Bachelor's Thesis Computing Science

Radboud University Nijmegen

ATIA:
Automated Tongue Image Analysis for autoimmune disease detection with supervised machine learning models

Author:
Vamsi Yerramsetti
s1032599

First supervisor/assessor:
Dr. Prof Elena Marchiori

Second assessor:
Dr. Prof Johannes Textor

Second supervisor:
Dr. Prof Alessandro Torcinovich

August 23, 2022

Abstract

Tongue diagnosis is considered an effective and non-invasive technique commonly used to carry out secondary diagnosis at any time and anywhere. Automated supervised machine and deep learning models have become a disruptive technology that leverages computerized algorithms to dissect complex data. Among the many clinical applications of such models, diagnostic imaging has the most promise due to its cognizant detection and ability to quantify various diseases. However, with the existence of various models adopting diverse strategies for distinct diseases, the search for the best fitting supervised machine and deep learning model becomes a strenuous task. This paper focuses on diagnosing a prevalent autoimmune disease known as chronic gastritis. Contrary to the standard invasive methods of diagnosis, we perform automated tongue image analysis on a tongue image dataset using a set of well-established supervised machine learning models and a pre-trained deep neural network. The primary objective of this paper is to accurately distinguish and classify images of tongues infected with chronic gastritis from healthy ones using supervised machine learning models. The secondary objective of the paper is to perform a comparative study between the performance of the tested models, thus answering the research question: *"Which supervised machine / deep learning model yields the best performance in diagnosing autoimmune diseases via tongue image analysis?"*. The resultant experimental values of the pre-trained ResNet-50 convolutional neural network revealed the highest average accuracy of 90% among the supervised machine learning models. The model produced a precision of 81.81%, recall of 90%, F1 score of 85.71%, and an AUC of 0.9. It yielded the best performance for solving our binary classification problem.

Keywords- Supervised machine learning, Deep learning, Chronic gastritis, Autoimmune diseases, Tongue image analysis, Deep neural network, Convolutional neural network (CNN), Diagnostic imaging.

Contents

1 Acknowledgment **4**

2 Introduction **5**
2.1 Clinical background 6
2.1.1 Chronic Gastritis 6
2.1.2 Tongue symptoms 6

3 Preliminaries **7**
3.1 Binary image classification problems 7
3.2 Supervised machine learning 7
3.2.1 Deep learning 7
3.2.2 Transfer learning 8
3.3 K-fold cross validation 8
3.4 Neural Networks 9
3.4.1 Functionality 9
3.5 Convolutional neural networks. 10
3.5.1 Layers in a CNN 11
3.5.2 Loss function 12
3.6 Optimisation algorithms 13
3.7 Underfitting and overfitting 13
3.8 Vanishing and exploding gradient problem 13
3.9 Evaluation techniques 14
3.9.1 Confusion matrix 14
3.9.2 Evaluation metrics 14
3.9.3 ROC curve and AUC. 14

4 Related Work **16**
4.1 Proposed tongue segmentation framework 16
4.2 Tongue analysis by a VGG 19 CNN 16
4.3 Image Acquisition 17
4.4 Key differences and expansion of our work 17

5 Methodology **18**
5.1 Hypothesis 18
5.2 Data description 18
5.2.1 TongueSetA 19
5.2.2 TongueSetB 19
5.3 Data pipeline 19
5.4 Image preprocessing 19
5.5 Image segmentation 20
5.6 Feature Extraction 20
5.6.1 Geometric Features 21
5.6.2 Chromatic features 21
5.6.3 Textural Features 21

5.7 Test train split . . . 22
5.8 Implementing K Fold Cross Validation . . . 22
5.9 Setting the seed . . . 24
5.10 Training machine learning models . . . 24
5.11 Logistic Regression . . . 24
5.11.1 Motivation for using the logistic regression classifier . . . 25
5.11.2 Important characteristics of the logistic regression classifier . . . 25
5.11.3 Sigmoid function . . . 25
5.11.4 Decision boundary . . . 26
5.11.5 Training a multidimensional logistic regression classifier . . . 26
5.12 K-Nearest Neighbors (KNN) . . . 26
5.12.1 Lazy learner . . . 27
5.12.2 Euclidean distance . . . 27
5.12.3 Curse of dimensionality . . . 27
5.12.4 Choosing the K value . . . 28
5.12.5 Working of model . . . 28
5.13 Support Vector Classifier (SVC) . . . 28
5.13.1 Motivation of using SVCs . . . 29
5.13.2 Hyperplane . . . 29
5.13.3 Soft margin . . . 29
5.13.4 The “C” hyperparameter . . . 29
5.13.5 Kernelization . . . 30
5.13.6 Training an SVC . . . 30
5.14 Decision tree classifier . . . 31
5.14.1 Motivation for using decision trees . . . 32
5.14.2 Gini impurity . . . 32
5.14.3 Node purity . . . 32
5.14.4 Max Depth hyperparameter . . . 32
5.14.5 Training a decision tree classifier . . . 32
5.14.6 Shortcomings of decision trees . . . 33
5.15 Random Forest . . . 33
5.15.1 Motivation for using random forests . . . 34
5.15.2 Bootstrapping . . . 34
5.15.3 Bagging . . . 34
5.15.4 Random forest classifier . . . 34
5.16 Residual Networks . . . 35
5.17 Pre-trained ResNet-50 . . . 36
5.17.1 Motivation for using a pre-trained ResNet 50 . . . 36
5.17.2 Structure of ResNet-50 . . . 36
5.17.3 Freezing layers . . . 37
5.17.4 Working of a pretrained ResNet-50 . . . 38
5.17.5 Max and average pooling layer . . . 38
5.17.6 Network optimization . . . 38
5.17.7 Batch normalisation . . . 39
5.17.8 ADAM . . . 39
5.18 Evaluating machine learning models . . . 40
5.18.1 Statistical tests . . . 40
5.18.2 Bonferroni correction . . . 41

6 Results **42**
6.1 Logistic regression results . . . 42
6.1.1 Cross validation results . . . 42
6.1.2 Classification performance . . . 43
6.2 KNN results . . . 43
6.2.1 Cross validation results . . . 43
6.2.2 Classification performance of KNN (K=1) . . . 44
6.2.3 Classification performance of KNN (K=15) . . . 45
6.3 SVC results . . . 46
6.3.1 Cross validation results . . . 46
6.3.2 Classification performance of SVC with linear kernel . . . 46
6.3.3 Classification performance of SVC with RBF kernel . . . 47
6.4 Decision tree results . . . 48
6.4.1 Cross validation results . . . 48
6.4.2 Classification performance . . . 48
6.5 Random forest results . . . 49
6.5.1 Cross validation results . . . 49
6.5.2 Classification performance of random forest (no CV) . . . 50
6.5.3 Classification performance of random forest with CV . . . 50
6.6 Pre-trained ResNet-50 results . . . 51
6.6.1 Classification performance of pre-trained ResNet-50 . . . 51
6.6.2 Trends over epochs . . . 52

7 Comparative analysis and discussion **53**
7.1 t-SNE visualization of the dataset . . . 53
7.2 Accuracy comparison . . . 54
7.3 Recall comparison . . . 55
7.4 Precision comparison . . . 56
7.5 F1 score comparison . . . 56
7.6 AUC comparisons . . . 57
7.7 Ranking the models . . . 57
7.8 Implementing statistical tests . . . 58
7.9 Pre-trained ResNet-50 . . . 59
7.10 Ranking . . . 60
7.11 Performance comparison with state of the art . . . 61

8 Conclusions and future work **62**
8.1 Limitations . . . 62
8.2 Future work . . . 62

A Appendix **64**
A.1 Scikit-learn . . . 64
A.2 PyTorch . . . 65
A.3 Dataset and sample images . . . 65

Chapter 1

Acknowledgment

The work presented in this bachelor thesis was supported and supervised by Radboud University's honors academy of science[1]. The first five months of this year-long project consisted of building a knowledge base in the domain of machine learning under the MOCIA project[2]. This was followed by a computer vision research internship at Ca Foscari University of Venice under the mentorship of Dr. Prof Marcello Pelillo and Dr. Alessandro Torcinovich. The honors program of science facilitated this internship by awarding the author the honors grant of science and providing skill training workshops. I want to thank my mentors, Dr. Prof Elena Marchiori, Dr. Alessandro Torcinovich, and Dr. Prof Marcelo Pelillo, for their guidance throughout the project. I want to thank Dr. Prof Johannes Textor for assessing my thesis. Finally, I would like to thank the honors board of science and Radboud university for granting this prestigious opportunity and investing in my growth.

[1]Raboud honors academy of science website

[2]The MOCIA WP2 project website

Chapter 2

Introduction

Tongue image classification is defined as a significant element in Traditional Chinese Medicine (TCM) [62]. In recent decades, Chinese physicians have predicted a patient's health condition by examining the tongue's color, shape, and structure. However, inconsistent results are a massive limitation for traditional tongue inspection approaches [64]. With the advancement in digital clinical imaging tools and pattern recognition models, computer-aided tongue analysis is considered to be remarkable in TCM since the attained results have proven to be accurate, reliable, and consistent.

Supervised machine learning has revolutionized the diagnostic imaging landscape by introducing robust, data-driven, and statistical approaches to recognize patterns, self-learn, and accurately classify images. According to [20] AI-enabled systems and models will overtake and be more reliable than human medical practitioners by 2050 at the latest. The present bottleneck of the healthcare workforce and the staggering amount of human-made diagnostic errors [12] has led to automation taking a crucial role in healthcare. We strongly believe in investing and pivoting to machine learning-driven automated medical assistants to analyze a voluminous amount of cases. Simultaneously, doctors can focus on anomalies and erratic data. Automated diagnosis in an environment of limited human interaction is the need of the hour. Clinical applications have been greatly benefited by the constant improving in learning ability of such models. These models have the potential to become medical assistants to doctors and alleviate heavy repetitive, time-consuming workload [20]. In modern artificial intelligence, supervised machine learning consists of a family of models, each having its strengths and weaknesses. The models mainly differ in the strategy and the type of data they deal with. With an array of such models and countless diseases, the task of finding the right pair that yields optimal performance is a laborious task requiring an accurate comparative study. In this study, we narrow down the compared models to five supervised machine learning and a deep neural network. These models aim to be the best classifier by attempting to analyze a dataset of tongue images and classify them as infected with chronic gastritis or healthy.

Chronic gastritis affects the stomach's inner lining and is hard to detect without medical intervention as the symptoms tend to appear slowly over time. According to [45] half the world's population will encounter some form of gastritis at some point in their life. The traditional invasive way of diagnosing chronic gastritis is by performing an esophagogastroduodenoscopy (EGD). However, due to the autoimmune nature of chronic gastritis, symptoms affect the color, texture, and geometric nature of the patient's tongue. These features can be leveraged by supervised machine learning models to perform an efficient tongue image analysis for the diagnosis of chronic gastritis. There is a bottleneck of trained doctors in healthcare and a growing number of patients with autoimmune diseases. Additionally, there is limited research and data on machine learning based tongue analysis applications as most diseases in this scope have

established traditional methods of diagnosis. This resulted in a large research gap and has paved the path to finding innovative, efficient, and non-invasive automated diagnostic imaging techniques. Machine learning in diagnostic imaging is a broad field with diverse approaches and numerous models. Through an extensive comparative study, this paper aims to find and reason the best performing machine learning model for our binary image classification problem.

The rest of this paper is organized as follows: Chapter 3 introduces critical concepts and theory that we build on in this thesis. Chapter 4 highlights previous related research from experts in this domain, some of which we drew inspiration. Chapter 5 explains our study's methodology, models, and implemented techniques. Chapter 6 showcases the results of each model. In chapter 7, we compare the performance of the models, reason our observations and rank the models. Lastly, in chapter 8, we summarise our conclusions, highlight the limitations of this study and discuss how we can expand our study in the future.

2.1 Clinical background

This section provides a rudimentary biological overview of autoimmune diseases, chronic gastritis and its symptoms.

2.1.1 Chronic Gastritis

Autoimmune disease spawn when the body's natural defense system cannot distinguish between its cells and foreign cells, causing the body to attack normal cells mistakenly. Physical and visual changes to the tongue are common symptoms of such diseases. Chronic gastritis is the type of gastritis that hails from an autoimmune cause. It affects the mucus-lined layer of the stomach, known as the gastric mucosa. This inner lining is inflamed or irritated over an extended period causing stomachache, nausea, and vomiting [62]. Without proper treatment, chronic gastritis can progress over several months and years to the point where the stomach lining is extensively damaged. This puts the patient at risk of peptic ulcers, gastric polyps, anemia, and even non-cancerous tumors.

2.1.2 Tongue symptoms

There are many versions of chronic gastritis. We are interested in the most common variant which is type "A" chronic autoimmune gastritis. The human tongue can indicate signs of various illnesses, reflected by its surface coating, texture, chromatic, and geometric features. Type "A" chronic gastritis tends to form a yellow thickened coating on the human tongue caused by a bacterium called Helicobacter Pylori [47]. It is important to note that this coating can also be attributed to an autoimmune cause that affects acid secretion from the stomach [48].

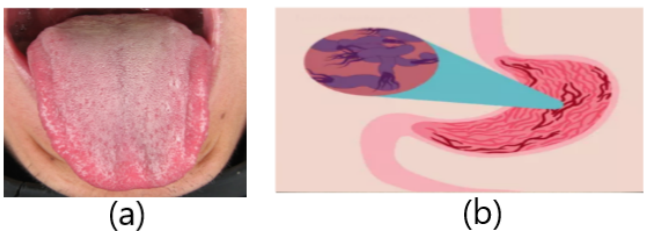


Figure 2.1: (a) Slight yellow pigmentation and coating on the tongue caused by Helicobacter pylori [26](b) Infection of the stomach's inner mucus-lined layer [36].

Chapter 3

Preliminaries

This chapter sheds light on core theoretical concepts and terminology this thesis incorporates and builds on. We provide fundamental background knowledge on supervised machine learning, K-fold cross-validation, convolutional neural networks, loss functions, optimization algorithms, machine learning roadblocks, and some popular evaluation techniques.

3.1 Binary image classification problems

The process of labeling and categorizing images based on a set of classification rules is known as image classification. When the data can only be classified between two classes, we encounter a binary classification problem. In this context, "binary" refers to the two possible classes. Binary image classification consists of defining the two target classes and training a model to recognize, differentiate and classify varying images using sample labeled images. Our use case is a binary classification problem as we wish to classify tongue images as infected or healthy.

3.2 Supervised machine learning

Supervised machine learning is a subcategory of machine learning (ML) and artificial intelligence (AI) that learns a mapping from the input to the output variables. The algorithm learns how the variables relate to the answer by providing rich insights that predict future outcomes based on historical patterns [33]. These models are distinguished by their training approach to classify data or predict outcomes using labeled datasets. The models modify their parameters as input data is fed into it until the model has been fitted to the best of its ability. This branch of machine learning is "supervised" as it uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Two popular use cases of supervised ML are regression and classification problems. We are interested in the latter as our focus is on diagnostic imaging.

3.2.1 Deep learning

Deep Learning is a subfield of supervised machine learning. Deep learning works on the same underlying principles of supervised machine learning. However, these models incorporate an artificial neural network whose structure is inspired by biological neural networks [23]. The structure is composed of nodes in layers that apply a nonlinear transformation to its input and uses what it learns to create a statistical model as output. We dive in-depth

into the structure and its attributes in section 5.16. While traditional machine learning algorithms are shallow, deep learning algorithms are stacked in a hierarchy of increasing complexity and abstraction. The goal of deep learning/statistical inference is to reduce the generalization risk (true risk) [23].

3.2.2 Transfer learning

Transfer learning (TL) is an ML approach that enables models to use previously gained knowledge from one task to solve another related task. The performance of models that use transfer learning is directly proportional to the similarity between the datasets of the two tasks. Transfer learning is the application of knowledge gained from completing one task to help solve a different but related problem. Transfer learning comes into effect as the model's last few layers are active and retrained on datasets of similar classification problems. This state-of-the-art approach has proven advantageous for deep neural networks that solve image classification problems [50, 23]. This is due to the trend of low-level image features (lines, corners) learned on one dataset repeating in another.

Let's assume we have two domains D_O and D_N each consisting of a feature set "$X = \{x_1, x_2, ..., x_n\}$" and marginal probability distribution "$P(X)$". The respective task in each domain "T_O" and "T_N" consists of a label space "Y" and predictive function "$f(X \rightarrow Y)$" that attempts to predict the respective label for every new instance the respective feature space. Transfer learning aims to improve the predictive function of a new task in the new domain by using knowledge from the old task in the old domain.

Therefore, f_N in $D_N(X, P(X)_N$ performing $T_N(Y, f_N)$ is improved by using knowledge in $D_O(X, P(X)_N$ from $T_N(Y, f_O)$ [50, 23, 66].

3.3 K-fold cross validation

When we encounter situations with scarce data availability, the train/validation split further decreases the size of the training set. The model is trained on fewer instances, which can hinder the performance. We can mitigate this problem using K-Fold Cross Validation (CV). The K parameter determines the number of groups into which the given data sample is split. The original training data is split into K non-overlapping subsets [4]. The model training and validation are executed K times, each time training on "K - 1" subsets and validating on a different subset (the one not used for training in that round). Finally, the training and validation errors are estimated by averaging over the results from the K experiments [4].

The first fold acts as a validation set. K-fold CV uses a performance measure PM_i (ex: MSE_i), computed on the observations in the held-out fold. After repeating the above process K times, we get K estimates of the test error. The average of these estimates produces the K-fold CV estimate. We can represent the K fold CV in the equation below. [16, 4].

$$CV_K = \frac{1}{k} \sum_{i=1}^{K} PM_i \tag{3.1}$$

Where PM_i is the performance measure of fold i.

Higher K (number of folds) infers that the model is trained on a more extensive training set and tested on a smaller test fold. Hence, this leads to a lower prediction error as the models see more of the available data. However, a very large K restricts the number of

possible sample combinations, thus limiting the number of distinct iterations. This could lead to high sensitivity of noisy data [4]. Lower K infers that the model is trained on a smaller training set and tested on a more extensive test fold. This leads to greater scope for the data distribution in the test fold to differ from the training set [4]. Hence, we should expect a higher prediction error on average. However, a lower K weakens the strength of the cross-validation as we are using smaller splits. Therefore a lower K could lead to less reliable results.

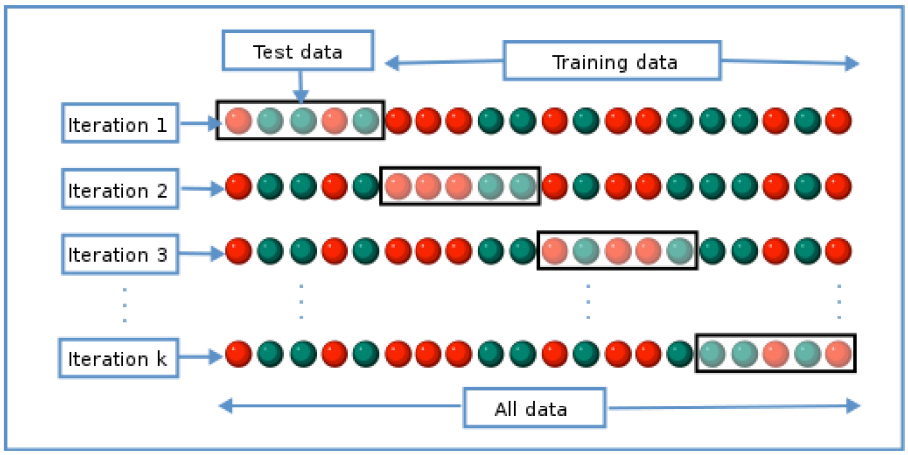


Figure 3.1: Iterative working of K-fold cross validation (File:K-fold cross validation EN.svg - Wikimedia Commons, 2022)

3.4 Neural Networks

Artificial neural networks (ANNs) are a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Their computational architectures are loosely inspired by biological neuron processes in our brain [8, 43]. Synapses in the brain allow neurons to receive, analyze, and transfer signals to one another. Similarly, neural networks have nodes connected by edges to imitate this behavior. Knowledge is acquired by implicitly learning simple mathematical rules instead of explicitly programming them. Such artificial neurons have adaptable synapses that can change their synaptic strengths, which defines the learning process. Knowledge is stored in such connections. Mimicking the ability to activate a neuron requires the neural networks to incorporate activation functions [8]. These functions specify how the weighted sum of the inputs is converted into an output, aid the network in learning complicated patterns and control the transition at each artificial neuron. When these nodes are connected hierarchically, they are called a deep neural network.

3.4.1 Functionality

Untouched input is sent to the first layer's input neurons. After processing the input, the first layer passes it to the neurons of the next layer. The subsequent layer processes the input from the previous layer, potentially transforming it and passing it to the next layer. This cycle is repeated from layer to succeeding layer until the last layer is reached. The activation mechanisms of the output neurons that make up the NN's result repeatedly alter the process before it reaches the final layer [35, 43]. The weight value given to an edge determines how important an edge is. As a result, the weights can alter the strength of the signal as it travels between neurons. In other words, most data required for the network to function is encoded in the weight values.

Forward propagation is the process of receiving an external input, processing it, and generating output. Most frequently, training a NN entails enabling the network to complete the required task on a subset of data observations, assessing its performance through a loss function, and then carrying out backpropagation [35]. In order to implement backpropagation, the algorithm must propagate the supervision generated by the loss backward. This starts with the output layer and propagates towards the input layer. During backpropagation the edge weights are adjusted per the estimated error. The network mistake rate is reduced by repeating these forward and backward processes.

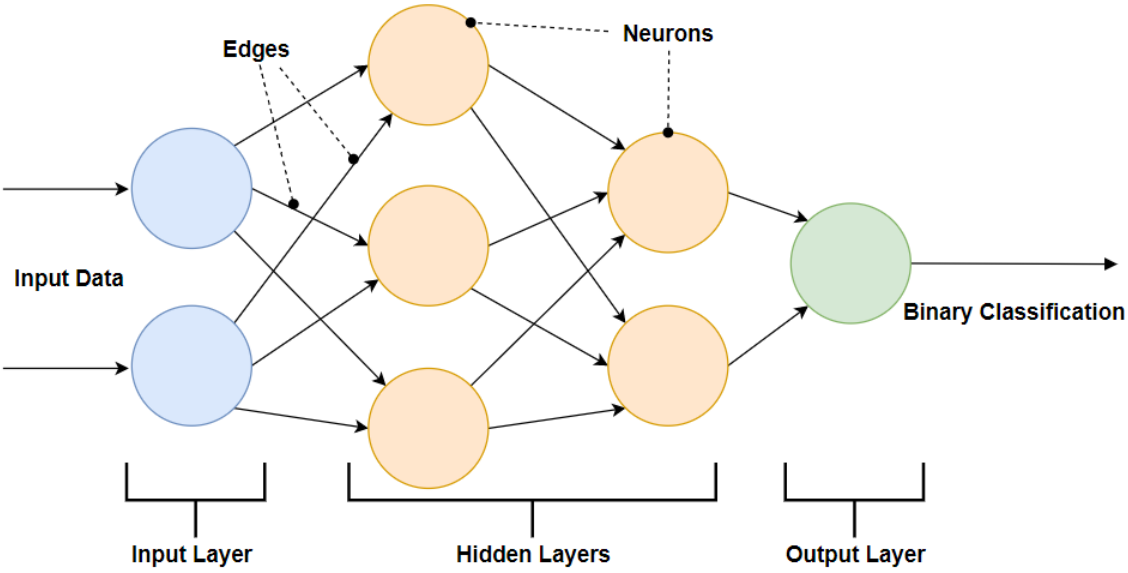


Figure 3.2: Deep neural network for binary classification

A neural network is trained for one cycle during an epoch using all of the training data. We only use each piece of information once within an epoch. Each epoch consists of one or more batches in which the neural network is trained using a portion of the dataset. The number of epochs should be set to a finite positive integer. The number of training samples that are processed in each iteration is changed is known as the batch size. The quantity of complete iterations through the training dataset is the number of epochs. The size of a batch must be greater or equal to the number of samples in the training dataset [9]. The epochs and batch size are both hyperparameters for the learning algorithm of the neural network.

The network evaluation is carried out on an untouched test set, while a set of training data is utilized to learn the appropriate weight values.

3.5 Convolutional neural networks.

In the field of deep learning, a convolutional neural network (CNN) is a class of deep neural networks that are most commonly applied to analyze visual imagery. CNN is a network that employs convolutional layers. The mathematical term "convolution" refers to the processing of an input signal with another "convolutive" signal by computing a "sliding" dot-product. [54]. We can represent this as:

$$f * g(x) = \int f(z)g(x-z)dz \tag{3.2}$$

Likewise, CNNs perform convolutions using a kernel over the input data to produce a feature map. The feature map is the output of each layer in a CNN and gives an

in-depth insight into the detected features by the CNN. Convolutions, non-linearities, and pooling operations are interleaved to make sense of the spatial coherence of the input (typically an image) and generate an appropriated output. Convolutional layers are typically arranged to gradually decrease the spatial resolution of the representations while increasing the number of convolutional channels (feature maps) [10, 3]. In traditional CNNs, the representations encoded by the convolutional blocks are processed by one or more fully-connected layers before emitting output.

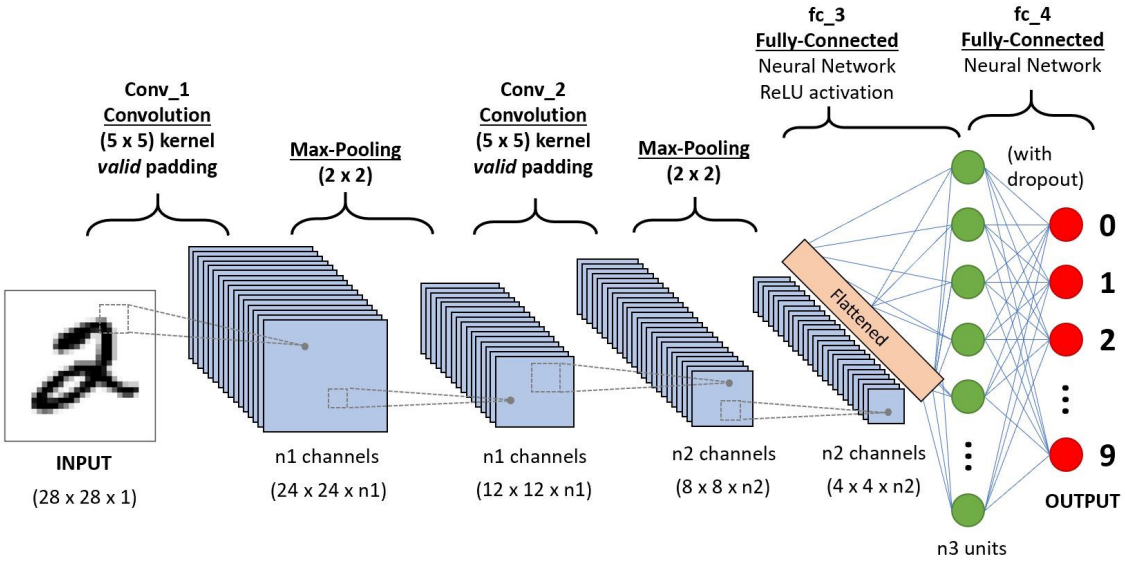


Figure 3.3: A CNN searching for various patterns and performing feature extraction by leveraging various sized kernels for each respective convolutional layer to accurately classify the MNIST image dataset [21]

3.5.1 Layers in a CNN

Convolutional layer

These layers perform convolutions using small-scaled filters/kernels that hover over the images in a set direction. Typically, the filter's size is smaller than the original image. Each filter produces an activation map after it convolves with the image. This kernel window operates a scalar product between the weights and the input values, generating an output value for each slide. Once the first kernel slides over the entire image, we can use another to obtain an output with the number of given channels. In this iterative manner we can perform feature extraction.

We can represent a convolution of input matrix $X : n_h * n_w$, kernel matrix $W : k_h * k_w$, scalar bias: b and output matrix Y: $(n_h - k_h + 1) * (n_w - k_w + 1)$

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{-a,-b} * x_{i+a,j+b} \tag{3.3}$$

This results in $Y = X * W + b$ where W and b are learnable parameters.

Pooling layers

Deep learning models are computationally expensive. Pooling layers attempt to reduce computational complexity by encapsulating the existence of features in sub-regions of the feature maps. Downsampling feature maps leads to a resolution reduction.

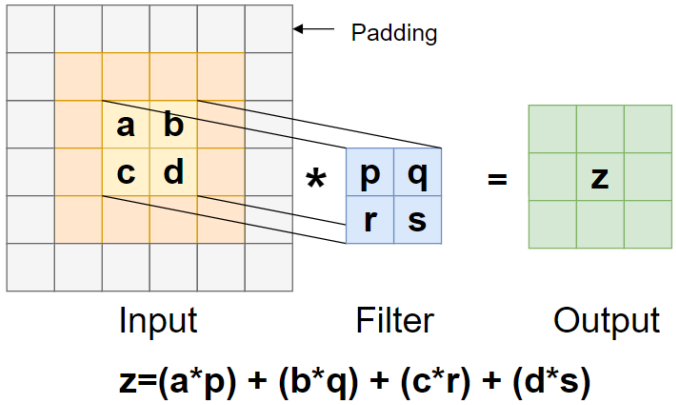


Figure 3.4: Representation of a convolution layer that applies a 2*2 filter on a padded image with a stride of 2.

Non-linearity layer

Non-linearity layer refine and bound the produced output. The non-linearity layer converts the images to a reduced form, but ensures that important features and components of the images are not lost.

Fully connected layers

Layers where every node is connected to a node in the previous and subsequent layers are known as fully connected layers. Feed forward neural networks accommodate fully connected layers.

CNN receives an image as its input (data with three dimensions: height, width, and the number of channels that corresponds to the color scale). Each neuron analyzes information within a particular receptive field of the image. The neurons are arranged in layers such that they can identify simple patterns in the first layer and increasingly complicated patterns in the succeeding layers.

3.5.2 Loss function

The "loss" in a neural network is its prediction error. This is done by calculating the gradients, which are used to update the weights in the neural network. This iterative process enables neural networks to train. The loss function quantifies the difference between the actual outcome and the predicted outcome. There are multiple loss functions, each specialized for different types of problems and machine learning models [53]. The equation below represents the binary cross entropy loss function.

$$Loss = -\frac{1}{z}\sum_{i=1}^{z} y_i * log\widehat{y_i} + (1 - y_i) * log(1 - \widehat{y_i}) \tag{3.4}$$

- y_i is the target label for training example i.
- $\widehat{y_i}$ is the i-th scalar value in the model output.
- z is the number of training examples.

The above expression is equivalent to the average result of the categorical binary cross entropy loss function applied to many independent classification problems, each problem having only two possible classes with target probabilities $\widehat{y_i}$ and $(1 - \widehat{y_i})$.

3.6 Optimisation algorithms

For a deep learning problem, we usually define a loss function first. Given the loss function, we can use an optimization algorithm in an attempt to minimize the loss. A loss function is often referred to as the objective function of the optimization problem. By tradition and convention, most optimization algorithms are concerned with minimization. Although optimization provides a way to minimize the loss function for deep learning, the goals of optimization and deep learning are fundamentally different [42]. The former is primarily concerned with minimizing an objective. In contrast, the latter focuses on finding a suitable model, given a finite amount of data. Training error and generalization error generally differ since the objective function of the optimization algorithm is usually a loss function based on the training dataset. The goal of optimization is to reduce the training error (empirical risk).

3.7 Underfitting and overfitting

When the model is too "simple" for the provided data caused by wrong initial assumptions, we term this setback as underfitting. This results in possible accurate initial predictions, high training error, and a high testing error. Such models have a high bias.[1]

Overfitting occurs when models are too "complex" for the provided data as it models the training data too well, resulting in wrong predictions for test/new data. Such models may have a low training error but tend to have a high test error. Overfit models have a high variance[2].

3.8 Vanishing and exploding gradient problem

The partial derivative of the loss function eventually approaches a value near zero and "vanishes" as the network's layers increase. Consequently, the value of the product of derivatives starts to decline.

The derivative of certain activation functions used to build the neural network gives rise to the vanishing gradient problem. The sigmoid function is an example of an activation function that compresses a vast input space into a narrow input area between 0 and 1. As a result, the output of the sigmoid function hardly changes when the input changes significantly. The derivative shrinks by dividing the gradients of subsequent layers by the gradients of early levels (layers close to the input layer). Therefore, if the gradients of later layers, for instance, are less than 1, their multiplication disappears relatively quickly. As neural networks backpropagate, learning is diminished in the early layer [17]. Using the LeakyReLU activation function can mitigate the vanishing gradient problem. LeakyReLU maps the values less than zeros to a minimal positive number instead of zero [29].

The exploding gradient problem is the exact reverse of the vanishing gradient. It is the scenario when the gradients become too large due to a large accumulation of gradient errors. This problem can be found in deeper layers that have exceptionally large changes to the parameters compared to the minimalist change in the early layers. This results in the weights not being updated. Initializing smaller weights and lower learning rates and using a standard loss function can help prevent exploding gradients.

[1] The error caused by incorrect assumptions by the machine learning model is known as the bias

[2] The difference between the machine learning model between the training and test data is termed as the variance

3.9 Evaluation techniques

In this section we discuss popular evaluation techniques used in the field to evaluate the performance of a machine learning classifier.

3.9.1 Confusion matrix

The confusion matrix is a class by class summary of prediction results by a classifier on a classification problem. The number of correct and incorrect predictions are represented with count values among each class. Apart from the errors being made by the classifier, the confusion matrix gives in-depth insight into the type of errors being made [15, 40]. Our paper deals with a binary classification problem so our confusion matrix is a 2*2 grid. In this way, we assign the healthy row as "negative" and the infected row as "positive". We then assign the healthy column of predictions as "true" and the infected as "false". This results in four important components that make up our confusion matrix:

1. True positives (TP): correctly predicted infected images.
2. False positive (FP): incorrectly predicted infected images.
3. True negative (TN): correctly predicted healthy images.
4. False negative (FN): incorrectly predicted healthy images.

3.9.2 Evaluation metrics

Calculating the accuracy of the model is the standard way to assess the performance of NNs and supervised machine learning models [40]. Additionally, we calculate the F1 score, recall, and precision to gain in-depth knowledge of how sensitive the model is to certain labels. Accuracy is the measure of correctly predicted data points out of all the data points. Precision refers to the number of true positives divided by the total number of positive predictions. The recall is the ratio between the numbers of positive samples correctly classified as positive to the total number of positive samples. Finally, the F1 score is defined as the harmonic mean between precision and recall. Co-relating the components of the confusion matrix and the performance metrics mentioned above, we get the following equations:

1. Accuracy = $\frac{TP+TN}{TP+FP+FN+TN}$
2. F1 Score = $\frac{2TP}{2TP+FP+FN}$
3. Precision = $\frac{TP}{TP+FP}$
4. Recall = $\frac{TP}{TP+FN}$

3.9.3 ROC curve and AUC.

The receiver operating characteristic curve (ROC curve) is a graphical representation of the performance of the classification model under all classification thresholds. The x-axis is the false positive rate, whereas the y-axis is the true positive rate. The classification threshold is inversely proportional to the number of classified positives. In diagnostic imaging, ROC curves are a great way of representing the connection/tradeoff between diagnosis specificity and sensitivity. The total area under the curve (AUC) is the area below the ROC curve. AUC measures how well predictions are ranked and the quality of the model's prediction independent of the chosen classification threshold [40]. In our scenario, the higher the AUC, the better the model is at distinguishing between infected and healthy tongue images.

When a model in binary classification provides us with a score rather than the prediction itself, we typically need to apply a threshold to turn the score into a prediction. It is apparent to pick 0.5 as a threshold because the score's purpose is to convey the perceived likelihood of obtaining the infected class per our model [30]. It makes sense to change the forecast to 1, if having infected images is more likely than having healthy images. The natural cutoff of 0.5 assures that the likelihood of infected images being present is larger than the probability of healthy images. The Scikit-Learn module uses 0.5 as the default threshold.

Chapter 4

Related Work

In this chapter, we describe state-of-the-art research for automated tongue image analysis, diagnostic imaging using supervised machine learning models, feature extraction, and comparative studies between supervised and deep models for classification problems.

Our research revolves around perfecting and automating tongue image analysis. In this process, we aim to extract and draw patterns between the desirable features from the tongue images, which are iteratively trained until the model can classify and produce a diagnosis. Diagnostic tongue image analysis is a niche field of research that has seen quite some exploration but is evolving relatively slowly, with beginnings focusing on automated tongue image detection and segmentation.

4.1 Proposed tongue segmentation framework

Obafemi-Ajayi et al [31] propose a fully automatic tongue detection and tongue segmentation framework, which is an essential step in computer-aided tongue image analysis. This framework and its variants have been widely used in forthcoming clinical research and studies. It is important to note that the product of this study is not specific to any disease but performs tongue detection and segmentation without adjusting any parameters. Inspired by [31] David Zhang et al [60] propose cutting-edge methods to perform tongue image segmentation along with other preprocessing/image augmentation tasks. Their method captures gross shape features in the parameter space by leveraging a steep decent method on its energy function. They use a bi-elliptical deformable contour (BEDC) model [34] that is based on a bi-elliptical deformable template (BEDT) [34]. The second so-called "Snake" approach is made by combining a polar edge detector, and active contour model (ACM) techniques [14]. It resulted in an average true positive percent of 97.1%.

4.2 Tongue analysis by a VGG 19 CNN

Rajakumaran and Sasikala's tongue diagnosis system [39] uses a VGG 19 CNN model to perform feature extraction on a dataset of 996 tongue images. The study classifies the tongue images across 12 autoimmune and heart diseases (some of which are organ-specific gastritis) using a Random forest and Gaussian naive Bayes models. The results show that the deep neural network and random forest combination performed the best with the precision, recall, accuracy, and F1-Score of 93.80%, 93.70%, 93.70%, and 93.68% respectively.

4.3 Image Acquisition

Kim et al, Zhang et al, and Jiang et al [49, 7] used CCD cameras to outline tongues, whereas Zhang et al [7] placed D65 fluorescent tubes around the 8-bit resolution CCD camera to maintain uniform illumination. It is important to mention that we observe a growing trend of smartphone camera usage in studies focusing on building Tongue Diagnosis Systems (TDS) [60, 49]. The reason for this pivot to essentially lower resolution cameras and images is to be able to create accurate and robust models that works swiftly on mobile applications. This leads to the creation of flexible models that can process and diagnose tongue images captured in different environments, lumination and quality. Therefore these studies deal with noise removal by using a varied sizes of kernels and filters. Dulam, Ramesh and G [11] and Rajakumaran and Sasikala's TDS [39] deal with noise removal by using a specific sized Gaussian kernel, which in theory is used as a filter. However, the core results of the research by Otsu and Hsu et al [18, 19] under consideration by Hu et al [19] showed actual promise for the usage of smartphone cameras for tongue region segmentation and tongue diagnosis. The algorithm was tested in smartphones such as Samsung Galaxy S2 and S3 and performed exceptionally well.

4.4 Key differences and expansion of our work

In parallel with multiple previous studies, we use mainstream supervised machine learning models on a binary classification problem. However, we can see a few areas where previous work in this field can be expanded.

1. While most studies utilize two or three similar supervised machine learning models; we worked with five distinct ML models and a reliable pre-trained deep neural network. This gives extensive depth to our paper as we can draw more comparisons, thus filling in our targeted research gap.

2. We use a pre-trained ResNet-50, which handles the vanishing gradient problem and is known to deliver higher accuracy in image classification tasks when compared to other established deep learning models such as VGGNet and GoogleLeNet [61, 21].

3. While most studies focus on solving multi-label classification problems with a wide array of diseases to diagnose, we narrowed it down to binary classification problems for diagnosing one specific autoimmune disease. This is advantageous as we are able to reason and draw in-depth conclusions about the feature extraction part of our process as well as show focused results in our confusion matrix. That being said, our models will still be expandable to other autoimmune diseases as we finetune the parameters to identify geometric, chromatic, and textural changes in the tongue, which are common autoimmune disease symptoms.

Chapter 5

Methodology

This chapter provides a complete overview of our data pipeline. We explain all methods and techniques used in our experiments, from data reprocessing to model evaluation. This chapter consists of in-depth insight into each model and its inner working. We motivate our model selections and reason all our decisions and design choices.

5.1 Hypothesis

After studying the dataset, interpreting the classification problem, and understanding the various pros and cons of the selected models, we made a hypothesis prior to commencing the experiments. We predicted the pre-trained ResNet-50 would be the best performing model due to its reputable performance with image classification problems. Moreover, we predicted a better feature extraction by the model due to its ability to execute rigorous convolutions on the images. However, CNNs like ResNet-50 thrive in terms of performance when provided with large datasets due to their inherent complexity. So we accounted for the limited tongue dataset in our hypothesis. This led us to our subsequent hypothesis, which predicted that the RBF SVC model would be a close second if not equivalent to the pre-trained ResNet-50 in terms of performance.

5.2 Data description

The tongue dataset in this paper is a union of two datasets from external tongue image analysis studies that collected these images for a clinical study on autoimmune disease-related tongue symptoms. Due to privacy requests[1] from both parties, the majority of images in the datasets and codebase are sealed, and the details of their origin are private. For the sake of convenience, we have labeled these sub-datasets as TongueSetA and TongueSetB. We were assured by the authors that both datasets consisted of images that are infected with chronic gastritis and healthy tongue images. Our dataset has a cumulative size of 300 images.

The images in both sub-datasets were captured on a horizontal plane, with the subject opening her mouth and sticking her complete tongue out. Images were confined to the lower face region encapsulating the lips, tongue, and chin of the subject. Due to the near identical camera lens aperture and illumination settings, the image resolution, quality, and axis of the capture of both datasets are nearly identical. For the most part, this avoided any irreducible errors that may have arisen due to dataset differences that cannot be solved by image augmentation.

[1] All parties entered a verbal agreement that all images besides the sample ones present in this thesis, the codebase we developed, and details of origin/metadata is sealed.

5.2.1 TongueSetA

TongueSetA consists of 30 tongue images that have been collected over the south Asian subcontinent in 2019. The images were of size 650 x 450 pixels in the .png format. The dataset consists of 15 ill-tongue images and 15 healthy ones. The images were captured in uniform illumination by using dental lighting and a Sony DSC-RX10M4.

5.2.2 TongueSetB

TongueSetB consists of 270 tongue images in png format that have been collected over North America in 2018. The images were of size 768 x 576 pixels in the .bmp format. The dataset consists of 70 ill-tongue images and 200 healthy ones. The images were captured in uniform illumination by using standard D65 fluorescent tubes with a 6-bit CCD camera. The images were captured at a straight vertical angle with the subjects on a dental saddle seat.

5.3 Data pipeline

We illustrate the implemented data pipeline of our study in figure 5.1. The process starts with initial image prepossessing steps on the raw tongue dataset and ends with evaluating the prediction and performance of each model.

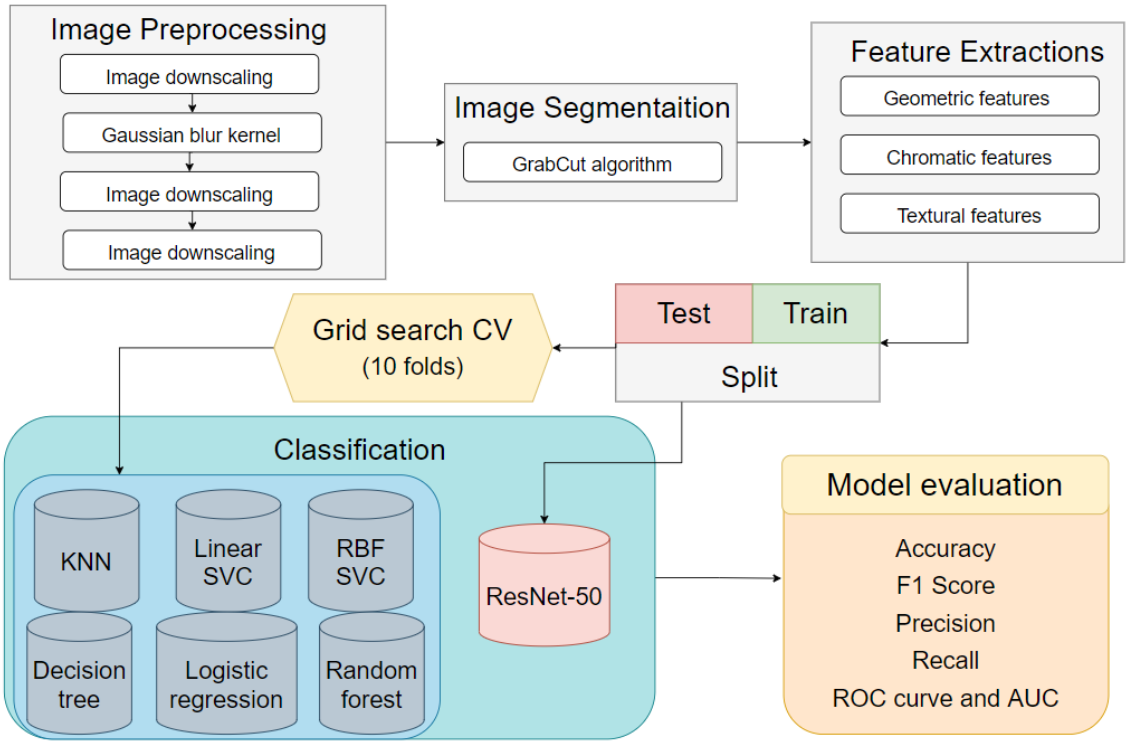


Figure 5.1: Visualization of the data pipeline

5.4 Image preprocessing

We resized the image to a 224 * 224 pixel frame. This is done to facilitate mini-batch learning as we need to specify a uniform size for the images in each batch [39, 60]. Additionally, this strengthens the model's convergence ability. All images then went through a 5x5 gaussian blur kernel. This is a common practice used after we have downsampled images and prior to image resampling [39, 28]. Using a gaussian blur avoids spurious

high-frequency attributes in downsampled images. The preprocessing for the convolutional neural network consisted of an extra step. We converted the colored images to grayscale ones. Colored images have multiple channels leading to multiple convolutions, and a larger computational workload [28]. Whereas grayscale images store values in a single array (black and white) hence requiring only a single convolutional, which results in a significant reduction in computational workload. It is important to note that we did not implement any of the preprocessing steps for the pre-trained ResNet-50 besides resizing the images to a 224 * 224 pixel frame and applying the Gaussian blur. We explain and motivate the exact techniques used for each model in the following sections.

5.5 Image segmentation

This paper analyzed the tongue and the tongue alone. Therefore we had to segment the images in such a way that we have efficient background-foreground extraction. We used a graph-cut based image segmentation method inspired by [25] known as the GrabCut algorithm [41]. This method was used to extract the tongue alone as foreground and blacken the remainder as background for all models. We give the algorithm a specified bounding box around the tongue. The algorithm then starts segmenting by estimating the color distribution of the tongue by using a Gaussian mixture model. We explored other segmentation methods, such as edge/region-based and threshold segmentation methods. However, we selected the GrabCut algorithm due to its robust nature and insensitivity to unknown position-frequency correlations that many methods can not efficiently execute[41]. The GrabCut algorithm was used in the prepossessing of all the models besides the pre-trained ResNet-50. The network learns background-foreground extraction independently while training.

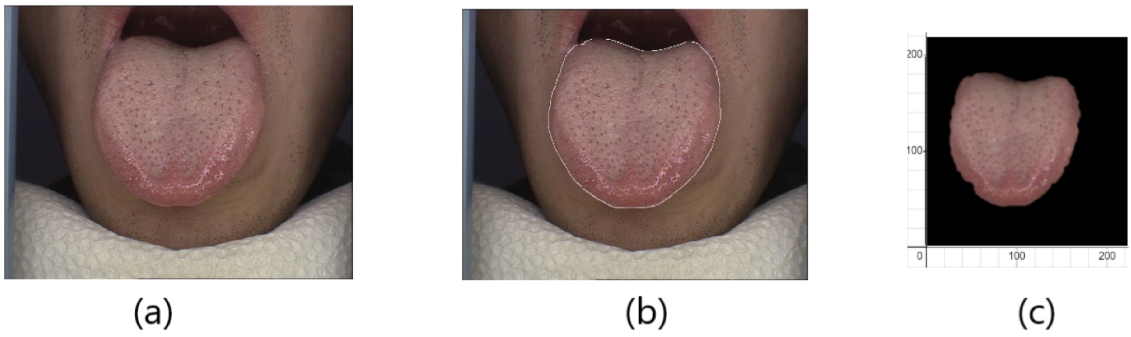


Figure 5.2: (a) Is a raw tongue image from TongueSetA. (b) The tongue is segmented and outlined by the GrabCut algorithm. (c) The segmented image is passed through a Gaussian blur and downsampled to 223*223 pixels.

5.6 Feature Extraction

We followed a kindred feature extraction process to that of Dulam, Ramesh, and G [11].There are three main categories of tongue features that we built our models to analyze. These are the geometric, textural, and chromatic features of the tongue. From these three categories, we obtained and analyzed nine meta-features of the tongue. We manually extracted the features from the tongue dataset which were used during the training of all the tested machine learning models besides the the pre-trained ResNet-50. The pre-trained ResNet-50 learns to extract features while training and did not require us to extract the three feature categories.

5.6.1 Geometric Features

Different parts of the tongue correlate with different diseases. Chronic gastritis symptoms usually affect the lower half of the tongue. We break down the tongue into seven geometric-based attributes to extract this area as a feature. Some are trivial and self-explanatory, such as tongue surface area, height, width, and height-width ratio. The remaining attributes are:

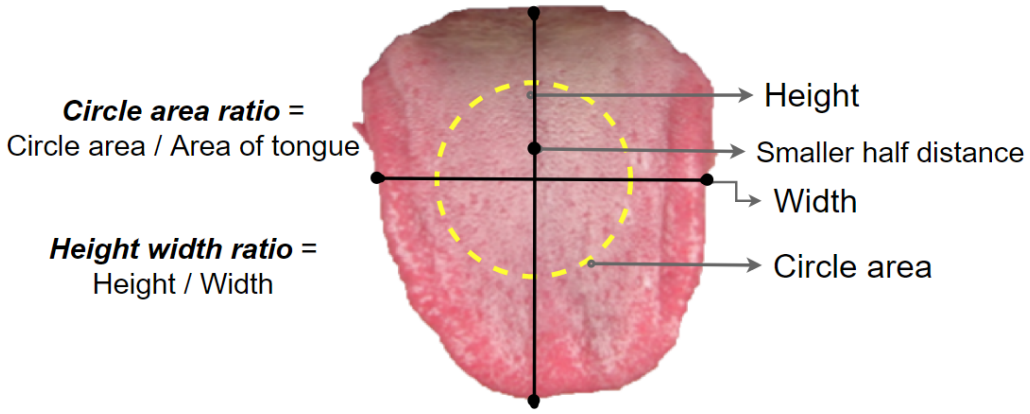


Figure 5.3: Extracted geometric features from a sample tongue image

Smaller half distance

We select between the lesser values attribute between the height and the width. This attribute is then halved and makes up the smaller half distance.

Circle area

This is the area that is formulated using the smaller half distance of the tongue. This region of the tongue is prone to exhibiting chronic gastritis symptoms.

Circle area ratio

The ratio between the circle area and the total area of the tongue. Circle area ratio is useful for extracting the infected yellow coating region of the tongue.

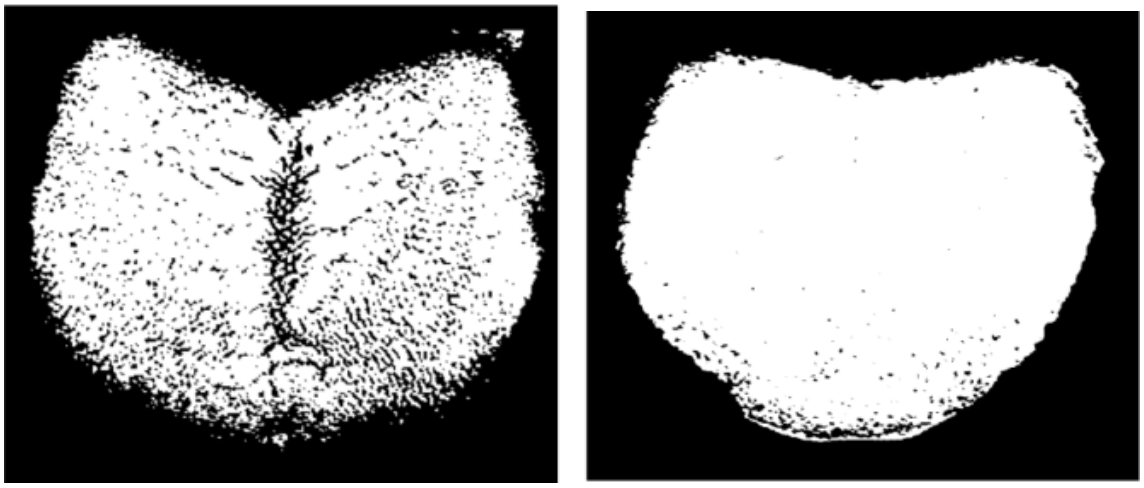
5.6.2 Chromatic features

We used the RBG scale to classify the tongue images into two color classes: pink (255, 182, 193) and yellow (255,255,0). We calculated the Manhattan distance between the individual RGB values of each pixel and the base color values. The images are then classified into one of the two color classes based on the average Manhatten distance of the pixels in the foreground that we obtain from the GrabCut algorithm. A healthy tongue is usually pinkish red and is classified into the healthy pink class. In contrast, a tongue infected with chronic gastritis has a yellowish region on the smaller half area of the tongue due to Helicobacter Pylori bacterial growth [11, 45].

5.6.3 Textural Features

Besides the yellowish pigmentation, chronic gastritis tends to create a coating over the tongue's surface. This affects the tongue's texture. A healthy tongue has a smooth texture, whereas an infected tongue has a coating over most of the tongue's surface area [45, 11, 36]. Out of the three feature classes, textural attributes are the hardest to extract and interpret from preprocessed images. We binarized two sample tongue images to showcase textural

differences between infected and healthy tongues. We did not involve binarization in our preprocessing but as a visualization tool to showcase this feature in this paper.



Lack of coating as textural features are clearly exhibited. This may be a healthy tongue.

Possible coating in the circle area of the tongue surface. This could be due to chronic gastritis.

Figure 5.4: Binarization performed on two sample tongue images.

5.7 Test train split

In modern-day healthcare, sensitive data such as tongue imagery is minimal. Datasets of tongue images infected with type A chronic gastritis are very scarce as traditional diagnosis methods don't require tongue imagery. In this study, we dealt with a relatively small dataset consisting of a limited amount of infected images. Since our study dealt with a low amount of infected tongue images and involved a deep neural network, we used a 90/10 test-train split. This was the optimal balance given our constraints as a CNN requires a test set of at least 30 images to be effective [38]. 90% of images (270 images) were allocated to the training set. The training split fed to machine learning models enabling them to self-learn and make the predicted diagnosis. We had a 1:3 dataset size ratio between TongueSetA and TongueSetB. Therefore we have 195 healthy images (13 images from TongueSetA and 172 images from TongueSetB) and 75 healthy images (14 images from TongueSetA and 61 images from TongueSetB)..

The validation set, also known as the test split, is used to evaluate the final model objectively. The validation dataset is unseen data and is not used for training. After the model is done training, it attempts to classify the respective unseen images. We can evaluate the performance and derive the model's accuracy based on how well the model performs in the testing phase. Our validation/test split consists of 30 images, of which 20 are healthy, and ten are infected. Two healthy images from TongueSetA and 18 healthy images from TongueSetB make up the healthy images of the validation set. One infected image from TongueSetA and nine infected from TongueSetB make up the infected images in the validation set.

5.8 Implementing K Fold Cross Validation

With the previously mentioned data scarcity hurdles in place and a high risk of overfitting, we resorted to implementing K-fold cross-validation (CV) in all the models besides the pre-trained ResNet-50. Cross-validation is generally not used in deep learning models due to more significant computational expense. We used transfer learning instead.

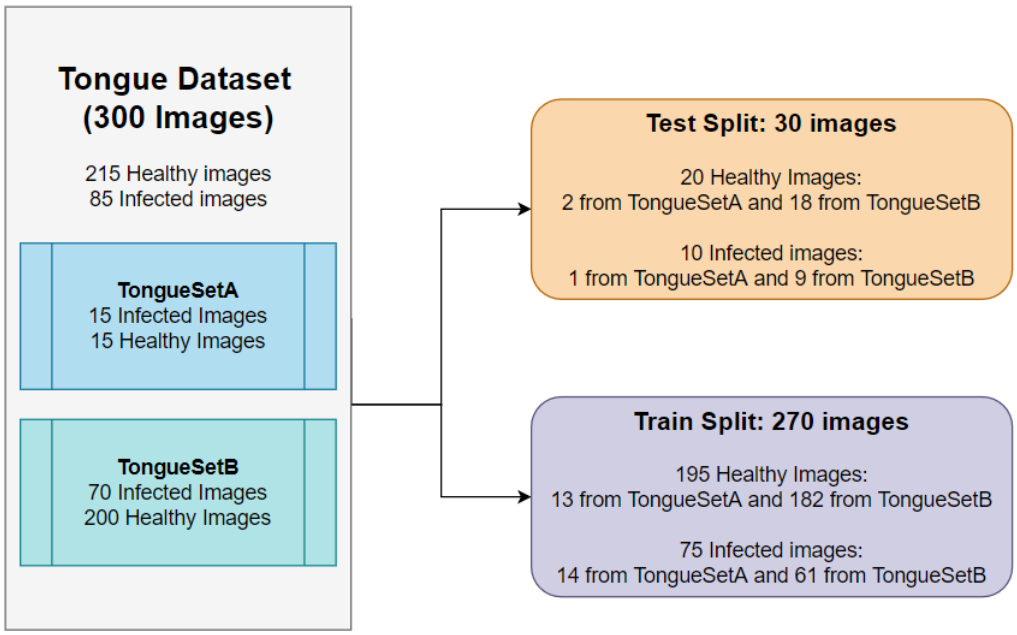


Figure 5.5: Visualisation of the used test- train split

Finetuning specific critical hyperparameters in each model can lead to better performance. This can be done by using a higher-level cross-validation construct known as GridSearchCV. GridSearchCV takes a cross-validation engine like K-fold CV as an argument to perform a grid search over a predefined set of hyperparameters. This iterative technique produces the best fitting value for the selected hyperparameter at each fold. We take the mode over the produced values and set the respective hyperparameter of that value. In theory, this style of finetuning should yield the best model performance. It is essential to select the appropriate number of folds (K) when implementing K-fold CV on any problem. The number of folds is directly proportional to the variance and inversely proportional to the bias in the error estimates. The default value of the "K" parameter in GridSearchCV per Scikit-learn is five. This means five folds of CV occurs with each grid search iteration. We changed the "K" to 10 as a 10-fold CV is reputed to perform well in practice and is not computationally intensive. We selected the respective hyperparameter for each model based on its magnitude of impact on model performance. We deeply motivate our selection when we explain the relationship between the finetuned hyperparameter and the respective model. In the table below, we state the finetuned hyperparameter for each model as per the Scikit-learn library.

Model	**Finetuned hyperparameter**	**Grid**
KNN	n_neighbors	[1, 3, 5...33]
Logistic Regression	C (inverse of regularization strength)	[1, 5, 10...500]
SVC	C (inverse of regularization strength)	[1, 5, 10....500]
Decision tree classifier	max_depth	[3,5,7,9]
Random Forest	max_depth	[3,5,7,9]

Table 5.1: Finetuned hyperparamters for each model

It is important to note that the upper bounds of the grid for each model's hyperparameter were intended to be large values that were nowhere near the optimal value. We wanted to track how the CV performs over a large grid size. We set the upper bound of the KNN model using the following heuristic. The square root of the number of images in the training set is the ideal value for n_neighbors [22]. We doubled this value and set it as our upper bound for the grid of the n_neighbors hyperparameter. Traditionally, nearly all decision trees and random forest models for binary classification problems similar to our dataset size have a max depth between 3-7. Hence we set the upper bound to nine for the grids of these two models. We explain the details, impact, and motivation for the selection of each hyperparameter in the respective model's section.

5.9 Setting the seed

Randomness is very prominent in machine learning models. Non-determinism can impact results stability, model convergence rate, and quality. Therefore despite using previous input and script, reproducing exact results is a challenging task. Randomness can appear in many phases of machine learning. Upsampling data in data preprocessing, random data splitting in K-fold CV, and random weight initialization can cause randomness.

Seed Nr	**Seed value**
1	12243
2	24323
3	16787

Table 5.2: The set seeds and their respective value in each model

Computers are deterministic and do not generate actual random values. Therefore introducing and setting a "seed" in our pipeline can lead to output reproducibility. The seed is a random number/vector used to initialize a pseudorandom number generator. The seed only affects random values. In our study, we set a random seed to the weight initialization parameter in each model. We repeated this process thrice with a random seeds for each cycle. We then averaged the result. We were aware that this approach would not be too practical on the KNN model as there is no training period which we explain in section 5.12.1.

5.10 Training machine learning models

We adopted a consistent approach to training and evaluating the models. After data preprocessing, test/train split, and performing K-fold cross-validation(except for the pre-trained ResNet-50), we began the training phase of our models. In the scope of our binary classification problem, training a model infers teaching the model to classify tongue images based on identifying distinguishable features. The model is explicitly told the correct output for every input. If the model produces the wrong results, it adjusts its parameters to provide better results in the next round of training. Although each model has unique characteristics and approaches, the "supervised" training style remains constant.

5.11 Logistic Regression

Logistic regression models are typically used in machine learning for solving binary classification problems. It returns the probability of a particular event taking place by construct-

ing the event's logarithm of the odds as a linear combination of the independent variables[2]. We used a logistic regression classifier for our binary multivariable classification problem.

5.11.1 Motivation for using the logistic regression classifier

Unlike linear regression, logistic regression provides discrete output. Logistic regression classifiers are well known for their impressive performance in binary classification problems. Moreover, logistic regression models have historically proven to be computationally efficient to train with a straightforward implementation [11].

5.11.2 Important characteristics of the logistic regression classifier

- The basic assumptions logistic regression models are the absence of multicollinearity, independence of errors, lack of strongly influential outliers, linearity in the logit for continuous variables, and independence of errors.
- These models are generally not sensitive to outliers because the sigmoid function tapers the outliers.
- Logistic regression classifiers are effective only on linearly separable data. This limitation of the classifier leads to poor performance on non-linearly separable data.

5.11.3 Sigmoid function

The y-coordinate of the sigmoid function bounds any value to a probability as it ranges from 0 and 1. Binary classification problems expect a binary output and not a probability.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.1}$$

where:
$\sigma(x) \in [0, 1]$ = Probability $x \in \mathbb{R}$ = input

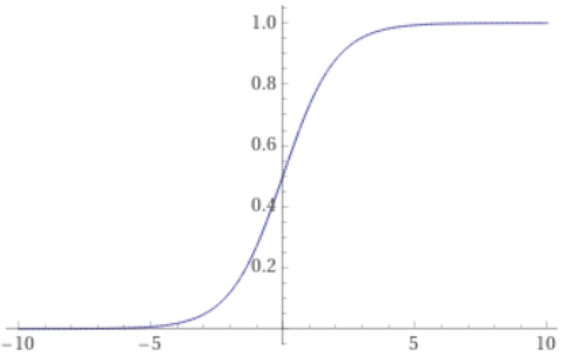


Figure 5.6: Graphical representation of the sigmoid function

Therefore by setting a threshold, we can map the resultant probability of the sigmoid function to a binary output. The standard threshold in classification problems is 50%. In the case of diagnostic imaging, images that produce a probability more significant than the threshold are classified as infected. In contrast, ones that fall under the threshold are classified as healthy. We can represent the threshold with y being the final prediction.

$$y \in [0, 1] \quad y = \begin{cases} 1 & \sigma(x) \geq threshold \\ 0 & \sigma(x) < threshold \end{cases} \tag{5.2}$$

[2]We represent dependant variables/features as x and the predictions/ response variables as y

Altering the threshold depends on the application. If the model is sensitive to false positives, we increase the threshold, whereas models with a high recall tend to have a lower threshold. Logistic regression uses this combination of a preset threshold and sigmoid function to classify data.

5.11.4 Decision boundary

Decision boundaries are boundaries used by the model to make decisions. The area enclosed by these decision boundaries is known as the decision region. In binary classification problems, graphically represented data points that fall on the side of the decision boundary are classified into one class. In contrast, the data points that fall on the other side are classified into the second class. We can represent the decision boundary for the multi-dimension logistic regression model as:

$$b + w^{\mathbf{T}} x = 0 \tag{5.3}$$

where:

1. b is the intercept bias term.
2. $w^{\mathbf{T}}$ is the transpose of the weight array that consists of the weights of the independent variable "x".

5.11.5 Training a multidimensional logistic regression classifier

The logistic regression classifier uses the logistic sigmoid function to perform binary classification to the 0.5 (50%) threshold set. Images are then mapped to a class based on where they lie on the decision boundary. Our logistic regression classifier uses the popular L-BFGS optimization algorithm ("Limited memory Broyden Fletcher Goldfarb Shanno") [6]. The L-BFGS algorithm enables the model to train by iteratively finetuning the weights and bias.

$$\sigma(b + w^{\mathbf{T}} x) = \frac{1}{1 + e^{-(b + w^{\mathbf{T}} x)}} \qquad [x, w \in \mathbb{R}^d] \tag{5.4}$$

where:

1. d is the input dimension.

The above process is done automatically using the *sklearn.linear_model.LogisticRegression* class in the scikit-learn library. We finetune the "C" hyperparameter based on the cross-validation results. The "C" parameter is the inverse of the model's regularisation strength. Regularization is crucial as it prevents the model from overfitting by penalizing extreme parameters and outlier values in the training data. Low values of "C" leads to a tradeoff between model complexity and cost of data fitting. High values of "C" reflects real-world data by giving high weight to training data.

After the training phase, the model classified the unlabelled images in the test set. We evaluated the model's objective performance using the selected evaluation metrics.

5.12 K-Nearest Neighbors (KNN)

K-Nearest neighbors (KNN) is a non-parametric supervised learning algorithm that can be used on regression and classification problems. In this paper, we focus on the KNN classifier. The "K" is the number of nearest neighbors considered for every image classification. The model classifies an image by:

1. Finding the class with the most number of neighboring images classified to it.

2. The image is then classified to the most common class among the K nearest neighbors.

5.12.1 Lazy learner

KNN is a unique supervised learning algorithm known not to require training on the dataset. While most supervised learning models make predictions on the test set by training on the training set, KNN does not require the traditional machine learning training phase. Instead, the model trains by sorting the entire image dataset to find the K nearest neighbors at each iteration. Therefore KNN makes accurate predictions by analyzing image data similarities/relations and refined distance metrics.

KNN is coined as "Lazy learner" as it does not learn/formulate a discriminative function from the training set but memorizes the training dataset instead. A common misconception is that KNN becomes computationally cheaper with no training phase. Although there may be an absence of a training phase in KNNs, the prediction phase is computationally expensive. The model searches through the entire dataset for the K nearest neighbors of every prediction. Without a training phase and no optimization function fitted to the data, KNN does not have a loss function that can be minimized during "training".

5.12.2 Euclidean distance

KNN models require a distance function to calculate the distance between the image data to find the nearest neighbors. There are several types of distance functions. In this paper, we focus on the Euclidian distance function as it is used in the *KNeighborsClassifier* class of the *sci-kit learn* library. The Euclidean distance is the Pythagorean distance between the respective Cartesian coordinates of two plotted points representing an image in the dataset.

$$Euclidean\ distance(p, q) = \sqrt{\sum_{i-1}^{d}(q_i - p_i)^2} \tag{5.5}$$

where:

1. p is a point with Cartesian coordinates (p_1, p_2 ... p_d).

2. q is a point with Cartesian coordinates (q_1, q_2 ... q_d).

3. d is the input dimension.

5.12.3 Curse of dimensionality

KNNs require large amounts of data to handle problems that involve numerous features. Hence there is a high risk of overfitting as it is hard to distinguish between noise and actual data. KNN performs the best with low dimensionality (as demonstrated in a study by Gu and Shao in 2014[3]). In high-dimensional data, multiple points are often arranged in a circular shape with the query point at the center. Therefore the distance from the query point to all arranged data points is nearly identical. The phenomenon when the calculated Euclidean distance is incompetent under high dimensional data due to nearly equidistant vectors to the search query vector is known as the curse of dimensionality.

[3]As of 8/20/2022 this study is no longer publicly available

5.12.4 Choosing the K value

We must select the correct K that fits our image data when implementing a KNN on a classification problem. Low values of K lead to weak predictions and a decrease in accuracy. The precision and recall become more sensitive to outliers. Hence there is a high chance of overfitting. However, high values of K may lead to underfitting.

We used two variants of KNN with differing K values. The first KNN model had a K of 1. There was no form of cross-validation and hyperparameter finetuning prior to assigning the K for this model. We anticipated sub-optimal performance from this KNN. However, we wanted to showcase the impact of selecting the appropriate K value on performance. The second variant of the KNN had a K value of 15. We obtained this value by finetuning the "n_neighbors" parameter by 10-fold GridSearchCV as explained in section 5.8.

5.12.5 Working of model

The following pseudocode represents the inner working of a KNN classifier. We load in the datapoints and set K to a positive integer value.

Algorithm 1 K Nearest Neighbors

Input: $k > 0$, Loaded Data
Output: Predicted class
  Distances [ ]
  $K - Neighbors$ [ ]
  $size = Number\ of\ datapoints$
  for 1 to *size* :
    Distances [ ] .insert (Euclidean Distance (test data, **for each:** training data row))
      ▷*Euclidean distance between test data and each row of training data.*
  end for
  Sort.Distances[ ]
      ▷*Sorts the distances in ascending order*
  K-Neighbors [ ] = Top K rows(*Distances* [ ])
      ▷*Returns the top K rows in the sorted distances array.*
  Predicted class = **Mode** (K-Neighbors [ ])
      ▷*Returns the most frequent class of these rows.*
return Predicted class

5.13 Support Vector Classifier (SVC)

Support Vector Machines (SVMs) perform the classification by constructing a hyperplane such that all points of infected images are on one side of the hyperplane, whereas all healthy images are on the other side. Support Vector Classifiers (SVCs) are an extension of maximal margin classifiers. SVMs may use linear non-linear approaches to classify data, whereas SVCs use linear approaches. While there could be multiple such hyperplanes, SVC tries to find the one that best separates the two classes that maximizes the distance to points in either class. This distance is called the margin, and the points that fall exactly on the margin are the supporting vectors. SVC solves a convex optimization problem that maximizes this margin where the points of each class should be on the correct side of the hyperplane.

5.13.1 Motivation of using SVCs

SVCs are known to be versatile, easy to interpret, and robust. However, our core reason for implementing SVCs in this study is their impressive performance on smaller datasets. The generalization error in SVCs is greater than or equal to the dataset's ratio of support vectors to samples. Therefore, a smaller dataset leads to a more generalizable SVC model, producing better results.

5.13.2 Hyperplane

A hyperplane is a flat affine subspace of dimension $n-1$ in an n-dimensional search space. The hyperplane is used to segregate data points from different classes. Machine learning models such as SVCs make their predictions using a hyperplane. We can mathematically define a hyperplane in an n-dimensional search space with the following equation:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 \cdots + \beta_n x_n = 0 \tag{5.6}$$

where:

1. n is the input dimension.
2. $\beta_0, \beta_1 ... \beta_n$, are the parameters.
3. x is a data point, with coordinate vector: $\{x_1, x_2 .... x_n\}^T$

If the above equation is satisfied, the hyperplane lies precisely between the two classes. However, if one of the following inequalities is satisfied, then point x_i is classified as the class of the respective side of the hyperplane.

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 .... + \beta_n x_n > 0 \tag{5.7}$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 .... + \beta_n x_n < 0 \tag{5.8}$$

5.13.3 Soft margin

SVCs tend to misclassify a few training images with the hope of a better performance in classifying the remaining images. Therefore they are termed soft margin classifiers. The margin is "soft" and relatively smaller as the model allows some images to be classified on the wrong side of the margin. SVC leverages support vectors to make its predictions. Support vectors are observations that lie on the margin or the wrong side.

5.13.4 The "C" hyperparameter

SVMs introduce "slack variables." These variables permit each and every observation to be on the wrong side of the margin or hyperplane. Slack acts as a penalty for incorrect classifications. Observations classified correctly have a slack of 0, and observations classified incorrectly have a positive slack. Moreover, each slack variable informs the model of the relative location of the respective observation from the margin and hyperplane. They allow individual observations to be on the wrong side of the margin or the hyperplane.

Slack variables are regulated by the hyperparameter "C." If "C" is 0, the classifier is not penalized by slack. Therefore the optimization can afford to use in the entire search space. Hence large misclassifications would be acceptable. However, the decision boundary itself would be linear, which may lead to underfitting. A high value of "C" infers that even a small slack is highly penalized, so the classifier cannot afford to misclassify a data point. This may lead to very complex decision boundaries and hence overfitting. Additionally, the "C" hyperparameter regulates the bias-variance tradeoff in

the SVC. High values of "C" lead to wider margins. This results in a greater number of support vectors as observations on the wrong side of the margin/hyperplane. This leads to low variance and high bias. However, lower values of "C" result in narrow margins and a lesser number of support vectors. Therefore lower values of "C" leads to a low bias but high variance. Hence it is crucial to choose an appropriate "C". We set the value of "C" by finetuning the "C" hyperparameter by 10-fold cross-validation as explained in section 5.8.

5.13.5 Kernelization

Linear SVCs can address linearly separable problems by finding the best fitting hyperplane. However, in most real-world situations, data is not linearly separable. Therefore, this approach does not work. Instead, we use the "kernel trick" to transform lower dimension input space into a higher one. The model uses the same feature space. However, it does not compute the coordinates of higher dimensional data. Transforming lower dimensional data to a high one can make non-linearly separable data linearly separable. The model can then find the best fitting hyperplane to begin the classification.

RBF Kernel

There are several variants of kernels that are used to perform kernelization. In this study, we experimented with an SVC having the standard linear kernel (no kernel trick is done here) and an SVC with a Radial Basis Function kernel (RBF). The RBF kernel is widely used due to its similarity with the Gaussian distribution. The RBF kernel performs kernelization by computing the similarity between observations in the test set. The RBF kernel can be mathematically defined with the following equation:

$$k(x_m, x_n) = \exp\left(-\frac{\|x_m - x_n\|^2}{2\sigma^2}\right) \tag{5.9}$$

where:

1. x_m and x_n are two observations
2. $||x_m - x_n||$ is the Euclidean distance between the two observations
3. σ^2 is a variance hyperparameter that can be either set by hand or automatically computed through a predefined rule of thumb.

5.13.6 Training an SVC

Using the concepts we have explained in the previous section, we explain the SVC's training process on our dataset. The goal of SVC is to maximize the width of the margin. The classifier uses support vectors to construct decision boundaries. Every training image is classified based on its distance from the support vector.

We can represent the hyperplane equation in equation 5.8 as a dot product of two vectors:

$$\vec{w} = [\beta_0, \beta_1..., \beta_n],\ \vec{x} = [x_0, x_1..., x_n] \tag{5.10}$$

where:

1. $\vec{w}$ is the perpendicular vector to the straight line
2. $\vec{x}$ is the point on the straight line

We want to find: $\vec{w} \cdot \vec{x} + b = 0$

Defining the decision rule

For the sake of clarity we abbreviate infected images as "pos" and healthy images as "neg". We define the support vectors as:

$$\vec{w} \cdot \vec{x}_{pos} + b = 1 \quad or \quad \vec{w} \cdot \vec{x}_{neg} + b = 1 \tag{5.11}$$

We represent the remaining images as:

$$\vec{w} \cdot \vec{x}_{pos} + b > 1 \quad or \quad \vec{w} \cdot \vec{x}_{neg} + b < 1 \tag{5.12}$$

Defining the margin

The width of the margin is:

$$\frac{(\vec{x}_{pos} - \vec{x}_{neg}) \cdot \vec{w})}{\|w\|} \tag{5.13}$$

Since equation 5.13 equals 1 we can rewrite it as:

$$\frac{2}{\|w\|} \tag{5.14}$$

Finding the margin which is maximally equidistant between the two classes is equal to minimizing the following formula.

$$\frac{1}{2} \|w\|^2 \tag{5.15}$$

Optimizing the goal function

SVCs classify by maximising the margin such that they can construct by learning a appropriate decision boundary/separating hyperplane. However this is constrained optimization problem as are preforming binary classification. SVCs optimize equation 5.15 by using the "Lagrangian transform" [63]. From the "Lagrangian transform" we get two "Lagrangian multipliers" that aim to turn a constrained optimization problem into an unconstrained optimization problem. There we are able to maximize the margin without any constraints. In the case of RBF SVC, we perform kernelization and increase the dimensionality to solve $\phi(x)$ in equation 5.16.

Predictions

Finally, SVM uses the following equation to make predictions:

$$w^T \phi(x) = \sum_n \alpha_i y_n k(x_n, x) \tag{5.16}$$

5.14 Decision tree classifier

Decision trees are a non-parametric supervised learning algorithm used for classification and regression tasks. This model incorporates a tree structure that is organized hierarchically. They consist of a root node, branches, internal nodes, and leaf nodes. Branches represent potential alternative class predictions, whereas the leaf nodes represent the actual class. Besides the leaf nodes in a binary decision tree, all nodes have two branches that signify a positive or negative response. Tree-based decision trees stratify the search space into smaller decision regions. Predictions for observations are made based on either the mean or the model of training observations for a given class. This paper examines a binary decision tree classifier that aims to predict a qualitative response.

5.14.1 Motivation for using decision trees

Decision tree classifiers widen our domain of tested supervised machine learning models by introducing agile tree-based methods. These modes are known for their simplicity. Decision trees are trivial to interpret and implement as they mirror the basic human decision process through tree structures. Unlike most classification models, decision trees efficiently manage qualitative predictors without the need to create indicator variables.

5.14.2 Gini impurity

The Gini index represents the purity of the dataset. In a decision tree classifier, the Gini impurity is the probability that the classifier missclasifies a random observation. Therefore decision trees with a low Gini impurity are desired. We calculate the Gini impurity by subtracting the sum of squared probabilities of one class from another. We can mathematically represent the Gini impurity of the dataset "D" in our binary classification problem as follows:

$$G(D) = \sum_{i=1}^{C} p(i) * (1 - p(i)) \tag{5.17}$$

where:

1. i is a class (C) which is 0 for healthy and 1 for infected.

2. $p(i)$ is the probability of training observations being classified to class i.

The amount of impurity removed with each split is calculated by deducting the Gini impurity of the entire dataset from the weight of a branch's impurity. This resultant value is termed the Gini gain. A higher Gini gain infers a better split.

5.14.3 Node purity

Node purity is a statistic used to determine whether we split at each node. When a node's data is evenly split, it is 100% impure. In the situation where all of its data is from a single class, it is 100% pure. The Gini impurity is inversely proportional to node purity. Therefore model optimization is done by reducing impurity and increasing node purity.

5.14.4 Max Depth hyperparameter

The max depth in a binary decision tree classifier is the number of nodes from the root down to the furthest leaf node. Its depth signifies the number of splits the tree makes before reaching the prediction. The *sklearn.tree.DecisionTreeClassifier,* class in the scikit-learn library, sets the max depth of the decision tree to "None." This means nodes are expanded until all leaves are pure or until all leaves contain less than the minimum number of samples required to split an internal node (this is 2 for binary classification problems). It is essential that we bound the max depth to prevent overfitting. We finetune this hyperparameter using a ten-fold GridSearchCV as explained in section 5.8. Theoretically, the max depth of a decision tree classifier can be $n - 1$, where n is the number of training samples. However, this is never the case as the model will undoubtedly overfit. Based on similar binary classification problems of similar dataset sizes, we search for the optimal max depth between 3 and 9.

5.14.5 Training a decision tree classifier

Training a decision tree classifier consists of two phases:

- Induction: Building the decision tree
- Pruning: Removing redundant sections of the tree.

Induction

A decision tree classifier makes predictions by learning decision rules based on the training data. During the training phase, the model adopts an inductive inference learning style. It makes inductions based on the training data and creates a decision tree adhering to the induced decision rules. The inner working of the inductive learning phase of our decision tree classifier is as follows:

Algorithm 2 Decision tree classifier training

Input: labelled loaded training data "X", max_depth)
Output: Decision tree
  while current depth is not greater than max_depth all partitions are processed **do**
    maxGain $\Longleftarrow 0$
    splitA $\Longleftarrow$ *null*
      for all possible splits "s" in X **do**
        $G_{gain} \leftarrow$ *Gini gain*(s) ▷ *The highest Gini gain gives the best split*
        if $G_{gain} >$ maxGini **then**
          maxGain $\leftarrow G_{gain}$
          splitA $\leftarrow s$
        end if
      end for
      Partition(X, splitA)
  end while

Pruning

Pruning is a data compression approach that decreases the size of decision trees by deleting redundant parts of the tree that do not impact the model's classifying performance. Pruning lowers the final classifier's complexity, which increases predicted accuracy by reducing the risk of overfitting.

In this study, we used a popular bottom-up pruning approach known as minimum cost complexity pruning [37]. The technique uses a complexity parameter $\alpha_i > 0$, a cost complexity measure $R_\alpha(T)$ and a leaf node misclassification rate $R(T)$ where T is the number of leaf nodes.

$$R_\alpha(T) = R(T) + \alpha|T| \tag{5.18}$$

At each iteration, the sub-tree that minimizes $R_\alpha(T)$ is pruned. We are then left with a pruned final tree.

5.14.6 Shortcomings of decision trees

Despite their simplicity, decision trees do not have the same level of accuracy and performance as other mainstream machine learning models. Decision trees are deterministic and extremely greedy. Hence decision tree classifiers tend to overfit when fed in new data. These models can turn out to be non-robust. We accounted for these drawbacks and hence experimented with an approach that is an advantageous extension of decision trees which is explained in the following section.

5.15 Random Forest

Ensemble methods are a group of machine learning techniques that combine similar model predictions to compute their predictions. Random forests are an ensemble learning method for tasks that require constructing multiple decision trees at training time. Random forests

predict and classify an observation by selecting the most occurring predicted class from a set of individual decision trees. They use two critical optimization techniques known as bagging and boosting to make predictions. We explain these techniques in the following sections.

5.15.1 Motivation for using random forests

Random forests are more robust than decision trees. They aggregate the predictions between multiple decision trees to counter overfitting. This approach reduces errors caused by bias leading to higher accuracy. Although random forests are relatively more complex, they introduce a new perspective in dealing with classification problems. In theory, combining several base models makes random forests resistant to outliers and corrupt data. This is a critical aspect that automated diagnostic assistants are expected to have.

5.15.2 Bootstrapping

Bootstrapping is a resampling technique for accurately determining a statistic. In an image classification scenario, this strategy involves randomly resampling images in the dataset without any replacement. These statistics are aggregate quantities that cannot be directly observed (ex: standard deviation, mean). Bootstrapping on an image classification dataset is done as follows.

1. Generate "B" sub-datasets [4] from the original image dataset X=$\{x_1, x_2...x_n\}$.
2. Calculate the mean of class instances from each sub-dataset.
3. Calculate the mean of the means from step 2.

5.15.3 Bagging

Bagging or bootstrap aggregation reduces variance in statistical learning models by aggregating the predictions over multiple decision trees. In image classification tasks, bagging works as follows.

1. The training set of images X=$\{x_1, x_2...x_n\}$ are divided into "B" sub training sets.
2. Each training set is used to train a decision tree. We can denote the prediction of each tree as $f_1(X), f_2(X)...f_n(X)$.
3. The average prediction of all decision trees is the final prediction. This is a simple way to reduce the variance and increase the prediction accuracy of the model.

Bagging can be mathematically represented as follows:

$$f_{bag}(X) = \frac{1}{B}\sum_{b=1}^{B} f_b(X) \tag{5.19}$$

5.15.4 Random forest classifier

In a traditional bag decision tree, every node analyses all "n" features of training data and chooses one feature and a threshold value where the split is optimal. Therefore a split point is chosen. On the other hand, random forest classifiers only look at a subset of the "n" features to choose the split point. A common heuristic followed in random forest classification problems is the number of features a node considers equates to the square root of the number of total features [28]. The binary classification problem in this study consists of nine features. Therefore each node in our random forest classifier decides its

[4] Overlapping sub-datasets are allowed

split point by considering three randomly selected features as three is the square root of 9. By implementing this heuristic, we reduced training time and increased performance. The inner working of random forest classifiers is as follows:

1. Splits the data into overlapping sub-datasets
2. Trains each sub-dataset against a decision tree.
3. Determines predictions based on the mode of predictions from the decision trees.

Random forest classifiers are an enhanced version of the bagged decision trees. They can decorrelate the decision tree with minor parameter adjustments. The main difference is the number of parameters required to construct their structures. Bagged decision trees can be constructed from one parameter. Random forests require only two parameters. The first is the number of trees. The number of trees equals the number of sub-datasets constructed from the total dataset. Additionally, random forests require each node's number of considered features to search for the optimal split point. These are all the "n" features for bagged decision trees.

5.16 Residual Networks

Residual neural networks (ResNet) are a variant of convolutional neural networks designed for computer vision tasks. In 2015, the ResNet model won the prestigious large-scale image classification and detection competition ImageNet [44]. ResNets consist of residual blocks. These blocks are stacks of layers oriented in a manner that allows adding the output of one layer to another deeper layer within the respective residual block. However, adding layers can increase the complexity and expressiveness of the network. ResNets overcome this hurdle by introducing residual connections. These connections act as shortcuts that allow the network to skip selected residual blocks. Residual connections facilitate information flow across many layers ("information highways") as inputs can forward propagate faster through the residual connections across layers. These connections are advantageous as they allow the ResNet to mitigate degradation/accuracy saturation as adding more layers to a deep learning model leads to higher training error.

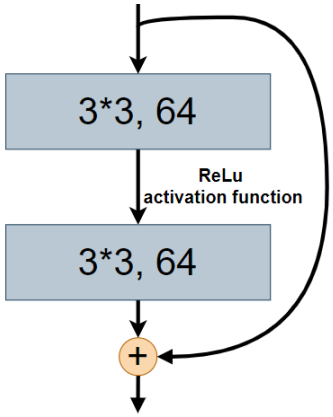


Figure 5.7: Residual block in a ResNet. The residual connection shown in the figure skips two layers of 64 3*3 kernels each.

Identity mapping in residual blocks is done using the residual function: $y = f(x * w + x)$. Information can move freely across the entire network by modeling residual functions as identity mappings. The residual function makes it possible to represent any layer as a function of the initial input. Additionally, residual connections optimize ResNets by avoiding the vanishing gradient problem as they allow the passing of gradient information through the layers and make it easier to fit the identity function.

5.17 Pre-trained ResNet-50

The ResNet-50 is a pre-trained state-of-the-art residual network that is fifty layers deep. It was trained on the ImageNet dataset, which consists of 1,281,167 training images and 50,000 testing images that seek to be classified between 1,000 classes.

5.17.1 Motivation for using a pre-trained ResNet 50

ResNet models are considered the "backbone" of several computer vision tasks [21]. They have been extensively trained on the ImageNet dataset and can efficiently handle the vanishing gradient problem. We selected a pre-trained ResNet-50 over other ResNet variants as we believe it provides a balance between computational cost and analysis depth. After reflecting on other studies that show promise of desirable performance of ResNet-50 on small-scale datasets[57, 32, 1], we felt it was the right fit for our dataset size.

5.17.2 Structure of ResNet-50

The traditional ResNet-50 uses a softmax activation function in the output layer. However, our output layer consists of one neuron with a sigmoid activation function. We believe a sigmoid activation function is a better fit for our binary classification problem as it asymptotes the output range of 0 to 1. The structure of ResNet-50 is as follows:

Residual block	**Number of layers**	**Output size in pixels**	**Kernel size**
Convolution 1	1	112 *112	$\begin{bmatrix} 7*7, 64 \end{bmatrix}$
Max Pooling	0	56*56	$\begin{bmatrix} 3*3 \end{bmatrix}$
Convolution 2	9	56*56	$\begin{bmatrix} 1*1, 64 \\ 3*3, 64 \\ 1*1, 256 \end{bmatrix}$ *3
Convolution 3	12	28*28	$\begin{bmatrix} 1*1, 128 \\ 3*3, 128 \\ 1*1, 512 \end{bmatrix}$ *4
Convolution 4	18	14*14	$\begin{bmatrix} 1*1, 256 \\ 3*3, 256 \\ 1*1, 1024 \end{bmatrix}$ *6
Convolution 5	9	7*7	$\begin{bmatrix} 1*1, 512 \\ 3*3, 512 \\ 1*1, 2048 \end{bmatrix}$ *3
Average pooling	0	1*1	$\begin{bmatrix} 3*3 \end{bmatrix}$
Output Layer	1	1*1	No kernel

Table 5.3: Structure of ResNet-50

It might be challenging to interpret the configuration of each residual block from the table above. Hence we explain the working of one of the residual blocks. The second convolution residual block consists of 64 kernels of size 1 * 1, 64 kernels of size 3 * 3, and 256 kernels of size 1 * 1. Each of these kernel sizes is implemented in its layer resulting in three layers. These three layers are repeated thrice. Therefore we have nine layers in the second convolution residual blocks.

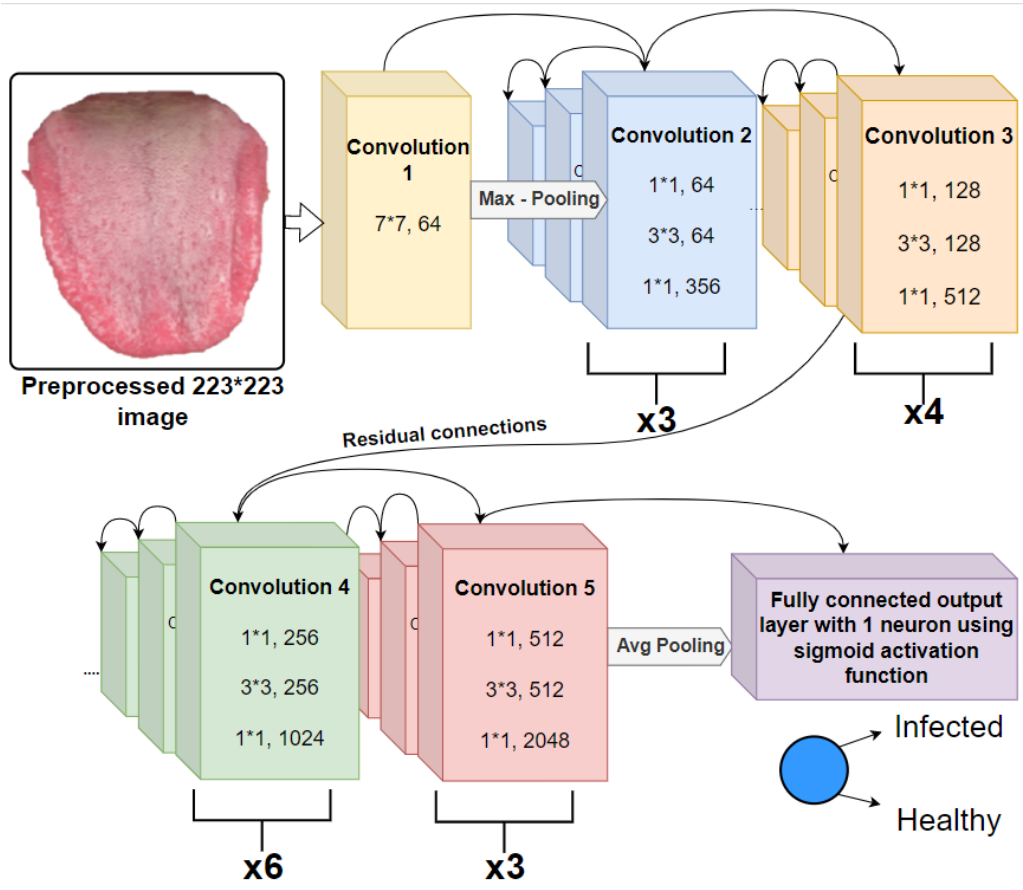


Figure 5.8: Diagram representation of the ResNet-50 architecture

Stride and padding

Stride denotes pixels the kernel moves in each step of the convolution. Selecting the appropriate stride is computationally essential as it can reduce spatial resolution and reduce overlapping of receptive fields. We stuck to the default stride of a ResNet-50 which is 2.

Padding is applied to the outer frame of the image. The filer covers more area on padded images. This practice leads to a more accurate analysis of images by the CNN. We stuck to the default padding of a ResNet-50 which is 2.

The output shape $n_h * n_w$ with a stride height s_h, stride width s_w, padding width p_w and padding height p_h can be represented as:

$$\left\lfloor \frac{n_h - k_h + p_h + s_h}{s_h} \right\rfloor * \left\lfloor \frac{n_w - k_w + p_w + s_w}{s_w} \right\rfloor \tag{5.20}$$

where:

1. k_h = kernel height
2. k_w = kernel width
3. n_h = kernel height

5.17.3 Freezing layers

Freezing layers in a deep neural network is a transfer learning technique that prevents weights from being modified. Freezing is an efficient way to increase training speed without minimal effect on accuracy. We froze all layers besides the last two layers. With

computational resource constraints in place, this technique helped fasten the training process.

5.17.4 Working of a pretrained ResNet-50

After we loaded the preprocessed image dataset into the pre-trained ResNet-50, the network passed the input layer to subsequent layer per the ResNet-50 architecture. It performed the respective convolution and pooling operations till we reached the final layer. This process is termed forward propagation. The network then "backpropagates" using an optimization algorithm that updates the weights enabling the network to "learn." Backpropagation reuses the stored intermediate values from forwarding propagation to avoid duplicate calculations. We repeated this training process for 100 epochs. The network was then evaluated based on its performance on the training set. In this section, we provide a detailed insight into the training process of the ResNet-50.

5.17.5 Max and average pooling layer

This pooling layer has only one kernel. The kernel slides over the input, and it chooses the maximum value among the covered ones for each channel and each slide. If we have an input of "c" channels then the output also has "c" channels. Average pooling is a deterministic pooling operation with no parameters and is used to calculate the average value of the elements in the pooling window.

5.17.6 Network optimization

This section explains how the pretrained ResNet-50 optimizes using stochastic gradient descent. We first show the inner working of gradient descent.

Algorithm 3 Gradient decent

for all $e \in \{0, 1, \dots \mathcal{E}\}$
$\hat{y} = f(X)$
$E(f) = \text{BCE}(\hat{y}, y)$
$w^{e+1} = w^e - \eta \nabla_w E(f)$

- f is a classification model that outputs the predictions $\hat{y}$, in our case it is the ResNet-50 model.
- $E(f)$ is the Binary Cross-Entropy (BCE) loss. A high BCE leads to poor predictions. Our objective consists of minimizing it bringing it as close as possible to 0.
- $\nabla_w E(f)$ is the gradient, that is a vector containing the derivatives of $E(f)$ for all w_j.
- η is the learning rate hyperparameter that scales the gradient updates.

Using the gradient descent we get the stochastic gradient descent algorithm as follows:

Algorithm 4 Stochastic gradient descent

for all $e \in \{0, 1, \dots, \mathcal{E}\}$
Generate a random partition of the dataset $X : B(X) = B_1, \dots, B_b$
for all $k \in \{0, 1, \dots, b\}$
Perform one step of gradient descent with B_k instead of X

B_k is a mini-batch. With b mini-batches we can perform b weights updates in one epoch. Based on the size of the mini-batch we have three subtypes of gradient descent:

1. $b = 1$: corresponds to the original case of gradient descent
2. $b = |X|$: we use only one observation per update. This method is called (Pure) Stochastic Gradient Descent (SGD), but has the disadvantage that the weights updates become sensitive to the noise present in a single image
3. $1 < b < |X|$: we use a mini-batch of observations, neither too small, nor too large.

5.17.7 Batch normalisation

Through normalizing the inputs to the layers by re-centering and re-scaling, we used batch normalization to make artificial neural networks faster and more stable. Batch normalization is applied to specific layers[5] and operates as follows

1. In the initial step of each training cycle, we normalize the inputs (of batch normalization) by dividing along their standard deviation and subtracting their mean, which is determined based on the statistics of the current minibatch.
2. Next, a scaling coefficient and scale offset are applied.

$$x' = \frac{x - E[x]}{\sqrt{var(x)}} \tag{5.21}$$

where:

1. x' is the new value of a single image.
2. E[x] is its mean within a batch
3. var(x) is its variance within a batch.

From 5.25 we get the following equation where γ and β are learned per layer and make sure that batch normalisation can learn the identity function.

$$x'' = \gamma * x' + \beta \tag{5.22}$$

Regularization is the main positive side effect of batch normalization. When the input distribution to our network changes, an internal covariate shift occurs. Hidden layers attempt to learn how to adjust to the new input distribution when it changes. This leads to a longer training time. The pre-trained ResNet-50 uses batch normalization to reduce the speed up the training process. We set each mini batch to a size of six images.

5.17.8 ADAM

In this paper, we used the ADAM optimization algorithm. ADAM adapts to the previous history of gradients obtained from "stochastic gradient descent". It combines the best properties of the AdaGrad and RMSProp optimization algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. ADAM is known to have a quick computation time[6] and fewer parameters for tuning, making it a convenient choice for our diagnostic imaging use case [22]. The working of ADAM is as follows:

ADAM uses two state variables:

[5] Batch normalization can also be applied to all layers

[6] According to [65] SGD yields better results than ADAM over an extensive number of epochs.

1. $v_t \rightarrow \beta_1 v_{t-1} + (1 - \beta_1) g_t$
2. $s_t \rightarrow \beta_2 s_{t-1} + (1 - \beta_1) {g^2}_t$

β_1 and β_2 are nonnegative parameters which we set to 0.9 and and 0.99. g_t is the gradient w.r.t. stochastic objective at timestep t. s_t is the update biased first moment estimate. v_t is the update biased second raw moment estimate at time stamp t. η is the learning rate. Since binary classification problems typically benefit a significant amount of bias towards smaller values, we set v_0 and s_0 to 0. This is a common initialisation when using ADAM [67]. We renormalise to:

$$\hat{v}_t = \frac{v_t}{1 - {\beta^t}_1}, \hat{s}_t = \frac{s_t}{1 - {\beta^t}_2} \tag{5.23}$$

We then rescale the gradient to get:

$$g'_t = \frac{\eta \hat{v}_t}{\hat{s}_t + \in} \tag{5.24}$$

ADAM optimizes by using the following update function:

$$x_t \rightarrow x_{t-1} - g'_t \tag{5.25}$$

This process repeats until convergence is met.

5.18 Evaluating machine learning models

After the machine learning model finishes training on the training dataset, it attempts to classify the unlabelled test/validation dataset. The model has never encountered this dataset before and attempts to classify the images by finding patterns and using classification rules it has learned from the training phase. Based on how well the model classifies on the test set, we can evaluate the performance of the entire model on the image classification problem. In this paper, we use the following evaluation metrics for every model:

- Confusion matrix.
- Average accuracy.
- Precision, recall and F1 score with their macro and weighted average between the two classes.
- ROC cruve and AUC.

5.18.1 Statistical tests

In diagnostic imaging, models are evaluated based on their recall and specificity. In this study, we encountered models that have identical said parameters. In order to distinguish and rank models accordingly, we compared the performance of the best-trained classifiers using the non-parametric McNemar's statistical test. Counting the number of agreements and disputes among the trained classifiers, as shown in table 5.4, is the first step in using McNemar's test.

	Model 1: Correct	**Model 2: Wrong**
Model 2: Correct	+/+	+/-
Model 2: Wrong	-/+	-/-

Table 5.4: Statistical distribution between two classifiers

The second stage is to determine whether the error rates of the two classifiers are comparable. The following formula computes the chi-square value and compares it to the theoretical chi-square with one degree of freedom.

$$X^{-2} = \frac{\left|(\eta_{(-/+)} - \eta_{(+/-)}\right| - 1)^2}{\left|\eta_{(+/-)} - \eta_{(+/+)}\right|} \tag{5.26}$$

Each η value checks the marginal homogeneity of two dichotomous variables. Assume we have two variables. The first is the number of common positives and negatives from model 1 and model 2 respectively. The second variable is the number of common negatives and positives from model 1 and model 2 respectively. Therefore $\eta_{(-/+)} - \eta_{(+/-)}$ checks the marginal homogeneity between the two variables.

5.18.2 Bonferroni correction

The Bonferroni correction is applied to P values when numerous dependent or independent statistical tests are run simultaneously on a single data set. It was created to address the problem that as the number of tests increases, the risk of concluding that a significant difference exists when it does not increase. The Bonferroni correction is obtained by dividing the P values by the total number of comparisons. We adjusted the P value using the Bonferroni correction when comparing various classifiers.

Chapter 6

Results

In this chapter, we formalize our results for each model based on the evaluation metrics and techniques we decided to use. We present the resultant value of the finetuned hyperparameter for each tested model via K-fold CV and plot the CV results. The confusion matrices for each model were normalized row-wise to enhance the clarity of how well the model identifies images within the infected and healthy class. The classification report of each model represents the average model performance over three different seeds. We plot and present each model's ROC curve and AUC to showcase how well the model distinguishes images from either class at respective threshold settings. As per section 3.9.3, we stuck to the Scikit-learn module default threshold of 0.5 for all ROC curves.

6.1 Logistic regression results

6.1.1 Cross validation results

After performing a 10-fold grid search CV on the logistic regression model, we obtained the finetuned value of hyperparameter "C" to be: **110**.
Based on the deductions and reasonings from [52, 24, 13], a "C" value of 110 for a small-scale dataset is considered to be on the higher side. A high C value infers that the model generates a hyperplane with smaller margins sensitive to outliers.

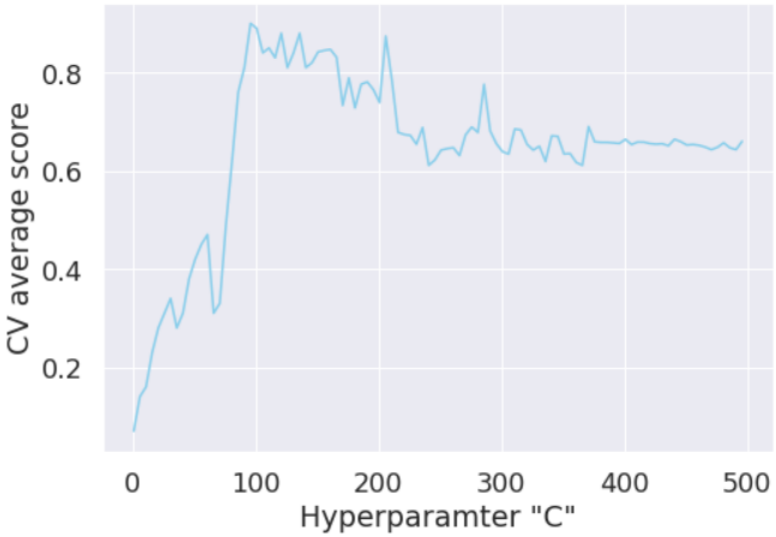
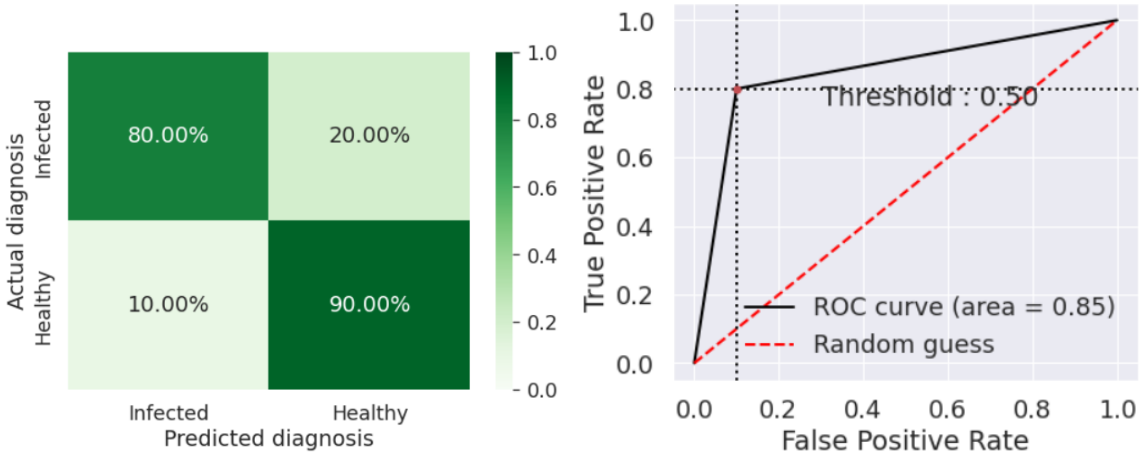


Figure 6.1: Grid search CV score of the logistic regression model

6.1.2 Classification performance

The model delivers an 87% average accuracy. We derive an AUC of 0.85 from the ROC curve. In modern-day diagnostic imaging, an AUC of 0.85 is considered "excellent" and has good discriminatory ability [27]. 85% percent of the time, the model accurately determines that a randomly chosen patient with a diagnosis has a higher absolute risk than a randomly chosen patient without that diagnosis.



Confusion matrix ROC curve

Figure 6.2: Classification performance of the logistic regression model

We trust the model's results, which produced an impressive weighted precision average. The model performed slightly better in accurately predicting healthy images than infected images.

Classification report of logistic regression				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.90	0.90	0.90	20
Infected tongues	0.80	0.80	0.80	10

accuracy			0.87	30
macro avg	0.85	0.85	0.85	30
weighted avg	0.87	0.87	0.87	30

6.2 KNN results

6.2.1 Cross validation results

After performing a 10-fold grid search CV on the logistic regression model, we obtained the finetuned value of hyperparameter "n_neighbors" to be: **15**.
Therefore, the KNN makes classifications based on the most occurring class of its 15 nearest neighbors

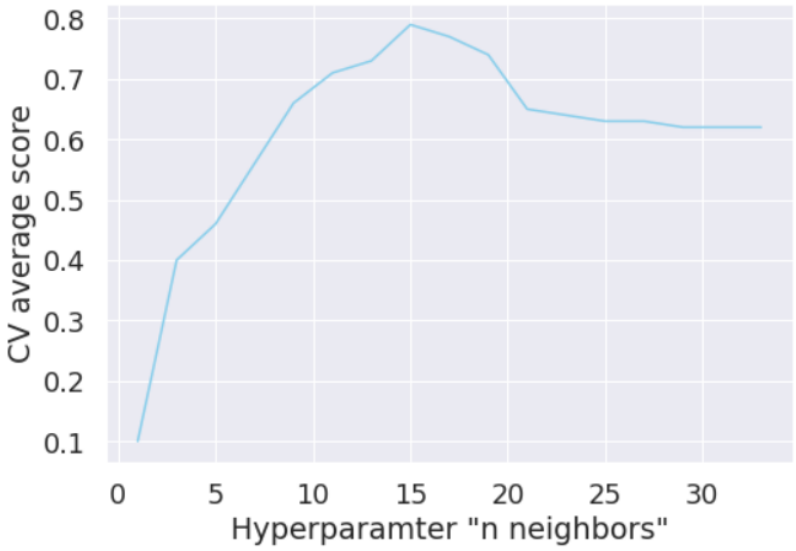
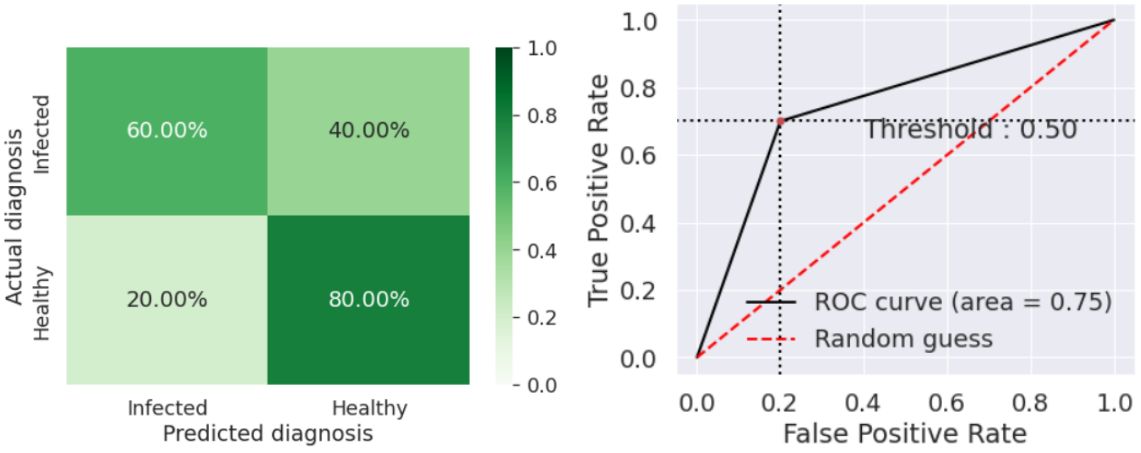


Figure 6.3: Grid search CV score of the KNN model

6.2.2 Classification performance of KNN (K=1)

This KNN model did not undergo any cross-validation. We tested with a K=1 to showcase the effect of the absence of cross-validation on a lazy learner model. The model produces a 77% average accuracy. We derived an AUC of 0.75 from the ROC curve. In modern-day diagnostic imaging, an AUC of 0.75 is considered "acceptable" and has decent discriminatory ability [27]. 75% percent of instances, the model accurately determines that a randomly chosen patient with a diagnosis has a higher absolute risk than a randomly chosen patient without that diagnosis.



Confusion matrix ROC curve

Figure 6.4: Classification performance of the KNN (K=1)

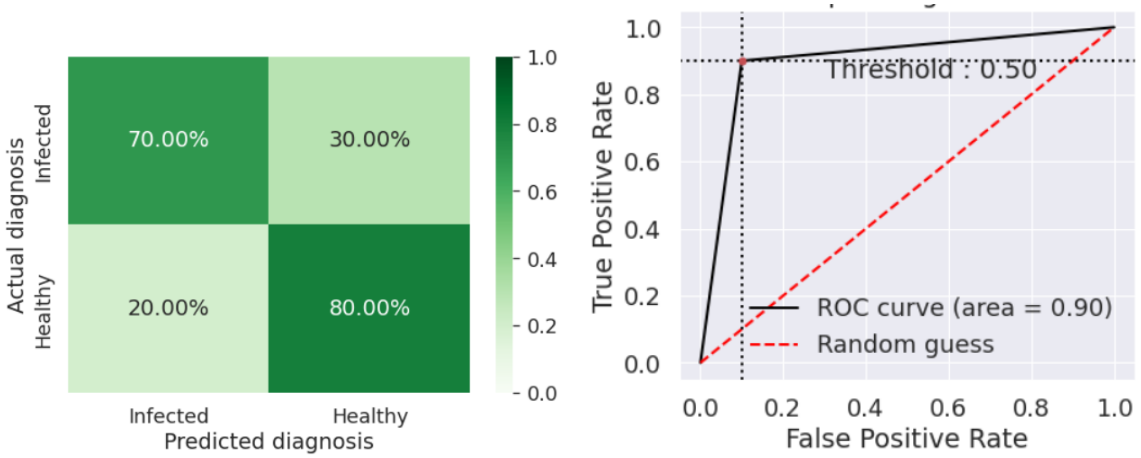
The model misclassified 40% of the infected images. Since no cross-validation took place, the n_neighbors hyperparameter was not finetuned, resulting in a "K" that did not fit with the dataset. KNN makes its predictions based on a single sample. This approach makes the KNN susceptible to several types of distortions, including noise, outliers, and corrupt data. Therefore the model was more inclined to classify images as healthy as they make up two-thirds of the test set.

Classification report of KNN (K=1)				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.84	0.80	0.82	20
Infected tongues	0.64	0.70	0.60	10

accuracy			0.77	30
macro avg	0.74	0.75	0.74	30
weighted avg	0.77	0.77	0.77	30

6.2.3 Classification performance of KNN (K=15)

The model underwent 10-fold grid search cross-validation. We tested with a K=15 to showcase the effect of the cross-validation on a lazy learner model. The model produces an 87% average accuracy. We derive an AUC of 0.9 from the ROC curve. In modern-day diagnostic imaging, an AUC of 0.9 is considered "outstanding" and has very good discriminatory ability [27]. 90% percent of instances, the model accurately determines that a randomly chosen patient with a diagnosis has a higher absolute risk than a randomly chosen patient without that diagnosis.



Confusion matrix ROC curve

Figure 6.5: Classification performance of the KNN (K=15)

By performing grid search CV and finetuning the number of neighbors, the model improved its average weighted accuracy by 10%. In the comparative study, we only consider the finetuned KNN (K=15) model as it is the better performing KNN model out of the two.

Classification report of KNN (K=15)				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.89	0.85	0.87	20
Infected tongues	0.73	0.80	0.76	10

accuracy			0.83	30
macro avg	0.81	0.82	0.82	30
weighted avg	0.84	0.83	0.84	30

6.3 SVC results

6.3.1 Cross validation results

After performing a 10-fold grid search CV on the SVC with an RBF kernel, we obtained the finetuned value of hyperparameter "C" to be: **145**.
The deductions and reasonings from [52, 24, 13] infer that a "C" value of 145 for a small scale dataset is considered to be relatively large. A high "C" value infers that the model generates a hyperplane with smaller margins that is sensitive to outliers. This trend may seem contradictory, as SVCs are known to have "large" margins. However, as mentioned in section 5.13.3, SVCs incorporate soft margins treated as small margins when performing cross-validation.

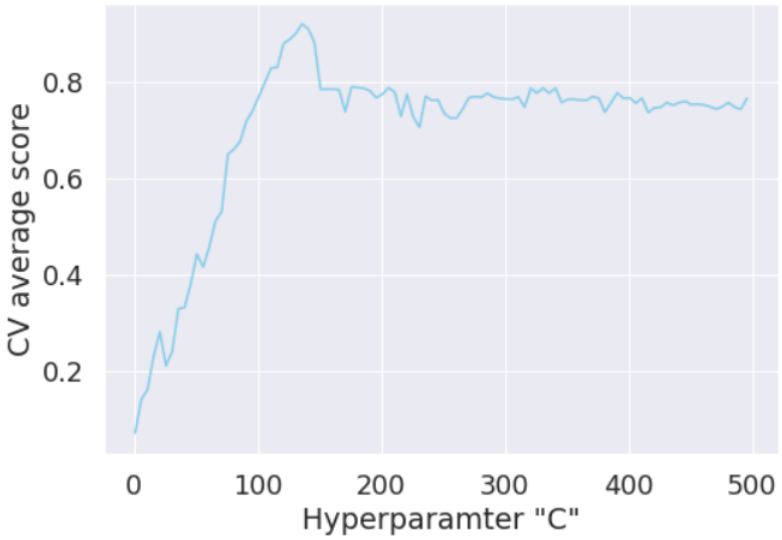
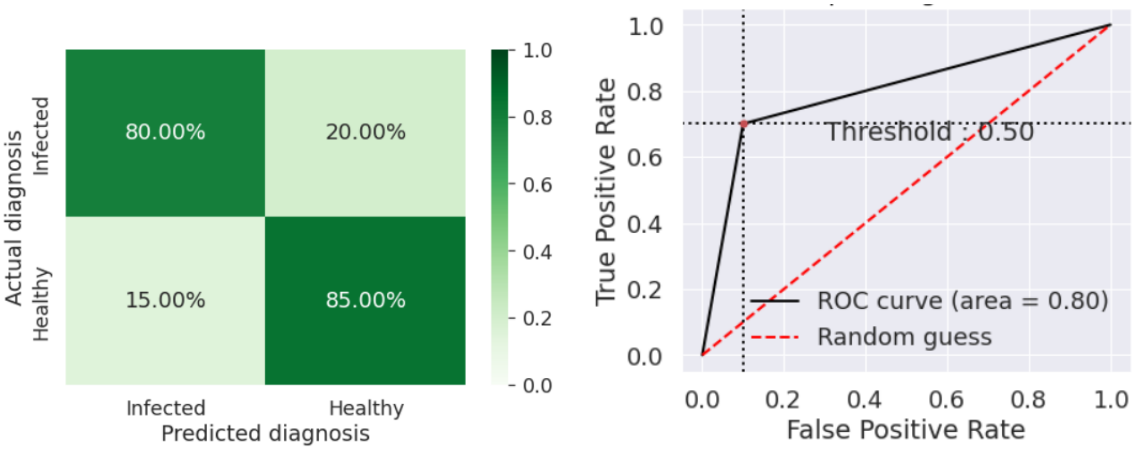


Figure 6.6: Grid search CV score of the SVC model with RBF kernel

6.3.2 Classification performance of SVC with linear kernel

This model does not perform kernelization and aims to find the best fitting hyperplane. Therefore it struggles to classify non-linearly separable data. The model delivers an 83% average accuracy. We derived an AUC of 0.8 from the ROC curve. In modern-day diagnostic imaging, an AUC of 0.8 is considered "excellent" and has good discriminatory ability [27]. 80% percent of the time, the model accurately determines that a randomly chosen patient with a diagnosis has a higher absolute risk than a randomly chosen patient without that diagnosis



Confusion matrix ROC curve

Figure 6.7: Classification performance of SVC with a linear kernel

Classification report of linear SVC				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.86	0.90	0.88	20
Infected tongues	0.78	0.70	0.74	10

accuracy			0.83	30
macro avg	0.82	0.80	0.81	30
weighted avg	0.83	0.83	0.83	30

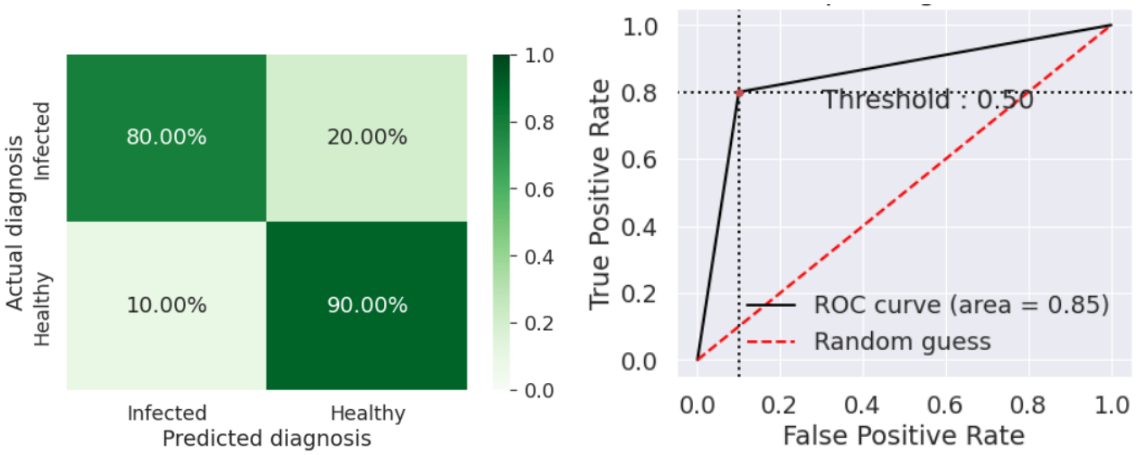
6.3.3 Classification performance of SVC with RBF kernel

This model performs kernelization and redefines the search space by introducing higher dimensionality. The model delivers an 87% average accuracy. We derived an AUC of 0.85 from the ROC curve.

Classification report of SVC with a RBFkernel				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.90	0.90	0.90	20
Infected tongues	0.80	0.80	0.80	10

accuracy			0.87	30
macro avg	0.85	0.85	0.85	30
weighted avg	0.87	0.87	0.87	30

Using the RBF kernel, the SVC improved with a 4% increase in average weighted accuracy. We consider both variants of the SVC in the comparative study.



Confusion matrix ROC curve

Figure 6.8: Classification performance of SVC with a RBF kernel

6.4 Decision tree results

6.4.1 Cross validation results

After performing a 10-fold grid search CV on the tree model, we obtained the finetuned value of hyperparameter "max_depth" to be: **5**.

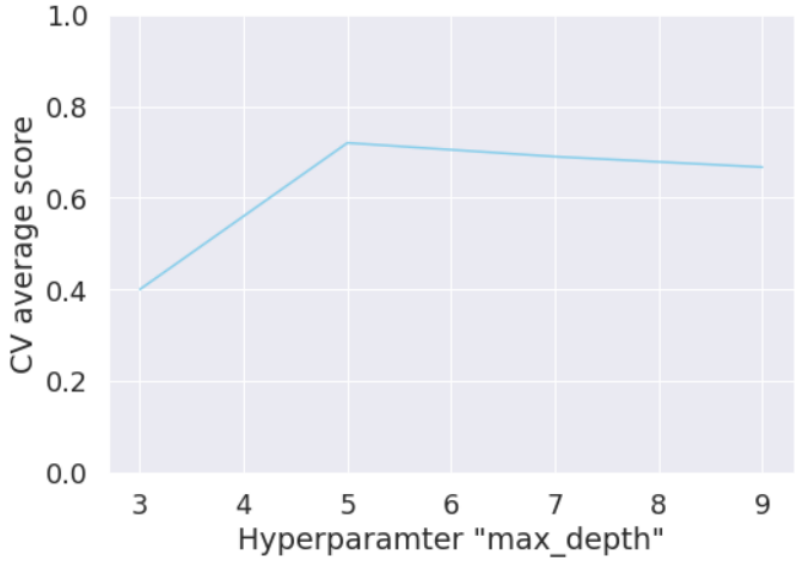
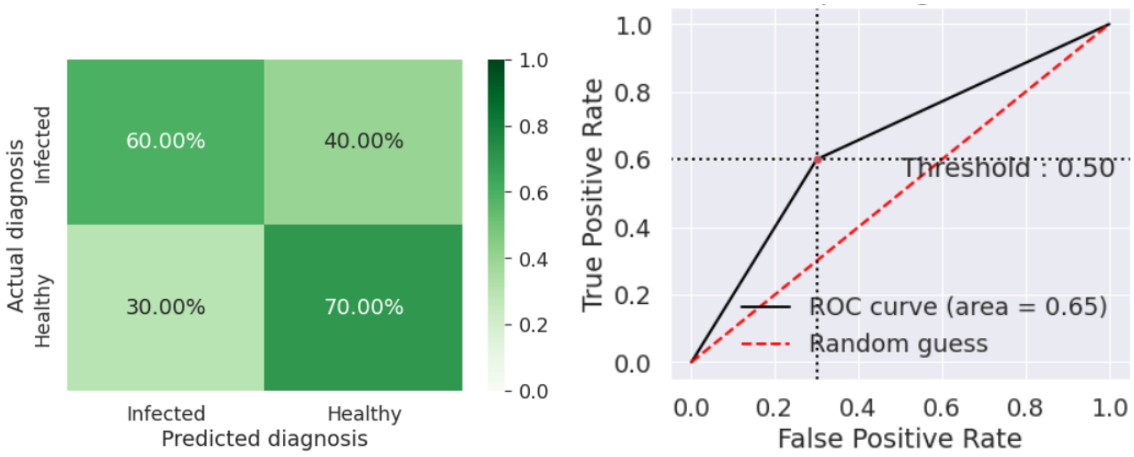


Figure 6.9: Grid search CV score of the decision tree model

6.4.2 Classification performance

The model delivers a 67% weighted average accuracy. We derived an AUC of 0.65 from the ROC curve. In modern-day diagnostic imaging, an AUC of 0.65 is considered "not acceptable" and has poor discriminatory ability [27]. 65% percent of the time, the model accurately determines that a randomly chosen patient with a diagnosis has a higher absolute risk than a randomly chosen patient without that diagnosis.



Confusion matrix ROC curve

Figure 6.10: Classification performance of the decision tree model

Classification report of decision tree				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.78	0.70	0.74	20
Infected tongues	0.50	0.80	0.55	10
accuracy			0.67	30
macro avg	0.64	0.65	0.64	30
weighted avg	0.69	0.67	0.67	30

6.5 Random forest results

6.5.1 Cross validation results

After performing a 10-fold grid search CV on the random forest model, we obtained the finetuned value of hyperparameter "max_depth" to be: **5**.
Therefore, the random forest model makes classifications based on an ensemble of decision trees with five as the max depth.

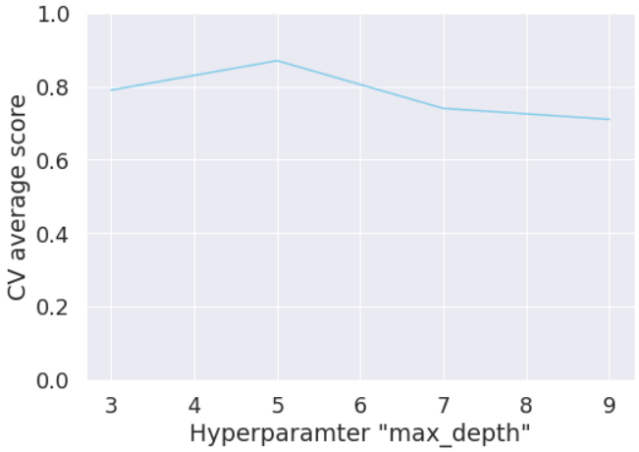
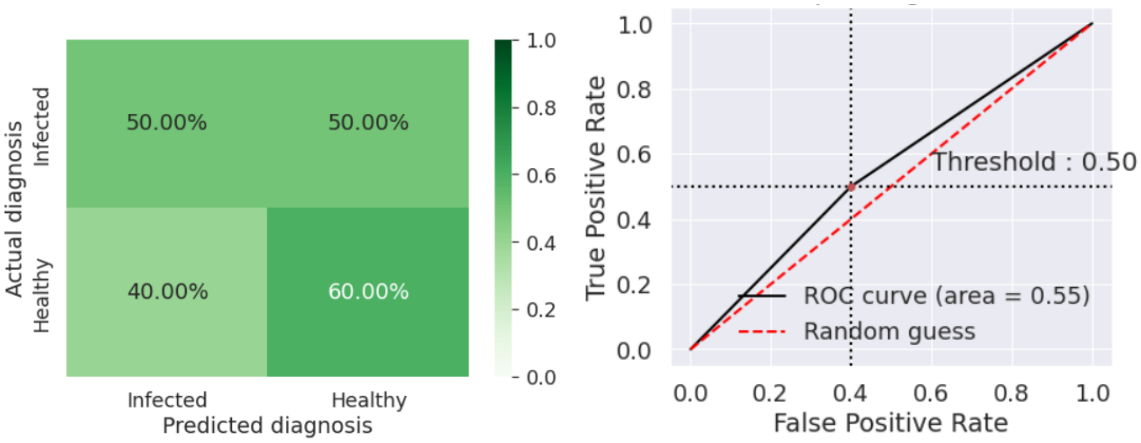


Figure 6.11: Grid search CV score of the random forest model

6.5.2 Classification performance of random forest (no CV)

The model did not undergo any cross-validation. Therefore there was no bound on the max depth for the model. The model produces a 57% average accuracy. We derived an AUC of 0.5 from the ROC curve. In modern-day diagnostic imaging, an AUC of 0.5 is considered highly unreliable. [27]. 50% percent of instances, the model accurately determines that a randomly chosen patient with a diagnosis has a higher absolute risk than a randomly chosen patient without that diagnosis



Confusion matrix ROC curve

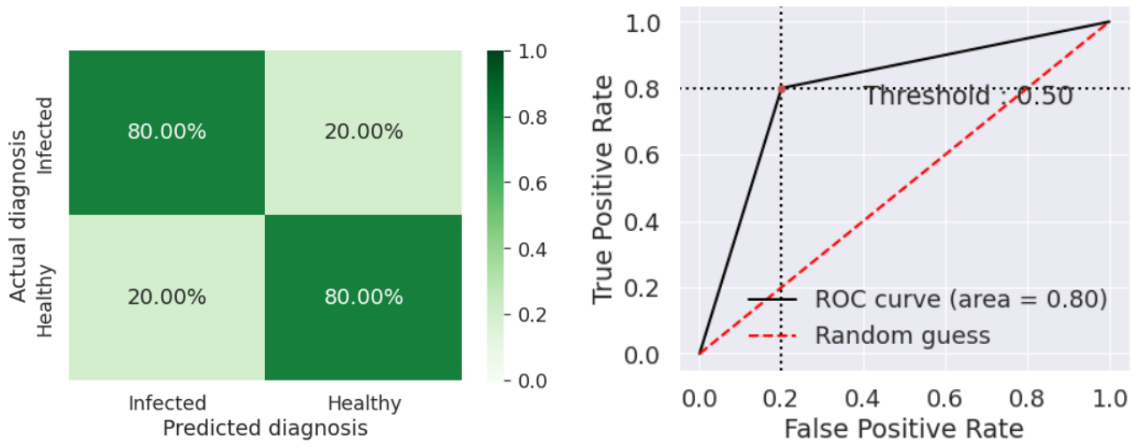
Figure 6.12: Classification performance of random forest model with no CV

Classification report of random forest (no CV)				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.71	0.60	0.65	20
Infected tongues	0.38	0.50	0.43	10
accuracy			0.57	30
macro avg	0.55	0.55	0.54	30
weighted avg	0.60	0.57	0.58	30

6.5.3 Classification performance of random forest with CV

We wanted to illustrate the impact CV has on ensemble methods. By finetuning the max depth of the random forest model, we indirectly finetune all the decision trees in the ensemble. The model produces an 80% average accuracy. We derived an AUC of 0.8 from the ROC curve.

By performing grid search CV and finetuning the number of neighbors, the model improved its average weighted accuracy by 23%. In the comparative study, we only consider the finetuned random forest.



Confusion matrix ROC curve

Figure 6.13: Classification performance of random forest model with CV

Classification report of random forest with CV				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.89	0.80	0.84	20
Infected tongues	0.67	0.80	0.73	10

accuracy			0.80	30
macro avg	0.78	0.80	0.78	30
weighted avg	0.81	0.80	0.80	30

6.6 Pre-trained ResNet-50 results

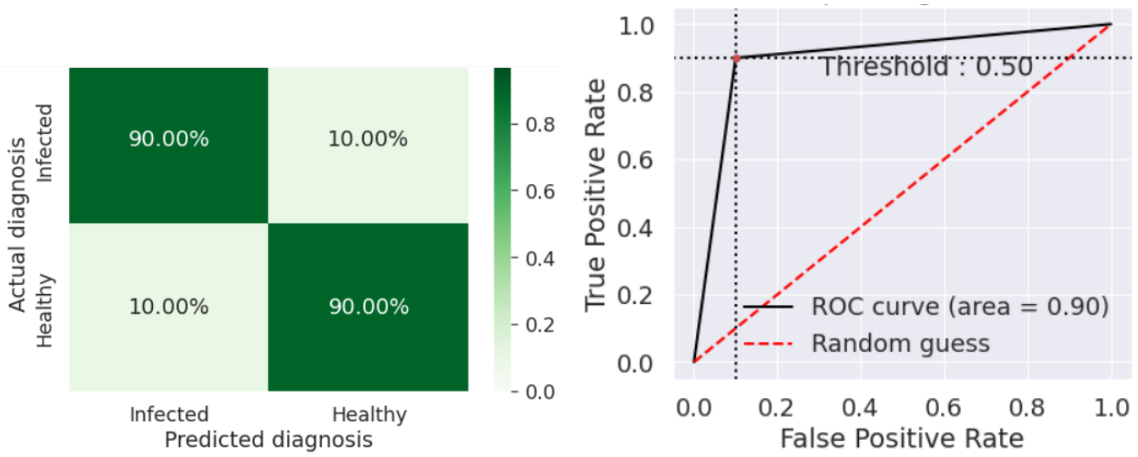
Deep learning models are relatively complex and computationally expensive. Therefore splitting the dataset and training "K' models is inefficient. Due to the ResNet's rigorous training and feature extraction methodology, we decided not to implement CV on the ResNet-50.

6.6.1 Classification performance of pre-trained ResNet-50

The model produces a 80% average accuracy. We derived an AUC of 0.9 from the ROC curve.

Classification report of ResNet-50				
	Precision	**Recall**	**F1 score**	**Support**
Healthy tongues	0.95	0.90	0.92	20
Infected tongues	0.82	0.90	0.86	10

accuracy			0.90	30
macro avg	0.88	0.90	0.89	30
weighted avg	0.90	0.90	0.90	30

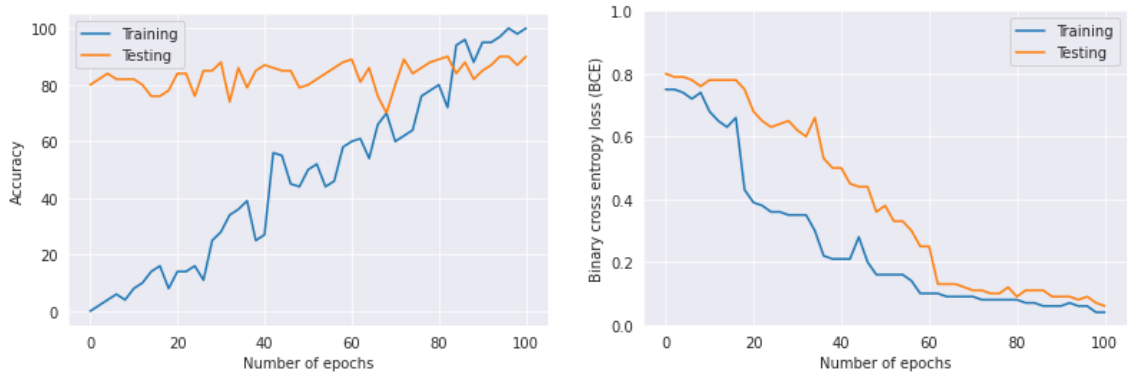


Confusion matrix ROC curve

Figure 6.14: Classification performance of pre-trained ResNet-50

6.6.2 Trends over epochs

The accuracy of the ResNet-50 increases with the number of epochs and then saturates. The binary cross entropy loss decreases with the number of epochs and then saturates. The network is optimized by reducing the loss function and increasing its accuracy. Despite having a small training dataset, we do not observe any unusual significant dips in the accuracy vs. epochs trend. This observation infers that the initial learning parameters were set appropriately, and the model does not overfit



Variation in accuracy with the number of epochs Variation in BCE loss with the number of epochs

Figure 6.15: Optimization of ResNet-50 over 100 epochs

Chapter 7

Comparative analysis and discussion

In this chapter, we compare, justify and rank the performance of our models. We present our observations, motivate our deductions and justify the best fitting model for diagnosing chronic gastritis's clinical binary classification problem. We draw parallels between various models and obtain a deeper insight into the dataset's characteristics.

7.1 t-SNE visualization of the dataset

T-distributed Stochastic Neighbor Embedding (t-SNE) is a technique to visualize high-dimensional data using feature extraction. T-SNE maximizes the distance between most different observations in a high-dimensional space to a two-dimensional/three-dimensional space. Similar observations that are in proximity of one another and may become clustered. T-SNE maps observations to clusters and represents the data in a two or three-dimensional space. We get an in-depth insight into the nature of our dataset using t-SNE. Our dataset is five-dimensional. Principal Component Analysis (PCA) reduces dimensions and captures visualization data. Therefore it is limited to capturing only the linear structures in the features [5]. On that account, we opted to use t-SNE as a data visualization tool rather than PCA as the latter struggles to visualize non-linear data.

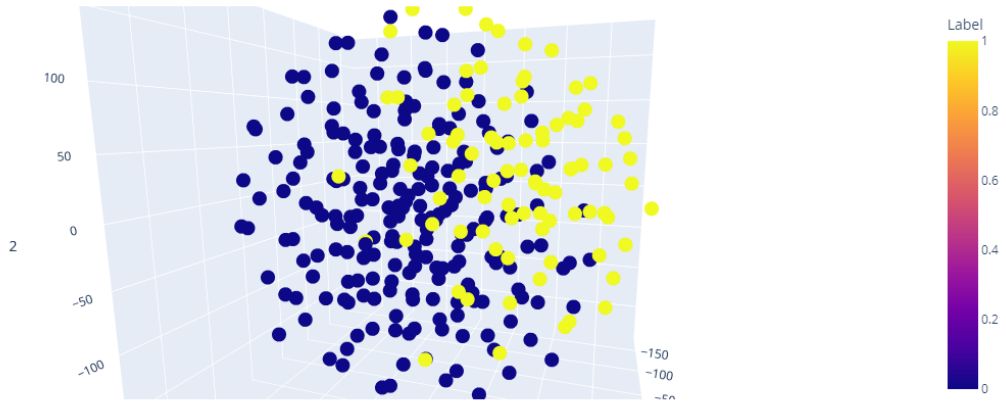


Figure 7.1: 3D t-SNE visualisation of the entire dataset

The above visualization showcases the nature of our dataset. Our dataset is, for the most part, linearly separable. We could construct a hyperplane that can split the

infected images (yellow) from the healthy images (blue) with reasonable accuracy.

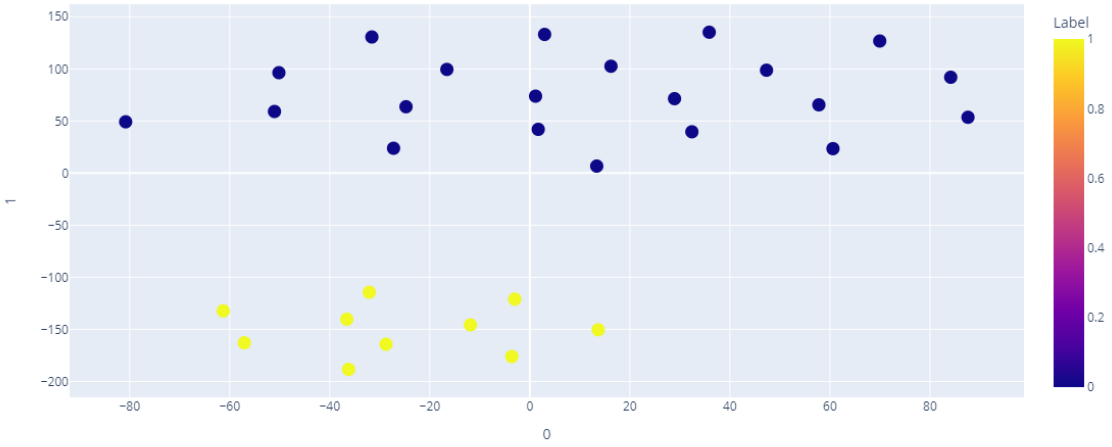


Figure 7.2: 2D t-SNE visualisation of the training dataset

The visualization of the training dataset shows clear linear separability. This characteristic is probably why the SVCs and logistic regression model performed considerably well.

7.2 Accuracy comparison

The average accuracy of a model is the gold standard for evaluating a machine learning model. The decision-making ability of a machine learning model is strongly influenced by its accuracy. According to Zare et al [59] an accuracy of 70% and higher is considered "realistic model performance." Accuracy within the range of 70%-90% is ideal as it is consistent with industry standards. However, the contrast and reverberations of machine learning models differing in produced accuracy depend on the use case. Since the stakes are higher in automated clinical diagnostic imaging, models with higher accuracy are given more weightage [59, 58]. Medical bodies have not set an agreed accuracy threshold for considering diagnostic ML models. Additionally, ML models for automated tongue analysis are scarce. Based on previous related research, most well-constructed models produced an accuracy within 80-91% [11, 58, 39] depending on the target disease.

The pre-trained ResNet-50 produced the highest average accuracy with 90%. From figure 6.15, we observed the pre-trained ResNet-50 converging towards the last 35 epochs as fewer errors were made by the model during training and fluctuations in the learning rate decreased. This infers that the optimization conducted by ADAM on the BCE loss was effective. Our statements were supported by the accuracy trend in figure 6.15, which shows an accuracy of roughly 90%. Despite its simplicity, the decision tree model produced the lowest accuracy. The decision tree model changes its structure for every new image leading to overfitting. Therefore the model struggled when fed into the test dataset as it was greedy and not learning. The accuracy results from the logistic regression, linear SVC, and RBF SVC led us to make an interesting observation regarding the nature of our dataset. The RBF SVC kernel had a 3.33% percent higher accuracy than the linear SVC. This should mean the dataset benefits from kernelization and hence is not entirely linearly separable. However, the logistic regression model, which thrives on linearly separable data, produced the same accuracy as the RBF SVC. We reasoned this unusual correlation by analyzing the loss functions of each classifier. Linear SVC attempts to reduce the hinge loss [56] whereas logistic regression attempts to

Figure 7.3: Average accuracy of each model

reduce the log loss. According to [56], hinge loss is sensitive to noise, whereas log loss is sensitive to outliers. Therefore we learned that our dataset consists of fewer outliers and a reasonable amount of noise. When we applied the RBF kernel to the SVC, the accuracy increased and was equivalent to that of logistic regression. Based on the performance of these three models, we learn that our data is, for the most part, linearly separable.

7.3 Recall comparison

In diagnostic imaging and healthcare in general, it is paramount that diagnostic techniques have minimal false negatives [2]. False negatives are the most harmful incorrect diagnosis as infected patients are labeled healthy. Hence we critically analyze the produced recall from each model. In our case, the recall is the probability that the classifier detects an infected image.

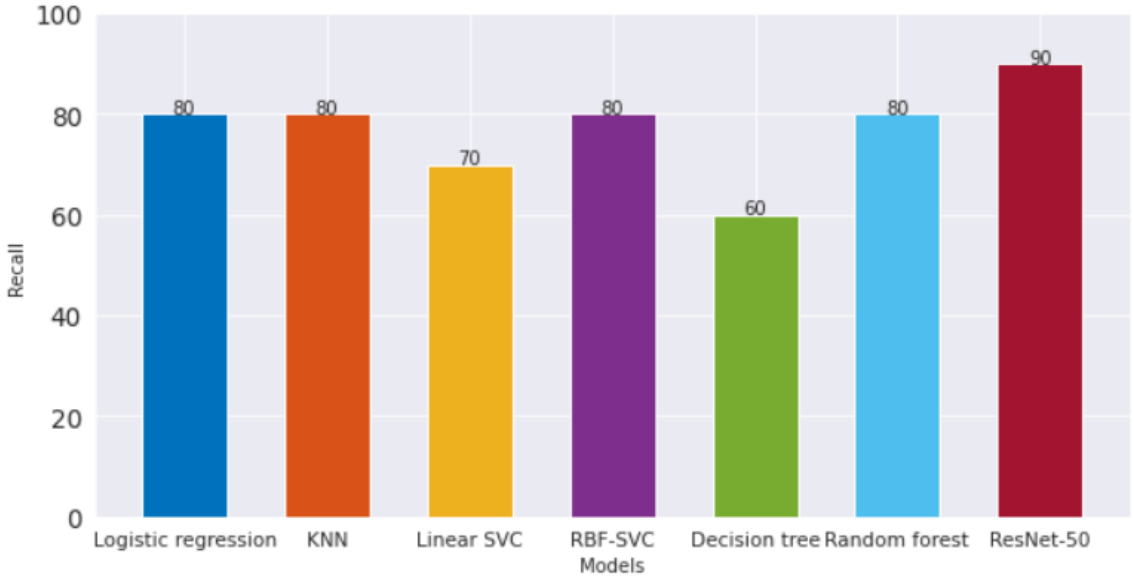


Figure 7.4: Recall of each model

We observed an interesting performance increase with the random forest model.

Although the ensemble model did not perform as well as most other models, it showed a spike in the recall. According to Want et al[51], the impressive recall of random forests can be attributed to the classifier's random under-sampling, which removes class imbalances. We drew parallels from our study to theirs[53] macrosomia prediction random forest model attributed to a 62.11% increase in recall.

7.4 Precision comparison

Apart from accurately detecting infected images, we expect the classifiers to not misclassify healthy images as infected. Therefore an optimal classifier has a minimum of false positives. Although in modern-day diagnostic imaging, false positives may not be necessarily "harmful" as false negatives, they can cause confusion and resource wastage and hence are not desirable [2].

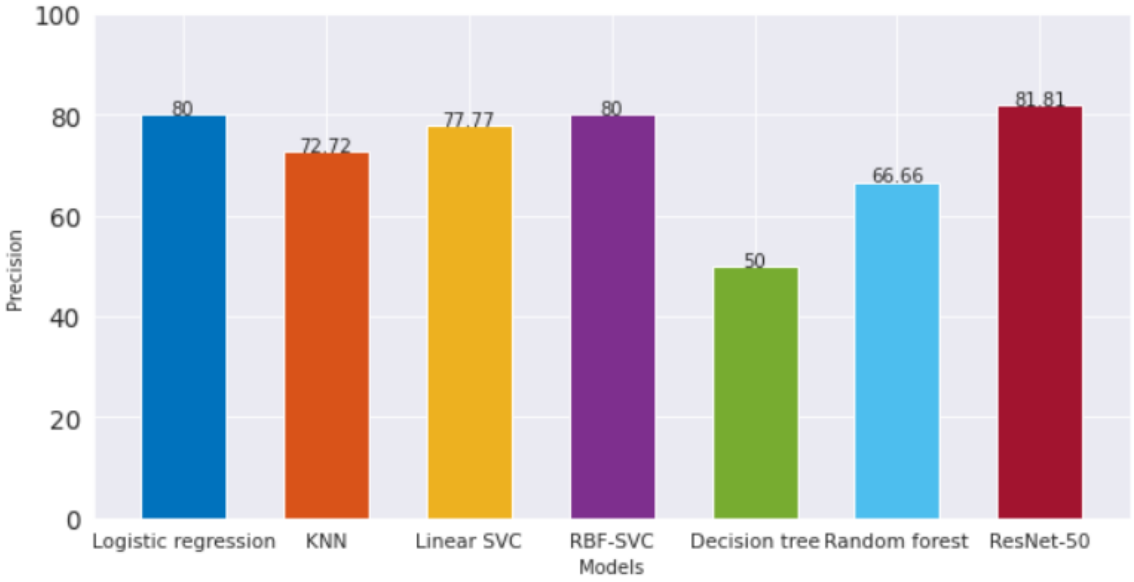


Figure 7.5: Precision of each model

Although the pre-trained ResNet has the best precision compared to its other error rates, the model suffers a relatively high misclassification rate of healthy images. The logistic regression model and an RBF SVC are trialing behind by only 1.81% , compared to the 10% difference in accuracy. The marginal difference could be caused by many reasons ranging from lack of epochs [39, 28] to framework selection. The pre-trained ResNet-50, logistic regression, and RBF-SVC nearly have the same probability of classifying infected images as infected. Hence they roughly have the same amount of false positives.

7.5 F1 score comparison

Per industry standard [2, 15, 10, 9] an F1 score greater than 80% is considered desirable. Thus the pre-trained ResNet is the only model that fits in this category. Since the F1 score is the harmonic mean between precision and recall, we used it as a measure to obtain an impression of how well the models perform under these two metrics.

Figure 7.6: F1 score of each model

7.6 AUC comparisons

The AUC of a model is directly proportional to the model's ability to classify healthy and infected images accurately. The ROC curves of each model represent its recall graphically. They help pinpoint the exact trade-off between the true positive and false-positive rates. In our case, this was at the default preset threshold of 0.5.

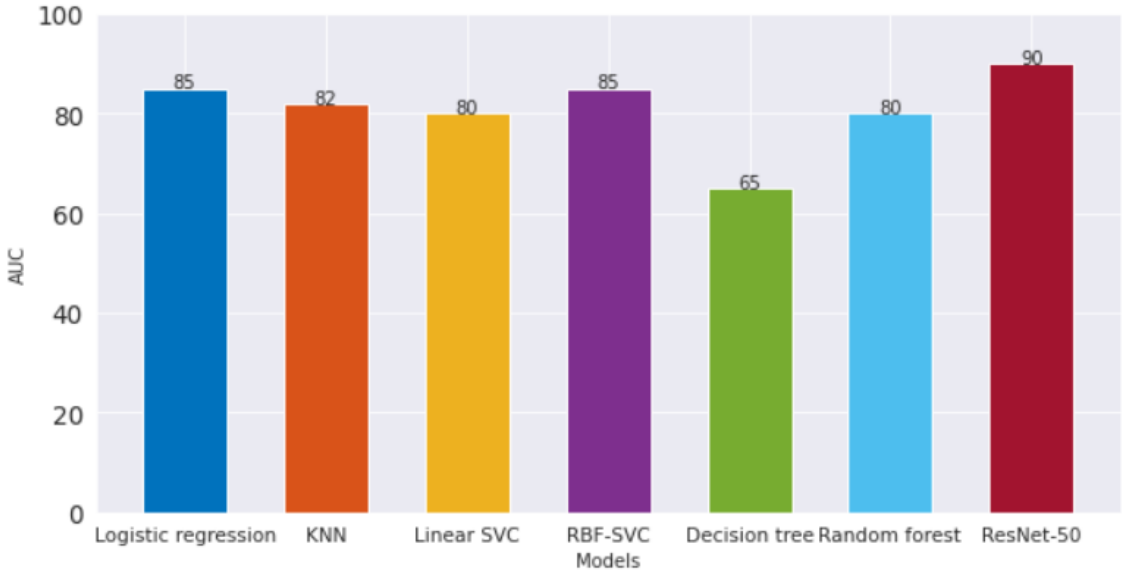


Figure 7.7: AUC of each model

The AUC reflects the model's accuracy. Excellent models have an AUC close to 1, such as the pre-trained ResNet-50, Linear SVC, and logistic regression. Models with no class separation capacity present have an AUC of 0.5. The decision tree classifier lies in the proximity of this benchmark, signifying its weak distinguishing ability.

7.7 Ranking the models

At this point of our study, we have thoroughly evaluated the entire set of classifiers with the established evaluation metrics. We then ranked the models to declare the best performing

model on the tongue dataset.

Decision tree classifier

We start with the worst performing model, which is undeniably the decision tree classifier. The model showcased the lowest performance in every evaluation metric. The model was susceptible to overfitting and was not robust. Its low AUC indicates that the classifier does not have the adequate distinguishing ability. The inference from the model's confusion matrix validates our claims as 40% of infected images (FN) were misclassified, and so were 30% of healthy images (FP). The alarming amount of false negatives and the general weak performance of the models makes them unsuitable for chronic gastritis diagnosis via tongue images.

Random forest classifier

The Random forest showed reasonably good recall and hence the ability to detect infected images. This impressive recall is reflected in the high AUC of the model. Despite differing marginally, the ensemble-based classifier fell short in precision, F1 score, and overall accuracy. Diagnostic agents need to classify healthy images accurately without high misclassification errors. The random forest performed considerably better than the decision tree classifier, which supports the notion of ensemble-based diagnostic assistants. The random forest model can potentially be considered for a recall-centered automated chronic gastritis detection that strongly focuses on the infected image class.

Linear SVC

Linear SVC models performed neck to neck in nearly every evaluation metric with the KNN classifier. KNN classified the infected images better than the linear SVC resulting in a higher recall. In contrast, linear SVC classified the healthy images better than the KNN, resulting in higher precision. They both produce the same accuracy and have the same AUC. In diagnostic imaging, recall and precision are essential parameters for evaluating classifiers. Regarding the F1 score, the KNN model has a marginal edge over the linear SVC. Therefore we ranked the linear SVC lower.

7.8 Implementing statistical tests

KNN, RBF SVC, logistic regression, and the pre-trained ResNet-50 had nearly identical performances. Specifically between RBF SVC and logistic regression in nearly every evaluation metric. We resorted to McNemar's test to compare the proportion of errors between the models. The test declares a result to be statistically significant if its value is less than the Bonferroni corrected critical value. Since we compared four models with an alpha of 0.05, the Bonferroni corrected critical value is the alpha divided by the number of compared models, which is 0.0125. We did three iterations of tests.

Run 1				
Models	**KNN**	**RBF SVM**	**Logistic regression**	**Pre-trained ResNet-50**
KNN	X	0.93453	0.88453	0.34522
RBF SVM	0.93452	X	1.0000	0.84354
Logisitc Regression	0.88453	1.0000	X	**0.01116**
Pre-trained ResNet-50	0.34522	0.84354	**0.01116**	X

Table 7.1: McNemar's test results for the first iteration

Run 2				
Models	**KNN**	**RBF SVM**	**Logistic regression**	**Pre-trained ResNet-50**
KNN	X	0.62351	0.88453	0.01929
RBF SVM	0.62351	X	0.59943	0.84354
Logisitc Regression	0.88453	0.59943	X	0.23433
Pre-trained ResNet-50	0.01929	0.84354	0.23433	X

Table 7.2: McNemar's test results for the second iteration

Run 3				
Models	**KNN**	**RBF SVM**	**Logistic regression**	**Pre-trained ResNet-50**
KNN	X	1.0000	0.66541	0.33243
RBF SVM	1.0000	X	0.23453	**0.00964**
Logisitc Regression	0.66541	0.23453	X	0.55643
Pre-trained ResNet-50	0.33243	**0.00964**	0.55643	X

Table 7.3: McNemar's test results for the third iteration

Lower statistical errors indicate better model performance. We find a significant proportion of errors between the pre-trained ResNet-50 and the logistic regression model in the first iteration. Next, we find a significant proportion of errors between the RBF SVC and the pre-trained ResNet-50 in the third iteration. All other combinations of pooling dimensions in all other runs show no significant result.

KNN

We ranked the KNN classifier the lowest among the four models from the McNemar's tests. Before the tests, the KNN model had a noticeable discrepancy in performance, which was validated by the statistical test. The "lazy" nature of the KNN model leads it not to analyze the features optimally. KNNs do not elaborate image attributes to their respective classes but rather identify pixel differences to make predictions. Nevertheless, KNNs can perform image analysis on tongue images to detect chronic gastritis. They can certainly be considered a potential candidate for diagnostic assistance.

Logistic regression vs RBF SVC

The logistic regression classifier produced impressive results. The model fitted well with our data and kept the number of false negatives to a minimum. However, logistic regression models do not work well with non-linearly separable data. In healthcare, such data is very prominent and needs to be accounted. We ranked the logistic regression model lower than the RBF SVC as RBF SVC efficiently deals with non-linearly separable data and is versatile. The model works with the hinge loss, therefore making it resistant to outliers. Both these models are promising candidates for tongue image analysis and can be applied to a domain of related autoimmune diseases. These models thrive in situations with resource constraints as they are computationally cheaper than deep learning models.

7.9 Pre-trained ResNet-50

The pre-trained ResNet-50 showcased impressive classification ability. It demonstrated impressive feature mapping and was able to accurately identify patterns. The model

produced the highest accuracy and had the least false negatives. The deep neural network performed exceptionally well in terms of recall, precision, and F1 score. Moreover, the transfer learning occurs smoothly, and the loss function is constantly optimized (figure 6.15). Although the residual model is complex and computationally heavy, its nature is robust and does not overfit the dataset. We firmly believe that the results and performance will improve if the same model is trained on a larger tongue dataset.

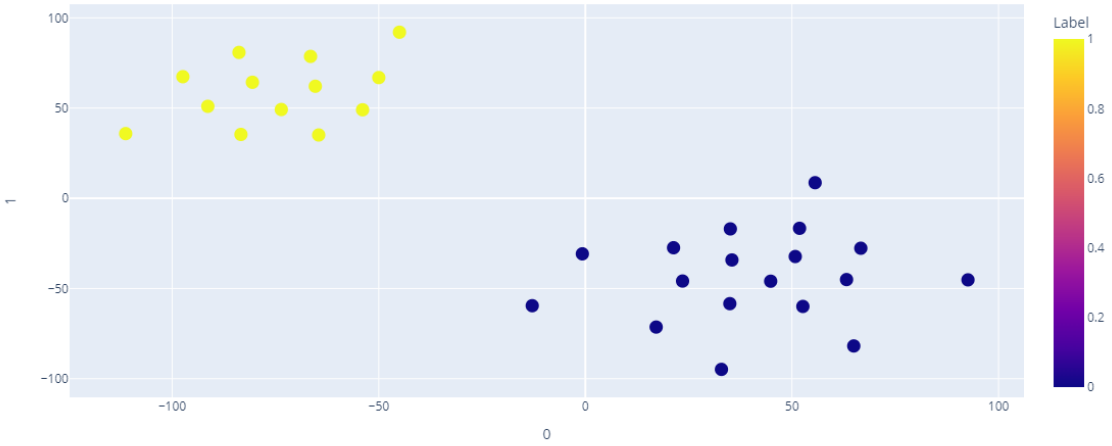


Figure 7.8: 2D t-SNE visualisation of the pre-trained ResNet-50 training dataset after classification

We analyzed the misclassifications of the pre-trained ResNet-50. The model would misclassify the same infected image as healthy (False negatives) and two healthy images as infected (False positives). After investigating the learned embeddings, we concluded that the model misclassified the infected image based on chromatic features and always inferred it as pink. The two false negatives were caused due to wrong interpretation of coating (textural feature) and color.

7.10 Ranking

In summary, we construct the following ranking of models based on how well they fit and perform automated image analysis on a tongue dataset to solve the binary classification problem of diagnosing chronic gastritis.

Ranking	**Model**
1	Pre-trained ResNet-50
2	SVC with RBF kernel
3	Logistic regression classifier
4	KNN classifier
5	Linear SVC
6	Random forest classifier
7	Decision tree classifier

7.11 Performance comparison with state of the art

In order to understand how this study's results compare with state of the art, we performed a comparative analysis between the performance of the best-performing model in our study and state-of-the-art models in studies mentioned in the related work chapter. The table below compares the image tongue analysis performance between the pre-trained ResNet-50 and state-of-the-art models. Although the state-of-the-art models performed image tongue analysis, it is essential to note that they were designed to identify various tongue-related diseases and not specifically type "A" chronic gastritis. Although the studies focus on automated tongue image analysis for autoimmune/tongue-related disease detection, the respective tasks are not entirely focused on diagnostic imaging of chronic gastritis with Dulam, Ramesh and G's [11] and Rajakumaran and Sasikala's [39] study being the only exceptions. Therefore we could not derive a direction comparison with our results.

Author	**Model**	**Targeted disease**	**Accuracy**	**Precision**	**Recall**	**F1 score**
Rajakumaran and Sasikala [39]	VGG 19	Organ specific chronic gastritis	93.70%	93.80%	93.70%	93.68%
Dulam, Ramesh and G [11]	Linear SVC	Chronic gastritis	60%	60%	60%	60%
Mansour, Althobaiti, Ashour[28]	ASDL-TCI with a synergic network	Autoimmune diseases (chromatic feature extraction)	98.30%	98.4%	97.30%	-
Song and Chao[46]	ResNet-50 and Inception_v3	Autoimmune disease classification	95.92%	91.19%	93.45%	92.66%
Our study	**Pre-trained ResNet-50**	**Chronic gastritis**	**90%**	**81.81%**	**90%**	**85.71%**

Chapter 8

Conclusions and future work

The primary contribution of this study is the comparisons and evaluation of various machine learning models and strategies that can best perform tongue image analysis for the classification of chronic gastritis. The primary objective of this paper is achieved as most of the developed models were able to distinguish and classify tongue images infected with chronic gastritis accurately. Our initial hypothesis turned out to be accurate, and we fulfill the second objective of this paper by declaring that the ResNet 50 was the best performing model for the said binary classification task within this study.

Furthermore, the study finds that machine learning models can find a clear correlation between the palette of a patient's tongue and their healthiness by analyzing geometric, chromatic, and textual features. This mapping can be leveraged to build better classification models that analyze the demographic of the tongue for disease detection.

Moreover, we observe a poor performance in decision tree classifiers which may indicate an incompatible approach for tongue analysis. However, the usage of ensemble methods drastically improves performance and classification ability. Real-world clinical use cases consist of resource and data scarcity constraints. Adopting a logistic regression classifier or RBF SVC rather than a deep learning model can lead to optimal results in such scenarios.

We believe that the results, trends, and observations we derived from our comparative study showcased strong promise for using machine learning classifiers as automated diagnostic assistants in healthcare.

8.1 Limitations

Despite adopting various approaches and techniques to mitigate data scarcity issues, the major roadblock in this study was the small size of the tongue data set. More specifically, the number of infected tongue images. Due to these constraints, we were forced to simplify our data pipeline and focus on a binary classification problem rather than a multi-class classification problem on various autoimmune diseases.

Although using a grayscale led to a lower computational work load, we believe that using coloured images on the pre-trained ResNet-50 would lead to better performance [55].

8.2 Future work

We wish to extend our study in three phases.

Autoimmune diseases such as heart disease, hyperthyroid, and other gastritis variants develop visual symptoms on the tongues. We wish to expand the use case of such models into multi-label classifiers that can also accurately diagnose these diseases.

Since our study showcases the correlation between tongue symptoms and patient condition, we aim to explore the performance of state-of-the-art machine learning models on a much larger dataset.

Finally, we will pivot to testing reliable algorithms, models, and appropriate data processing techniques on a tongue dataset derived via a smartphone camera. This shift will enable us to build a real-time versatile mobile diagnostic assistant that can potentially be advantageous in healthcare.

Appendix A

Appendix

The codebase of this study has been fully developed with the Python programming language. Python is considered the best fit for machine learning and AI-based projects. The libraries and frameworks used in this study support Python.

A.1 Scikit-learn

Scikit-learn (sklearn)[1] is a popular machine learning library that supports classification, regression, and clustering algorithms. We used this library extensively to build, train and test the five core machine learning models. The table below lays out the sklearn classes[2] for each model and technique. We used OpenCV and scikit-image for some of the preprocessing steps.

Scikit-learn library	
Model	**sklearn class**
Logistic regression	sklearn.linear_model.LogisticRegression
KNN	sklearn.neighbors.KNeighborsClassifier
SVC	sklearn.svm.SVC
Decision tree	sklearn.tree.DecisionTreeClassifier
Random forest	sklearn.ensemble.RandomForestClassifier

Technique	**Class**
Gridsearch CV	sklearn.model_selection.GridSearchCV
GrabCut algorithm	cv.grabCut
Test-train split	sklearn.model_selection.train_test_split
t-SNE	sklearn.manifold.TSNE
Gray Scale	rgb2gray

[1] The documentation of **sklearn version 1.1.2** can be found here

[2] Click on the class to view its documentation

A.2 PyTorch

PyTorch[3] is an extensively used machine learning library that is specialised towards the development of deep learning models. Pytorch was used to focilitate the working of the pre-trained ResNet-50. We used the **numpy** [4] library to handle all mathematical computations on multidimensional arrays. The table below lays out the used PyTorch classes for the ResNet-50 and implemented deep learning techniques.

PyTorch library	
Purpose	**PyTorch class**
Pre-trained ResNet-50	torchvision.models.ResNet50_Weights
ADAM	torch.optim.Adam
Test-train split	torch.nn.BCELoss
BCE loss	torch.utils.data.DataLoader
Gray Scale	torchvision.transforms.Grayscale

A.3 Dataset and sample images

As mentioned in section 5.2, the TongueDataset is sealed per the owner's instructions. However, we display six raw sample images the owner let us publish. These six images were not used in the study and serve as sample images to explain concepts in this thesis and visualize the actual dataset. These images cannot be used for any purpose besides viewing without the author's consent.

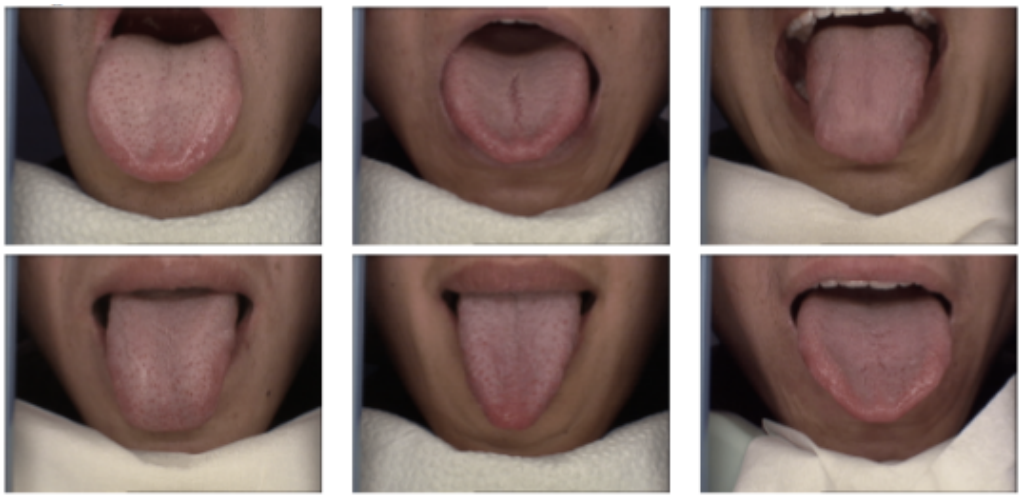


Figure A.1: Sample tongue images courtesy of Dr. Ravi Teja

[3]The documentation of **PyTorch 1.12** can be found here
[4]The documentation of **numpy** can be found here

Bibliography

[1] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[2] Anthony K Akobeng. Understanding diagnostic tests 1: sensitivity, specificity and predictive values. *Acta paediatrica*, 96(3):338–341, 2007.

[3] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[4] Davide Anguita, Luca Ghelardoni, Alessandro Ghio, Luca Oneto, and Sandro Ridella. The 'k'in k-fold cross validation. In *20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 441–446. i6doc. com publ, 2012.

[5] Farzana Anowar, Samira Sadaoui, and Bassant Selim. Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne). *Computer Science Review*, 40:100378, 2021.

[6] Steven J Benson and Jorge J More. A limited memory variable metric method in subspaces and bound constrained optimization problems. In *in Subspaces and Bound Constrained Optimization Problems*. Citeseer, 2001.

[7] Vibha Bhatnagar and Prashant Bansod. Challenges and solutions in automated tongue diagnosis techniques: A review. *Critical Reviews™ in Biomedical Engineering*.

[8] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[9] Jason Brownlee. What is the difference between a batch and an epoch in a neural network. *Machine Learning Mastery*, 20, 2018.

[10] Rhea Chowers and Yair Weiss. Why do cnns learn consistent representations in their first layer independent of labels and architecture? *arXiv preprint arXiv:2206.02454*, 2022.

[11] S Dulam, V Ramesh, and G Malathi. Tongue image analysis for covid-19 diagnosis and disease detection. *International Journal of Advanced Trends in Computer Science and Engineering*, pages 7924–7928, 2020.

[12] Ejemai Amaize Eboreime, Obinna Idika, Kasarachi Omitiran, Oghenekome Eboreime, and Latifat Ibisomi. Primary healthcare planning, bottleneck analysis and performance improvement: An evaluation of processes and outcomes in a nigerian context. *Evaluation and program planning*, 77:101712, 2019.

[13] Bradley Efron. The efficiency of logistic regression compared to normal discriminant analysis. *Journal of the American Statistical Association*, 70(352):892–898, 1975.

[14] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys (CSUR)*, 25(2):73–169, 1993.

[15] Muhammad Hasnain, Muhammad Fermi Pasha, Imran Ghani, Muhammad Imran, Mohammed Y Alzahrani, and Rahmat Budiarto. Evaluating trust prediction and confusion matrix measures for web services ranking. *IEEE Access*, 8:90847–90861, 2020.

[16] Kristoffer H Hellton and Nils Lid Hjort. Fridge: Focused fine-tuning of ridge regression for personalized predictions. *Statistics in Medicine*, 37(8):1290–1303, 2018.

[17] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[18] Min-Chun Hu, Ming-Hsun Cheng, and Kun-Chan Lan. Color correction parameter estimation on the smartphone and its application to automatic tongue diagnosis. *Journal of medical systems*, 40(1):1–8, 2016.

[19] Min-Chun Hu, Guang-Yu Zheng, Yian-Ting Chen, and KC Lan. Automatic tongue diagnosis using a smart phone. In *2014 IEEE international conference on systems, man, and cybernetics*, 2014.

[20] Muhammad Javed Iqbal, Zeeshan Javed, Haleema Sadia, Ijaz A Qureshi, Asma Irshad, Rais Ahmed, Kausar Malik, Shahid Raza, Asif Abbas, Raffaele Pezzani, et al. Clinical applications of artificial intelligence and machine learning in cancer diagnosis: looking into the future. *Cancer cell international*, 21(1):1–11, 2021.

[21] Seunghyeon Kim, Jihoon Ryoo, Dongyeob Lee, and Youngho Kim. Residual quantity in percentage of factory machines using computervision and mathematical methods. *arXiv preprint arXiv:2111.05080*, 2021.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[24] Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y Ng. Efficient l˜ 1 regularized logistic regression. In *Aaai*, volume 6, pages 401–408, 2006.

[25] Bin Liu, Guangqin Hu, Xinfeng Zhang, and Yiheng Cai. Application of an improved grab cut method in tongue image segmentation. In *International Conference on Intelligent Computing*, pages 484–495. Springer, 2018.

[26] Xuan Liu, Zhu-Mei Sun, Yan-Na Liu, Qing Ji, Hua Sui, Li-Hong Zhou, Fu-Feng Li, and Qi Li. The metabonomic studies of tongue coating in h. pylori positive chronic gastritis patients. *Evidence-Based Complementary and Alternative Medicine*, 2015, 2015.

[27] Jayawant N Mandrekar. Receiver operating characteristic curve in diagnostic test assessment. *Journal of Thoracic Oncology*, 5(9):1315–1316, 2010.

[28] Romany F Mansour, Maha M Althobaiti, and Amal Adnan Ashour. Internet of things and synergic deep learning based biomedical tongue color image analysis for disease diagnosis and classification. *IEEE Access*, 9:94769–94779, 2021.

[29] Stamatis Mastromichalakis. Alrelu: A different approach on leaky relu activation function to improve neural networks performance. *arXiv preprint arXiv:2012.07564*, 2020.

[30] Sarang Narkhede. Understanding auc-roc curve. *Towards Data Science*, 26(1):220–227, 2018.

[31] Tayo Obafemi-Ajayi, Ratchadaporn Kanawong, Dong Xu, Shao Li, and Ye Duan. Features for automated tongue image shape classification. In *2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops*, pages 273–279. IEEE, 2012.

[32] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs. *arXiv preprint arXiv:1811.12019*, 1:2, 2018.

[33] FY Osisanwo, JET Akinsola, O Awodele, JO Hinmikaiye, O Olakanmi, and J Akinjobi. Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3):128–138, 2017.

[34] Bo Pang, David Zhang, and Kuanquan Wang. The bi-elliptical deformable contour and its application to automated tongue segmentation in chinese medicine. *IEEE transactions on medical imaging*, 24(8):946–956, 2005.

[35] Jigneshkumar L Patel and Ramesh K Goyal. Applications of artificial neural networks in medical science. *Current clinical pharmacology*, 2(3):217–226, 2007.

[36] Molly Perencevich and Robert Burakoff. Use of antibiotics in the treatment of inflammatory bowel disease. *Inflammatory bowel diseases*, 12(7):651–664, 2006.

[37] Andreas L Prodromidis and Salvatore J Stolfo. Cost complexity-based pruning of ensemble classifiers. *Knowledge and Information Systems*, 3(4):449–469, 2001.

[38] Pavlo M Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. 2017.

[39] S Rajakumaran and J Sasikala. An efficient machine learning based tongue color analysis for automated disease diagnosis model. *Int. J. Adv. Res. Eng. Technol.*, 11(12):718–734, 2020.

[40] Yoram Reich and SV Barai. Evaluating machine learning models for engineering problems. *Artificial Intelligence in Engineering*, 13(3):257–272, 1999.

[41] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. " grabcut" interactive foreground extraction using iterated graph cuts. *ACM transactions on graphics (TOG)*, 23(3):309–314, 2004.

[42] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[43] Warren S Sarle. Neural networks and statistical models. 1994.

[44] Rajat Vikram Singh. Imagenet winning cnn architectures–a review. *Rajat Vikram Singh–Institue of Software Research at Carnegie Mellon University*, 2015.

[45] Max Siurala, Pentti Sipponen, and Matti Kekki. Chronic gastritis: dynamic and clinical aspects. *Scandinavian Journal of Gastroenterology*, 20(sup109):69–76, 1985.

[46] Chao Song, Bin Wang, and Jiatuo Xu. Classifying tongue images using deep transfer learning. In *2020 5th International conference on computational Intelligence and Applications (ICCIA)*, pages 103–107. IEEE, 2020.

[47] RG Strickland and IR Mackay. A reappraisal of the nature and significance of chronic atrophic gastritis. *The American journal of digestive diseases*, 18(5):426–440, 1973.

[48] Zhu-Mei Sun, Jie Zhao, Peng Qian, Yi-Qin Wang, Wei-Fei Zhang, Chun-Rong Guo, Xiao-Yan Pang, Shun-Chun Wang, Fu-Feng Li, and Qi Li. Metabolic markers and microecological characteristics of tongue coating in patients with chronic gastritis. *BMC Complementary and Alternative Medicine*, 13(1):1–10, 2013.

[49] Marzia Hoque Tania, Khin Lwin, and Mohammed Alamgir Hossain. Advances in automated tongue diagnosis techniques. *Integrative Medicine Research*, 8(1):42–56, 2019.

[50] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[51] Fangyi Wang, Yongchao Wang, Xiaokang Ji, and Zhiping Wang. Effective macrosomia prediction using random forest algorithm. *International Journal of Environmental Research and Public Health*, 19(6):3245, 2022.

[52] Peiming Wang and Martin L Puterman. Mixed logistic regression models. *Journal of Agricultural, Biological, and Environmental Statistics*, pages 175–200, 1998.

[53] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9(2):187–212, 2022.

[54] Eric W Weisstein. Convolution. *https://mathworld. wolfram. com/*, 2003.

[55] Felix A Wichmann, Lindsay T Sharpe, and Karl R Gegenfurtner. The contributions of color to recognition memory for natural scenes. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 28(3):509, 2002.

[56] Yichao Wu and Yufeng Liu. Robust truncated hinge loss support vector machines. *Journal of the American Statistical Association*, 102(479):974–983, 2007.

[57] Masafumi Yamazaki, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, and Kohta Nakashima. Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds. *arXiv preprint arXiv:1903.12650*, 2019.

[58] Ming Yin, Jennifer Wortman Vaughan, and Hanna Wallach. Understanding the effect of accuracy on trust in machine learning models. In *Proceedings of the 2019 chi conference on human factors in computing systems*, pages 1–12, 2019.

[59] Mohammad Erfan Zare, Yanzhong Wang, Atefeh Nasir Kansestani, Afshin Almasi, and Jun Zhang. Procalcitonin has good accuracy for prognosis of critical condition and mortality in covid-19: a diagnostic test accuracy systematic review and meta-analysis. *Iranian Journal of Allergy, Asthma and Immunology*, 19(6):557–569, 2020.

[60] David Zhang, Hongzhi Zhang, Bob Zhang, et al. *Tongue image analysis*. Springer, 2017.

[61] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018.

[62] Junwen Zhang, Guangqin Hu, and Xinfeng Zhang. Extraction of tongue feature related to tcm physique based on image processing. In *2015 12th international computer conference on wavelet active media technology and information processing (ICCWAMTIP)*, pages 251–255. IEEE, 2015.

[63] Lili Zhao, You Zhou, Feng Wu, and Mingjing Ai. Inter-layer correlation considered rd models and lagrange multiplier for svc mgs coding. *Signal, Image and Video Processing*, 8(8):1581–1589, 2014.

[64] Liu Zhi, David Zhang, Jing-qi Yan, Qing-Li Li, and Qun-lin Tang. Classification of hyperspectral medical tongue images for tongue diagnosis. *Computerized Medical Imaging and Graphics*, 31(8):672–678, 2007.

[65] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Chu Hong Hoi, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, 33:21285–21296, 2020.

[66] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[67] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11127–11135, 2019.