MELANOMA DIAGNOSIS USING DEEP LEARNING TECHNIQUES ON DERMATOSCOPIC IMAGES

Project Report

Submitted in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND **ENGINEERING**

By

G. Deeshma Lavanya (181FA04143)
V. Sai Koushik (181FA04179)
T. Lakshman Sai (181FA04328)

Under the Supervision of
Mr.S. Deva Kumar
M.tech(ph.D)
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH (Deemed to be UNIVERSITY)
-Est u/s 3 of UGC act 1956
VADLAMUDI, GUNTUR – 522213
ANDHRA PRADESH, INDIA

JUNE 2022

VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH (Deemed to be UNIVERSITY)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project report entitled "**MELANOMA DIAGNOSIS USING DEEP LEARNING TECHNIQUES ON DERMATOSCOPIC IMAGES**" is the bonafide record of project work carried out under my supervision by **G. Deeshma Lavanya (181FA04143), V. Sai Koushik(181FA04179), T. Lakshman Sai (181FA04328)** during the academic year 2021-2022, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of Vignan's Foundation for Science, Technology and Research (Deemed to be University), Guntur. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

HOD, CSE
(Dr. Venkatesulu Dondeti)
Professor

Project Guide
(Mr. S. Deva Kumar)
Assistant Professor

DECLARATION

We hereby declare that the project report entitled "**MELANOMA DIAGNOSIS USING DEEP LEARNING TECHNIQUES ON DERMATOSCOPIC IMAGES**" has been written by us and has not been submitted either in part or whole for the award of any degree, diploma or any other similar title to this or any other university.

1. G. Deeshma Lavanya (181FA04143)
2. V. Sai Koushik (181FA04419)
3. T. Lakshman Sai (181FA04328)

Date: 28-06-2022

Place: VADLAMUDI

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to acknowledge the assistance and cooperation we have received from several persons while undertaking this B. Tech. Final Year Project. We owe special debt of gratitude to **Mr.S. Deva Kumar,** Department of Computer Science & Engineering, for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us.

We also take the opportunity to acknowledge the contribution of **Prof. Dr. Venkatesulu Dondeti,** Head, Department of Computer Science & Engineering, for his full support and assistance during the development of the project.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

G. Deeshma Lavanya (181FA04143)

V. Sai Koushik (181FA04179)

T. Lakshman Sai (181FA04328)

ABSTRACT

Cancer is the most threatening of them all. Among all the variants of cancer, skin cancer is spreading rapidly. It happens because of the abnormal growth of skin cells. The increase in ultraviolet radiation on the Earth's surface is also helping skin cancer spread in every corner of the world. Benign and malignant types are the most common skin cancers people suffer from. People go through expensive and time-consuming treatments to cure skin cancer but yet fail to lower the mortality rate. To reduce the mortality rate, early detection of skin cancer in its incipient phase is helpful. In today's world, deep learning is being used to detect diseases. The convolutional neural network (CNN) helps to find skin cancer through image classification more accurately. This research contains information about many CNN models and a comparison of their working processes for finding the best results. Pre-trained models like VGG16, VGG19, ResNet50, XcpetionNet, EfficientNet and self-built model (sequential) are used to analyze the process of CNN models. These models work differently as there are variations in their layer numbers. Depending on their layers and work processes, some models work better than others. An image dataset of benign and malignant data has been taken from Kaggle. In this dataset, there are 6594 images of benign and malignant skin cancer. Using different approaches, we have gained accurate results for VGG16 (79%), VGG19 (55%), Efficient Net(78%), ResNet50 (77%), Sequential Model (82%). This research compares these outcomes based on the model's work process. Our comparison includes model layer numbers, working process, and precision. The Sequential model has given us the highest accuracy of 82%.

.

CONTENTS

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

According to the World Cancer Research Fund, among all cancers, skin cancer is the 19th most common. People in the USA, Canada, and Australia have been diagnosed at the highest increasing rate over the past few decades. Skin cancer happens due to the uneven development of melanocytic skin cells [1]. Among all skin cancers, malignant and benign are the deadliest. A malignant tumor is a type of cancerous tumor that spreads and expands in a patient's body. They can infiltrate other tissues and organs and develop and spread unchecked. Many malignant skin growths have symptoms that can be identified as precursors. A precursor is a group of aberrant cells that may develop into cancer. Precancerous is another term for a precursor. Some precancerous skin growths have a minimal chance of developing into cancer, whereas others have a very high chance. There are many kinds of malignant skin growth, like melanoma, carcinoma, sarcoma, squamous cell carcinoma, and skin lymphoma [2]. The importance of detecting and treating cancer in early malignant skin growth cannot be overstated [3]. In many cases, complete excision (surgical removal) leads to healthiness. On the contrary, a benign tumor has the capability to develop, but it is not going to spread. When it comes to benign skin growths, knowing the common signs and symptoms of those that could be malignant is critical, as is seeking medical attention when skin growths show suspect. Benign skin growths include seborrheic keratoses, cherry angiomas, dermatofibromas, skin tags (acrochordon), pyrogenic granulomas, and cysts (epidermal inclusions). Here, Figure 1 shows cancer cases in men and women of different ages. Incidence rates rise steadily from around the age of 20 to 24 and more sharply in males from around the age of 55 to 59. Females aged 90 and up had the greatest rates, while males aged 85 to 89 had the lowest. Females have much greater rates of cancer than males in earlier age groups, while females have significantly lower rates of cancer in older age groups. The disparity is greatest between males and females between the ages of 20 and 24, when girls have a 2.5-fold greater age-specific incidence rate than males [4].

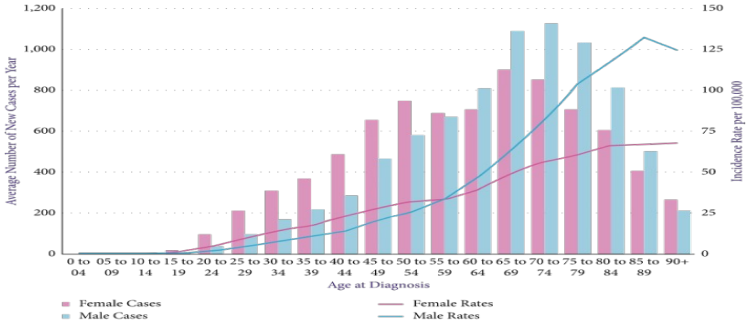

Fig 1.Melanoma skin cancer incidence by age

There are many deadly diseases in the current world. Skin cancer is one of them. Skin cancer cells grow and spread like tumors in the human body. If left unchecked, this tumor can become deadly by affecting other body tissues and organs. People go through expensive and time-consuming treatments to cure skin cancer but yet fail to lower the mortality rate. Detection of skin cancer at an early stage can help to reduce mortality. Machine Learning (ML) models have come up with the solution. Deep Learning, notably the Convolution Neural Network, can be used to identify skin cancer quickly and cheaply using image classification. It has become a lifesaver for poor people. These ML models are more accurate and faster in terms of detecting skin cancer via image classification. Medical science is developing in today's world. Previously, skin cancer was detected manually, which was difficult and expensive. But due to the advancement of deep learning in the medical science field, it has become much easier. For this reason, the CNN is proposed in the systems of this study to detect skin cancer.

1.2 PROBLEM STATEMENT

To identify which Deep learning technique gives the best accuracy. To predict whether the given image of a particular skin lesion is cancerous or non-cancerous.

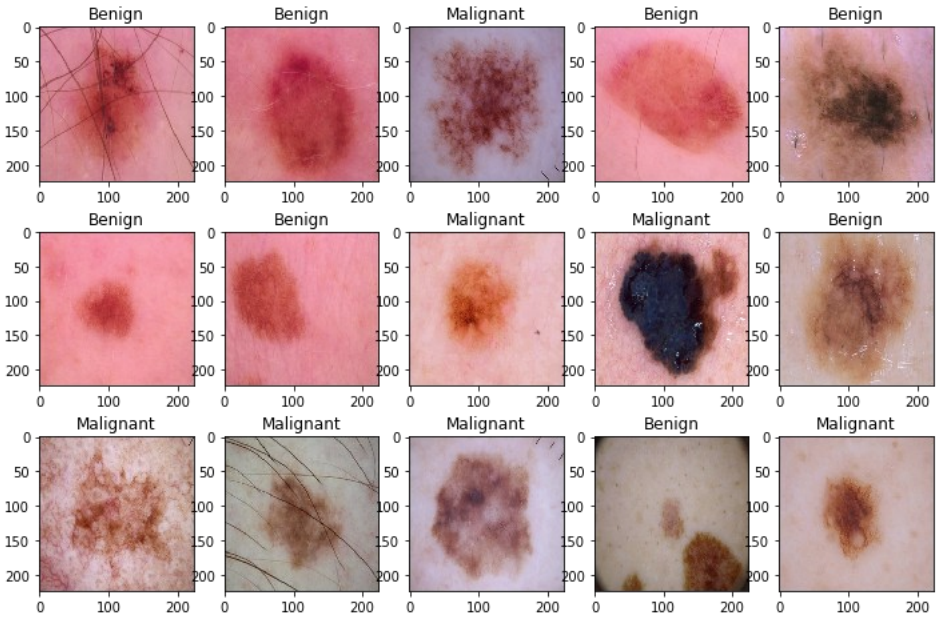

Fig.2 Benign vs Malignant

Chapter 2

LITERATURE REVIEW

Many researchers have applied CNN architectures to skin cancer datasets to develop a better solution to detect skin cancer at an incipient phase. Using cnn to analyze the skin lesions in dermoscopy images was introduced by **V. NWOGU AND I. SINGH, "ANALYZING SKIN LESIONS IN DERMOSCOPY IMAGES USING CONVOLUTIONAL NEURAL NETWORKS," IN *PROCEEDINGS OF THE 2018 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS (SMC)*, PP. 4035–4040, MIYAZAKI, JAPAN, 2018**. In this paper, we discuss the problem of automatic skin lesion analysis, specifically melanoma detection and semantic segmentation. We accomplish this by using deep learning techniques to perform classification on publicly available dermoscopic images. Skin cancer, of which melanoma is a type, is the most prevalent form of cancer in the US and more than four million cases are diagnosed in the US every year. In this work, we present our efforts towards an accessible, deep learning-based system that can be used for skin lesion classification, thus leading to an improved melanoma screening system. For classification, a deep convolutional neural network architecture is first implemented over the raw images. In addition, hand-coded features such as 166-D color histogram distribution, edge histogram and Multiscale Color local binary patterns are extracted from the images and presented to a random forest classifier. The average of the outputs from the two mentioned classifiers is taken as the final classification result. The classification task achieves an accuracy of 80.3%, AUC score of 0.69 and a precision score of 0.81. For segmentation, we implement a convolutional-deconvolutional architecture and the segmentation model achieves a Dice coefficient of 73.5%.

A convolutional-deconvolutional architecture is used to segment the data. Another group of researchers worked on the same dataset, but they used cnn based on symptomatic feature extraction in **M. R. HASAN, S. D. BARMAN, AND S. D. ISLAM, "SKIN CANCER DETECTION USING CONVOLUTIONAL NEURAL NETWORK," IN PROCEEDINGS OF THE 2019 5TH INTERNATIONAL CONFERENCE ON COMPUTING AND ARTIFICIAL INTELLIGENCE-ICCAI 19, PP. 254–258, ASSOCIATION FOR COMPUTING MACHINERY, NEW YORK, NY, USA, 2019**. Skin cancer is an alarming disease for mankind. The necessity of early diagnosis of the skin cancer have been increased because of the rapid growth rate of Melanoma skin cancer, itś high treatment costs, and death rate. This cancer cells are detected manually and it takes time to cure in most of the cases. This paper proposed an artificial skin cancer detection system using image processing and machine learning method. The features of the affected skin cells are extracted after the segmentation of the dermoscopic images using feature extraction technique. A deep learning based method

convolutional neural network classifier is used for the stratification of the extracted features.After segmenting the dermoscopic images using the feature extraction approach, the characteristics of the afflicted skin cells are retrieved in this work. They got an accuracy of 89.5%.

After segmenting the dermoscopic images using the feature extraction approach, the characteristics of the afflicted skin cells are retrieved in this work. They got an accuracy of 89.5%. A paper was published on the vision-based classification of skin cancer. They used CNN and the VGG16 in **S. KALOUCHE, "VISION-BASED CLASSIFICATION OF SKIN CANCER USING DEEP LEARNING," PP. 1–6, 2016**. This study proposes the use of deep learning algorithms to detect the presence of skin cancer, specifically melanoma, from images of skin lesions taken by a standard camera. Skin cancer is the most prevalent form of cancer in the US where 3.3 million people get treated each year. The 5-year survival rate of melanoma is 98% when detected and treated early yet over 10,000 people are lost each year due mostly to late-stage diagnoses. Thus, there is a need to make melanoma screening and diagnoses methods cheaper, quicker, simpler, and more accessible. This study aims to produce an inexpensive and fast computer-vision based machine learning tool that can be used by doctors and patients to track and classify suspicious skin lesions as benign or malignant with adequate accuracy using only a cell phone camera. The data set was trained on 3 separate learning models with increasingly improved classification accuracy. The 3 models included logistic regression, a deep neural network, and a fine-tuned, pre-trained, VGG-16 Convolutional Neural Network (CNN). Preliminary results show the developed algorithm's ability to segment moles from images with 70% accuracy and classify skin lesions as melanoma with 78% balanced accuracy using a fine-tuned VGG-16 CNN.

A deep convolutional neural network, logistic regression, and a fine-tuned, pretrained VGG16 were among the three models used in that system. A region-based cnn with resnet152 was also applied by a group of researchers on the isic dataset of 2742 dermoscopic images, where they got an accuracy of 90.4% in **M. F. J. ACOSTA, L. Y. C. TOVAR, M. F. GARCIA-ZAPIRAIN, M. B. G. ZAPIRAIN, AND W. S. PERCYBROOKS, "MELANOMA DIAGNOSIS USING DEEP LEARNING TECHNIQUES ON DERMATOSCOPIC IMAGES," BMC MEDICAL IMAGING, VOL. 21, NO. 6, 2021.** This research shows the comparison between some CNN models based on their work processes. CNNs are one of the most common types of neural networks that have been used for picture recognition and image classification. CNNs are also commonly utilized in domains such as object

detection, face recognition, and so on. Through back propagation, CNN learns to build spatial hierarchies of information automatically and adaptively using many essential elements, such as pooling (mx/average) layers, convolution layers, and fully connected layers [9]. These features help to identify skin cancer better than dermatologists can with their eyes. These features are used to get the best results. A comparison has been made between the received results and various CNN models. The dataset is fully prepared for our model to work on. The convolutional layer is utilized to separate different functionality from input pictures. Pre trained models like SVM, VGG16, and ResNet50 are used, as well as some sequential models with different parameters. This research could work better than the existing systems and help dermatologists around the world.

In **T. GUERGUEB AND M. A. AKHLOUFI, "MELANOMA SKIN CANCER DETECTION USING RECENT DEEP LEARNING MODELS," 2021 43RD ANNUAL INTERNATIONAL CONFERENCE OF THE IEEE ENGINEERING IN MEDICINE & BIOLOGY SOCIETY (EMBC), 2021, PP. 3074-3077, DOI: 10.1109/EMBC46164.2021.9631047** melanoma is considered as one of the world's deadly cancers. This type of skin cancer will spread to other areas of the body if not detected at an early stage. Convolutional Neural Network (CNN) based classifiers are currently considered one of the most effective melanoma detection techniques. This study presents the use of recent deep CNN approaches to detect melanoma skin cancer and investigate suspicious lesions. Tests were conducted using a set of more than 36,000 images extracted from multiple datasets. The obtained results show that the best performing deep learning approach achieves high scores with an accuracy and Area Under Curve (AUC) above 99%.

Chapter 3

REQUIREMENT ANALYSIS

3.1 REQUIREMENT ANALYSIS

The project involved analyzing the design of few applications so as to make the application more users friendly. To do so, it was really important to keep the navigations from one screen to the other well ordered and at the same time reducing the amount of typing the user needs to do. In order to make the application more accessible, the browser version had to be chosen so that it is compatible with most of the Browsers.

3.2 REQUIREMENT SPECIFICATION

3.2.1. FUNCTIONAL REQUIREMENTS

- Data Collection
- Data Preprocessing
- Training And Testing
- Modeling
- Predicting

3.2.2. NON FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, “how fast does the website load?” Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non- functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

- Usability requirement
- Serviceability requirement
- Manageability requirement
- Recoverability requirement

- Security requirement
- Data Integrity requirement
- Capacity requirement
- Availability requirement
- Scalability requirement
- Interoperability requirement
- Reliability requirement
- Maintainability requirement
- Regulatory requirement
- Environmental requirement

Here, are some examples of non-functional requirement:

- Users must upload dataset
- The software should be portable. So moving from one OS to other OS does not create any problem.
- Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

3.2.3. ADVANTAGES OF NON-FUNCTIONAL REQUIREMENT

Benefits/pros of Non-functional testing are:

- The nonfunctional requirements ensure the software system follow legal and compliance rules.
- They ensure the reliability, availability, and performance of the software system
- They ensure good user experience and ease of operating the software.
- They help in formulating security policy of the software system.

3.2.4. DISADVANTAGES OF NON-FUNCTIONAL REQUIREMENT

Cons/drawbacks of Non-function requirement are:

- None functional requirement may affect the various high-level software subsystem
- They require special consideration during the software architecture/high-level design phase which increases costs.
- Their implementation does not usually map to the specific software sub-system,
- It is tough to modify non-functional once you pass the architecture phase.

Functional Requirements

- Graphical User interface with the User.

Software Requirements

For developing the application the following are the Software Requirements:

- Python.

Operating Systems supported

1. Windows 7
2. Windows XP
3. Windows 8 or more

Technologies and Languages used to Develop

1. Python

Hardware Requirements

For developing the application the following are the Hardware Requirements:

- Processor: Pentium IV or higher
- RAM: 256 MB
- Space on Hard Disk: minimum 512MB

3.3. SYSTEM SPECIFICATION:

3.3.1. HARDWARE REQUIREMENTS:

- **System** : Pentium IV 2.4 GHz.
- **Hard Disk** : 40 GB.
- **Floppy Drive** : 1.44 Mb.
- **Monitor** : 14’ Colour Monitor.
- **Mouse** : Optical Mouse.
- **Ram** : 512 Mb.

3.3.2. SOFTWARE REQUIREMENTS:

- **Operating system** : Windows 7 Ultimate.
- **Coding Language** : Python.
- **Front-End** : Python.

Chapter 4

PROPOSED METHODOLOGY

4.1 PROPOSED SYSTEM

We have acquired a total of 3,297 dermoscopic images of unique malignant and benign skin lesions from over 2000 patients. Our methodology involves use of Deep learning techniques, Convolutional Neural Network(CNN) to be precise such as Xception Net, Effifcient Net, ResNet50 to classify the skin lesions as cancerous or non-cancerus.

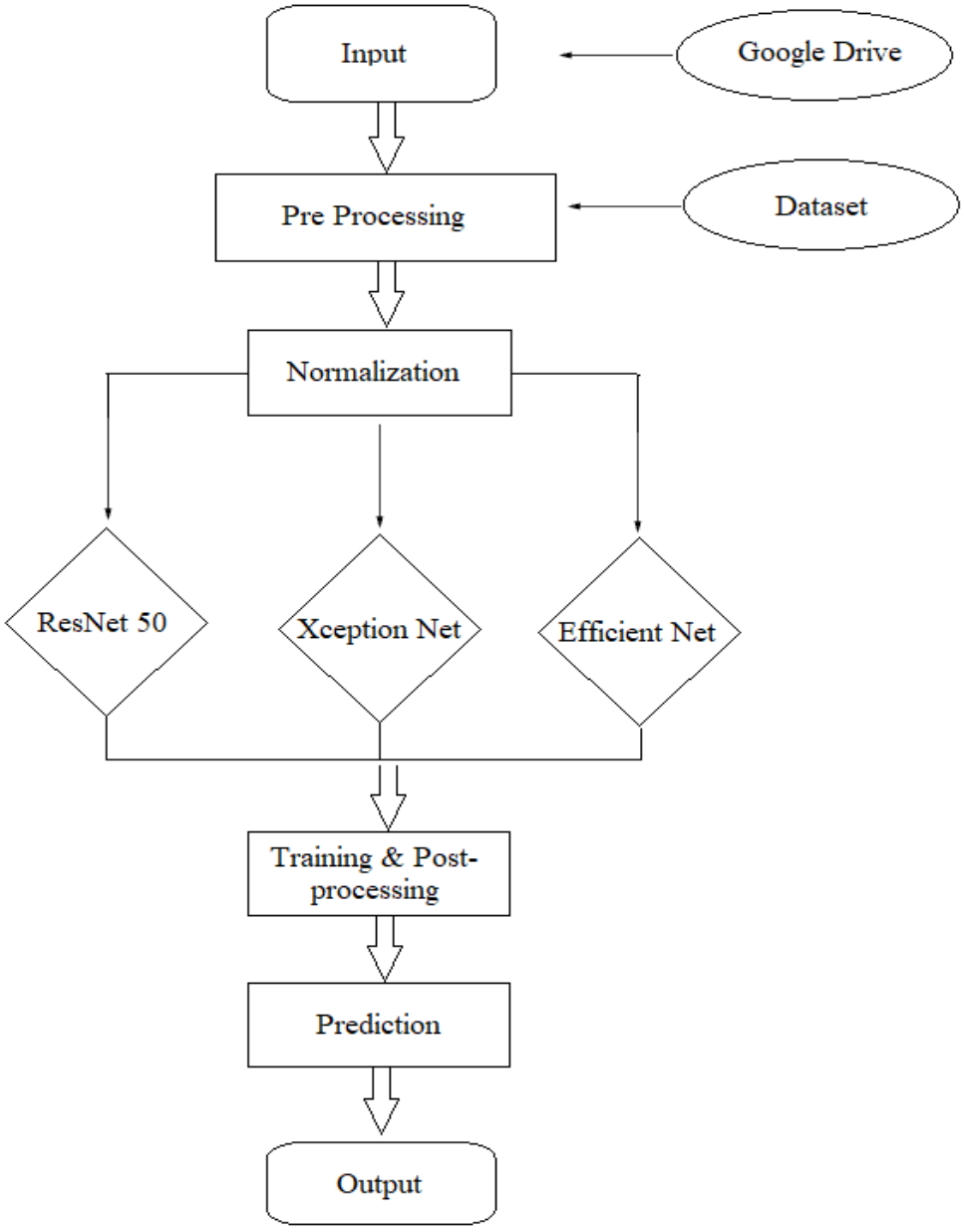

Fig.3. Flowchart

4.2 TENSORFLOW FRAMEWORK

Tensor flow is an open-source software library. Tensor flow was originally developed by researchers and engineers. It is working on the Google Brain Team within Google's Machine Intelligence research organization the purposes of conducting machine learning and deep neural networks research. It is an opensource framework to run deep learning and other statistical and predictive analytics workloads. It is a python library that supports many classification and regression algorithms and more generally deep learning. TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google, TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). Tensor Flow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. 5 Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. The name Tensor Flow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

4.3 NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of highlevel mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Num array into Numeric, with extensive modifications. NumPy is opensource software and has many contributors. The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier. An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric also variously called Numerical Python extensions or NumPy Hugunin, 8

a graduate student at Massachusetts Institute of Technology (MIT) joined the Corporation for National Research Initiatives (CNRI) to work on J Python in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported num-array's features to Numeric, releasing the result as NumPy 1.0 in 2006. This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy.

4.4 MATPLOT

Mat plot is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, WX Python, Qt, or GTK+. There is also a procedural "Pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter, has an active development community and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012 and further joined by Thomas Caswell.

4.5 PANDAS

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users. Pandas is a python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible opensource data analysis/manipulation tool available in any language. It is already well on its way toward this goal. Explore data analysis with Python. Pandas Data Frames make manipulating your data easy, from selecting or replacing columns and indices to reshaping your data. Pandas is well suited for many different kinds of data: 12 Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet Ordered and unordered (not necessarily fixed-frequency) time series data. Arbitrary matrix data 9 (homogeneously typed or heterogeneous) with row and column labels Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a Pandas data structure.

4.6 KERAS

KERAS is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides. Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel. Keras is a minimalist Python library for deep learning that can run on top of Theano or Tensor Flow. It was developed to make implementing deep learning models as fast and easy as possible for research and development.

FOUR PRINCIPLES:

• 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.

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

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

• Python: No separate model files with custom file formats. Everything is native Python. Keras is designed for minimalism and modularity allowing you to very quickly define deep learning models and run them on top of a Theano or TensorFlow backend.

Chapter 5

METHODOLOGY

5.1 CONVOLUTIONAL NEURAL NETWORK

Deep learning is an AI function that mimics the workings of the human brain in processing data for use in detecting objects, recognizing speech, translating languages, and making decisions. Deep learning AI is able to learn without human supervision, drawing from data that is both unstructured and unlabeled.

Convolutional Neural Networks (CNN) are artificial feed-forward networks where singular neurons are arranged in such a way that they react to the visual field especially covering districts. Fukushima's work first registers models dependent on these neighborhood networks among neurons and on progressively coordinated changes of the picture. The output layer is tailored to the application's requirements. The weighted total is computed by the activation function and bias is added. This determines whether or not a neuron should be activated. Rectified Linear Unit (ReLU), Softmax, sigmoid, and other activation functions are available and can be employed depending on the task.

A CNN uses a system much like a multilayer perceptron that has been designed for reduced processing requirements. CNN hidden layers consist of different layers like convolution, pooling and some of the activation functions, filters, optimizers etc.

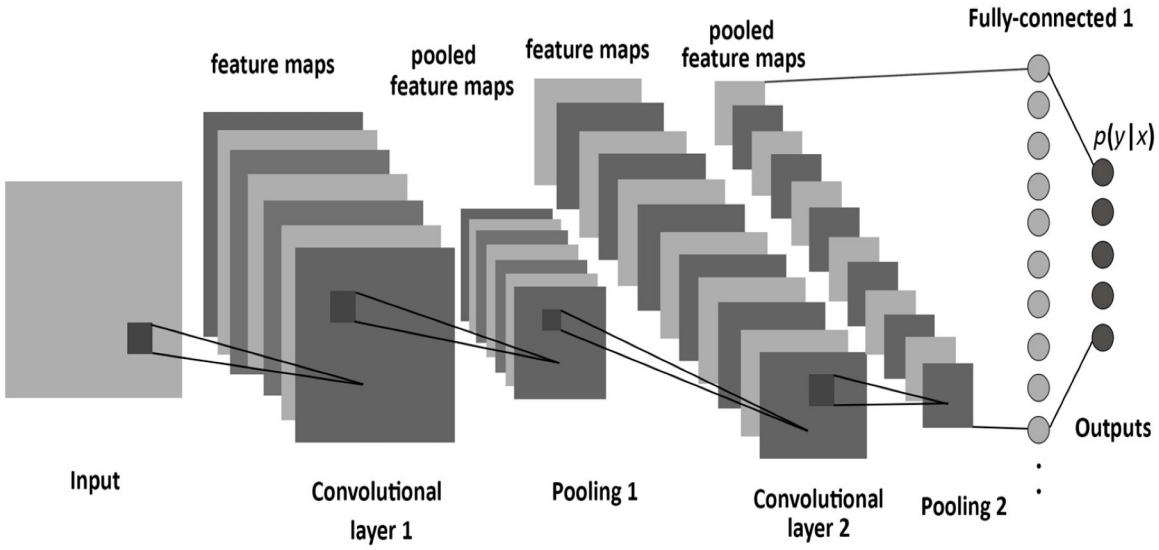

Fig.4. Architecture of deep convolutional neural network

Basic structure of CNN, where C1, C3 are convolution layers and S2, S4 are pooled/sampled layers.

5.1.2. CONVOLUTION LAYER

For every CNN architecture, convolutional layers are the most integral feature. It comprises a collection of convolutional kernels (filters) that are highly integrated to build an output feature map from the source data (N-dimensional parameters). Figure 4 shows how a convolutional network takes data from an input and gives an output.

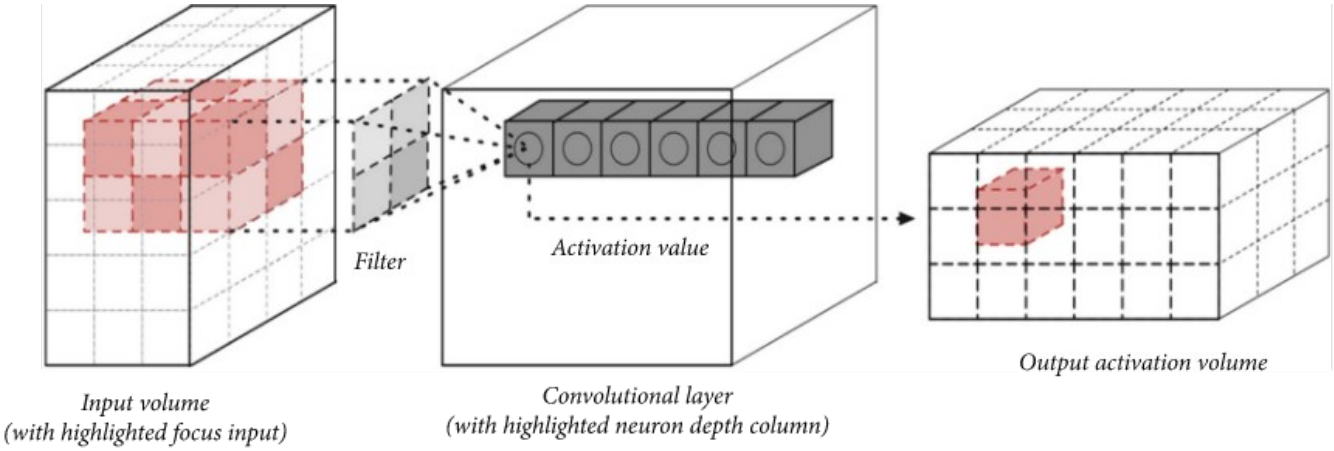

Fig.5. Convolution layer

A single convolutional layer has an input volume. From the input source, it filters out some segments of data, with the activation value reserved for those. Then the activation value is processed and sent to the output activation volume. All the communication is done using kernel tricks. A kernel is an array of continuous or integer values, with every value reflecting the kernel weight. All the kernel weights are allocated random integers when the training process of a CNN model begins (other ways of initializing the weights are also available). The weights are then fine-tuned with each training epoch, and the kernel learns to extract significant information.

5.1.3. POOLING LAYER

Feature maps are subsampled using pooling layers (produced following convolution operations), compressing larger feature maps into smaller ones. The most significant features (or content) for each pool stage are always kept when the feature maps are shrunk. Pooling is done in the same way as convolution is done, by determining the operation stride and the size of the region of pooling. In different pooling layers, several types of pooling techniques are utilized, among them are maximum,

average, minimum, gated, and tree pooling, to name a few [14]. The most popular and often used pooling approach is max-pooling**.**

5.1.4. FULLY CONNECTED

Fully connected layers, where each cell in one layer is related to every cell in the preceding layer, make up the last component (or layers) of the convolution layer (which is used for classification). The CNN architecture's output unit (classifier) is the final layer of fully connected layers. FC layers (FCLs) are one of many kinds of feedforward artificial neural networks (ANN) that work similarly to a neural network like multilayer perceptron (MLP) [15]. The fully connected layers receive input after the last convolutional or pooling layer in accordance with a set of parameters (feature maps), which are compressed to generate a variable, which is then passed into the FC layer to construct CNN's end result.

5.1.5. CONFUSION MATRIX

A confusion matrix (also known as an error matrix) is a quantitative approach to describing picture categorization accuracy, and it is a table that summarizes the results of a classification model. The number of correct and incorrect estimates is tallied and burned away by class. The confusion matrix is based on true negative (TN), false negative (FN), true positive (TP), and false positive (FP). The confusion matrix (CM) helps to find other results more accurately. Some of the equations used to calculate CM are given below in equations :

$$\mathrm{TN}_i = \sum_{j=1, j\neq 1}^{n} \sum_{k=1, k\neq 1}^{n} a_{ij},$$

$$\mathrm{FN}_i = \sum_{j=1, j\neq 1}^{n} a_{ij}.$$

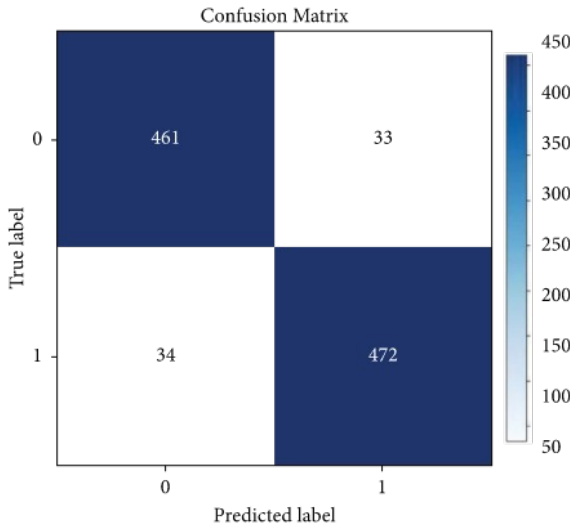

Fig.6. Sample Confusion Matrix

CM has two main points. One is the true label, where all the stored true value is saved. This label is situated on the *x*-axis. Another is the predicted label, where results from the trained systems are given to compare with the original true value. This label is situated on the *y*-axis. From all the values, the true positive part gives the correct result. The true positive part can be found on a graph where values with the same label from both axes (*x* and *y*) are matched. With the help of CM, other equations can be run as stated below.

$$\text{Accuracy} = \frac{\#\text{ of correct predictions}}{\text{total }\#\text{ of predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}.$$

In the above equation , the accuracy formula is given. Accuracy can be calculated through CM parameters. Here, the total number of correct predictions is divided by the total number of predictions:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}.$$

In the above equation, the precision formula is given. Precision can also be calculated with CM parameters. Here, the total number of true positive cases is divided by the summation of the number of true positive cases and the number of false positive cases.

5.2 BLOCK DIAGRAM

The model consists of two phases- pre processing phase and implementing phase

1. **Pre-processing :** In this step, the images present in the dataset are pre-processed to remove noise, fill in blanks, and so on.
2. **Implementing phase:** The pre-processed images are now sent into each model, which decides whether or not the lesion is cancerous.

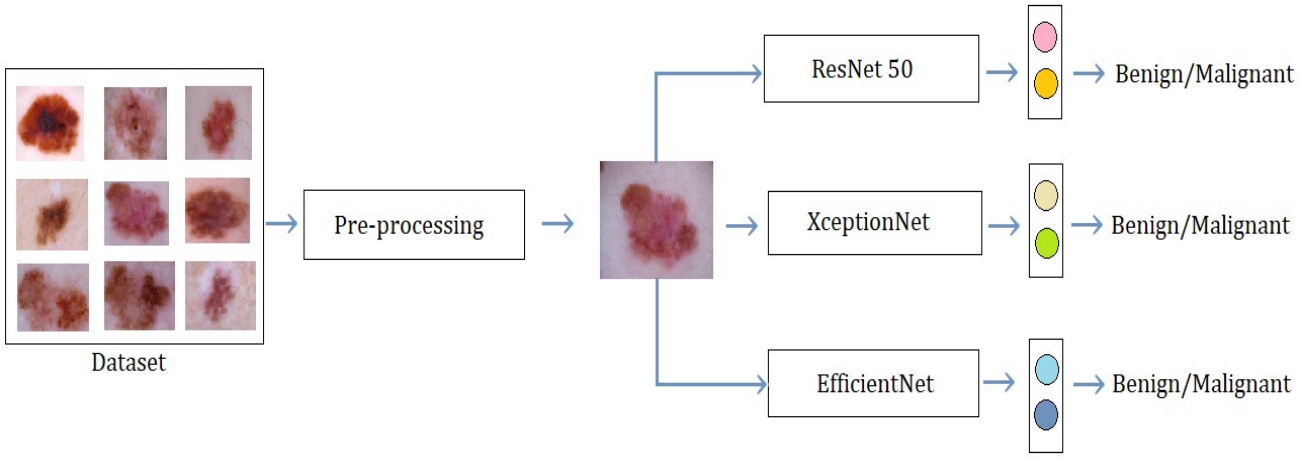

Fig.7. Block diagram

5.2.1. DATA VISUALIZATION

In the first step, let us visualize the total number of images in our dataset in both categories. We can see that there are 3,297 images in which 1800 images belong to the class benign and 1497 images belong to the class malignant.

5.2.2. DATA AUGMENTATION

In the next step, we augment our dataset to include more number of images for our training. In this step of data augmentation, we rotate and flip each of the images in our dataset.

5.2.3. SPLITTING THE DATA

In this step, we split our data into the training set which will contain the images on which the CNN model will be trained and the test set with the images on which our model will be tested.

5.2.4. BUILDING THE MODEL

In the next step, we build our Sequential CNN model with various layers such as Conv2D, MaxPooling2D, Flatten, Dropout and Dense.

5.2.5. TRAINING THE CNN MODEL

This step is the main step where we fit our images in the training set and the test set to our Sequential model we built using Keras library. However, we can train for more number of epochs to attain higher accuracy lest there occurs over-fitting.

5.2.6. LABELING THE INFORMATION

After building the model, we label two probabilities for our results. [‘0’ as ‘benign’ and ‘1’ as ‘malignant’]. The boundary is set in rectangle color using the RGB values.

5.3. CONVOLUTIONAL NEURAL NETWORK MODELS

The CNN model handles data in a grid pattern, such as images. It is intended to automatically learn the spatial hierarchies of features. CNN analyzes the raw pixel data from the image, trains the model, and then extracts the features for improved categorization. A number of CNN models like SVM, VGG16, ResNet50, and sequential are presented here that use unstructured growth of skin cancer images as inputs to specify benign and malignant skin cancer.

5.3.1. RESNET 50

ResNet50 is a residual network with 50 layers and 26 million parameters. In 2015, Microsoft introduced the residual network, a deep convolutional neural network model. Rather than learning features, the residual network learns from residuals, which are the subtraction of features learnt before the inputs of the layer. The skip connection was used by ResNet to transport information across layers [22]. Figure 8 shows how all the 50 layers are connected to each other in ResNet50. The architecture of ResNet50 is separated into 4 stages, as seen in the picture given. An impression with a size that is multiples of 32 and a channel width of three can be accepted by the system. For the sake of clarity, we will suppose the filter size is 224 by 224 by 3. For preliminary convolution and max-pooling, each ResNet design employs seventy-seven percent and thirty-three percent kernel sizes, correspondingly. Following that, the network's first step begins, which is made up of

three residual blocks, all with all three layers. In all 3 layers of stage 1's unit, the kernels then used to execute the convolution process are 64, 64, and 128 in size, correspondingly.

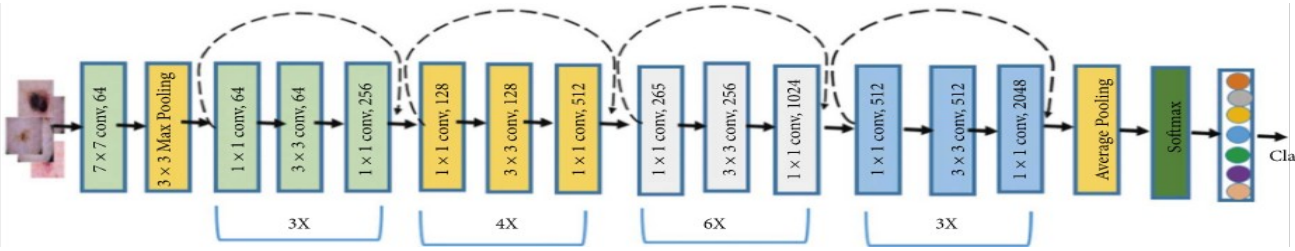

Fig.8. ResNet 50 Architecture

The curvy arrow indicates the same connection. The dotted connection arrows represent that stride two is used for convolution in the residual block, providing a 1/2 input in terms of height and breadth but twice the channel width. The average pooling layer is the system's bottom layer. Then comes a fully linked layer of a thousand neurons. Malignant (total 2994 images) and benign (total 3600 images) cancer images are included in our dataset.

5.3.2. EFFICIENT NET

EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient. Unlike conventional practice that arbitrary scales these factors, the EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. For example, if we want to use 2^N times more computational resources, then we can simply increase the network depth by α^N, width by β^N, and image size by γ^N, where α,β,γ are constant coefficients determined by a small grid search on the original small model. EfficientNet uses a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way.

The compound scaling method is justified by the intuition that if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image. The base EfficientNet-B0 network is based on the inverted bottleneck residual blocks of MobileNetV2, in addition to squeeze-and-excitation blocks.

Efficient Net also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, with an order of magnitude fewer parameters.

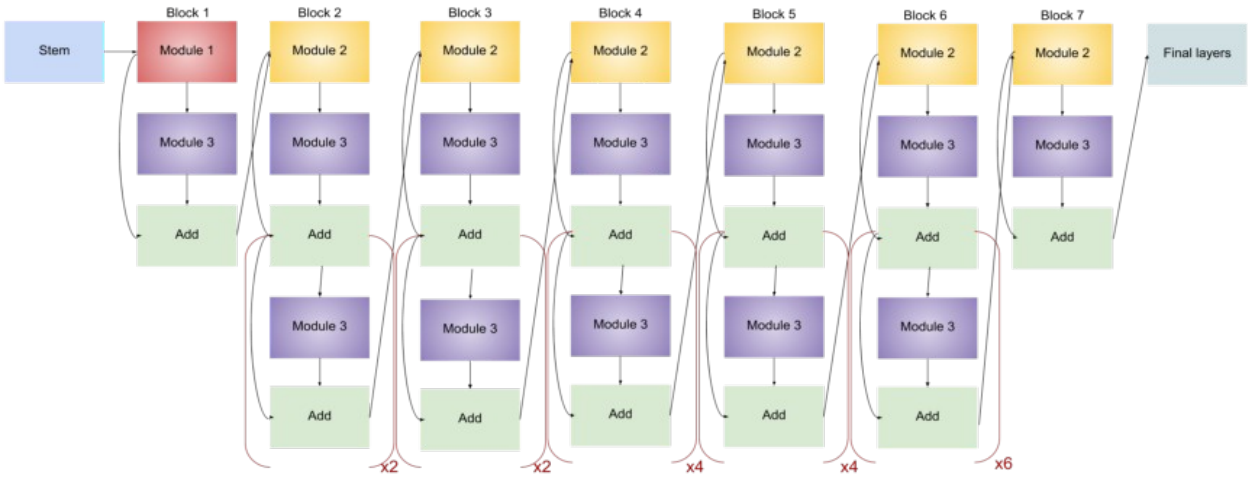

Fig.9. Efficient Net Architecture

5.3.3. XCEPTION NET

Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. This network was introduced Francois Chollet who works at Google, Inc. (Fun-Fact: He is the creator of keras).

Xception is also known as “extreme” version of an Inception module. Hence, let us look at the Inception module before delving into Xception.

Xception stands for “extreme inception”, it takes the principles of Inception to an extreme. In Inception, 1x1 convolutions were used to compress the original input, and from each of those input spaces we used different type of filters on each of the depth space. Xception just reverses this step. Instead, it first applies the filters on each of the depth map and then finally compresses the input space using 1X1 convolution by applying it across the depth. This method is almost identical to a depthwise sepa- rable convolution, an operation that has been used in neural network design as early as 2014. There is one more difference between Inception and Xception. The presence or absence of a non-

linearity after the first operation. In Inception model, both operations are followed by a ReLU non-linearity, however Xception doen't introduce any non-linearity.

The data first goes through the entry flow, then after than it. goes through the middle flow (repeating itself 8 times in this middle flow), and finally through the exit flow.

Xception implemented using the TensorFlow framework by Google and trained on 60 NVIDIA K80 GPUs each.

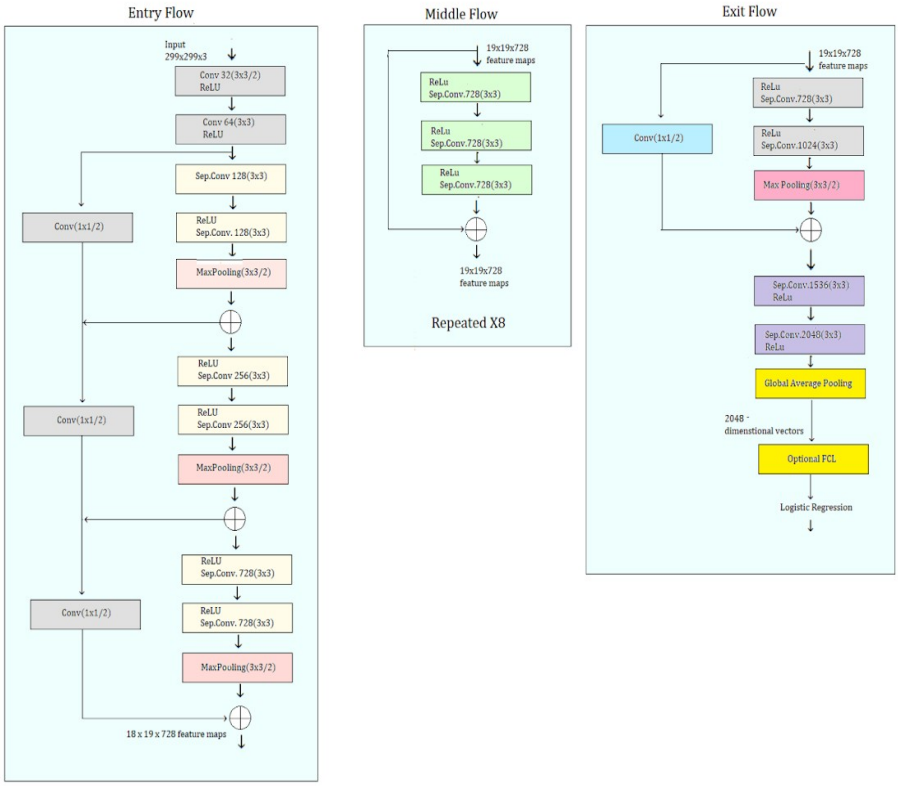

Fig.10. Xception Net Architecture

Chapter 6
IMPLEMENTATION

6.1 USING RESNET 50

```
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from keras.layers import Input, Lambda, Dense, Flatten, GlobalAveragePooling2D, MaxPooling2D,
Dropout
from keras.models import Model
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.models import Sequential
import numpy as np
from keras.models import Model
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard



from google.colab import drive
drive.mount('/content/drive')



from warnings import filterwarnings
filterwarnings('ignore')



train_set='/content/drive/MyDrive/Melanoma Dataset/data/train'
test_set='/content/drive/MyDrive/Melanoma Dataset/data/test'
image_input = Input(shape=(224, 224, 3))
model = ResNet50(input_tensor=image_input, weights='imagenet', include_top=False)
model.summary()
```

```
last_layer = model.get_layer('conv5_block3_add').output
out = Dense(1, activation='sigmoid', name='output')(last_layer)
custom_vggt_model = Model(image_input, out)
custom_vgg_model.summary()



from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

#Training Set
train_set = train_datagen.flow_from_directory(r'/content/drive/MyDrive/Melanoma Dataset/data/train',
                                              target_size=(64,64),
                                              batch_size=16,
                                              class_mode='binary')
#Validation Set
test_set = test_datagen.flow_from_directory(r'/content/drive/MyDrive/Melanoma Dataset/data/test',
                                            target_size=(64,64),
                                            batch_size = 16,
                                            class_mode='binary',
                                            shuffle=False)



classifier=Sequential()
classifier.add(Flatten())
classifier.add(Dense(units=128,activation='relu'))
classifier.add(Dense(units=1,activation='sigmoid'))
```

```
adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=Fal
se)
classifier.compile(optimizer=adam,loss='binary_crossentropy',metrics=['accuracy'])
history = classifier.fit(train_set,
                         steps_per_epoch=(2224/16),
                         epochs = 30,
                         validation_data = test_set,
                         validation_steps =(400/16),
                         #callbacks=[tensorboard]
                         );



import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
%matplotlib inline
import tensorflow
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg', target_siz
e=(64, 64))
img = image.img_to_array(img1)
img = img/255
# create a batch of size 1 [N,H,W,C]
img = np.expand_dims(img, axis=0)
prediction = classifier.predict(img, batch_size=None,steps=1) #gives all class prob.
if(prediction[:,:]>0.5):
    value ='Malignant :%1.2f'%(prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(alpha=0.8))
else:
    value ='Benign :%1.2f'%(1.0-prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(alpha=0.8))

plt.imshow(img1)
plt.show()



import pandas as pd
test_set.reset
ytesthat = classifier.predict_generator(test_set)
df = pd.DataFrame({
    'filename':test_set.filenames,
    'predict':ytesthat[:,0],
    'y':test_set.classes
})
```

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)
df['y_pred'] = df['predict']>0.5
df.y_pred = df.y_pred.astype(int)
df.head(20)


misclassified = df[df['y']!=df['y_pred']]
print('Total misclassified image from 660 Validation images : %d'%misclassified['y'].count())


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

conf_matrix = confusion_matrix(df.y,df.y_pred)
sns.heatmap(conf_matrix,cmap="YlGnBu",annot=True,fmt='g');
plt.xlabel('predicted value')
plt.ylabel('true value');


img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg')
plt.imshow(img1);
#preprocess image
img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg', target_siz
e=(64, 64))
img = image.img_to_array(img1)
img = img/255
img = np.expand_dims(img, axis=0)
x1 = classifier.evaluate_generator(train_set)
x2 = classifier.evaluate_generator(test_set)
```

```
print('Training Accuracy  : %1.2f%%     Training loss  : %1.6f'%(x1[1]*100,x1[0]))
print('Validation Accuracy: %1.2f%%     Validation loss: %1.6f'%(x2[1]*100,x2[0]))


from sklearn.metrics import confusion_matrix
import pandas as pd
confusion_df = pd.DataFrame(confusion_matrix(df.y,df.y_pred),
                     columns=["Predicted Class" + str(class_name) for class_name in [0,1]],
                     index = ["Class" + str(class_name) for class_name in [0,1]])

print(confusion_df)


from sklearn.metrics import classification_report
print(classification_report(df.y,df.y_pred))
```

6.2 USING XCEPTION NET

```
from tensorflow.keras.applications.xception import Xception
from keras.layers import Input, Lambda, Dense, Flatten, Conv2D, MaxPooling2D
from keras.models import Model
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard


from google.colab import drive
drive.mount('/content/drive')


image_input = Input(shape=(224, 224, 3))
train_set='/content/drive/MyDrive/Melanoma Dataset/data/train'
test_set='/content/drive/MyDrive/Melanoma Dataset/data/test'
model = Xception(input_tensor=image_input, weights='imagenet', include_top=False)
model.summary()


last_layer = model.get_layer('block14_sepconv2_bn').output
out = Dense(1, activation='sigmoid', name='output')(last_layer)
custom_model = Model(image_input, out)
custom_model.summary()


from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255,
```

```
                    shear_range=0.1,
                    zoom_range=0.1,
                    horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

#Training Set
train_set = train_datagen.flow_from_directory(r'/content/drive/MyDrive/Melanoma Dataset/data/train',
                                 target_size=(64,64),
                                 batch_size=16,
                                 class_mode='binary')
#Validation Set
test_set = test_datagen.flow_from_directory(r'/content/drive/MyDrive/Melanoma Dataset/data/test',
                                target_size=(64,64),
                                batch_size = 16,
                                class_mode='binary',
                                shuffle=False)



classifier=Sequential()
classifier.add(Flatten())
classifier.add(Dense(units=128,activation='relu'))
classifier.add(Dense(units=1,activation='sigmoid'))
adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=Fal
se)
classifier.compile(optimizer=adam,loss='binary_crossentropy',metrics=['accuracy'])
history = classifier.fit(train_set,
                steps_per_epoch=(2224/16),
                epochs = 30,
                validation_data = test_set,
                validation_steps =(400/16),
```

```
                #callbacks=[tensorboard]
                );

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


%matplotlib inline
import tensorflow
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg', target_siz
e=(64, 64))
img = image.img_to_array(img1)
img = img/255
# create a batch of size 1 [N,H,W,C]
```

```
img = np.expand_dims(img, axis=0)
prediction = classifier.predict(img, batch_size=None,steps=1) #gives all class prob.
if(prediction[:,:]>0.5):
    value ='malignant :%1.2f'%(prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(alpha=0.8))
else:
    value ='benign :%1.2f'%(1.0-prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(alpha=0.8))

plt.imshow(img1)
plt.show()


import pandas as pd
test_set.reset
ytesthat = classifier.predict(test_set)
df = pd.DataFrame({
    'filename':test_set.filenames,
    'predict':ytesthat[:,0],
    'y':test_set.classes
})



pd.set_option('display.float_format', lambda x: '%.5f' % x)
df['y_pred'] = df['predict']>0.5
df.y_pred = df.y_pred.astype(int)
df.head(10)


misclassified = df[df['y']!=df['y_pred']]
```

```
print('Total misclassified image from 660 Validation images : %d'%misclassified['y'].count())


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

conf_matrix = confusion_matrix(df.y,df.y_pred)
sns.heatmap(conf_matrix,cmap="YlGnBu",annot=True,fmt='g');
plt.xlabel('predicted value')
plt.ylabel('true value');


img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg')
plt.imshow(img1);
#preprocess image
img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg', target_siz
e=(64, 64))
img = image.img_to_array(img1)
img = img/255
img = np.expand_dims(img, axis=0)



x1 = classifier.evaluate(train_set)
x2 = classifier.evaluate(test_set)


print('Training Accuracy  : %1.2f%%     Training loss  : %1.6f'%(x1[1]*100,x1[0]))
print('Validation Accuracy: %1.2f%%     Validation loss: %1.6f'%(x2[1]*100,x2[0]))
```

```
from sklearn.metrics import confusion_matrix
import pandas as pd
confusion_df = pd.DataFrame(confusion_matrix(df.y,df.y_pred),
                columns=["Predicted Class" + str(class_name) for class_name in [0,1]],
                index = ["Class" + str(class_name) for class_name in [0,1]])
print(confusion_df)


from sklearn.metrics import classification_report
print(classification_report(df.y,df.y_pred))
```

6.3 USING EFFICIENT NET

```
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB4
from keras.layers import Input, Lambda, Dense, Flatten, GlobalAveragePooling2D, MaxPooling2D,
Dropout
from keras.models import Model
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.models import Sequential
import numpy as np
from keras.models import Model
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard



from google.colab import drive
drive.mount('/content/drive')



from warnings import filterwarnings
filterwarnings('ignore')



image_input = Input(shape=(224, 224, 3))
train_set='/content/drive/MyDrive/Melanoma Dataset/data/train'
test_set='/content/drive/MyDrive/Melanoma Dataset/data/test'
model = EfficientNetB4(input_tensor=image_input, include_top=True,weights='imagenet')
model.summary()
```

```
last_layer = model.get_layer('top_activation').output
out = Dense(1, activation='relu', name='output')(last_layer)
custom_model = Model(image_input, out)
custom_model.summary()


from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

#Training Set
train_set = train_datagen.flow_from_directory(r'/content/drive/MyDrive/Melanoma Dataset/data/train',
                                              target_size=(64,64),
                                              batch_size=16,
                                              class_mode='binary')
#Validation Set
test_set = test_datagen.flow_from_directory(r'/content/drive/MyDrive/Melanoma Dataset/data/test',
                                            target_size=(64,64),
                                            batch_size = 16,
                                            class_mode='binary',
                                            shuffle=False)

classifier=Sequential()
classifier.add(Flatten())
classifier.add(Dense(units=128,activation='relu'))
classifier.add(Dense(units=1,activation='sigmoid'))
adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=Fal
se)
```

```
classifier.compile(optimizer=adam,loss='binary_crossentropy',metrics=['accuracy'])
history = classifier.fit(train_set,
                         steps_per_epoch=(2224/16),
                         epochs = 30,
                         validation_data = test_set,
                         validation_steps =(400/16),
                         #callbacks=[tensorboard]
                         );

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


%matplotlib inline
import tensorflow
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
```

```
import numpy as np
img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg', target_siz
e=(64, 64))
img = image.img_to_array(img1)
img = img/255
# create a batch of size 1 [N,H,W,C]
img = np.expand_dims(img, axis=0)
prediction = classifier.predict(img, batch_size=None,steps=1) #gives all class prob.
if(prediction[:,:]>0.5):
    value ='abnormal :%1.2f'%(prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(alpha=0.8))
else:
    value ='normal :%1.2f'%(1.0-prediction[0,0])
    plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(alpha=0.8))

plt.imshow(img1)
plt.show()



import pandas as pd
test_set.reset
ytesthat = classifier.predict_generator(test_set)
df = pd.DataFrame({
    'filename':test_set.filenames,
    'predict':ytesthat[:,0],
    'y':test_set.classes
})
pd.set_option('display.float_format', lambda x: '%.5f' % x)
df['y_pred'] = df['predict']>0.5
df.y_pred = df.y_pred.astype(int)
df.head(10)
```

```
misclassified = df[df['y']!=df['y_pred']]
print('Total misclassified image from 660 Validation images : %d'%misclassified['y'].count())



from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

conf_matrix = confusion_matrix(df.y,df.y_pred)
sns.heatmap(conf_matrix,cmap="YlGnBu",annot=True,fmt='g');
plt.xlabel('predicted value')
plt.ylabel('true value');



img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg')
plt.imshow(img1);
#preprocess image
img1 = image.load_img(r'/content/drive/MyDrive/Melanoma Dataset/data/test/benign/1.jpg', target_siz
e=(64, 64))
img = image.img_to_array(img1)
img = img/255
img = np.expand_dims(img, axis=0)

x1 = classifier.evaluate_generator(train_set)
x2 = classifier.evaluate_generator(test_set)
print('Training Accuracy  : %1.2f%%     Training loss  : %1.6f'%(x1[1]*100,x1[0]))
print('Validation Accuracy: %1.2f%%     Validation loss: %1.6f'%(x2[1]*100,x2[0]))
```

```
from sklearn.metrics import confusion_matrix
import pandas as pd
confusion_df = pd.DataFrame(confusion_matrix(df.y,df.y_pred),
                columns=["Predicted Class" + str(class_name) for class_name in [0,1]],
                index = ["Class" + str(class_name) for class_name in [0,1]])

print(confusion_df)


from sklearn.metrics import classification_report
print(classification_report(df.y,df.y_pred))
```

Chapter 7
RESULTS

7.1. RESNET 50

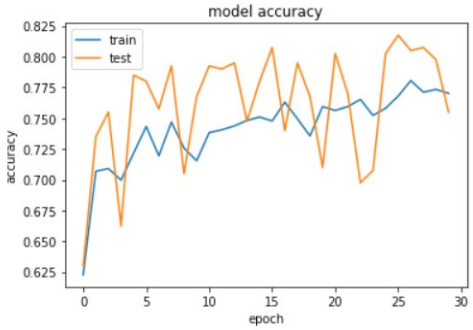

Fig.11. Model Accuracy for ResNet 50

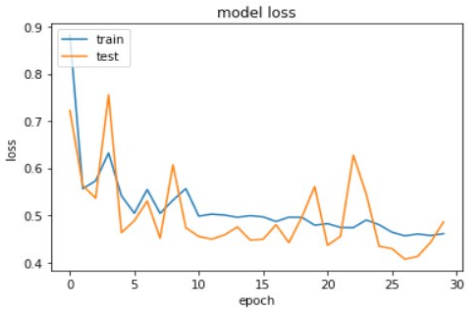

Fig.12. Model Loss for ResNet 50

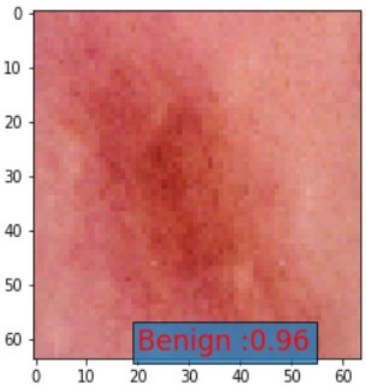

Fig.13. Prediction of ResNet50

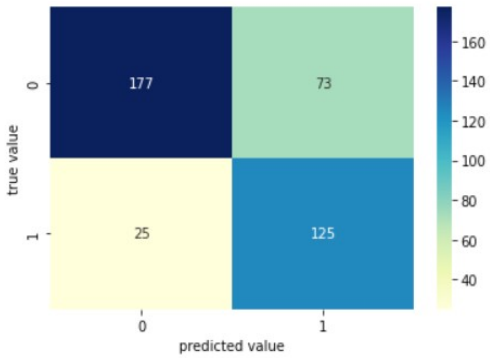

Fig.14. Confusion Matrix by ResNet 50

index	filename	predict	y	y_pred
0	benign/1.jpg	0.07396113872528076	0	0
1	benign/10.jpg	0.6508373022079468	0	1
2	benign/11.jpg	0.19748717546463013	0	0
3	benign/12.jpg	0.5899068117141724	0	1
4	benign/13.jpg	0.09980210661888123	0	0
5	benign/136.jpg	0.039054274559020996	0	0
6	benign/137.jpg	0.48087048530578613	0	0
7	benign/138.jpg	0.4539111256599426	0	0
8	benign/139.jpg	0.07626989483833313	0	0
9	benign/14.jpg	0.09345915913581848	0	0

Fig.15. Prediction Table by ResNet50

```
Training Accuracy  : 77.65%     Training loss  : 0.457778
Validation Accuracy: 75.50%     Validation loss: 0.485903
```

Fig.16. Training and Validation Accuracy by ResNet 50

	precision	recall	f1-score	support
0	0.88	0.71	0.78	250
1	0.63	0.83	0.72	150
accuracy			0.76	400
macro avg	0.75	0.77	0.75	400
weighted avg	0.78	0.76	0.76	400

Fig.17. Classification Report by ResNet 50

7.2. EFFICIENT NET

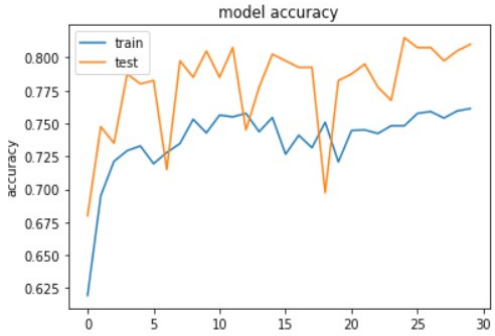

Fig.18. Model Accuracy for Efficient Net

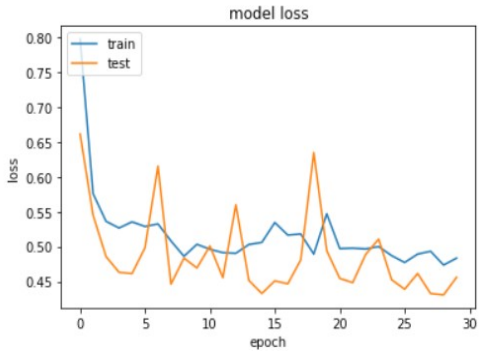

Fig.19. Model Loss for Efficient Net

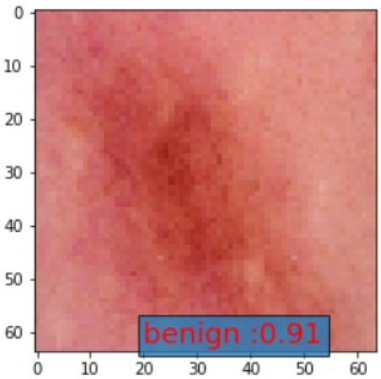

Fig.20. Prediction by Efficient Net

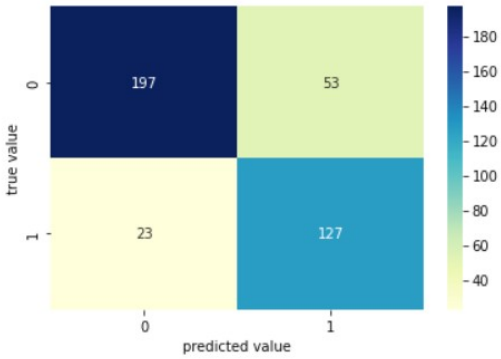

Fig.21. Confusion Matrix by Efficient Net

index	filename	predict	y	y_pred
0	benign/1.jpg	0.11050206422805786	0	0
1	benign/10.jpg	0.38663530349731445	0	0
2	benign/11.jpg	0.24666818976402283	0	0
3	benign/12.jpg	0.45066359639167786	0	0
4	benign/13.jpg	0.08523216843605042	0	0
5	benign/136.jpg	0.09321627020835876	0	0
6	benign/137.jpg	0.4248608946800232	0	0
7	benign/138.jpg	0.44206759333610535	0	0
8	benign/139.jpg	0.11612939834594727	0	0
9	benign/14.jpg	0.17706969380378723	0	0

Fig.22. Prediction Table by Efficient Net

```
Training Accuracy  : 77.38%     Training loss  : 0.465619
Validation Accuracy: 81.00%     Validation loss: 0.456058
```

Fig.23. Training and Validation Accuracy by Efficient Net

	precision	recall	f1-score	support
0	0.90	0.79	0.84	250
1	0.71	0.85	0.77	150
accuracy			0.81	400
macro avg	0.80	0.82	0.80	400
weighted avg	0.82	0.81	0.81	400

Fig.24. Classification report by Efficient Net

7.3 XCEPTION NET

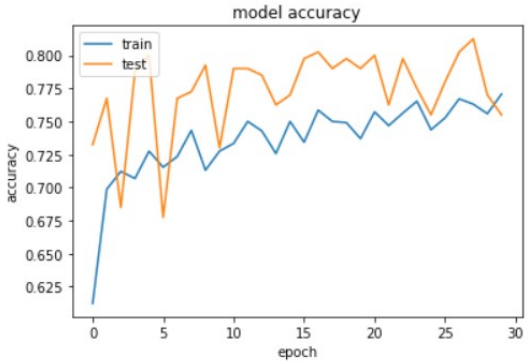

Fig.25. Model Accuracy of Xception Net

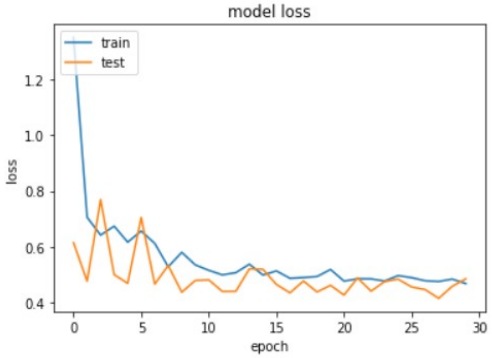

Fig.26. Model Loss of Xception Net

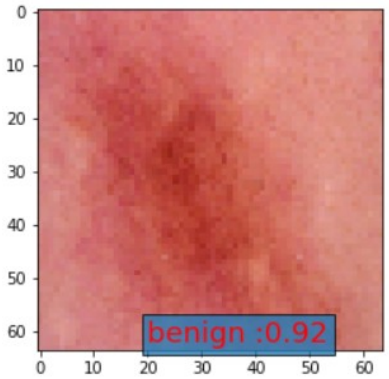

Fig.27. Prediction by Xception Net

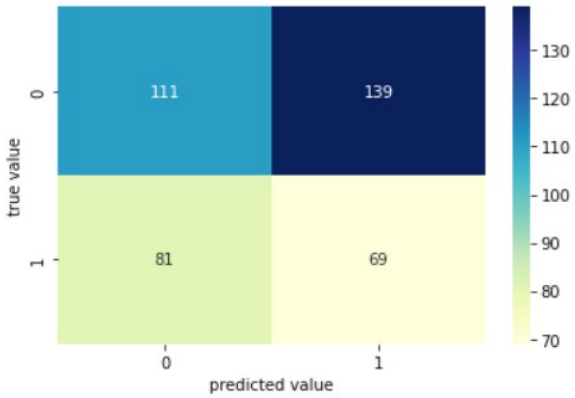

Fig.28. Confusion Matrix by Xception Net

index	filename	predict	y	y_pred
0	benign/1.jpg	0.24836382269859314	0	0
1	benign/10.jpg	0.9140493869781494	0	1
2	benign/11.jpg	0.07759043574333191	0	0
3	benign/12.jpg	0.6751601696014404	0	1
4	benign/13.jpg	0.8914955854415894	0	1
5	benign/136.jpg	0.06472885608673096	0	0
6	benign/137.jpg	0.5935950875282288	0	1
7	benign/138.jpg	0.4449429512023926	0	0
8	benign/139.jpg	0.16545188426971436	0	0
9	benign/14.jpg	0.8065744042396545	0	1

Fig.29. Prediction Table by Xception Net

```
Training Accuracy  : 77.28%     Training loss  : 0.441629
Validation Accuracy: 76.97%     Validation loss: 0.432598
```

Fig.30. Training and Validation Accuracy by Xception Net

	precision	recall	f1-score	support
0	0.81	0.76	0.78	360
1	0.73	0.78	0.76	300
accuracy			0.77	660
macro avg	0.77	0.77	0.77	660
weighted avg	0.77	0.77	0.77	660

Fig.31. Classification Report by Xception Net

REFERENCES

[1] F. Dimitriou et al., "The World of Melanoma: Epidemiologic, Genetic, and Anatomic Differences of Melanoma Across the Globe," Current Oncology Reports, vol. 20, no. 11, 2018, doi: 10.1007/s11912-018-0732-8.

[2] D. Schadendorf *et al.*, "Melanoma," *The Lancet*, vol. 392, no. 10151, pp. 971–984, 2018, doi: 10.1016/S0140-6736(18)31559-9.

[3] E. Goodarzi *et al.*, "Epidemiology, incidence and mortality of thyroid cancer and their relationship with the human development index in the world: An ecology study in 2018," *Advances in Human Biology*, vol. 9, no. 2, p. 162, 2019, doi: 10.4103/aihb.aihb_2_19.

[4] U. O. Dorj, K. K. Lee, J. Y. Choi, and M. Lee, "The skin cancer classification using deep convolutional neural network," *Multimedia Tools and Applications*, vol. 77, no. 8, pp. 9909–9924, 2018, doi: 10.1007/s11042-018-5714-1.

[5] H. A. Haenssle *et al.*, "Man against Machine: Diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists," *Annals of Oncology*, vol. 29, no. 8, pp. 1836–1842, 2018, doi: 10.1093/annonc/mdy166.

[6] R. Marks, "Epidemiology of melanoma," *Clinical and Experimental Dermatology*, vol. 25, no. 6, pp. 459–463, 2000, doi: 10.1046/j.1365-2230.2000.00693.x.

[7] D. Bisla, A. Choromanska, R. S. Berman, J. A. Stein, and D. Polsky, "Towards automated melanoma detection with deep learning: Data purification and augmentation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2019-June, pp. 2720–2728, 2019, doi: 10.1109/CVPRW.2019.00330.

[8] Y. Li and L. Shen, "Skin lesion analysis towards melanoma detection using deep learning network," *Sensors (Switzerland)*, vol. 18, no. 2, pp. 1–16, 2018, doi: 10.3390/s18020556.

[9] S. Hosseinzadeh Kassani and P. Hosseinzadeh Kassani, "A comparative study of deep learning architectures on melanoma detection," *Tissue and Cell*, vol. 58, no. April, pp. 76–83, 2019, doi: 10.1016/j.tice.2019.04.009.

[10] E. Nasr-Esfahani *et al.*, "Melanoma detection by analysis of clinical images using convolutional neural network," *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, vol. 2016-Octob, pp. 1373–1376, 2016, doi: 10.1109/EMBC.2016.7590963.

[11] A. A. Adegun and S. Viriri, "Deep learning-based system for automatic melanoma detection," *IEEE Access*, vol. 8, pp. 7160–7172, 2020, doi: 10.1109/ACCESS.2019.2962812.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, 2015.

[13] P. Mirunalini, A. Chandrabose, V. Gokul, and S. M. Jaisakthi, "Deep Learning for Skin Lesion Classification," no. March, 2017, [Online]. Available: http://arxiv.org/abs/1703.04364

[14] M. F. Jojoa Acosta, L. Y. Caballero Tovar, M. B. Garcia-Zapirain, and W. S. Percybrooks, "Melanoma diagnosis using deep learning techniques on dermatoscopic images," *BMC Medical Imaging*, vol. 21, no. 1, pp. 1–11, 2021, doi: 10.1186/s12880-020-00534-8.

[15] M. A. Kadampur and S. al Riyaee, "Skin cancer detection: Applying a deep learning based

model driven architecture in the cloud for classifying dermal cell images," *Informatics in Medicine Unlocked*, vol. 18, no. August 2019, p. 100282, 2020, doi: 10.1016/j.imu.2019.100282.
[16] M. Ramachandro, T. Daniya, and B. Saritha, "Skin Cancer Detection Using Machine Learning Algorithms," *3rd IEEE International Virtual Conference on Innovations in Power and Advanced Computing Technologies, i-PACT 2021*, 2021, doi: 10.1109/i-PACT52855.2021.9696874.
[17] S. Wang and M. Hamian, "Skin Cancer Detection Based on Extreme Learning Machine and a Developed Version of Thermal Exchange Optimization," *Computational Intelligence and Neuroscience*, vol. 2021, 2021, doi: 10.1155/2021/9528664.
[18] M. Dildar *et al.*, "Skin cancer detection: A review using deep learning techniques," *International Journal of Environmental Research and Public Health*, vol. 18, no. 10, 2021, doi: 10.3390/ijerph18105479.
[19] M. B. Alazzam, F. Alassery, and A. Almulihi, "Diagnosis of Melanoma Using Deep Learning," *Mathematical Problems in Engineering*, vol. 2021, 2021, doi: 10.1155/2021/1423605.