ABSTRACT

Skin cancer has been gradually increasing among the people for decades.Every year the number of new cases of melanoma,and ,the death rate is increasing.In 2020 325,00 cases of melanoma were found among which 57,000 people died.Classifying these melanoma from images is a difficult task because of their minute variation from non cancerous skin lesions.In this project we are going to build a computer-aided diagnosis system for classifying skin lesions.In the preprocessing steps hair and other artifacts from the image are removed using morphological filtering.And to extract the foreground or lesion region from the image we used grab-cut segmentation.Melanoma and benign lesions are detected according to the image processing method using CNN's.Pre trained CNN's like Resnet50,MobileNet,VGG16 were used in this project for the image classification task.And also to improve the efficiency Data Augmentation techniques like Rotation are also employed.

KEYWORDS: Computer Aided Diagnosis,HSV,Melanoma.

Table of Contents

CHAPTER-1

SUMMARY OF BASE PAPER

1.1 Introduction

Title	Automated deep learning approach for classification of malignant melanoma and benign skin lesions
Journal Name	Multimedia Tools and Applications
Publisher	Springer
Year	2022
Indexed in	Science Citation Index Expanded

Table 1.1 Base Paper Info

● In recent decades, the incidence of skin cancer has increased, making it a major health problem worldwide. Although the dermatological diagnosis of skin cancer is very effective, it is very difficult for an experienced dermatologist to accurately classify malignant melanoma and benign skin lesions from many dermatological images. Therefore, the development of a non-invasive CAD (Computer Aided Diagnostic) system for the classification of skin lesions is very important. Image preprocessing, segmentation, feature extraction and classification are the four major steps in CAD. The consecrated neural network (CNN) and the deep learning methods have increased due to the diagnosis of cancer. There are a series of data, such as data sets and medical images transmission. This is a useful tool. Simple adjustments for architectures and use require complete data and expensive calculations compared to CNN training to increase operational speed. Initially, the process called "data expansion data " by creating new data in the original data can increase the amount of income data. The diagnostic system is a result of a reasonable classification to discriminate malignant damage to cancer to cope with skin damage diagnostic systems.

Literature Survey

S.NO	TITLE	YEAR	TECHNIQUE
1	Skin Lesion Classification Using Convolutional Neural Network With Novel Regularizer	2019	A novel regularizer technique to classify skin lesions into benign or malignant
2	Classification of skin lesions using transfer learning and augmentation with Alex-net. PLoS One	2019	Using the theory of transfer learning and the pre-trained deep neural network
3	Classification of malignant melanoma and benign skin lesions: implementation of automatic ABCD rule	2016	The ABCD rule of Dermoscopy is a scoring method used by dermatologists to quantify dermoscopy findings
4	Data augmentation for skin lesion analysis	2018	investigated the impact Of data augmentation scenarios for melanoma classification trained on three CNNs (Inception-v4, ResNet, and DenseNet)
5.	Automatic detection of malignant melanoma using macroscopic image	2014	New methods to weaken the effect of nonuniform illumination, correction of the effect of thick hairs and large glows on the lesion and also, a new threshold-based\ segmentation algorithm

1.2 Novelty

- The dataset was having less malicious images that caused the imbalance. I fixed this and removed the imbalance.
- The quality of the image deteriorated after the image was preprocessed according to the paper. So I Enhanced using techniques like Gaussian Blur.
- Stopping early helped reduce computation time by building a CNN model using the Model_checkpoint method.

1.3 Proposed Work

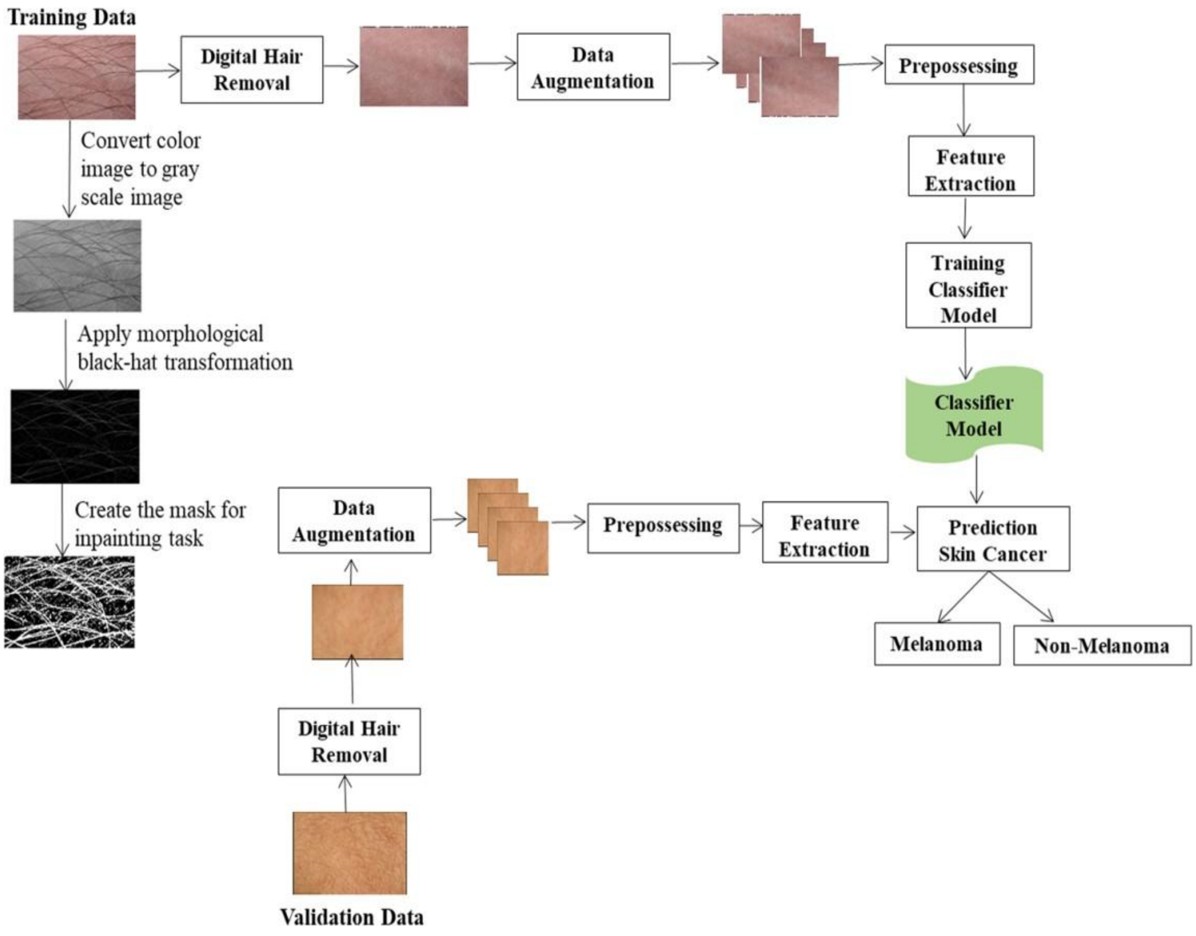


Fig.1.1

1.4 Acquired Dataset

The dataset contains 930 dermoscopic skin images , including Benign skin tumors(nevus, seborrheic keratosis) and Malignant Melanoma.

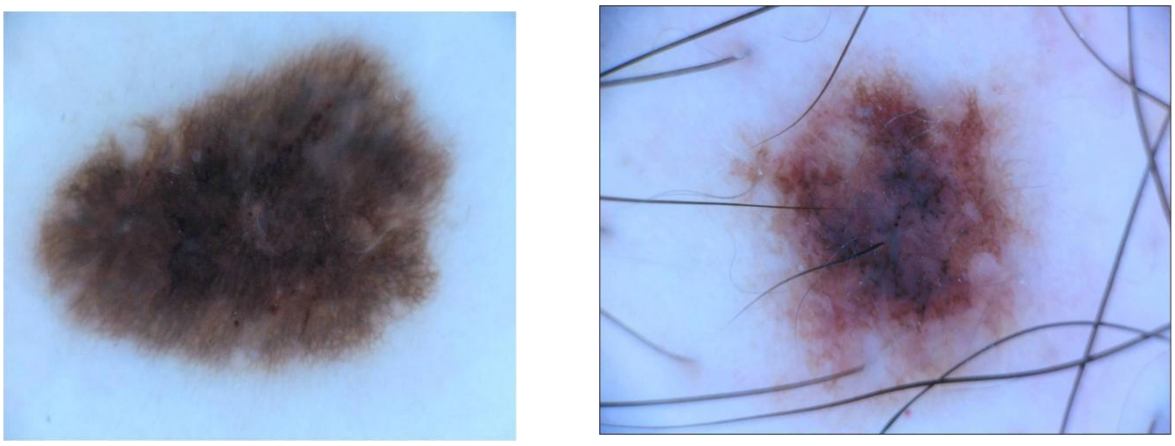


Fig.1.2

we acquired our images from ISIC 2017 database and then to scale up this dataset we had used data augmentation techniques like image rotation in which a single image is roasted by 4 angles(0,90,180,270) thus we finally get 930*4 im

CHAPTER-2
MERITS AND DEMERITS

2.1 Merits

1.UtilizinghybridCNN(VGG16andResNet50withSVMclassifier)modelshadhelped to increase the precision in classification.
2.This learning models have been shown to be highly accurate in a variety of image classification tasks, when we have modified with pre-trained models. This systems can use these patterns to provide reliable and accurate classification results, helping to detect potential cases of skin cancer early. Early detection of disease improves the patient's condition with good treatment.

2.2 Demerits

1.Automated systems may not incorporate human intuition, judgment and domain knowledge, which can be valuable in some cases. Because it is limited to the patterns and rules found in the training data, it may not be able to account for contextual or subtle factors.
2.Deep learning models for training are often based on large, diverse and well-defined datasets. These datasets are difficult and time-consuming to acquire, especially for medical imaging. Before using an automated system in a clinical setting, it must be thoroughly tested to prove its ability to work.

CHAPTER -3

Source Code and Snapshots

3.1 Code For Hair Removal:-

```
import cv2

import numpy as np import os
from google.colab import drive

inDirectory = '/content/drive/MyDrive/Benign' outDirectory =
'/content/drive/MyDrive/outBenign' kernel_size = 10
for filename1 in os.listdir(inDirectory):

img = cv2.imread(os.path.join(inDirectory, filename1)) gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY) gray = cv2.addWeighted(gray, 0.3, gray, 0.59, 0)
gray = cv2.addWeighted(gray, 1, gray, 0.11, 0)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (kernel_size, kernel_size))
blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, kernel)
_, mask = cv2.threshold(blackhat, 10, 255, cv2.THRESH_BINARY) mask =
cv2.inpaint(img, mask, 3, cv2.INPAINT_TELEA) cv2.imwrite(os.path.join(outDirectory,
filename1), mask)
```

OUTPUT OF HAIR REMOVAL

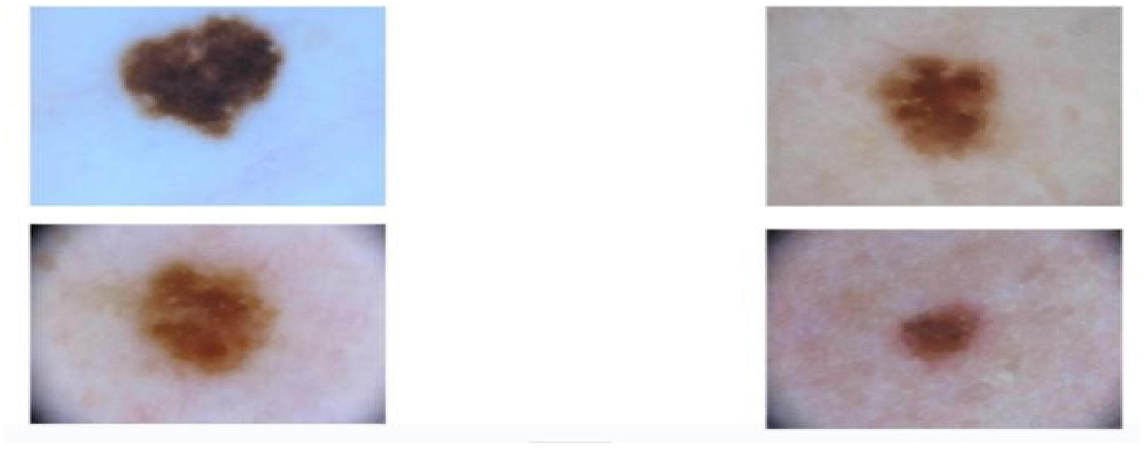


Fig.3.1

3.2 Code For Image Sharpening:-

```
import cv2 import os
inDirectory = '/content/drive/MyDrive/outBenign' outDirectory =
'/content/drive/MyDrive/blurremovedBenign' kernel_size = (5,5)
unsharp_weight = 1.5
blur_weight = -0.5
for filename1 in os.listdir(inDirectory):
img = cv2.imread(os.path.join(inDirectory, filename1)) blur = cv2.GaussianBlur(img,
kernel_size, 0)
unsharp_mask = cv2.addWeighted(img, unsharp_weight, blur, blur_weight, 0)
cv2.imwrite(os.path.join(outDirectory, filename1), unsharp_mask)
```

OUTPUT OF IMAGE SHARPENING:

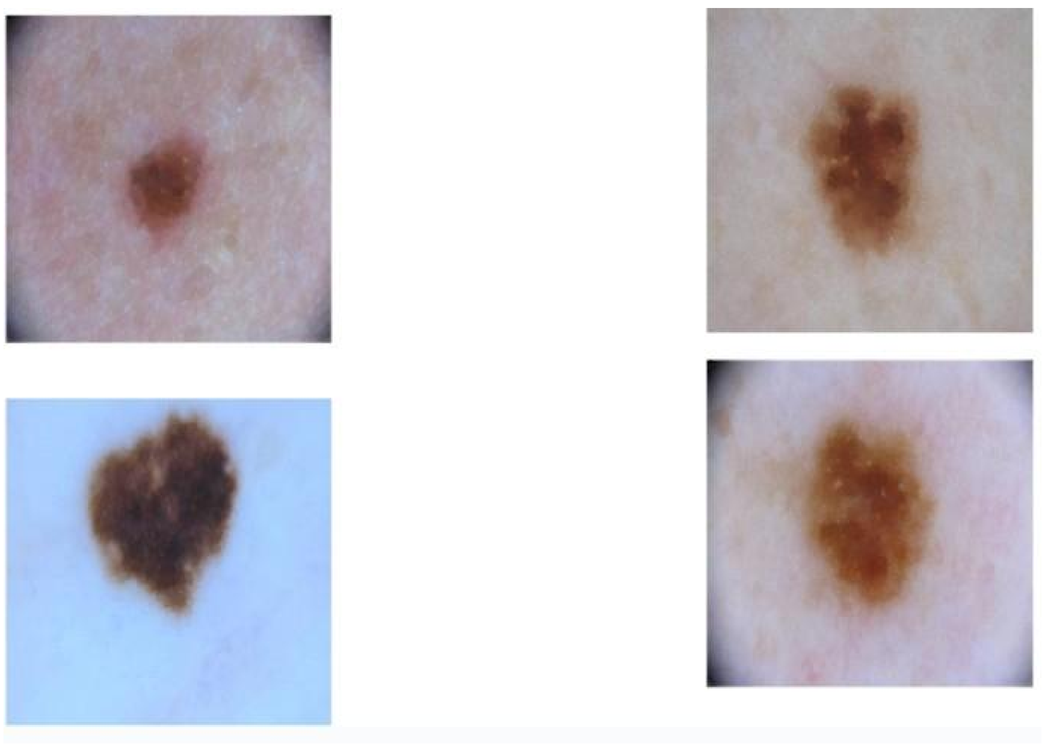


Fig.3.2

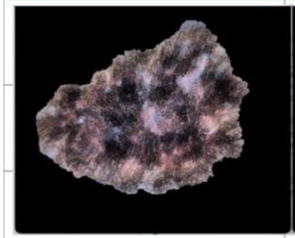
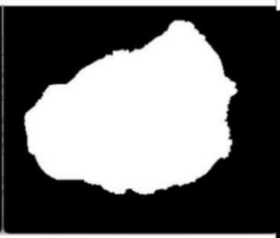
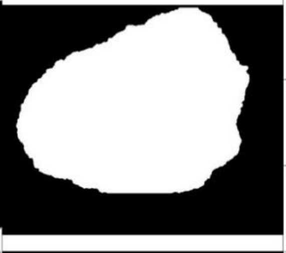
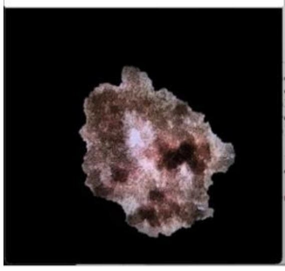
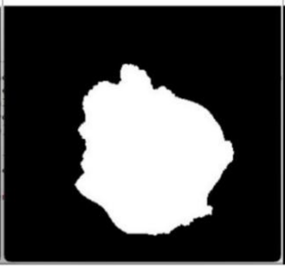
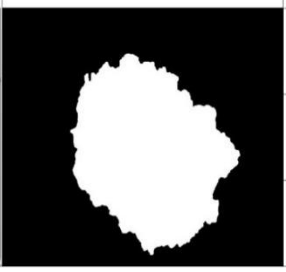
3.3 Code For Image Segmentation:-

```
import cv2 as c import os
import numpy as np

from google.colab.patches import cv2_imshow image = c.imread('ISIC_0000020.JPG')
```

```
hsv = c.cvtColor(image, cv2.COLOR_BGR2HSV)
clahe = c.createCLAHE(clipLimit=2.0, tileGridSize=(8,8)) hsv[:,:,2] =
clahe.apply(hsv[:,:,2])
bgr = c.cvtColor(hsv, cv2.COLOR_HSV2BGR) image_mask = np.zeros(image.shape[:2],
np.uint8) background = np.zeros((1,65), np.float64) foreground = np.zeros((1,65),
np.float64)
rect = (10, 12,250,270)

c.grabCut(bgr, image_mask, rect, background,foreground,30, c.GC_INIT_WITH_RECT)
mask2 = np.where((image_mask==2)|(image_mask==0), 0, 1).astype('uint8') segmented =
bgr*mask2[:,:,np.newaxis]
mask2[mask2 == 1] = 255

mask2[mask2 == 0] = 0

ground_truth = c.imread('ISIC_0000020_segmentation.png', c.IMREAD_GRAYSCALE)
ground_truth = c.resize(ground_truth, (300, 300))
intersection = np.logical_and(ground_truth, mask2) union = np.logical_or(ground_truth,
mask2)
jc = np.sum(intersection) / np.sum(union) c_imshow(segmented) c_imshow(mask2)
print('Jaccard Coefficient:', jc)
c.waitKey(0) c.destroyAllWindows()
```

OUTPUT OF IMAGE SEGMENTATION:-

Based on this Jaccard coefficient value we can evaluate the correctness of our segmented images.Segmented images are compared with ground truth images which were present along with dataset and jaccard coefficient value is calculated for them.

Grab-Cut	Mask	Ground-Truth Image	Jaccard Coefficient Value
			0.855
			0.814

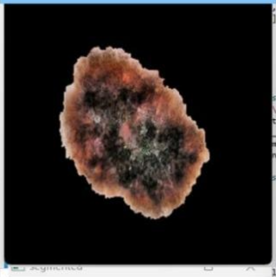
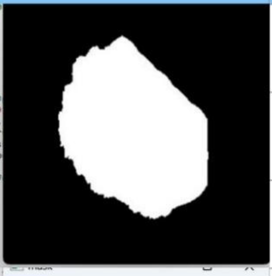
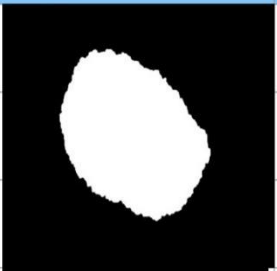
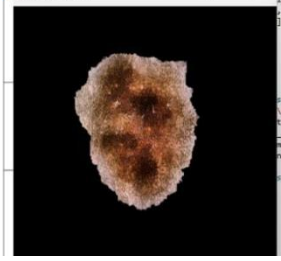
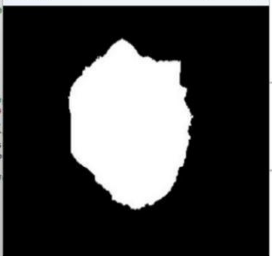
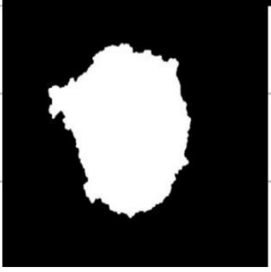
Grab-Cut	Mask	Ground-Truth Image	Jaccard Coefficient Value
			0.904
			0.881

Fig 3.3

3.4 Code For Image Augmentation:-

```
from PIL import Image import os
inDirectory = "/content/drive/MyDrive/blurremovedBenign" outDirectory =
"/content/drive/MyDrive/augmentedBenign" angles = [0, 90, 180, 270]
for filename1 in os.listdir(inDirectory):

lesion = Image.open(os.path.join(inDirectory, filename1)) for angle in angles:
rotated_lesion = lesion.rotate(angle)

new_filename = filename[:-4] + "_" + str(angle) + ".jpg"
rotated_lesion.save(os.path.join(outDirectory, new_filename1))
```

OUTPUT OF AUGMENTATION:

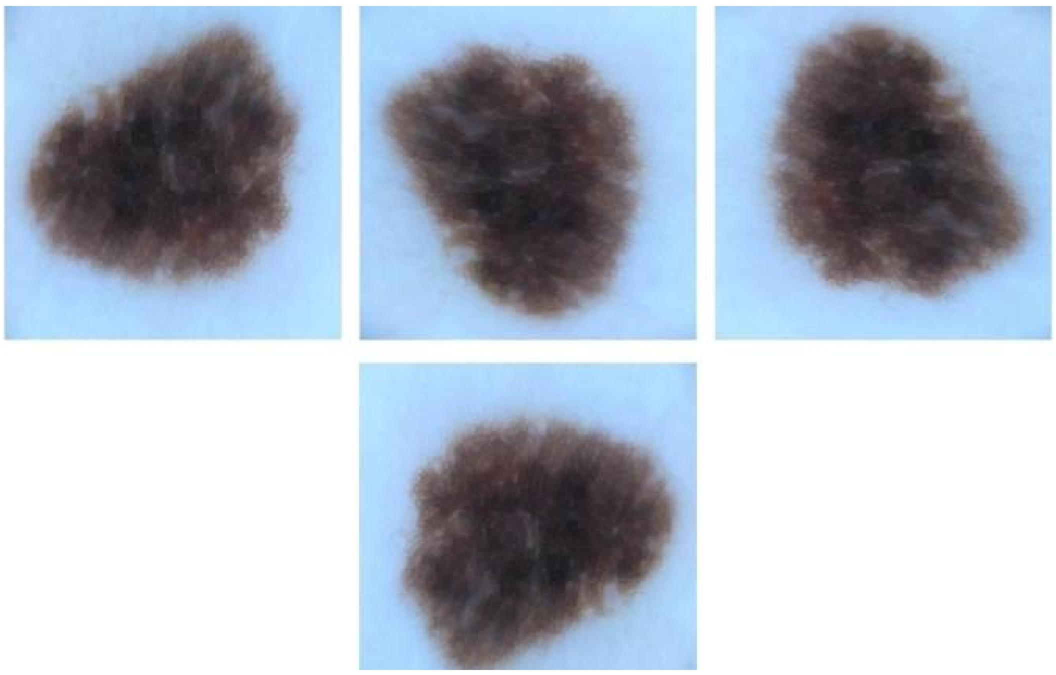


Fig.3.4

3.5 Code For ABCD Feature Extraction:-

3.5.1Asymmetry:

```
import cv2
import numpy as np
import glob

a=[]
b=[]
c=[]
d=[]
ABCD = []
for fil in glob.glob('/content/drive/MyDrive/sample/*.JPG'):
    # Load the image
    img1 = cv2.imread(fil)
    img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

    # Calculate the center of gravity
    M = cv2.moments(img)
    cg_x = int(M['m10']/M['m00'])
    cg_y = int(M['m01']/M['m00'])

    # Draw two orthogonal axes
    cv2.line(img, (cg_x - 100, cg_y), (cg_x + 100, cg_y), (0, 0, 255), 2)
    cv2.line(img, (cg_x, cg_y - 100), (cg_x, cg_y + 100), (0, 0, 255), 2)

    # Divide the lesion into four quadrants
    quadrants = [
        img[0:cg_y, 0:cg_x],
        img[0:cg_y, cg_x:],
        img[cg_y:, 0:cg_x],
        img[cg_y:, cg_x:]
    ]

    # Calculate the mean and standard deviation for each quadrant
    mean_bright = []
    mean_color = []
    std_bright = []
    std_color = []
```

```
  for quadrant in quadrants:
      if quadrant.ndim == 2:
          # Quadrant is grayscale, with only one channel
          mean_bright.append(np.mean(quadrant))
          mean_color.append(0)
          std_bright.append(np.std(quadrant))
          std_color.append(0)
      else:
          # Quadrant is color, with three channels
          mean_bright.append(np.mean(quadrant[:,:,0]))
          mean_color.append(np.mean(quadrant[:,:,1:], axis=(0,1)))
          std_bright.append(np.std(quadrant[:,:,0]))
          std_color.append(np.std(quadrant[:,:,1:], axis=(0,1)))
```

```
  # Calculate the asymmetry score for each axis
  axis1_score = abs(mean_bright[0]-mean_bright[2])/max(std_bright[0], std_bright[2])
  axis2_score = abs(mean_bright[1]-mean_bright[3])/max(std_bright[1], std_bright[3])
  if(axis1_score>0.2 and axis2_score >0.2):
    asymmetry_score=2;
  elif(axis1_score>0.2 or axis2_score>0.2):
    asymmetry_score=1;
  else:
    asymmetry_score=0;
  print("Asymmetry score:", asymmetry_score)
  a.append(asymmetry_score)
```

3.5.2 Border:

```
----------------------------------------BORDER---------------------------------------------

for fil in glob.glob('/content/drive/MyDrive/output_directory3/*.png'):
  # Load the image
  img = cv2.imread(fil)
```

```
# Divide the image into eight equal parts (slices)
slices = []
for i in range(8):
  y1 = i * img.shape[0] // 8
  y2 = (i+1) * img.shape[0] // 8
  slice = img[y1:y2, :]
  slices.append(slice)
```

```
# Initialize the border score
border_score = 0
```

```
# Loop over each slice and calculate the border score
for slice in slices:
  # Convert the slice to grayscale
  gray = cv2.cvtColor(slice, cv2.COLOR_BGR2GRAY)
```

```
  # Apply a Canny edge detector to the slice
  edges = cv2.Canny(gray, 100, 200)
```

```
  # Calculate the number of contours in the slice
  contours, _ = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
  num_contours = len(contours)
```

```
  # Calculate the border score for this slice
  slice_border_score = 1 if num_contours > 1 else 0
```

```
  # Add the slice border score to the overall border score
  border_score += slice_border_score
```

```
# Print the overall border score
print('Border score:', border_score)
b.append(border_score)
```

3.5.3 Color:

```
import cv2
import numpy as np
import glob

# Define the suspicious colors
suspicious_colors = {
    "white": (255, 255, 255),

    "red": (0, 0, 255),
    "black": (0, 0, 0),
    "light_brown": (165, 123, 63),
    "dark_brown": (60, 20, 20),
    "blue_gray": (100, 149, 237)
}

# Load all the images in the directory
for fil in glob.glob('/content/drive/MyDrive/sample/*.JPG'):
    # Load the image
    img = cv2.imread(fil)

    # Convert the image to HSV color space
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Initialize the color score to 0
    color_score = 0

    # Calculate the total number of pixels in the image
    num_pixels = img.shape[0] * img.shape[1]

    # Check for each suspicious color
    for color_name, color_value in suspicious_colors.items():
        # Create a mask for the color
        lower_range = np.array([color_value[0]-10, 100, 100])
        upper_range = np.array([color_value[0]+10, 255, 255])
        mask = cv2.inRange(hsv_img, lower_range, upper_range)

        # Calculate the number of pixels of the color
        num_color_pixels = cv2.countNonZero(mask)

        # Check if the color is present in the image
        if num_color_pixels >= num_pixels * 0.05:
            color_score += 1

    print("Color score:", color_score)
      c.append(color_score)
```

3.5.4 Dermoscopic Structures:

```
------------------------------------------DERMOSCOPIC STRUCTURES--------------------
for fil in glob.glob('/content/drive/MyDrive/output_directory3/*.png'):
   img = cv2.imread(fil)
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   blur = cv2.GaussianBlur(gray, (5, 5), 0)
   ret, thresh = cv2.threshold(blur, 0, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
   contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

   network = False
   structureless = False
   branched_streaks = False
   dots = False
   globules = False

   for contour in contours:
      area = cv2.contourArea(contour)
      if area < 20:
         continue
      x, y, w, h = cv2.boundingRect(contour)
      aspect_ratio = float(w) / h

      if aspect_ratio > 5:
         network = True
      elif aspect_ratio < 1.5:
         if area > 50:
            structureless = True
         elif area > 20:
            dots = True
      elif aspect_ratio < 2.5:
         if area > 70:
            branched_streaks = True
      elif aspect_ratio < 5:
         if area > 70:
            globules = True

   score = 0
   if network:
      score += 1
   if structureless:
      score += 1
   if branched_streaks:
      score += 1
   if dots:
      score += 1
   if globules:
      score += 1
```

```
    print(f"Dermoscopic score: {score}")
    d.append(score)
```

```
import os
output_dir1 = '/content/drive/MyDrive/total1/benign'
output_dir2 = '/content/drive/MyDrive/total1/Melanoma'
if not os.path.exists(output_dir1):
 os.makedirs(output_dir1)
if not os.path.exists(output_dir2):
 os.makedirs(output_dir2)

input_dir = '/content/drive/MyDrive/output_directory3'
```

```
i=0;
for filename in os.listdir(input_dir):
   img = cv2.imread(os.path.join(input_dir, filename))
   ABCD[i]=float(a[i])*1.3+float(b[i])*0.1+float(c[i])*0.5+float(d[i])*0.5
   print(ABCD[i])
   if(ABCD[i]>4.45):
     cv2.imwrite(os.path.join(output_dir1, filename), img)
   else:
     cv2.imwrite(os.path.join(output_dir2, filename), img)
   i=i+1
```

3.6 Code For MobileNet Model:-

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet import ResNet101
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from google.colab import drive
drive.mount('/content/gdrive')
# Load the dataset
data_dir = '/content/gdrive/MyDrive/total1'
batch_size = 16
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)
val_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)

# Define the model
base_model = MobileNet(include_top=False, input_shape=(224, 224, 3), pooling='avg')
x = Dense(1, activation='sigmoid')(base_model.output)
model = Model(inputs=base_model.input, outputs=x)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

```
# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator,
    verbose=1,
    validation_steps=len(val_generator),
    steps_per_epoch=len(train_generator)
)

# Evaluate the model
test_loss, test_acc = model.evaluate(val_generator, verbose=1)
print('Test accuracy:', test_acc)
```

3.7 Code For VGG16 Model:-

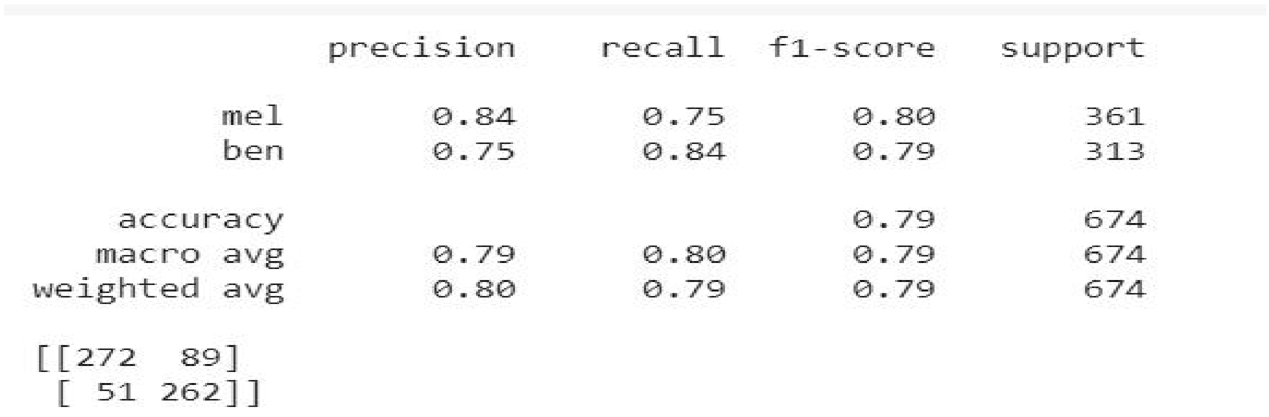
```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import cv2
import pandas as pd
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.vgg16 import VGG16
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from keras import models
from keras.optimizers import Adam
from keras.optimizers import SGD
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
drive.mount('/content/gdrive')

# Load the image data and labels
images = np.load('/content/gdrive/MyDrive/images.npy')
labels = np.load('/content/gdrive/MyDrive/labels.npy')

# Set the image dimensions and batch size
img_height, img_width = 224, 224
batch_size = 16
```

```
# Split the data into training, validation, and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(images, labels, test_size=0.2,
random_state=42)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.2,
random_state=42)
# Define the VGG16 model and add additional layers
vgg_model = Sequential()
```

```
pretrained_model = VGG16(include_top=False,
                   input_shape=(224, 224, 3),
                   pooling='avg',classes=2,
                   weights='imagenet')
# Unfreeze the layers in the pre-trained model
for layer in pretrained_model.layers[:-10]:
    layer.trainable = False
vgg_model.add(pretrained_model)
vgg_model.add(layers.Flatten())
vgg_model.add(layers.Dense(1024, activation='relu'))
vgg_model.add(layers.Dropout(0.5))
vgg_model.add(layers.Dense(1024, activation='relu'))
vgg_model.add(layers.Dropout(0.5))
vgg_model.add(layers.Dense(1, kernel_regularizer=l2(0.01), activation='linear'))
# Compile the model
opt = SGD(learning_rate=0.0001,momentum=0.9)
vgg_model.compile(optimizer=opt, loss='hinge', metrics=['accuracy'])
```

```
# Set the path to save the model in your Google Drive
model_path = '/content/gdrive/MyDrive/Models/vggfinetune.h5'
os.makedirs(os.path.dirname(model_path), exist_ok=True)
model_checkpoint = ModelCheckpoint(model_path, monitor='val_accuracy',
save_best_only=True)
# Train the model with data augmentation
history=vgg_model.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test,
Y_test), epochs=100, callbacks=[model_checkpoint])
```

RESULTS FOR VGG16 MODEL:-



```
              precision    recall  f1-score   support

         mel       0.84      0.75      0.80       361
         ben       0.75      0.84      0.79       313

    accuracy                           0.79       674
   macro avg       0.79      0.80      0.79       674
weighted avg       0.80      0.79      0.79       674

[[272  89]
 [ 51 262]]
```

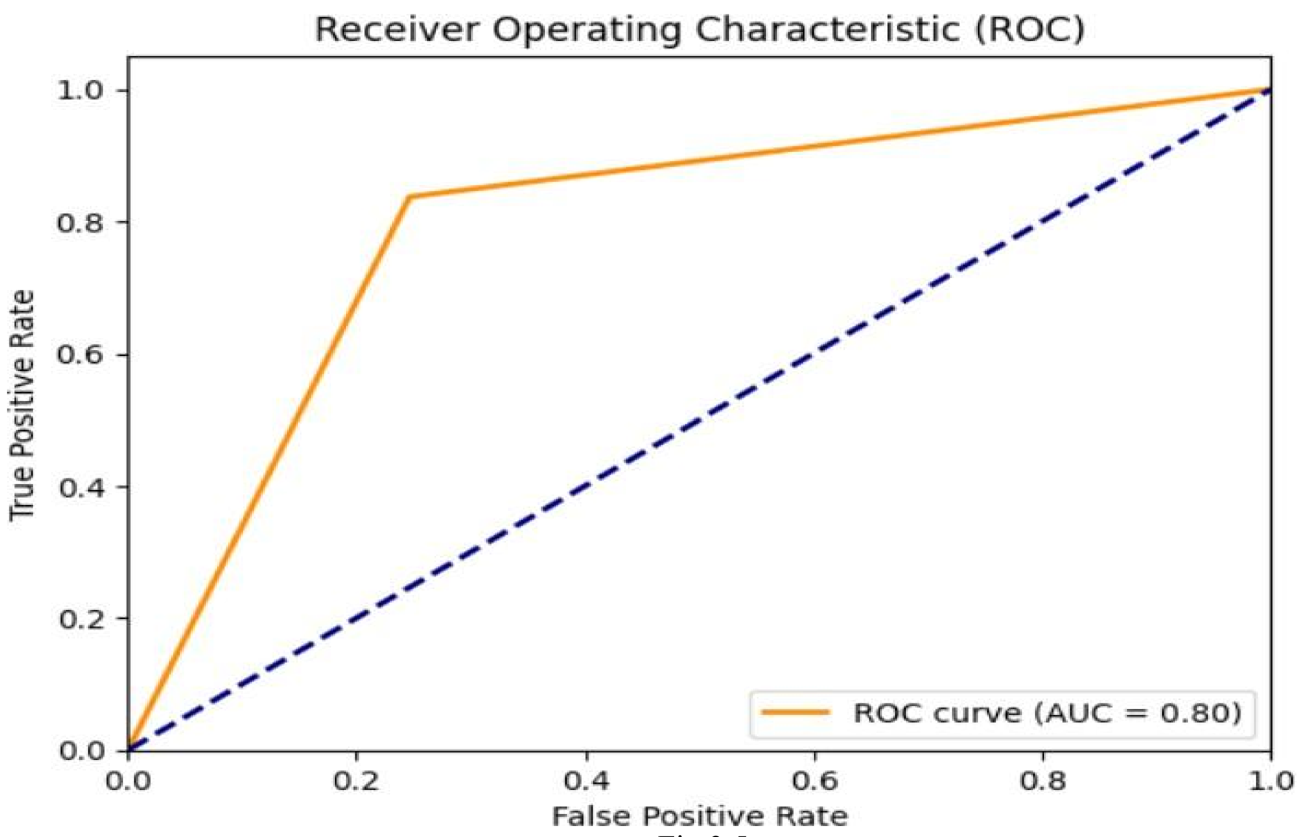


Fig.3.5

3.8 Code For ResNet50 Model:-

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.python.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
import numpy as np
import os
import cv2
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split, KFold
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.applications.resnet50 import ResNet50
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from keras import models
from keras.optimizers import Adam
from google.colab import drive
from tensorflow.keras.regularizers import l2
from keras import models
drive.mount('/content/drive')
images = np.load('/content/drive/MyDrive/images.npy')
labels = np.load('/content/drive/MyDrive/labels.npy')
img_height,img_width = 224,224
batch_size=40
kfold = Kfold(n_splits=5, shuffle=True, random_state=42)

for fold, (train_idx, val_idx) in enumerate(kfold.split(images, labels)):
    print(f"Fold: {fold}")
    X_train, X_val = images[train_idx], images[val_idx]
    Y_train, Y_val = labels[train_idx], labels[val_idx]

    resnet_model = Sequential()
    pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(224,224,3),
                   pooling='avg',classes=2,
                   weights='imagenet')
    resnet_model.add(pretrained_model)
    resnet_model.add(Flatten())
    resnet_model.add(Dense(512,activation='relu'))
    resnet_model.add(Dense(1,
kernel_regularizer=tf.keras.regularizers.l2(0.01),activation='linear'))

    for layer in pretrained_model.layers[:-16]:
        layer.trainable = False

    opt = Adam(learning_rate=0.001)
    resnet_model.compile(optimizer = 'adam', loss = 'hinge', metrics = ['accuracy'])

    # Set the path to save the model in your Google Drive
    model_path = f'/content/drive/MyDrive/Models/new_fold{fold}.h5'
    os.makedirs(os.path.dirname(model_path), exist_ok=True)
    model_checkpoint = ModelCheckpoint(model_path, monitor='val_accuracy',
save_best_only=True)

    history = resnet_model.fit(X_train, Y_train, batch_size=batch_size,
validation_data=(X_val, Y_val), epochs=100, callbacks=[model_checkpoint])
```

RESULTS OF ResNet50 Model:-

```
              precision    recall  f1-score   support

         mel       0.91      0.94      0.92       712
         ben       0.93      0.89      0.91       636

    accuracy                           0.92      1348
   macro avg       0.92      0.91      0.91      1348
weighted avg       0.92      0.92      0.92      1348

[[668  44]
 [ 70 566]]
```

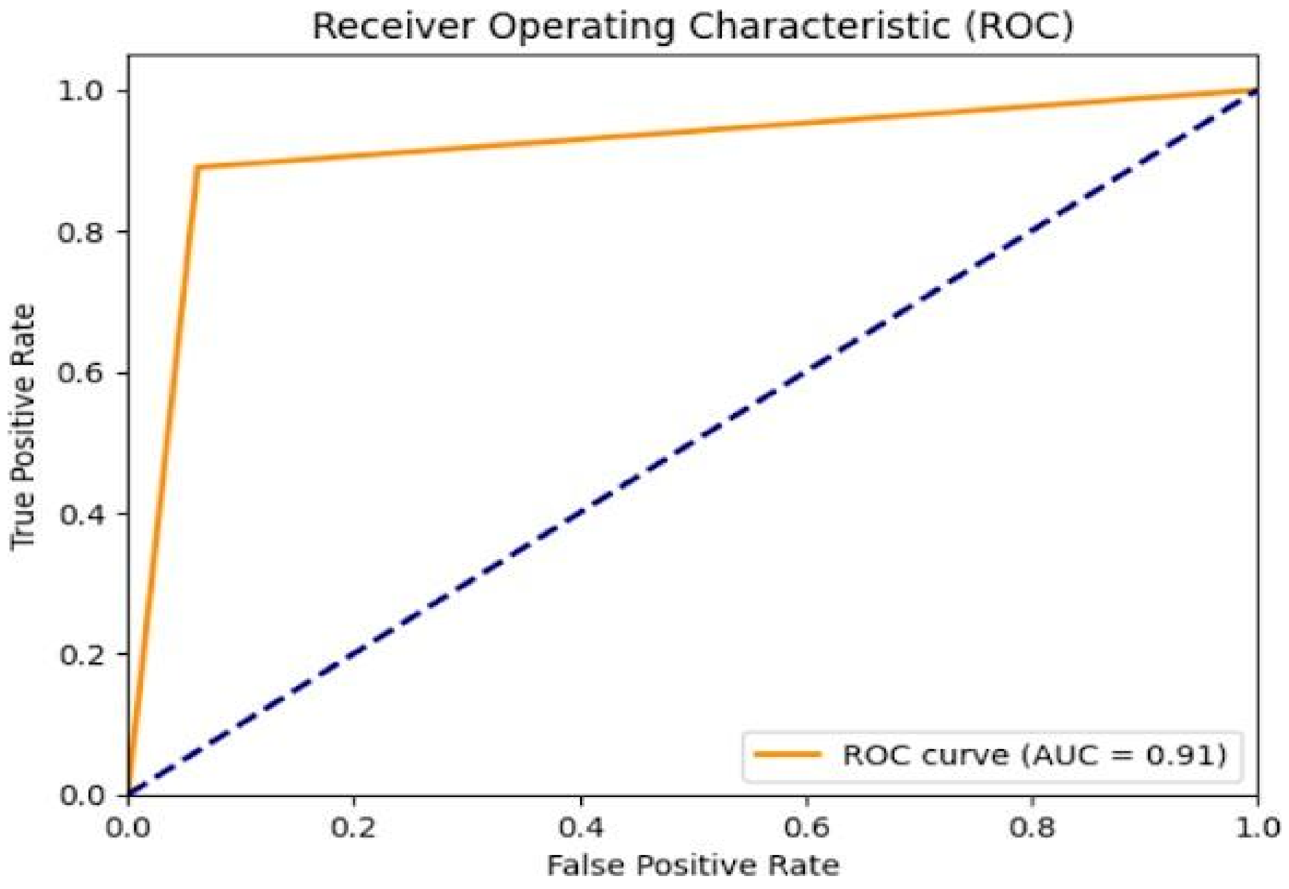


Fig.3.6

CHAPTER-4

CONCLUSION AND FUTURE PLANS

4.1 Conclusion

we can conclude that we have designed a model that helps to classify the skin cancer as melanoma or benign.we have used the Hybrid models like VGG-16,ResNet50 with SVM classifier to improve the correctness of classification.we have learned to extract the features from the images.These features are used to train the SVM classifier.These model can take a input image and classify the image as benign or melanoma.Finally we checked our results with the help of metrics like accuracy,precision,F1-Score and got accuracy around 91% using ResNet50 model.This can help the patients and doctors to classify the skin cancer as early as possible.

4.2 Future Plans

1.We can explore other advanced pre-built models and integrate them into your system. Other CNN models like InceptionV3, DenseNet or EfficientNet gave good results in image classification problems.
2.working with university and doctors(or)researchers to know the working of the model and checking the accuracy of the model with the help of real images
3.Developing a web app to upload the images and categorize the skin cancer

CHAPTER 5

REFERENCES

5.1 Base Paper

https://link.springer.com/article/10.1007/s11042-022-13081-x

5.2 Other References

1. Abbas Q, SadafM, Akram A (2016) Prediction of dermoscopy patterns for recognition of both melanocytic and non-melanocytic skin lesions. *Computers 5(3):13*.

2.Brinker TJ, Hekler A, Utikal JS, Grabe N, Schadendorf D, Klode J, Berking C, Steeb T, Enk AH (2018)and C. Von Kalle: Skin cancer classification using convolutional neural networks: systematic review. *JMed Internet Res 20(10):e11936*.

3.Dalila F, Zohra A, Reda K, Hocine C (2017) Segmentation and classification of melanoma and benign skin lesions. *Optik 140:749–761*.

4.Garnavi R, Aldeen M, Bailey J (2012) Computer-aided diagnosis of melanoma using border-and wavelet based texture analysis. *IEEE Trans Inf Technol Biomed 16(6):1239–1252*.

5.Hardie R, Ali R, Silva D, Kebede TM (2018) Skin lesion segmentation and classification for ISIC 2018using traditional classifiers with hand-crafted features. *arXiv preprint arXiv:1807.07001*.

6.Kasmi R, Mokrani K (2016) Classification of malignant and benign skin lesions: implementation ofautomatic ABCD rule. *IET Image Proc 10(6):448–455*.

7.Telea A (2004) An image inpainting technique based on the fast marching method.*J Graph Tools 9(1):23–34*.