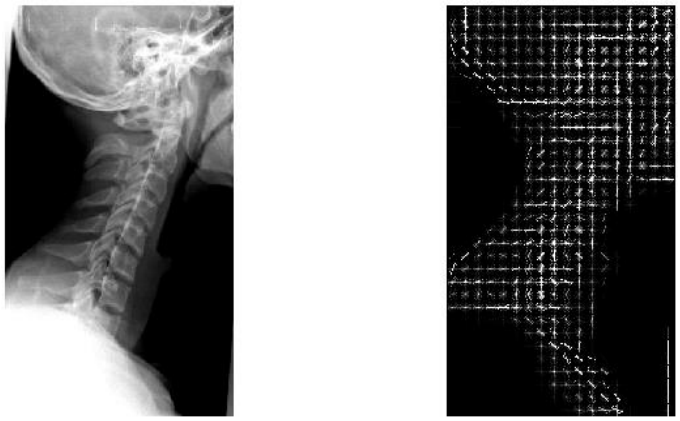


Figure 16: Left, Input image from IRMA dataset; Right, resulted HoG image

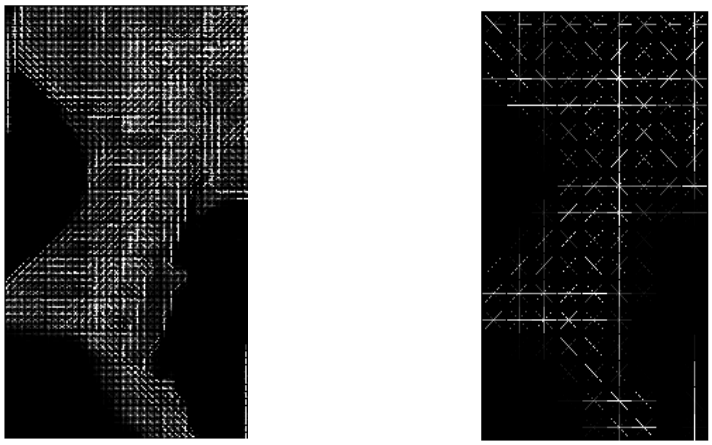


Figure 17: Left, The effect of increasing number of orientation bins; Right, The effect of increasing number of cell size

2.3.2 Scale Invariant Feature Transform (SIFT)

SIFT [12] features that have many properties that make them suitable for matching differing images containing the same object or scene. The features are invariant to image scaling and rotation, and partially invariant to change in illumination. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition. Following are the major stages of computation used to generate the set of image features:

1. Scale-space extrema detection: The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.

$$L(x, y, \sigma) = G((x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma} e^{-(x^2+y^2)/2\sigma^2}$$

$$D(x, y, \sigma) = (D(x, y, K\sigma) - D(x, y, \sigma)) * I(x, y)$$

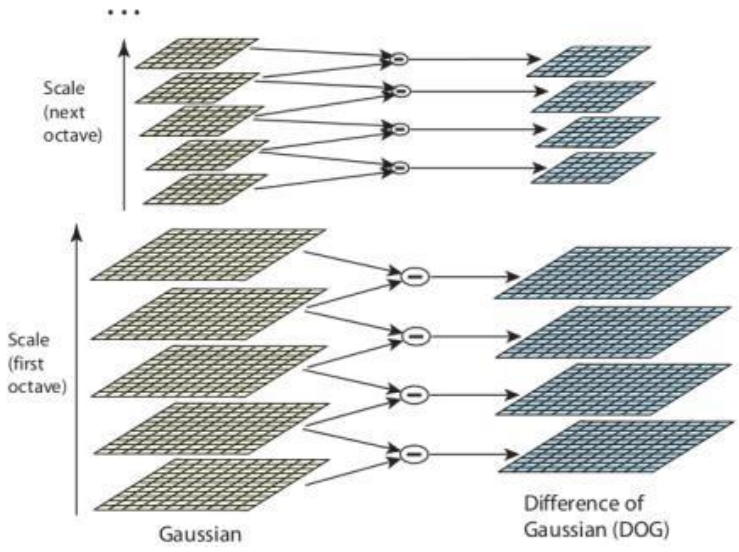


Figure 18: Scale Space Extrema

2. Keypoint localization: Once a keypoint candidate has been found by comparing a pixel to its neighbors, the next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge.

3. Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

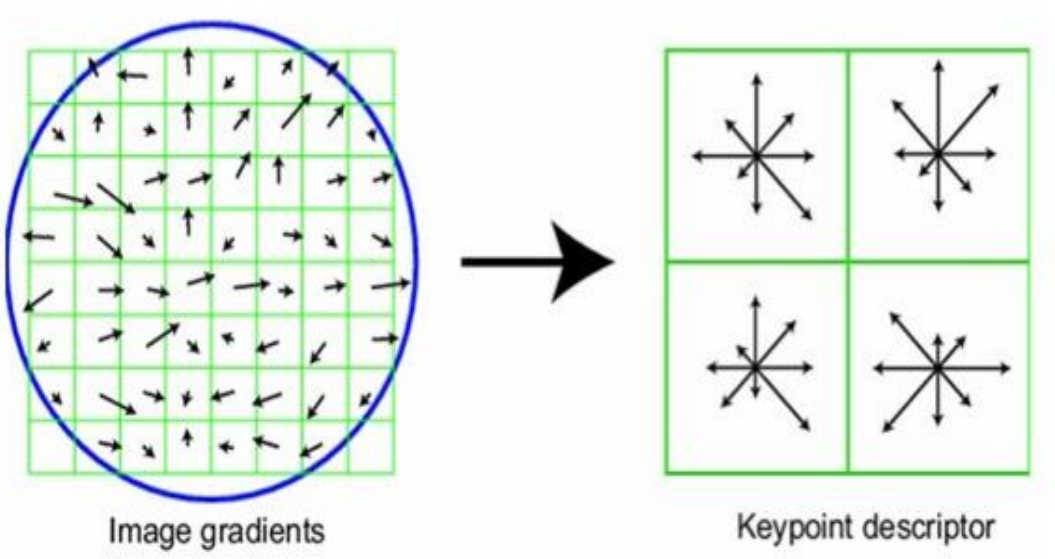


Figure 19: Left, Gradient Orientaion hostogram; Right, Keypoint descriptor

4. Key point descriptor: The local image gradients are measured at the selected scale in the region around each key point. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination. This approach transforms image data into scale-invariant coordinates relative to local features. An important aspect of this approach is that it generates large numbers of features that densely cover the image over the full range of scales and locations. A typical image of size 500x500 pixels will give rise to about 2000 stable features (although this number depends on both image content and choices for various parameters). SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors.

2.3.3 Fourier-Mellin Transform

The Fourier-Mellin transform [13] is a useful mathematical tool for image recognition because its resulting spectrum is invariant in rotation, translation and scale. The Fourier Transform itself (FT) is translation invariant and its conversion to log-polar coordinates converts the scale and rotation differences to vertical and horizontal offsets that can be measured. A second FFT, called the Mellin transform (MT) gives a transform-space image that is invariant to translation, rotation and scale.

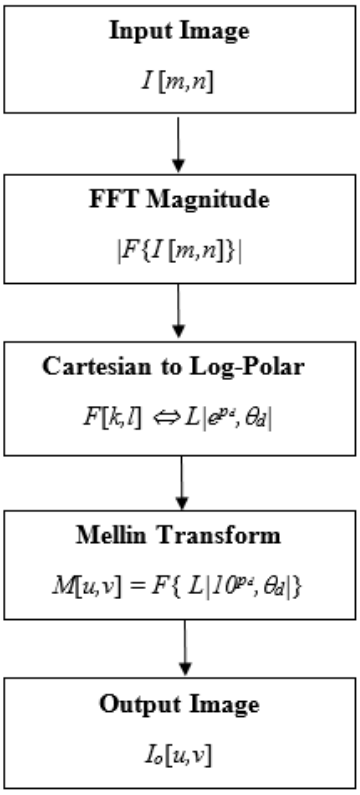


Figure 20: Block diagram of Fourier Mellin Transform

The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image. The number of frequencies corresponds to the number of pixels in the spatial domain image, i.e. the image in the spatial and Fourier domain are of the same size.

For a square image of size N×N, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) e^{-2\pi(\frac{ki}{N}+\frac{lj}{N})}$$

The FFT is projected onto the log-polar plane by the coordinate transform shown below.

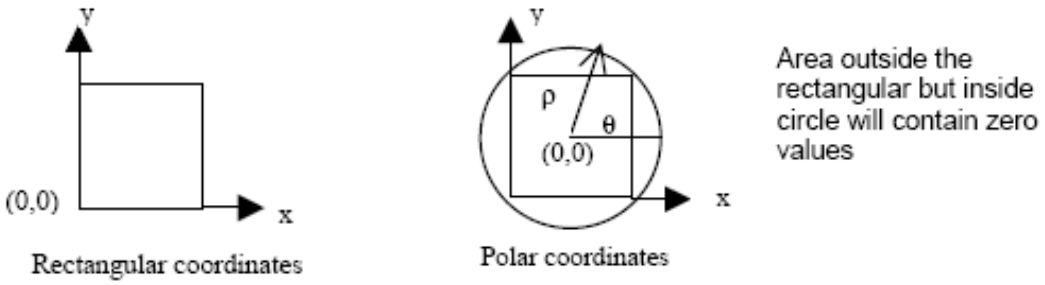


Figure 21: Transformation from rectangular to polar co-ordinates

For the conversion from Cartesian coordinates to Log-Polar coordinates the following equation is true:

$$r = \sqrt{x^2 + y^2}$$

The origin (m_0,n_0) should be at the center of the image matrix to ensure the maximum number of pixels is included. If the image consists of a square $N * N$ matrix then the coordinates of the origin are:

$$m_0 = \frac{N}{2} \; ; \; n_0 = \frac{N}{2} \qquad for\ N\ odd$$

$$m_0 = \frac{N-1}{2} \; ; \; n_0 = \frac{N-1}{2} \qquad for\ N\ even$$

Since the pixels in Cartesian coordinates cannot be mapped one-to-one onto pixels in the Log-Polar coordinate space, an average of the surrounding pixels needs to be calculated. The standard methods to do this includes nearest neighbor, bilinear and bi-cubic resampling. The relationship between the polar coordinates (ρ, θ) used to sample the input image and the polar coordinates of the log-polar image (r,θ) can be described by:

$$(\rho, \theta) = (e^r, \theta)$$

The next step is to get a transform-space image that is a rotation and scale invariant representation of the original image. This is also a Fourier transform. If we make a change in coordinates from Cartesian to a Log-Polar system, we can directly perform a DFT over the image to obtain the scale and rotation invariant representation.

Chapter 3

Deep Learning

Deep Learning can be summed up as a sub field of Machine Learning studying statistical models called deep neural networks. The latter are able to learn complex and hierarchical representations from raw data, unlike hand crafted models which are made of an essential features engineering step.

In this chapter, we will introduce a brief background for deep learning and its models, we draw an attention towards state-of-art convolutional neural networks and its architecture.

3.1 Deep Learning Background

In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straight-forward for computers—problems that can be described by a list of formal, mathematical rules. The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognizing

spoken words or faces in images. A solution to these intuitive problems is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all of the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning.

Learned representations often result in much better performance than can be obtained with hand-designed representations. They also allow AI systems to rapidly adapt to new tasks, with minimal human intervention. A representation learning algorithm can discover a good set of features for a simple task in minutes, or a complex task in hours to months. Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers. Of course, it can be very difficult to extract high-level, abstract features from raw data. Many of these factors of variation, such as a speaker's accent, can be identified only using sophisticated, nearly human-level understanding of the data. When it is nearly as difficult to obtain a representation as to solve the original problem, representation learning does not, seem to help us. Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning allows the computer to build complex concepts out of simpler concepts. Figure1 shows how a deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are in turn defined in terms of edges.

A matter of fact is that deep learning has been around for many decades, However, until 2006, deep, fully connected neural networks were commonly outperformed by shallow architectures that used feature engineering. In that year, however, Hinton and Salakhutdinov (2006) [14] introduced a novel way to pre-train deep neural

networks. Later, Vincent et al. (2008) [15] showed that similar effects can be obtained by using autoencoders. Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation and bioinformatics where they produced results comparable to and in some cases superior to human experts. Three additional reasons have recently helped deep architectures obtain state of the art performance: large datasets, faster, parallel computers and a plethora of machine learning insights into sparsity, regularization and optimization. Because deep learning models learn from raw inputs and without manual feature engineering they require more data. In this time of "big data" and crowd sourcing, many researchers and institutions can easily and cheaply collect huge datasets which can be used to train deep models with many parameters.

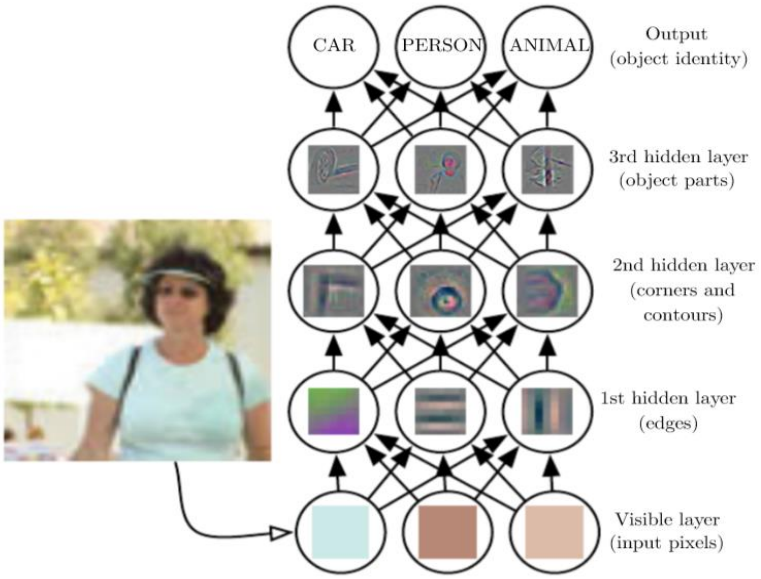


Figure 22: Hierarchal representation of features in deep learning model .

3.2 Convolutional Neural Networks

3.2.1 Overview

Convolutional Neural Networks (CNN) are very similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other and they still have a loss function (e.g. SVM/Softmax) on the last fully-connected layer. [16]

A CNN is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and vastly reduce the amount of parameters in the network. [17] The convolution operation is used on small regions so as to avoid the situation when if all the layers are fully connected billions of parameters will exist. Convolutional networks use shared weights in the convolutional layers i.e. for each pixel in the layer same filter (weights bank) is used to reduce the required memory size and improve performance.

3.2.2 Architecture

Traditional Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any

connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.

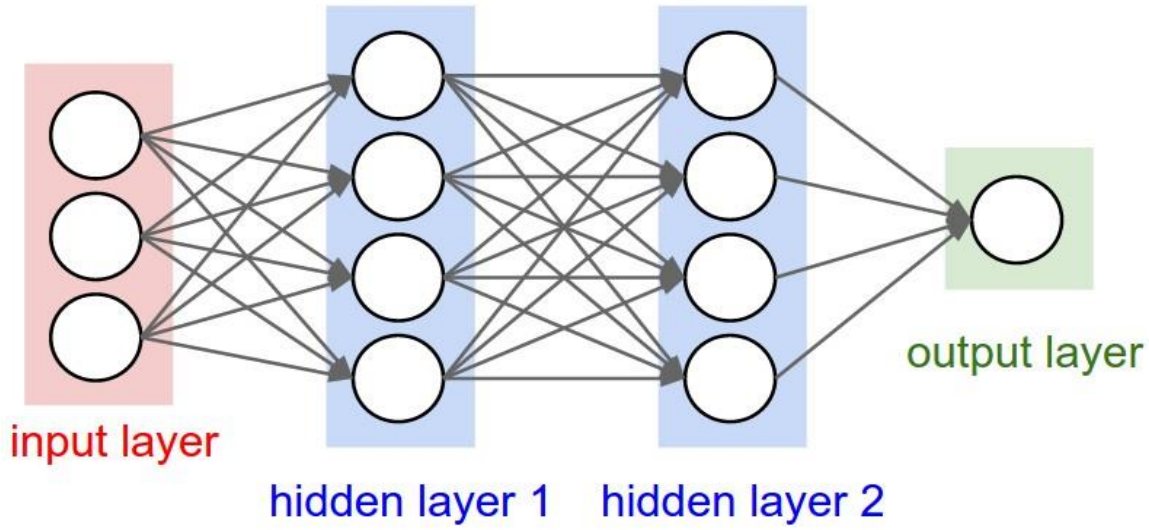


Figure 23: : Traditional Neural Network

Traditional neural networks do not scale well with images. Taking for example the famous CIFAR-10 dataset, images are only of size 32x32x3 (32 wide, 32 highs, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. For example, the input images in CIFAR-10 are an input volume of activations, and the volume has

dimensions 32x32x3 (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension. [16]

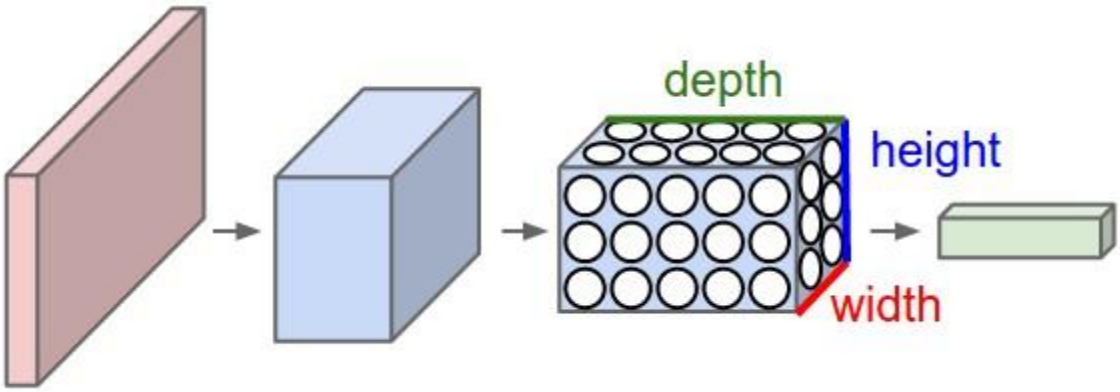


Figure 24: Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations.

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. We will stack these layers to form a full ConvNet architecture.

3.2.3 Sparse connectivity

The sparse connectivity property means that each neuron is connected to only a limited number of other neurons. This property has been observed in mammalian brains and have inspired a variety of machine learning algorithms. A notable result was achieved through the sparse coding model of Olshausen and Field (1996). Given small patches from images of natural scenes, the model is able to produce Gabor-like filters, resembling properties of simple cells found in mammalian primary visual

cortex. Another example is the optimal brain damage method of [18], which can be used to prune synaptic connections in a neural network, making connectivity sparse. Due to sparse connectivity drastic reduce in the number of parameters is noticed compared to a fully connected network, reducing overfitting and more importantly, computational complexity of the network.

One of the basic concepts in deep CNNs is the receptive field, or field of view, of a unit in a certain layer in the network. Unlike in fully connected networks, where the value of each unit depends on the entire input to the network, a unit in convolutional networks only depends on a region of the input. This region in the input is the receptive field for that unit. The concept of receptive field is important for understanding and diagnosing how deep CNNs work. Since anywhere in an input image outside the receptive field of a unit does not affect the value of that unit, it is necessary to carefully control the receptive field, to ensure that it covers the entire relevant image region. [19]

The receptive field size of a unit can be increased in a number of ways. One option is to stack more layers to make the network deeper, which increases the receptive field size linearly by theory, as each extra layer increases the receptive field size by the kernel size. Sub-sampling on the other hand increases the receptive field size multiplicatively. Modern deep CNN architectures like the VGG networks [20] and Residual Networks [21, 22] use a combination of these techniques.

Not all pixels in a receptive field contribute equally to an output unit's response. Intuitively it is easy to see that pixels at the center of a receptive field have a much larger impact on an output. In the forward pass, central pixels can propagate information to the output through many different paths, while the pixels in the outer area of the receptive field have very few paths to propagate its impact. In the backward pass, gradients from an output unit are propagated across all the paths, and therefore, the central pixels have a much larger magnitude for the gradient from that output. [19]

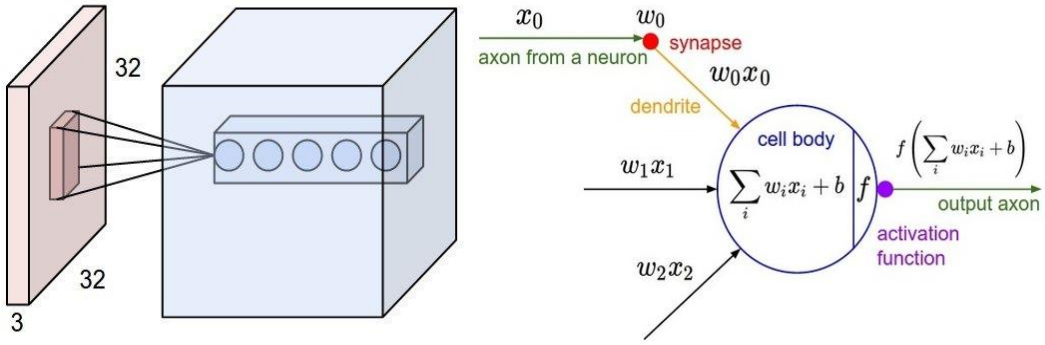


Figure 25: An example of sparse connectivity, and computations in a neuron.

.

In the figure above at the left, an example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth, (i.e. all color channels). [16] Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input. At the right, a neuron computes a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

3.2.4 Shared Weights

Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. Neurons in CNNs share weights unlike in MLPs where each neuron has a separate weight vector. This sharing of weights ends up reducing the overall number of trainable weights hence introducing sparsity. Utilizing the weights sharing strategy, neurons are able to perform convolutions on the data with the convolution filter being formed by the weights.

Denoting a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size [55x55x96] has 96 depth slices, each of size [55x55]), we are going to constrain the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, the first convolutional layer would have 96 unique set of weights

(one for each depth slice). Assuming we are using a 5x5x3 filter, then we have 96*5*5*3 = 7,200 unique weights, or 7,296 parameters (+96 bias). Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

Since all the neurons in a single depth slice are using the same weight vector, then the forward pass of the convolutional layer can in each depth slice be computed as a convolution of the neuron's weights with the input volume. This is why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.

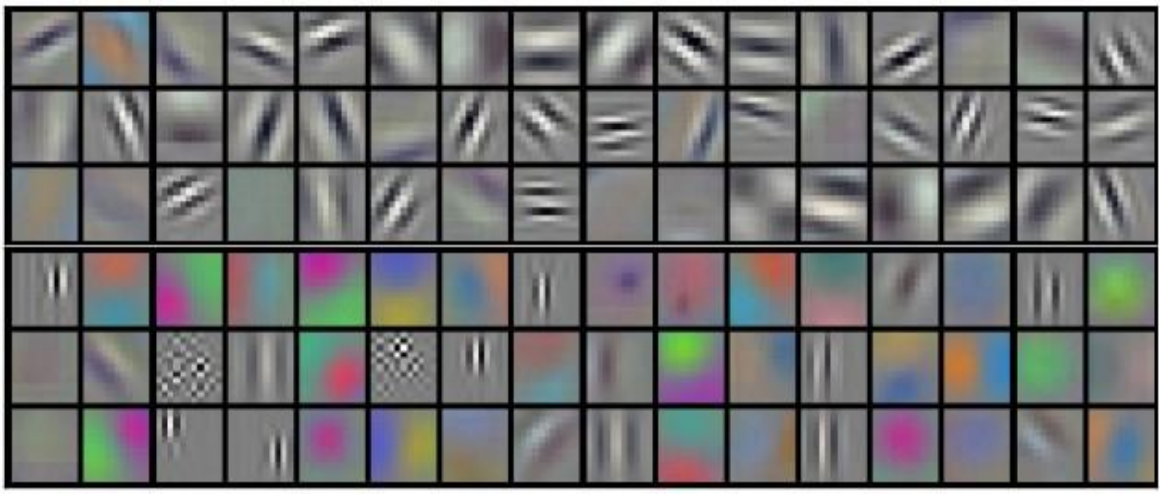


Figure 26: Example filters learned by Krizhevsky et al.

Each of the 96 filters shown above is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the convolutional layer output volume. [16]

3.2.5 Convolutional layer

Earlier we discussed the receptive field. Alongside the filter size there are 3 main parameters that we can change to modify the behavior of each layer. After we choose the filter size, we also have to choose the **depth**, **stride** and the **padding.** The **depth** of the output volume is a hyper parameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. Let us keep in mind that if we are using 20 filters, then after the first convolutional layer a single image creates 20 new images.

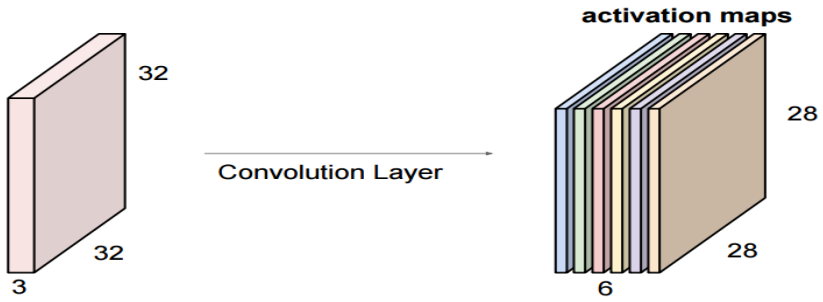


Figure 27: Input image of size 32x32x3 convolved with 6 5x5x3 filters each one looking for a particular pattern on the image.

The **stride** with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially. [23]

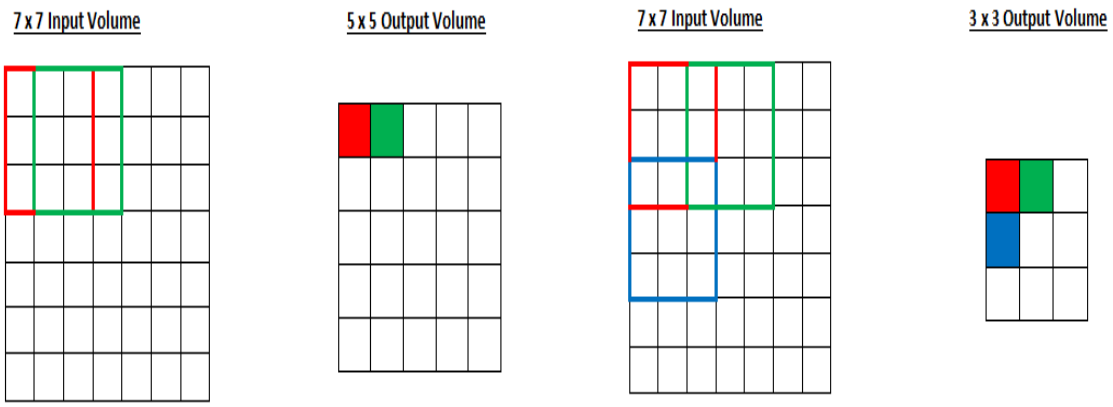


Figure 28: Left: Stride = 1, Right: Stride = 2

If we tried to set our stride to 3, then we would have issues with spacing and so we must make sure the receptive field fits on the input volume. Normally, increasing the stride means that we want the receptive fields to overlap less and smaller spatial dimensions is needed.

Sometimes it will be convenient to **pad** the input volume with zeros around the border. The size of this zero-padding is a hyper parameter. The nice feature of zero

padding is that it will allow us to control the spatial size of the output volumes. We should notice from figure 5, when we apply six 5 x 5 x 3 filters to a 32 x 32 x 3 input

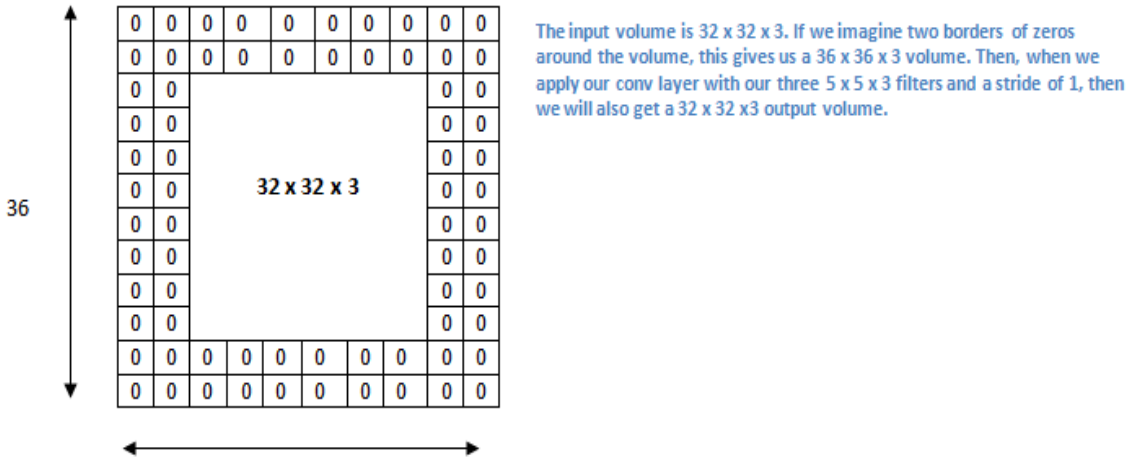


Figure 29: Padding a 32x32x3 image to obtain same input dimensions after convolution

volume, the output volume is 28x28x6. The spatial dimensions decrease. As we keep applying Conv Layers, the size of the volume will decrease faster than we would like. In the early layers of our network, we want to preserve as much information about the original input volume so that we can extract those low level features, and so if we want to keep the same input volume in our output for this particular example, we should use zero padding of 2 to obtain a 36x36x3 image.

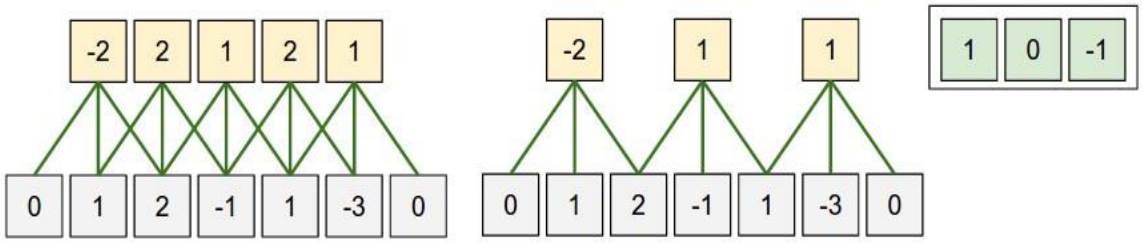


Figure 30: Effect of stride and padding.

In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1. Left: The neuron stridden across the input in stride of S = 1, giving output of size (5 - 3 + 2)/1+1 = 5. Right: The neuron uses stride of S = 2, giving output of size (5 - 3 + 2)/2+1 = 3.

3.2.6 Activation functions

After each convolutional layer, it is conventional to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the convolutional layers (just element wise multiplications and summations).

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions to be used in neural networks.

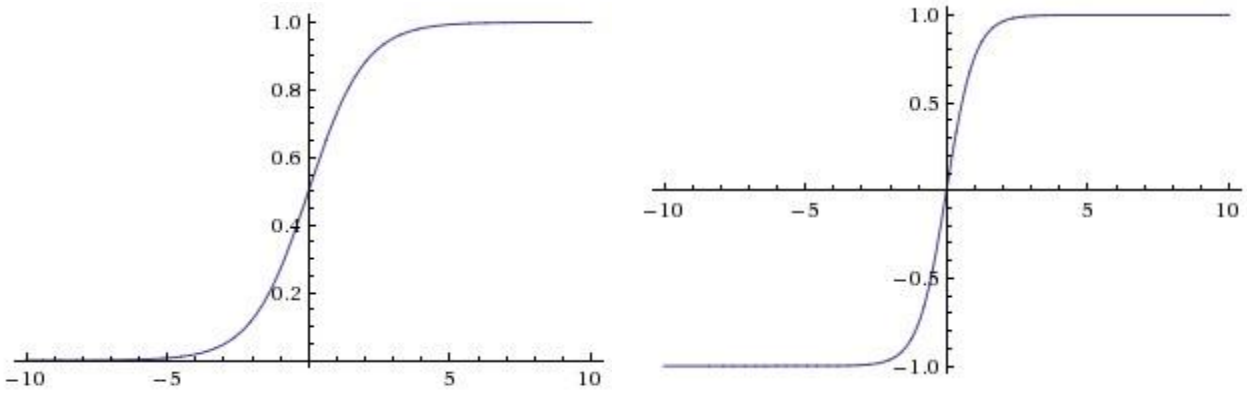


Figure 31: Left: Sigmoid function. Right: Tanh function.

Sigmoid: The sigmoid non-linearity has the mathematical form $\sigma(x) = \frac{1}{(1+e^{-x})}$ and is shown in the image above on the left. It takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1).

Tanh: The tanh non-linearity is shown on the image above on the right. It squashes a real-valued number to the range [-1, 1]. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity. Also note that the tanh neuron is simply a scaled sigmoid neuron.

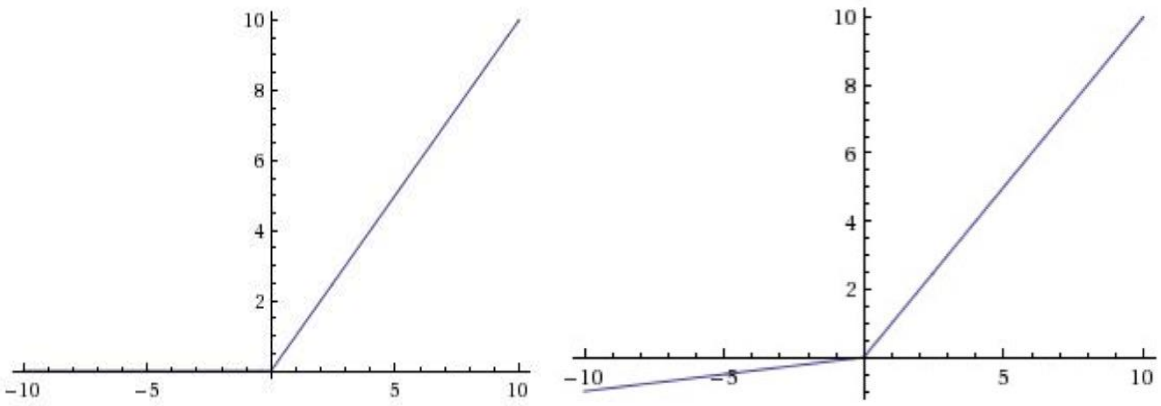


Figure 32: : Left: Rectified Linear Unit (ReLU) activation function. Right: Leaky Rectified Linear Unit (Leaky ReLU).

ReLU: The Rectified Linear Unit has become very popular in the last few years and is the one we are using in our research. It computes the function$f(x) = \max(0, x)$. In other words, the activation is simply thresholded at zero. There are several pros and cons to using the ReLUs:

- (+) It was found to greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
- (+) It helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.
- (+) Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- (-) Unfortunately, ReLU units can be fragile during training and can "die".

Leaky ReLU: Leaky ReLUs are one attempt to fix the "dying ReLU" problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope (of 0.01, or so). That is, the function computes $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$ where α is a small constant. The slope in the negative region can also be made into a parameter of each neuron.

Maxout: Other type of activation functions have been proposed that do not have the functional form $f(w^T x + b)$ here a non-linearity is applied on the dot product between the weights and the data. ne relatively popular choice is the Maxout neuron that generalizes the ReLU and its leaky version. The Maxout neuron computes the function $\max(w_1^T x + b_1, w_2^T x + b_2)$. Notice that both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have $w_1, b_1 = 0$). The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU). However, unlike the ReLU neurons it doubles the number of parameters for every single neuron, leading to a high total number of parameters. [24]

3.2.7 Pooling layer

It is common to periodically insert a Pooling layer in-between successive Convolutional layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height. The depth dimension remains unchanged. [1]

In our research we used Max pooling technique to down sample the input images, where MAX operations would be taking a max over 4 numbers in case of a 2x2 filter. It is worth noting that there are only two commonly seen variations of the max pooling layer found in practice: A pooling layer with filter size 3x3, stride = 2 (also called overlapping pooling), and more commonly filter size 2x2, stride = 2. Pooling sizes with larger receptive fields are too destructive.

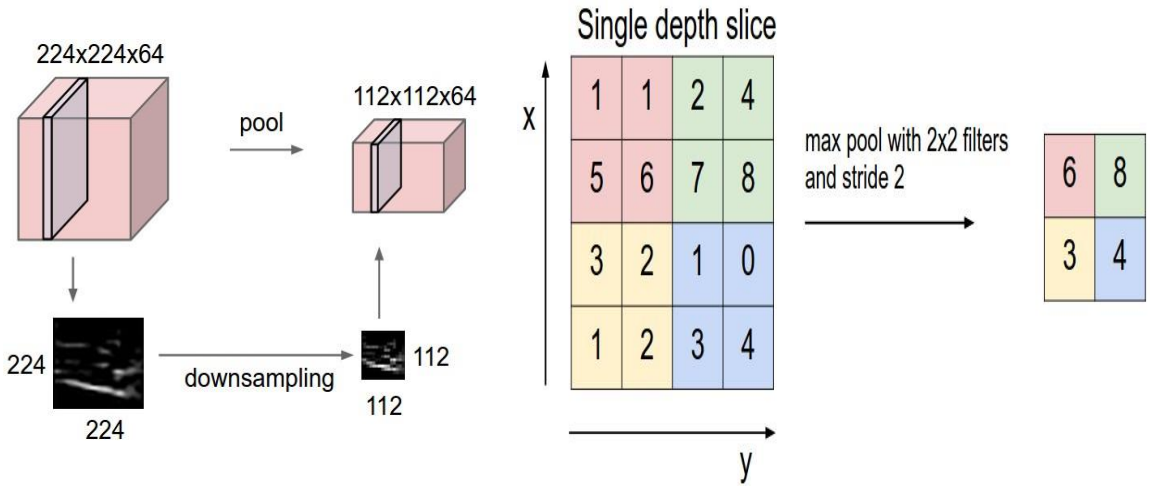


Figure 33: Pooling layer effect.

Pooling layer down samples the volume spatially, independently in each depth slice of the input volume. In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. The most common down sampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

The Sum pooling / Mean pooling work in a similar manner, by taking the sum of the inputs and dividing it by the receptive field size instead of taking the maximum.

The conceptual difference between these approaches lies in the sort of invariance which they are able to catch. Max pooling is sensitive to existence of some pattern in pooled region. Sum pooling (which is proportional to Mean pooling) measures the mean value of existence of a pattern in a given region.

Mean pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.

3.2.8 Fully connected layer

The Fully Connected layer is a traditional Multi-Layer Perceptron that uses a Softmax activation function in the output layer (other classifiers like SVM can also

be used). The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer.

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

Apart from classification, adding a fully-connected layer is also a usually cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better. Essentially the convolutional layers are providing a meaningful, low-dimensional, and somewhat invariant feature space, and the fully-connected layer is learning a possibly non-linear function in that space. [25]

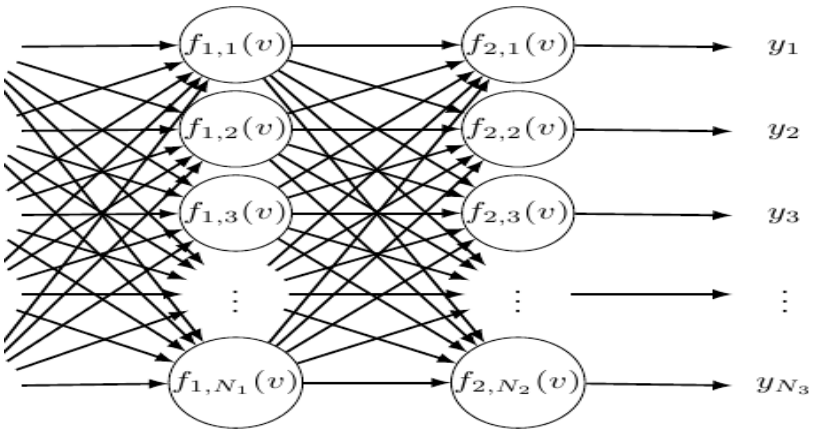


Figure 34: 2 fully connected layers.

Now that we have spoken about the 3 main layers that construct a CNN with several types of activation functions, check the image below to get a concrete sense of how it may look.

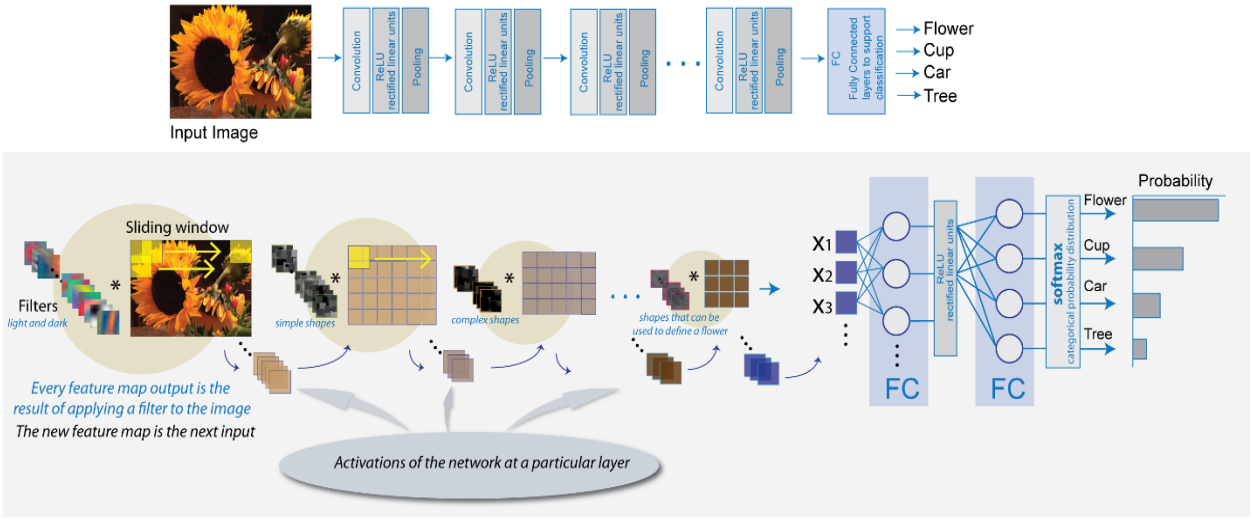


Figure 35: 3 Main layer CNN with several types of activation functions

3.2.9 Regularization

3.2.9.1 Regularization L2

The first main approach to overcome overfitting is the classical weight decay, which adds a term to the cost function to penalize the parameters in each dimension, preventing the network from exactly modeling the training data and therefore help generalize to new examples:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_{k} W_k^2$$

With W_k^2 a vector containing all the network parameters.

3.2.9.2 Dropout

The numerous nonlinear hidden layers in deep neural networks makes them very significant in a manner that it learns very complicated relationships between inputs and outputs. This leads to overfitting. To address the mentioned problem many ways have come to solve it such L1 and L2 regularization or creating a stopping criteria for the training when the performance of the validation is getting much worse. One technique which appeared to prove its robustness in solving such problems is Dropout [26]. It prevents overfitting by dropping out units (both hidden and visible) randomly in a neural network.

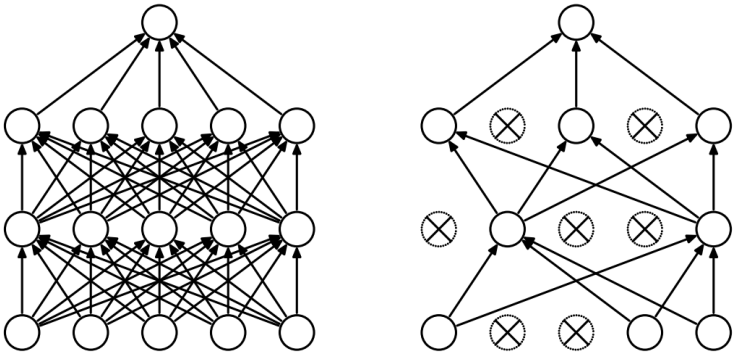


Figure 36: (a) Standard Neural Net, (b) After applying dropout

Standard backpropagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout breaks up these co-adaptations by making the presence of any particular hidden unit unreliable. Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout.

For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely.

One way to study the effect of dropout in all aspects is to test its improvements on various datasets. To study the effect of dataset size on dropout, it was observed that for extremely small data sets (100, 500) dropout does not give any improvements. As the size of the data set is increased, the gain from doing dropout increases up to a point and then declines.

3.2.9.3 Batch Normalization

The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers so that small changes to the network parameters amplify as the network becomes deeper, this change in distribution distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. We call this change in the input distribution Covariate shift. Addressing the problem of vanishing gradients for a saturating nonlinearity e.g. Sigmoid, it can't be used for deep networks as they tend to get stuck in the saturation region as the network grows deeper. Eliminating the aforementioned problems came with the proposed idea of Batch Normalization [27] as a layer within the architecture model layers, in which we normalize the input layer parameters which fixes their means and variances. Batch Normalization also reduces the dependence of gradients with respect to the parameters or their initial values Which allows to use much higher learning rates, taking in account the risk of divergence, that guarantee a faster training. Eventually, batch normalization introduced the possibility of using saturating nonlinearities by preventing the network from getting stuck in the saturation regime.

By whitening activations at every training step or even after certain interval we should consider these modifications with the optimization steps so the gradient descent step will not only update the parameters but also will update the normalization which reduces the effect of gradient step.

Due to the expensive computations of whitening of each layer, instead we will normalize each scalar feature independently, by making it have the mean of zero and the variance of 1. However, normalizing each input of a layer may change what the layer can represent. So, to preserve the identity transform, a pair of parameters (alpha of k), (beta of k) are presented such that they scale and shift the normalized value:

$$y\ (k)\ =\ \gamma\ (k)xb\ (k)\ +\ \beta\ (k)$$

These parameters are learned along with the original model parameters such that adjusting $\gamma\ (k) = \sqrt{Var[x\ (k)\ ]}$ and $\beta\ (k)\ =\ E[x\ (k)\ ]$ can restore and recover our original activations

But to achieve the desired efficacious advantages of batch normalization we need adjust some training parameters besides modifications in the network itself:

1- Increase learning rate.
2- Remove Dropout layer.
3- Reduce L2 weight regularization.
4- Accelerate learning rate decay.
5- Shuffle training examples.

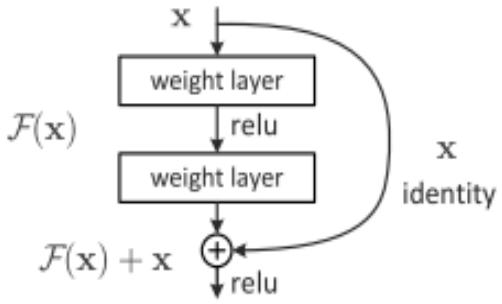
3.2.10 Data Augmentation

Data augmentation It is a method of boosting the size of the training set so that the model cannot memorize all of it. This can take several forms depending of the dataset.

For instance, if the objects are supposed to be invariant to rotation such as galaxies or planktons, it is well suited to apply different kind of rotations to the original images.

3.3 Residual Neural Networks(ResNet)

In recent years Deep Convolutional Neural Networks (CNN) demonstrated a high performance on image classification tasks. Experiments showed that the number of layers (depth) in a CNN is correlated to the performance in image recognition tasks. This led to the idea that deeper networks should perform better. Creating deep networks is not as simple as adding layers. One problem is the vanishing/exploding gradients, which hamper the convergence. This obstacle can be overcome by normalized initialization and intermediate normalization layers, so that networks start converging for stochastic gradient descent (SGD) using the backpropagation algorithm. Another problem is the degradation, if the depth of a network increases, the accuracy gets saturated and then degrades rapidly.



Instead of adding stacked layers directly, we explicitly let these layers fit a residual mapping. Formally, denoting the desired underlying mapping as H(x), we let the stacked nonlinear layers fit another mapping of F(x): = H(x) − x.

The original mapping is recast into F(x)+x. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. If an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

So each Residual Unit can be expressed in a general form:

$$\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l),$$
$$\mathbf{x}_{l+1} = f(\mathbf{y}_l),$$

where x(l) and x(l+1) are input and output of the l-th unit, and F is a residual function, h (x [l]) = x(l) is an identity mapping and f is a (ReLU) function.

The central idea of ResNets is to learn the additive residual function F with respect to h(x[l]), with a key choice of using an identity mapping h (x[l]) = x(l). This is realized by attaching an identity skip connection ("shortcut").

If f is also an identity mapping: x (l+1) = y(l), we can obtain:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l).$$

Recursively: x [l+2] = x [l+1] + F (x [l+1], W [l+1]) = x[l] + F (x[l], W[l]) + F (x[l+1], W[l+1]). etc., So forward propagation can be expressed by:

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

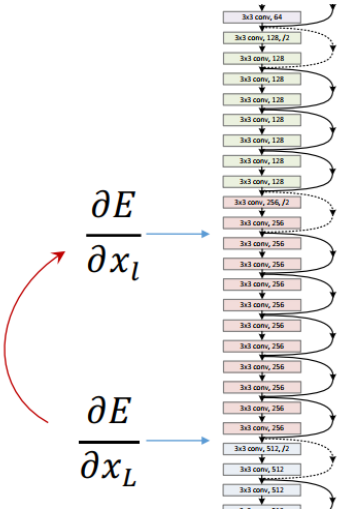
For any deeper unit L and any shallower unit (l), The general equation exhibits some properties:

- The feature x(L) of any deeper unit L can be represented as the P(L−1) feature x(l) of any shallower unit l plus a residual function in a form of $\sum_{i=l}^{L-1} F(x_i)$, indicating that the model is in a residual fashion between any units L and l.

- The feature x(L) = x(0) + $\sum_{i=0}^{L-1} F(x_i, W_i)$, of any deep unit L, is the summation of the outputs of all preceding residual functions (plus x[0]). This is in contrast to a "plain network" where a feature x(L) is a series of matrix-vector products

The general formula also leads to nice backward propagation properties. Denoting the loss function as ε, from the chain rule of backpropagation we have:

$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial E}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1} F(x_i)\right)$$



This equation indicates that the gradient $\frac{\partial E}{\partial x(l)}$ can be decomposed into two additive terms: a term of $\frac{\partial E}{\partial x(l)}$ that propagates information directly without concerning any weight layers, and another term of $\frac{\partial E}{\partial x(L)}\left(\frac{\partial E}{\partial x(l)}\sum_{i=l}^{L-1} F(x_i)\right)$ that propagates through the weight layers. The additive term of $\frac{\partial E}{\partial x(L)}$ ensures that information is directly propagated back to any shallower unit l.

Also the equation suggests that it is unlikely for the gradient $\frac{\partial E}{\partial x(l)}$ to be canceled out for a mini-batch, because in general the term $\frac{\partial E}{\partial x(L)}\left(\frac{\partial E}{\partial x(l)}\sum_{i=l}^{L-1} F(x_i)\right)$ cannot be always -1 for all samples in a mini-batch. This implies that the gradient of a layer does not vanish even when the weights are arbitrarily small.

Chapter 4

Experiment

In this chapter, we will propose our ideas and architectures, introducing a brief analysis on our dataset and how it affects the learning process.

4.1 Dataset

Dataset used in this thesis is the IRMA dataset supplied by the imageCLEF organization, it consists of a collection of 14410 x-ray image over different anatomical/biological/body orientation available in ".png" format. Images are split into 12677 training image and 1733 testing image. The dataset is annotated using the IRMA code which is a 13-character string containing four mono-hierarchal axes representing the information on technical, biological and anatomical details of the image in the form of TTTT-DDD-AAA-BBB (illustrated in Table 1). Each selection of characters' hierarchy represents the technical details of the image; sample images form the data set is depicted in Fig 37

T	Image modality
D	Body orientation
A	Body region examined
B	Biological system examined

Table 1: Sections of IRMA code

IRMA dataset is considered quite challenging to work on, due to several problems including data imbalance which has been noted as one of the most challenging aspects in the dataset, the high variability of subclasses inside every main class and sample density, different image sizes, brightness and scale of the object in the image. The dataset consists over 9 main classes with 192 hierarchal sub classes according to the IRMA code, in this thesis we used the five of the main classes (Cranium, Spine, Upper extremity, chest, Lower extremity) with the anatomical point of view (illustrated in Table1), discarding the classes with low sample density and high subclasses variability.

Figure 37: Examples from IRMA dataset

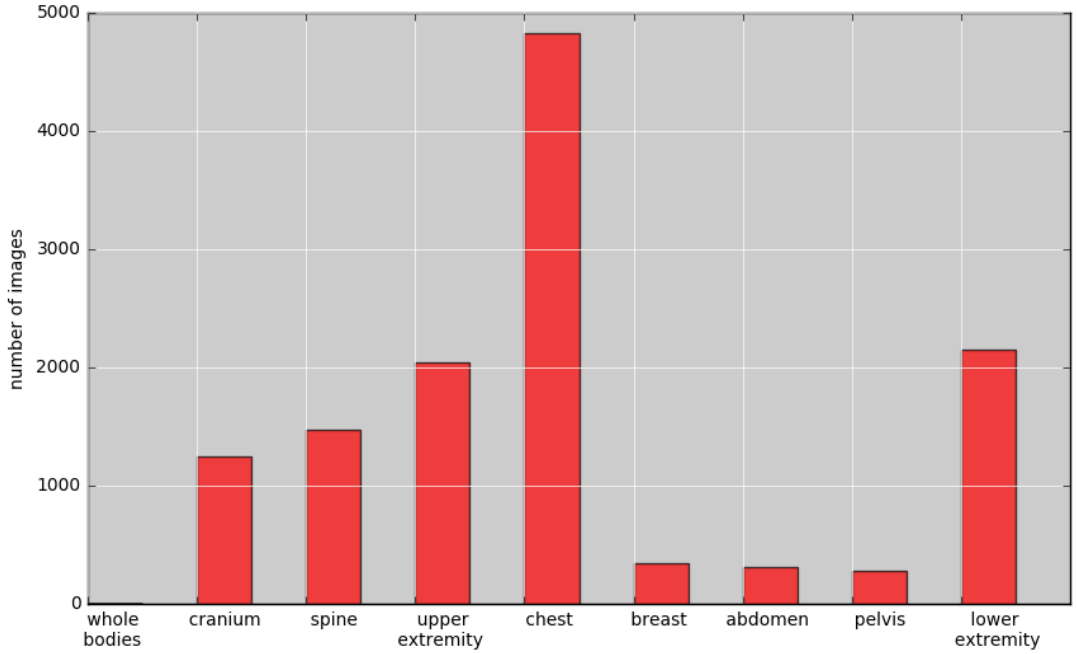


Figure 38: Training set classes Distribution

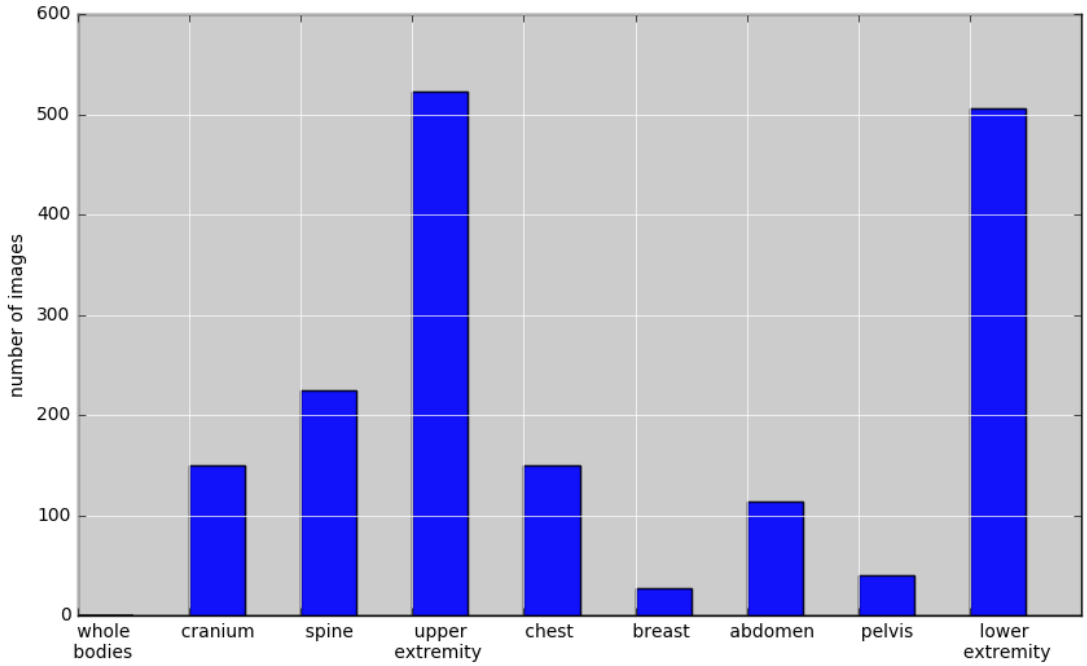


Figure 39: Testing set classes Distribution

4.2 Network Configuration

All ConvNets layer configurations are designed using the same principle inspired by [28]; [29] in order to measure the improvement accomplished by increasing depth in a fair setting. In this section, we first describe the layout of our principle configuration.

4.2.1 Architecture

To measure the effect of image vs patch similarity measuring, all configurations of the ConvNets have input of fixed size 120 x 120 image or 64 x 64 image grayscale image, an elementary preprocessing is to compute mean image across training set and subtract it from all training/validation/testing sets to center the image values around the mean, the image is passed through a stack of convolution-ReLu layers configuration inspired by the design principle of the VGGNet [29], where the convolutional layer filters sizes are of small receptive fields: 3 x 3 and 5 x 5 based on the layers configuration, the convolution stride is set to 1 pixel in all configurations. Spatial pooling in form of max pooling is used to reduce the dimensionality of feature maps, pooling is performed over a 2 x 2-pixel window with stride of two pixels. A stack of convolutional layers (with different order and depth based on the configuration) is followed by 3 fully connected layers of different channel sizes, having third layer classifying the image to one of the classes represented in the dataset. The final layer is a softmax layer. All hidden layers are followed by ReLu layer [28]

4.2.2 Configurations

The ConvNet configurations used in this report are presented in Table X, one in each column, all the configurations follow the same design principle, having fixed parameters and evaluating the effect of depth in improving the results.

4.2.3 Choice of Filter

The choice of filter sizes is inspired by the work of [29] in the VGGNet, using filters with smaller receptive field (3 x 3) and stride of one to convolve with every pixel in the image, we are able to represent larger filters by stacking multiple (3 x 3), for an example stacking two (3 x 3) filters has the same receptive field as one (5 x 5) filter as shown in the figure x and stacking three (3 x 3) filters has the same effective receptive field as one (7 x 7). Stacking more convolution filters has the benefits of increasing the non-linearity (every Conv Layer is followed by a ReLu layer), which increase the capability of the ConvNet decision function to be more discriminative, decreasing the number of parameters with a large factor, as for an example, assuming both input and output of a two (3 x 3) Conv Layer stack is preserved (having same number of channels C), the stack has number of parameters of 2 (3^2C^2) = $18C^2$ parameter, while one (5 x 5) Conv Layer has (5^2C^2) = $25C^2$,

And lastly, increasing the depth (number of layers) increases the ConvNet capacity to learn more discriminative features and to build a hierarchal representation from low level to high level features to outline the

4.3 Tools and Environments

Caffe: a clean and modifiable framework for state-of-the-art deep learning algorithms. The framework is a C++ library with Python and MATLAB bindings for training and deploying general purpose convolutional neural networks and other deep models efficiently. By separating model representation from actual implementation, Caffe allows experimentation and seamless switching among platforms for ease of development and deployment from prototyping machines to cloud environments

Caffe provides a complete toolkit for training, testing, fine-tuning, and deploying models. it's likely the fastest available implementation of these algorithms, making it immediately useful for both research and industrial purposes. Caffe separates the

model representation from the implementation. model definitions are written as configuration files using the Protocol Buffer language. Caffe supports network architectures in the form of arbitrary directed acyclic graphs. Upon instantiation, Caffe reserves exactly as much memory as needed for the network, and abstracts from its underlying location in host or GPU. Switching between a CPU and GPU implementation is exactly one function call. Caffe stores and communicates data in 4-dimensional arrays called blob of images (or other data), parameters, or parameter updates Models are saved to disk as Google Protocol Buffers1, which have several important features: minimal-size binary strings when serialized, efficient serialization, a human-readable text format compatible with the binary version, and efficient interface implementations in multiple languages, most notably C++ and Python.

MATLAB Neural Network Toolbox provides algorithms, pretrained models, and apps to create, train, visualize, and simulate both shallow and deep neural networks including convolutional neural networks (ConvNets, CNNs) and Autoencoders for image classification, regression, and feature learning.

FloydHub is a Platform-as-a-Service for training and deploying your deep learning models in the cloud. Floyd comes with fully configured CPU and GPU environments primed for deep learning. It includes CUDA, cuDNN and preinstalled Deep learning frameworks like Caffe, we used a NVIDIA Tesla K80 GPU hosted on floyhub for training and testing our networks, also we used an offline machine with NVIDIA GTX 960M balancing the load of training the networks between the offline and the online environments.

4.4 Training

Training procedure of the ConvNets followed the well-known steps, training is accomplished by optimizing a multinomial logistic regression function using minibatch (of size 256) gradient descent with momentum (set to 0.9) based on backpropagation [30], training is regularized with L_2 penalty set to XX. Drop out regularization layers are introduced in some networks configurations after the first two fully connected layers, drop ratio set to 0.5. Dataset is split into train/validate/test sets; validation set is used to control number of iterations using early stop criteria. weights are pre-initialization using Xavier initializations [31]

Chapter 5

Results and Discussion

In this chapter, we report the results of both hand crafted based features and deep learning based ones, evaluate which performance is more efficient and easy to implement.

5.1 Accuracy

Over the years, various works like [20], [21], [22], [23] have considered 1617 images in 6 classes, 9100 images in 40 classes, 6231 images, 5000 images in 20 classes respectively, from the IRMA dataset. Hence, it is very difficult to compare the proposed method with the previous works directly. The performance of the proposed architectures is evaluated by measuring classification accuracy defined as follows:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FN+FP}$$

Where TP = true positive, FN = false negative, FP = false positive, and TN = true negative.

The results obtained after training the neural networks on the dataset are illustrated in the table 4, compared to another hand crafted features, it is evident that best performing CNN configurations can outperform handcrafted features easily on the classification task in CBIR system [7].

Feature	**Accuracy [%]**
Fourier Mellin	46.9
Gabor histogram	75.6
Tamura histogram	80.7
Color histogram	82.1
Local features	87
Network A	43.1
Network B	92.03
Network C	92.69
Network D	89.28
Network E	88.38
Network F	89.15
Network G	90.38

Table 2: Performance comparison of proposed architectures and hand-crafted features

5.2 Effect of Regularization

To prevent the neural network from overfitting, dropout (dpout) [24] is used in the first two fully-connected layers, where the output of each hidden neuron is set to zero with probability 0.5. In this way, the neural network will sample a different architecture when an input image is presented, benefiting the generalization of the neural network. Table 5 demonstrated the effect of dropout technique on the accuracy of the architectures.

	w/o Aug [%]	w/ Aug [%]
Network	5 classes	5 classes
A	43.1	-
B	91.11	-
C	-	-

Table 3: Effect of Augmentation on Network Performance

	w/o dpout [%]		w/ dpout [%]	
Network	3 classes	4 classes	3 classes	4 classes
E	91.81	86.18	92.04	87.41
F	91.81	85.73	93.16	87.93
G	-	-	-	-

Table 4: Effect of Dropout on Network Performance

Data augmentation is another technique that we use to prevent overfitting, by boosting the size of the training set so that the model cannot memorize all of it. This can take several forms depending on the dataset. For instance, if the objects are supposed to be invariant to rotation such as x-ray images of upper and lower extremity, it is well suited to apply different kinds of rotations to the original images in order increase the model capacity to perceive variance in the images. The augmentation techniques used in this paper is XX. It is evident that the use of regularization boosts the performance of the CNN by increasing the generalization ability for better classification results. In table 6 we represent the effect of drop out and augmentation on CNN accuracy.

	w/o Aug [%]		w/ Aug [%]	
Network	3 classes	4 classes	3 classes	4 classes
E	91.81	86.18	92.51	87.54
F	91.81	85.73	92.49	87.09
G	-	-	-	-

Table 5: Effect of Augmentation on Network Performance

5.3 End to End CBIR

After training several networks, best performing models are used to test the effectiveness of the representations learned in an end to end CBIR system. Precision and recall are chosen to evaluate system performance. Precision is the fraction of retrieved instances that are relevant to the query, while recall is fraction of the instances that are relevant to the query that are successfully retrieved, precision-recall curve of the end to end system is shown in figures.

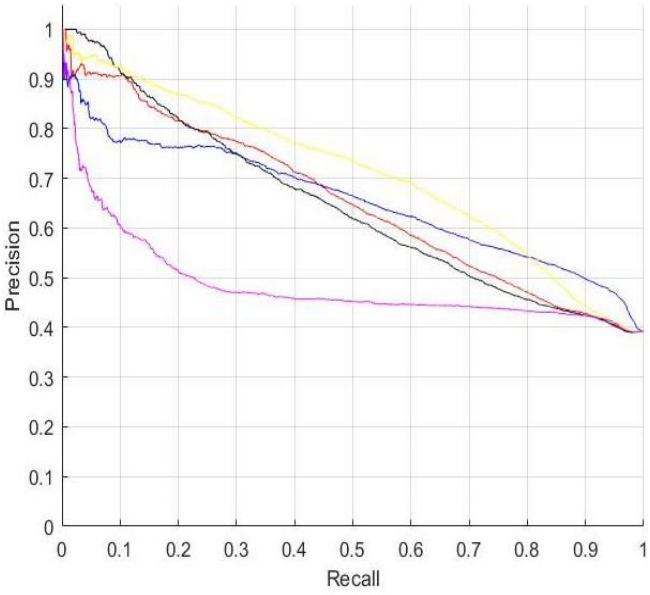


Figure 40: Limb class PR-Curve

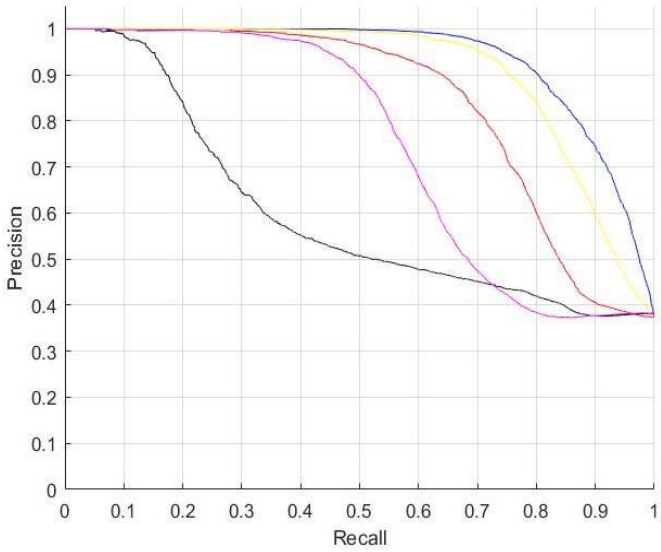


Figure 41: Chest class PR-Curve

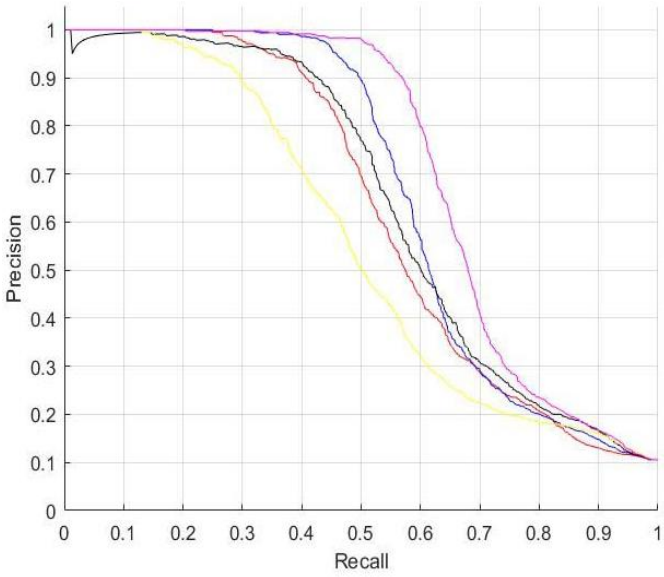


Figure 42: Cranium class PR-Curve

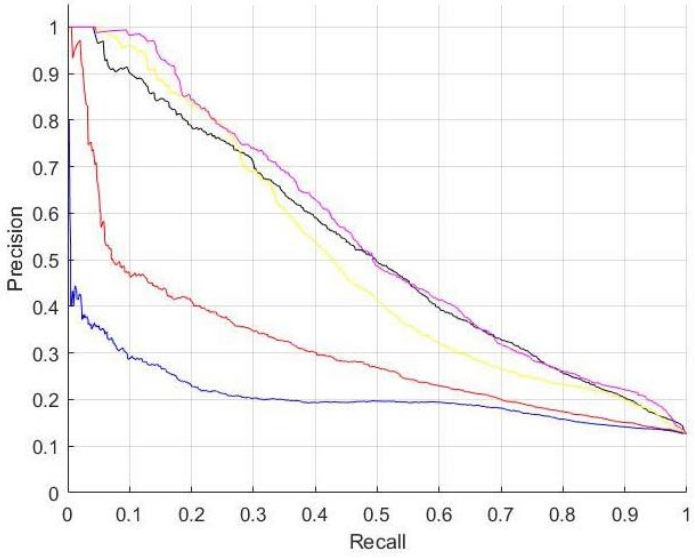


Figure 43: Spine class PR-curve

	Accuracy [%]	
Network	3 classes	4 classes
E	93.24	88.38
F	93.84	89.15
G	93.24	90.38

Table 6: Effect of Augmentation and Dropout combined

5.4 ResNet Results

A	B	C	D	F
Conv1(3-10) Conv2(3-10)	Conv1(5-10) Conv2(5-10)	Conv0(3-10)	Conv0(5-10)	Conv0(3-10)
		MaxPool		MaxPool
Res(conv1+conv2)		Conv1(3-10) Conv2(3-10)	Conv1(5-10) Conv2(5-10)	Conv1(3-10) Conv2(3-10)
MaxPool				
Conv3(3-10) Conv4(3-10)	Conv3(5-10) Conv4(5-10)	Res(conv1+conv2)		Res(conv1+conv2)
		Conv3(3-10) Conv4(3-10)	Conv3(5-10) Conv3(5-10)	MaxPool
Res(conv3+conv4)				Conv3(3-10) Conv4(3-10)
MaxPool		Res(conv1+conv2)		
FC-9610		MaxPool		MaxPool
FC-8192		FC-9610		FC-640
FC-classes		FC-8192		FC-512
Softmax		FC-classes		FC-classes
		Softmax		Softmax

Table 7: ResNet Results on different models

	w/o Batch Normalization [%]	w/ Batch Normalization [%]
A	87.1	-
B	88.6	-
C	91.1	92.4
D	89.7	90.3
F	-	91.8

Table 8: ResNet Performance with BatchNorm and without BatchNorm

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we explored the performance of CNN to extract representative and discriminative features from x-ray images to be used in a CBIR system, different configurations for CNN with the study of effect of adding drop-out and data augmentation were tested to report the results using coarser to finer training classes (three, four and five). ResNet architecture was employed while increasing the depth to overcome the vanishing gradient problems. The best performing configuration demonstrated the ability to outperform hand crafted features.

6.2 Future Work

In our future work, we will focus on increasing the depth of the network, while using better data preparation and augmentation techniques and strong regularizes "for an example batch normalization" to avoid over fitting problems. We will utilize more of the dataset classes to explore the ability of CNN to perform under heavily imbalanced dataset.

6.3 Bibliography

[1] Müller, H., Michoux, N., Bandon, D., &Geissbuhler, A. (2004). A review of content-based image retrieval systems in medical applications—*clinical benefits and future directions.International journal of medical informatics*, 73(1), 1-23.

[2] Wei, C. H., Li, C. T., & Wilson, R. (2009). A Content-Based Approach to Medical Image Database Retrieval.

[3] Akgül, C. B., Rubin, D. L., Napel, S., Beaulieu, C. F., Greenspan, H., &Acar, B. (2011). Content-based image retrieval in radiology: current status and future directions. *Journal of Digital Imaging*, 24(2), 208-222.

[4] Lehmann, T. M., Güld, M. O., Thies, C., Fischer, B., Spitzer, K., Keysers, D. Ney, H., Kohnen, M., Schubert, H., & Wein, B. B. (2004): Content-based image retrieval in medical applications. *Methods of Information in Medicine 43(4):354–61*.

[5] Emmanuel, M., Babu, D. R. R., Potdar, G. P., Sonkamble, B. A., & Game, P. (2007): Content-based medical image retrieval. *IET-UK International Conference on Information and Communication Technology in Electrical Sciences (ICTES 2007), 712– 717, IEEE*.

[6] Wanjale, K., Borawake, T., & Chaudhari, S. (2010): Content based image retrieval for medical images techniques and storage methods-review paper. *IJCA Journal, 1(19):105–107*.

[7] Tamura, H., Mori, S., Yamaeaki, J.: Texture features corresponding to visual perception. *IEEE Trans. on Systems, Man and Cybernetics 6, 460–473 (1978)*

[8] R.Jain, R. Kasturi and B.G. Schunck "Machine Vision" Published by McGraw-Hill, Inc., ISBN 0-07-032018-7, 1995, ch.7, sec.7.2,pp 234-239

[9] Wikipedia contributors, "Gabor filter," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Gabor_filter&oldid=785195174

[10] Wikipedia contributors, "Local binary patterns," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Local_binary_patterns&oldid=789135291

[11] Wikipedia contributors, "Histogram of oriented gradients," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Histogram_of_oriented_gradients&oldid=768396739

[12] D.G. Lowe," Discriminative Image Features from Scale-Invariant Keypoints," *CA Patent 5 236 445, January,5,2004*

[13] Wikipedia contributors, "Mellin transform," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Mellin_transform&oldid=754008020

[14] Hinton, G. E. and Salakhutdinov, R. (2006). "Reducing the dimensionality of data with neural networks. *Science*." 313 (5786), 504–507.

[15] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). "Extracting and composing robust features with denoising autoencoders*." In ICML 2008 241,*515

[16] A.Karpathy, Convolutional Neural Networks for Visual Recognition, Retrieved from http://cs231n.github.io/convolutional-networks/

[17] http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[18] Y. LeCun, J. S. Denker, and S. A. Solla. "Optimal brain damage." *In Advances in Neural Information Processing Systems 2,* pages 598–605, 1990.

[19] W. Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. "Understanding the effective receptive field in deep convolutional neural networks." *In Advances in Neural Information Processing Systems,* pp. 4898-4906. 2016.

[20] K. Simonyan, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556 (2014).*

[21] K. He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Identity mappings in deep residual networks*." In European Conference on Computer Vision, pp. 630-645. Springer International Publishing, 2016.*

[22] K. He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." *In Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 770-778. 2016.

[23] A. Deshpande, A beginner's guide to understanding Convolutional Neural Networks, Retrieved from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[24] A. Karpathy, Convolutional Neural Networks for Visual Recognition, Retrieved from http://cs231n.github.io/neural-networks-1/

[25] U. Karn, An Intuitive Explanation of Convolutional Neural Networks, Retrieved from https://ujjwalkarn.me/2016/08/11/intuitive-explanation-ConvNets/

[26] Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research* 15, no. 1 (2014): 1929-1958.

[27] Sergey Ioffe, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *In International Conference on Machine Learning*, pp. 448-456. 2015.

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

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

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

[31] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics,* pp. 249-256. 2010.