University College of Southeast Norway
Faculty of Technology and Maritime Sciences
-
Master Thesis in System Engineering With Embedded Systems
Kongsberg Department of Engineering
May 19, 2017

Qinghui LIU

Deep Learning Applied to Automatic Polyp Detection in Colonoscopy Images

Abstract

Deep learning is an improvement to the neural network that contains more computational layers that allow for higher levels of abstraction and prediction in the data. So far, it is becoming a leading machine learning tool for general imaging and computer vision. Current trends in research have also demonstrated that deep convolutional neural networks (DCNNs) are very effective in automatically analyzing images. However, the requirement of large number of annotated samples prohibits its wide use in medical image analysis, since collecting and labeling a large amount of data is difficult due to the challenges in obtaining the data from the medical domain.

Polyps are known as possible colorectal cancer precursors, and their early detection is of great importance, but highly challenging from an image processing standpoint. In this work, we evaluate several state-of-the-art machine learning techniques and deep learning methods in the medical image processing domain and research solutions about how they can be more efficiently utilized for automatic detection of polyps in endoscopy and colonoscopy images.

This work proposes an effective transfer learning (TL) framework relying on pre-trained DCNNs using a large collection of natural ImageNet images. This has been achieved by evaluating various kinds of cutting edge techniques including both traditional machine learning methods by training feature-based classifiers from scratch and modern DCNNs algorithms with (TL) and fine tuning pre-trained models. We transfer learned ImageNet weights as initial weights, and then fine-tune this model combined with a new deep classifier called fully connected networks (FCNs) with data augmentation and patch-extraction of colonoscopy images to automatically detect polyps. In case of insufficient colonoscopy images, patch-based data augmentation and deep features extracted using TL strategy can provide sufficient and balanced classification information.

With the proposed TL framework with our optimized hyper-parameters, the system achieved overall 96.00% polyp detection precision and sensitivity, which outperformed the traditional machine learning classification methods in each defined performance metric. Moreover, the TL framework proposed is scalable and flexible so that it can easily be extended to include other types of disease detection in the future and also be able to integrate one more DCNNs model to boost its generalizing capabilities.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my thesis advisors, Dr. António L. L. Ramos at USN, Norway, and Dr. Sergio L. Netto at UFRJ, Brazil, for their guidance and support while conducting this research work as well as in the writing of this thesis. I could not have had better advisors and mentors for my master study and research. I am very thankful to Dr. António L. L. Ramos for giving me this task and to Dr. Sergio L. Netto and his research group, specially Lucas Cinelli and Bruno Afonso, for sharing their expertise and experience in the field of Machine Learning.

I would like to thank Dr. Olaf Hallan Graven, our head of department, for providing all the necessary means and facilities to carry out this task. I also take this opportunity to express my gratitude to my lecturers and faculty members at USN for sharing their expertise, and sincere and valuable guidance and encouragement.

I would also like to thank all my classmates, specially my lab-mates Zhili Shao, Blessing, Vegard, and David, for the stimulating discussions and for all the fun we have had in the past two years. My sincere thanks also go to Mr. Paul Stupkin for reading part of the manuscript and for the valuable comments on the English grammar.

Last but not the least, I would especially like to thank my family. Words cannot express how grateful I am to my mother-in-law, father-in-law, my mother and father for all of the sacrifices you have made on my behalf. To my beloved wife, GU Yan, I can’t thank you enough for supporting me for everything, and for encouraging me throughout this experience.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence.
API	**A**pplication **P**rogramming **I**nterface.
BN	**B**atch **N**ormalization.
CAD	**C**omputer **A**ided **D**iagnosis.
CLBP	**C**ompleted **L**ocal **B**inary **P**attern.
CNN	**C**onvolutional **N**eural **N**etwork.
ConvNet	**Conv**olutional **Net**work.
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture.
cuDNN	**cu**DA **D**eep **N**eural **N**etwork library.
CV	**C**ross **V**alidation.
CWC	**C**olor **W**avelet **C**ovariance.
DCNN	**D**eep **C**onvolutional **N**eural **N**etworks.
DL	**D**eep **L**earning.
DT	**D**ecision **T**ree.
FCN	**F**ully **C**onnected **N**etwork.
GI	**G**astro-**I**ntestinal.
GP	**G**aussian **P**rocess.
GPU	**G**raphics **P**rocessing **U**nits.
ILSVRC	**I**mageNet **L**arge-**S**cale **V**isual **R**ecognition **C**hallenge.
KNN	**K**-Nearest **N**eighbors.
LBP	**L**ocal **B**inary **P**attern.
ML	**M**achine **L**earning.
MLP	**M**ulti-**L**ayer **P**erceptron.
NN	**N**eural **N**etwork.
PCA	**P**rincipal **C**omponent **A**nalysis.
RBF	**R**adical **B**asis **F**unction.
ReLU	**Re**ctified **L**inear **U**nit.
RF	**R**andom **F**orests.
ResNet	**Res**idual **Net**work.
RMS	**R**oot **M**ean **S**quared.
ROI	**R**egion **O**f **I**nterest.
SGD	**S**tochastic **G**radient **D**escent.
SIFT	**S**cale **I**nvariant **F**eature **T**ransform.
SVM	**S**upport **V**ector **M**achine.
TL	**T**ransfer **L**earning.
TPE	**T**ree-structured **P**arzen **E**stimator.
TSCH	**T**exture **S**pectrum and **C**olor **H**istogram.
TSH	**T**exture **S**pectrum **H**istogram.
VCE	**V**ideo **C**apsule **E**ndoscopy.
VGG	**V**isual **G**eometry **G**roup.
WCE	**W**ireless **C**apsule **E**ndoscopy.

Chapter 1

Introduction

1.1 Background

Colorectal cancer is the third most common type of cancer in men and women in the United States of America and also the second highest cause of cancer deaths [1]. Early detection of polyps, protrusions from the colon surface, is vital to the prevention of colorectal cancers, since colorectal cancer is highly curable when it is detected early. It often begins as a benign polyp of the tissue lining the colon or rectum and, without proper treatment at early stage, it will eventually develop into a cancer. Therefore, one of the major goals of endoscopy and colonoscopy is early detection of polyps and cancers.

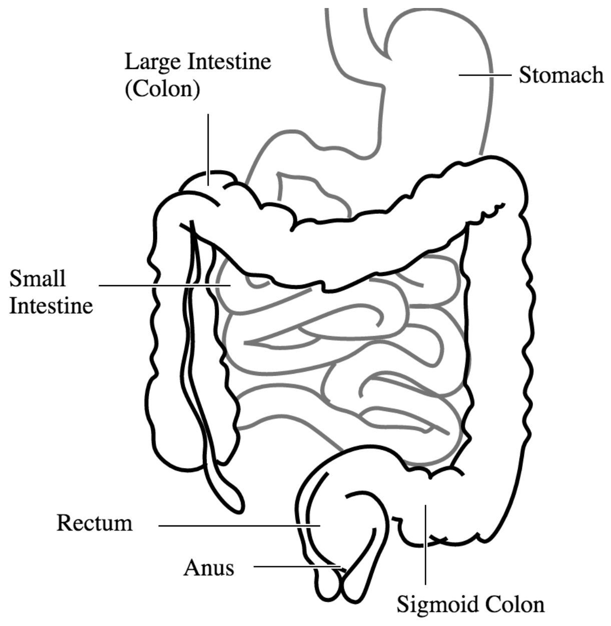


FIGURE 1.1: Gastro-Intestinal Track Diagram (Image from Wikimedia Commons).

Endoscopy, colonoscopy and wireless capsule endoscopy

The conventional endoscopy performs a visual inspection of the gastro-intestinal (GI) track (see Figure 1.1) using a lighted, flexible tube with a tiny video camera at its tip (endoscope). There are two basic types of endoscopy: upper endoscopy carried out by inserting a flexible endoscope through the mouth to collect images from the esophagus,

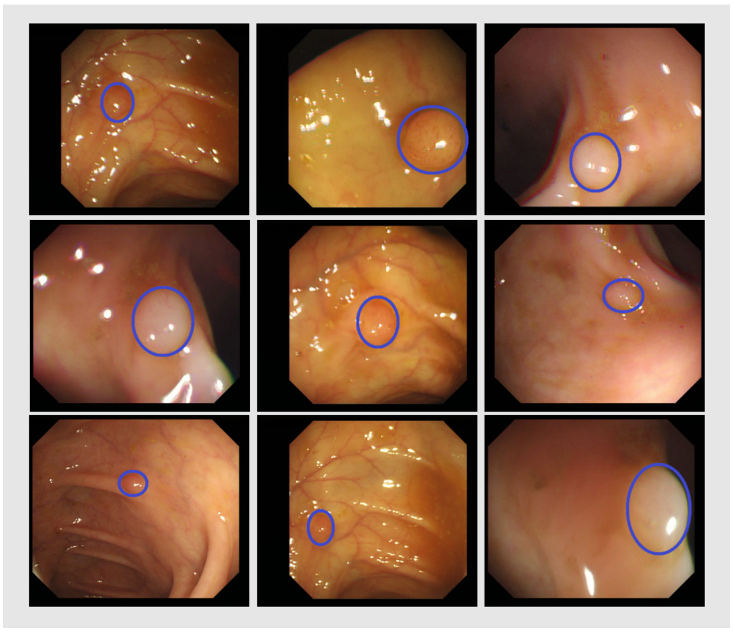


FIGURE 1.2: Examples of the type of polyp in colonscopy images.

stomach, and small intestines; and Colonoscopy, performed by inserting the endoscope via the anus to examine the large intestine, colon, and rectum.

Wireless (video) capsule endoscopy (WCE/VCE) is a noninvasive technology designed primarily to provide diagnostic imaging of the small intestine in a less invasive manner. The capsule measures 26 by 11 mm, the size of a large vitamin pill, and is propelled through the small bowel by peristalsis. Wireless capsule endoscopes have also been developed for the esophagus and colon, but their use in those areas is not yet as popular [54]. Colonoscopy is still the preferred technique for colon cancer screening and prevention.

Appearances of polyps

Polyps appears in different shapes ranging from flat to predunculated forms. The flat polyps are often attached to the colon wall by their base, predunculated polyps are attached via a stem. The figure 1.2 shows some examples of colonic polyps extracted from different colonscopy videos from CVC-ColonDB.

Besides, a polyp may appear in scale depending on the distance between the polyps and the colonoscopy camera. This is shown in the Fig.1.3 where the same polyp appears different in scale in each image.

1.2 Problem statement

Colonoscopy is an operator dependent procedure wherein human factors such as fatigue and insufficient attentiveness can lead to the miss-detection of polyps during long and back-to-back procedures. The average polyp miss-rate is estimated around 4-12%. Patients with missed polyps may be diagnosed with a late stage colorectal cancer with the survival rate of less than 10%. In addition, though WCE is now also used for colon examination, including the identification of polyps, after ingestion of the capsule,

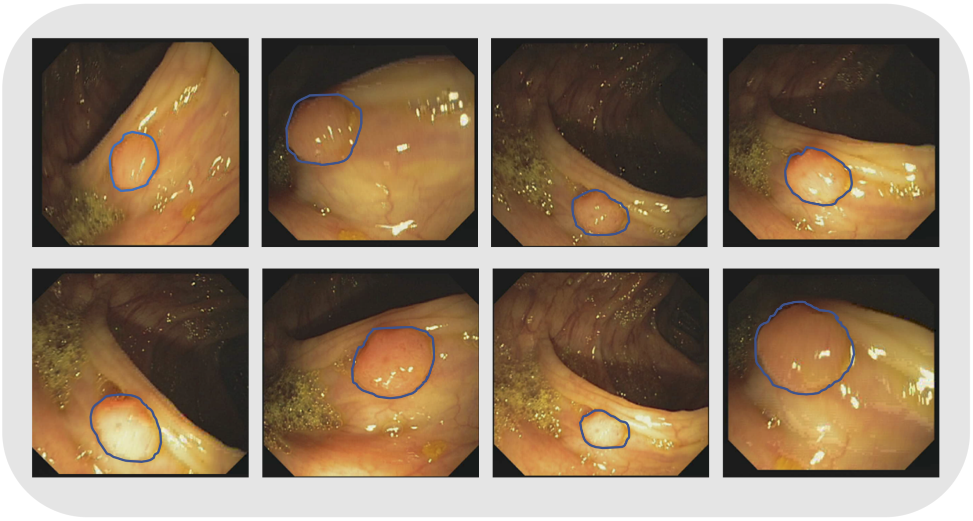


FIGURE 1.3: A polyp appears in different scales and shades in colonscopy videos from CVC-ClinicDB.

over 50,000 images are captured for analysis, which is time-consuming for physicians to assess manually.

To reduce the miss-detection of polyps caused by human factors and the cost and time of screening a large number of colonoscopy or WCE frames, a large number of techniques have been studied and exploited recently for the automatic detection of polyps in colonic images.

However, computer-aided automatic detection of polyps is still a difficult task due to the variety of shape, size, color, texture and size scale in the captured images. Additionally, the complex structure of the GI tract, similar color between polyp and non-polyp regions, poor image quality, and image variation of the same polyp caused by frequent camera angle changes creates further challenges.

1.3 Motivation

Current trends in research have demonstrated that deep learning methods, especially deep convolutional neural networks (DCNNs), are very effective for automatic analysis of images. So far, DCNNs have become a leading machine learning tool for general imaging and computer vision. Indeed, recent advances in deep learning frameworks and and methods have shown great potential to enhance the performance in computer vision applications, owing to their robust learning capabilities [17]. This captured our curiosity to explore and develop an effective approach based on cutting edge DL algorithms to solve a real world problem in medical image analysis. This work, focusing on automatic polyp detection, can potentially be a life savior, and builds upon our initial study presented in [31].

1.4 Objective

The objective of this work is to develop high performance, scalable and reliable automated polyp detection systems that can tolerate polyp variability. By handling differences such as shape, size, color, and texture, computer-aided automatic polyp detection systems become more feasible in clinical practice.

1.5 Approach

To achieve our objectives and desired outcomes, we chose the SCRUM methodology. This choice was based on the type of project to be carried out. This method allowed us to achieve maximum efficiency using iterative weekly sprints on which the work done the last week was reviewed and new tasks were organized and defined for the next week. We proposed the following sub-tasks:

- Researching work on the topics related to automatic polyp detection
- Studying and evaluating imaging processing algorithms and the state-of-the-art machine learning approaches.
- Extensively studying DCNNs algorithms and choosing the most appropriate models for automatic polyp detecting tasks.
- Developing pre-processing techniques for dataset preparation and performing primarily experiments on the domain dataset.
- Designing and implementing the DCNNs models for the detection of polyps.
- Performing extensive tests and fine-tuning the models to obtain the best performance.
- Final evaluation and suggesting future work.

We first studied the literature that focused on image processing algorithms and machine learning methods for polyp classification. We then built tools to evaluate these techniques. Meanwhile, we extensively studied and investigated the newest DCNNs architectures which could be employed in our work. Then we developed a scalable transfer learning framework to utilize pre-trained DCNN models for polyp detection tasks. After the proposed DCNNs models were implemented, we performed extensive tests and fine-tuning of the models in order to obtain the best performance. Eventually, we evaluated all supposed methods using comprehensive performance metrics and suggested an outlook on future work.

1.6 Outline

The remainder of this thesis is organized as follows.

- In Chapter 2, we provide an overview literature discussion on topics related to automatic polyp detection. This covers a brief description about traditional texture/shape based methods, conventional machine learning classification and deep learning concepts.

- Chapter 3 contains our complete methodologies and various frameworks for polyp detection by using both traditional machine learning methods and new DCNNs algorithms, which covers the description of low-lever image processing techniques, various popular classifiers and the cutting edge DCNNs framework which are employed in our work.
- In Chapter 4, we present in-depth information about our design and implementation. We discuss experimental results we have performed and further evaluate the proposed methods by well defined performance metrics.
- Chapter 5 provides conclusions of our work, summarizing our main contributions and achievements, and providing a suggested outlook on future work.

Chapter 2

Literature review

This chapter covers general aspects of ML and DL methods in order to build the necessary foundations to understand to scope and results presented in this work. First, an overview of machine learning techniques is given with a brief discussion of different learning types such as supervised and unsupervised learning, reinforcement learning and so on. Subsequently different low-level feature extraction approaches, which include the texture, shape and fusion of texture and shape features, are presented separately in the context of automatic polyp detection. Next, the chapter focuses on discussing deep learning methods, which represent the current state-of-the-art of current work and future trends. We first present deep learning concepts in general. Then several cutting edge DCNN models including AlexNet, VGG Net, GoogLeNet and ResNet, are described in detail since DCNN models are used as an important part of our work. In the subsequent sections, we analyze different deep learning applications for the automatic detection of polyp and publications related to this topic, which are grouped into two separate sections namely CNN-based CAD systems and pre-trained CNNs according to their utilized methods. Finally, we summarize and evaluate the results against our specific requirements.

2.1 Machine learning

The fields of Machine Learning (ML) and Deep Learning (DL) have been experiencing great progress in recent years and many useful techniques have been developed. These techniques are currently playing an important role in fields such as medical image processing and computer-aided diagnosis (CAD).

2.1.1 Overview

The typical goal of machine learning is to determine a mapping from input patterns to an output value [4]. A machine learning algorithm can be expressed as a function $y(x)$ that uses a input x and generates an output y. The output is usually encoded in the same way as the target vectors [4]. The form of the function $y(x)$ is determined during the training or learning phase, based on a training data set. Once the model is trained, it is then using new date referred to as the test set.

Machine learning algorithms are typically classified into three categories, based on the nature of the training signals or feedback to the learning system, as follows [43]:

- **Supervised learning**: In supervised learning problems, the training data is made up of tuples (x_i, y_i), where x_i is the input and y_i the corresponding target vector [10]. The goal is to learn a general rule, also called mapping function $f : X \rightarrow$

Y, that maps inputs to outputs. Supervised learning tasks can be further classified into classification and regression categories based on the desired output of a machine-learned system [4]:

  - In classification, inputs are grouped into two or more classes or categories, where the output variable is a category as well, e.g., "disease" or "no disease".
  - In regression, the output is a continuous real value rather than a discrete category, such as the prediction of the price a house.

- **Unsupervised learning**: In unsupervised learning tasks, no corresponding labels are given to the algorithms during training process. The algorithms are left to their own devises to find structure in the input data (x). Unsupervised learning problems can be further divided into clustering and density estimation problems as described in the following.
  - In clustering, the goal is to discover the inherent groups in similar examples based on measured or perceived similarities from the input data, such as grouping polyps by shapes.
  - In density estimation, the objective is to determine the distribution of the input data in some space.
- **Reinforcement learning**: It is concerned with the interaction tasks with a dynamic environment in which it can choose and perform a suitable actions in a given scenario, such as driving a car or playing a game [4].

In most of machine learning techniques the main stages are the feature extraction/descriptor step, and a decision-making stage called a classification step. Additional steps could be added prior to the feature extraction stage such as image smoothing or noise filtering and region-of-interest(ROI) selection.

There are primarily two types of features namely, the shape/geometric features and texture-color features. Both types of features have been utilized in the literature for polyp detection in medical images. To improve further the quality of the features and to have more information acquired on the images, feature fusion approach have been employed as well. This is done by combining geometric and textual features of the image to benefit from the information that both provide.

2.1.2 Texture features

The early work of Iakovidis et al. [24], investigated four texture extraction methods for the discrimination of gastric polyps in endoscopic videos, namely Color Wavelet Covariance (CWC), Texture Spectrum Histogram (TSH), Texture Spectrum and Color Histogram (TSCHS), and Local Binary Pattern (LBP). Results reported so far support the feasibility of using texture (and color information) feature analysis for detection of polyps.

A similar comparative study of Li et al. [30] was published in 2012 where three different color spaces, namely RBG, Lab, and HIS, were used to examine the performance. It claims that Lab color space coupled with CLBP (completed LBP) color features showed the best experimental results reaching a 77.20% detection rate. Both cases used the SVM as classifier.

However, texture-color based analysis has two major limitations [23]: it uses a fixed size analysis window; and relies heavily on an exhaustive training set of images, which make them very sensitive to parameters tuning.

Nawarathna et al. [37] made use of texton histograms for identifying abnormal regions with different classifiers such as SVM and KNN. An accuracy of 95.27% was obtained for polyp detection by using Schmid filter bank based textons and SVM classifier. Nawarathna et al. [36] later extended futher this approach by using an local binary patterns (LBP) feature. In addition, a bigger filter bank (Leung–Malik), which includes Gaussian filters, were proposed for capturing texture more effectively. These approaches only use texture features without any color or geometrical features. The best performance of 92% accuracy was obtained based on the Leung–Malik-LBP filter bank with KNN classifier.

Yuan and Meng [59] utilized scale invariant feature transform (SIFT) feature vectors with K-means clustering for bag of features representation of polyps. The authors calculated weighted histograms of the visual words by integrating histograms in both saliency and non-saliency regions. These were fed into an SVM classifier and experiments on 872 images with 436 polyp frames showed that 92% detection accuracy was obtained.

2.1.3 Shape features

The objective of shape-based methods is to localize those specific appearances that most polyps commonly have in endoscopy frames. One method, as Hwang et al. [23] suggested, is to utilize elliptical shape features to detect the shots of polyps, assuming that polyps tend to have an elliptical shape.

Hwang et al. further improved their method in [22] by involving a watershed segmentation base on Gabor texture features and K-means clustering, prior to identifying polyp candidates by extracting curvature-based geometric information from the resulting segments. But the author only tested their method using a small dataset of 128 images containing 64 polyp shots and 64 non-polyp images, which make the results (100% sensitivity and over 81% specificity) little convincing, since a small number of test dataset tend to lead to over tuning that may easily create an illusion of good performance.

A more sophisticated shape-based method, introduced by Bernal et al. [7], employed valley information and a region growing approach to find polyps. Bernal et al. called it Sector Accumulation – Depth of Valleys Accumulation (SA-DOVA). The method was further improved and renamed as Window Median DOVA (WM-DOVA). Further performance evaluations presented in [8] claim to have reduced the number of false positives around vascular structures and specular reflections. In principle, WM-DOVA not only exploited some available methods of low-level image processing such as valleys or edges analysis, but also first introduced a searching method for concavity of boundaries and elements of the scene such as blood vessels and specular highlights [8]. Nonetheless, shape-based approaches tend to mislead a polyp detector towards other polyp-like structures such as fecal content and reflection spots [51].

2.1.4 Texture and shape features

Both texture-based and shape-based methods have benefits and drawbacks. For that reason, more recent systems have considered combining them as an attempt to obtain improved performance. Mamonov et al. [33] presented an algorithm of polyp detection in colon capsule endoscopy, which is referred to as binary classification with pre-selection. This algorithm relies on geometrical analysis and the texture content of the frame. They assumed the polyps are characterized as protrusions that are mostly round in shape, and then considered a best fit ball radius as a decision parameter of the binary classifier. In addition, the author also introduced a pre-selection procedure used to discards the frames with too much or too little texture content, considering that the surface of polyps is often highly textured. Meanwhile, too much texture tends to imply the presence of bubbles or trash liquids. Therefore, it makes sense to discard the frames with both too little and too much texture information in them.

Although, the above algorithm demonstrated high per polyp sensitivity (81.2%) and high per patient specificity (92.2%) by a thorough statistical testing with a rich data set, there are still some drawbacks and areas of improvement as listed below:

- It did not detect the actual location of polyp in a colon. This problem is particularly exacerbated in capsule colonoscopy.
- Its effectiveness lies partially in the use of a pre-selection criterion; however, the pre-selection approach proposed was robust in some sense, but not sophisticated enough. It was less effective in filtering out frames with bubbles.
- It only utilized texture and geometry information, but the color content was discarded since the frame had to be converted to grayscale before processing.
- It just used a binary classifier; however, more advanced classification techniques such support vector machines (SVM) may improve its detection performance.

Another work by Tajbakhsh et al. [51] proposed a hybrid context-shape approach, which utilizes texture context information to remove non-polyp structures and shape information to reliably localize polyps. The proposed system consists of four stages as follows:

- Constructing Edge Maps for input images.
- Refining the edge map using context information.
- Localizing polyp candidates using shape information.
- Placing a band around each polyp candidate.

The suggested system had been tested using two public polyp database containing 300 unique polyps, and achieved a sensitivity of 88.0%. However, the author also pointed out that the suggested system might fail to detect the polyps with faint gradients around their boundaries, resulting in a polyp localization failure. In addition, unsuccessful edge classification could also lead to localization failures.

2.1.5 Classifiers

In most cases that we studied, SVM has been the widely used classifier in medical image processing. SVM determines some support vectors from the feature space which are helpful to determine the optimal hyperplane to separate a set of objects with maximum margin [12]. However, there are no single classification methods which outperforms all others on all data sets and there are also some other state of the art classifiers such as Random Forests (RF) [32], KNN and so on. We will evaluate all of them in this work.

2.2 Deep learning

More recently, Deep learning (DL) techniques have become state-of-the-art for many image and signal processing tasks. DL is a new branch of ML that is based on a set of algorithms to model high level abstractions in data by extracting multiple processing layers, which allows the systems to be able to learn complex mapping functions directly from input data $f : X \to Y$. DL is indeed moving ML closer to the one of its original goals: Artificial Intelligence (AI), which was acknowledged as one of the top 10 breakthroughs of 2013 [11].

There are various deep learning architectures have been extensively studied in recent years, which include deep belief network (DBN) [19], autoencoder [55], deep convolutional neural network (DCNN) [28], recurrent neural network (RNN), region-based convolutional neural network (R-CNN) [16], signal processing [21, 44] and so on. They have been successfully applied in various areas, such as natural language processing [3, 34, 56], computer vision [45, 50, 58], and so on. However, current trends in research have demonstrated that DCNNs are highly effective in automatically analyzing images, that is the reason they are nowadays the first choice in complex computer vision applications. We therefore choose to utilize DCNN techniques as well in our work.

2.2.1 Deep architectures

The main power of CNNs lies in its deep architectures [14, 47, 49], which allows for extracting a great number of features at multiple levels of abstraction. Nowadays, there are various state-of-the-art deep CNN models developed as presented as following.

AlexNet [28] developed by Krizhevsky, Sutskever, and Hinton in 2012 was the first time a model performed so well on ImageNet dataset, which achieved a top-5 error of 15.4% (Top-5 error is the rate at which, given an image, the model does not output the correct answer with its top-5 predictions). AlexNet was composed by 5 convolutional layers along with 3 fully connected layers, which illustrated the power and benefits of CNNs and backed them up with record breaking performance in the competition of 2012 ILSVRC(ImageNet Large-Scale Visual Recognition Challenge). And more, the techniques utilized by AlexNet such as data augmentation and dropout are still used today.

VGG Net [9] was proposed by the Oxford Visual Geometry Group (VGG) in ILSVRC 2014 best utilized with its 7.3% error rate. This model consists of five main groups of convolution operations. Adjacent convolution groups are connected via max-pooling layers. Each group contains a series of 3x3 convolutional layers (i.e. kernels). The number of convolution kernels stays the same within the group and increases from 64 in the first group to 512 in the last one. The total number of learnable layers could be 11, 13, 16, or 19 depending on the number of convolutional layers in each group. Figure 2.1 illustrate the architecture of 16-layer VGG net (VGG16). VGG Net is one of the most influential architectures since it strengthened the intuitive notion that CNNs have to have deep layers for making this hierarchical representation of visual data to work.

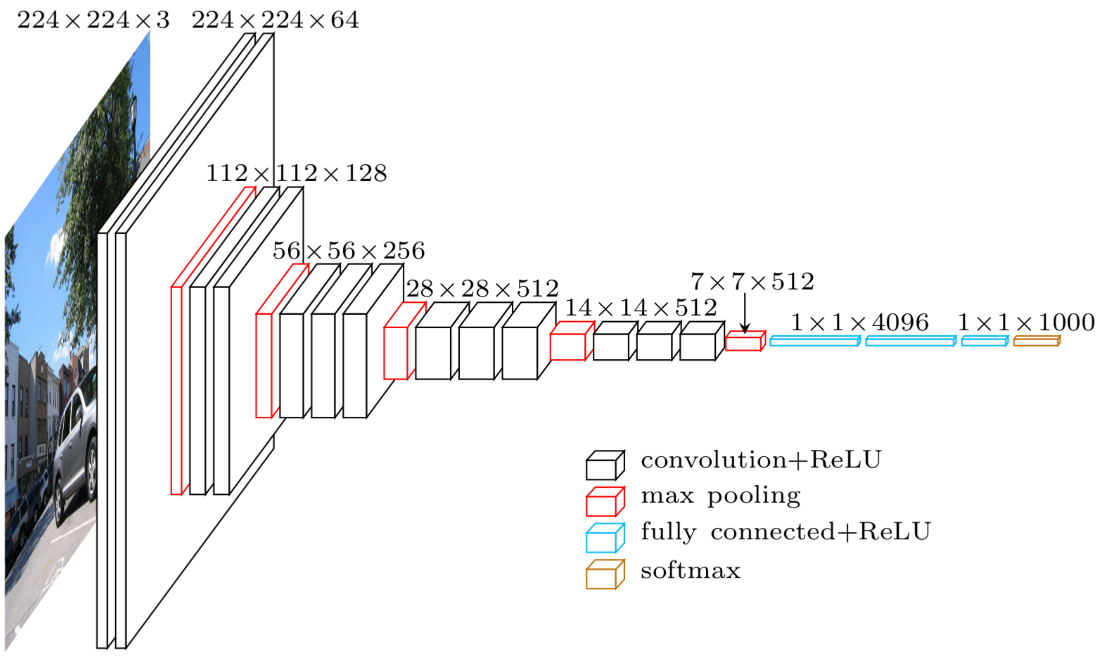


FIGURE 2.1: The architecture of VGG16 model [9].

GoogLeNet [49] was the winner of ILSVRC 2014 with a top-5 error of 6.7%. The authors introduced an novel Inception module which performs pooling and convolutional operations in parallel. GoogLeNet used 9 inception modules with over 100 layers in total but had 12x fewer parameters than AlexNet. It was the first model that introduced the idea that CNN layers with different kernel filters can be stacked up and operating in parallel. Utilizing the creative inception module, GoogLeNet can lead to improved performance and computationally efficiency, since it avoided stacking all convolution layers and adding huge numbers of filters sequentially which require a greater number of computational and memory resources and increase the chance of over-fitting issue as well.

ResNet were originally introduced in the paper "Deep Residual learning for Image Recognition" [18] by He et.al. It won the championship of ILSVRC 2015 with a new 152-layer convolutional network architecture (ResNet152) trained on an 8 GPU machine for two to three weeks. It achieved an incredible top-5 error of 3.6% that set new records in classification, detection, and localization. Resnets architectures were demonstrated with 50, 101 and 152 layers. The deeper ResNets got, the more its performance grew.

The authors of ResNet proposed a residual learning approach to ease the difficulty of training deeper networks by reformulating the layers as residual blocks, with each block containing two branches, one directly connecting input to the output, the other

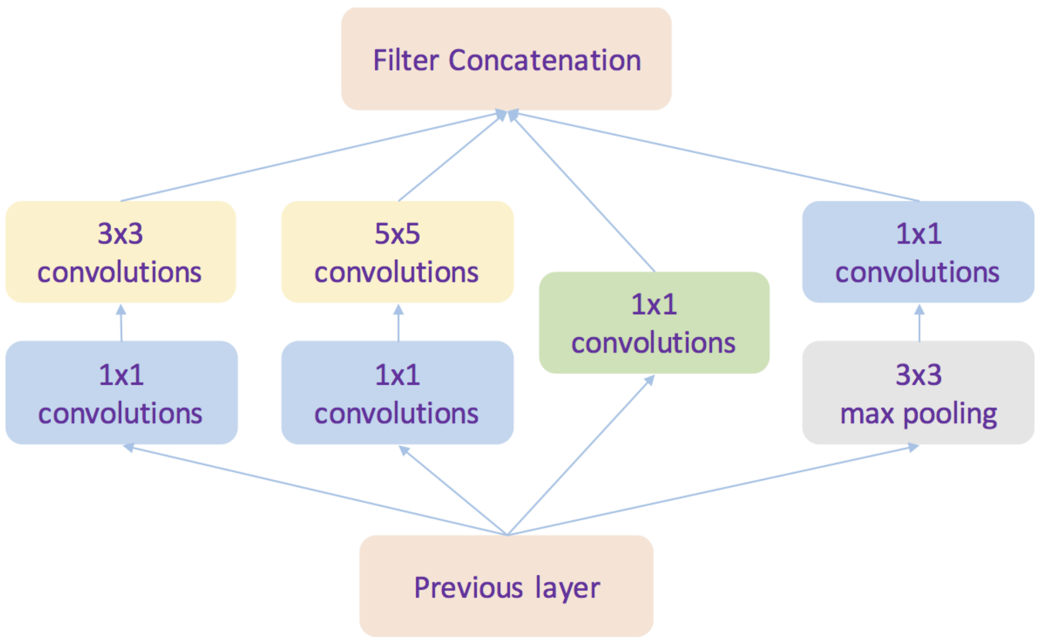


FIGURE 2.2: The structure of an Inception module of GoogLeNet.

performing two to three convolutions and calculating the residual function with reference to the layer inputs. The outputs of these two branches are then added up as shown in Figure 2.3.

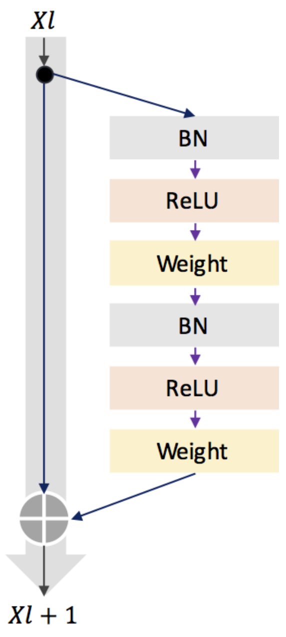


FIGURE 2.3: The residual block for residual learning approach.

2.2.2 CNNs-based CAD systems

With the revival of CNNs techniques, the medical image processing field has also been experiencing a new generation of CAD systems with more promising performance. Wimmer et al. applied CNNs for the computer assisted diagnosis of celiac disease based on endoscopic images of the duodenum in [57]. To evaluate which network configurations are best suited for the classification of celiac disease, the author trained several different CNN models with different numbers of layers and filters and different filter dimensions. The results of the CNNs are compared with the results of popular general purpose image representations methods. The results show that the deeper CNN architectures outperform these comparison approaches and that combining CNNs with linear support vector machines furtherly improves the classification rates for about 3–7% leading to distinctly better results (up to 97%) than those of the comparison methods.

Jia et al. employed Deep CNNs for detection of bleeding in GI 10,000 Wireless Capsule Endoscopy (WCE) images [25]. The WCE is a non-invasive image video method for examination small bowel disease. They claimed F-measure approximately to 99%. Pei etal. mainly focused on evaluation of contraction frequency of bowel by investigation diameter patterns and length of bowel by measuring temporal information [38].

A popular approach of automatic feature extraction from endoscopy images adopted using CNN [61]. Then the features vector to the SVM for classification and detection of gastrointestinal lesions. The proposed system realized on 180 images for lesions detection and 80% accuracy reported. Similarly hybrid approach used by [15]. Fast features extraction using CNN architectures and then the extracted features passed to SVM for detection of inflammatory GI disease in WCE videos. The experiments conducted on 337 annotated inflammatory images and 599 non-inflammatory images of the GI tract. Training set containing 200 normal and 200 abnormal while the test set containing 27 normal and 27 abnormal and obtained an overall accuracy upto 90%.

There are several recent works [41, 52, 53] that have exploited CNNs-based methods for automatic detection of polyps in endoscopy and colonoscopy images. Though DL approaches have the property of extracting a set of discriminating features at multiple levels of abstraction by exploiting the input image pixel directly, it usually requires a large amount of training dataset that might be quite rare in some medical imaging fields. Ribeiro et al. [40] proposed a method allowing the use of small patches to increase the size of the database and classify different regions in the same image and then train the CNNs.

In yet another work, Tajbakhsh et al. proposed a new polyp detection method based on the unique 3-way image presentation and CNNs in [52]. The 3-way image represents the three major types of polyp features, namely (1) color and texture clues, (2) temporal features, and (3) shape in context. This method fully utilizes a variety of polyp features such as color, texture, shape, and temporal information in multiple scales, which enable more accurate polyp detection in [52].

To train the CNNs, the author first collected all the generated polyp candidates and grouped them into true and false detections, then collected the three sets of patches Pc, Pt, and Ps at multiple scales, translations, and orientations, and finally, total of 400,000 patches were labeled as positive or negative and resized to 32x32 pixels for the entire training dataset. The evaluations based on a large annotated polyp database showed a superior performance and significantly reducing polyp detection latency and the number of false positives [52]. There was one drawback that this method was not reliant on the future frames and avoiding the delayed feedback on the locations of polyps.

2.2.3 Pre-trained CNNs

The above methods need to train CNNs from scratch with a large amount of training database that might be quite rare in medical fields. The updated work of Tajbakhsh et al. [53] tried to address the problem by making use of pretrained CNNs, with sufficient fine-tuning, to eliminate the need for training CNNs from scratch.

The author considered four distinct medical imaging applications (polyp detection, pulmonary embolism detection, colonoscopy frame classification and intima-media boundary segmentation) in three specialties (radiology, cardiology, and gastroenterology) involving classification, detection, and segmentation, and investigated how the performance of deep CNNs trained from scratch compared with the pre-trained CNNs fine-tuned in a layer-wise manner. Their experiments demonstrated that [53]:

- Use of a pre-trained CNN with adequate fine-tuning outperformed or, in the worst case, performed as well as a CNN trained from scratch.
- Fine-tuned CNNs were more robust to the size of training sets than CNNs trained from scratch
- Neither shallow tuning nor deep tuning was the optimal choice for a particular application.
- Layer-wise fine-tuning scheme could offer a practical way to reach the best performance for the application at hand based on the amount of available data.

These results showed the knowledge transfer from natural images to medical images is possible and suggested [53] that the layer-wise fine-tuning might offer a practical way to achieve the best performance for some medical image application based on the amount of available data.

2.3 Summary

In summary, we discussed all of the polyp detection approaches covered so far with machine learning and deep learning techniques, classifiers utilized along with the dataset as well as performance details (whenever available). We can see that plenty of improvements was done either in the pre-processing techniques, feature extraction algorithms, classification methods or in all, and there is a clear trend toward the use of deep learning frameworks, especially CNN-based architectures. However, it can also be seen that these proposed methods are tuned to obtain the best achievable detection accuracy results for their corresponding datasets, so our belief is that the majority of these methods have more or less over-fitting or under-fitting problems.

Chapter 3

Methodology

In this chapter, we describe different techniques in detail for automatic polyp detection. The first section of this chapter presents our 3 major frameworks (ML-framework, DL-framework and TL-framework) for automatic detecting polyps in colonic images, and we also describe a scalable framework for computer-aided diagnosis systems based on the fusion of overall state-of-the-art techniques to generalize and extend our project in future with versatile capabilities in medical domain.

The subsequent section analyses various image preprocessing methods that are utilized in our work and also are necessary for most machine learning and deep learning systems. These techniques cover histogram modification, noise filtering, data augmentation, and dimension reduction. Next, the chapter focuses on neural networks design methodologies that mainly cover all the necessary algorithms to build a effective artificial neural network such as feed-forward structure, activation functions, softmax functions, loss functions, regularization, gradient descent optimizers and backpropagation methods.

Finally, in the last section, we describe all necessary methodologies for designing deep convolutional networks that represent state-of-the-art now, which include the convolution algorithm with zero-padding and stride methods, pooling and dropout techniques. At last, we describe the deep learning model - 50-layer ResNets with its detail structure. ResNet50 is the major deep convolutional network architecture utilized in our project.

3.1 Proposed frameworks

In this work, we propose and test 3 different methodologies for automatic detection of colorectal polyps as shown in Figure 3.1. The first detection scheme named **ML-framework** stands for the traditional machine learning classification methods based on a set of low-level feature descriptors. The second one called **DL-framework** is to make use of deep learning methods (mostly CNNs-based architectures) for image classification. The last scheme called **TL-framework** presents transfer learning (TL) strategies utilized for automatic polyp detection. We will discuss them in detail in later sections.

In addition, based on above three proposed detection methods, we layout a generalized but scalable framework for computer-aided diagnosis (CAD) systems [31] in which fusion of machine learning algorithms and deep learning techniques are employed to further generalize and boost system's performance and robustness. This scheme is flexible and easy to add new types of data in future as needed in order to detect or predict other types of diseases. Generally, it consists of four stages: preprocessing, feature extraction, classification and post-processing as shown in Figure 3.2. Here red dash line represents the process for training the system.

First, the preprocessing stage is quite import to properly prepare the data by removing noise or unwanted parts of the data. The objective of preprocessing is to refine the quality of digital images. It can consist of subsampling, enhancing, edge detecting, scaling or extracting research of interest (ROI) patches, and so on. It has a lot of impact on the following feature extraction and classification processing.

For the feature extraction phase, the focus is on the extraction of some key characteristics of candidates such as texture and shape by a set of low-level image processing algorithms. However, more and more DL techniques like CNNs have been recently utilized as feature descriptor in medical image analysis. We also took advantage of deep CNNs techniques in our work.

In classification stage, many kinds of classifiers are utilized to discriminating multiple objects on the base of features defined and extracted from previous phase. Finally, the post-processing stage is needed to properly display the results, formulate diagnosis reports, or localize and annotate the diseases for further evaluations by medical physicians.

The purpose of this suggested CAD architecture is to be as a roadmap for making versatile CAD systems in future by reproducing, generalizing, and extending our work on automatic polyp detection systems.

3.2 Image preprocessing

Image preprocessing here refers to processing of digital images by low-level algorithms, i.e removing the noise in an image using a digital computer. Preprocessing is a common and necessary step in machine learning pipeline. For mathematical analysis, an image is defined as 2-dimension function $f(x,y)$, where x and y are spatial coordinates, and the amplitude of f is called the intensity or gray level of the image at the point of coordinates (x,y). A digital image is composed of a finite number of elements or pixels described by $x, y,$ and f. Pixel is the basic cell and the most widely used term to denote the elements of a digital image.

Various algorithms and methodologies have been developed in image processing during the past decades such as contrast and edge enhancement. In our work, we evaluated some important algorithms to preprocess our images, including histogram modification, contrast stretching, noise filtering, PCA etc.

3.2.1 Histogram modification

Histogram has a lot of importance in image enhancement. It reflects the characteristics of image. By modifying the histogram, image characteristics can be modified. One such example is Histogram Equalization. Histogram equalization is a nonlinear stretch that redistributes pixel values so that there is approximately the same number of pixels with each value within a range.

Meanwhile, the contrast stretching methods are designed exclusively for frequently encountered situations, since some images are homogeneous i.e., they do not have

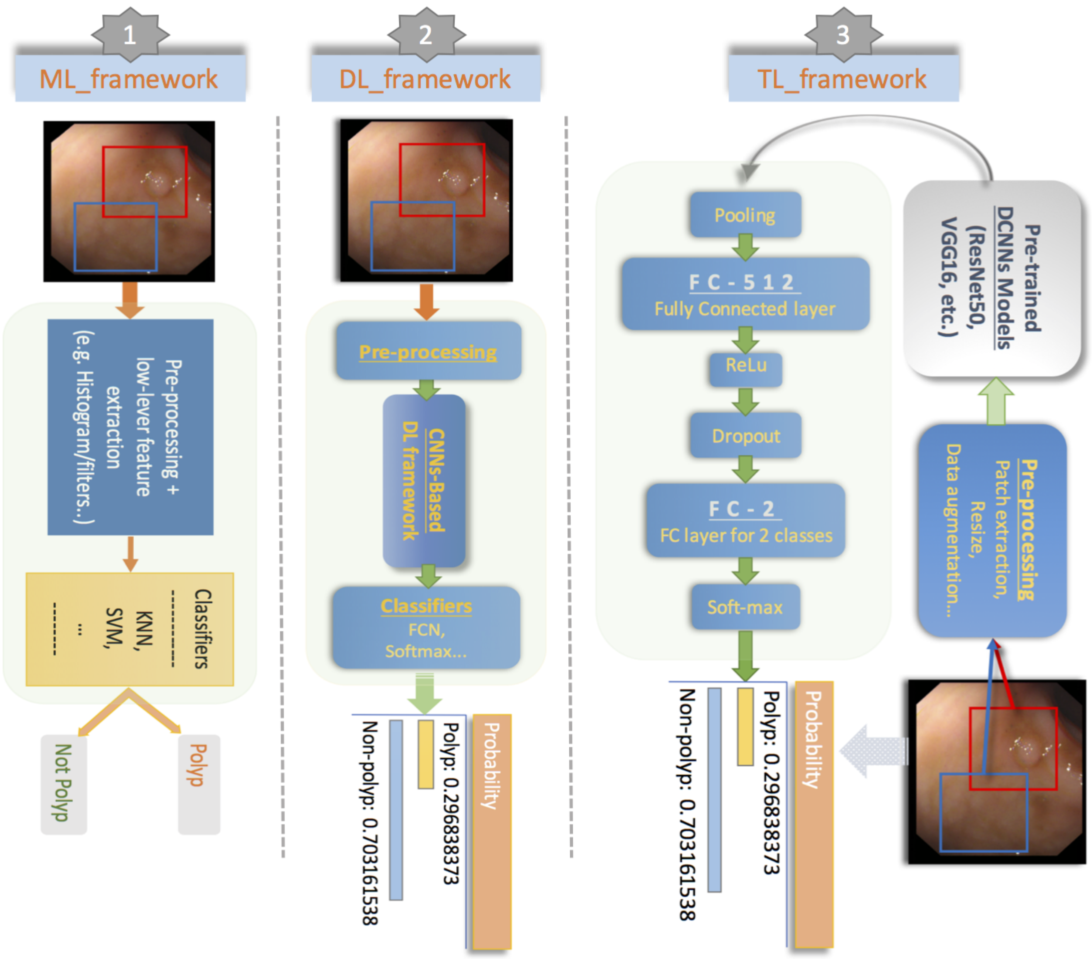


FIGURE 3.1: Three frameworks for automatic polyp detection. ML-framework stands for traditional machine learning methods; DL-framework presents deep learning (mostly CNNs-based architectures) classification techniques, and TL-framework is the scheme of utilizing transfer learning strategies on pre-trained deep neural networks such as pre-trained ResNet50, VGG16 etc.

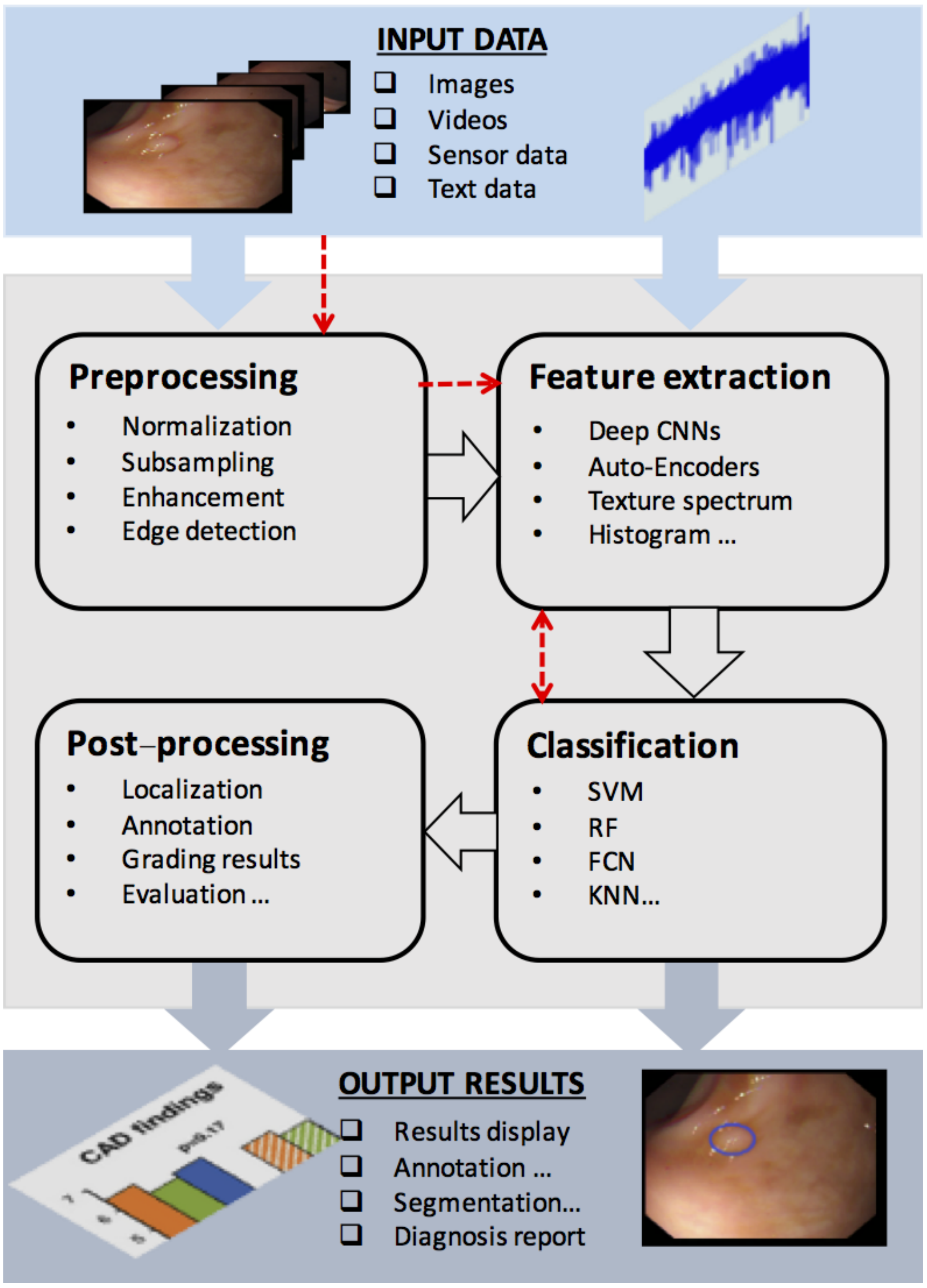


FIGURE 3.2: A generalized framework for computer-aided diagnosis systems which can be extended to detect or predict other types of diseases in future. The framework has further evolved from our previous work [31].

much change in their levels. They are characterized as the occurrence of very narrow peaks. Different stretching techniques have been developed to stretch the narrow range to the whole of the available dynamic range as well. The figure 3.3 shows the different histogram performance based on three algorithms: contrast stretching, histogram equalization and adaptive equalization.

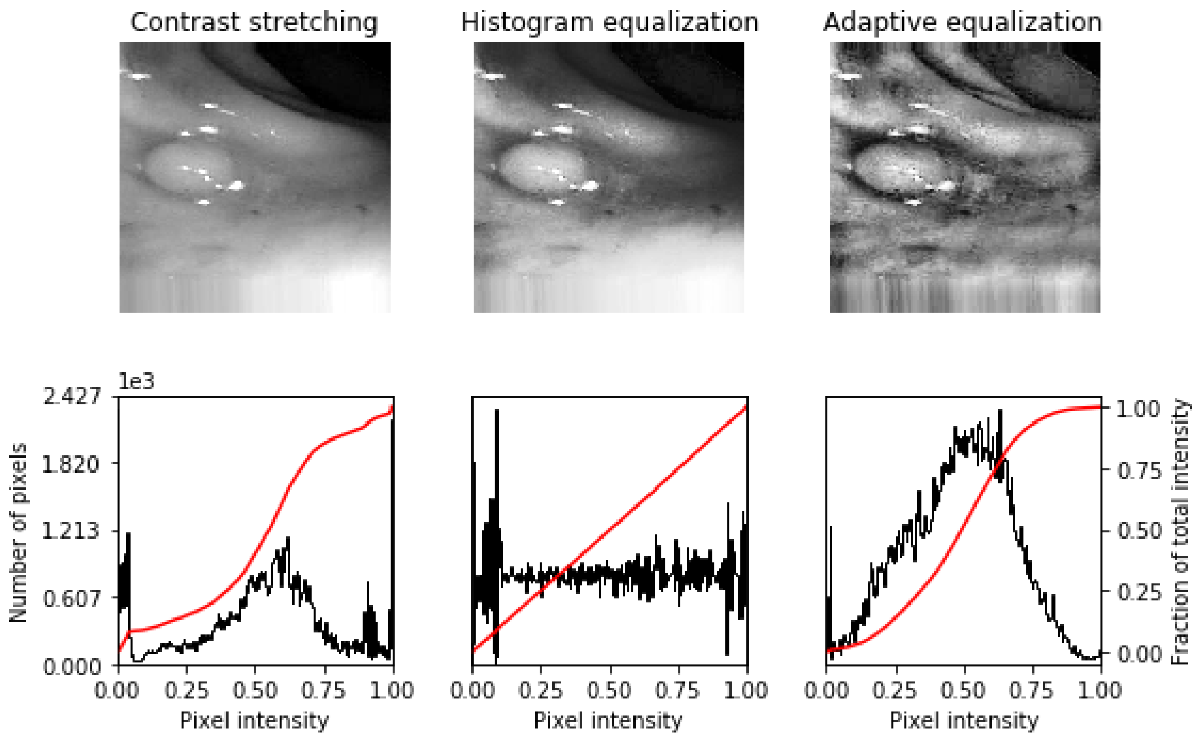


FIGURE 3.3: Performance evaluation of different histogram algorithms.

3.2.2 Noise filtering

Noise Filtering is used to filter the unnecessary information from an image. It is also used to remove various types of noises from the images. Various filters like mean, median, max, min, sobel, prewitt, canny, laplace etc., are available for our project. We evaluate most of them with our images. The Figure 3.4 visualizes the performances of a number of different filtering algorithms.

3.2.3 Data augmentation

Data augmentation is one way to fight over-fitting for small training dataset. Overfitting happens when a model exposed to too few examples learns patterns that do not generalize to new data, i.e. when the model starts using irrelevant features for making predictions. In our work, we make use of data augmentation methods to enlarge our training dataset. For example, assume a training set of 100 images of polyps and non-polyps. By rotating, mirroring (horizontal and vertical flip), shift (width and height), zoom, and adjusting contrast, etc. It is possible to generate additional over than 2000 images. The figure 3.5 shows an example of data augmentation form one polyp image to 10 additional images by random rotation, horizontal and vertical flip, shift and zoom

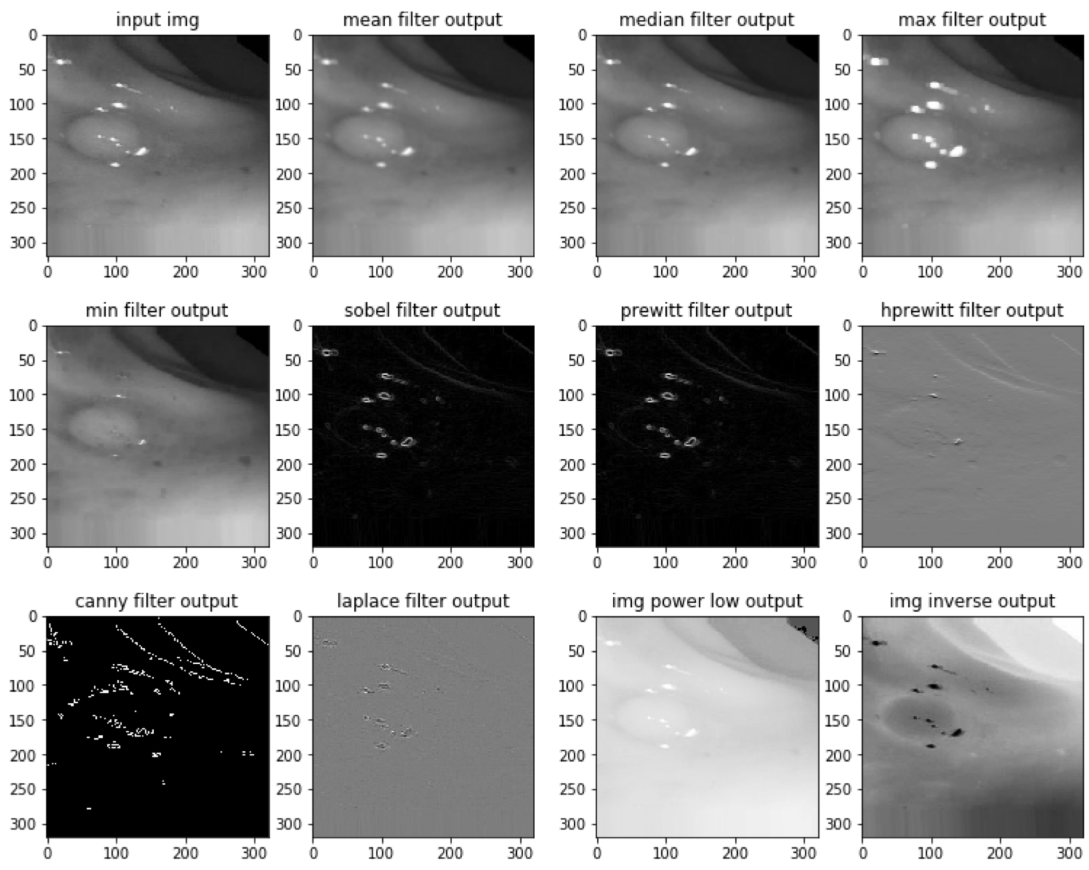


FIGURE 3.4: Performance evaluations of different filters.

methods. In many machine learning applications, data augmentation approach allow to build better models.

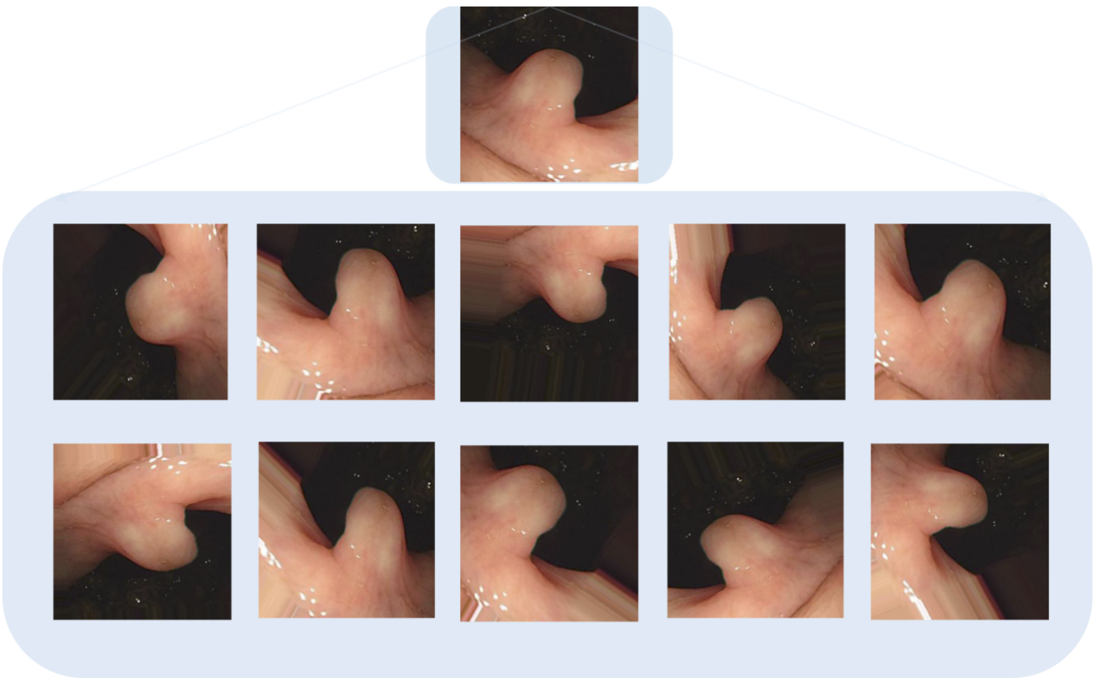


FIGURE 3.5: Examples of data augmentation results.

3.2.4 Dimension reduction

Dimensionality reduction is an useful way to reduce the running time of machine learning algorithms, since the number of input features has an effect upon runtime. Principal component analysis (PCA) is an algorithm which can be used for dimensionality reduction. Basically, PCA can be represented in major 4 steps.

- Normalize the data to have features on the same scale.
- Calculate the covariance matrix to measure of how two different variables change together.
- Find the eigenvectors of the covariance matrix.
- Translate the data to be in terms of the components.

The covariance between X and Y, can be given by the following formula 3.1, and then covariance matrix can be computed with this form 3.2:

$$cov = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)} \tag{3.1}$$

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix} \tag{3.2}$$

3.3 Neural network design

Neural Networks are a group of models based on biological neural networks. Figure(3.6) shows how a general neuron looks. Where **W** is a matrix and **X** is an input column vector containing all pixel data of an image. For instance, **X** can be a [32*32*3 x 1] column vector, and **W** is a [2 x 32*32*3] matrix, and the output is a vector **Y** of N class scores (class-1, to class-N). That is $Y(x_i, W, b) = \sigma(Wx_i + b)$. The weights **W** are learn-able and control the strength of influence of each input. Where activation function is usually an abstraction representing the firing rate in the cell.

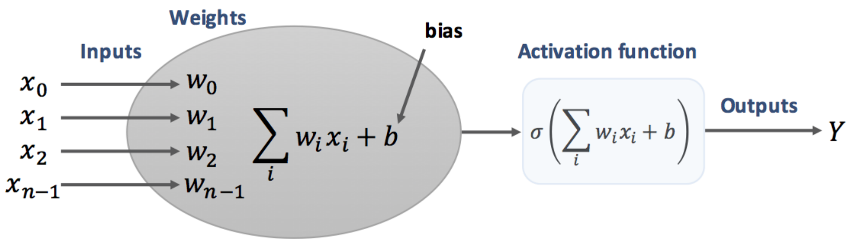


FIGURE 3.6: Mathematical model of a neuron.

3.3.1 Neural networks

Neural networks take inputs and transform them by a series of hidden layers. Figure 3.7 shows a 3-layer feed-forward neural network with 2 hidden layers. Each hidden layer consists 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.

A feed-forward neural network takes in an input, then that input "trickles" through the network and the neural network returns an output vector. More formally, call a_j^i the activation output of the j^{th} neuron in the i^{th} layer, where a_j^i is the j^{th} element in the input vector.

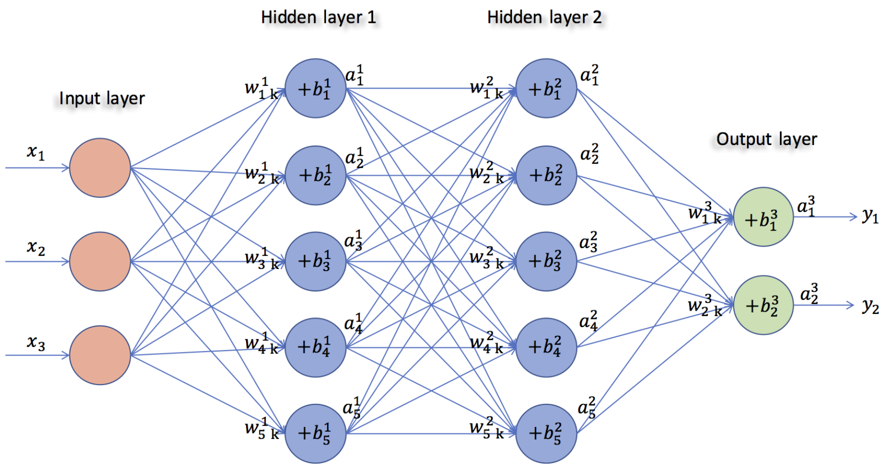

Hidden layer 1
Hidden layer 2
Input layer
Output layer
x_1
x_2
x_3
$w^1_{1\,k}$ $+b^1_1$ a^1_1
$w^1_{2\,k}$ $+b^1_2$ a^1_2
$w^1_{3\,k}$ $+b^1_3$ a^1_3
$w^1_{4\,k}$ $+b^1_4$ a^1_4
$w^1_{5\,k}$ $+b^1_5$ a^1_5
$w^2_{1\,k}$ $+b^2_1$ a^2_1
$w^2_{2\,k}$ $+b^2_2$ a^2_2
$w^2_{3\,k}$ $+b^2_3$ a^2_3
$w^2_{4\,k}$ $+b^2_4$ a^2_4
$w^2_{5\,k}$ $+b^2_5$ a^2_5
$w^3_{1\,k}$ $+b^3_1$ a^3_1 y_1
$w^3_{2\,k}$ $+b^3_2$ a^3_2 y_2

FIGURE 3.7: 3-layer feed-forward neural network model.

Then we can relate the next layer's input to it's previous via the following relation:

$$a_j^i = \sigma\left(\sum_k w_{jk}^i a_k^{i-1} + b_j^i\right) \tag{3.3}$$

Where in equation (3.3)

- σ is the activation function;
- w_{jk}^i is the weight from the k^{th} neuron in the $(i-1)^{th}$ layer to the j^{th} neuron in the i^{th} layer;
- b_j^i is the bias of the j^{th} neuron in the i^{th} layer;
- a_j^i represents the activation value of the j^{th} neuron in the i^{th} layer.

Sometimes we write z_j^i to represent $\sum_k (w_{jk}^i \cdot a_k^{i-1}) + b_j^i$, in other words, the activation value of a neuron before applying the activation function.

$$z_j^i = \sum_k w_{jk}^i a_k^{i-1} + b_j^i \tag{3.4}$$

3.3.2 Activation functions

The most important unit in neural network structure is a scalar-to-scalar function called "the activation function or threshold function or transfer function", output a result value called the "unit's activation". An activation function for limiting the amplitude of the output of a neuron. The goal of an activation function is to transform its input to an output that makes binary decisions more separable. The widely-used activation functions are sigmoid, tanh, and the rectified linear unit (ReLU), since they avoid saturation issues and make learning faster than other functions.

Sigmoid (non-linear) functions have the mathematical form as below. They are often used for mathematical convenience because their derivatives are very easy to calculate, which we will use to calculate the weight updates in training algorithms.

$$a_j^i = \sigma(z_j^i) = \frac{1}{1+e^{-z_j^i}} \tag{3.5}$$

The Tanh functions with the mathematical form as below are related linearly and can be seen as a rescaled version of the sigmoid function so that its output range is between -1 to 1.

$$a_j^i = \sigma(z_j^i) = \tanh(z_j^i) \tag{3.6}$$

The ReLu functions are the most popular choice for deeper architectures. It can be seen as a ramp function whose range lies above 0 to infinity, so that it is much easier to calculate than the sigmoid function. The biggest benefit of ReLU is that it bypasses the vanishing gradient problem.

$$a_j^i = \sigma(z_j^i) = \max(0, z_j^i) \tag{3.7}$$

3.3.3 Softmax functions

In our work, we make use of Softmax functions as the output of a classifier which represent the probability distribution over C classes, in our case, $C = 2$ since we have only 2 classes: polyp and non-polyp. This function is a normalized exponential and is defined as:

$$y_c = \varrho(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{d=1}^{C} e^{z_d}} \quad \text{for } c = 1 \cdots C \tag{3.8}$$

Here the softmax function ϱ takes as input a C-dimensional vector $\mathbf{z}$ and outputs a C-dimensional vector $\mathbf{y}$ of real values between 0 and 1. The denominator $\sum_{d=1}^{C} e^{z_d}$ acts as a regularizer to make sure that $\sum_{c=1}^{C} y_c = 1$.

Loss functions

A loss function, or a cost function is used for parameter estimation in training neural networks. The choice of the loss function is an important aspect for designing a deep neural network. In our project, we make use of cross-entropy loss function which is defined as:

$$\mathcal{L}(X, Y) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \ln a(x^{(i)}) + \left(1 - y^{(i)}\right) \ln\left(1 - a(x^{(i)})\right) \tag{3.9}$$

Here $X = \{x^{(1)}, \ldots, x^{(n)}\}$ is the set of input examples in the training dataset, and $Y = \{y^{(1)}, \ldots, y^{(n)}\}$ is the corresponding set of labels for those input examples. The $a(x)$ is the output of the neural network given input x, which is typically restricted to the open interval (0, 1) by using a ReLU 3.7 or sigmoid 3.5 activation function.

Regularization

Regularization is a very important technique in neural network design to prevent overfitting. Regularization works by extending the loss function with a regularization penalty ($R(W)$) as:

$$L = \underbrace{\mathcal{L}(X, Y)}_{\text{loss function}} + \underbrace{\lambda R(W)}_{\text{regularization penalty}} \tag{3.10}$$

Then the loss function can be weighted by a hyper-parameter λ in order to prevent the coefficients. The most common regularization penalty is the $L2$ norm that is utilized in our design. It is defined as:

$$R(W) = \sum_k \sum_l W_{k,l}^2 \tag{3.11}$$

3.3.4 Gradient descent optimizers

Gradient descent is one of the most popular algorithms to optimize neural networks. There are five popular optimization techniques: Stochastic gradient descent (SGD), SGD+momentum, Adagrad [13], Adadelta [60] and Adam [27] – methods for finding local optimum (global when dealing with convex problem) of certain differentiable functions. The gradient descent algorithm is used in every layer to update the weights in the direction of the negative gradient by backpropgation learning algorithm.

In our work, we choose Adadelta as the optimizer of the model, since in practical, Adadelta seems to be "safer" because it doesn't depend so strongly on setting of learning rates, and base on our own experiments as well, it alway gave us the quickest convergence and performed better than AdaGrad or SGD and Momentum with decaying learning rate. The full algorithm of Adadelta is as shown in Figure 3.8.

```
Require: Decay rate $\rho$, Constant $\epsilon$
Require: Initial parameter $x_1$
1: Initialize accumulation variables $E[g^2]_0 = 0$, $E[\Delta x^2]_0 = 0$
2: for $t = 1 : T$ do %% Loop over # of updates
3:     Compute Gradient: $g_t$
4:     Accumulate Gradient: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t^2$
5:     Compute Update: $\Delta x_t = -\frac{\text{RMS}[\Delta x]_{t-1}}{\text{RMS}[g]_t} g_t$
6:     Accumulate Updates: $E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho) \Delta x_t^2$
7:     Apply Update: $x_{t+1} = x_t + \Delta x_t$
8: end for
```

FIGURE 3.8: Algorithm of computing Adadelta [60].

Although adadelta algorithm strive to do away with learning rate tuning, in practice the issue isn't completely solved. Setting and tuning constant ϵ and decay rate

ρ are still important and necessary in our work to achieve sound performance curve while the adaptation can effectively counter the learning rate with its own scaling if the optimization directs it in that direction. The constant ϵ can be consider as the 'learning rate' of adadelta because it actually determines the update of Δx_t since $RMS[\Delta x]_t = \sqrt{E[\Delta x^2]_t + \epsilon}$ and $E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$, where RMS stands for root mean squared.

Backpropagation

Technically, the backpropagation algorithm is a supervised learning method for training the weights in multilayer feed-forward neural networks. The algorithm can be divided into two phases: propagation and weight update.

The propagation covers 2 steps: first forward propagation of a training input through the neural network and then backward propagation of the generated deltas (the error between the targeted and actual output value). While the weight update must follow 2 steps as well: first, the weight's delta and input activation are multiplied to determine the gradient of the weight, and a ratio of that gradient is then subtracted from the weight.

3.4 Convolutional networks

Convolutional networks (ConvNets) [29], also known as convolutional neural networks (CNNs), are a specialized kind of neural networks for processing data that has a known, grid-like topology [17]. In principle, though CNNs/ConvNets are very similar to regular neural networks which consist of neurons with learnable weights and biases, ConvNets architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network than regular neural nets do, because regular neural nets don't scale well to full images [26]. For instance, if taking an images of size 200x200x3 (200 wide, 200 high, 3 color channels) as inputs, so a single fully-connected neuron in a first hidden layer would have 200*200*3 = 120,000 weights. Moreover, we would almost have to have many such neurons, so the parameters would grow up quickly which would cause over-fitting issues.

3.4.1 Convolutional layer

The Convolutional layer is the core building block of a convolutional network which takes the convolution operation of the input image with convolution matrices (also known as kernel filters) to generate the output feature maps. Figure (3.9) is an example of a convolution operation with 2-kernel filters (5x5x3x2) on a RGB image of size 28x28x3. The output feature maps can be produced by the form below:

$$(k \star im)(x,y) = \sum_{c=0}^{2}\sum_{n=0}^{4}\sum_{m=0}^{4} k(n,m,c).im(x+n-2, y+m-2, c) \tag{3.12}$$

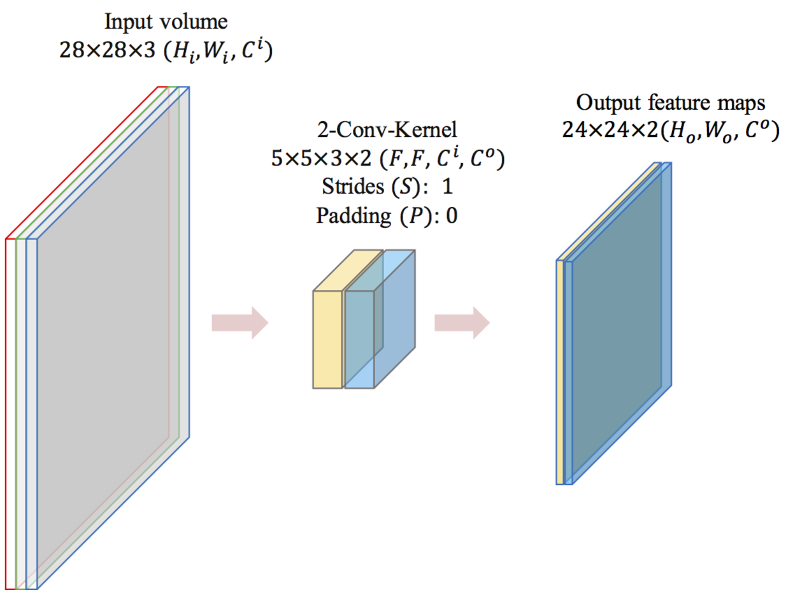

Input volume
28×28×3 (H_i,W_i, C^i)
2-Conv-Kernel
5×5×3×2 (F,F, C^i, C^o)
Strides (S): 1
Padding (P): 0
Output feature maps
24×24×2(H_o,W_o, C^o)

FIGURE 3.9: Convolutional operation examples on a 3-channel image.

Here the input volume size is represented as $H_i \times W_i \times C^i$, and the kernel filter setting is $F \times F \times C^i \times K$ where F stands for the size of the kernel, C^i is the channels of the kernel (must be equal to the channels of input) and K stands for the number of kernel filters, if given a stride of S and a zero-padding of P, the volume of output maps ($H_o \times W_o \times C^o$) can be produced by the forms below:

- $H_o = (H_i - F + 2P)/S + 1$ (the output dimension of hight)
- $W_o = (W_i - F + 2P)/S + 1$ (the output dimension of width)
- $C^o = K$ (the output channels or depths)

Stride and padding

As we can see, there are 2 key hyper parameters control the size of the output volume: stride and padding. Stride controls how the filter convolves around the input volume. When the stride is 1 then the filters would move one pixel at a time. When the stride is 2 then the filters would jump 2 pixels at a time as we slide them around. This will produce smaller output maps spatially. However, sometimes it would be necessary to pad the input volume with zeros around the border. The feature of zero-padding allows us to control the spatial size of the output feature maps. Figure 3.10 illustrates an example of setting for zero-padding and stride to produce the output spatial size as the same with the input volume.

3.4.2 Pooling layer

Pooling layer (also known as subsampling layer) is a popular approach to mainly down-samples the input volume spatially, and hence to reduce the amount of parameters and computation in the network which also give the network more invariance and robustness to control overfitting.

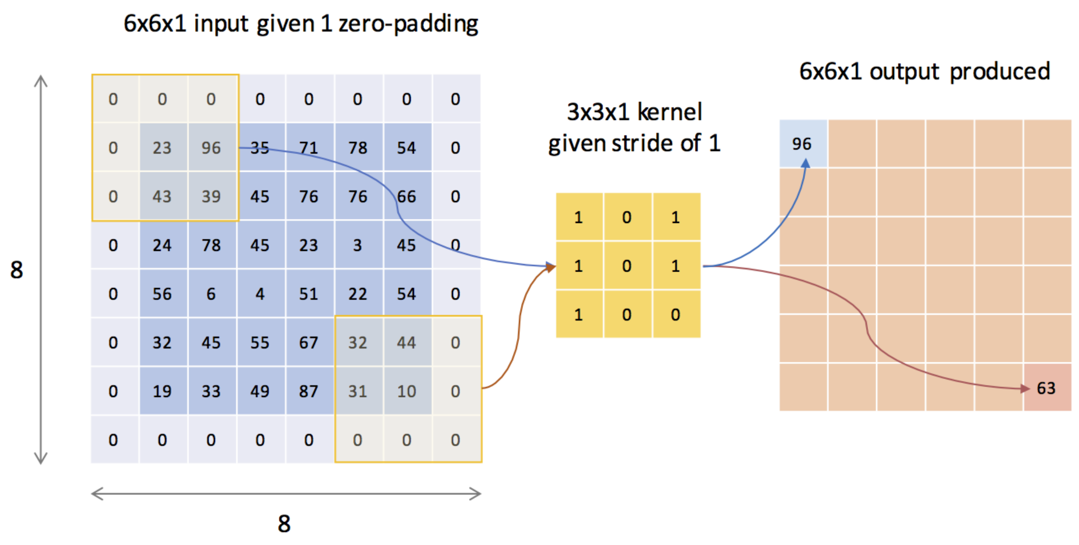


FIGURE 3.10: An example of setting zero-padding and strides.

In practice, pooling layers are commonly stacked in-between successive convlutional layers in a ConvNets model. The most used method for pooling layer in image processing tasks is max pooling. Max pooling decreases the dimension of input volume simply by taking only the maximum value from a fixed region while average pooling taking the average of each groups as shown in Figure 3.11.

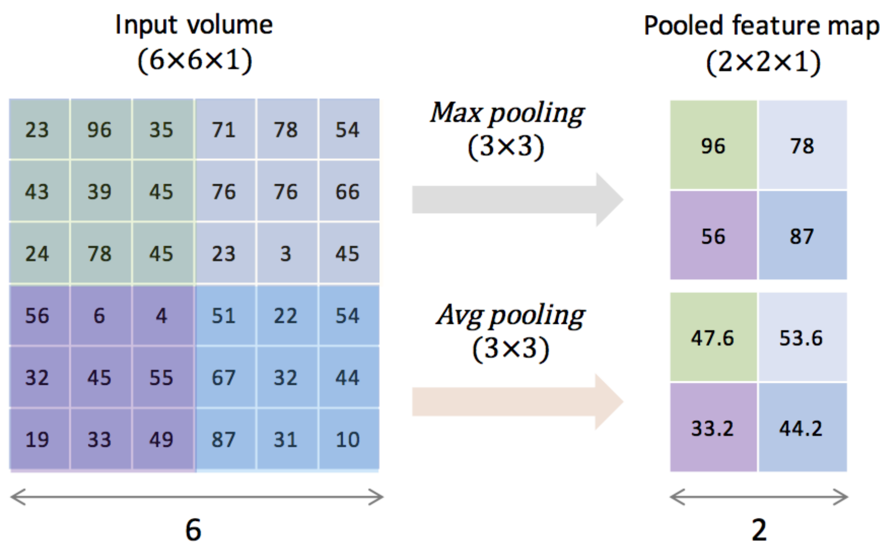


FIGURE 3.11: Max and average pooling examples for subsampling features.

In addition to max pooling method, average pooling or even L2-norm pooling was often used historically. However, it has recently fallen out of favor compared to the max pooling, which has been shown to work better in practice [26]. But we still made use of both max pooling and average pooling methods in our neural networks, and it demonstrated that average pooling performed better in some situation than max pooling.

3.4.3 Dropout layer

In our work, we introduced dropout layers to avoid over-fitting problems. The idea of dropout is simplistic in nature. This layer "drops out" a random set of activations by setting them to zero in that layer that would force the network to learn the multiple characteristics of input example to be redundant and robust, so that the network could be able to provide the right output even if some of the activations are dropped out. Figure 3.12 illustrates an example of applying dropout methods on a neural network.

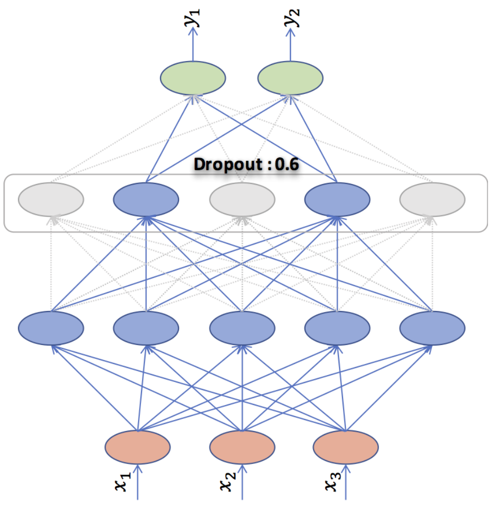


FIGURE 3.12: An example of applying dropout to a neural network.

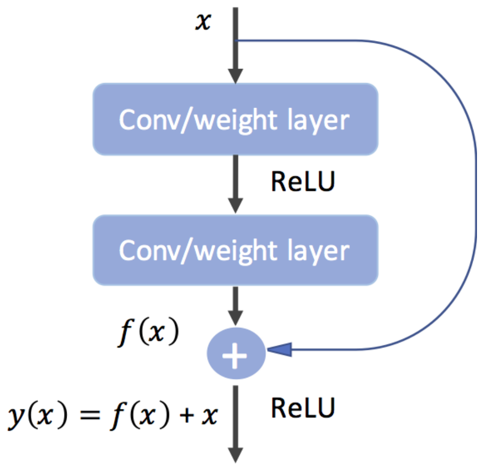


FIGURE 3.13: The structure of a residual block for ResNet.

3.4.4 ResNet architecture

In our work, we propose to utilize 50-layer ResNets as our deep learning model. ResNets reformulate the layers as residual blocks. The idea behind residual blocks is that the input x goes through some convolution layers, and you will get the result $f(x)$. That result is then added to the original input x. Let's call that $y(x) = f(x) + x$. In traditional CNNs, your $y(x)$ would just be equal to $f(x)$, so instead of just computing that transformation from x directly to $f(x)$, in ResNet we're computing the term of y that add $f(x)$ to the identity x as shown in Figure 3.13.

The residual network design addresses the problem of vanishing gradients in the simplest way possible, since the main challenge in training deeper networks is that accuracy degrades with network depth. The concept of residual learning behind is a great innovation and becoming one of the hot new ways to build deep convolutional neural networks. Safe to say, the ResNet model is now the best single CNN architecture for object detection, which is the main reason we choose this model for our work. Figure 3.14 illustrates ResNets with 50 layers. ResNets use bottleneck blocks of different numbers of repetitions which converges very fast and can be trained with hundreds or thousands of layers.

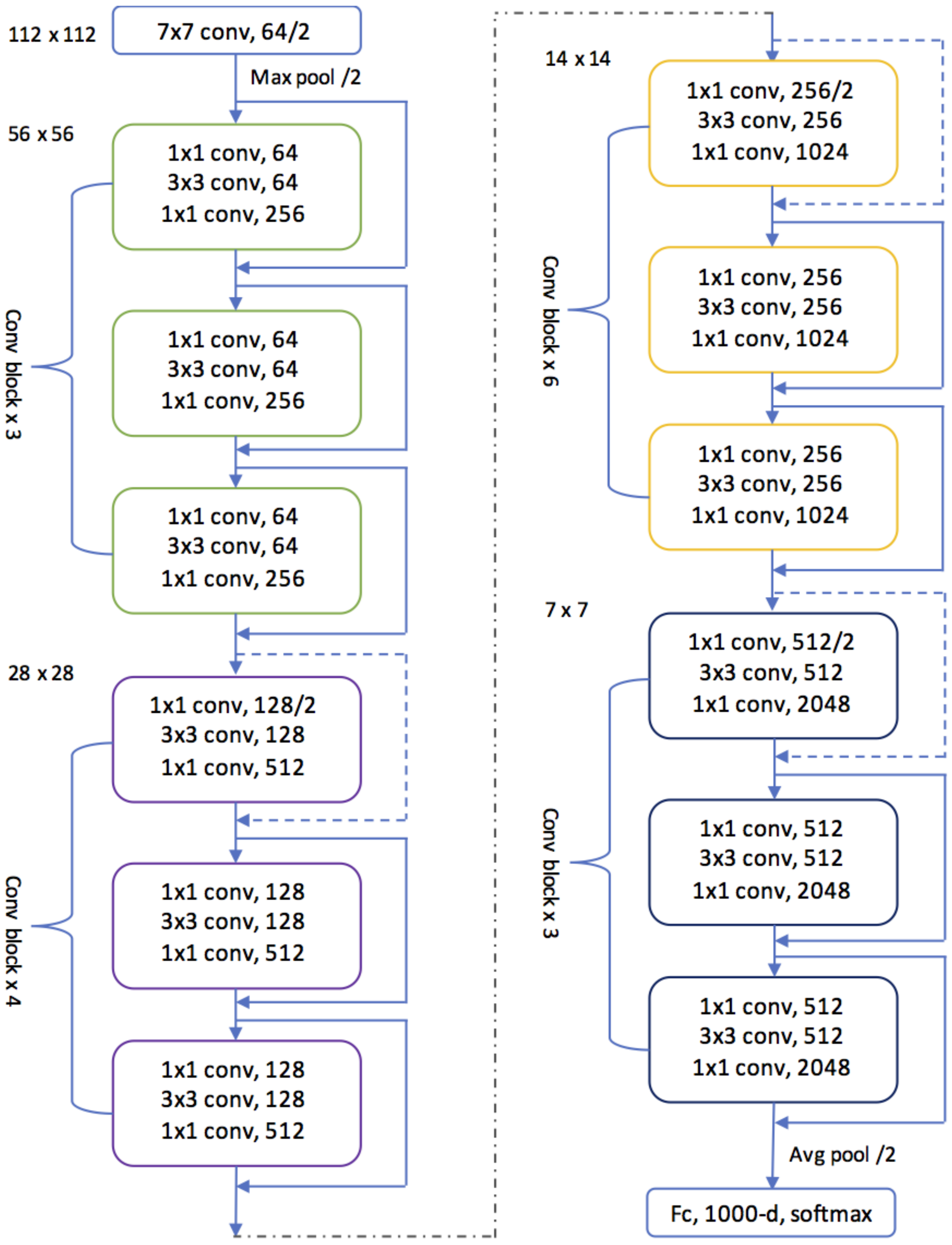


FIGURE 3.14: The architecture of 50-layer ResNet.

Chapter 4

Implementation and Results

In this chapter, we present in-depth information about our design, implementation and experiments on proposed different methodologies for automatic polyp detection. First, we describe our project requirements on both hardware configuration and software toolkits and libraries that are necessary to implement our design. We then provide the detailed information about our dataset preparation including patch extraction strategies and data augmentation process. Next,the definitions of performance metrics are presented in order to measure the effectiveness of our applications, including accuracy, precision, sensitivity/recall, F1-score, and specificity. Then, the chapter focuses on the implementation details about traditional machine learning methods discussed in previous chapters for polyp detection tasks. We describe 10-different classifiers utilized in our application which cover KNN, Liner SVM, RBF SVM, SGD, Decision tree, MLP, Random forest and so on. The experimental results in terms of each classifier's performance are therefore analyzed by visualization and comparison.

The most important part and major contributions of our work are related to Deep CNNs, we therefore present a comprehensive discussion in the following sectors with all aspects in terms of implementation, experimentation and evaluation with regard to our proposed deep learning framework. We first analyze two deep learning schemes: full-training and transfer learning based on related experimental results. We then highlight our transfer learning architecture along with its specific hyper-parameter list. Finally, we describe the detailed process of hyper-parameter fine-tuning by our unique experimentation and hand-tuning strategy. By analyzing a large number of learning curves, we therefore demonstrate our practical fine-tune and training skills like k-fold cross validation methods etc. The eventually implemented 9 deep models are illustrated with comprehensive evaluation and discussions along with our key findings and strategies.

4.1 Project requirements

4.1.1 Hardware requirements

Deep learning is a field with intense computational requirements, so advanced DCNNs always make use of the computational power of the graphics processing units (GPUs) to speed up all computation work, as with no GPUs this might take weeks or even months for an experiment to finish, or run an experiment for a day or more only to see that the chosen parameters were incorrect. In our work, we use one NVIDIA GTX970 GPU with 4GB RAM plus one CPU of Intel Core i7-6700@3.40GHz with 16GB RAM as our hardware platform. Table 4.1 shows the basic configurations and the tested configurations for our project.

TABLE 4.1: System configuration requirements.

	Basic configuration	**Tested configuration**
OS	Windows 10 or Ubuntu 14.04	Windows 10
CPU	Intel Core i5 2.7GHz	Itel Core i7 3.40GHz
RAM	8GB	16GB
GPU	N/A	Nvidia GeForce GTX 970
RAM	N/A	4GB

4.1.2 Software toolkits and libraries

There are many open source deep learning toolkits currently available. It is much more efficient to utilize the resources available in a deep learning toolkit than writing a deep learning algorithm from scratch. After careful evaluation based on the specific requirements and time constraints of our project, we chose to use the below listed toolkits and libraries in this work.

- **CUDA and cuDNN:** CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by NVIDIA and implemented by the GPUs that they produce. The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. Please refer to NVIDIA.cuDNN.

- **TensorFlow:** TensorFlow [2] is an open source Python library for fast numerical computing created and released by Google and released under the Apache 2.0 open source license. It is a foundation library that can be used to create Deep Learning models directly or by using other wrapper libraries like Keras that simplify the process built on top of TensorFlow. It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines. Please refer to Tensorflow.org.

- **Keras:** Keras is an open source API written in Python which uses as backend either Theano or Tensorflow. It was developed with a focus on enabling fast experimentation, so that it is easier to build complete solutions, and is easy to read with the greatest selection of state-of-the-art algorithms (optimizers, normalization routines, activation functions). Please refer to Keras.io.

- **Other APIs:** Besides the above libraries, we also utilize some other open source APIs that focus on more specific tasks, which include OpenCV, Pandasm, Numpy, Matplotlib, Scripy, H5py, QtPy, and so on. For more details, please refer to Appendix A.

4.2 Input data preparation

4.2.1 Dataset description

We validated our proposed methodology on our patch-balanced dataset generated from CVC-ColonDB dataset [7], which contained 300 colonoscopy frames with a total of 300 polyp instances extracted from 15 different colonoscopy video studies. These frames were selected in order to maximize the visual differences between them and provide an annotation of the region of interest (ROI) for all 300 images selected from all the sequences.

4.2.2 Patch extraction and augmentation

As the CVC-ColonDB dataset was very small and extremely unbalanced, we decided to utilize patch extraction and data augmentation techniques to generate a larger balanced dataset from the original dataset.

We propose the following methodology for patch extraction:

- Positive patches: we extract a patch (300*300) which covers the whole polyp from every frame (574*500).
- Negative patches (non-polyp patches): we crop the region which does not contain any part or only cover a little part of polyp from each frame.

Figure 4.1 illustrates the process of extracting positive and negative patches from a positive frame (containing a polyp)

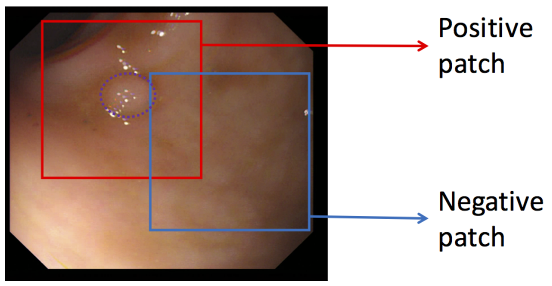


FIGURE 4.1: Patch extraction examples from a frame with a polyp.

After patch extraction, we make use of data augmentation techniques with horizontal and vertical flips, random rotations and so on to artificially boost the amount of positive and negative samples. Finally we generate our new balanced dataset with 2200 training samples and 400 test samples. The positive and negative sets are equal in size as shown in Table 4.2

4.2.3 Performance metrics

We propose the following performance measure metrics to indicate the effectiveness of our application, where TP stands for True Positive, FN- False Negative, TN- True Negative, FP- False Positive, P- Positive, and N- Negative, as shown in the table 4.3.

TABLE 4.2: Patch-balanced dataset after data augmentation.

Original CVC-ColonDB dataset			
Train		**Test**	
Positive	Negative	Positive	Negative
260 (574x500)	0	40 (574x500)	0
By patch extraction and data augmentation			
Patch-balanced Dataset for our experiments			
Train (81.2%)		**Test (18.8%)**	
Postive (50%)	Negative (50%)	Postive (50%)	Negative (50%)
1100 (300x300)	1100 (300x300)	200 (300x300)	200 (300x300)

TABLE 4.3: Definition of performance metrics.

	Polyp in the image	**No polyp in the image**
Predicts polyp presence	**TP**	**FP**
Predicts no polyp presence	**FN**	**TN**

- Accuracy: The proportion of all predictions that are correct. Accuracy is a measurement of how good a model is.

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

- Precision: The proportion of all positive predictions that are correct. Precision is a measure of how many positive predictions were actual positive observations.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

- Recall/Sensitivity: The proportion of all real positive observations that are correct.

$$\text{Recall/Sensitivity} = \frac{\text{TP}}{\text{TP+FN}}$$

- Specificity: The proportion of all real negative observations that are correct.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN+FP}}$$

- F1-Score: The harmonic mean of precision and recall.

$$\text{F1-score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

In practice, sensitivity/recall indicates how good a test is at detecting the positives, while specificity represents how good a test is at avoiding false alarms, and precision illustrates how many of the positively classified were relevant. Sensitivity, specificity and precision are the most used performance metrics in the medical field.

4.3 Traditional ML methods

4.3.1 Detecting process

We first propose to conduct some experiments on traditional machine learning methods with a set of popular classifiers to establish a benchmark of detection performance on the patch-balanced datasets. Figure 4.2 illustrates the traditional machine learning scheme for polyp detection.

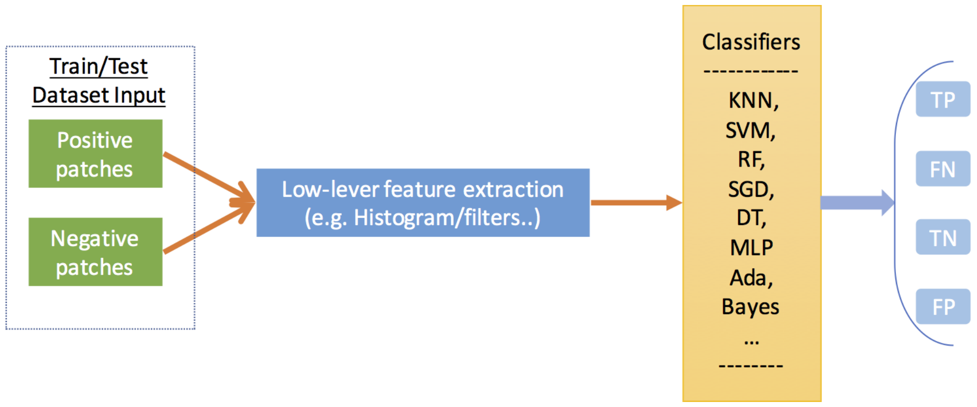


FIGURE 4.2: Traditional machine learning process for polyp detection.

Though SVM seems the most popular classifier and has achieved very good performance in image classification tasks according to our literature review, there are no single classification methods which outperforms all others on all data sets. Therefore, we decided to evaluate 10 different state-of-the-art classifiers together to compare their performance on our own data set. The 10 classifiers are shown below.

- KNN : K-nearest Neighbor (KNN) classifier implements based on the K nearest neighbors of each query point.
- Linear SVM: An implementation of SVM with a linear kernel requiring only one hyper parameter C which reduces the training and testing times by trading off misclassification of training examples against simplicity of the decision surface.
- RBF SVM: Another implementation of SVM with radical basis function (RBF) kernel requiring 2 parameters C and γ which defines how much a single training example has.
- SGD: Stochastic gradient descent(SGD) classifier requires a number of hyper parameters and is sensitive to feature scaling.
- GP: Gaussian Process (GP) classifier uses the whole sample's information to perform the prediction.
- DT: Decision tree (DT) classifier is a non-parametric method by using a tree-like decision model.

- RF: Random Forest (RF) classifier is a collection of ensemble decision trees, each tree in RF is built or grown from randomly selected subset.
- MLP: Multi-layer Perceptron (MLP) classifier is based on the feedforward ANN with multiple layers of nodes.
- AdaBoost: AdaBoost is a popular boosting method achieved by combining multiple weak learners into a single stronger classifier.
- Naive Bayes: Naive Bayes classifier is a set of learning algorithms based on Bayes' theorem with the naive assumption of independence between every pair of features.

4.3.2 Benchmark results

Table 4.4 shows the detailed results for each classifier's performance in terms of defined measure metrics, and Figure 4.3 visualizes the experimental results. As we can see from the data and figures, the Random Forest (RF) classifier had the overall best performance with average precision, recall and F1-score at 77%, which exceeded our expectations on SVM classifiers' performance. Both the linear SVM and RBF SVM had worse average performance than KNN classifier (at 76%) and GP classifier (at 73%). The experiments on traditional machine learning classification methods achieved general benchmarks with 77% overall precision, recall and F1-score for automatic detection of polyps in the patch-balanced dataset. Our next objective was to beat the benchmark and improve the overall detection performance by utilizing promising deep learning techniques.

4.4 Deep CNNs methods

4.4.1 Full training vs transfer learning

It is a great challenge to train DCNNs from scratch (full training). Not only because CNN requires a large number of domain tagged datasets which is difficult to achieve in the medical field, but also the training DCNNs require a lot of computing resources, without which the training process would be very time-consuming. Additionally, training DCNNs is often complicated by over-fitting and convergence problems, and it is often necessary to optimize a large number of learning parameters and architectures of the network to achieve proper convergence which requires a great deal of expertise and effort to ensure that all layers are learning at a considerable rate. As one of our experiments demonstrates in the figure 4.4, it came to a situation where the weights of the CNNs not converging even after 30 epochs when we tried to full-train a 5-layer CNNs with our patch-balanced dataset for over 10 hours with just 200 epochs.

In view of the above difficulties, a promising alternative to full training DCNNs is to transfer learning and fine-tune DCNNs pre-trained by a large labeled dataset from a different domain (e.g. ImageNet [42], which contains 1.2 million images with 1000 categories). The pre-trained models have been applied successfully to various computer vision tasks as a feature generator or as a baseline for transfer learning [39, 46]. In our work, we are using ResNet50 model [18] with weights pre-trained on ImageNet. Motivations for this model were a simultaneously deeper as well as computationally

TABLE 4.4: Classifiers comparison results.

Classifiers	Subset	Precision	Recall	F1-score
	polyp	0.74	0.79	0.77
KNN	none	0.77	0.73	0.75
	avg	0.76	0.76	0.76
	polyp	0.69	0.62	0.65
Linear SVM	none	0.66	0.72	0.68
	avg	0.67	0.67	0.67
	polyp	0.71	0.75	0.73
RBF SVM	none	0.74	0.69	0.71
	avg	0.72	0.72	0.72
	polyp	0.68	0.68	0.68
SGD	none	0.68	0.69	0.68
	avg	0.68	0.68	0.68
	polyp	0.71	0.76	0.73
Gaussian Process	none	0.74	0.69	0.71
	avg	0.73	0.72	0.72
	polyp	0.67	0.62	0.64
Decision Tree	none	0.64	0.69	0.67
	avg	0.65	0.65	0.65
	polyp	0.78	0.76	0.77
Random Forest	none	0.77	0.78	0.77
	avg	0.77	0.77	0.77
	polyp	0.68	0.68	0.68
MLP Classifier	none	0.68	0.68	0.68
	avg	0.68	0.68	0.68
	polyp	0.67	0.68	0.68
Ada Boost	none	0.68	0.67	0.67
	avg	0.67	0.67	0.67
	polyp	0.68	0.73	0.70
Naive Bayes	none	0.71	0.66	0.68
	avg	0.69	0.69	0.69

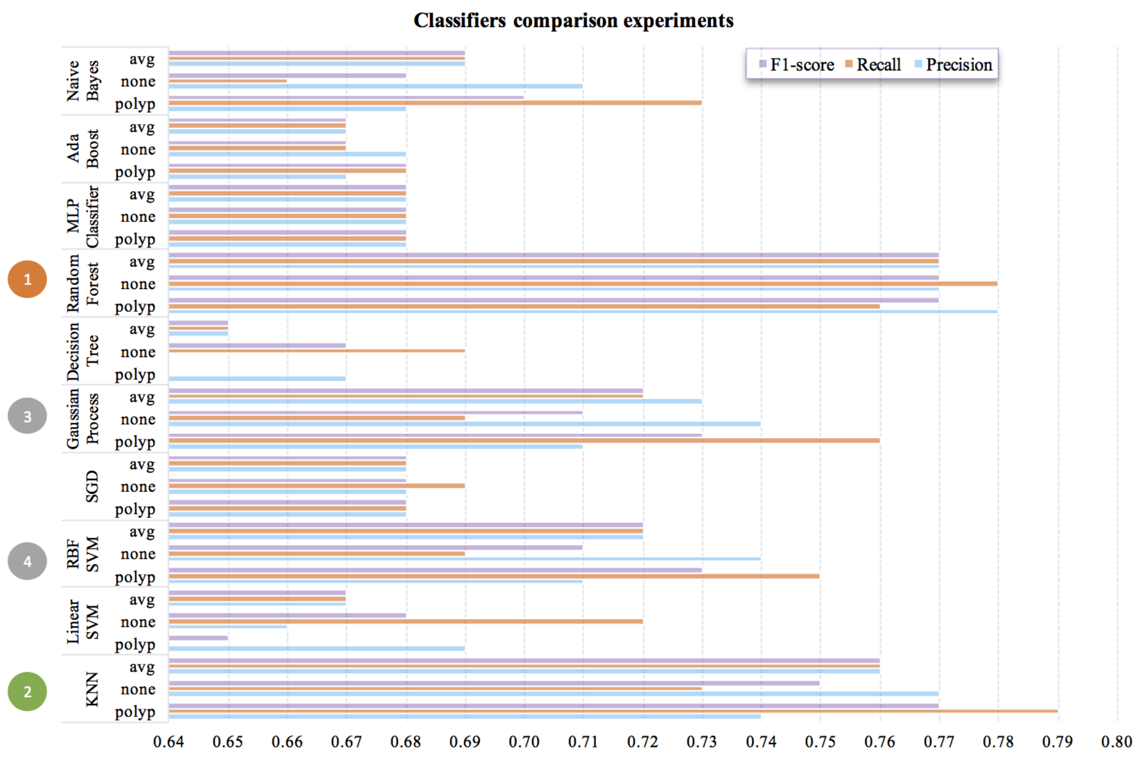


FIGURE 4.3: Classifiers comparison results of visualization. RF classifier achieved the overall best performance in terms of avg precision, recall and F1-score at 77%.

inexpensive architecture. These weights are ported from the ones released by Kaiming He under the MIT license.

4.4.2 Transfer learn and fine-tune

To transfer learn and fine tune ResNet50, we first design a new top layer to replace the FC-1000-d layer of ResNet50 as shown in Table 4.5. The new top layer consists of 2 new FC layers (FC-512 and FC-2) and one dropout layer between the 2 FC layers, as shown in Figure 4.5. The new top layer also uses softmax as the output layer to predict a separate probability for each of our categories: polyp or no-polyp, and the probabilities will all add up to 1.

We then transfer the learned ImageNet weights as the initial weights, and fine-tune the customized model with the new top layer by training and running back propagation on the built-in ResNet50 with our patch-balanced polyp dataset as shown in Figure 4.5.

4.4.3 Hyper parameters

The process of tweaking parameters for a given neural network architecture is known as hyper-parameter optimization. There is a brief list of hyper-parameters we tuned in our work:

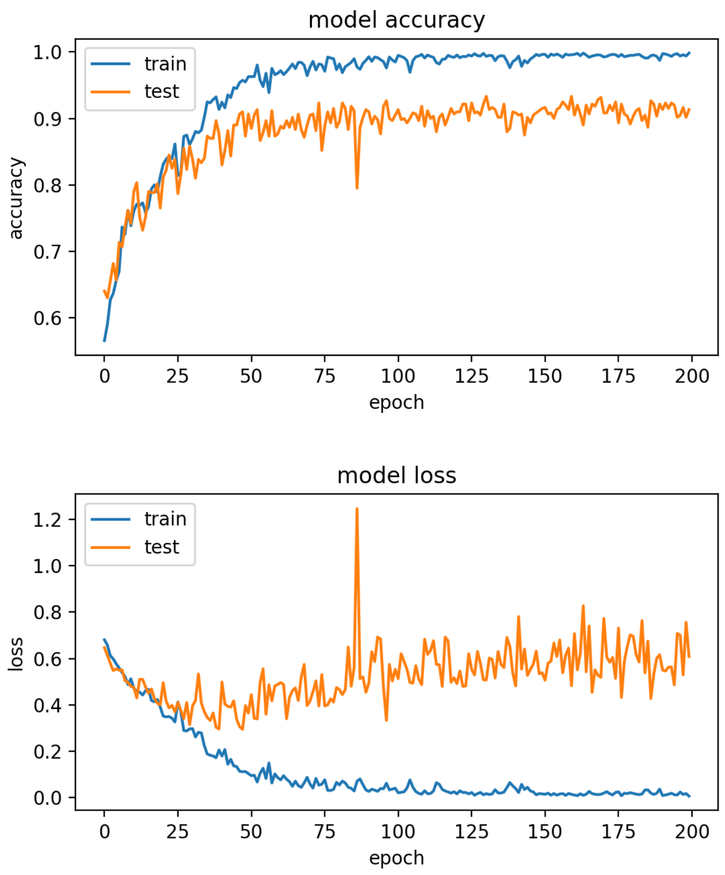


FIGURE 4.4: Learning curve of 5-layer CNNs by full-training from scratch.

- Learning rate (η) : Learning rate is one of the most important and sensitive parameters that multiplies the computed gradient in the update. The most common approach here is to start with a small learning rate and increase it exponentially if two epochs in a row reduce the error, while on the other hand decrease it rapidly if a significant error increase occurs.

- Decay rate (ρ): When training a deep neural networks, it is necessary to lower the learning rate as the training progresses by setting a proper decay rate. The learning rate is a parameter that determines how much an updating step influences the current value of the weights, while the weight decay is an additional term in the weight update rule that prevents over fitting and leads to convergence faster. Adadelta [60] uses exponential decaying methods. The detail algorithm was presented in the chapter 3.4.4.

- Batch size (Bs): In practice, batch size and learning rate are linked. If a batch size is too small then the gradients would become more unstable and would need to reduce the learning rate. And more, the higher the batch size, the more memory space we will need. Due to the limits of hardware configurations, the maximum batch size is up to 10 with 224x224 image size as input, and maximum 32 batch size for 100x100 inputs.

- Input size (Is): The size of image resized to feed to the model, which is really linked to batch size that depends on the GPU's capability, so we have to compromise on the setting because of the limitation of our hardware configuration as

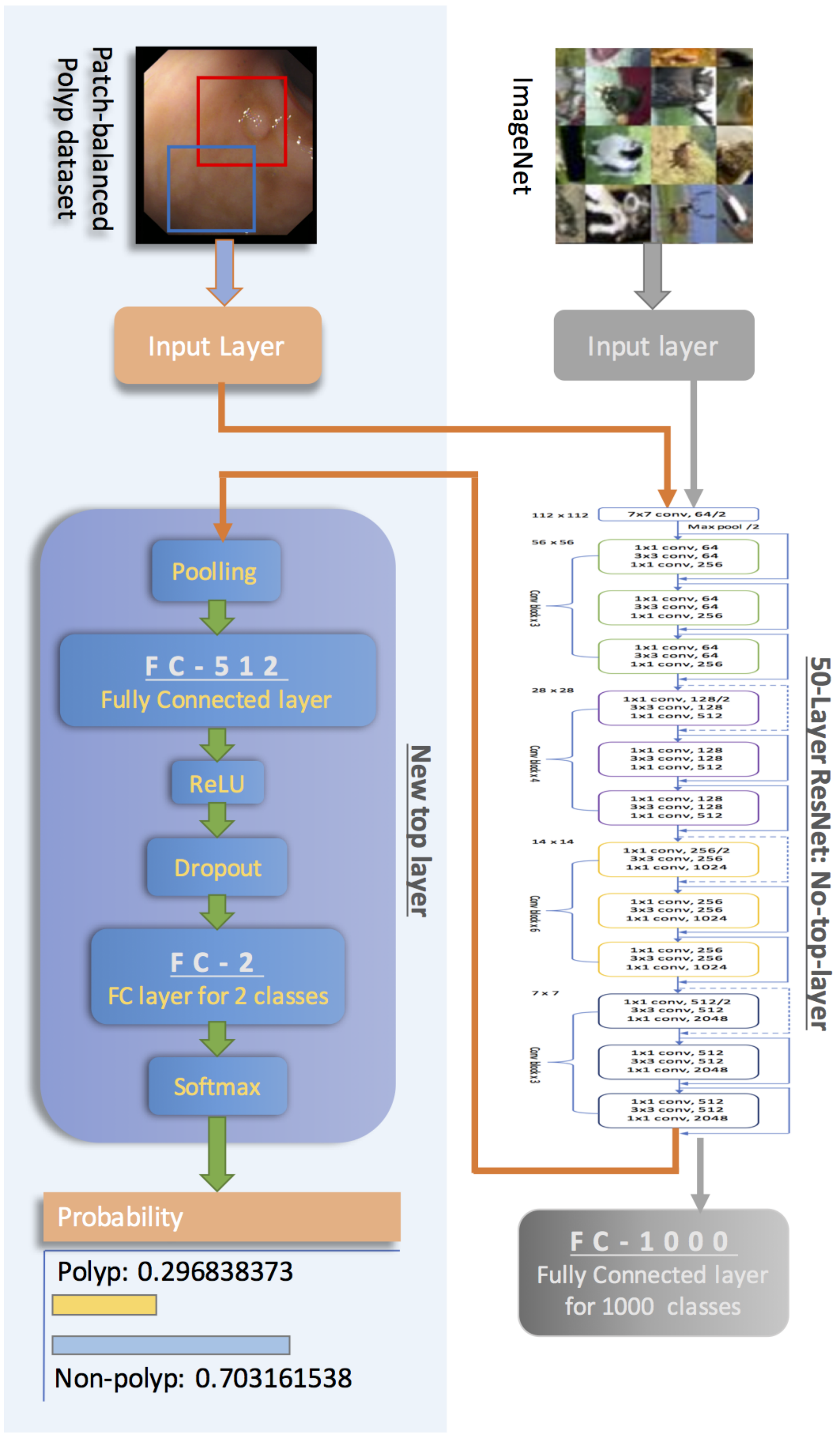


FIGURE 4.5: Transfer learning architecture, by full-tuning 50-layer ResNet model with a new designed top layer that consists of FC-512, Dropout, FC-2, and softmax output.

TABLE 4.5: The architecture of 50-layer RestNet.

Layer name	Output size	50-layer ResNet	
conv1	112 x 112	7 x 7, 64, stride 2	
3 x 3 max pool, stride 2			
conv2_x	56 x 56	1 x 1, 64 3 x 3, 64 1 x 1, 256	x 3
conv3_x	28 x 28	1 x 1, 128 3 x 3, 128 1 x 1, 512	x 4
conv4_x	14 x 14	1 x 1, 256 3 x 3, 256 1 x 1, 1024	x 6
conv5_x	7 x 7	1 x 1, 512 3 x 3, 512 1 x 1, 2048	x 3
FC	1 x 1	average pool, 1000-d fc, softmax	

mentioned above.

- Training epochs (Te): One epoch means one forward pass and one backward pass of all the training samples. Early stopping method can be applied given enough training dataset along with k-fold cross validation strategy. Typically a patience number should be defined first. Patience number stands for the number of epochs to wait before early stop if no progress on the validation set. The patience number is often set somewhere between 3 and 20.

- Dropout rate (Dr): Dropout is a simple but quite effective way to regularize the neural networks and address the over-fitting problem. It has been demonstrated that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets [48]. However, too high rate could result in under-fitting problem as well base on our experimentations.

- Pooling size (Ps): In our neural networks, we utilized average pooling methods before the fully connected layers in order to reduce the resolution of the feature map but retain features of the map required for classification through translational and rotational invariants. Its default filter size is 7×7 which should decrease carefully to smaller filters (2×2, 3×3, or 5×5) in order to fit different input image sizes.

4.4.4 Experimentation and hand-tuning

For our unique problems and deep architecture, experimentation and hand-tuning (hand-random-search) can be an ideal way to start fine-tuning our hyper parameters, since a few training epochs and evaluations can give us a good judgment which settings would be suitable or not, and then we can give better and better configuration for the next set of parameters.

We fist conduct experiments to primarily establish a proper setting range for the most important 3 hyper parameters (learning rate, decay rate, and drop rate) of the TL framework, whereby we can avoid wasting time on poor settings which would likely have resulted in worse performance caused by over-fitting or non-convergent problems. However, for our project, fine-tuning hyper-parameter directly on the whole dataset is too costly and time-consuming considering the limitation of hardware. Therefore, we decided to take the following hand-tuning strategy:

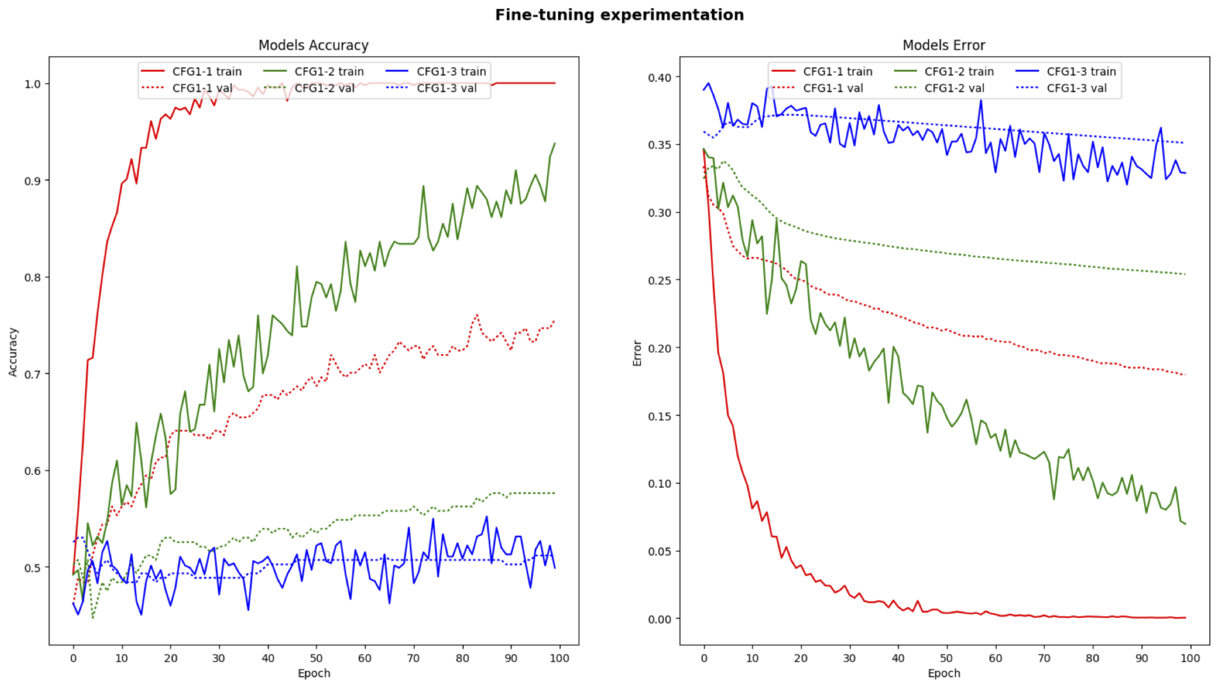
- Set up a subset database (sub-sample 650 images from our patch-balanced dataset).
- Tune a hyper-parameter one time on the subset database with a small input size and big batch size to reduce training time.
- Figure out a rough setting range of the hyper-parameter by observing and analyzing the experimental results and related learning curves.
- Further fine-tune parameters within the rough range on the whole dataset with increased input sizes.
- Determine a more accurate setting range of hyper-parameter by careful trade-offs in tuning among different combination settings.

By using a small dataset we could quickly get a rough but valuable information about the detail performance of our network against different parameter settings. Figure 4.6 illustrates an experimentation for searching proper learning rates. It only took us about 1 hour to get a valuable indication that the learning rate should not be set lower than 0.01 for our transfer learning system.

We then repeated a few similar experiments afterward with a set of different learning rates ranging from 0.01 to 0.09. At last we worked out reasonable learning rates that could be in the range of 0.04-0.07. Based on the same strategy, we then fixed the learning rate at 0.05 to tune the dropout rate as shown in Figure 4.7, and the decay rate as shown in Figure 4.8.

The other hyper parameters (input size, batch size, epoch and pooling size) are linked and depend on the size of networks and GPU's capability. We have to take some trade-offs and compromise on their settings based on our hardware configuration so that their varying ranges are not so wide as learning rate or drop rate. Thus, it is easier for us to optimize since there are only several optional combination settings for them. Figure 4.9 illustrate an experiment for tuning the batch-size. The results indicated the proper range of batch-size could be from 16 to 24 giving the input image resize of 100x100.

When we determined the rough range of each key hyper parameter on the subset database, we need to conduct a large number experiments on the whole dataset to further evaluate and adjust the setting ranges. Through observing plenty of experimental results, we finally established reasonable configuration ranges of the considered hyper parameters for our TL framework as shown in Table 4.6.



CFG1-1: **η:0.01**, ρ: 0, Dr: 0.5, Is: 100×100, Bs: 20, Ps: 3×3

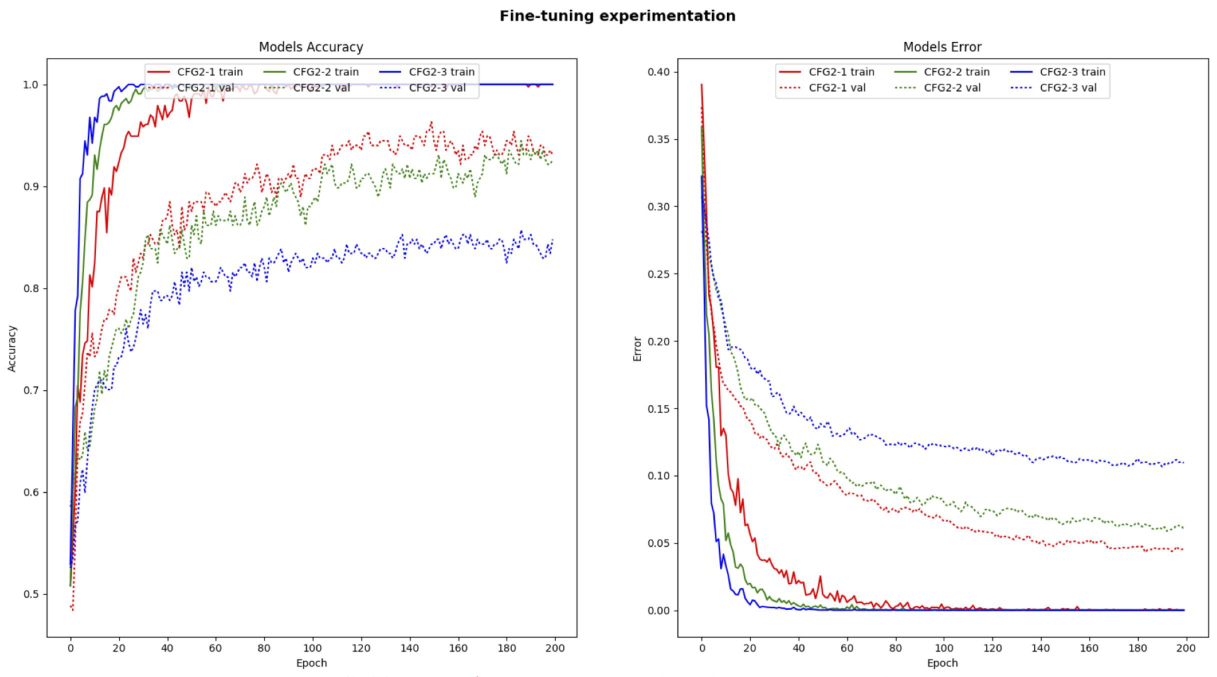
CFG1-2: **η: 0.001**, ρ: 0, Dr: 0.5, Is: 100×100, Bs: 20, Ps: 3×3

CFG1-3: **η: 0.0001**, ρ:0, Dr:0.5, Is: 100×100, Bs: 20, Ps:3×3

FIGURE 4.6: Fine-tuning learning rate experimentation. We first only tuned learning rates and fixed other parameters (ρ:0, Dr:0.5, Ps:3$\times$3) with fixed small input image size (100 $\times$ 100) and big batch size (20) in order to reduce the training time. It is clear that CFG1-1 configured with η:0.01 performed much better than the other two models (CFG1-2 and CFG1-3) with setting of lower learning rates at η:0.001 and η:0.0001 separately. The results showed that the proper range of learning rate should be above 0.01 for our system. Based on this general idea, we can avoid wasting time to try the learning rates lower than 0.01 for the next experimentations.

TABLE 4.6: The suggested setting ranges of hyper parameters for the proposed TL framework.

Hyper	Setting Range		Examples		
Parameters	From	To	CFG-1	CFG-2	CFG-3
η	0.045	0.055	0.049	0.05	0.05
ρ	0.002	0.0027	0.0025	0.002	0.0023
Dr	0.75	0.85	0.75	0.8	0.85
Is	100x100	224x224	100x100	150x150	224x224
Te	20	200	200	100	30
Bs	5	32	32	25	8
Ps	2x2	7x7	3x3	3x3	7x7

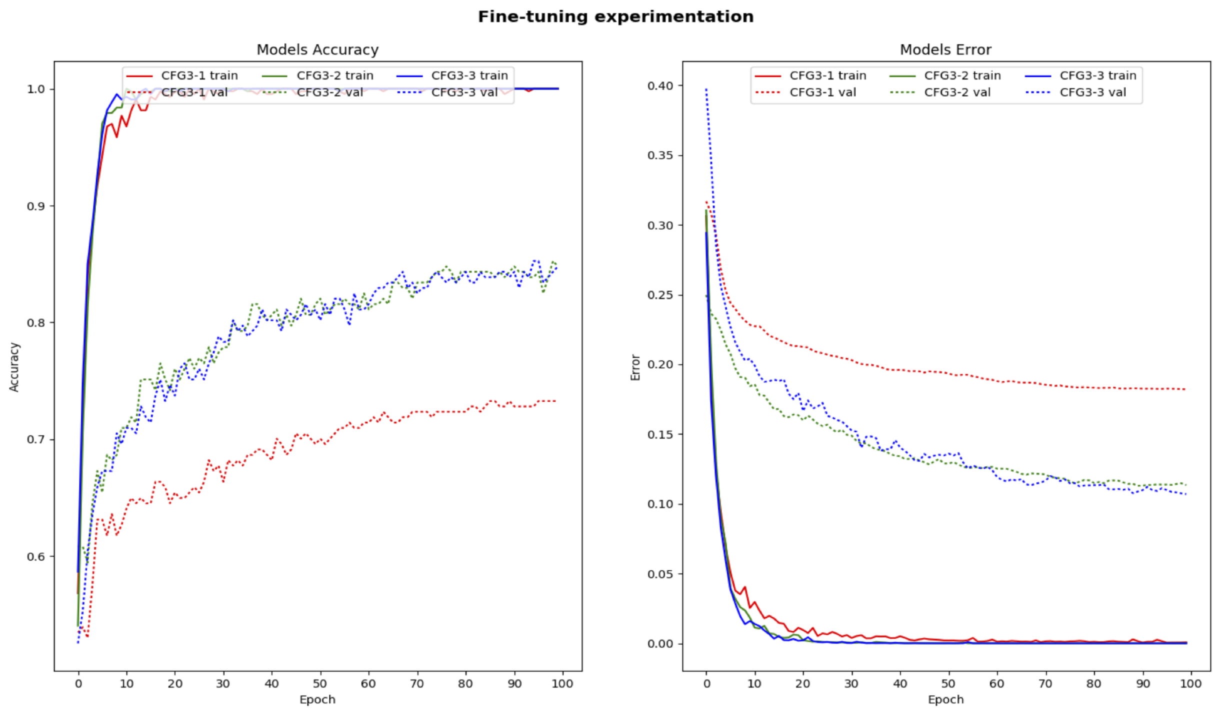


CFG2-1: η: 0.05, *ρ*: 0, ***Dr*: 0.8**, *Is*: 100×100, Bs: 20, Ps: 3×3

CFG2-2: η: 0.05, *ρ*: 0, ***Dr*: 0.7**, *Is*: 100×100, Bs: 20, Ps: 3×3

CFG2-3: η: 0.05, ρ: 0, ***Dr*: 0.6**, *Is*: 100×100, Bs: 20, Ps: 3×3

FIGURE 4.7: Dropout rate tuning experimentation.



CFG3-1: η: 0.05, ***ρ*: 0.01**, *Dr*: 0.5, *Is*: 100×100, Bs: 20, Ps: 3×3

CFG3-2: η: 0.05, ***ρ*: 0.001**, *Dr*: 0.5, *Is*: 100×100, Bs: 20, Ps: 3×3

CFG3-3: η: 0.05, **ρ: 0.0001**, *Dr*: 0.5, *Is*: 100×100, Bs: 20, Ps: 3×3

FIGURE 4.8: Tuning decay rate experimentation.

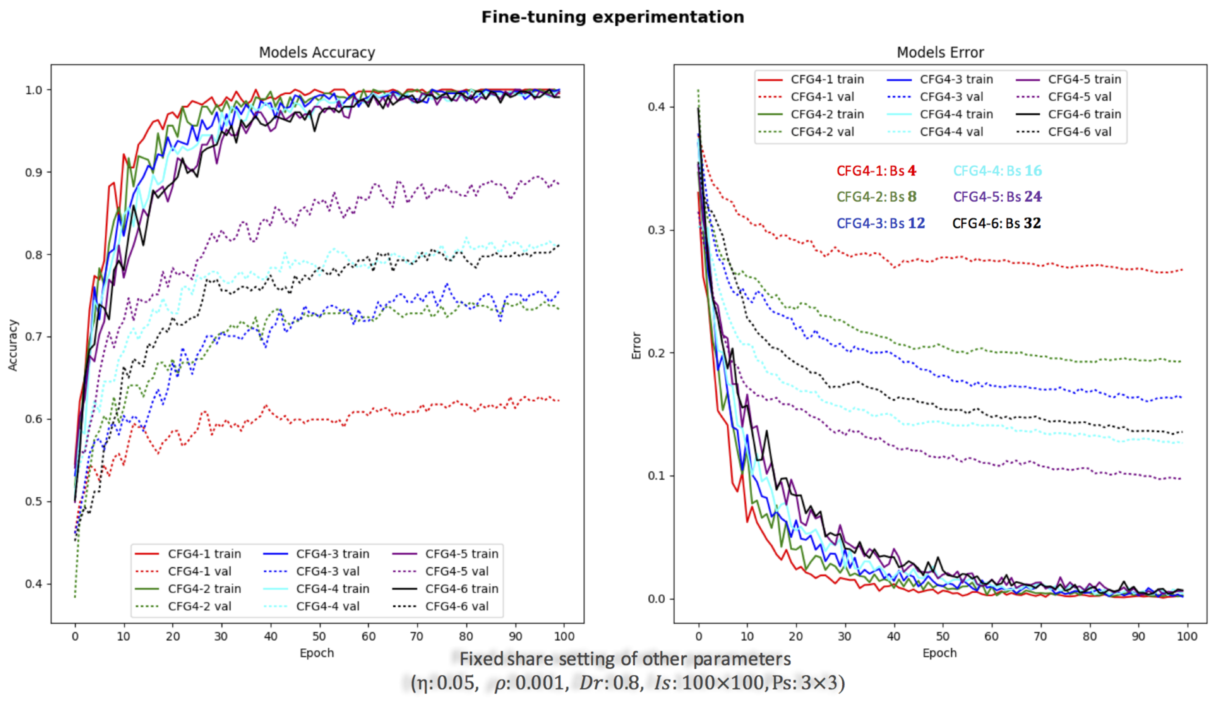


FIGURE 4.9: Batch-size experimentation.

4.4.5 K-fold cross validation

In our work, K-fold (K) Cross Validation (CV) approach is employed to estimate the performance of various models configured by different setting of hyper parameters. K-fold CV method works by splitting the dataset into K parts (e.g. $K = 3$, 5 or 10). Each split of the data is called a fold. Each model is trained on K-1 folds with one held back and validated on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the validation set.

TABLE 4.7: An example of 3-fold cross validation process.

Train data			Test data
random split by 3			
fold-1	fold-2	*fold-3 for val*	test data
fold-1	*fold-2 for val*	fold-3	test data
fold-1 for val	fold-2	fold-3	test data

The table 4.7 shows an example of 3-fold CV process. The training data is split into 3 folds, the folds 1-2 first become the training set. Fold 3 here is denoted as the validation fold for tuning the hyper-parameters. Once the training process of given epochs is completed, the model is tested a single time on the test data (marked yellow). Therefore, after running 3-fold CV we gain 3 different performance scores on the test dataset that we can summarize using a mean and a standard deviation. The result is more accurate because the model is trained and evaluated multiple times on different data.

4.5 Evaluation and discussion

We conducted a set of different experiments on 9 models with specific input sizes and optimized hyper-parameters as shown in the Tables 4.8 and 4.9. Their training curves are showing in Figures 4.20 to 4.12. Basically, model-0, -1, and -2 have the same input size of 100x100; model-3, -4, and -5 use a slightly bigger input size of 150x150; while model-5, -6, and -8 take the default input size of 224x224. Figure 4.10 visualized the overall performance of these 9 different models compared with the benchmark of RF-BM, where RF-BM stands for the benchmark of random forest classifier.

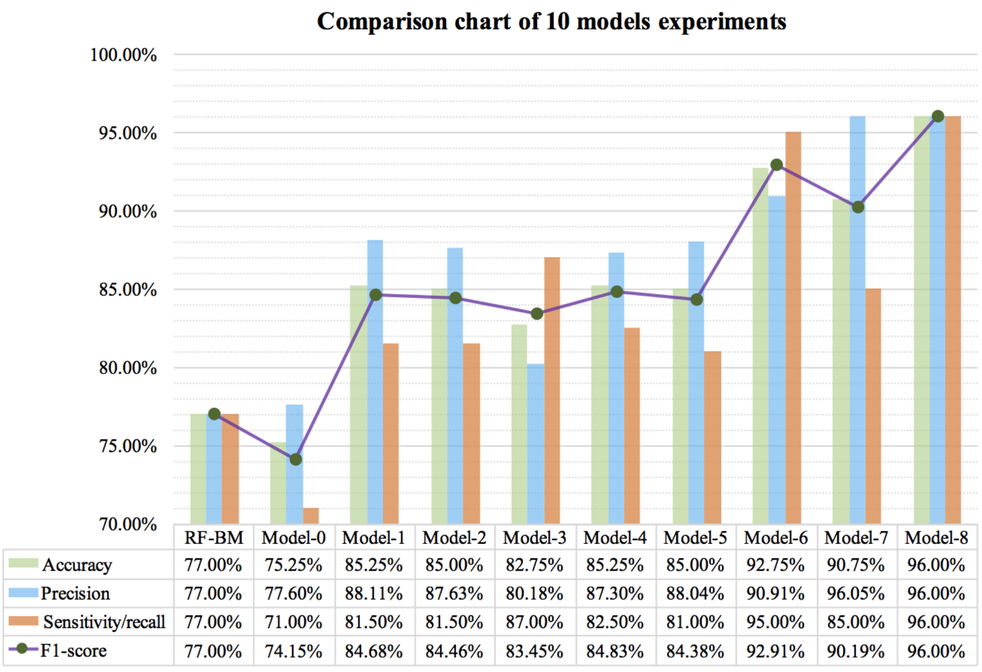
We eventually achieved the best overall 96.00% polyp detection accuracy, precision, sensitivity, specificity, and F1-score with Model-8 configured by the optimized hyper-parameters (η:0.05, ρ:0.0025, Bs:10, Dr:0.8, Ps:7x7, and Te:50), which outperformed the traditional machine learning classification methods in each defined performance metric, such as feature-based SVM (avg-72%) and RF (avg-77%).

TABLE 4.8: Model-1 to model-5 polyp detection results.

Hpyer-Parameters	**Model-1**	**Model-2**	**Model-3**	**Model-4**	**Model-5**
K	3	3	3	5	3
Te	150	200	50	100	150
Is	100x100	100x100	150x150	150x150	150x150
Bs	25	32	10	10	16
Dr	0.8	0.75	0.805	0.803	0.8
η	0.049	0.049	0.049	0.049	0.049
ρ	0.0025	0.0025	0.002	0.0023	0.0025
Ps	3x3	3x3	3x3	3x3	3x3
Test Results					
FP	22	23	43	24	22
FN	37	37	26	35	38
TP	163	163	174	165	162
TN	178	177	157	176	178
Performances					
Accuracy	85.25%	85.00%	82.75%	85.25%	85.00%
Precision	88.11%	87.63%	80.18%	87.30%	88.04%
Sensitivity/recall	81.50%	81.50%	87.00%	82.50%	81.00%
F1-score	84.68%	84.46%	83.45%	84.83%	84.38%
Specificity	89.00%	88.50%	78.50%	88.00%	89.00%

4.5.1 Impact of input size

From what we can observe, the larger input size used to train the models, the better performance achieved given proper hyper parameters. For instance, by comparing



	RF-BM	Model-0	Model-1	Model-2	Model-3	Model-4	Model-5	Model-6	Model-7	Model-8
🟩 Accuracy	77.00%	75.25%	85.25%	85.00%	82.75%	85.25%	85.00%	92.75%	90.75%	96.00%
🟦 Precision	77.00%	77.60%	88.11%	87.63%	80.18%	87.30%	88.04%	90.91%	96.05%	96.00%
🟧 Sensitivity/recall	77.00%	71.00%	81.50%	81.50%	87.00%	82.50%	81.00%	95.00%	85.00%	96.00%
● F1-score	77.00%	74.15%	84.68%	84.46%	83.45%	84.83%	84.38%	92.91%	90.19%	96.00%

FIGURE 4.10: 10 Models performance comparison, where RF-BM stands for the benchmark of Random Forest classifier, and model-0 to model-8 present different settings of hyper-parameters ($K, Te, Is, Bs, Dr, \eta, \rho$ and Ps).

TABLE 4.9: Model-6 to model-8 polyp detection results.

Hpyer-Parameters	Model-6	Model-7	*Model-8*	Model-0	Model-3
K	3	3	**3**	3	3
Te	*50*	*50*	***50***	*50*	*50*
Is	*224x224*	*224x224*	*224x224*	*100x100*	*150x150*
Bs	10	5	**10**	10	10
Dr	0.805	0.805	**0.8**	0.805	0.805
η	0.049	0.049	**0.05**	0.049	0.049
ρ	0.002	0.0025	**0.0025**	0.002	0.002
Ps	7x7	7x7	**7x7**	3x3	3x3
Test Results					
FP	19	7	**8**	41	43
FN	10	30	**8**	58	26
TP	190	170	**192**	142	174
TN	181	193	**192**	159	157
Performances					
Accuracy	92.75%	90.75%	***96.00%***	75.25%	82.75%
Precision	90.91%	***96.05%***	***96.00%***	77.60%	80.18%
Sensitivity/recall	95.00%	85.00%	***96.00%***	71.00%	87.00%
F1-score	92.91%	90.19%	***96.00%***	74.15%	83.45%
Specificity	90.50%	***96.50%***	***96.00%***	79.50%	78.50%

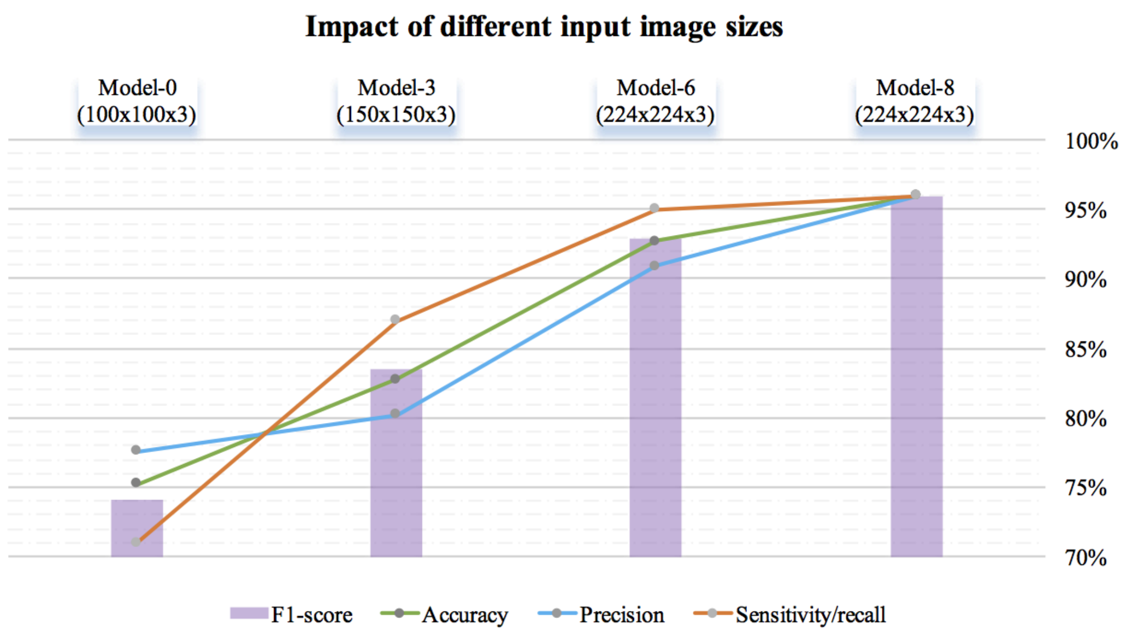


FIGURE 4.11: The impact of different input image sizes. The larger input sizes could dramatically improve the model's performance, while greater number of system memories and training time is required as well.

Model-0 (100x100), Model-3 (150x150) with Model-6 (224x224), the hyper parameters almost have the same setting, but Model-6 achieved much better performances than Model-3 and Model-0 - accuracy (92.75% vs 82.75% vs 75.25%), precision (90.91% vs 80.18% vs 77.60%), sensitivity (95.00% vs 87.00% vs 71.00%), and F1-score (92.91% vs 83.45% vs 74.15%) as shown in the figure 4.11. The best sensitivity rates achieved by input size of 100x100 and 150x150 are only 81.50% (Model-1) and 87.00% (Model-3) separately among all the models, while for the input size of 224x224, we finally achieved 96.00% sensitivity and F1-score with Model-8.

Though the larger input sizes improve the performance of models, a longer time is required for each training epoch, and a greater number of system memories are occupied as well. Sometimes the out-of-memory issue may occur during training DCNNs with too big input sizes. In this case, either the input size and batch size must be decreased, or the system hardware configuration improved with greater memory and more powerful CPUs to solve the problem.

4.5.2 Tackling over-fitting

Deep Neural networks is known to overfit easily due to the large number of parameters. In our experiments, the overfitting was expected to be significant since the dataset was small even if we utilized data augmentation. To Tackle over-fitting problems, we introduced a dropout layer built-in the top FCNs layer in our proposed TL framework. We traded off three key hyper parameters- dropout rate, learn rate, and decay rate to effectively prevent over-fitting problems. For instance, by comparing Model-6 and Model-8, we can see that Model-6 had a slight over-fitting problem with K-2 fold training process as shown in the learning curve of Model-6, while after increasing the decay rate from 0.002 to 0.0025, learn rate from 0.049 to 0.05, and slightly downing the drop rate from 0.805 to 0.8, Model-8 mitigated the over-fitting problem and achieved better performance than Model-6 as shown in Table 4.9.

Meanwhile, from our observation, we also found bigger batch size could mitigate over-fitting issues in some way, and greater input sizes commonly require a slightly higher dropout rate and decay rate to avoid over-fitting than smaller input sizes, however, too high dropout rate or decay rate could result in under-fitting problems as well, as shown in the curve of Model-7 and Model-0.

However, through our experiments, we can observe surprisingly that it does not lead to an overall performance improvement by increasing k-fold or training epochs by comparing the model-1 with model-2, or model-4 with model-5 or model-3. Meanwhile, the decay rates should also be altered a little higher to avoid over-fitting if the training epochs are increased significantly.

4.5.3 Fine-tuning hyper-parameter

Fine-tuning the hyper parameters of a neural network is a tricky process, and there are many different approaches. We utilized hand-tuning methods (hand-random-search on experimentations) for our project rather than automatic tuning algorithms such as **Grid search** [20] or **Random search** [5]. Because these automatic methods would take too long time to finish tuning process. For instance, if we take the grid search method which is a simple and straightforward algorithm. We just need to define a set of hyper parameter values to train the model for all possible parameter combinations and select the best one. Imagine that we need to optimize 7 parameters. Let's assume that we just try only 10 different values per each parameter. Therefore, we need to make almost 10,000,000 (10^7) evaluations. Assuming that the model trains 1 hour on average we would have finished the tuning process in almost 700 years. Even if we choose random-search algorithm (instead of trying all possible combinations we only just take a randomly selected subset of the parameter combinations), it would also take at least 7 years by best guess. Although there are other automatic tuning methods such as **Bayesian Optimization** [35] and **TPE algorithms** [6] that show great improvement over the grid-search or random-search methods by allowing to learn from the training history and give better and better estimations for the next set of parameters, it would still take too long time to apply these algorithms. Therefore, experimentation and hand-tuning is still the best approach for unique problems and deep neural networks.

4.5.4 Generalization

As we can observe in our experiments, the proposed TL models generalize quite well given that the training accuracy are almost all 100%. First, dropout strategy improves the generalization capability of our models by a large rate (at 0.8% for Model-8). Second, for the structure of ResNets, batch normalization applied in convolutional blocks also help improve both the training speed and generalization. Another important reason is that we replace the fully-connected layer of ResNet50 by a global average pooling layer, and 2 FC layers before the softmax output layer, which greatly reduces the amount of parameters. Thus, our TL DCNNs models demonstrate very strong generalization capability with the state-of-the-art performance.

4.5.5 Constraints

Since we are using the pre-trained model, the convolutional filters, the kernel size, and the number of layers are fixed in our TL architecture. We are also slightly constrained in terms of the model architecture. For example, we can't arbitrarily take out certain convolutional layers from ResNet50. However, the input layer with different image size can be customized due to parameter sharing.

Another constraint of our work is the hardware. When training a deep neural networks, the system has to keep all the intermediate activation outputs for the backwards pass. So we need to compute how much memory it will need to store all the relevant activation outputs in the forward pass, in addition to other memory constraints such storing the weights on the GPU and so on. Since our model is quite deep with 50-layers, we have to take a smaller batch-size as we do not have enough system and GPU memory. For instance, we are not able to take batch-size over 10 given the input size of 224x224 due to our GPU's memory constraints, which actually limited our system's performance. In practice, especially in the case of deep learning with GPUs, larger batches are very attractive computationally and it is very common to take larger batch-sizes that fully leverage the GPU.

4.5.6 Proposed strategy

Based on all the results and analysis above, it is clear that hyper-parameter optimization is the key to ensure the model does not over-fit the training dataset by tuning which could make the model achieve the best generalized performance on test domain data. However, hyper-parameter optimization is still very much an open question in deep learning pipeline. There are currently no good theoretical frameworks for doing so automatically. Therefore experimentation and random search (hand-tuning) can be the best strategy to start fine-tuning hyper parameters so far.

For our proposed TL framework for automated polyp detection, the best setting of hyper parameters to obtain promising performance would be one similar to the configuration of Model-8 ($\eta = 0.05$, $\rho = 0.0025$, $Dr = 0.8$, $Bs = 10$, $Ps = 7 \times 7$, $Is = 224 \times 224 \times 3$, and $Te = 50$). However, in practice, the setting of hyper parameters might need to be altered carefully with the variability in the size and resolution of source data. There could be trade-offs in tuning among drop rate, decay rate, learning rate, and so on according to different domain dataset, to achieve state-of-the-art performance with strong generalization capability. From what we can observe on a large number of experimental data and figures, we highlight some useful strategies as below:

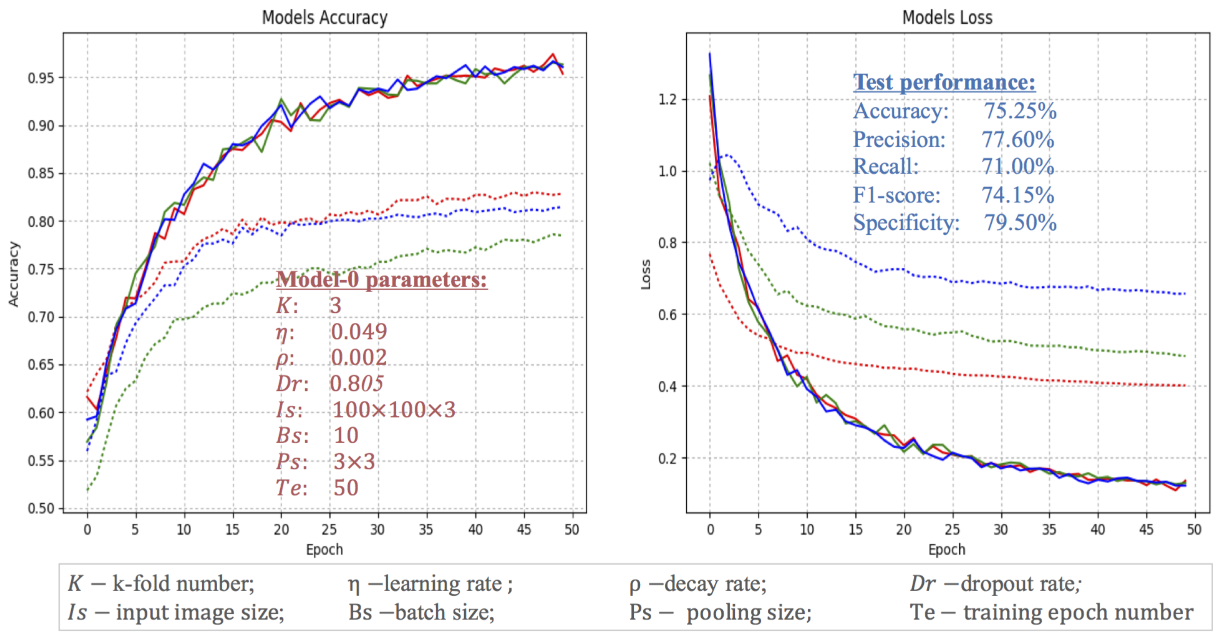
- Tuning some key hyper parameters on a small subset database could allow you to quickly establish a rough but very valuable tuning range of each parameter. The subset should be sub-sampled from your own entire dataset.

- Once you establish a rough tuning range of each hyper parameter, you could further conduct a set of specific experimentation within the range but with a smaller scale to alter each parameter one time.

- After above two steps, you could obtain both more accurate setting ranges of each parameter and high valuable insights on the performance of your system against different settings.

In addition, from what we can observe on a large number of experiments, the system's test performance, in terms of accuracy, precision, sensitivity, specificity and F1-score, can be significantly affected by some just slight changes among several key hyper parameters like dropout rate, decay rate and learning rate in our case. For instance, looking at Model-6, -7, and Model-8 in Table 4.9, Model-8 has just slightly increased the learning rage to 0.05 from 0.049, and decreased the dropout rate to 0.8 from 0.805, and keep the day rate at 0.0025 same with Model-7, but surprisingly Model-8 finally yields much better results than Model-6 and Model-7.

All in all, DNN hyper-parameter tuning is still considered as a "dark art", mastering the 'dark art' requires not only a solid background in machine learning algorithms, but also extensive experience working with real-world datasets.



Model-0 parameters:

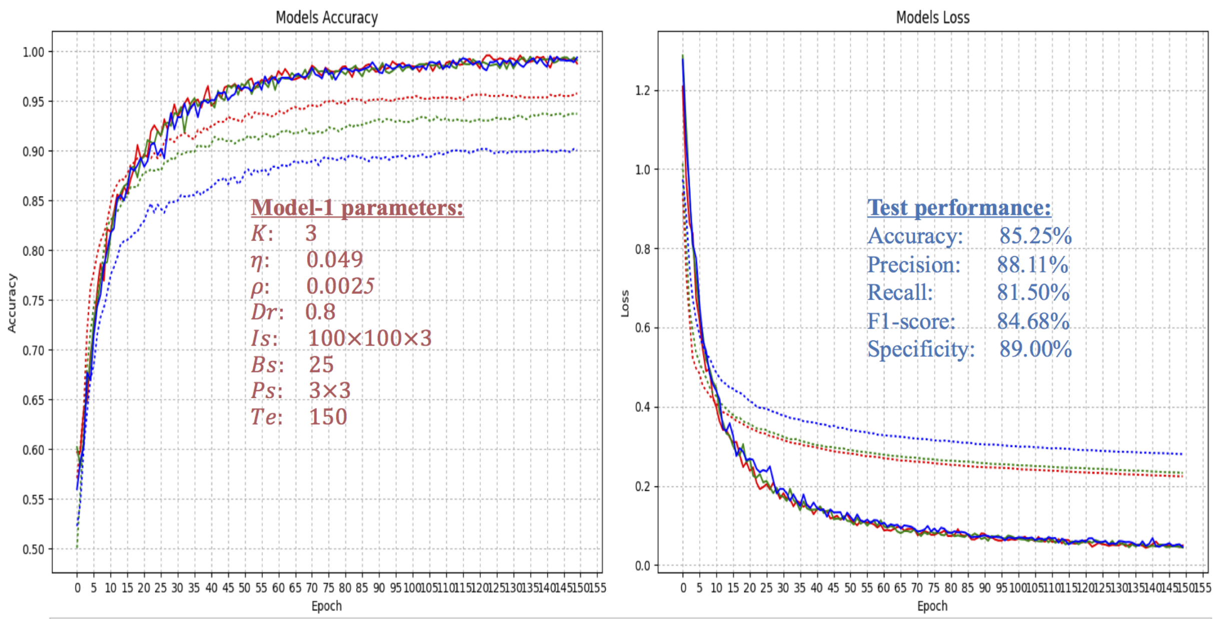
K:	3
η:	0.049
ρ:	0.002
Dr:	0.805
Is:	100×100×3
Bs:	10
Ps:	3×3
Te:	50

Test performance:

Accuracy:	75.25%
Precision:	77.60%
Recall:	71.00%
F1-score:	74.15%
Specificity:	79.50%

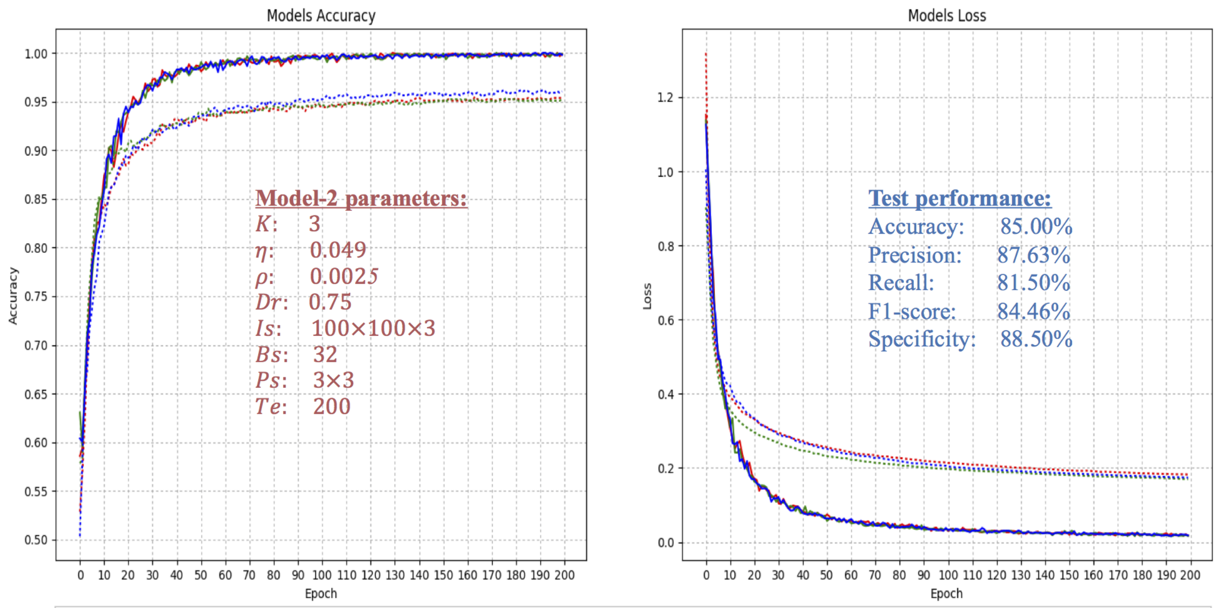
K – k-fold number; η –learning rate ; ρ –decay rate; Dr –dropout rate; Is – input image size; Bs –batch size; Ps – pooling size; Te – training epoch number

FIGURE 4.12: The learning curve of Model-0.



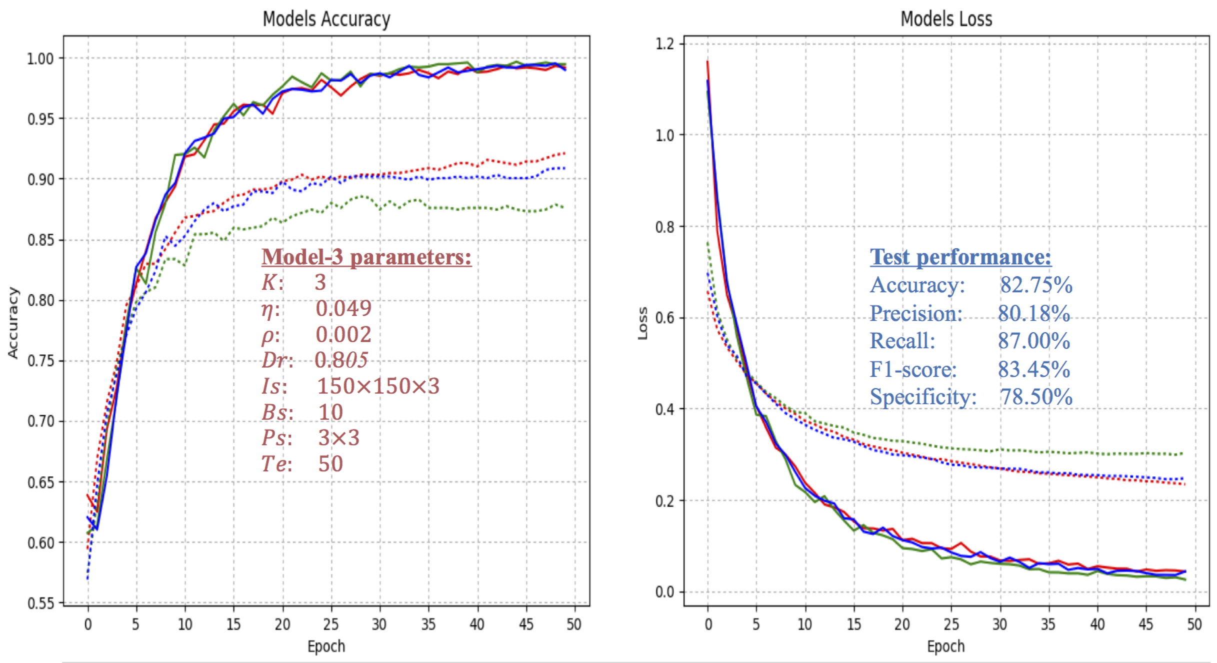
K — k-fold number;	η —learning rate ;	ρ —decay rate;	*Dr* —dropout rate;
Is — input image size;	Bs —batch size;	Ps — pooling size;	Te — training epoch number

FIGURE 4.13: The learning curve of Model-1.



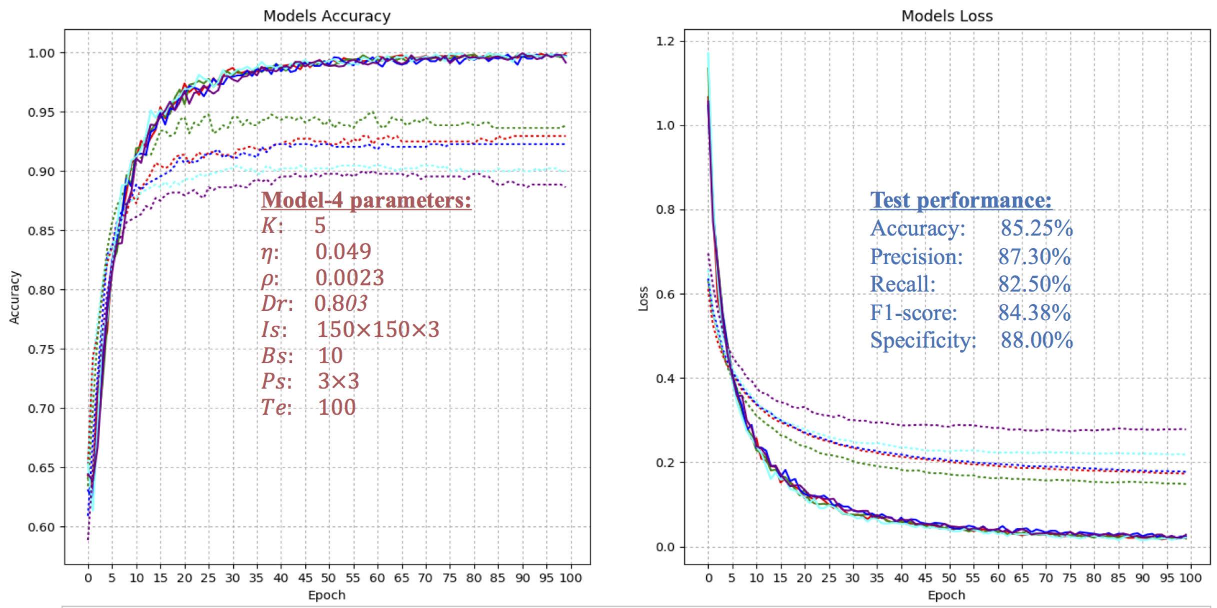
K — k-fold number;	η —learning rate ;	ρ —decay rate;	*Dr* —dropout rate;
Is — input image size;	Bs —batch size;	Ps — pooling size;	Te — training epoch number

FIGURE 4.14: The learning curve of Model-2.



K — k-fold number;	η —learning rate ;	ρ —decay rate;	*Dr* —dropout rate;
Is — input image size;	Bs —batch size;	Ps — pooling size;	Te — training epoch number

FIGURE 4.15: The learning curve of Model-3.



K — k-fold number;	η —learning rate ;	ρ —decay rate;	*Dr* —dropout rate;
Is — input image size;	Bs —batch size;	Ps — pooling size;	Te — training epoch number

FIGURE 4.16: The learning curve of Model-4.

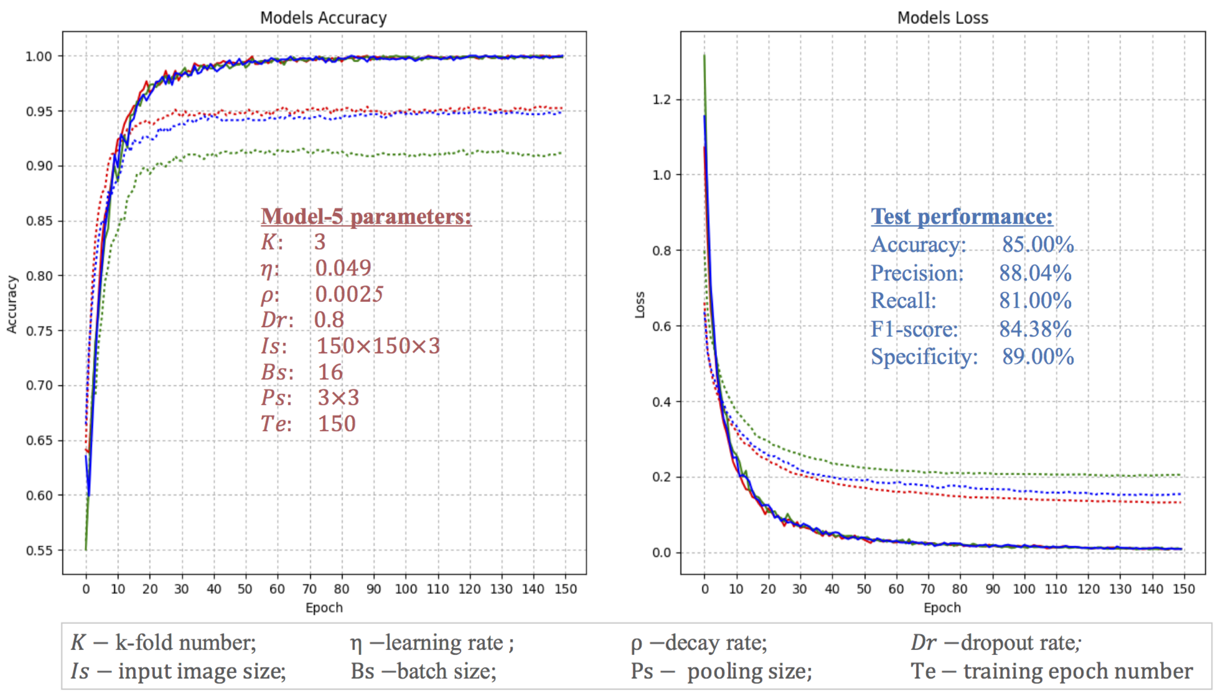


FIGURE 4.17: The learning curve of Model-5.

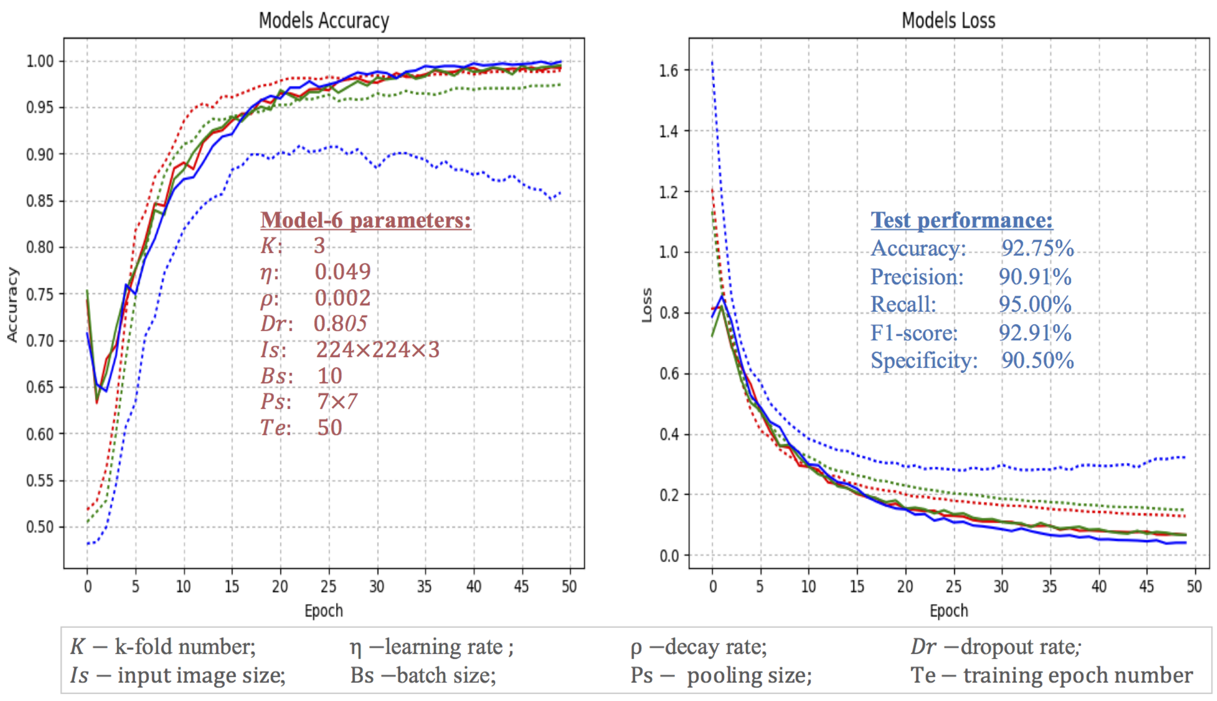
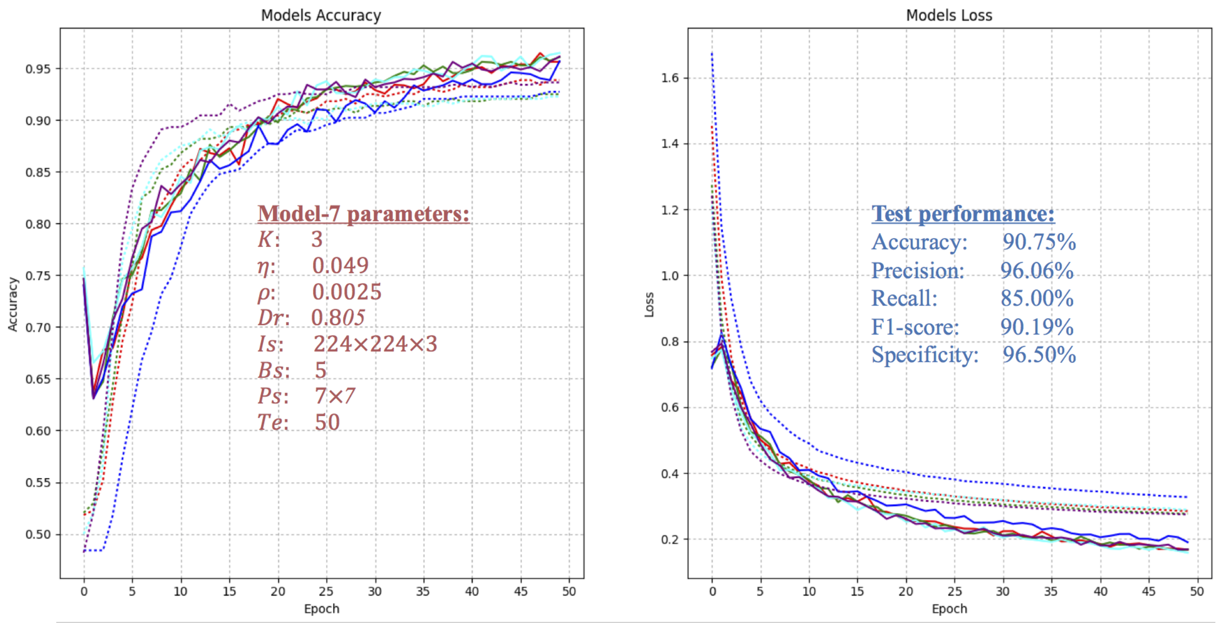
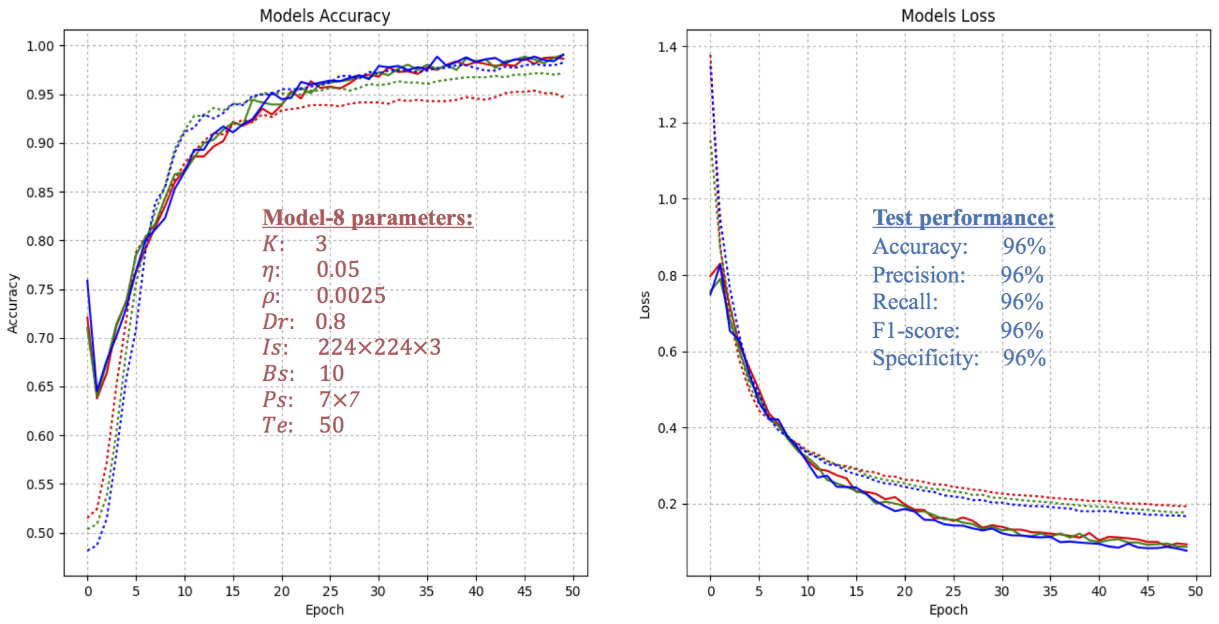


FIGURE 4.18: The learning curve of Model-6.



K — k-fold number;	η —learning rate ;	ρ —decay rate;	*Dr* —dropout rate;
Is — input image size;	Bs —batch size;	Ps — pooling size;	Te — training epoch number

FIGURE 4.19: The learning curve of Model-7.



K — k-fold number;	η —learning rate ;	ρ —decay rate;	*Dr* —dropout rate;
Is — input image size;	Bs —batch size;	Ps — pooling size;	Te — training epoch number

FIGURE 4.20: The learning curve of Model-8.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we investigated various techniques and solutions for automatic detection of polyps in endoscopic images. The goal of our study is to explore the use of the cutting-edge machine learning, computer vision and deep learning algorithms to achieve automated disease diagnosis.

We first studied and discussed work on topics related to the automatic polyp detection in colon images. We consider shape and texture-based classification (such as SVM, KNN, etc.) techniques as the conventional machine learning methods for distinguishing with deep learning based ones. For traditional ML-based techniques, we first provided an overview of machine learning approaches with a brief discussion of different learning types such as supervised and unsupervised learning and so on. Then we discussed different feature extraction and classification algorithms utilized for polyp detection tasks which covered shape and texture-color based methods. As for DL-based techniques, we first studied a set of state-of-the-art deep learning networks such as ALexNet, VGG Net, GoogLeNet, and ResNet which have demonstrated outstanding effectiveness in image classification domain which also can be applied into medical image processing pipelines. Subsequently CNN-based CAD systems along with pre-trained CNNs techniques were discussed.

Based on our literature review, we first proposed our three different schemes for automatic detection of colorectal polyps named ML-framework, DL-framework and TL-framework separately standing for machine learning, deep learning and transfer learning frameworks. We also provided a scalable CAD framework which consisted of 4 flexible modules based on the fusion of a set of state-of-the-art image processing algorithms in order to generalize and extend our work in future with versatile capabilities in the medical domain. automatic polyp detection. We then presented and analyzed various image preprocessing methods including histogram modification, noise filtering, data augmentation and dimension reduction etc. The next most important part of our work is related to the detailed design methodologies of deep neural networks that are also our major contributions. We analyzed the cutting edge techniques and algorithms that are all necessary to build a high effective deep learning network. That covered general neuron algorithm, feed-forward network, activation and loss functions with regularization approach, gradient descent optimization algorithms with the backpropagation process. And last we described the key techniques in detail for deep ConvNets that covered the convolution algorithm with stride and padding methods, different pooling techniques and dropout methodologies etc. Finally, we analyzed the

50-layer ResNet architecture that was the major deep learning model utilized in our transfer-learning framework.

In the implementation phase, we developed a set of software tools to extract patches from the ground truth CVC-ColonDB and enlarged the data set by automatic augmentation algorithms and finally made our patch-balanced dataset with sufficient size for our research and experiments. Meanwhile, we built 10-classifiers (Linear SVM, RBF SVM, KNN, RF,GP, SGD, MLP, Adaboost and Bayes) along with a set of low-level feature extractors (Histogram and a set of different filters) to evaluate the performance for detecting polyps by making use of these classifiers with low-level feature extractors. We then established the benchmarks from these experiments on our own dataset by using these conventional machine learning methods, which can be used later as a comparison base against DCNNs' performance.

Based on our extensively study and research on different cutting-edge DCNNs techniques, we successfully developed an effective transfer learning architecture which consists of a new FCNs classifier and input layer combined with a pre-trained 50-layer ResNet model. We implemented the proposed TL-framework by Python with Tensorflow and CUDA as backend to make the best use of the parallel computational power of GPUs.

DCNNs are very sensitive to the setting of their hyper-parameters. In our TL-framework, we provide 8 hyper parameters that include learn rate (η), decay(ρ), batch size (Bs), input size (Is), epoch number (Te), dropout rate (Dr), k-fold number (K), and pooling size (Ps). These hyper parameters make our system very flexible and scalable. However, fine-tuning the hyper parameters is a tricky process. Though there are some automatic fine-tuning approaches such as grid search, random search, or Bayesian optimization and TPE algorithms, etc. All these methods either are too costly and time-consuming or too difficult to apply in unique deep neural networks. Therefore, experimentation with hand-tuning is still the best approach till now for fine-tuning deep learning systems. In our work, we creatively made an high effective hand-tuning strategy with first establishing a rough range of each hyper parameter by conducting a set of quick experiments on a small sub-sampled training set, and then further fine-tuning each parameter on the whole dataset to determine a more accurate setting range. This unique hand-tuning methods saved us a lot of time to search and select the best and most suitable setting of the hyper-parameter to obtain better performance in terms of accuracy, precision, sensitivity and so on.

We finally achieved overall 96.00% detection accuracy and precision, 96.00% sensitivity and specificity, and 96.00% f1-score by using the proposed TL framework with our optimized hyper-parameters, which outperformed the traditional machine learning classification methods in each defined performance metric. Moreover, the TL framework proposed is scalable and flexible so that it can easily be extended to include other types of disease detection in future.

5.2 Future work

Though this work contains an extensive evaluation of the hyper-parameters, due to the limitation of hardware and the time constraints, our experiments are still too limited to achieve the best optimization results, so future work should also focus on examining wider ranges of the hyper-parameters. The most interesting direction would be to explore automatic hyper-parameter optimization methods to replace manually tuning hyper-parameter which was highly necessary for future work.

We have only used colonoscopy images for this thesis to evaluate our methods. It is therefore necessary to further collect a greater number of capsule endoscopy images or other larger medical datasets, in order to re-evaluate, qualify and respectively confirm the results of this work, and then we could further optimize the proposed framework to be able to detect all different polyp morphological types in future.

Our work unveils another interesting direction for future work, from a practical point of view, to combine pre-trained DCNNs with traditional classifiers. For instance, to train SVM or Random Forest classifier by the low-level features learned from pre-trained DCNNs models could be possible to achieve better classification accuracy.

In addition, though the proposed TL framework allows the use of more than one pre-trained model, we only tested the ResNet50 model in this work, so it would be valuable to further add some other cutting-edge pre-trained models such as Google Inception, to achieve better performance and further boost its generalizing capabilities in future.

Finally, we layout a generalized scalable framework for computer-aided diagnosis systems in which fusion of a set of cutting-edge machine learning algorithms and deep learning techniques are employed together to boost CAD system's performance and robustness. This proposed framework in the chapter of methodology shows us an interesting and feasible direction for making versatile CAD systems in future by reproducing, generalizing and extending our current work on automatic polyp detection systems.

Appendix A

Required toolkits and libraries

Based on the specific requirements and time constraints of our project, we chose to use Python programming language and the below toolkits and libraries in this work.

Python:

Python is an interpreted, object-oriented, high-level programming language which is developed under an OSI-approved open source license, making it freely usable and distributable. And its high-level built in data structures, dynamic typing and dynamic binding, make it very attractive for rapid prototyping and application development especially in the big data and deep learning domain. For more information, please refer to Python.org.

CUDA and cuDNN:

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by NVIDIA and implemented by the GPUs that they produce. The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. Please refer to NVIDIA.cuDNN.

TensorFlow:

TensorFlow [2] is an open source Python library for fast numerical computing created and released by Google and released under the Apache 2.0 open source license. It is a foundation library that can be used to create Deep Learning models directly or by using other wrapper libraries like Keras that simplify the process built on top of TensorFlow. It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines. Please refer to Tensorflow.org.

Keras:

Keras is an open source API written in Python which uses as backend either Theano or Tensorflow. It was developed with a focus on enabling fast experimentation, so that it is easier to build complete solutions, and is easy to read with the greatest selection of state-of-the-art algorithms (optimizers, normalization routines, activation functions). Please refer to Keras.io.

OpenCV:

OpenCV is a famous open source computer vision library. It is free for both commercial and research use under a BSD license. The library is cross-platform, and runs on Windows, Linux, Mac OS X, mobile Android and iOS with support of C/C++, Python and Java interfaces. The library itself is written in C/C++, but Python bindings are provided when running the installer. We utilized OpenCV 3.0 in our application. For more details, please refer to OpenCV.org.

Scikit-learn:

Scikit-learn is an open source library built on Numpy, Scipy and Matplotlib. It is developed by a large community of developers and machine learning experts. Scikit-learn provides a set of tools for many of the standard machine-learning tasks (such as clustering, classification, regression, etc.). It can be commercially usable under BSD license. For more details, please refer to Scikit-learn.org

Others:

There are also some other open source APIs utilized in our applications that include **NumPy**, **SciPy**, **Matplotlib**, **Pandas**, **H5py**, **QtPy**, etc. We will not present them here in more detail, since it is convenient to get these resources on-line.

Appendix B

Code snippets for implementation

B.1 ResNet50 model

The code snippets for implementing ResNet50 model are presented in Listing B.1, B.2, and B.3. We will not discuss these functions in more detail, however, it is worth noting that the new top layer is created from line-51 to line-58 in Listing B.1, and the short snippet in Listing B.2 illustrates the definition of `identity_block` function which is to build the blocks without convolutional layers in ResNet. Accordingly, the function `conv_block` presented in Listing B.3 is to build the blocks of convolutional layers.

```
1  def resnet50_model(img_rows, img_cols, color_type=3, num_class=3,
2                     lr=0.05, dec=0.0027,dr=0.8, ps = (3,3)):
3      """
4      Resnet Model for Keras
5      Model Schema is based on
6      https://github.com/fchollet/deep-learning-models/blob/master/
   resnet50.py
7      ImageNet Pretrained Weights
8      https://github.com/fchollet/deep-learning-models/releases/download/
   v0.2/resnet50_weights_th_dim_ordering_th_kernels.h5
9      Parameters:
10       img_rows, img_cols - resolution of inputs
11       channel - 1 for grayscale, 3 for color
12       num_class - number of class labels for our classification task
13     """
14
15     bn_axis = 3
16
17     img_input = Input(shape=(img_rows, img_cols, color_type))
18     x = ZeroPadding2D((3, 3))(img_input)
19     x = Convolution2D(64, 7, 7, subsample=(2, 2), name='conv1')(x)
20     x = BatchNormalization(axis=bn_axis, name='bn_conv1')(x)
21     x = Activation('relu')(x)
22     x = MaxPooling2D((3, 3), strides=(2, 2))(x)  # dim_ordering='th'
23
24     x = conv_block(x, 3, [64, 64, 256], stage=2, block='a', strides=(1,
   1))
25     x = identity_block(x, 3, [64, 64, 256], stage=2, block='b')
26     x = identity_block(x, 3, [64, 64, 256], stage=2, block='c')
27
28     x = conv_block(x, 3, [128, 128, 512], stage=3, block='a')
29     x = identity_block(x, 3, [128, 128, 512], stage=3, block='b')
30     x = identity_block(x, 3, [128, 128, 512], stage=3, block='c')
31     x = identity_block(x, 3, [128, 128, 512], stage=3, block='d')
```

```

    x = conv_block(x, 3, [256, 256, 1024], stage=4, block='a')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='b')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='c')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='d')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='e')
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block='f')

    x = conv_block(x, 3, [512, 512, 2048], stage=5, block='a')
    x = identity_block(x, 3, [512, 512, 2048], stage=5, block='b')
    x = identity_block(x, 3, [512, 512, 2048], stage=5, block='c')

    model = Model(img_input, x)

    # Load ImageNet pre-trained data
    weights_path = './model/
    resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'
    model.load_weights(weights_path)

    # Truncate and replace softmax layer for transfer learning
    x = AveragePooling2D(pool_size=ps, name='avg_pool')(x)
    x_newfc = Flatten(input_shape=model.output_shape[1:])(x)
    x_newfc = Dense(512, activation='relu')(x_newfc)
    x_newfc = Dropout(dr)(x_newfc)
    x_newfc = Dense(num_class, activation='softmax', name='fc10')(
    x_newfc)

    # Create another model with our customized softmax
    model = Model(img_input, x_newfc)
    # model.summary()
    adade = Adadelta(lr=lr, decay=dec)
    model.compile(optimizer=adade,
                  loss='categorical_crossentropy',
                  metrics=['accuracy',
                           metrics.mse,
                           metrics.precision,
                           metrics.recall,
                           metrics.f1score]
                  )

    return model
```

LISTING B.1: Create ResNet50 model with a customized top layer for transfer learning.

```
def identity_block(input_tensor, kernel_size, filters, stage, block):
    """
    The identity_block is the block that has no conv layer at shortcut
    Arguments
        input_tensor: input tensor
        kernel_size: defualt 3, the kernel size of middle conv layer at
    main path
        filters: list of integers, the nb_filters of 3 conv layer at
    main path
        stage: integer, current stage label, used for generating layer
    names
        block: 'a','b'..., current block label, used for generating
    layer names
    """

    nb_filter1, nb_filter2, nb_filter3 = filters
    bn_axis = 3
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = Convolution2D(nb_filter1, 1, 1, name=conv_name_base + '2a')(
   input_tensor)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
    x = Activation('relu')(x)

    x = Convolution2D(nb_filter2, kernel_size, kernel_size,
                      border_mode='same', name=conv_name_base + '2b')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2b')(x)
    x = Activation('relu')(x)

    x = Convolution2D(nb_filter3, 1, 1, name=conv_name_base + '2c')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2c')(x)

    x = merge([x, input_tensor], mode='sum')
    x = Activation('relu')(x)
    return x
```

LISTING B.2: Build non-convolutional layer blocks of RestNet.

```
def conv_block(input_tensor, kernel_size, filters, stage, block, strides
   =(2, 2)):
    """
    conv_block is the block that has a conv layer at shortcut
    # Arguments
        input_tensor: input tensor
        kernel_size: defualt 3, the kernel size of middle conv layer at
   main path
        filters: list of integers, the nb_filters of 3 conv layer at
   main path
        stage: integer, current stage label, used for generating layer
   names
        block: 'a','b'..., current block label, used for generating
   layer names
    Note that from stage 3, the first conv layer at main path is with
   subsample=(2,2)
    And the shortcut should have subsample=(2,2) as well
    """

    nb_filter1, nb_filter2, nb_filter3 = filters
    bn_axis = 3
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = Convolution2D(nb_filter1, 1, 1, subsample=strides,
                      name=conv_name_base + '2a')(input_tensor)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
    x = Activation('relu')(x)

    x = Convolution2D(nb_filter2, kernel_size, kernel_size, border_mode=
   'same',
                      name=conv_name_base + '2b')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2b')(x)
    x = Activation('relu')(x)

    x = Convolution2D(nb_filter3, 1, 1, name=conv_name_base + '2c')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2c')(x)

    shortcut = Convolution2D(nb_filter3, 1, 1, subsample=strides,
                             name=conv_name_base + '1')(input_tensor)
    shortcut = BatchNormalization(axis=bn_axis, name=bn_name_base + '1')
   (shortcut)

    x = merge([x, shortcut], mode='sum')
    x = Activation('relu')(x)
    return x
```

LISTING B.3: Build convolutional layer blocks of ResNet.

B.2 Feature extraction and classifier

The below code snippets presented in Listing B.4 are used to implement low-level histogram feature extraction to train the proposed classifiers.

```
import numpy as np
import argparse
import imutils
import cv2
import os


def extract_color_histogram(image, bins=(8, 8, 8)):
    # extract a 3D color histogram from the HSV color space using
    # the supplied number of ‘bins‘ per channel
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([hsv], [0, 1, 2], None, bins,
                        [0, 180, 0, 256, 0, 256])

    # handle normalizing the histogram if we are using OpenCV 2.4.X
    if imutils.is_cv2():
        hist = cv2.normalize(hist)

    # otherwise, perform "in place" normalization in OpenCV 3
    else:
        cv2.normalize(hist, hist)

    # return the flattened histogram as the feature vector
    return hist.flatten()
```

LISTING B.4: Histogram features extraction and normalization.

The code snippet in Listing B.5 is to implement the comparison experiments on the 10 suggested classifiers.

```
# import the necessary packages
from sklearn.model_selection import train_test_split
```

```python
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import LinearSVC, SVC
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from imutils import paths

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-t", "--train", type=str, default='polyp2/train',
                help="path to input dataset")
ap.add_argument("-v", "--test", type=str, default='polyp2/test',
                help="path to input dataset")
args = vars(ap.parse_args())

# grab the list of images that we'll be describing
print("Describing images...")
imageTrainPaths = list(paths.list_images(args["train"]))
imageTestPaths = list(paths.list_images(args["test"]))

# initialize the data matrix and labels list
data = []
labels = []

data_test = []
labels_test = []

# loop over the input train images
for (i, imagePath) in enumerate(imageTrainPaths):
    # load the image and extract the class label
    image = cv2.imread(imagePath)
    label = imagePath.split(os.path.sep)[-2]

    # extract a color histogram from the image, then update the
    # data matrix and labels list
    hist = extract_color_histogram(image)
    data.append(hist)
    labels.append(label)

    # show an update every 1,000 images
    if i > 0 and i % 100 == 0:
        print("Processed {}/{}".format(i, len(imageTrainPaths)))

# loop over the input train images
for (i, imagePath) in enumerate(imageTestPaths):
    # load the image and extract the class label
    image = cv2.imread(imagePath)
    label = imagePath.split(os.path.sep)[-2]

    # extract a color histogram from the image, then update the
    # data matrix and labels list
    hist = extract_color_histogram(image)
    data_test.append(hist)
    labels_test.append(label)

    # show an update every 1,000 images
    if i > 0 and i % 100 == 0:
        print("Processed {}/{}".format(i, len(imageTestPaths)))

# encode the labels, converting them from strings to integers
le = LabelEncoder()
labels = le.fit_transform(labels)
labels_test = le.fit_transform(labels_test)

# partition the data into training and testing splits, using 75%
# of the data for training and the remaining 25% for testing
#print("[INFO] constructing training/testing split...")
#(trainData, testData, trainLabels, testLabels) = train_test_split(
#    np.array(data), labels, test_size=0.25, random_state=42)

X_train = np.array(data)
y_train = labels
X_test = np.array(data_test)
y_test = labels_test

names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "SGDClassifier",
         "Gaussian Process",
         "Decision Tree", "Random Forest", "MLPClassifier", "AdaBoost",
         "Naive Bayes"]

classifiers = [
    KNeighborsClassifier(59),
    LinearSVC(),
    SVC(kernel='poly',C=0.1,gamma=0.01,degree=3),
    SGDClassifier(loss="log", n_iter=10),
    GaussianProcessClassifier(1.0 * RBF(1.0), warm_start=True),
    DecisionTreeClassifier(max_depth=15),
    RandomForestClassifier(n_estimators=100, max_features='sqrt'),
    MLPClassifier(alpha=1),
    AdaBoostClassifier(learning_rate=0.1),
    GaussianNB()]

# closs_validation accuracy experiments
results = {}
for name, clf in zip(names, classifiers):
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    results[name] = scores

for name, scores in results.items():
    print("%20s | Accuracy: %0.2f%% (+/- %0.2f%%)" % (name, 100 * scores
    .mean(), 100 * scores.std() * 2))

# iterate over classifiers by using fixed additional validation data
for name, model in zip(names, classifiers):
    print("Training and evaluating classifier {}".format(name))
    model.fit(X_train, y_train)

    predictions = model.predict(X_test)
    print(classification_report(y_test, predictions, target_names=le.
    classes_))
```

LISTING B.5: Classifiers comparison experiments.

B.3 Data augmentation

The below code snippet presented in Listing B.6 are used for image data augmentation by random rotation, flips etc.

```
# image datasets extended by image data generator
# should generate one type/class by one type/class
# the folder structure as blow
# put only one type in the train folder for one time.
# the code need reflector in future

"""directory structure:
'''
dataset/
    train/
        Type_1/
            001.jpg
            002.jpg
            ...
"""

from keras.preprocessing.image import ImageDataGenerator

img_dir = '/Users/liuqh/Desktop/dataset/train'
sav_dir = '/Users/liuqh/Desktop/new'

datagen = ImageDataGenerator(
    rotation_range = 90,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)

i = 1
for batch in datagen.flow_from_directory(img_dir,
                                          target_size=(224,224),
                                          shuffle=False,
                                          batch_size= 100,
                                          save_prefix='_gen',
                                          save_to_dir=sav_dir):
    i += 1
    if i > 66:
        break
```

LISTING B.6: Image data augmentation.

B.4 Batch-image resize

The code snippet in List B.7 is used for batch resizing images.

```python
# Image pre-process tool - automatic batch-resize images
# This scrip should be put in the same folder of images
# after run, resized impages will be saved into a new created folder
# /resized/..image.jpg
# change size prarameter according to your needs, the default settings are
# 512x512
from PIL import Image
import os, sys

# for python3, needed define cmp function,
# for python2.7, don't need define it, can remove it.
def cmp(a,b):
    return (a > b) - (a < b)

def resizeImage(infile, output_dir="resized/", size=(512,512)):
     outfile = os.path.splitext(infile)[0]+"_resized"
     extension = os.path.splitext(infile)[1]

     if (cmp(extension, ".jpg")):
        return

     if infile != outfile:
        try :
            im = Image.open(infile)
            im.thumbnail(size, Image.ANTIALIAS)
            im.save(output_dir+outfile+extension,"JPEG")
        except IOError:
            print ("cannot reduce image for {}".format(infile))


if __name__=="__main__":
    output_dir = "resized"
    dir = os.getcwd()

    if not os.path.exists(os.path.join(dir,output_dir)):
        os.mkdir(output_dir)

    for file in os.listdir(dir):
        resizeImage(file)
```

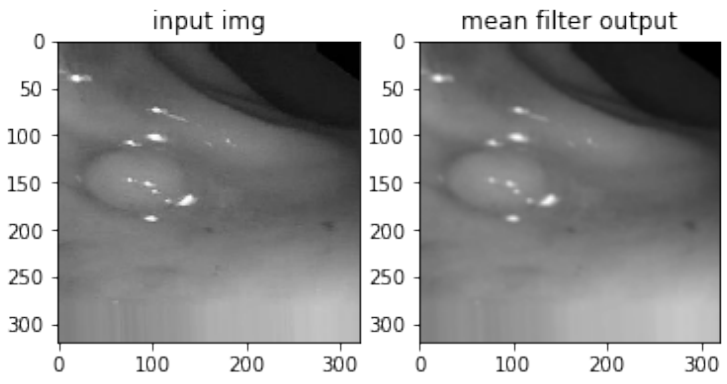
LISTING B.7: Image batch resizing function .

B.5 Low-level image process test

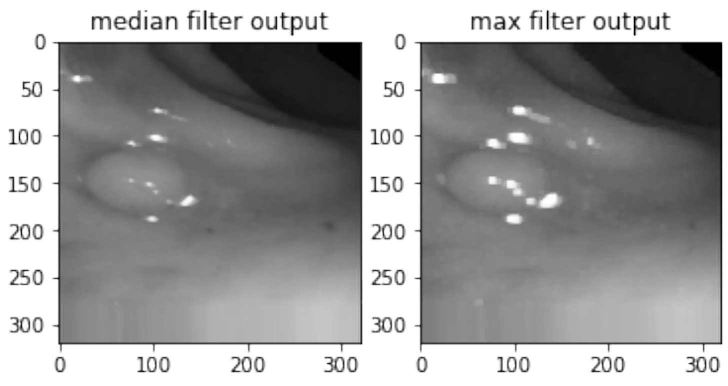
Low-level-image-process-test

```
In [1]: # Author: Qinghui Liu, for my master thesis project,
        # Date: 2017-03

        # use the belwo command convert the file to latex
        # $: jupyter nbconvert /path/to/mynotebook.ipynb --to latex
        # basic image processing and analysis by python

        # ####1. Spatial Filters - linear filters and non-linear filters ####
        # linear ones include mean, laplacian, and laplacian of gaussian
        # non-linear ones include median, maximum, minimum, sobel, prewitt and cann
        # Four padding approaches: Zero padding, constant, nearest neighbor and re
        # ---mean filter
        import numpy as np
        import skimage
        from skimage import feature
        import skimage.io as sio
        import scipy.misc
        import scipy.ndimage as sn
        from scipy.misc.pilutil import Image
        from matplotlib import pyplot
        #import matplotlib.pyplot as plt

        # plotting inline in Jupter Notebook
        %matplotlib inline
        #matplotlib.rcParams['font.size'] = 8

        # open image and convert it to grayscale
        img_dir = '/Users/liuqh/Desktop/keras/data/train/polyp/_0_1452.jpeg'
        #a = Image.open(img_dir).convert('L')
        a = sio.imread(img_dir)
        a = skimage.color.rgb2gray(a)
        pyplot.subplot(1,2,1)
        pyplot.title('input img')
        pyplot.imshow(a,cmap='gray')

        # initializeing the filter of size 5x5
        # divided by 25 for normalization
```

```
k = np.ones((5,5))/25

# perform convolution
b = sn.filters.convolve(a,k)

pyplot.subplot(1,2,2)
pyplot.imshow(b,cmap='gray') # try cmap = 'bone_r' or other parameters
pyplot.title('mean filter output')
pyplot.show()
# convert ndarray to an image
b = scipy.misc.toimage(b)
b.save('mean_output.png')
```

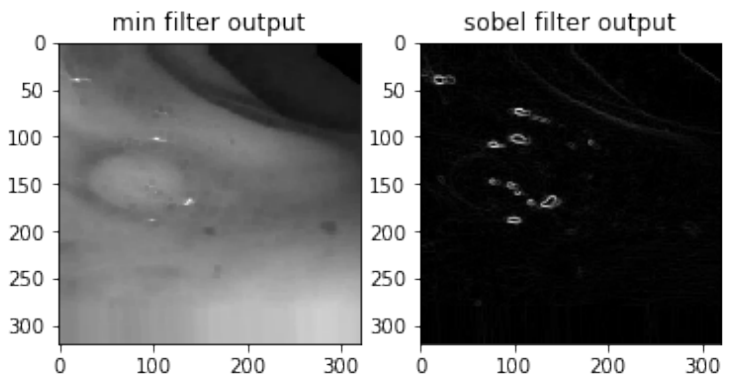


```
In [2]: ### Median Filter
        # median filter - one popular non-linear filter
        b_median = scipy.ndimage.filters.median_filter(a, size=5, footprint=None, c
                                                       mode ='reflect', cval=0.0, or

        b_median = scipy.misc.toimage(b_median)
        pyplot.subplot(1,2,1)
        pyplot.imshow(b_median,'gray')
        pyplot.title('median filter output')

        b_median.save('b_median.png')

        ### Max Filter
        # this filter enhances the bright points

        b_max = sn.filters.maximum_filter(a, size=5,
                                          footprint=None,
                                          output=None,
                                          mode ='reflect',
                                          cval=0.0, origin=0)
        b_max = scipy.misc.toimage(b_max)
        pyplot.subplot(1,2,2)
        pyplot.imshow(b_max,'gray')
        pyplot.title('max filter output')
        b_max.save('b_max.png')
```

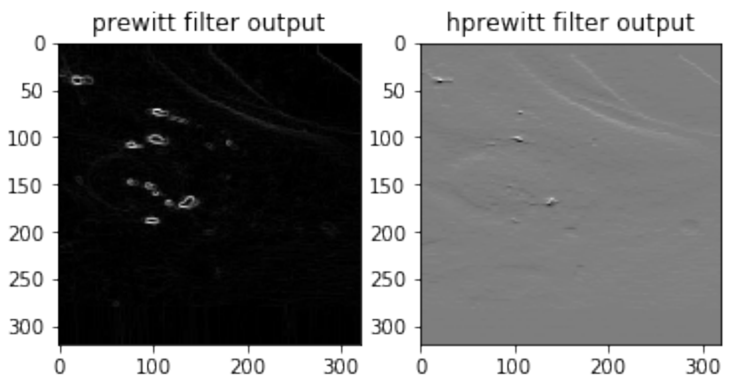


```
In [3]: ### Min Filter
        # this filter enhances the darkest points

        b_min = sn.filters.minimum_filter(a, size=5,
                                          footprint=None,
                                          output=None,
                                          mode ='reflect',
                                          cval=0.0, origin=0)
        b_min = scipy.misc.toimage(b_min)
        pyplot.subplot(1,2,1)
        pyplot.imshow(b_min,'gray')
        pyplot.title('min filter output')
        b_min.save('b_min.png')

        ### Edge detection
        # Sobel, and Prewitt filters are used to enchance all edges
        # horizontal or vertical - sobel or prewitt just enhance all vertical or ho

        from skimage import filter
        b_edge = filter.sobel(a) # try to use sobel_v(a) or sobel_h(a)
        pyplot.subplot(1,2,2)
        pyplot.imshow(b_edge,'gray')
        pyplot.title('sobel filter output')

        b_edge = scipy.misc.toimage(b_edge)
        b_edge.save('b_edge.png')
```

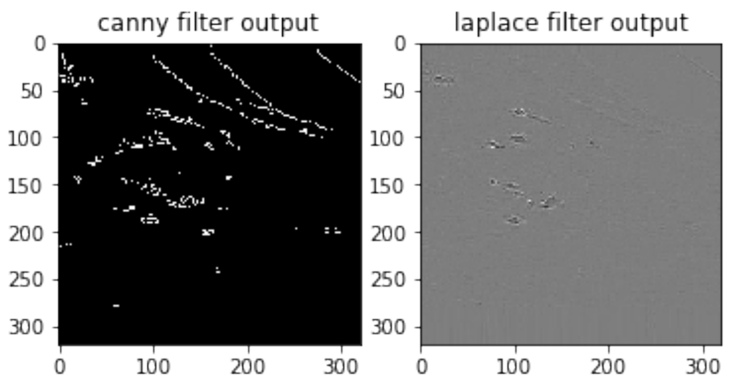
```
/usr/local/Cellar/anaconda2/lib/python2.7/site-packages/skimage/filter/__init__.py:
  warn(skimage_deprecation('The `skimage.filter` module has been renamed '
```



```
In [4]: # prewitt and hprewitt filters

        b_prewitt = filter.prewitt(a,mask = None)
        pyplot.subplot(1,2,1)
        pyplot.imshow(b_prewitt,'gray')
        pyplot.title('prewitt filter output')

        b_prewitt = scipy.misc.toimage(b_prewitt)
        b_prewitt.save('b_prewitt.png')

        b_hprewitt = filter.prewitt_h(b_min,mask = None)
        pyplot.subplot(1,2,2)
        pyplot.imshow(b_hprewitt,'gray')
        pyplot.title('hprewitt filter output')

        b_hprewitt = scipy.misc.toimage(b_hprewitt)
        b_hprewitt.save('b_hprewitt.png')
```



```
In [5]: # canny and laplace filters
        b_canny = feature.canny(a, sigma=0.1)
        pyplot.subplot(1,2,1)
        pyplot.imshow(b_canny,'gray')
        pyplot.title('canny filter output')

        b_canny = scipy.misc.toimage(b_canny)
        b_canny.save('b_canny.png')

        #b_laplace = skimage.filters.laplace(a,ksize = 3)
        b_laplace = sn.filters.laplace(a,mode='reflect')
        pyplot.subplot(1,2,2)
        pyplot.imshow(b_laplace,'gray')
        pyplot.title('laplace filter output')

        b_laplace = scipy.misc.toimage(b_laplace)
        b_laplace.save('b_laplace.png')
```

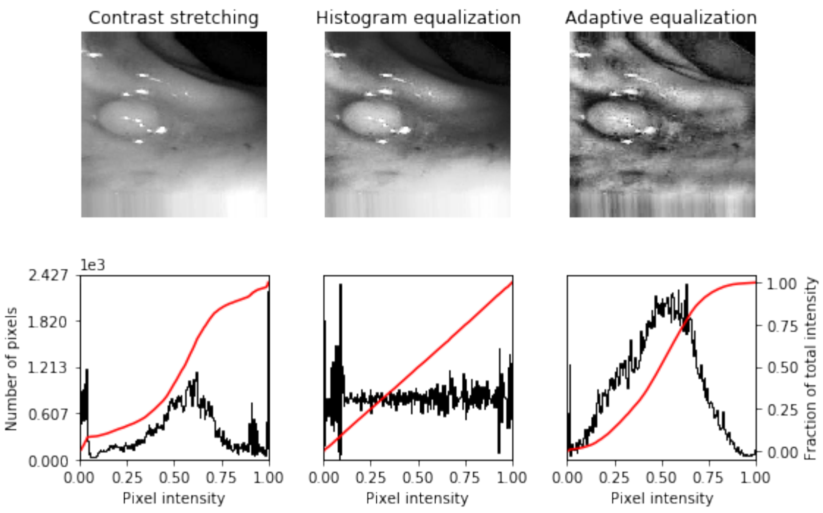


```
In [6]: # Histogram Equalization
        # refer to link: http://scikit-image.org/docs/dev/auto_examples

        from skimage import data, img_as_float
        from skimage import exposure

        def plot_img_and_hist(img, axes, bins=256):
            """Plot an image along with its histogram and cumulative histogram.

            """
            img = img_as_float(img)
            ax_img, ax_hist = axes
            ax_cdf = ax_hist.twinx()

            # Display image
            ax_img.imshow(img, cmap='gray')
            ax_img.set_axis_off()
            ax_img.set_adjustable('box-forced')

            # Display histogram
            ax_hist.hist(img.ravel(), bins=bins, histtype='step', color='black')
            ax_hist.ticklabel_format(axis='y', style='scientific', scilimits=(0, 0)
            ax_hist.set_xlabel('Pixel intensity')
            ax_hist.set_xlim(0, 1)
            ax_hist.set_yticks([])

            # Display cumulative distribution
            img_cdf, bins = exposure.cumulative_distribution(img, bins)
            ax_cdf.plot(bins, img_cdf, 'r')
            ax_cdf.set_yticks([])

            return ax_img, ax_hist, ax_cdf
```

```
In [7]: img = a

        # Contrast stretching
        p2, p98 = np.percentile(img, (2, 98))
        img_rescale = exposure.rescale_intensity(img, in_range=(p2, p98))

        # Equalization
        img_eq = exposure.equalize_hist(img)

        # Adaptive Equalization
        img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
```

```
/usr/local/Cellar/anaconda2/lib/python2.7/site-packages/skimage/util/dtype.py:110:
  "%s to %s" % (dtypeobj_in, dtypeobj))
```

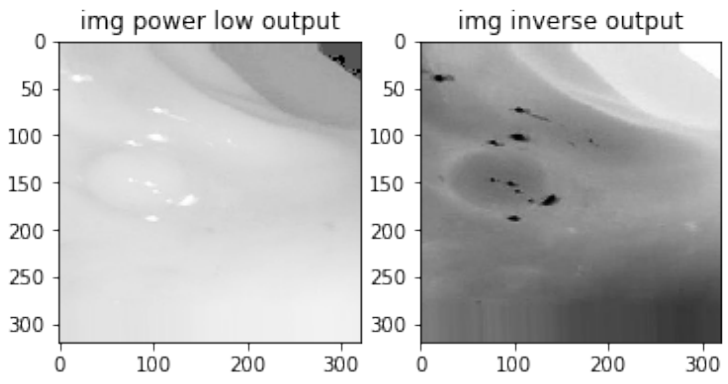
```
In [8]: # Display results
        fig = pyplot.figure(figsize=(8, 5))
        axes = np.zeros((2, 3), dtype=np.object)
        axes[0, 0] = fig.add_subplot(2, 3, 1)
        for i in range(1, 3):
            axes[0, i] = fig.add_subplot(2, 3, 1+i, sharex=axes[0,0], sharey=axes[
        for i in range(0, 3):
            axes[1, i] = fig.add_subplot(2, 3, 4+i)

        #ax_img, ax_hist, ax_cdf = plot_img_and_hist(img, axes[:, 0])
        #ax_img.set_title('Low contrast image')

        ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_rescale, axes[:, 0])
        y_min, y_max = ax_hist.get_ylim()
        ax_img.set_title('Contrast stretching')

        ax_hist.set_ylabel('Number of pixels')
        ax_hist.set_yticks(np.linspace(0, y_max, 5))

        ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_eq, axes[:, 1])
        ax_img.set_title('Histogram equalization')

        ax_img, ax_hist, ax_cdf = plot_img_and_hist(img_adapteq, axes[:, 2])
        ax_img.set_title('Adaptive equalization')

        ax_cdf.set_ylabel('Fraction of total intensity')
        ax_cdf.set_yticks(np.linspace(0, 1, 5))
        # prevent overlap of y-axis labels
        fig.tight_layout()
        pyplot.show()
```



```
In [9]: # Power law transformation
        # t(i,j) = kI(i,j)^r
        import math, numpy
        im = Image.open(img_dir).convert('L')
        imp = scipy.misc.fromimage(im)
        gamma = 0.2 # try to use other numbers to text 0.5, 1, 2, 5, etc
        imp1 = imp.astype(float)
        imp3 = numpy.max(imp1)
        imp2 = imp1/imp3
        # compute gamma-correction
        imp3 = numpy.log(imp2)*gamma
        # perform gamma-correction
        c = numpy.exp(imp3)*255.0
        # convert c to type int
        c1 = c.astype(int)
        # convert c1 from ndarray to image
        im_pl = scipy.misc.toimage(c1)
        im_pl.save('b_powerlaw.png')
```

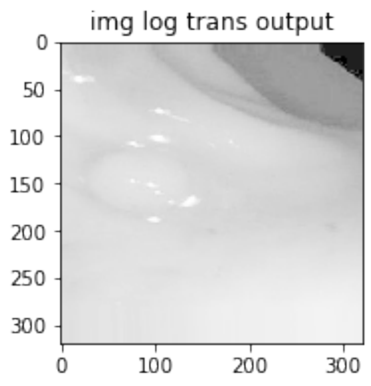
```
/usr/local/Cellar/anaconda2/lib/python2.7/site-packages/ipykernel/__main__.py:11: F
```

```
In [10]: ## image inverse transfomation
         # t(i,j) = L - 1 -I(i,j),  transfrom dark intensities to bright intensitie
         # vice versa
         im2 = 255-imp
         im3 = scipy.misc.toimage(im2)
         im3.save('b_invers.png')

         pyplot.subplot(1,2,1)
         pyplot.imshow(im_pl,'gray')
         pyplot.title('img power low output')

         pyplot.subplot(1,2,2)
         pyplot.imshow(im3,'gray')
         pyplot.title('img inverse output')

Out[10]: <matplotlib.text.Text at 0x1111d62d0>
```



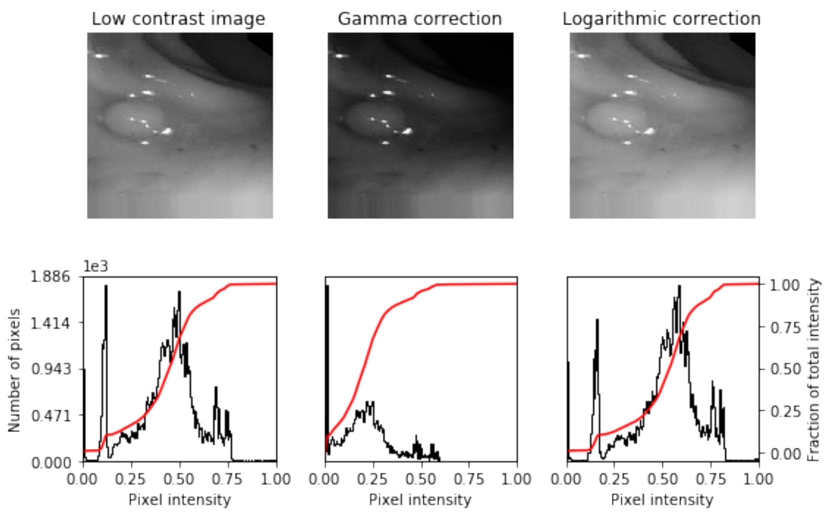
```
In [11]: # log transformation
         # t(i,j) = k* log(1+I(i,j)) where
         # k = (L-1)/log(1+|I_max|), I_max is maximum magnitude

         p1=imp1.astype(float)
         p2=numpy.max(p1)
         c=(255.0*numpy.log(1+p1))/numpy.log(1+p2)
         c1=c.astype(int)
         im_log = scipy.misc.toimage(c1)
         im_log.save('b_logTrans.png')

         pyplot.subplot(1,2,2)
         pyplot.imshow(im_log,'gray')
         pyplot.title('img log trans output')
```

```
Out[11]: <matplotlib.text.Text at 0x1123bcb90>
```



```
In [12]: # Gamma and log contrast adjustment

         # Gamma
         gamma_corrected = exposure.adjust_gamma(img, 2)

         # Logarithmic
         logarithmic_corrected = exposure.adjust_log(img, 1)

In [13]: # Display results
         fig = pyplot.figure(figsize=(8, 5))
         axes = np.zeros((2, 3), dtype=np.object)
         axes[0, 0] = pyplot.subplot(2, 3, 1, adjustable='box-forced')
         axes[0, 1] = pyplot.subplot(2, 3, 2, sharex=axes[0, 0], sharey=axes[0, 0],
                                  adjustable='box-forced')
         axes[0, 2] = pyplot.subplot(2, 3, 3, sharex=axes[0, 0], sharey=axes[0, 0],
                                  adjustable='box-forced')
         axes[1, 0] = pyplot.subplot(2, 3, 4)
         axes[1, 1] = pyplot.subplot(2, 3, 5)
         axes[1, 2] = pyplot.subplot(2, 3, 6)

         ax_img, ax_hist, ax_cdf = plot_img_and_hist(img, axes[:, 0])
         ax_img.set_title('Low contrast image')

         y_min, y_max = ax_hist.get_ylim()
         ax_hist.set_ylabel('Number of pixels')
         ax_hist.set_yticks(np.linspace(0, y_max, 5))

         ax_img, ax_hist, ax_cdf = plot_img_and_hist(gamma_corrected, axes[:, 1])
```

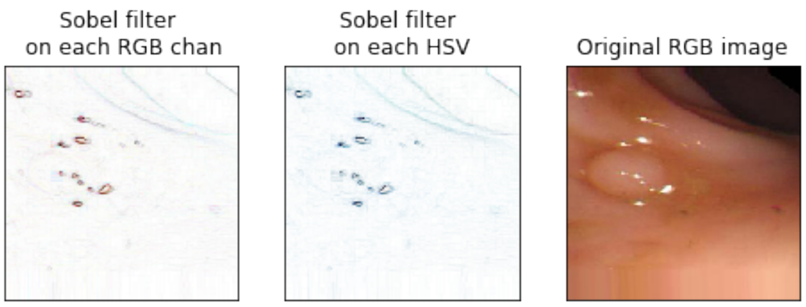
```
ax_img.set_title('Gamma correction')

ax_img, ax_hist, ax_cdf = plot_img_and_hist(logarithmic_corrected, axes[:,
ax_img.set_title('Logarithmic correction')

ax_cdf.set_ylabel('Fraction of total intensity')
ax_cdf.set_yticks(np.linspace(0, 1, 5))

# prevent overlap of y-axis labels
fig.tight_layout()
pyplot.show()
```



```
In [24]: # adapting gray-scale filters to RGB images
         # each_channel, pass each of RGB channels to the filter
         # hsv_vaule, convert RGB to HSV and pass the value channel to the filter
         # the result is inserted back to HSV and then converted back to RGB

         from skimage.color.adapt_rgb import adapt_rgb, each_channel, hsv_value
         from skimage import filters
         from skimage.exposure import rescale_intensity


         @adapt_rgb(each_channel)
         def sobel_each(image):
             return filters.sobel(image)
```

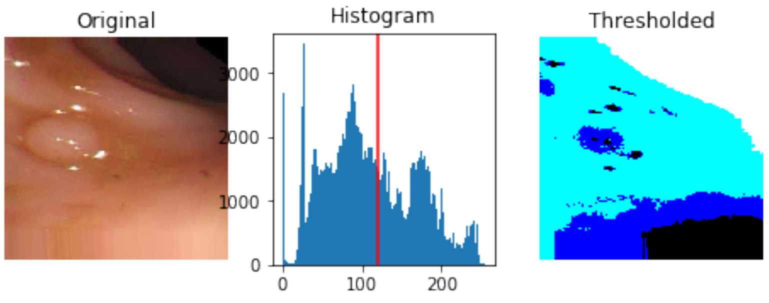
```
@adapt_rgb(hsv_value)
def sobel_hsv(image):
    return filters.sobel(image)

image = sio.imread(img_dir) # rgb imaaage
fig = pyplot.figure(figsize=(8, 4))
ax_each = fig.add_subplot(131, adjustable='box-forced')
ax_hsv = fig.add_subplot(132, sharex=ax_each, sharey=ax_each,
                         adjustable='box-forced')
ax_orig = fig.add_subplot(133, adjustable='box-forced')
# We use 1 - sobel_each(image)
# but this will not work if image is not normalized
ax_each.imshow(rescale_intensity(1 - sobel_each(image)))
ax_each.set_xticks([]), ax_each.set_yticks([])
ax_each.set_title("Sobel filter \n on each RGB chan")

# We use 1 - sobel_hsv(image) but this will not work if image is not norma
ax_hsv.imshow(rescale_intensity(1 - sobel_hsv(image)))
ax_hsv.set_xticks([]), ax_hsv.set_yticks([])
ax_hsv.set_title("Sobel filter \n on each HSV ")

ax_orig.imshow(image)
ax_orig.set_xticks([]), ax_orig.set_yticks([])
ax_orig.set_title("Original RGB image")
```

Out[24]:
```
<matplotlib.text.Text at 0x115857690>
```



In [25]:
```
## Thresholding
# create a binary image from a grayscale image
```

```
from skimage.filters import threshold_otsu,threshold_isodata, threshold_li
#image = skimage.color.rgb2gray(image)
#image= img_eq#.astype(float)
thresh = threshold_otsu(image)
#thresh = threshold_li(image)
#thresh = threshold_isodata(image)

binary = image < thresh # try to test other thresholds

fig, axes = pyplot.subplots(ncols=3,figsize=(8, 2.5))
ax = axes.ravel()
ax[0] = pyplot.subplot(1, 3, 1, adjustable='box-forced')
ax[1] = pyplot.subplot(1, 3, 2)
ax[2] = pyplot.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0], adjustable='bo

ax[0].imshow(image, cmap=pyplot.cm.gray)
ax[0].set_title('Original')
ax[0].axis('off')

ax[1].hist(image.ravel(), bins=256)
ax[1].set_title('Histogram')
ax[1].axvline(thresh, color='r')

# binary.astype(np.uint8), np.float, np.uint16
ax[2].imshow(binary.astype(np.float), cmap=pyplot.cm.gray)
ax[2].set_title('Thresholded')
ax[2].axis('off')

pyplot.show()
```



```
In [ ]:
```

Bibliography

[1] Web Page. 2016. URL: http://www.cancer.org/research/cancerfactsstatistics/cancerfactsfigures2016/.

[2] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[3] O. Abdel-Hamid et al. "Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4277–4280. ISBN: 1520-6149. DOI: 10.1109/ICASSP.2012.6288864.

[4] Yuichiro Anzai. *Pattern recognition and machine learning*. Elsevier, 2012.

[5] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[6] James S Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554.

[7] J. Bernal, J. Sánchez, and F. Vilariño. "Towards automatic polyp detection with a polyp appearance model". In: *Pattern Recognition* 45.9 (2012), pp. 3166–3182. ISSN: 0031-3203. URL: http://www.sciencedirect.com/science/article/pii/S0031320312001185.

[8] Jorge Bernal et al. "WM-DOVA maps for accurate polyp highlighting in colonoscopy: Validation vs. saliency maps from physicians". In: *Computerized Medical Imaging and Graphics* 43 (2015), pp. 99–111. ISSN: 0895-6111. URL: http://www.sciencedirect.com/science/article/pii/S0895611115000567.

[9] Ken Chatfield et al. "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531* (2014).

[10] Adam Coates. "Demystifying unsupervised feature learning". PhD thesis. Stanford University, 2012.

[11] J Cohen. "Breakthrough technologies 2013: memory implants". In: *MIT Technology Review. Retrieved October* 7 (10), p. 2015.

[12] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20 (1995), pp. 273–297.

[13] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[14] David Eigen et al. "Understanding deep architectures using a recursive convolutional network". In: *arXiv preprint arXiv:1312.1847* (2013).

[15] Spiros V Georgakopoulos et al. "Weakly-supervised Convolutional learning for detection of inflammatory gastrointestinal lesions". In: *Imaging Systems and Techniques (IST), 2016 IEEE International Conference on*. IEEE. 2016, pp. 510–514.

[16] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[18] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[19] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[20] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. “A practical guide to support vector classification”. In: (2003).

[21] Wang Hui-Hui et al. “Audio signals encoding for cough classification using convolutional neural networks: A comparative study”. In: *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pp. 442–445. DOI: 10.1109/BIBM.2015.7359724.

[22] S. Hwang and M. E. Celebi. “Polyp detection in Wireless Capsule Endoscopy videos based on image segmentation and geometric feature”. In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 678–681. ISBN: 1520-6149. DOI: 10.1109/ICASSP.2010.5495103.

[23] S. Hwang et al. “Polyp Detection in Colonoscopy Video using Elliptical Shape Feature”. In: *2007 IEEE International Conference on Image Processing*. Vol. 2, pp. II –465–II –468. ISBN: 1522-4880. DOI: 10.1109/ICIP.2007.4379193.

[24] D. K. Iakovidis et al. “A comparative study of texture features for the discrimination of gastric polyps in endoscopic video”. In: *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, pp. 575–580. ISBN: 1063-7125. DOI: 10.1109/CBMS.2005.6.

[25] Xiao Jia and Max Q-H Meng. “A deep convolutional neural network for bleeding detection in Wireless Capsule Endoscopy images”. In: *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the*. IEEE. 2016, pp. 639–642.

[26] Andrej Karpathy. “Stanford University CS231n: Convolutional Neural Networks for Visual Recognition”. In: (). URL: http://cs231n.stanford.edu/syllabus.html.

[27] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

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

[29] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[30] B. Li, M. Q. H. Meng, and C. Hu. “A comparative study of endoscopic polyp detection by textural features”. In: *Intelligent Control and Automation (WCICA), 2012 10th World Congress on*, pp. 4671–4675. DOI: 10.1109/WCICA.2012.6359363.

[31] Qinghui Liu and António L. L. Ramos. “Advances and Future Perspectives on Computer-Aided Diagnosis: The Case of Automatic Polyp Detection Based on Gastrointestinal Imaging”. In: *The 21th International Conference on Emerging Trends and Technologies in Designing Healthcare Systems*. Society For Design And Process Science. SDPS. 2016, 216–222.

[32] Gilles Louppe. “Understanding random forests: From theory to practice”. In: *arXiv preprint arXiv:1407.7502* (2014).

[33] A. V. Mamonov et al. “Automated Polyp Detection in Colon Capsule Endoscopy”. In: *IEEE Transactions on Medical Imaging* 33.7 (2014), pp. 1488–1502. ISSN: 0278-0062. DOI: 10.1109/TMI.2014.2314959.

[34] Andriy Mnih and Geoffrey E Hinton. “A scalable hierarchical distributed language model”. In: *Advances in neural information processing systems*. 2009, pp. 1081–1088.

[35] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*. Vol. 37. Springer Science & Business Media, 2012.

[36] Ruwan Nawarathna et al. “Abnormal image detection in endoscopy videos using a filter bank and local binary patterns”. In: *Neurocomputing* 144 (2014), pp. 70–91.

[37] Ruwan Dharshana Nawarathna et al. “Abnormal image detection using texton method in wireless capsule endoscopy videos”. In: *International Conference on Medical Biometrics*. Springer. 2010, pp. 153–162.

[38] Mengqi Pei et al. “Small bowel motility assessment based on fully convolutional networks and long short-term memory”. In: *Knowledge-Based Systems* 121 (2017), pp. 163–172.

[39] Otávio AB Penatti, Keiller Nogueira, and Jefersson A dos Santos. “Do deep features generalize from everyday objects to remote sensing and aerial scenes domains?” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2015, pp. 44–51.

[40] E. Ribeiro, A. Uhl, and M. Häfner. “Colonic Polyp Classification with Convolutional Neural Networks”. In: *2016 IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 253–258. DOI: 10.1109/CBMS.2016.39.

[41] E. Ribeiro et al. “Colonic Polyp Classification with Convolutional Neural Networks”. In: *2016 IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 253–258. DOI: 10.1109/CBMS.2016.39.

[42] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[43] Stuart Russell, Peter Norvig, and Artificial Intelligence. “A modern approach”. In: *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs* 25 (1995), p. 27.

[44] L. Santos-Mayo, L. M. San Jose-Revuelta, and J. Ignacio Arribas. “A computer-aided diagnosis system with EEG based on the P3b wave during an auditory odd-ball task in schizophrenia”. In: *IEEE Transactions on Biomedical Engineering* PP.99 (2016), pp. 1–1. ISSN: 0018-9294. DOI: 10.1109/TBME.2016.2558824.

[45] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 815–823.

[46] Ali Sharif Razavian et al. “CNN features off-the-shelf: an astounding baseline for recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 806–813.

[47] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[48] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[49] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[50] Yaniv Taigman et al. “Deepface: Closing the gap to human-level performance in face verification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1701–1708.

[51] N. Tajbakhsh, S. R. Gurudu, and J. Liang. “Automated Polyp Detection in Colonoscopy Videos Using Shape and Context Information”. In: *IEEE Transactions on Medical Imaging* 35.2 (2016), pp. 630–644. ISSN: 0278-0062. DOI: 10.1109/TMI.2015.2487997.

[52] N. Tajbakhsh, S. R. Gurudu, and J. Liang. “Automatic polyp detection in colonoscopy videos using an ensemble of convolutional neural networks”. In: *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, pp. 79–83. ISBN: 1945-7928. DOI: 10.1109/ISBI.2015.7163821.

[53] N. Tajbakhsh et al. “Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?” In: *IEEE Transactions on Medical Imaging* 35.5 (2016), pp. 1299–1312. ISSN: 0278-0062. DOI: 10.1109/TMI.2016.2535302.

[54] Pietro Valdastri, Massimiliano Simi, and Robert J. Webster. “Advanced Technologies for Gastrointestinal Endoscopy”. In: *Annual Review of Biomedical Engineering* 14.1 (2012), pp. 397–429. ISSN: 1523-9829. DOI: 10.1146/annurev-bioeng-071811-150006. URL: http://dx.doi.org/10.1146/annurev-bioeng-071811-150006.

[55] Pascal Vincent et al. “Extracting and composing robust features with denoising autoencoders”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103.

[56] Jason Weston et al. “Deep learning via semi-supervised embedding”. In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 639–655.

[57] G Wimmer et al. “Convolutional Neural Network Architectures for the Automated Diagnosis of Celiac Disease”. In: *International Workshop on Computer-Assisted and Robotic Endoscopy*. Springer. 2016, pp. 104–113.

[58] Xiang Wu, Ran He, and Zhenan Sun. “A lightened cnn for deep face representation”. In: *2015 IEEE Conference on IEEE Computer Vision and Pattern Recognition (CVPR)*. 2015.

[59] Yixuan Yuan and Max Q-H Meng. “Polyp classification based on bag of features and saliency in wireless capsule endoscopy”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3930–3935.

[60] Matthew D Zeiler. “ADADELTA: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[61] Rongsheng Zhu, Rong Zhang, and Dixiu Xue. “Lesion detection of endoscopy images based on convolutional neural network features”. In: *Image and Signal Processing (CISP), 2015 8th International Congress on*. IEEE. 2015, pp. 372–376.