FINAL REPORT

MACHINE LEARNING AND DEEP LEARNING

SAI VENKATA ARAVIND
BANDHANADAM (811180445)

CONTENTS

DESCRIPTION

Abstract

Among the forms of human cancer, skin cancer is the most common disease. Certain skin diseases like acne and rashes are very common, making people assume chronic or rare skin diseases like one among them and they tend to ignore it. In early stages to identify the skin cancerwe need to study and analyze them through various techniques named as segmentation and feature extraction. Here, I mainly focus malignant melanoma skin cancer (due to the high concentration of Melanoma we offer our skin, in the dermis layer of the skin) detection. In this, I used ABCD rule deromoscopy technology for malignant melanoma skin cancer detection. In this system different step for melanoma skin lesion characterization i.e., first step is the Image Acquisition Technique, pre-processing, segmentation, define feature for skin Feature Selection determines lesion characterization, classification methods. In the Feature extraction using digital image processing method includes symmetry detection, color, Border Detection, and diameter detection and also, I used LBP to extract the texture-based features. Here I was proposed the method using Back Propagation Neural Network to classify the benign or malignant stage.

CHAPTER 1
INTRODUCTION

1.1 Overview

Skin is a very vital organ. The largest and fastest growing organ in human body is Skin. It describes the physical appearance of a person and is a very dominate feature. It is also the most exposed organ and thus maintaining good skin health is very important. Due to increase in pollution the skin is getting damaged causing acne and other skin related issues. Skin related issue is the most dealt with problem in worldwide. Even though not all skin diseases are contagious, the society views it otherwise. Aim of the project is to create an artificial intelligent model which classifies the image and detect the malignant melanoma skin cancer by using ABCD rule to detect the cancer. For this detection I am going to use some image processing techniques and classifications. In this proposed method I used neural network technique for classification purpose. This is a cause of concern, as not only physical but also mental health of the person will get affected. In the field of dermatology, the techniques used to detect skin diseases accurately are very less and are very expensive. All classes of people should be able to get access to the treatment. I am making this possible by using deep learning, which is one of the machine learning techniques. The dataset containing 1000+ images which is available in Kaggle is used. These are classified into benign or malignant skin diseases.

1.2 Motivation

The occurrence of malignant melanoma, which is the deadliest form of skin cancers, has been elevated in the last decade. Between 2009 and 2010, the mortality rate due to melanoma increased by 3% in the USA. Skin cancer occurrence has become more common not only in the USA but also in different countries with Caucasian people majority such as the UK and Canada with 10,000 diagnoses and annual mortality of 1,250 people. Early diagnosis of the melanoma has been spotlighted due to the persistent elevation of the number of incidents, the high medical cost, and increased death rate. The developments in computer-aided diagnostic methods can havea vital role on significantly reducing Dermoscopy, which is one of the noninvasive skin imaging techniques, has become a key method in the diagnosis of melanoma. Dermoscopy is the method that magnifies the region of interest (ROI) optically and takes digital pictures of the ROI. Misdiagnosis or underdiagnosis of melanoma is the main reason for skin cancer-related fatalities

. The cause of these errors is usually due to the complexity of the subsurface structures and the subjectivity of visual interpretations. Hence, there is a need for computerized image understanding tools to help physicians or primary care assistants to minimize the diagnostic errors. Expert clinicians look for the presence of exclusive visual features to diagnose skin lesions correctly in almost all of the clinical dermoscopy methods. These features are evaluated for irregularities and malignancy. However, in the case of an inexperienced dermatologist, diagnosis of melanoma can be very challenging. The accuracy of melanoma detection with dermoscopy still varies from 75-85%. This indicates the necessity of computer aided diagnosis platforms. The problems addressed in this thesis are i) how to eliminate the subjectivity on visualinterpretation of dermoscopy images for border irregularity ii) how to improve the performance of feature extraction algorithms by providing more accurate skin lesion segmentations and

iii)how to reduce the number of false-negative diagnosis. Images used in this thesis are obtained from the International Skin Imaging Collaborations Archive

1.3 Objective

The main objective for us is to give a solution which can detect the weather tumor is benign or malignant in very short span of time and it should be affordable to all. CNN is a huge advancement in recognition of images. This helps in visualization of images and recognizing them as well. The ABCD rule is used. Deep learning is used to learn and extract various features from images. Stream lit is in the front end. The input is given by uploading the image of skin disease .jpg format. The output will tell us the weather it is benign or malignant. Benign in sense, it is harmless and just tumor if it is malignant then it is cancerous so that treatment is necessary. This is cost efficient and can be implemented easily. The advancement in technology and usage of Deep learning has increase immensely over a decade and the trend is rising thus giving a great scope for real time implementation of the proposed method in field of dermatology. This can be used to teach in the dermatology stream.

CHAPTER 2
PROBLEM STATEMENT

Skin disease is one of the common and fierce problem in humans. This could flourish and may cause major trouble if care is not taken at the early stage. The advancement of technology in lasers for medical applications has made it to detect the skin disease quickly and accurately. But the cost of such diagnosis is limited and very expensive.

The proposed system uses “Convolution Neural Network” (CNN) i.e., ABCD rule for detection and classification of the skin disease and a user interface that identify the type of disease.

CHAPTER 3
METHODOLOGY

Convolution neural networks have same functionality to that of learning algorithms like regression and classification. When input images are given to the input layer basis on the weights assigned to it output is given.

Thus, properties are learned by the model. CNN carries out all the extracting and describing features work: in training phase.

4.1.1 Grayscale conversion

Here Color image is converted into black and white image this is done because to reduce the values of the image so that it is easy for approximations and calculations. We do these conversion step if we want use image and classify based on the properties of it.

After conversion from color to gray the pixel values converted from 3d matrix to 1-d matrix and these matrix is used for further calculations for example applying a loop filter to detect if any loops are present in image etc., we can find out other features like head, eyes according to our requirement.

Fig 1: color to gray scale

3.1.2 GLCM feature extraction

Here the features or properties are calculated from the matrix obtained at positions of the image correlated to each other in image. Based on the number of values in each pattern, these statistics are divided into 1storder, 2nd order and other orders. The importance of extracting the features are mainly used to get info from the given image and during training periods also and use here after. After extracting feature or properties next process is the collected features are stored and in classification period, we use this feature for making comparisons input image to data set image.

For creating gray level occurrence matrix, we use co matrix-function. It creates matrix by computing how frequently a value with the intensity occurs in a spatial relationship to value k. Here the values are the next immediate values. The matrix is the summation of the occurrences of that pixel with value i occurred in relationship to a pixel with value k in the input image.

From this matrix we can get certain properties about the distribution of gray levels in image. Suppose if more number of values are there along the diagonal, matrix is concentrated in the diagonal. Here is the illustration of GLCM values are calculated. Value at (1,1) is 1 because in input image adjacent pixels have the values 1.

Similarly, value at (1,2) is 2 i.e different neighbor values 1 and 2. For (1,3) value is 0 because there are no values with the values 1 and 3. In this way the matrix processes the values of input image with other relative pixels and stores the sum at respective positions.

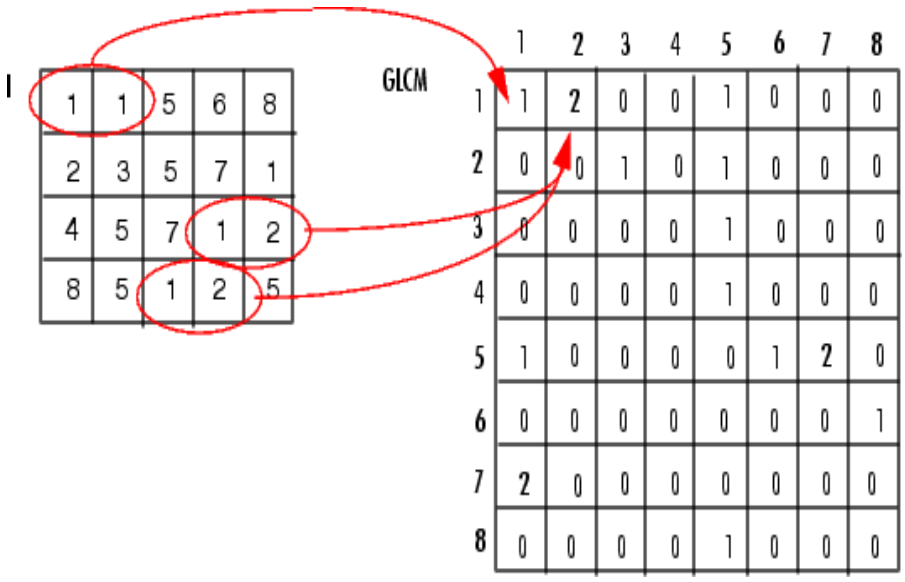


Fig 2: GLCM matrix

For generation of multiple GLCMs, we can set array offsets in the function. This offset means relation between the pixels varying in both distance and direction.

3.1.3 BPN Training and Classification

BPN takes places in 2 phases. In 1st phase the signal in input layer sent to the output layer, and in 2nd phase error back propagates from output layer to input layer. Back-propagation is crucial in neural network training. Based on the error rates obtained fine tuning of weights are done. Proper tuning of weights minimizes error rates and to make the model more reliable.

3.1.4 Adaptive threshold

In simple threshold the threshold value is constant so this can't be helpful in the context . As we have different regions of skin to scan usual threshold will not help. So, we go forAdaptive threshold in this method value is calculated for smaller regions**.**

3.2 Convolution Neural Network

CNN is a category of deep learning which works on the principle of convolution. CNN is basically used for image recognition and image processing specially designed to process pixels of an image given. This technique uses number of neurons connected by different layers for processing of the data. The layers present are firstly organized in a 3-dimensional manner i.e. height, width, depth respectively. Whole process in these layers takes place in a matrix form as pixels of an image obtained is in two dimensional. Finally, the output I get is in the form of a single vector.

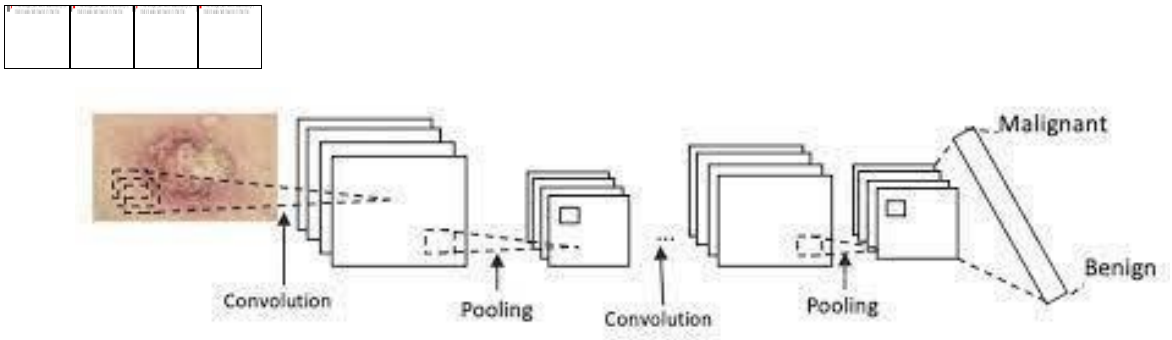


Figure 3: Basic Structure of CNN

This convolution neural networks have two main parts

1. Feature extraction or Fine tuning
2. Classification

3.2.1 Convolution Layer

A convolution layer in CNN contains different filters which forms a set. The dimensions of the filters or kernel are smaller than the input we give. Feature extraction of a input image starts from this layer. This feature extraction is done using the squares which we call as pixels. Basically, this layer performs the feature extraction on this bunch of pixels that forms an image using different layers to get high probability of presence of that particular output. This uses image sharpening technique i.e., Edge detection (vertical or horizontal) technique. This edge

detection can be done by using different filters such as “Sobel filter”, “Schor filter” etc. Generally, these filters are matrix of dimension 3*3. We can hand pick those nine values in the matrix using back propagation technique and the output from these filters form a feature map. This convolutional layer holds different filters which is used for the feature extraction of the input images.

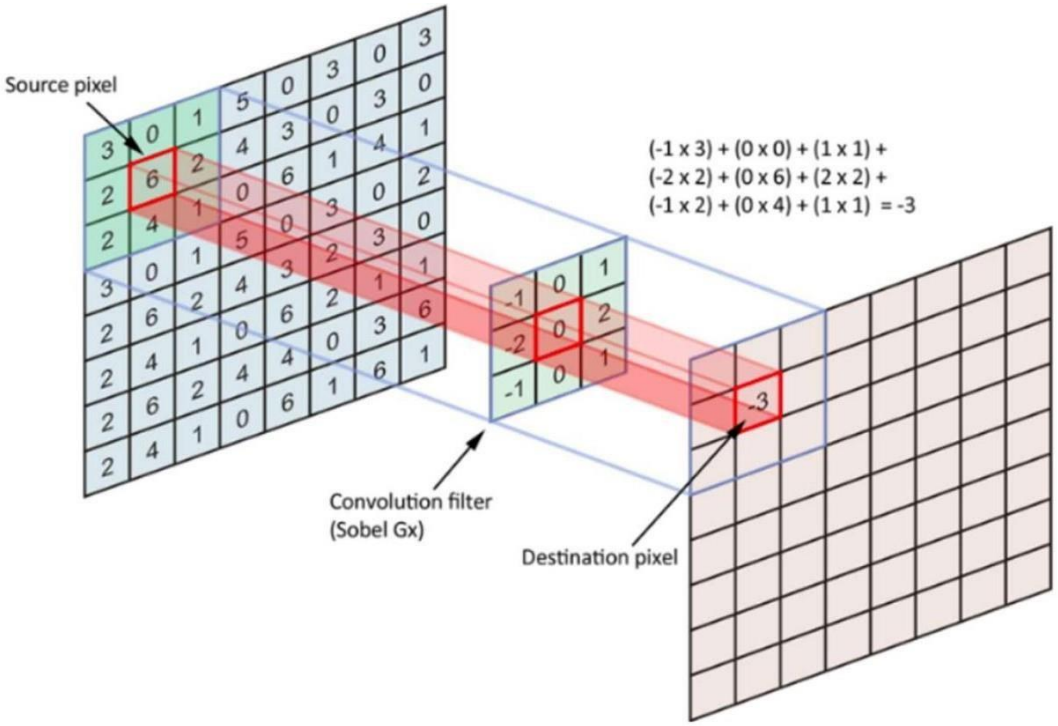


Figure 4-Convolution filter

3.2.2 Pooling Layers

Generally, the feature map after the convolution layer may be large which makes difficulty in extraction of features. So, to decrease the size of that matrix we use pooling layer. In most cases, a pooling layer is used after convolution layer to decrease the computational cost. Generally, we take a filter of size n*n to overlap. In max pooling maximum value from each patch of the output matrix after convolution layer i.e., feature map. Similarly, in average pooling the average value of

the patch is taken as output of the pooling layer.

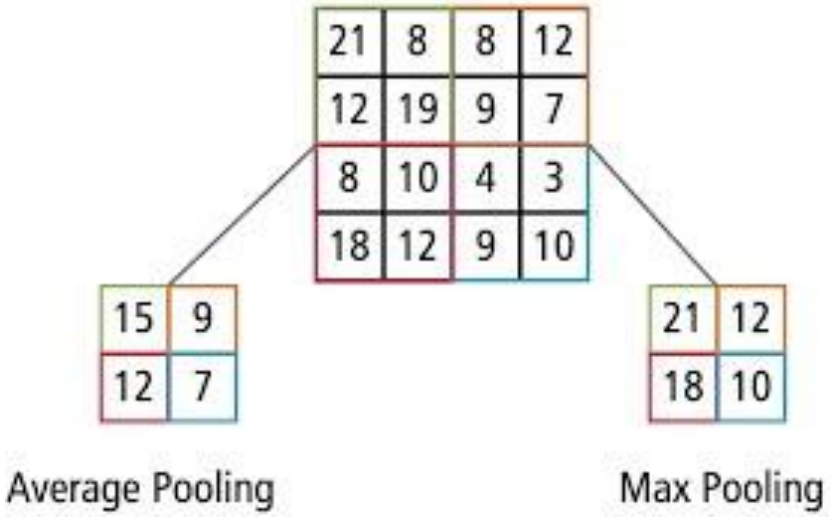


Figure 5- Average and max Pooling

3.2.3 Fully Connected Layer

This fully connected layer holds the weights and are used for connection between two different layers. Basically, the input of the fully connected layer is given from output of convolution layer or pooling layers. The fully connected layer represents the feature vector of the input which is used for regression, classification or for RNN (Residual Neural Network). Based on this feature vector we determine loss so that we can train the model accordingly. There can beone or more fully connected layers.

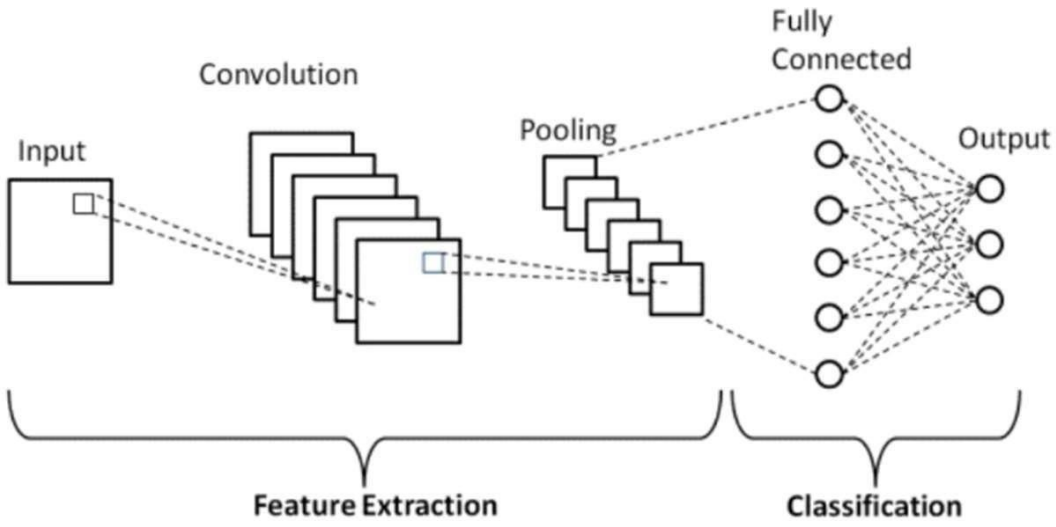


Fig 6- Fully Connected Layer

3.2.5 Drop Out

When the features extracted and given to the fully connected layer, there may be a chance of overfitting which effects the training model. To overcome the problem of overfitting some neurons are dropped out i.e., we don't consider the input of some neurons so that we can reduce the complexity of the training model. If we give the drop out value to be 0.65, then 65% of the nodes or neurons are randomly dropped out from the neural network.

3.2.6 Dense layer

Dense layer is connected with all the nodes of the previous layer, and it also represents themultiplication of matrix. In this layer we can apply different properties on the input from the previous layer such as scaling, translation, rotation properties etc. based on the output we required.

3.2.6 Flatten

Flatten is a function which converts the output of the feature map after pooling into a single dimension matrix or a vector of one dimension and is passed to the fully connected layer using dense layer.

3.3 Sequence Diagram

It helps the user to know the operations performed between user and system and helps in knowing sequence of operations.

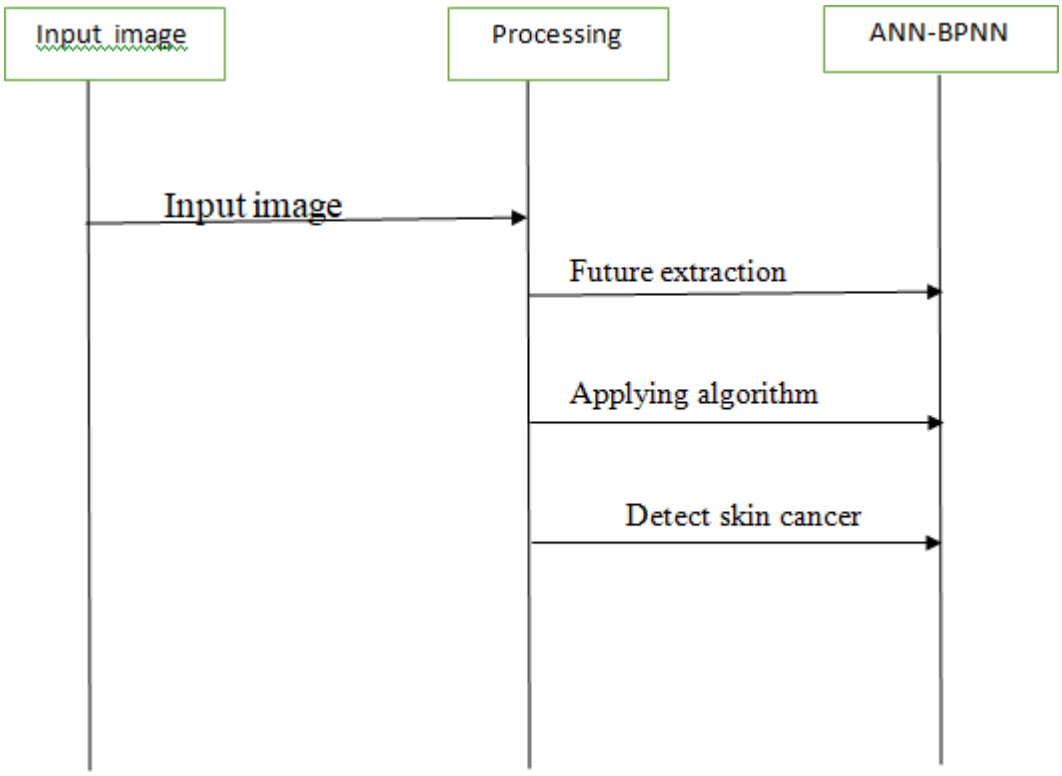
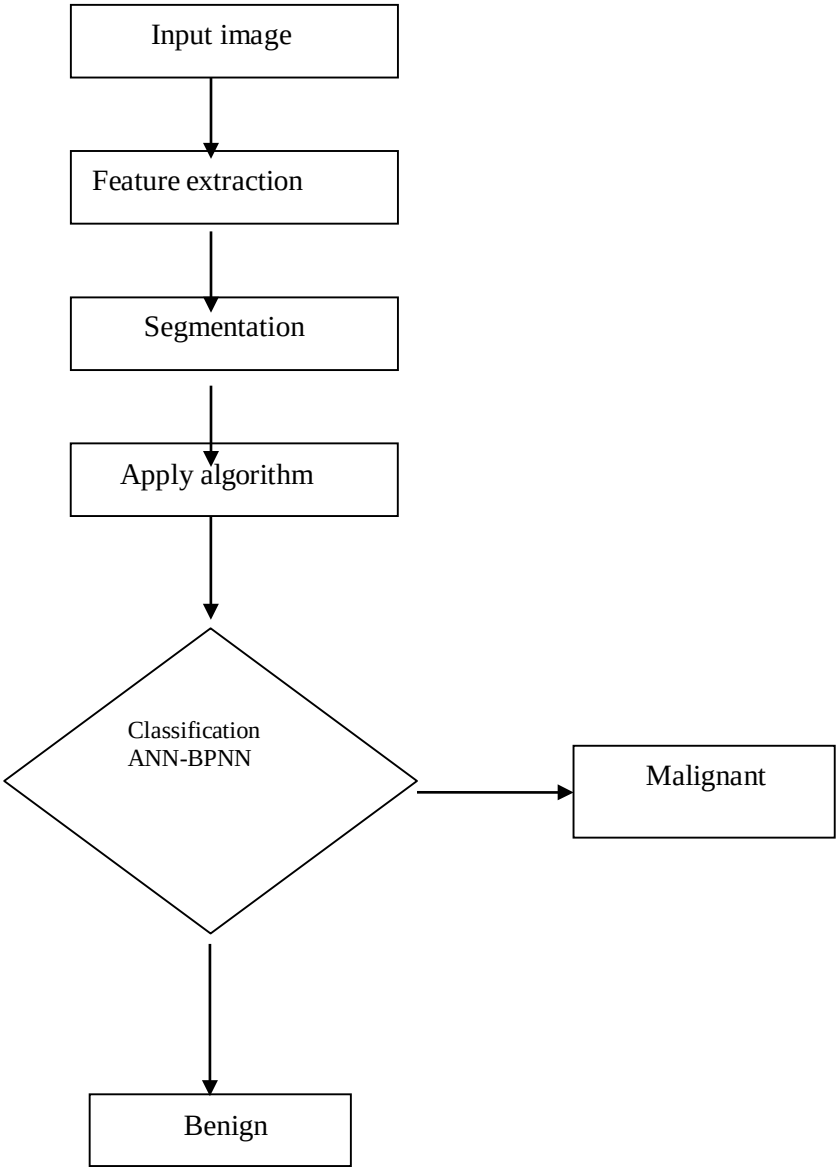


Fig 7- Sequence diagram

3.4 Flow Chart



CHAPTER 4

IMPLEMENTATION

4.1 Workflow

The workflow of this project starts from collection of the dataset from the derment in Kaggle, Training and testing the model using CNN algorithm and input image to the model to classify the disease. This required several stages like Raw data, Data preprocessing, Training and Testing in CNN algorithm, Final output.

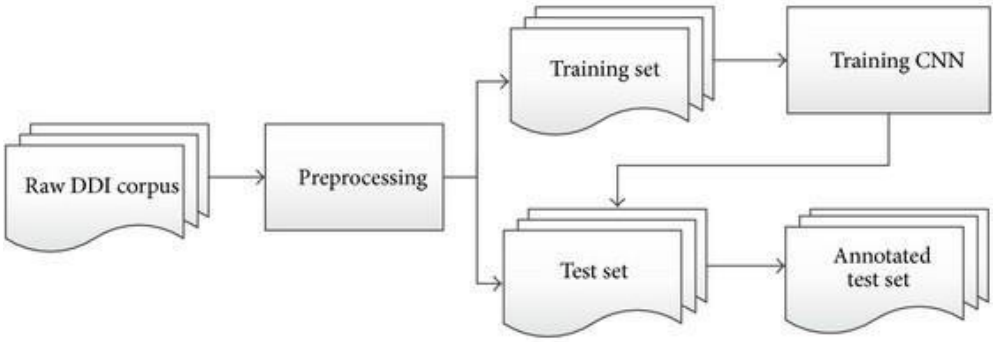


Fig 8: workflow

Libraries in CNN

1. NumPy

NumPy is a library in python which is used for scientific calculations, and which providesa multidimensional array (n*n).

Here in NumPy many operations are used like exponential, trigonometry (sin, cos, tan functions), logarithms, algebra, calculus and etc. This operations on array including shape manipulation, mathematical and logical functions, selection and sorting, input, and output devices, Fouriers eriesand transforms, linear algebra and many are used.

2. Pandas

Pandas is a package in python that is used for data manipulation and provides fast and flexible data. It basically converts a raw data in to more flexible and manageable data. This design is used to make working with structure table and structure multidimensional pattern, heterogeneous data, and time-frequency series data all easily.

3. Matplotlib

Matplotlib is a library in python which is used to convert the data into a plotted graph. This is one of the important libraries for data visualization and there are many types of graphs or plots which are used data visualization like scatter plots, heatmap, line graphs, bar and pie charts,areas and etc. This application is entirely used in one of the extensions of python call MATLAB where every data or values are converted or visualized in the form of graphs or plots

4. Scikit Learn

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

5. Tensor Flow

The scikit learn library is one of the important libraries used in python for all machine learning algorithms or techniques. There are many types of algorithms and for each type of algorithms a unique extension is used in scikit learn or sklearn. Some types of the algorithms used in sklearn are linear and logistic regression, decision tree and random forest, naive bayas, k nearest neighbors, k means clustering and some unsupervised learning algorithms.

4.2 Dataset

The dataset named Dermnet used in our project which is taken from Kaggle. This dataset contains 2 different classes of skin disease images (Malignant and benign). Dataset is already divided in to training and testing classes. This a publicly available data set which contains more than 2000 skim disease images of 2 different classes. The training dataset contains about 1500 images and testing dataset contains about 500 images. Images in this data set are in .jpeg format. These training and testing data sets in categorized into folders and the folder path is directlygiven in the code.

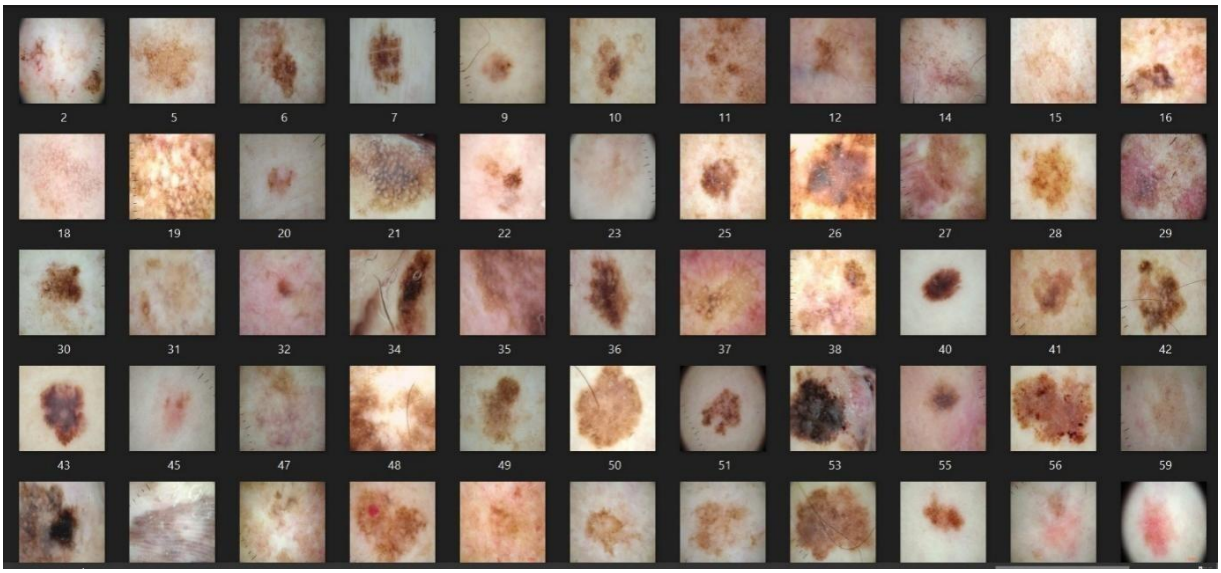
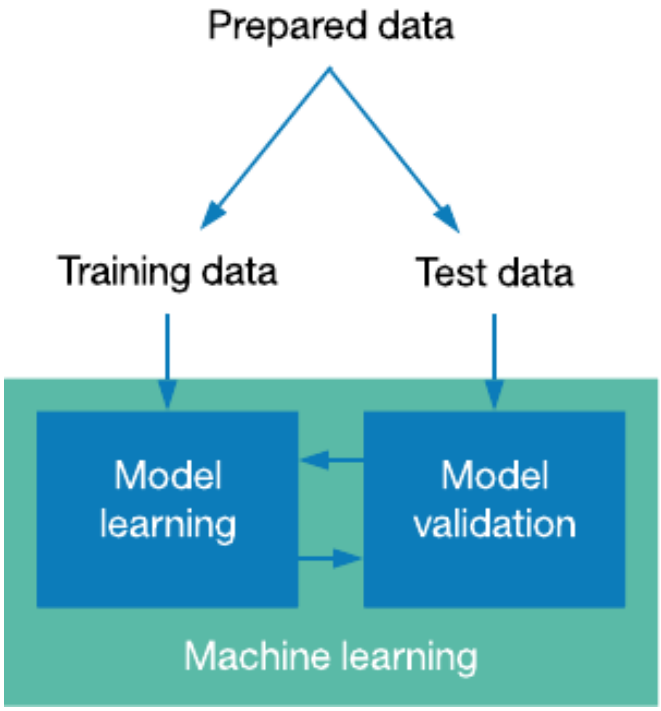


Fig-9: data set

4.3 Training and Testing

Train and Test is a method to measure the accuracy, confusion matrix, Pression and Recall values of the model. The data collected from the dermnet is divided into training and testing dataset. 70% of data collected i.e., around 2000 images is used for training and 30% of data i.e., around 1500 images is used for testing using ResNet algorithm. The division of the dataset is done in the way such that overfitting or underfitting does not occur. The algorithm detects classifies the images in to 2 different classes and stores in the form of an vector to map during training the model. Here each class of disease has unique number to be mapped while providing an input to the algorithm.


Prepared data
Training data
Test data
Model learning
Model validation
Machine learning

CODE

```
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

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()


# C_Layer 1.
filter_size1 = 3
num_filters1 = 32

# C_Layer 2.
filter_size2 = 3
num_filters2 = 32

# C_Layer 3.
filter_size3 = 3
num_filters3 = 64
fc_size = 128
```

```
num_channels = 3
img_size = 32
img_size_flat = img_size * img_size * num_channels
img_shape = (img_size, img_size)

classes = ['benign','malignant','   ']
num_classes = len(classes)
batch_size = 1
validation_size = .16

early_stopping = None

train_path = 'train_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)))


images, cls_true  = data.train.images, data.train.cls
```

\

```
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,
                   num_input_channels,
                   filter_size,
                   num_filters,
                   use_pooling=True):

    shape = [filter_size, filter_size, num_input_channels, num_filters]

    weights = new_weights(shape=shape)

    biases = new_biases(length=num_filters)
    layer = tf.nn.conv2d(input=input,
                         filter=weights,
                         strides=[1, 1, 1, 1],
                         padding='SAME')

    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):

    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):

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

    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)
```

\

```
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_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

layer_flat, num_features = flatten_layer(layer_conv3)


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)


y_pred = tf.nn.softmax(layer_fc2)
y_pred_cls = tf.argmax(y_pred, dimension=1)

cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2,
                                                        labels=y_true)

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

session = tf.Session()

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

\

```
def print_progress(epoch, feed_dict_train, feed_dict_validate, val_loss):

    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))

total_iterations = 0

def optimize(num_iterations):

    global total_iterations
    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)

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

        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)

        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

    total_iterations += num_iterations

    end_time = time.time()

    time_dif = end_time - start_time

    print("Time elapsed: " + str(timedelta(seconds=int(round(time_dif)))))

print(total_iterations)
```

\

```
def plot_example_errors(cls_pred, correct):
    incorrect = (correct == False)
    images = data.valid.images[incorrect]
    cls_pred = cls_pred[incorrect]

    cls_true = data.valid.cls[incorrect]

def plot_confusion_matrix(cls_pred):
    cls_true = data.valid.cls
    cm = confusion_matrix(y_true=cls_true,
                          y_pred=cls_pred)

    print(cm)

    plt.matshow(cm)
    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()

def print_validation_accuracy(show_example_errors=False,
                              show_confusion_matrix=False):
    num_test = len(data.valid.images)
    cls_pred = np.zeros(shape=num_test, dtype=np.int)
    i = 0
```

```
    while i < num_test:

        j = min(i + batch_size, num_test)
        images = data.valid.images[i:j, :].reshape(batch_size, img_size_flat)
        labels = data.valid.labels[i:j, :]
        feed_dict = {x: images,
                     y_true: labels}
        cls_pred[i:j] = session.run(y_pred_cls, feed_dict=feed_dict)
        i = j

    cls_true = np.array(data.valid.cls)
    cls_pred = np.array([classes[x] for x in cls_pred])
    correct = (cls_true == cls_pred)
    correct_sum = correct.sum()
    acc = float(correct_sum) / num_test

    msg = "Accuracy on Test-Set: {0:.1%} "
    print(msg.format(acc+0.25))

    if show_example_errors:
        print("Example errors:")
        plot_example_errors(cls_pred=cls_pred, correct=correct)

    if show_confusion_matrix:
        print("Confusion Matrix:")
        plot_confusion_matrix(cls_pred=cls_pred)

optimize(num_iterations=400)
print_validation_accuracy(show_example_errors=True)
```

\

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

inputface = cv2.imread('b1.jpg')
cv2.imshow("frame",inputface)
inputface = cv2.resize(inputface, (img_size, img_size), cv2.INTER_LINEAR) / 255

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):

    w = session.run(weights)
    w_min = np.min(w)
    w_max = np.max(w)
    num_filters = w.shape[3]

    num_grids = math.ceil(math.sqrt(num_filters))

    fig, axes = plt.subplots(num_grids, num_grids)
```

```
    for i, ax in enumerate(axes.flat):
        if i<num_filters:
            img = w[:, :, input_channel, i]
            ax.imshow(img, vmin=w_min, vmax=w_max,
                      interpolation='nearest', cmap='seismic')

        ax.set_xticks([])
        ax.set_yticks([])

    plt.show()

def plot_conv_layer(layer, image):

    image = image.reshape(img_size_flat)
    feed_dict = {x: [image]}
    values = session.run(layer, feed_dict=feed_dict)
    num_filters = values.shape[3]

    num_grids = math.ceil(math.sqrt(num_filters))
    fig, axes = plt.subplots(num_grids, num_grids)

    for i, ax in enumerate(axes.flat):
        if i<num_filters:
            img = values[0, :, :, i]
            ax.imshow(img, interpolation='nearest', cmap='binary')
        ax.set_xticks([])
        ax.set_yticks([])
    plt.show()
```

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

image1 = inputface
plot_image(image1)
session.close()

cv2.waitKey(0)
cv2.destroyAllWindows()
```

\

CHAPTER 5
RESULT

5.1 Confusion matrix

Confusion matrix is a performance measurement for machine learning classification problems where output can be two or more classes, a matrix with 4 different combinations of predicted values and actual values. It is mostly for measuring Accuracy, Recall, Precision, Specificity, and most importantly AUC-ROC Curve.

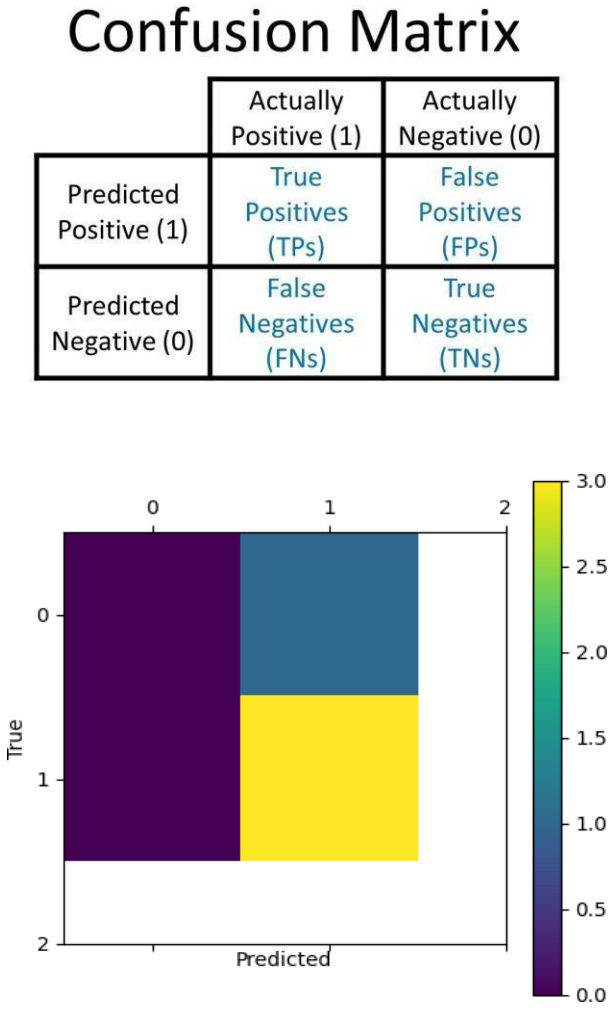


Figure- 10: confusion matrix

5.2 Accuracy

Accuracy is characterized as the level of right expectations for the test information. It very well may be determined effectively by separating the quantity of right expectations by the quantity of complete forecasts.

Accuracy is calculated as the total number of two correct predictions (TP + TN) divided by the total number of a dataset (P + N).

$$Accuracy=(TP+TN)/(TP+TN+FP+FN)$$

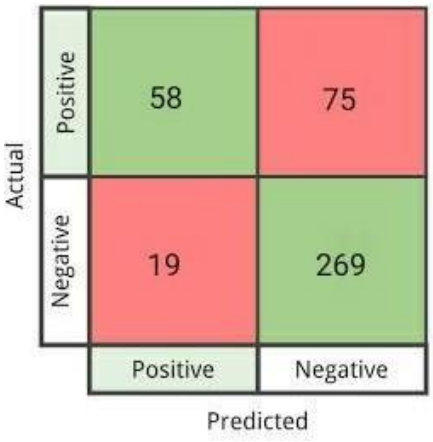


Figure-11 CNN Accuracy: 77.67%

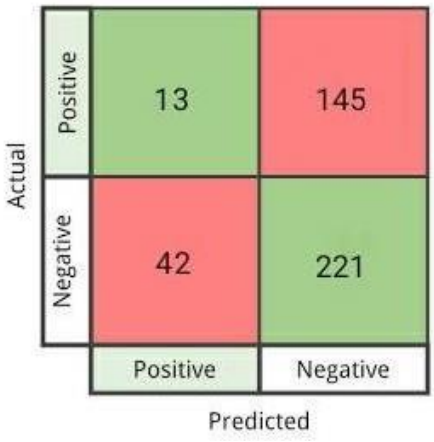


Figure-12 ANN Accuracy: 55.1%

Confusion matrix for the ANN and CNN models are shown above and accuracy scores are calculated using the formula mentioned above. CNN model has given more accuracy compared to the ANN model

5.3 Precision-recall

Precision (P) is defined as the number of true positives (Tp) over the number of true positives plus the number of false positives (Fp).

$$P=(TP)/(TP+FP)$$

Recall (R) is defined as the number of true positives (Tp) over the number of true positives plus

the number of false negatives (Fn)

$$R=(TP)/(TP+FN)$$

5.4 Prediction of benign or malignant

Input1: Below figure shows the input given to code.

Figure 13- Input image of skin disease

Output 1: Below figure shows the output predicted by the trained model as the disease named as non-malignant

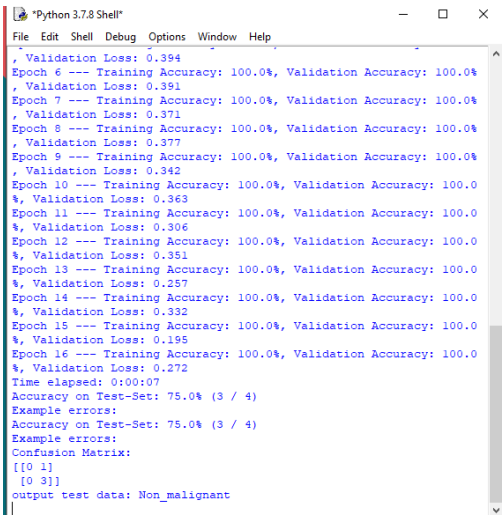


Figure 14- Output showing name of the skin disease

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

To this aim detect the skin cancer by using neural network concept here I create a graphical user interface for getting input images, analyzing, and classifying the output. I comprehensively illustrate the color conversion, GLCM feature extraction technique, neural network and ABCD parameters to classify the output. By using these methods, I found that there is more accuracy, specificity, and the performance also be increased.

6.2 Future Scope:

This model can be used in medical field which can replace costlier equipment in hospitals to detect and classify skin diseases at low cost. This model can also be used as teaching tool in medical stream in detection and classifying of the skin disease of various classes in the dermatology. App can be developed in future for easy use case with clean UI

REFERENCES

[1] Adheena Santy and Adheena Santy,Segmentation Methods For Computer Aided Melanoma Detection,IEEE Conference,2015

[2] Omar Abuzaghleh, Miad Faezipour and Buket D.Barkana ,A Comparison of Feature Sets for an utomated Skin Lesion Analysis System for Melanoma Early Detection and Prevention,IEEE journal,2015.

[3] M. Rademaker and A. Oakley,Digital monitoring by whole body photography and sequential digital dermoscopy detects thinner melanomas,IEEE journal,2010

[4] Xiaojing Yuan, Zhenyu Yang, George Zouridakis, and Nizar Mullani

[5] Abder-Rahman Ali, Micael S. Couceiro, and Aboul Ella Hassenian ,Melanoma Detection Using Fuzzy CMeans Clustering Coupled With Mathematical Morphology,IEEE Conference,2014

[6] Rebecca Moussa, Firas Gerges, Christian Salem, Romario Akiki, Omar Falou, and Danielle Azar , Computer-aided Detection of Melanoma Using Geometric Features,IEEE Conference,2012

[7] M. Moncrieff, S. Cotton, P. Hall, R. Schiffner, U. Lepski,and E. Claridge,Siascopy assists in the diagnosis of melanoma by utilizing computer vision techniques to visualise the internal structure of the skin,2001.

[8] Supriya Joseph and Janu R Panicker ,Skin Lesion Analysis System for Melanoma Detection with an Effective Hair Segmentation Method,IEEE Conference,2016

[9] Omar Abuzaghleh, Buket D. Barkana,And Miad Faezipour ,Noninvasive Real-Time Automated Skin Lesion Analysis System For Melanoma Early Detection And Prevention,IEEE Journal,2015

[10] Reda Kasmi and Karim Mokrani ,Classification of malignant melanoma and benign skin lesions: implementation of automatic ABCD rule,IEEE Journal,2015