EARLY STAGE DISEASE DIAGNOSIS USING HUMAN NAIL IN IMAGE PROCESSING

A PROJECT REPORT

Submitted by

NAVANISHA.D [REGISTER NO: 211417104161]

PAVITHRA.V [REGISTER NO: 211417104182]

RESHIKA.D [REGISTER NO: 211417104224]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.

ANNA UNIVERSITY: CHENNAI600 025

AUGUST 2021

BONAFIDE CERTIFICATE

Certified that this project report **"EARLY STAGE DISEASE DIAGNISIS USING HUMAN NAIL IN IMAGE PROCESSING"** is the bonafide work of "**NAVANISHA.D[REGNO:211417104161]and PAVITHRA.V[REGNO:211417104182] and RESHIKA.D[REGNO:211417104224]"** who carried out the project work under my supervision.

SIGNATURE

Dr.S.MURUGAVALLI, M.E,Ph.D.
HEAD OF THE DEPARTMENT

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

Mrs.A.KANCHANA.M.E
SUPERVISOR
ASSISTANT PROFESSOR

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on…..05-08-2021....

INTERNAL EXAMINER

EXTERNALEXAMINER

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr. P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI**, **Dr.C.SAKTHIKUMAR,M.E.,Ph.d** and **Tmt. SARANYASREE SAKTHIKUMAR B.E.,M.B.A.**for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.Mani, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CSE Departmen ,**Dr. S.MURUGAVALLIM.E.,Ph.D** for the support extended throughout the project.

We would like to thank my **Project Guide Mrs A.KANCHANA,M.E.** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

NAVANISHA.D[211417104161]

PAVITHRA.V[211417104182]

RESHIKA.D[211417104224]

ABSTRACT

Human's hand nail is analysed to identify many diseases at early stage of diagnosis. Study of person hand nail color helps in identification of particular disease in healthcare domain. The proposed system guides in such scenario to take decision in disease diagnosis. The input to the proposed system is person nail image. The system will process an image of nail and extract features of nail which is used for disease diagnosis. Here, first training set data is prepared using Weka tool from nail images of patients of specific diseases. A feature extracted from input nail image is compared with the training data set to get result. In this experiment we found that using color feature of nail image average 80% results are correctly matched with training set data during three tests conducted. Human nail consist of various features, out of which proposed system uses nail color changes for disease diagnosis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

S.NO	ABBREVIATON	EXPANSION
1.	CNN	Convolution Neural Network
2.	UML	Unified Modeling Language
3.	DWT	Discrete Wavelet Transform

CHAPTER 1

1. INTRODUCTION

1.1 Overview

In healthcare domain many diseases can be predicted by observing color of human nails. Doctors observe nails of patient to get assistance in diseases identification .Usually pink nails indicate healthy human .The need of system to analyze nails for diseases prediction is because human eye is having subjectivity about colors, having limitation in resolution and small amount of color change in few pixels on nail would not be highlighted to human eyes which may lead to wrong result whereas computer recognizes small color changes on nail. The proposed system will extract color feature of human nail image for disease prediction .The system is focusing on image recognition on the basis of human nail color analysis. Many diseases could be identified by analyzing nails of human hands. In this system human nail image is captured using camera. Captured image is uploaded to our system and region of interest from nail area is selected from uploaded image manually.Selected area is then processed further for extracting features of nail such as color of nail. This color feature of nail is matched using simple matcher algorithm for diseases prediction. In this way the system is useful in prediction of diseases in their initial stages. In Literature study we mentioned some of the diseases in their initial stages. In Literature study we mentioned some of the diseases with its related color changes in nails.

1.1 Problem Definition

Human nail can be used for the prediction of various systemic and dermatological diseases. The proposed system – Nail Image Processing System helps us to create a model which can perform the analysis of human nail and thereby help us in predicting various diseases. Common signs that may be noticeable around the nail are discoloration of nail to black, white, yellow or green,

thickening of nail, dry or scaly skin around the nail. This project contends a deep convolutional network to classify diseases from images. This work has been tested on our dataset and it results in great performance in feature extraction .This proposed system will help the doctors in the early diagnosis of diseases.

CHAPTER 2
LITERATURE SURVEY

1. TITLE: A system for nail color analysis in healthcare

AUTHOR: DM Shah, H.Pandit

YEAR: 2016

DESCRIPTION

This paper is concentrated on the system of image recognition on the premise of color analysis. Many diseases might be identified by analyzing nails of hands. The proposed system relies on the algorithm which automatically extracts only nails' area from scanned back side of palm. These selected pixels are processed for further analysis. The system is computer based, so small discontinuities in color values are observed, which we are able to detect color changes within the initial stage of disease. During this manner, system is quite useful in prediction of diseases in their initial stages.

MERITS:

- Reduces complexity and produces better accuracy in matching.
- The system is computer based, so small discontinuities in color values are observed.

DEMERITS:

- The nail fold shadows or fungus may cause mismatching without accurate result.
- It doesn’t consider additional traits for clear results.

2.TITLE: Study of Nail Unit using Image Processing Methods

AUTHOR: Trupti S.Indi, Yogesh A.Gunge

YEAR: 2016

DESCRIPTION:

This paper explores the prevailing research works associated with nail plate and nail matrix as tool for bio-metric system, nail fold capillaries to identify the disease severity levels & affected organs, nail surface as evidence in forensic science to identify the chemical effects, nail samples to identify drug intake and abuse etc. The proposed system relies supported the algorithm which automatically extracts only nails' area from scanned back side of palm. These selected pixels are processed for further analysis. The system is computer based, so small discontinuities in color values are observed, which we are able to detect color changes within the initial stage of disease. During this manner, system is type of useful in prediction of diseases in their initial stages disease. During this way, system is sort of useful in prediction of diseases in their initial stages.

MERITS:

- It can work for low resolution images, no special sensors /devices are required.
- It considers nail plate appearance only and complexity is less

DEMERITS:

- It gives high accuracy when fusion with other metrics only, fusion is still an issue.
- Acceptability is an issue and retrieving nail matrix feature is challengeable

2. TITLE; An image preprocessing method for fingernail segmentation in microscopy image

AUTHOR: Ting wie-houe Shih-Hsiung Lee,Chu-Sing Yang,Chein-Hui Yeh

YEAR: 2017

DESCRIPTION:

Digital image processing plays a key role in medical imaging. Nail diagnosis is one of the methods in medical imaging to predict the diseases. Nails can reflect the present health condition, genetically inheritance information, and historical information of drug or alcohol usage for the past months or even a year, etc. This paper explores the existing research works related to nail plate and nail matrix as tool for bio-metric system, nail fold capillaries to identify the disease severity levels & affected organs, nail surface as evidence in forensic science to identify the chemical effects, nail samples to identify drug intake and abuse etc. So that, Nail is analyzed by various imaging types and processing algorithms to recognize the person's uniqueness, health condition and its history. This paper also identifies the research challenges and issues.

MERITS:

- This paper proposes an image pre-processing method, trying to segment different parts of nail and provides significant effect.
- It can be used in medical diagnostic system, biometric authentication or other biometric application.

DEMERITS:

- One of the main drawbacks in biometric system is imposter attacks, that is people leave their palm/fingerprint whenever they touch an object and thus making a way of spoofing.
- Full nail plate cannot be used for authentication of the growth of nail plates.

CHAPTER 3
SYSTEM ANALYSIS

3.1 Existing System

There are different ways of disease diagnosis such as through various tests (blood test etc..) and symptom's available on various parts of body guides toward disease diagnosis patient should wait for the report to analyse the problem.

3.2 Proposed System

The main objective of this system design to provide an application for use in healthcare domain this is advantages in terms of cost and time the proposed system will take nail image as an input and will perform some processing on input image then finally it will predict probable disease this system can be used by people as well as by doctors in health care domain.

3.3 Requirement Analysis and Specification

3.3.1 Input Requirements

Input requirements consists of jpg, .png images of nails. The images are in the form of 128 rows and columns.

3.3.2 Output Requirements

Output requirements contains test image and trained dataset. The trained dataset is obtained from CNN model.

3.3.3 Functional Requirements

Functional requirements has Keras model and Json model

3.4 Technology stack

Domain -Deep Learning

Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabelled. Also known as deep neural learning or deep neural network.

Front end- Python 3.9

- **OpenCV**
- **NumPy**
- **scikit-learn**
- **pandas**
- **matplotlib**
- **keras**

OpenCV

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. It makes use of Numpy, which is a highly optimized library for numerical operations with MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. It is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures gets converted to/from NumPy arrays. This makes it easier to integrate it with other libraries which use NumPy. For example, libraries such as SciPy and Matplotlib. It is an open source computer vision and machine learning software library . It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

NumPy

NumPy is, just like SciPy, Scikit-Learn, Pandas, etc. one of the packages which cannot be missed while learning data science , mainly because this library provides an array data structure that holds some benefits over Python lists, such as: being more compact, faster access in reading and writing items, being more convenient and more efficient. To make a numpy array, np.array() function is used.

Scikit-learn

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib.

The functionality that scikit-learn provides include:

- **Regression**, including Linear and Logistic Regression
- **Classification**, including K-Nearest Neighbors
- **Clustering**, including K-Means and K-Means++
- **Model selection**
- **Preprocessing**, including Min-Max Normalization

Pandas

Pandas is the most popular python library that is used for data analysis. It provides highly optimized performance with back-end source code is purely written in C or Python.

We can analyze data in pandas with:

- ✓ Series
- ✓ Data Frames

Series

Series is one dimensional (1-D) array defined in pandas that can be used to store any data type.

Data Frames

Data Frames is two-dimensional (2-D) data structure defined in pandas which consists of rows and columns.

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It consists of several plots like line, bar, scatter, histogram etc.

Keras

Keras is an open-source neural-network **library** written in **Python**. It is capable of running on top of Tensor Flow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It is developed using four guiding principles:

1. **Modularity**: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.

2. **Minimalism**: The library provides just enough to achieve an outcome, no frills and maximizing readability.

3. **Extensibility**: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.

4. **Python**: No separate model files with custom file formats. Everything is native Python.

Back end - TENSORFLOW 2.0

Tensor Flow is an end-to-end open source platform for deep learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in deep learning and developers easily build and deploy deep learning powered applications. It is a general-purpose high-performance computing library open-sourced by Google in 2015. Since the beginning, its main focus was to provide high-performance APIs for building

Neural Networks (NNs). However, with the advance of time and interest by the Deep learning community, the lib has grown to a full deep learning ecosystem.

CHAPTER 4

SYSTEM DESIGN

4.1 Data Dictionary

Train.csv

NAIL IMAGE	LABEL	CLASS ID	CLASS
M1	TRAIN	0	MELANOMA
M2	TRAIN	0	MELANOMA
M3	TRAIN	0	MELANOMA
M4	TRAIN	0	MELANOMA
M5	TRAIN	0	MELANOMA
M6	TRAIN	0	MELANOMA
M7	TRAIN	0	MELANOMA
M8	TRAIN	0	MELANOMA
M9	TRAIN	0	MELANOMA
M10	TRAIN	0	MELANOMA
M11	TRAIN	0	MELANOMA
M12	TRAIN	0	MELANOMA
M13	TRAIN	0	MELANOMA
M14	TRAIN	0	MELANOMA
M15	TRAIN	0	MELANOMA
M16	TRAIN	0	MELANOMA
M17	TRAIN	0	MELANOMA
M18	TRAIN	0	MELANOMA
M19	TRAIN	0	MELANOMA
M20	TRAIN	0	MELANOMA

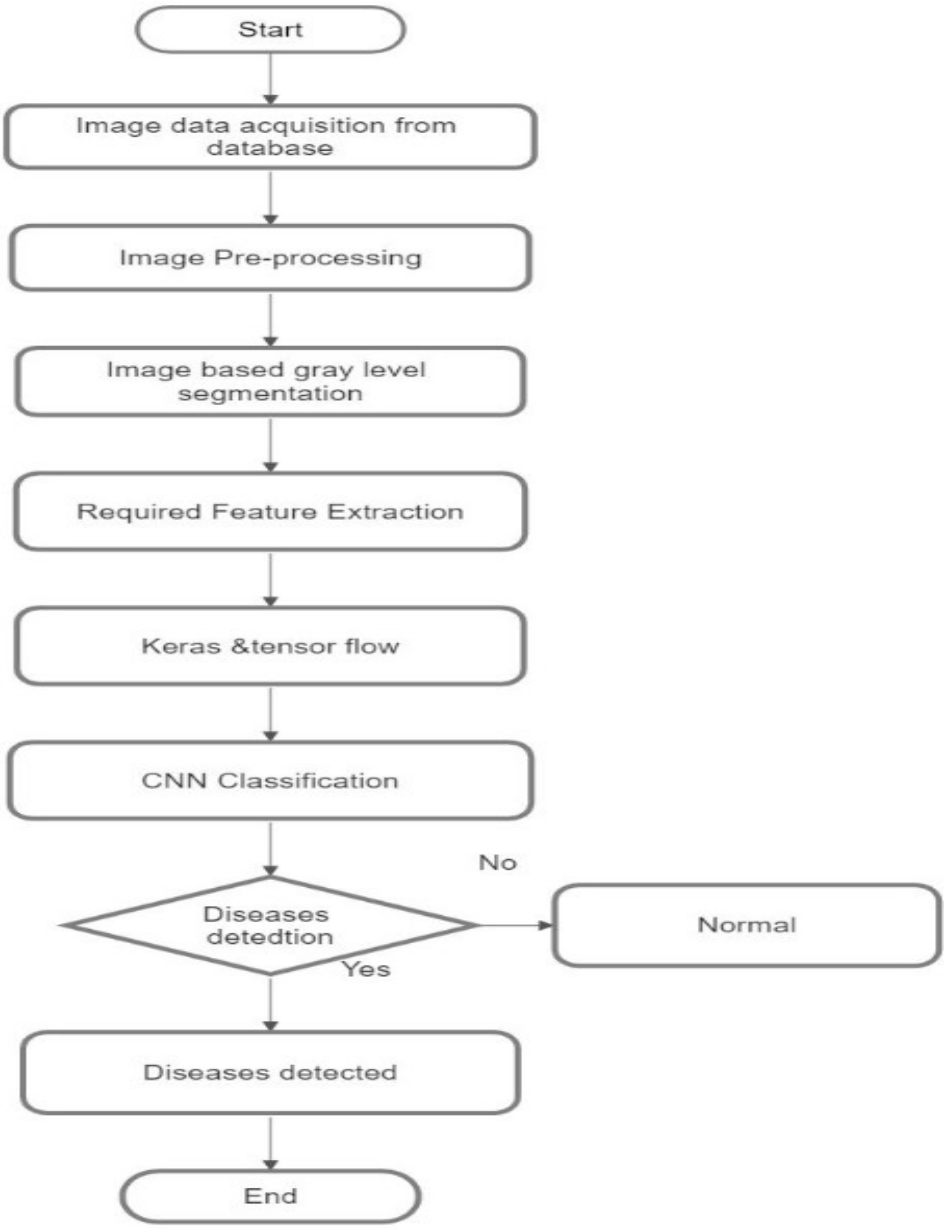
N1	TRAIN	1	NORMAL
N2	TRAIN	1	NORMAL
N3	TRAIN	1	NORMAL
N4	TRAIN	1	NORMAL
N5	TRAIN	1	NORMAL
N6	TRAIN	1	NORMAL
N7	TRAIN	1	NORMAL
N8	TRAIN	1	NORMAL
N9	TRAIN	1	NORMAL
N10	TRAIN	1	NORMAL
N11	TRAIN	1	NORMAL
N12	TRAIN	1	NORMAL
N13	TRAIN	1	NORMAL
N14	TRAIN	1	NORMAL

N15	TRAIN	1	NORMAL
N16	TRAIN	1	NORMAL
N17	TRAIN	1	NORMAL
N18	TRAIN	1	NORMAL
N19	TRAIN	1	NORMAL
N20	TRAIN	1	NORMAL
O1	TRAIN	2	ONYCHOLYSIS
O2	TRAIN	2	ONYCHOLYSIS
O3	TRAIN	2	ONYCHOLYSIS
O4	TRAIN	2	ONYCHOLYSIS
O5	TRAIN	2	ONYCHOLYSIS
O6	TRAIN	2	ONYCHOLYSIS
O7	TRAIN	2	ONYCHOLYSIS
O8	TRAIN	2	ONYCHOLYSIS
O9	TRAIN	2	ONYCHOLYSIS
O10	TRAIN	2	ONYCHOLYSIS
O11	TRAIN	2	ONYCHOLYSIS
O12	TRAIN	2	ONYCHOLYSIS
O13	TRAIN	2	ONYCHOLYSIS
O14	TRAIN	2	ONYCHOLYSIS
O15	TRAIN	2	ONYCHOLYSIS
O16	TRAIN	2	ONYCHOLYSIS
O17	TRAIN	2	ONYCHOLYSIS
O18	TRAIN	2	ONYCHOLYSIS
O19	TRAIN	2	ONYCHOLYSIS
O20	TRAIN	2	ONYCHOLYSIS

Test.csv

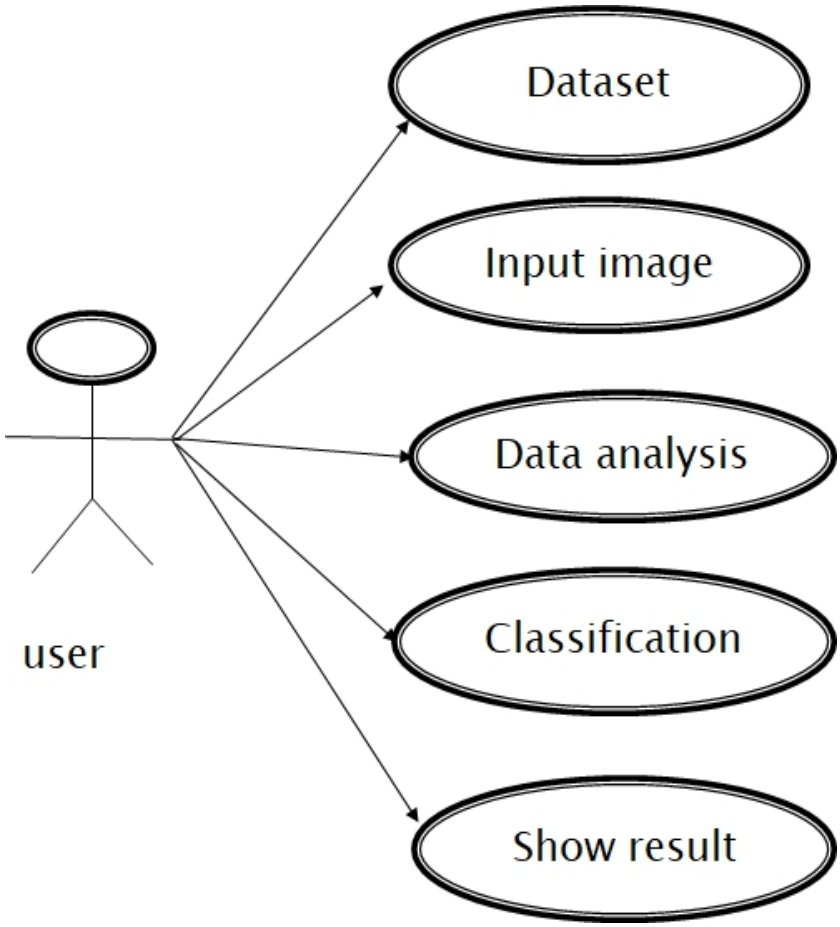
NAIL IMAGE	LABEL	CLASS ID	CLASS
N1	TEST	1	NORMAL
N2	TEST	1	NORMAL
N3	TEST	1	NORMAL
M18	TEST	0	MELANOMA
M19	TEST	0	MELANOMA
O4	TEST	2	ONYCHOLYSIS
O6	TEST	2	ONCHOLYSIS
M6	TEST	0	MELANOMA
O13	TEST	2	ONYCHOLYSIS

4.2 Data Flow Diagram

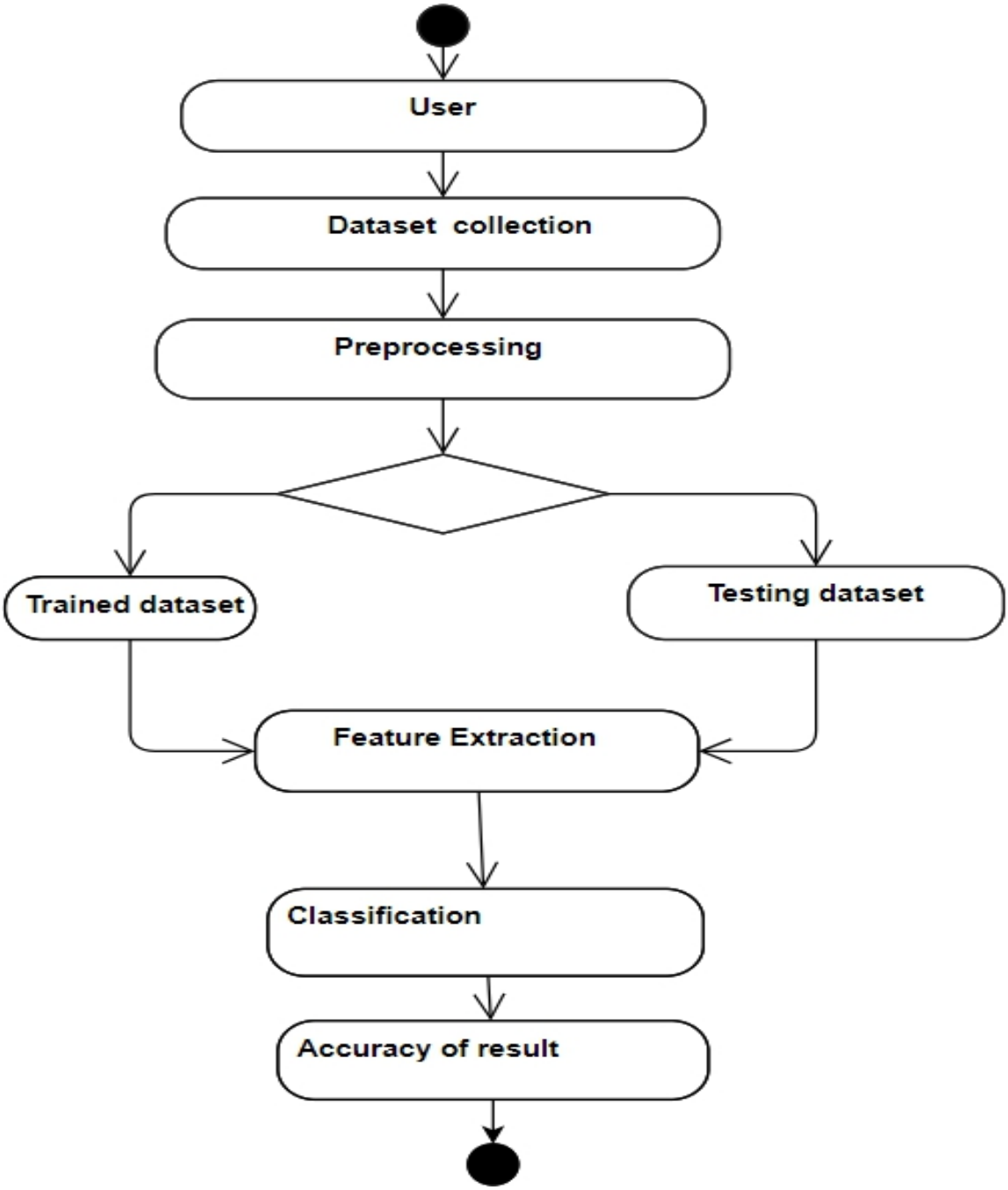


4.3 UML Diagram

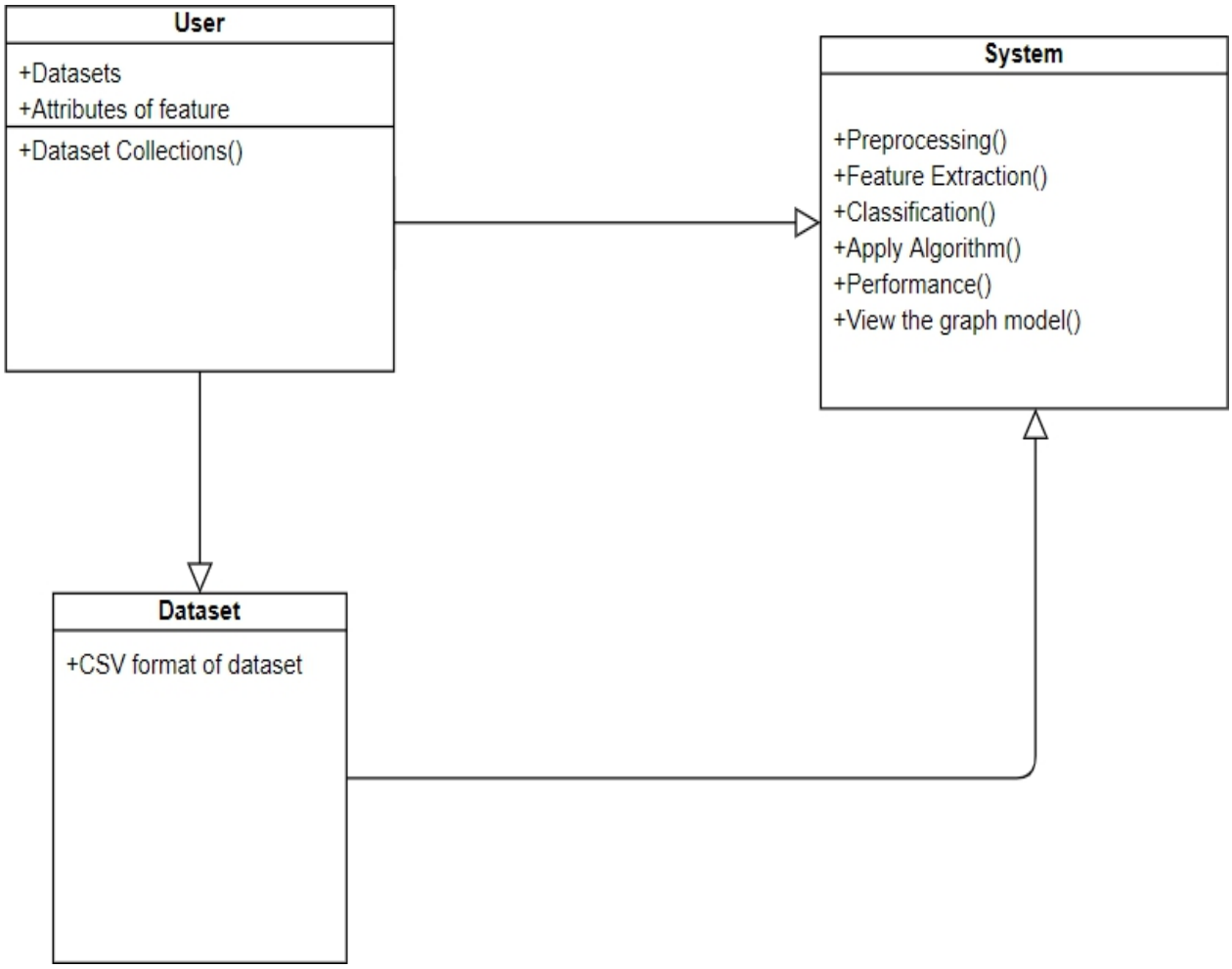
4.3.1 Use Case Diagram



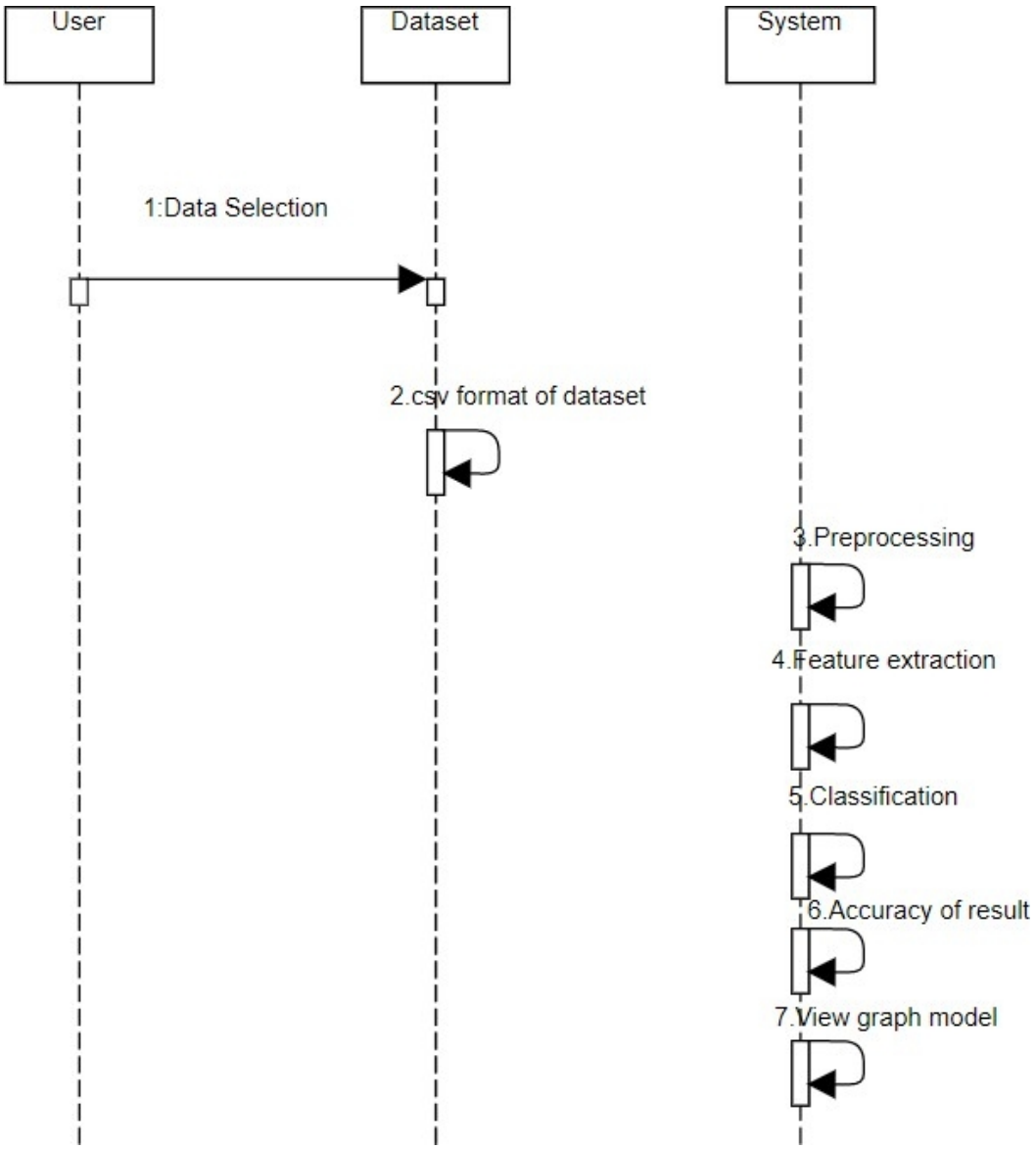
4.3.2 Activity Diagram



4.3.3 Class Diagram



4.3.4 Sequence Diagram



CHAPTER-5
SYSTEM ARCHITECTURE

5.1 Architecture Overview:

The overview of architecture is to test the image which undergoes preprocessing and the images are parted using feature extraction and undergoes with CNN classifier algorithm and the data set images are also parted using feature extraction, dataset image means in which the system organizes digital pictures into a location for fast sharing and retrieve- ability and these features are all trained and classified into CNN classifier and then the corresponding stages are classified.

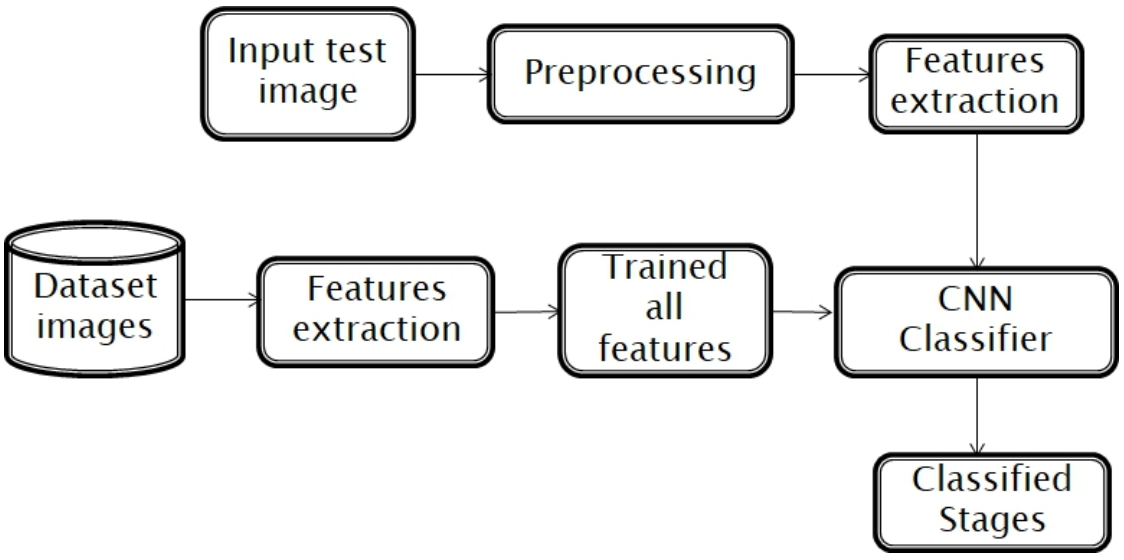
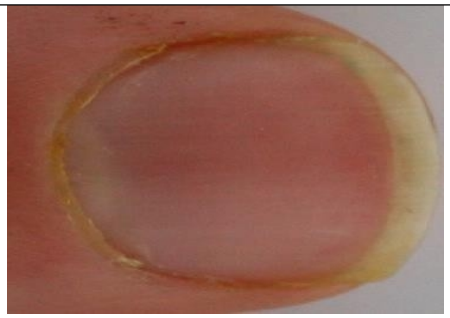
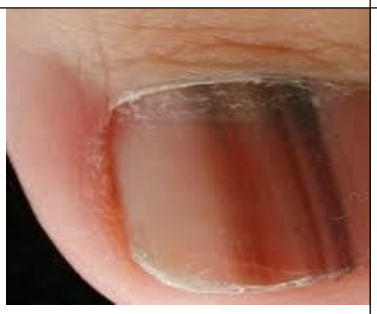


Figure 5.1 System architecture

5.2 Module Design Specification:

Exploring the Dataset

It is the action of retrieving both normal and abnormal nail image for further analysis. Which are all having either .jpg , .png , .bmp format. The dataset used for this project will be the nail images(Melanoma)from Kaggle. The dataset consists of training data, validation data, and testing data. The training data consists of 5,216 nail images with 3,875 images shown to have Melanoma, nail images shown to be Onycholysis and 1,341 images shown to be normal. The validation data is relatively small with only 16 images with 8 cases of Melanoma,8 cases of Onycholysis and 8 normal cases. The testing data consists of 624 images split between 390 Melanoma cases nail images Onycholysis cases and 234 normal cases

	Normal	Melanoma	Onycholysis
Train			
Test			

Validate			

Pre-processing

- Image pre-processing is the term for operation on images at lowest level of operation.
- The human nail image is given as input to the pre-processor.
- If the image are of poor contrast, the pre-processor will enhance the contrast for clear classification type.

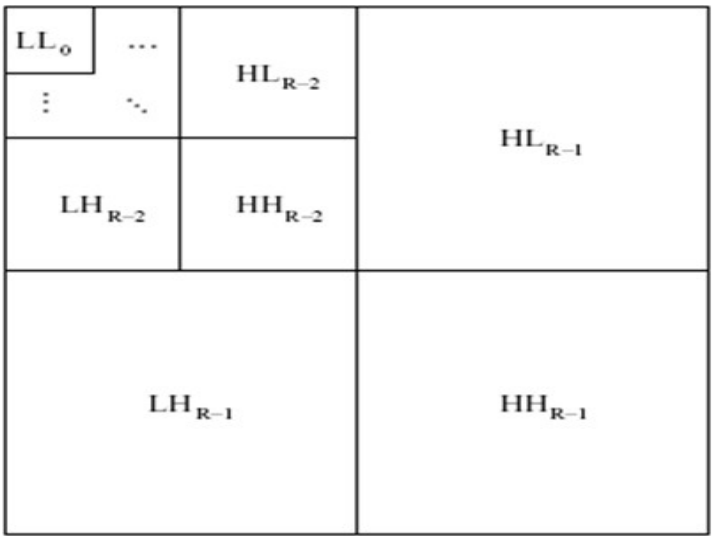
Steps involved:

- Acquire the dataset
- Resizing
- Conversion and
- Filtering

DWT

- Discrete Wavelet Transform is used in lossless image compression of gray level image.
- High quality images that require large storage are to be compressed.
- DWT transforms a discrete signal . L represent the low-pass filtered signal L(low frequency)allows the perfect reconstruction of original Image.

DWT Structure



LL: Horizontal Low pass & Vertical Low pass

LH: Horizontal Low pass & Vertical High pass

HL: Horizontal High pass & Vertical Low pass

HH: Horizontal High pass & Vertical High pass

Steps:

1. Digitize the source image into signal.

2. Decompose signal to wavelet (sub bands) LL,LH,HL,HH

3. DWT retains images from LL to produce next level of decomposition , because the low frequency images has finer frequency and time resolution than high frequency images.

4. For each level of decomposition DWT produces 4 images and size is reduced to 1/4 of original image.

Data Labelling

The steps followed in data labelling are

- Labelling :
  Labelling is the initial process of containing the images from which they are categorized into classes.

- Assigning Classes :
  Classes are assigned as normal and abnormal , normal indicates disease less nail and abnormal indicates disease affected nail.

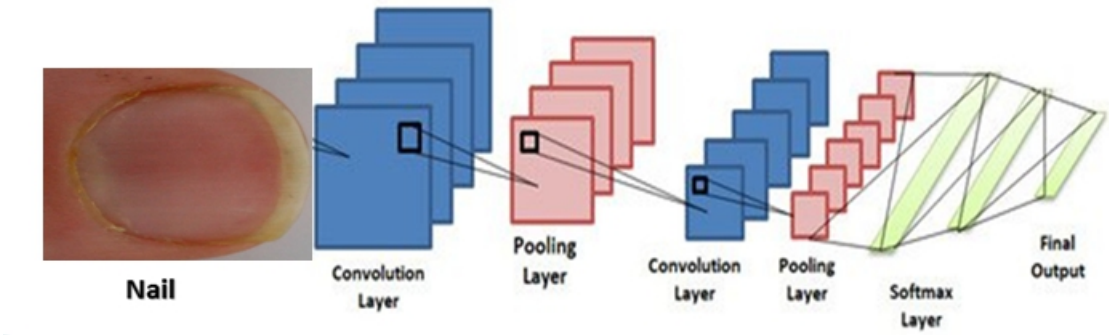
- Allocating Index:

Classes are allocated with index. Normal as index 0 and abnormal as index 1.

- Setting the Path:
  Finally path of the images are set to the program.

CNN (Convolutional Neural Network) Classification

- ✓ CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images.
- ✓ Their applications can be seen widely in the medical images analysis.
- ✓ The term 'Convolution" in CNN denotes that two images can be represented as **matrices** which are multiplied to give an output that is used to extract features from the image.



Layers of CNN

1. Input Layer

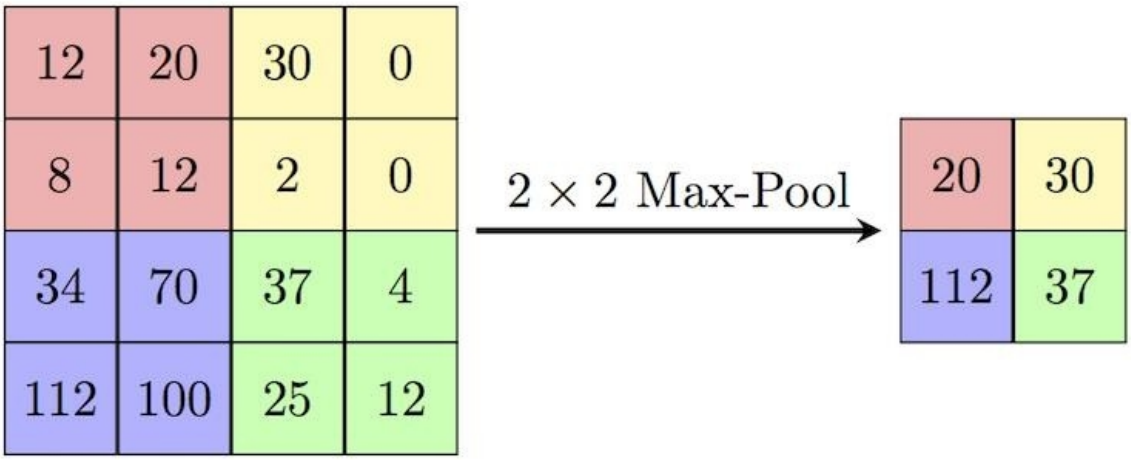
Input layer generally represents the pixel matrix of the image and brings the initial data into the system for further processing by subsequent layers

2. Convolutional Layer

This **layer** performs task called "**convolution**". The convolutional **layer** is **used** to extract image features and a filter of a particular size MxM. The dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM). The sum of those dot products is used to produce the output image which is fed as input to next layer.

3. Pooling Layer

Pooling layer is obtained by applying pooling operator to aggregate information within each small region of the input feature channels and then down sampling the results. Used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn . The most common approach used in pooling is max pooling**.**



4. Fully-Connected Layer

The input to the fully connected layer(FC) is the output from the final Pooling , which is flattened and then fed into the fully connected layer. This layer consist of the weights and biases along with the class score for each of the classification

category. Fully connected layers connect every neuron in one layer to every neuron in another layer.

5. Output Layer

The output layer is responsible for producing the final result. There must always be one output layer in a neural network. The output layer takes in the inputs which are passed in from the layers before it, performs the calculations via its neurons and then the output is computed.

Softmax Function

The softmax function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution. That is, softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels. The final output is calculated using softmax which gives the probability of each class for given features.

Training

The sample of data used to fit the model. The actual dataset that used to train the model (weights and biases in the case of CNN). The model sees and learns from this data. A number of recommended approaches in the toolkit that could be experimented with,

- ✓ Weight Initialization.
- ✓ Learning Rate.
- ✓ Activation Functions.
- ✓ Network Topology.
- ✓ Batches and Epochs.
- ✓ Regularization.
- ✓ Optimization and Loss.

- ✓ Early Stopping.

Testing

The dataset contains a test folder and in a test.csv file, the details related to the image path and their respective class labels are specified. The image path and labels are extracted using pandas. Then to predict the model, the images are resized to 30×30 pixels and numpy array containing all image data are made. From the sklearn.metrics, the confusion_matrix is imported and observed how the model predicted the actual labels. As a result, 95% accuracy of the model is achieved.

Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

5.3 Algorithm

Initial Setup

1. Install the anaconda navigator and python idle.
2. Create the environment (nail Packages) in the anaconda navigator with the required packages installed.

Dataset

1. Import the necessary packages.
2. Load the train images.
3. Load the test images.
4. Read the train images.
5. Read the test images.

Cnn_train_test

1. Import the necessary packages.
2. Initialize the filter_size and num_filters in the convolutional layers.
3. Specify the required classes (Pneumonia, Normal, Covid) for the categorization.
4. Specify the train_path, test_path, checkpoint_dir and val_path.
5. Read the dataset details.
6. Get some random images and their labels from the train set.
7. Plot the images and labels using the helper function.
8. Create new weights with the given shape of the image.
9. Create new biases, one for each filter.
10. Add the biases to the result of convolution.
11. Perform max pooling and padding in the pooling layer.
12. In the flatten layer, get the shape of the input image.
13. Create new weights and biases.
14. Classification and optimization are performed in the fully connected layer.
15. In the Tensorflow session, the epochs with validation loss, validation accuracy and training accuracy are predicted.
16. To measure the accuracy, the confusion matrix is constructed.

Output Procedure

1. Open the terminal with the environment created in the anaconda navigator.
2. Move to the particular Directory.
3. Run the cnn_train_test.py python file.
4. Accuracy of training, testing is obtained with the confusion matrix.
5. Actual output after classification is displayed with the input image as frame.

CHAPTER-6

SYSTEM IMPLEMENTATION

6.1 Coding

Dataset.py

```
import os
import glob
import numpy as np
import cv2
from sklearn.utils import shuffle
def load_train(train_path, image_size, classes):
    images = []
    labels = []
    ids = []
    cls = []
print('Reading training images')
    for fld in classes:   # assuming data directory has a separate folder for each class, and that each folder is named after the class
        index = classes.index(fld)
        print('Loading {} files (Index: {})'.format(fld, index))
        path = os.path.join(train_path, fld, '*g')
        files = glob.glob(path)
        for fl in files:
            image = cv2.imread(fl)
            image = cv2.resize(image, (image_size, image_size), cv2.INTER_LINEAR)
            images.append(image)
            label = np.zeros(len(classes))
            label[index] = 1.0
            labels.append(label)
            flbase = os.path.basename(fl)
```

```
            ids.append(flbase)
            cls.append(fld)
    images = np.array(images)
    labels = np.array(labels)
    ids = np.array(ids)
    cls = np.array(cls)
 return images, labels, ids, cls
def load_test(test_path, image_size):
  path = os.path.join(test_path, '*g')
  files = sorted(glob.glob(path))

  X_test = []
  X_test_id = []
  print("Reading test images")
  for fl in files:
      flbase = os.path.basename(fl)
      img = cv2.imread(fl)
      img = cv2.resize(img, (image_size, image_size), cv2.INTER_LINEAR)
      X_test.append(img)
      X_test_id.append(flbase)

  ### because we're not creating a DataSet object for the test images, normalization
happens here
  X_test = np.array(X_test, dtype=np.uint8)
  X_test = X_test.astype('float32')
  X_test = X_test / 255
return X_test, X_test_id
class DataSet(object):
```

```
  def__init__(self, images, labels, ids, cls):
    """Construct a DataSet. one_hot arg is used only if fake_data is true."""

self._num_examples = images.shape[0]



    # Convert shape from [num examples, rows, columns, depth]
    # to [num examples, rows*columns] (assuming depth == 1)
    # Convert from [0, 255] -> [0.0, 1.0].

    images = images.astype(np.float32)
    images = np.multiply(images, 1.0 / 255.0)

    self._images = images
    self._labels = labels
    self._ids = ids
    self._cls = cls
    self._epochs_completed = 0
    self._index_in_epoch = 0

  @property
  def images(self):
    return self._images

  @property
  def labels(self):
    return self._labels

  @property
```

```
def ids(self):
  return self._ids

@property
def cls(self):
  return self._cls

@property
def num_examples(self):
  return self._num_examples

@property
def epochs_completed(self):
  return self._epochs_completed

def next_batch(self, batch_size):
  """Return the next `batch_size` examples from this data set."""
  start = self._index_in_epoch
  self._index_in_epoch += batch_size

  if self._index_in_epoch > self._num_examples:
    # Finished epoch
    self._epochs_completed += 1

    # # Shuffle the data (maybe)
    # perm = np.arange(self._num_examples)
    # np.random.shuffle(perm)
    # self._images = self._images[perm]
    # self._labels = self._labels[perm]
```

```
      # Start next epoch

      start = 0
      self._index_in_epoch = batch_size
      assert batch_size <= self._num_examples
    end = self._index_in_epoch

    return self._images[start:end], self._labels[start:end], self._ids[start:end],
self._cls[start:end]


def read_train_sets(train_path, image_size, classes, validation_size=0):
  class DataSets(object):
    pass
  data_sets = DataSets()
 images, labels, ids, cls = load_train(train_path, image_size, classes)
  images, labels, ids, cls = shuffle(images, labels, ids, cls)  # shuffle the data
  if isinstance(validation_size, float):
    validation_size = int(validation_size * images.shape[0])
validation_images = images[:validation_size]
validation_labels = labels[:validation_size]
  validation_ids = ids[:validation_size]
validation_cls = cls[:validation_size]
train_images = images[validation_size:]
train_labels = labels[validation_size:]
train_ids = ids[validation_size:]
train_cls = cls[validation_size:]
data_sets.train = DataSet(train_images, train_labels, train_ids, train_cls)
```

```
  data_sets.valid = DataSet(validation_images, validation_labels, validation_ids,
validation_cls)
 return data_sets


def read_test_set(test_path, image_size):
  images, ids  = load_test(test_path, image_size)
  return images, ids
```

cnn_test_train.py

```
import  time
import math
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import dataset
import cv2
import os
from sklearn.metrics import confusion_matrix
from datetime import timedelta
# Convolutional Layer 1.
filter_size1 = 3
num_filters1 = 32
# Convolutional Layer 2.
filter_size2 = 3
num_filters2 = 32
# Convolutional Layer 3.
```

```
filter_size3 = 3
num_filters3 = 64

# Fully-connected layer.
fc_size = 128             # Number of neurons in fully-connected layer.
# Number of color channels for the images: 1 channel for gray-scale.
num_channels = 3
# image dimensions (only squares for now)
img_size = 32
# Size of image when flattened to a single dimension
img_size_flat = img_size * img_size * num_channels# Tuple with height and width
of images used to reshape arrays.
img_shape = (img_size, img_size)
# class info
classes = ['Abnormal','Normal','   ']
num_classes = len(classes)
# batch size
batch_size = 1
# validation split
validation_size = .16
# how long to wait after validation loss stops improving before terminating training
early_stopping = None

train_path = 'data/'
test_path = 'test/'
checkpoint_dir = "models/"
data = dataset.read_train_sets(train_path, img_size, classes,
validation_size=validation_size)
test_images, test_ids = dataset.read_test_set(test_path, img_size)
```

```
print("Size of:")
print("- Training-set:/t/t{}".format(len(data.train.labels)))
print("- Test-set:/t/t{}".format(len(test_images)))
print("- Validation-set:/t{}".format(len(data.valid.labels)))
# Get some random images and their labels from the train set.
images, cls_true  = data.train.images, data.train.cls
# Plot the images and labels using our helper-function above.
#plot_images(images=images, cls_true=cls_true)
##Helper-functions for creating new variables
def new_weights(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.05))
def new_biases(length):
    return tf.Variable(tf.constant(0.05, shape=[length]))
def new_conv_layer(input,              # The previous layer.
                   num_input_channels, # Num. channels in prev. layer.
                   filter_size,        # Width and height of each filter.
                   num_filters,        # Number of filters.
                   use_pooling=True):  # Use 2x2 max-pooling.

    shape = [filter_size, filter_size, num_input_channels, num_filters]

    # Create new weights aka. filters with the given shape.
    weights = new_weights(shape=shape)

    # Create new biases, one for each filter.
    biases = new_biases(length=num_filters)
    layer = tf.nn.conv2d(input=input,
                         filter=weights,
```

```
                    strides=[1, 1, 1, 1],
                    padding='SAME')

    # Add the biases to the results of the convolution.
    # A bias-value is added to each filter-channel.
    layer += biases

    if use_pooling:
        layer = tf.nn.max_pool(value=layer,
                               ksize=[1, 2, 2, 1],
                               strides=[1, 2, 2, 1],padding='SAME')

    layer = tf.nn.relu(layer)
    return layer, weights

def flatten_layer(layer):
    # Get the shape of the input layer.
    layer_shape = layer.get_shape()
    num_features = layer_shape[1:4].num_elements()
    layer_flat = tf.reshape(layer, [-1, num_features])
    return layer_flat, num_features

def new_fc_layer(input,
                 num_inputs,
                 num_outputs,
                 use_relu=True):

    # Create new weights and biases.
```

```
    weights = new_weights(shape=[num_inputs, num_outputs])
    biases = new_biases(length=num_outputs)
    layer = tf.matmul(input, weights) + biases

    # Use ReLU?
    if use_relu:
        layer = tf.nn.relu(layer)

    return layer

x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='x')
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)


##Convolutional Layer 1
layer_conv1, weights_conv1 = \
    new_conv_layer(input=x_image,
                   num_input_channels=num_channels,
                   filter_size=filter_size1,
                   num_filters=num_filters1,
                   use_pooling=True)


layer_conv1   # layer 1 output image data

layer_conv2, weights_conv2 = \
    new_conv_layer(input=layer_conv1,
```

```
                num_input_channels=num_filters1,
                filter_size=filter_size2,
                num_filters=num_filters2,
                use_pooling=True)

layer_conv3, weights_conv3 = \
    new_conv_layer(input=layer_conv2,
                num_input_channels=num_filters2,
                filter_size=filter_size3,
                num_filters=num_filters3,
                use_pooling=True)
layer_conv2
layer_conv3
##Flatten Layer

layer_flat, num_features = flatten_layer(layer_conv3)
## Fully-Connected Layer
layer_fc1 = new_fc_layer(input=layer_flat,
                    num_inputs=num_features,
                    num_outputs=fc_size,
                    use_relu=True)
layer_fc2 = new_fc_layer(input=layer_fc1,
                    num_inputs=fc_size,
                    num_outputs=num_classes,
                    use_relu=False)

## class prediction
y_pred = tf.nn.softmax(layer_fc2)
y_pred_cls = tf.argmax(y_pred, dimension=1)
```

```
## Cost-function to be optimized
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2,
                                                        labels=y_true)


##Optimization Method
cost = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
##Performance Measures
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

####TENSORFLOW SESSION
session = tf.Session()

session.run(tf.initialize_all_variables())
train_batch_size = batch_size


##Helper-function to perform optimization iterations
def print_progress(epoch, feed_dict_train, feed_dict_validate, val_loss):
    # Calculate the accuracy on the training-set.
    acc = session.run(accuracy, feed_dict=feed_dict_train)
    val_acc = session.run(accuracy, feed_dict=feed_dict_validate)
    msg = "Epoch {0} --- Training Accuracy: {1:>6.1%}, Validation Accuracy:
{2:>6.1%}, Validation Loss: {3:.3f}"
    print(msg.format(epoch + 1, acc, val_acc, val_loss))
```

```
# Counter for total number of iterations performed so far.
total_iterations = 0

def optimize(num_iterations):
    # Ensure we update the global variable rather than a local copy.
    global total_iterations

    # Start-time used for printing time-usage below.
    start_time = time.time()

    best_val_loss = float("inf")
    patience = 0

    for i in range(total_iterations,
                   total_iterations + num_iterations):
        x_batch, y_true_batch, _, cls_batch = data.train.next_batch(train_batch_size)
        x_valid_batch, y_valid_batch, _, valid_cls_batch =
data.valid.next_batch(train_batch_size)

        # Convert shape from [num examples, rows, columns, depth]
        # to [num examples, flattened image shape]

        x_batch = x_batch.reshape(train_batch_size, img_size_flat)
        x_valid_batch = x_valid_batch.reshape(train_batch_size, img_size_flat)

        # Put the batch into a dict with the proper names
        # for placeholder variables in the TensorFlow graph.
        feed_dict_train = {x: x_batch,
```

```
                    y_true: y_true_batch}

        feed_dict_validate = {x: x_valid_batch,
                              y_true: y_valid_batch}
        session.run(optimizer, feed_dict=feed_dict_train)


        # Print status at end of each epoch (defined as full pass through training dataset).
        if i % int(data.train.num_examples/batch_size) == 0:
            val_loss = session.run(cost, feed_dict=feed_dict_validate)
            epoch = int(i / int(data.train.num_examples/batch_size))

            print_progress(epoch, feed_dict_train, feed_dict_validate, val_loss)

            if early_stopping:
                if val_loss < best_val_loss:
                    best_val_loss = val_loss
                    patience = 0
                else:
                    patience += 1

                if patience == early_stopping:
                    break

    # Update the total number of iterations performed.
    total_iterations += num_iterations

    # Ending time.
```

```
    end_time = time.time()

    # Difference between start and end-times.
    time_dif = end_time - start_time

    # Print the time-usage.
    print("Time elapsed: " + str(timedelta(seconds=int(round(time_dif)))))

print(total_iterations)
##Helper-function to plot example errors
def plot_example_errors(cls_pred, correct):
    incorrect = (correct == False)

    # Get the images from the test-set that have been
    # incorrectly classified.
    images = data.valid.images[incorrect]

    # Get the predicted classes for those images.
    cls_pred = cls_pred[incorrect]

    # Get the true classes for those images.
    cls_true = data.valid.cls[incorrect]

    # Plot the first 9 images.
##    plot_images(images=images[0:9],
##                cls_true=cls_true[0:9],
##                cls_pred=cls_pred[0:9])
def plot_confusion_matrix(cls_pred):
    cls_true = data.valid.cls
```

```
    # Get the confusion matrix using sklearn.
    cm = confusion_matrix(y_true=cls_true,
                          y_pred=cls_pred)

    # Print the confusion matrix as text.
    print(cm)
    # Plot the confusion matrix as an image.
    plt.matshow(cm)

    # Make various adjustments to the plot.
    plt.colorbar()
    tick_marks = np.arange(num_classes)
    plt.xticks(tick_marks, range(num_classes))
    plt.yticks(tick_marks, range(num_classes))
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
##Helper-function for showing the performance
def print_validation_accuracy(show_example_errors=False,
                              show_confusion_matrix=False):

    # Number of images in the test-set.
    num_test = len(data.valid.images)
    cls_pred = np.zeros(shape=num_test, dtype=np.int)
    i = 0

    while i < num_test:
        # The ending index for the next batch is denoted j.
```

```
    j = min(i + batch_size, num_test)

    # Get the images from the test-set between index i and j.
    images = data.valid.images[i:j, :].reshape(batch_size, img_size_flat)



    # Get the associated labels.
    labels = data.valid.labels[i:j, :]

    # Create a feed-dict with these images and labels.
    feed_dict = {x: images,
                 y_true: labels}

    # Calculate the predicted class using TensorFlow.
    cls_pred[i:j] = session.run(y_pred_cls, feed_dict=feed_dict)

    # Set the start-index for the next batch to the
    # end-index of the current batch.
    i = j

cls_true = np.array(data.valid.cls)
cls_pred = np.array([classes[x] for x in cls_pred])

# Create a boolean array whether each image is correctly classified.
correct = (cls_true == cls_pred)
correct_sum = correct.sum()
acc = float(correct_sum) / num_test

# Print the accuracy.
```

```
    msg = "Accuracy on Test-Set: {0:.1%} ({1} / {2})"
    print(msg.format(acc, correct_sum, num_test))

    # Plot some examples of mis-classifications, if desired.
    if show_example_errors:
        print("Example errors:")
        plot_example_errors(cls_pred=cls_pred, correct=correct)

    # Plot the confusion matrix, if desired.
    if show_confusion_matrix:
        print("Confusion Matrix:")
        plot_confusion_matrix(cls_pred=cls_pred)


optimize(num_iterations=400) # We performed 100 iterations above.
print_validation_accuracy(show_example_errors=True)
print_validation_accuracy(show_example_errors=True,
show_confusion_matrix=True)

##### TESTING IMAGE INPUT  ######

inputface = cv2.imread('015.jpg')
cv2.imshow("frame",inputface)
inputface = cv2.resize(inputface, (img_size, img_size), cv2.INTER_LINEAR) / 255
##plt.imshow(inputface.reshape(img_size, img_size, num_channels))


def sample_prediction(test_im):
```

```
    feed_dict_test = {
        x: test_im.reshape(1, img_size_flat),
        y_true: np.array([[2,1,0]])
    }

    test_pred = session.run(y_pred_cls, feed_dict=feed_dict_test)
    return classes[test_pred[0]]

print("output test data: {}".format(sample_prediction(inputface)))



def plot_conv_weights(weights, input_channel=0):
    # Assume weights are TensorFlow ops for 4-dim variables
    # e.g. weights_conv1 or weights_conv2.

    # Retrieve the values of the weight-variables from TensorFlow.
    # A feed-dict is not necessary because nothing is calculated.
    w = session.run(weights)

    # Get the lowest and highest values for the weights.
    # This is used to correct the colour intensity across
    # the images so they can be compared with each other.

w_min = np.min(w)
    w_max = np.max(w)

    # Number of filters used in the conv. layer.
    num_filters = w.shape[3]
```

```
    # Number of grids to plot.
    # Rounded-up, square-root of the number of filters.
    num_grids = math.ceil(math.sqrt(num_filters))

    # Create figure with a grid of sub-plots.
    fig, axes = plt.subplots(num_grids, num_grids)

    # Plot all the filter-weights.
    for i, ax in enumerate(axes.flat):
        # Only plot the valid filter-weights.
        if i<num_filters:
            # Get the weights for the i'th filter of the input channel.
            # See new_conv_layer() for details on the format
            # of this 4-dim tensor.
            img = w[:, :, input_channel, i]

            # Plot image.
            ax.imshow(img, vmin=w_min, vmax=w_max,
                      interpolation='nearest', cmap='seismic')

        # Remove ticks from the plot.
        ax.set_xticks([])
        ax.set_yticks([])

    # Ensure the plot is shown correctly with multiple plots
    # in a single Notebook cell.
    plt.show()

def plot_conv_layer(layer, image):
```

```
# Assume layer is a TensorFlow op that outputs a 4-dim tensor
# which is the output of a convolutional layer,
# e.g. layer_conv1 or layer_conv2.

image = image.reshape(img_size_flat)

# Create a feed-dict containing just one image.
# Note that we don't need to feed y_true because it is
# not used in this calculation.
feed_dict = {x: [image]}

# Calculate and retrieve the output values of the layer
# when inputting that image.
values = session.run(layer, feed_dict=feed_dict)

# Number of filters used in the conv. layer.
num_filters = values.shape[3]

# Number of grids to plot.
# Rounded-up, square-root of the number of filters.
num_grids = math.ceil(math.sqrt(num_filters))

# Create figure with a grid of sub-plots.
fig, axes = plt.subplots(num_grids, num_grids)

# Plot the output images of all the filters.
for i, ax in enumerate(axes.flat):
```

```
# Only plot the images for valid filters.
        if i<num_filters:
            # Get the output image of using the i'th filter.
            # See new_conv_layer() for details on the format
            # of this 4-dim tensor.
            img = values[0, :, :, i]

            # Plot image.
            ax.imshow(img, interpolation='nearest', cmap='binary')

        # Remove ticks from the plot.
        ax.set_xticks([])
        ax.set_yticks([])

    # Ensure the plot is shown correctly with multiple plots
    # in a single Notebook cell.
    plt.show()


def plot_image(image):
    plt.imshow(image.reshape(img_size, img_size, num_channels),
               interpolation='nearest')
    plt.show()

image1 = test_images[0]
plot_image(image1)

plot_conv_weights(weights=weights_conv1)
```

```
plot_conv_layer(layer=layer_conv1, image=image1)


plot_conv_weights(weights=weights_conv2, input_channel=0)
plot_conv_layer(layer=layer_conv2, image=image1)
session.close()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

CHAPTER-7
SYSTEM TESTING

7.1Performance analysis

The performance of the CNNs trained with the nail dataset was estimated by the classification performance of the models with the normal, melanoma and Onycholysis validation datasets.The performance of fine image selector helps in assessing image quality with the change in the illumination and reduction noise level of the images. The levels of brightness and noise were gradually reduced to classify the image easily. If the validation loss decreases then the accuracy will increase. The number of epochs should be as high as possible and terminate training based on the error rates.An epoch is one learning cycle where the learner sees the whole training data set. Here we are having 15 epochs and we are getting 100% validation and training accuracy. In this experiment we found that using color feature of nail image average 80% results are correctly matched with training set data during three tests conducted and we are getting 80% accuracy on test dataset

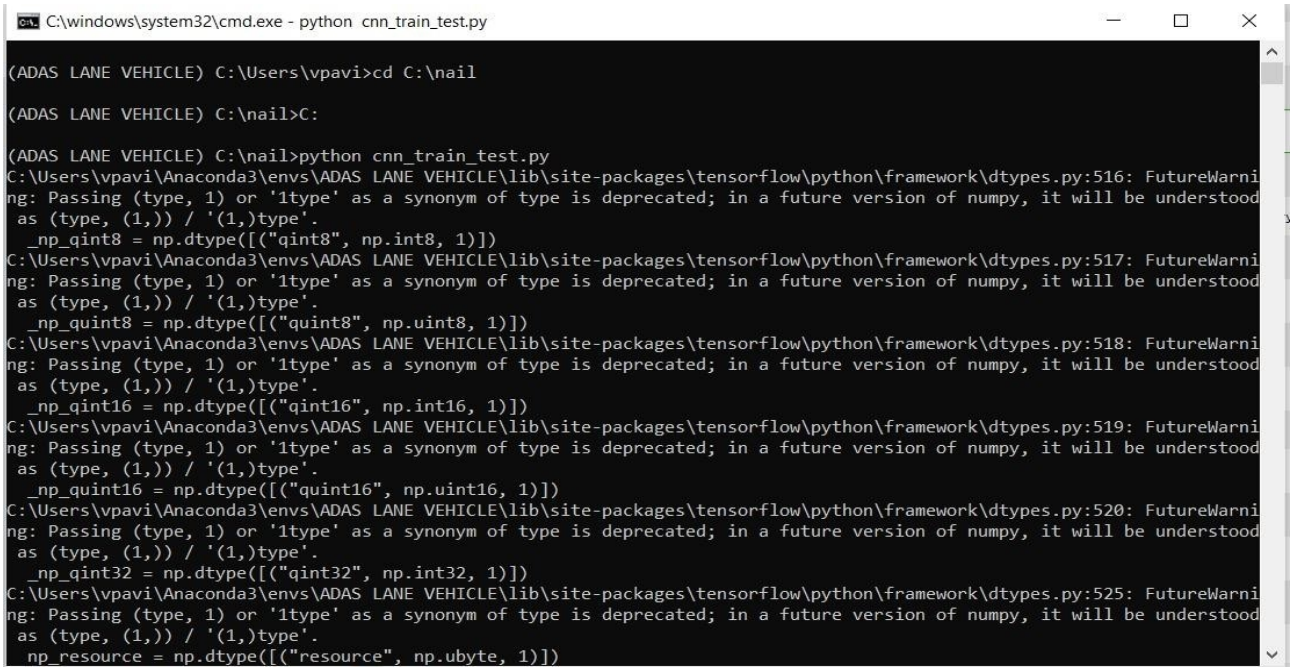
CHAPTER-8
CONCLUSION

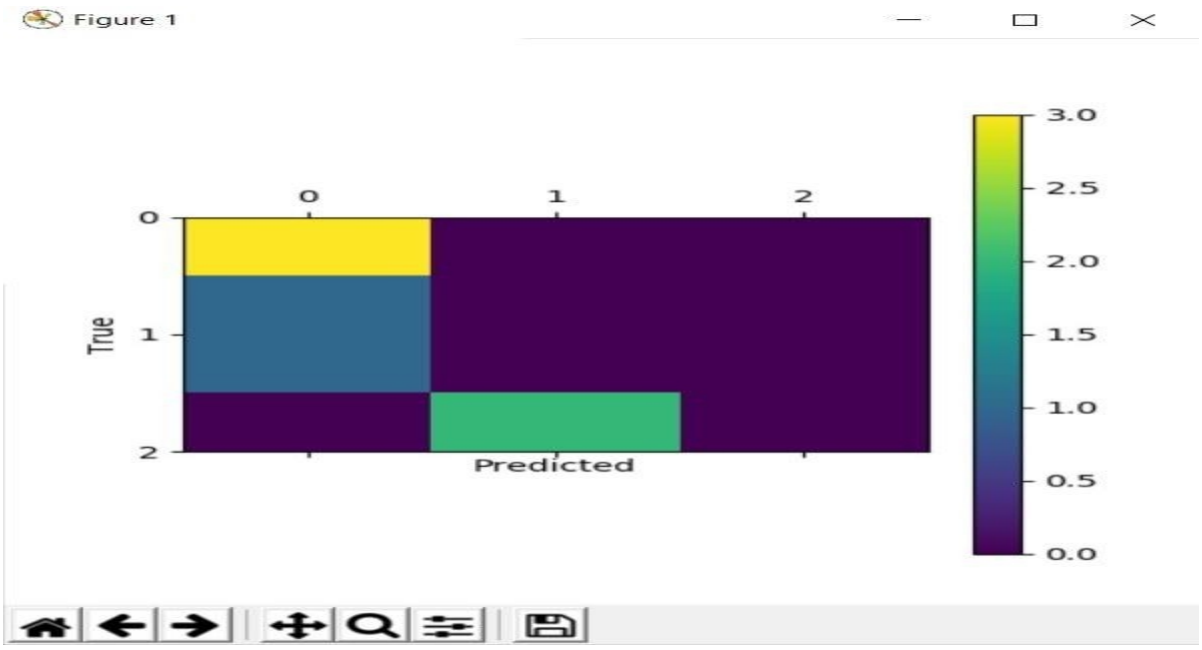
8.1 Conclusion and future enhancements

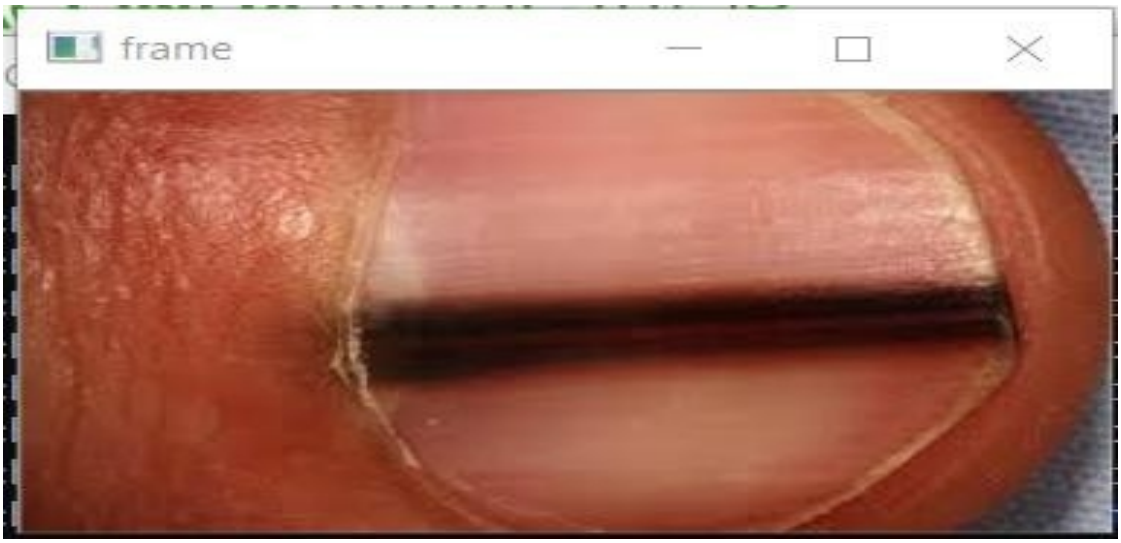
In the proposed technique we have trained a model that classifies the disease based on the pattern on the nail. This proposed system is able to predict the disease for the respective pattern of the nail with high accuracy. It is able to identify the small patterns also such that providing a system with higher success rate. The limitations of the existing model are eliminated by the proposed model. Moreover in the proposed system only the images of nails of fingers have been used for classifying the diseases, but in future we can combine other features of human body and predict various diseases based on the symptoms of patient and hence would be able to detect a lot of diseases with good precision and accuracy.

APPENDICES

A.1 Sample screens






frame

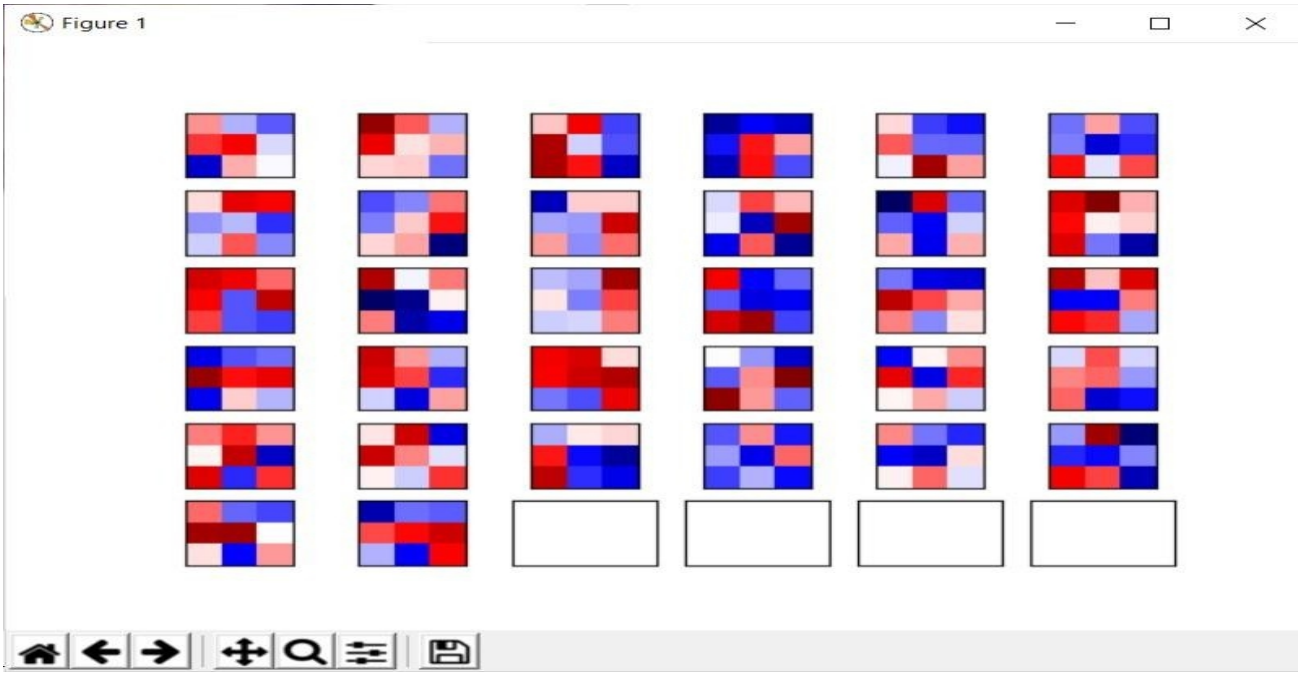


Figure 1

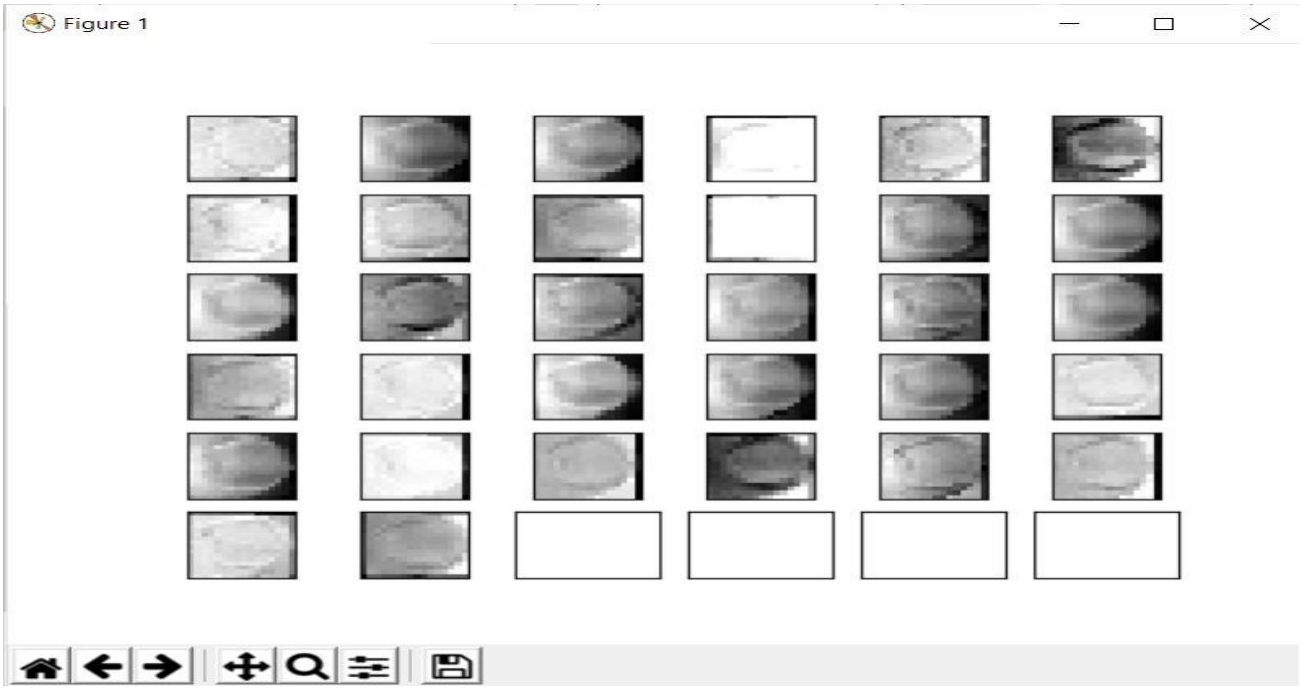


Figure 1


Figure 1


```
C:\windows\system32\cmd.exe - python  cnn_train_test.py
Epoch 4 --- Training Accuracy: 100.0%, Validation Accuracy:   0.0%, Validation Loss: 1.094
Epoch 5 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 1.081
Epoch 6 --- Training Accuracy:   0.0%, Validation Accuracy:   0.0%, Validation Loss: 1.084
Epoch 7 --- Training Accuracy:   0.0%, Validation Accuracy: 100.0%, Validation Loss: 1.067
Epoch 8 --- Training Accuracy:   0.0%, Validation Accuracy: 100.0%, Validation Loss: 1.036
Epoch 9 --- Training Accuracy:   0.0%, Validation Accuracy: 100.0%, Validation Loss: 1.065
Epoch 10 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.973
Epoch 11 --- Training Accuracy: 100.0%, Validation Accuracy:   0.0%, Validation Loss: 1.060
Epoch 12 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.890
Epoch 13 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.721
Epoch 14 --- Training Accuracy: 100.0%, Validation Accuracy:   0.0%, Validation Loss: 1.099
Epoch 15 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%, Validation Loss: 0.547
Time elapsed: 0:00:02
cnn_train_test.py:326: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning
, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use
 e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note
link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecation
s
  cls_pred = np.zeros(shape=num_test, dtype=np.int)
Accuracy on Test-Set: 80.0% (4 / 5)
Example errors:
Accuracy on Test-Set: 80.0% (4 / 5)
Example errors:
Confusion Matrix:
[[3 0 0]
 [0 1 0]
 [0 1 0]]
output test data: melanoma
```

A.2Publications

Journal name-International Journal of engineering Science and computing (IJESC)

Paper title-Early stage disease diagnosis using human nail in Image processing

Publication issue-Volume 11, Issue 04, April-2021

ISSN 2321 3361 © 2021 IJESC

IJESC

Research Article Volume 11 Issue No.04

Early Stage Diseases Diagnosis using Human Nail in Image Processing

A. Kanchna[1], D. Navanisha[2], V. Pavithra[3], D. Reshika[4]
Assistant Professor[1], UG Scholar[2, 3, 4]
Panimalar Engineering College, Chennai, Tamilnadu, India

Abstract:
Human's hand nail features are used to identify many diseases at early stage of diagnosis as nail is farthest from heart and last to receive oxygen in our body part. Nail colour is used to identify the diseases. The proposed system guides in such scenario to take decision in disease diagnosis. This paper contend deep learning, a convolutional neural network to classify disease diagnosis from nail images. Nail is taken as a input image in proposed system. The image of a nail is processed by system and extract features of nail which is used for disease diagnosis. By using Kaggle website the first training set data is prepared from nail images of affected nail. The input nail image is extracted from the feature is compared with the training data set to get result. In this experiment we found that using color feature of nail image average 65% results are correctly matched with training set data during three tests conducted. Human nail consist of various features, out of which proposed system uses nail color changes for disease diagnosis.

Keywords: Human Nail, Deep Learning, CNN.

I. INTRODUCTION

In healthcare domain many diseases are going to be predicted by observing color of human nails. Doctors observe nails of patient to induce assistance in diseases identification. Usually pink nails indicate healthy human. The need of system to research nails for diseases prediction is because human eye has subjectivity about colors, having limitation in resolution and tiny amount of color change in few pixels on nail wouldn't be highlighted to human eyes which may end in wrong result whereas computer recognizes small color changes on nail. The proposed system will extract color feature of human nail image for disease prediction. The system that target image recognition on the thought of human nail color analysis. Many diseases could even be identified by analyzing nails of human hands. During this method human nail image is captured using camera. Captured image is uploaded to our system and region of interest from nail area is chosen from uploaded image manually. Selected area is then processed further for extracting features of nail like color of nail. This color feature of nail is matched using simple matcher algorithm for diseases prediction. In this way the system is useful in prediction of diseases in their initial stages In Literature study we mentioned number of the diseases in their initial stages. In Literature study we mentioned number of the diseases with its related color changes in nails Human nail are going to be used for the prediction of various systemic and dermatological diseases.

The proposed system – Nail Image Processing System helps us to create a model which could perform the analysis of human nail and thereby help us in predicting various diseases. Common signs which be noticeable around the nail are discoloration of nail to black, white, yellow or green, thickening of nail, dry or scaly skin around the nail. This project contend a deep convolutional network to classify diseases from images. This work has been tested on our dataset and it ends up to great performance in feature extraction. This proposed system will help the doctors within the early diagnosis of diseases.

II. RELATED WORKS

Ting wie-houe proposed that the digital image processing plays a key role in medical imaging. Nail diagnosis is one in every of the methods in medical imaging to predict the diseases. Nails can reflect the current health condition, genetically inheritance information, and historical information of drug or alcohol usage for the past months or perhaps a year, etc. This paper explores the prevailing research works associated with nail plate and nail matrix as tool for bio-metric system, nail fold capillaries to spot the disease severity levels & affected organs, nail surface as evidence in forensic science to spot the chemical effects, nail samples to spot drug intake and abuse etc. So that, Nail is analyzed by various imaging types and processing algorithms to acknowledge the person's uniqueness, health condition and its history. This paper also identifies the research challenges and issues. ace shape of nails is given within the article DM Shah proposed that the paper is targeted on the system of image recognition on the idea of color analysis. In healthcare domain, study of human nail color is incredibly important. Many diseases might be identified by analyzing nails of hands. Human eye has limitation in resolution also as subjectivity in color analysis. The proposed system is predicted on the algorithm which automatically extracts only nails' area from scanned back side of palm. These selected pixels are processed for further analysis. The system is computer based, so small dis continuities in color values also are also observed, and that we can detect color changes within the initial stage of disease. During this way, system is sort of useful in prediction diseases in their initial stages It is focused on the system of image recognition on the thought of color analysis. The proposed system is predicted on the algorithm which automatically extracts only nails area from scanned back side of palm (Region of Interest). These selected pixels are processed for further analysis using median filters. The system is computer

based, so small discontinuities in color values are observed, which we are able to detect color changes within the initial stage of disease. During this method, system is all fairness of useful in prediction of diseases in their initial stages. Dr. M. Renuka Devi proposed that this paper explores the prevailing research works associated with nail plate and nail matrix as tool for bio-metric system, nail fold capillaries to identify the disease severity levels & affected organs, nail surface as evidence in forensic science to identify the chemical effects, nail samples to identify drug intake and abuse etc. The proposed system relies supported the algorithm which automatically extracts only nails' area from scanned back side of palm. These selected pixels are processed for further analysis. The system is computer based, so small discontinuities in color values are observed, which we are able to detect color changes within the initial stage of disease. During this manner, system is type of useful in prediction of diseases in their initial stages disease. During this way, system is sort of useful in prediction of diseases in their initial stages. Computer vision Based identification of nail's surface shape article presents the principle and application of a computer-vision-based method of identifying the surface shape of human' s nail. The tactic first acquires two images of a human nail respectively with the identical source of illumination but different light angles, obtains from the two images the data on the nail and calculates the surface shape of the nail within the tip. The foremost algorithm of obtaining the surface shape of nails is given within the article.

III. PROPOSED SYSTEM

The main objective of this system design to provide an application for use in healthcare domain this is an advantage in terms of cost and time. The proposed system will take nail image as an input and will perform some processing on input image then finally it will predict probable disease this system can be used by people as well as by doctors in health care domain. This methodology uses a deep transfer learning algorithm using CNN that extracts features of the nail image and describes whether the condition is normal or abnormal. If the condition is abnormal it checks whether it is melanoma or onycholysis and accordingly it generates the result.

Advantages

- In the proposed technique we have trained a model that classifies the disease based on the pattern on the nail.
- This proposed system is able to predict the disease for the respective pattern of the nail with high accuracy.
- It is able to identify the small patterns also such that providing a system with higher success rate.
- The limitations of the existing model are eliminated by the proposed model

MODULE 1: EXPLORING THE DATASET

It is the action of retrieving both normal and abnormal nail image for further analysis. Which are all having either .jpg, png, .bmp format. The dataset used for this project will be the nail images (Melanoma) from Kaggle. The dataset consists of training data, validation data, and testing data. The training data consists of 5, 216 nail images with 3,875 images shown to have Melanoma, nail images shown to be Onycholysis and 1,341 images shown to be normal. The validation data is relatively small with only 16 images with 8 cases of Melanoma, 8 cases of Onycholysis and 8 normal cases. The testing data consists of 624 images split between 390 Melanoma cases nail images Onycholysis cases and 234 normal cases.

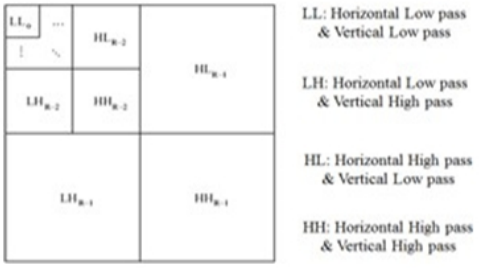
MODULE 2: PRE-PROCESSING

Image pre-processing is the term for operation on images at lowest level of operation. The human nail image is given as input to the pre-processor. If the images are of poor contrast, the pre-processor will enhance the contrast for clear classification type. Steps involved in pre-processing are collecting the dataset, resizing, conversion and filtering. In data collection, the data will be gathered using using Weka tool which consist of specific human nail diseases. Resizing images is a critical pre-processing step in computer vision. Resizing is altering the size of the image without cutting anything out. Principally, our deep learning models train faster on smaller images. An input image that is twice as large requires our network to learn from four times as many pixels — and that time adds up. Moreover, many deep learning model architectures require that our images are the same size and it helps to provide more accuracy in image. In pre-processing the image converted into gray scale image. Unnecessary noise is removed using median filter. These process are done to obtain the accuracy of the image. The gray scale image is compressed by DWT.

DWT (Discrete Wavelet Transform): Discrete Wavelet Transform is used in lossless image compression of gray level image. High quality images that require large storage are to be compressed. DWT transforms a discrete signal. L represent the low-pass filtered signal L (low frequency) allows the perfect reconstruction of original Image.

Steps

1. Digitize the source image into signal.
2. Decompose signal to wavelet (sub bands) LL, LH, HL, HH
3. DWT retains images from LL to produce next level of decomposition, because the low frequency images has finer frequency and time resolution than high frequency images.
4. For each level of decomposition DWT produces 4 images and size is reduced to 1/4 of original image.



MODULE 3: DATA LABELING

Data labeling is the process of identifying information adding one or more meaningful and informative labels to produce context. Labelers asked to tag all the images in a dataset where "does a photo contain a bird" is true. The tagging will be as simple yes/no or as granular as identifying the particular pixels in image related to the bird. The result's the trained model that may be want to make prediction on new data.

The steps followed in data labeling are:

Labeling:

Labeling is the initial process of containing the images from which they are categorized into classes.

Assigning Classes:
Classes are assigned as normal and abnormal, normal indicates disease less nail and abnormal indicates disease affected nail.

Allocating Index:
Classes are allocated with index. Normal as index 0 and abnormal as index 1.

Setting the Path:
Finally path of the images are set to the program.

MODULE 4: CONVOLUTIONAL NEURAL NETWORK CLASSIFICATION

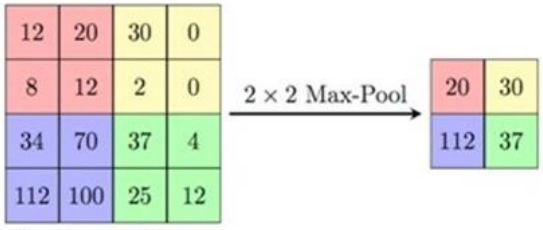
CNNs are a category of Deep Neural Networks which will recognize and classify particular features from images and are widely used for analyzing visual images.
Their applications can be seen widely within the medical images analysis.
The term 'Convolution" in CNN denotes that two images can be represented as matrices which are multiplied to give an output that is used to extract features from the image.

Convolutional Neural Network Layers

Input Layer
Input layer generally represents the pixel matrix of the image and brings the initial data into the system for further processing by subsequent layers

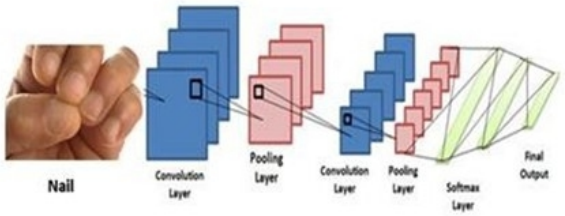
Convolutional layer
This layer performs task called "convolution". The con volutional layer is employed to extract image features and a filter of a specific size MxM. The dot product is taken between the filter and therefore the parts of the input image with reference to the scale of the filter (MxM). The sum of these dot products is employed to provide the output image which is fed as input to next layer

Pooling Layer
Pooling layer is obtained by applying pooling operator to aggregate information within each small region of the input feature channels and so down sampling the results. Accustomed reduce the dimensions of the feature maps. Thus, it reduces the quantity of parameters to hunt put . The foremost common approach employed in pooling is max pooling.

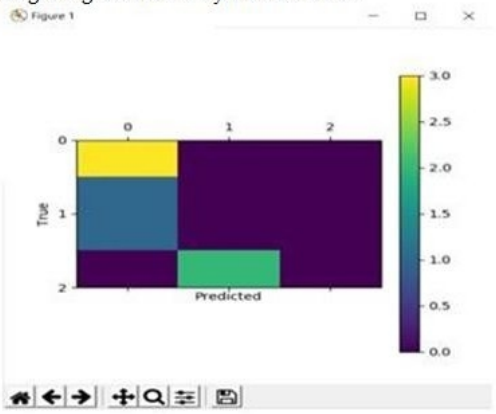


Fully-Connected Layer
The input to the fully connected layer (FC) is that the output from the ultimate Pooling, which is flattened and then fed into the fully connected layer. This layer carries with it the weights and biases together with the class score for every of the classification category. Fully connected layers connect every neuron in one layer to each neuron in another layer.

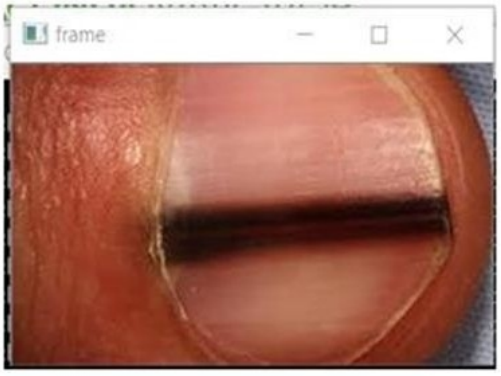
Output Layer
The output layer is chargeable for producing the ultimate result. There should be one output layer in an exceedingly neural network. The output layer takes within the inputs which are passed in from the layers before it, performs the calculations via its neurons and so the output is computed.

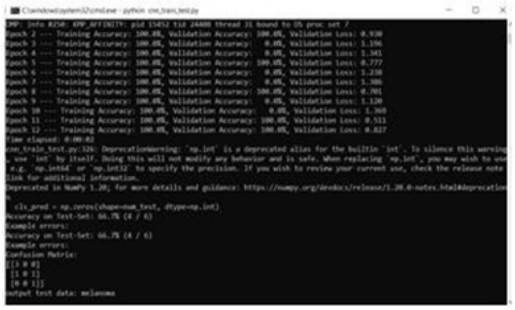


Performance Analysis
The performance of the CNNs trained with the nail dataset was estimated by the classification performance of the models with thenormal, melanoma and onycholysis validation datasets. The performance of fine image selector helps in assessing image quality with the change in the illumination and reduction noise level of the images. The levels of brightness and noise were gradually reduced to classify the image easily. If the validation loss decreases then the accuracy will increase. Number of epochs should be as high as possible and terminate training based on the error rates. epoch is one learning cycle where the learner sees the whole training data set. Here we are having 12 epochs and we are getting 100% validation and training accuracy. In this experiment we found that using color feature of nail image average 65% results are correctly matched with training set data during three tests conducted and we are getting 65% accuracy on test dataset.



IV. EXPERIMENTAL RESULTS





V. CONCLUSION

In the proposed technique we have trained a model that classifies the disease based on the pattern on the nail. This proposed system is able to predict the disease for the respective pattern of the nail with high accuracy. It is able to identify the small patterns also such that providing a system with higher success rate. The limitations of the existing model are eliminated by the proposed model. Moreover in the proposed system only the images of nails of fingers have been used for classifying the diseases, but in future we can combine other features of human body and predict various diseases based on the symptoms of patient and hence would be able to detect a lot of diseases with good precision and accuracy.

VI. REFERENCES

[1]. A. Bourquard, I. Butterworth, A. Sanchez-Ferro, L. Giancardo, L. Soenksen, C. Cerrato, R. Flores, and C. Castro-Gonzalez, "Analysis of white blood cell dynamics in nailfold capillaries," Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS, vol. 2015-Novem, pp. 7470–7473, 2015.

https://www.researchgate.net/publication/290194520_Analysis_of_White_Blood_Cell_Dynamics_in_Nailfold_Capillaries

[2]. A. Bourquard, A. Pablo-Trinidad, I. Butterworth, Á. Sánchez-Ferro, C. Cerrato, K. Humala, M. FabraUrdiola, C. Del Rio, B. Valles, J. M.Tucker-Schwartz, E. S. Lee, B. J. Vakoc, T. P. Padera, M. J. Ledesma-"Non-invasive detection of severe neutropenia in chemotherapy patients by optical imaging of nailfold microcirculation," Scientific Reports, vol. 8, no. 1, pp. 1–12, 2018. Carbayo, Y. B. Chen, E. P. Hochberg, M. L. Gray, and C. Castro-González,

https://doi.org/10.1038/s41598-018-23591-0

[3]. M. EtehadTavakol, A. Fatemi, A. Karbalaie, Z. Emrani, and B.-E.Erlandsson, "Nailfold Capillaros copy in Rheumatic Diseases: Which Parameters Should Be Evaluated?," Bio Med research international, vol. 2015, p. 974530, 2015.

https://www.researchgate.net/publication/281409144_Nailfold_Capillaroscopy_in_Rheumatic_Diseases_Which_Parameters_Should_Be_Evaluated

[4]. M. Cutolo, A. Sulli, M. E. Secchi, S. Paolino, and C. Pizzorni, "Nailfoldca pillaroscopy is useful for the diagnosis and follow-up of autoimmune rheumatic diseases. A future tool for the analysis of micro vascular heart involvement?," Rheumatology, vol. 45, pp. iv43–iv46, oct 2006.

https://doi.org/10.1093/rheumatology/kel310

[5]. A. Karbalaie, M. Etehadtavakol, F. Abtahi, A. Fatemi, Z. Emrani, and B.-E.Erlandsson, "Image enhancement effect on inter and intra-observer reliability of nailfold capillary assessment," Microvascular Research, vol. 120, pp. 100 – 110, 2018.

https://www.researchgate.net/publication/338521756_DA-CapNet_Dual_Attention_Deep_Learning_based_on_U-Net_for_Nailfold_Capillary_Segmentation

[6].A. Karbalaie, Z. Emrani, A. Fatemi, M. Etehadtavakol, and B.-E.Erlandsson, "Practical issues in assessing nail fold capillaroscopic images: a summary," Clinical rheumatology, pp. 1–12.

https://doi.org/10.1007/s10067-019-04716-w

[7]. F. Isgrò, F. Pane, G. Porzio, R. Pennarola, and E. Pennarola, "Segmentation of nailfold capillaries from microscopy video sequences," Proceedings of CBMS 2013 - 26th IEEE International Symposium on Computer-Based Medical Systems, pp. 227–232, 2013.

https://doi.org/10.1109/CBMS.2013.6627793

[8]. J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431–3440, 2015.

https://doi.org/10.1109/CVPR.2015.7298965

[9]. O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convol utional networks for biomedical image segmentation," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2015

https://arxiv.org/abs/1505.04597

[10]. L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille,"Deeplab: Semantic image segmentation with deep convolutional nets, convolution, and fully connected crfs," IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 4, pp. 834–848, 2017.

https://www.researchgate.net/publication/303812083_DeepLa b_Semantic_Image_Segmentation_with_Deep_Convolutional _Nets_Atrous_Convolution_and_Fully_Connected_CRFs

REFERNCES

[1] A. Bourquard, I. Butterworth, A. Sanchez-Ferro, L. Giancardo,L. Soenksen, C. Cerrato, R. Flores, and C. Castro-Gonzalez, " Analysis of white blood cell dyna -mics in nail fold capillaries," Proceedings of the Annual International Confe - rence of the IEEE Engineering In Medicine and Biology Society, EMBS, vol. 2015-Novem, pp. 7470–7473, 2015.

[2] A. Bourquard, A. Pablo-Trinidad, I. Butterworth, Á. Sánchez-Ferro,C. Cerrato, K. Humala, M. Fabra Urdiola, C. Del Rio, B. Valles, J. M.Tucker-Schwartz, E. S. Lee, B. J. Vakoc, T. P. Padera, M. J. Ledesma-"Non-invasive detection of severe neutropenia in chemotherapy patientsby optical imaging of nailfold microcirculation," Scientific Reports, vol. 8, no. 1, pp. 1–12, 2018. Carbayo, Y. B. Chen, E. P. Hochberg, M. L. Gray, and C. Castro-González,

[3] M. Etehad Tavakol, A. Fatemi, A. Karbalaie, Z. Emrani, and B.-E. Erlandsson, "Nailfold Capillaroscopy in Rheumatic Diseases: Which Parameters Should Be Evaluated?," BioMed research international, vol. 2015, p. 974530, 2015.

[4] M. Cutolo, A. Sulli, M. E. Secchi, S. Paolino, and C. Pizzorni, "Nailfold capillaroscopy is useful for the diagnosis and follow-up of autoimmune rheumatic diseases. A future tool for the analysis of microvascular heart involvement?," Rheumatology, vol. 45, pp. iv43–iv46, oct 2006.

[5] O. Wilhelmsson, "Evaluation of video stabilisation algorithms in dynamiccapillaroscopy," 2018.

[6] A. Karbalaie, M. Etehadtavakol, F. Abtahi, A. Fatemi, Z. Emrani, and B.-E. Erlandsson, "Image enhancement effect on inter and intra-observer reliability of nailfold capillary assessment," Microvascular Research, vol. 120, pp. 100 – 110, 2018.

[7] A. Karbalaie, Z. Emrani, A. Fatemi, M. Etehadtavakol, and B.-E. Erlandsson, "Practical issues in assessing nailfold capillaroscopic images: a summary," Clinical rheumatology, pp. 1–12.

[8] F. Isgrò, F. Pane, G. Porzio, R. Pennarola, and E. Pennarola, "Segmentation of nailfold capillaries from microscopy video sequences," Proceedings of CBMS 2013 - 26th IEEE International Symposium on Computer-Based Medical Systems, pp. 227–232, 2013.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431–3440, 2015.

[10] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2015

[11] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in European conference on computer vision, pp. 818–833, Springer, 2014.

[12] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, convolution, and fully connected crfs,” IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 4, pp. 834–848, 2017.