Skin Cancer prediction using CNN

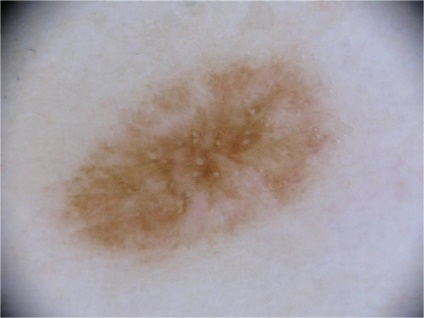
**Introduction-**

Skin cancer, the most common human malignancy, is primarily diagnosed visually, beginning with an initial clinical screening and followed potentially by dermoscopic analysis, a biopsy and histopathological examination. Automated classification of skin lesions using images is a challenging task owing to the fine-grained variability in the appearance of skin lesions. In order to diagnose skin cancer speedily at the earliest stage and solve some of the aforementioned problems, there has been extensive research solutions by developing computer image analysis algorithms. In this project, an augmented assistance to the dermatologist is provided using deep learning.

The dataset is taken from the ISIC (International Skin Image Collaboration) Archive. It consists of 700 pictures of benign type skin cancer and 700 pictures of malignant classified type skin cancer. The pictures have all been resized to low resolution (128 x 128 x 3) RGB. The task of this kernel is to create a model, which can classify a skin cancer visually into benign and malignant.

It has 2 different classes of skin cancer which are listed below:

1. Benign

1. Malignant

In this project, we will try to detect 2 different classes of skin cancer using Convolution Neural Network with Keras TensorFlow in backend and then analyse the result to see how the model can be useful in practical scenario.

**Project code-**

#Skin cancer prediction using CNN

#%%

#Importing the required libraries

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Conv2D

from keras.layers import MaxPool2D , Flatten , Dense, InputLayer, BatchNormalization, Dropout

from keras.callbacks import ModelCheckpoint

from keras.regularizers import l2

from matplotlib import pyplot

#%%

#Loading the dataset using Keras ImageDataGenerator and splitting it into train and test set by 20%

#Adding data augmentation for avoiding overfitting

imagegen = ImageDataGenerator(validation\_split = 0.2, rotation\_range=10, shear\_range=0.5, horizontal\_flip = True)

train = imagegen.flow\_from\_directory('D:/Academic/Main/TY/Sem-5/DLFL/IA1/training/', subset = 'training', target\_size=(128,128))

val = imagegen.flow\_from\_directory('D:/Academic/Main/TY/Sem-5/DLFL/IA1/training/', subset = 'validation', target\_size=(128,128))

#%%

#CNN Model Building

model = Sequential()

#Input layer

model.add(InputLayer(input\_shape = (128 , 128, 3)))

model.add(BatchNormalization())

#1st Conv block

model.add(Conv2D(25 , (5,5), activation = 'relu', strides = (1,1), padding = 'same', kernel\_regularizer=l2(0.01), bias\_regularizer=l2(0.01)))

model.add(Conv2D(25 , (5,5), activation = 'relu', strides = (1,1), padding = 'same', kernel\_regularizer=l2(0.01), bias\_regularizer=l2(0.01)))

model.add(MaxPool2D(pool\_size = (2,2), padding ='same'))

model.add(BatchNormalization())

model.add(Dropout(0.25))

#2nd Conv block

model.add(Conv2D(50 , (5,5), activation = 'relu', strides = (2,2), padding = 'same', kernel\_regularizer=l2(0.01), bias\_regularizer=l2(0.01)))

model.add(Conv2D(50 , (5,5), activation = 'relu', strides = (2,2), padding = 'same', kernel\_regularizer=l2(0.01), bias\_regularizer=l2(0.01)))

model.add(MaxPool2D(pool\_size = (2,2), padding ='same'))

model.add(BatchNormalization())

model.add(Dropout(0.25))

#3rd Conv block

model.add(Conv2D(70 , (3,3), activation = 'relu', strides = (2,2), padding = 'same', kernel\_regularizer=l2(0.01), bias\_regularizer=l2(0.01)))

model.add(Conv2D(70 , (3,3), activation = 'relu', strides = (2,2), padding = 'same', kernel\_regularizer=l2(0.01), bias\_regularizer=l2(0.01)))

model.add(MaxPool2D(pool\_size = (2,2), padding ='same'))

model.add(BatchNormalization())

model.add(Dropout(0.25))

#dense block

model.add(Flatten())

model.add(Dense(units=256, activation = 'relu', kernel\_regularizer=l2(0.01), bias\_regularizer=l2(0.01)))

total\_cat = 2 #Enter the total number of categories here

#Output layer

model.add(Dense(units = total\_cat , activation = 'softmax'))

#Compiling the model

model.compile(loss = 'binary\_crossentropy', optimizer = 'adam' , metrics =['accuracy'])

#Fitting the model on the data

history=model.fit\_generator(train, epochs=100 ,validation\_data = val)

#%%

#Visualization of the model performance

def summarize\_model(model, history,train , val):

# evaluate the model

\_, train\_acc = model.evaluate(train, verbose=1)

\_, test\_acc = model.evaluate(val, verbose=1)

print('Train: %.3f, Test: %.3f' % (train\_acc, test\_acc))

# plot loss during training

pyplot.subplot(211)

pyplot.title('Loss')

pyplot.plot(history.history['loss'], label='train')

pyplot.plot(history.history['val\_loss'], label='test')

pyplot.legend()

# plot accuracy during training

pyplot.subplot(212)

pyplot.title('Accuracy')

pyplot.plot(history.history['accuracy'], label='train')

pyplot.plot(history.history['val\_accuracy'], label='test')

pyplot.legend()

pyplot.show()

#%%

#Evaluate model behavior

summarize\_model(model, history,train , val)

#%%

#Making predictions for test set

pred=model.predict\_generator(val)

#%%

#Getting the true classes for the test set

true\_classes = val.classes

class\_labels = list(val.class\_indices.keys())

#%%

#Getting the class value for the made predictions

import numpy

predicted\_classes = numpy.argmax(pred, axis=1)

#%%

#Analysis of the model using precision and recall

import sklearn.metrics as metrics

report = metrics.classification\_report(true\_classes, predicted\_classes, target\_names=class\_labels)

print(report)

#%%

#Building the confusion matrix

confusion\_matrix = metrics.confusion\_matrix(y\_true=true\_classes, y\_pred=predicted\_classes)

#%%

#Visualizing the confusion matrix

import seaborn as sn

import pandas as pd

import matplotlib.pyplot as plt

array = confusion\_matrix

df\_cm = pd.DataFrame(array, range(2), range(2))

# plt.figure(figsize=(10,7))

sn.set(font\_scale=1.4) # for label size

sn.heatmap(df\_cm, annot=True, annot\_kws={"size": 16}) # font size

plt.show()

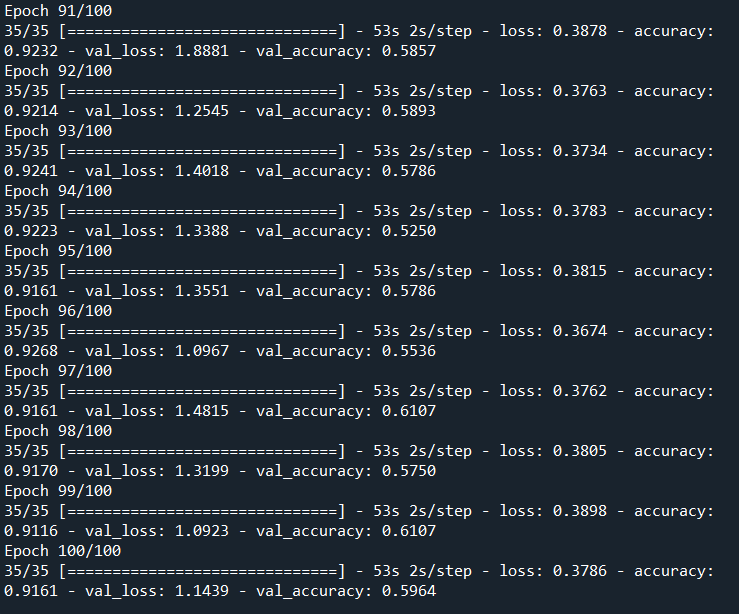
#%%

#Saving the model

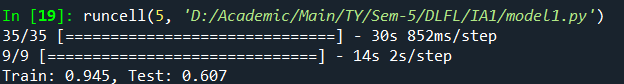
model.save('model1.h5')

**Output-**

Last 10 epochs:



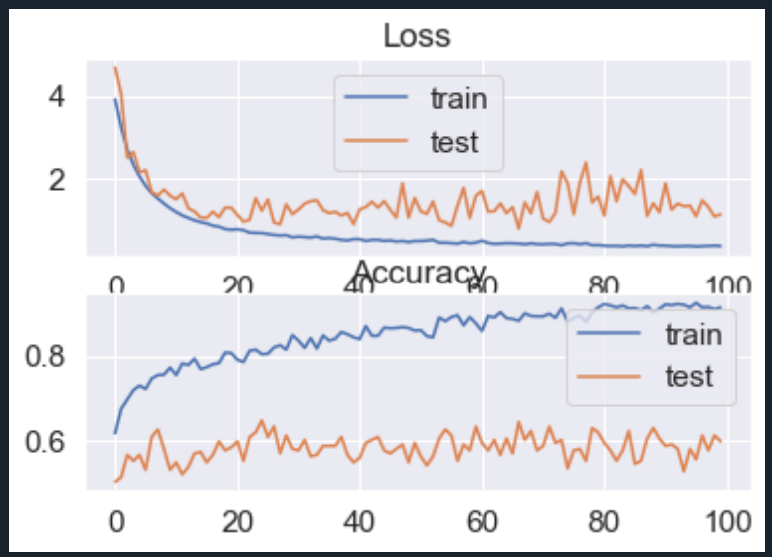
Final Training and Test accuracy:



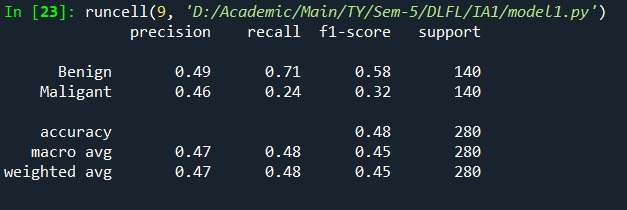
Highest val\_accuracy observed:



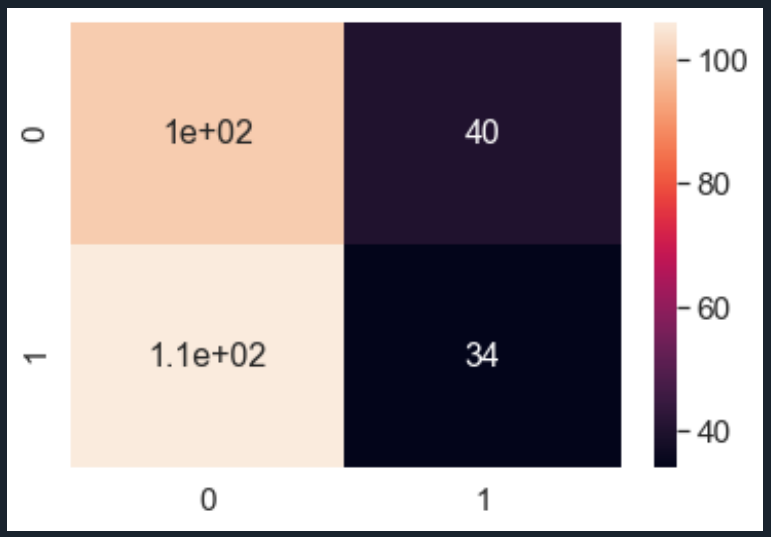
Loss and Accuracy Plots:



Precision and recall:



Confusion matrix:



**Observation-**

* In the above snapshots of the output we can observe that the training loss decreased smoothly with the increasing number of epochs whereas the validation loss fluctuates. Training accuracy increases continuously with the increasing number of epochs whereas validation accuracy becomes constant after some time, showing signs of overfitting.
* From the confusion matrix we can observe that true benign (class 0) have 140 images, out of which 100 were predicted correctly, true malignant (class 1) have 140 images, out of which only 34 were predicted correctly.
* The best model validation accuracy went up to 65%

**Conclusion-**

The training accuracy is 94.5% but the validation accuracy is less that is 60.7%. That means that the model is learning and performing a lot better on the training set but is unable to make accurate predictions on the validation set.

It is observed that the validation accuracy is getting constant after a point, main reason could be overfitting, for further work, we could work on making the model a bit less complex and introduce learning rate reduction and regularization to reduce overfitting of the model for more accurate results.

**Code explanation link-** (Use quality of 1080p)

<https://youtu.be/jwbmGtymkoQ>