Detect Lung Cancer through CT Scan images using deep learning.

A Thesis presented in partial fulfilment of the requirements for the degree of

Bachelor of Science (Hons) in

Information Technology and Management

at Information Technology Unit, Faculty of Science, University of Colombo, Sri Lanka.

Piriyanka Periyathamby

2023

Declaration

I hereby declare that the work reported in this research report was exclusively carried out by me under the supervision of Dr. Uditha Prabhath Liyanage. It describes the results of my own independent research except where due reference has been made in the text. No part of this research report has been submitted earlier or concurrently for the same or any other degree.

Candidate:
Piriyanka Periyathamby
Reg. No: 2018s17107
Index No: s 14569

Date:…… Signature:……

Certification of Approval

We hereby declare that this research report is from the student's own, and effort and all other sources of information used have been acknowledged. This report has been submitted with our approval.

Supervisor:
Dr. U.P. Liyanage Senior Lecturer,
Department of Statistics & Computer Science,
University of Kelaniya.

Date:…… Signature: ……

Coordinator:
Dr. S.K.P. Eranga
IT Major Theme Coordinator,
Department of Mathematics,
University of Colombo.

Date:…… Signature: ……

ACKNOWLDGEMENT

In preparation of my assignment, I had to take the help and guidance of some respected persons, who deserve my deepest gratitude. As the completion of this assignment gave me much pleasure, I would like to show my gratitude to my project supervisor for he helps, simulating suggestions, supervision and giving me a good guideline for assignment throughout numerous consultations, to continue my project successfully. Also, I would like to express my sincere gratitude to my project coordinator of the Individual Project for his immense support in the project proposal phase and for his wise project coordination. I would like to pay my gratitude for giving comments with valuable ideas to improve my project.

Many people, especially my all my colleagues and all those who have not been mentioned specifically, but who has played a supportive role during this study for lending a helpful hand in various ways have made valuable comments and suggestions on my paper which gave me an inspiration to improve the quality of the assignment.

Most importantly this would not have been possible without the love and patience of my parents. They have been a constant source of love, concern, support, and strength throughout all these years. I am grateful to them for placing their trust on me and cheering me patiently. It was very helpful during the critical period of my academic life which motivated me to successfully complete this research.

Table of Contents

List of Figures

ABSTRACT

A malignant lung tumor called lung cancer is distinguished by unchecked cell proliferation in the lung tissues. Lung cancer patients' chances of survival can be increased, and many lives can be saved with early detection. Due to the structure of cancer cells, diagnosing cancer is one of the most difficult tasks for radiologists, hence using a computer-aided diagnosis method can be helpful.

As a result, lung cancer early detection and prediction should be crucial to the diagnosis process and increase patient survival rates. In order to improve accuracy, this project shows how to detect lung cancer based on CT scans by effectively classifying and predicting lung cancer using deep learning models.

The architecture is trained using the pre-processed CT images. The patient input images are then put to the test using deep learning models. This study's primary goal is to find out if a patient's lung contains a cancerous tumor. By creating a manual CNN model and using transfer learning-based VGG-16 and Inception V3 architectures to train the model to increase accuracy, we propose a convolutional neural network-based classification technique. The VGG-16 model achieved accuracy of 94.08%, and the Inception V3 model achieved accuracy of 92.64%. Regarding the lung cancer detection study using CT scan images, the reported test accuracy of 89.00% suggests that the CNN model approach used in this study was able to accurately detect the presence or absence of lung cancer in the test images.

These models' performances are compared in this study. For best results, hyperparameter tuning was done. To allow for user interaction, a web application is created that allows users to upload CT scan images. The model created analyzes the uploaded image, predicts the presence of lung cancer, and displays the prediction on the website.

ABBREVATION

CNN	Convolutional neural network
CT	Computed tomography
MRI	Magnetic Resonance Imaging
AI	Artificial Intelligence
API	Application programming interface
CAD	Computer-assisted diagnostic
TP	True positive
TN	True negative
FN	False negative
FP	False positive

Chapter 1

INTRODUCTION

1.1 General Background

Lung cancer is a disease that occurs due to the uncontrolled growth of cells. The most common cancer, lung cancer has a high mortality rate and has a significant negative impact on the patient. Overall, the lungs' job is to assist with breathing, which entails drawing air from the atmosphere and supplying the bloodstream with oxygen. The outer and inner layers of each lung are separated by a sac-like structure called the pleural sac, which houses a fluid known as the pleural fluid. Since lung cells have the potential to mutate, they are unable to carry out the same functions as a normal, healthy cell. Tumors develop as a result of this. Small cell lung carcinoma (SCLC) and non-small cell lung carcinoma are the two main types of cancerous tumors (NSCLC)

One of the most fatal diseases, lung cancer has a death rate of 19.4%. Early lung tumor detection is possible using a variety of imaging techniques, including computed tomography (CT), sputum cytology, chest X-rays, and magnetic resonance imaging (MRI). Tumors are categorized into two categories during detection: I non-cancerous (benign) and (ii) cancerous tumors (malignant). The leading cause of lung cancer is cigarette smoking, but there are other risk factors as well, including air pollution and excessive alcohol consumption. The symptoms of lung cancer that has spread to the brain include vision problems and weakness on one side of the body.

Chest pain, shortness of breath, coughing up blood, and these are all symptoms of primary lung cancer. Unfortunately, most cancer diagnoses are made in the later stages of the disease, largely because there aren't any early-stage symptoms. When cancer is discovered in its early stages, however, the chance of survival increases.

The manual examination and diagnosis method can be greatly expanded with the introduction of image manipulation techniques. Many experts are working to find early signs of cancer using

computer learning techniques. The neural network plays a crucial role in the detection of cancer cells among normal tissue, which in turn serves as a crucial tool for developing an assistive AI-based cancer diagnosis. Here, we demonstrate how to distinguish between benign and malignant lung tumors with the aid of Convolutional Neural Network (CNN) and transfer learning models. Deep learning and machine learning algorithms were traditionally designed to work with a specific feature-space distribution. When the feature-space distribution changes, models must be completely redesigned, and collecting the necessary training data takes time.

Deep learning models consequently need a lot of labelled data when they are being trained. As a result, it is nearly impossible to build a machine learning-based model for a tar- get domain with little labelled data for supervised learning. Transfer learning would greatly improve learning efficiency and accuracy. The dataset for our project is the IQ-OTHNCCD lung cancer dataset.

We design a manual CNN model and suggest a classification method based on convolution neural networks. Additionally, the model was trained using our dataset using the VGG-16 and Inception V3 architectures with some fine-tuning. Pre-trained networks for Image-Net are primarily trained using real-world images. You have to fine-tune your network using a process called transfer learning because there is such a big difference between natural images and medical images like CT/MRI. The VGG-16 and Inception V3 architectures were used with some fine-tuning to train the model with our dataset in a comparative study of these models. Pre-trained Image-Net networks are primarily trained using real-world images. You need to fine-tune your network, a process known as transfer learning, because there is a significant difference between natural images and medical images like CT/MRI. Successful lung cancer screenings have the potential to save a great number of lives, making them very important. Examining cancer screenings by hand can be very time- and money-consuming, and it is not always error-free because classifying tumors calls for a high level of radiological expertise. Therefore, deep learning can aid in improving screening quality while lowering associated costs.

1.2 Objective & Scope

Our project's primary goal is to find any lung cancer tumors that may be present in a patient. To achieve the best performance, numerous deep learning models were developed. to User-centered design should be a component of the project. Users in this project would include radiologists, doctors, or any other medical personnel involved in the diagnosis of lung cancer. Web applications are created to allow for user interaction. The user must upload a CT scan to the application. The application pre-processes the CT scan and infers the image to the predictive model after the user selects which scan, they wish to predict. After that, the user sees the model's output. The system must be able to provide information that the user can understand and use to their advantage.

The diagnosis of lung cancer is extremely difficult, but with early detection, a patient's chance of survival is very high. A more effective deep learning model would be able to overcome these difficulties and contribute to a decrease in the death rate from lung cancer because, depending on the patient's stage, only 3 scans out of 200–300 images would reveal cancer.

Chapter 2

LITERATURE SURVEY

The best methods for diagnosing and identifying the disease by utilizing image recognition, machine learning, and deep learning have been proposed for a number of lung cancer diagnosis and diagnostic systems.

We looked at a number of lung cancer diagnosis systems that have been put forth to assist radiologists and clinicians in detecting and classifying the condition with the best possible outcomes by utilizing various image processing, machine learning, and deep learning techniques.

2.1 Image processing approaches

Image processing technique [1] has a great role in medical image analysis in classifying accurately. Image processing techniques Used in nodule classification for lung CT images. Numerous studies adopted segmentation, morphological operations, and contour filter approaches for better nodule detection.

2.2 Machine learning approaches

Machine learning is the study of techniques that enable computers to recognize patterns in data and use those patterns to solve problems. The objective is to develop mathematical models that, when given input data, can be trained to generate useful outputs. Training data is given to machine learning models, which are then tuned by an optimization algorithm to produce precise predictions for the training data. Medical image processing, computer-aided diagnosis, image interpretation, image registration, image segmentation, image retrieval, and image analysis are just a few applications where machine learning techniques have recently played a significant role. These methods utilize standard learning-free algorithms like Support Vector Machines (SVM), Neural Networks (NN), and KNN, etc.

2.3 Deep learning approaches

Machine learning algorithms have limitations when it comes to processing natural images, requiring time and expert knowledge for feature tuning. Deep learning approaches are becoming more popular due to their ability to handle raw data, automatically learn features, and work quickly. Deep learning has shown promising results in a variety of fields such as speech recognition, text recognition, lip reading, computer-aided diagnosis, face recognition, and drug discovery. In medical image analysis, deep learning algorithms are especially useful due to their ability to automatically extract features from large datasets and represent multiple levels of abstraction.

In the research paper[2] by Ruchita Tekade titled "Lung Nodule Detection and Classification Using Machine Learning Techniques" published in 2018, various image pre-processing methods were discussed to detect regions of interest (ROI) in lung CT scan images that

represent lung nodules. These pre-processing methods included thresholding, clearing borders, and morphological operations such as erosion, closing, and opening. The paper also explored the use of machine learning techniques, such as Support Vector Machine (SVM) and Convolutional Neural Network (CNN), to classify lung nodules and non-nodules objects in patient lung CT scan images using sets of lung nodule regions. The study reported an SVM sensitivity of 90% and a CNN accuracy of 96.6%.

Swathi Velugoti 1, Revuri Harshini Reddy 2, Sadiya Tarannum 3, Sama Tharun Kumar Reddy UG Students, Department of Computer Science and Engineering,Guru Nanak Institutions Technical Campus, Hyderabad – Telangana , India This paper [1] focuses on the early detection of lung cancer, as prevention is not currently possible due to the unclear cause of the disease. The proposed solution uses image processing and machine learning techniques to classify the presence of lung cancer in CT images and blood samples. The study shows that CT scan reports are more effective than mammography, and therefore, patient CT scan images are categorized as either normal or abnormal. Abnormal images are then segmented to focus on the tumor portion and features are extracted from these images for classification using Support Vector Machines (SVM) and image processing techniques. The proposed method achieved a CNN accuracy of 93.0%, indicating successful detection of lung cancer and its stages, and aiming to provide more accurate results.

Suren Makajua , P.W.C. Prasad*a, Abeer Alsadoona , A. K. Singhb , A. Elchouemi In the research paper titled [3], the authors explored various computer-aided techniques that use image processing and machine learning. The main objective of this study was to assess and compare these techniques, identify the best performing method, and propose a new model with improvements. The authors ranked lung cancer detection techniques based on their detection accuracy and analyzed them at each step, highlighting their limitations and drawbacks. While some techniques had low accuracy, others had higher accuracy but did not reach 100%. The authors aimed to increase accuracy towards 100%, and found that SVM methods provided an accuracy of 86.66%.

Rotem et al, Christian Jacob, and Jorg Denzinger [4] proposed a Lung Nodule Detection system in CT Images using Deep Convolutional Neural Networks. They utilized the publicly available Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI)

database which contains CT scan images with various shapes and sizes. The system employs a deep Convolutional Neural Network (CNN) that is trained using the back-propagation algorithm to extract lung nodules in subvolumes of CT images. The proposed system achieves a sensitivity of 78.9% (True positive rate) with only 20 False Positives (FPs) per scan. Notably, the system accomplishes these results without any segmentation methods or FPS reduction process. However, the drawback of the system is its lower sensitivity compared to other models.

In a paper[8] by Taolin Jin, Hui Cui, and colleagues, the focus was on early detection of lung cancer using deep spatial lung characteristics. To achieve this, they developed a CNN architecture by enhancing the original 2D architecture of CNN. The researchers first segmented the 2D CT scans and then transformed them into 3D visualizations to reveal the segmented volume of the lung. Through analysis of the segmented CT scans, the CNN was utilized to differentiate between cancerous and non-cancerous scans.

It can be inferred that machine learning algorithms are resource-intensive, have computational constraints, and demand significant effort for feature engineering. Deep learning has proven to be the most effective method for feature extraction, object classification, and medical imaging. Transfer learning is a useful approach for reducing processing time and achieving superior accuracy.

Chapter 3

DESIGN AND IMPEMENTATION

3.1 Design

3.1.1 Algorithm

• **Step 1**: Collect the dataset.

• **Step 2**: Split the dataset to train and test folders.

• **Step 3**: Loading dataset and image
    - Data normalization
        * Resizing images to 224*224.
• **Step 4**: Image Augmentation by image Data Generator library.
    - Scaling, Shearing, Zooming, flipping and rescale.
    - Rotation, width height and channel shifting.
    - Changing Brightness range
• **Step 5**: Model Building
    - Import all Libraries.
    - Initializing the model
• **Step 6**: In case of Implementing Transfer Learning
    - Import and Load the Pre-Trained model.
    - Resize according to model used.
    - Freeze all layers in the base model to stop updating.
    - Add layers on top of the output of layers from the base model.
    - Changing output Layer and Re-Training new data
• **Step 7**: In case of Implementing designed CNN Architecture
    - Adding Convolutional Layers and Pooling layers
    - Adding fully Connected Layers
    - Adding Output Layer
    - Configuring and compiling
    - Optimizing model
        * Hyperparameter Tuning
    - Training and Evaluation of model
    - Saving the Model and Prediction

• **Step 8**: Building UI Webpage for prediction

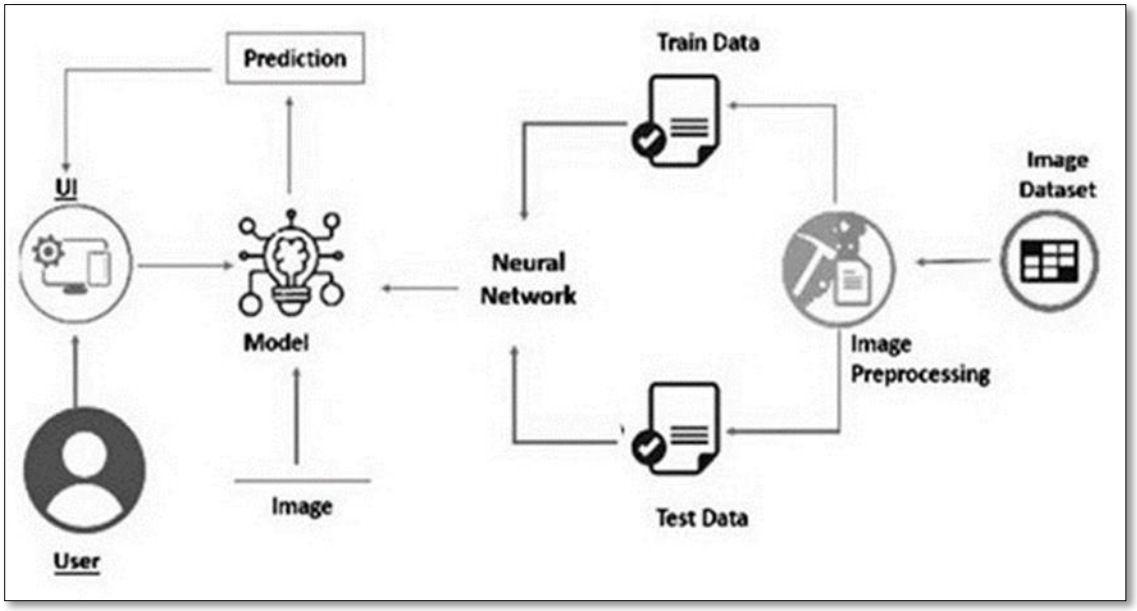


Figure 3.1: Block Diagram

3.1.2 Packages Used

Tensor flow: TensorFlow is an open-source platform for machine learning that provides a comprehensive set of flexible tools, libraries, and community resources. Developed by Google, TensorFlow is widely used for research on machine learning and deep learning. It serves as a backend for Keras, a high-level deep learning API, allowing for efficient and easy-to-use neural network development.

Scikit-learn: Scikit-learn, also referred to as sklearn, was utilized to plot the confusion matrix. It was specifically designed to work alongside other Python numerical and scientific libraries like NumPy and SciPy. Additionally, Scikit-learn seamlessly integrates with various other Python libraries such as Matplotlib and plotly for plotting. It is a Python module built on top of SciPy, intended for machine learning purposes.

PIL: Python Imaging Library (PIL) is a free and open-source library that extends the Python programming language by providing support for various image file formats, allowing for the manipulation, opening, and saving of images.

Keras: Keras is a high-level deep learning API that simplifies the code in other frameworks such as TensorFlow and Theano. It is written in Python and runs on top of the TensorFlow machine learning platform. Keras is designed to be easy and intuitive to use due to its high-level nature. It utilizes various optimization techniques to make the high-level neural network API more performant and user-friendly. Keras is highly scalable, supports multiple platforms and backends, and is a powerful tool for deep learning tasks.

NumPy: NumPy is a Python library that enables efficient numerical computing, particularly for mathematical tasks. In the project, NumPy was extensively used for data preprocessing and preparation, thanks to its highly efficient n-dimensional array (ND array). NumPy can also perform mathematical operations more efficiently compared to the math library in Python.

Matplotlib: Matplotlib is a plotting library for Python that simplifies the creation of a wide range of graphs and visualizations for programmers. One of its great features is its seamless integration with Python, making it easy to create visualizations.

3.2 Data Collection

Information is crucial to artificial intelligence because a computer cannot learn without it. It is the element that enables the teaching of algorithms the most. We require the collection of training and test image data for convolutional neural networks. To validate our model, two directories one for training images and the other for testing images will be created.

The IQ-OTH/NCCD lung cancer dataset collection evaluates the variability of tumor unidimensional, bidimensional, and volumetric measurements on com putted tomographic (CT) scans in patients with non–small cell lung cancer. It a web-accessible international resource for development, training, and evaluation of computer-assisted diagnostic (CAD) methods for lung cancer diagnosis. IQ- OTH/NCCD slides were marked by oncologists and radiologists in these two centers. The IQ-OTH/NCCD lung cancer dataset The Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases (IQ-OTH/NCCD) lung cancer dataset was collected in 2019. The dataset contains a total of 1900 images representing CT scans slices

of 110 cases. Each scan contains several slices. The number of these slices range from 80 to 200 slices, each of them represents an image of the human chest with different sides and angles. In our project, we took 1800 images and grouped them into two classes: cancer (Malignant) and non-cancer (Benign cases). The CT scans were originally collected in DICOM format. All images were de-identified before performing analysis.

3.3 Implementation

3.3.1 Image Pre-processing

The aim of pre-processing is an improvement of the image data that suppresses un- desired distortions or enhances some image features relevant for further processing and analysis tasks. Images from the dataset are accessible in DICOM format. Medical imaging datasets now follow the DICOM (Digital Imaging and Communications in Medicine) standard. Images that need to be classified are converted into the.jpg format and given the same labels. Therefore, the dataset's images are first converted and saved in the jpeg format. The transformed images are organized into two distinct directories, Cancer (Malignant) and Non-Cancer (Benign). Using CNN, significant features from the image are extracted. All image processing is done implicitly by the CNN classifier. These images are given as input to CNN for classification of image as cancerous and noncancerous.

Normalization

The range of values for pixel intensity is modified through the normalization process. In the dataset, there will be a variety of sizes for the images. This again puts the classification accuracy in threat, which must be avoided. Data normalization involves converting every image to a uniform pixel size. For the CNN and VGG16 models, the images in the dataset are resized to the same dimensions of 224x224 pixels, and 299x299 pixels for the Inception model.

Image Augmentation

Image augmentation are manipulations applied to images to create many altered versions of the same image in order to expose the model to a wider array of training examples. For example, randomly altering rotation, brightness, or scaling of an input image so that a model considers what an image subject looks like in a variety of situations. It helps to expose our classifier to a

wider variety of situations so as to make our classifier more robust. Augmentation is a technique largely used to enhance the training, to increase testing accuracy of convolutional neural networks and boost the performance of deep networks. Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc.

Horizontal Flip Augmentation

An image flip means reversing the rows or columns of pixels in the case of a vertical or horizontal flip respectively.

Rotation

Rotates the image by a specified degree. here we rotate in range of 8 degree.

Horizontal Shift

A horizontal shift augmentation means pixels of the image will shift horizontally without changing the dimension of the image. Some pixels of the image will be clipped off and there will be a new area where new pixels of the image will be added to get the same image dimension.

The Keras Image Data Generator class has two main arguments width shift range and height shift range and based on these values the image pixels will move either to the left-right and top-bottom.

Brightness Range

Used to change brightness of light in image. Below value 1 it will start darkening the image. And if you go above to 1 it will start brightening the image.

Channel Shift

Channel shift randomly shifts the channel values by a random value chosen from the range specified by channel shift range. For our project we chose 0.15 as channel shift range.

Random Zoom Augmentation

A zoom augmentation randomly zooms the image in and either adds new pixel values around the image or interpolates pixel values respectively. Image zooming can be configured by the zoom range argument to the Image Data Generator constructor. For our project we chose 0.15 as Zoom range.

Shearing

Shearing is also used to transform the orientation of the image. Shear transformation slants the shape of the image. This is different from rotation in the sense that in shear transformation, we fix one axis and stretch the image at a certain angle known as the shear angle. For our project we chose shear range=0.10

Fill mode.

There are several options which can be used to fill these regions.

• Nearest: This is the default option where the closest pixel value is chosen and repeated for all the empty values.

• Reflect: This mode creates a ‘reflection’ and fills the empty values in a reverse order of the known values.

• Wrap: Wrap effect copies the values of the known points into the unknown points, keeping the order unchanged.

Image Data Generator

The Image Data Generator API in Keras makes it simple to create an augmented image generator. A real-time data augmentation tool called Image Data Generator creates batches of image data. To easily perform image augmentation, use Kera's Image Data Generator class.

- Import the Image Data Generator Library.
- Image Data Generator Class configuration.
- Use Image Data Generator's pre-processing tools to prepare the trainset and test set.

3.3.2 Loading the dataset.

The Google Colab maintains the entire image datasets in a directory. The images used for practice and testing can be found in the directory. The images for each instance are named in the directory by the patient's id. JPEG versions of the IQ-OTHNCCD images are available.

Two separate directories, Cancer (Malignant) and Non-Cancer (Benign), contain the stacked images. Then, using the Image Data Generator function, images are loaded.

3.3.3 Model Building

Convolutional neural networks (CNNs, also known as ConvNets) are a subset of deep neural networks that are most frequently used to analyze visual data. As a result of their shared-weights architecture and translation invariance properties, they are also referred to as shift invariant or space invariant artificial neural networks (SIANN). They have uses in natural language processing, financial time series, recommender systems, image classification, medical image analysis, and image and video recognition.

Convolutional Neural Networks (CNN) have the added benefit of requiring fewer computations while also making it easier to learn a feature regardless of where it is in the image. Convolutional and pooling layers make up a deep convolutional neural network (DCNN), which are typically switched between. Despite being referred to as convolutions informally, the layers are actually just layers. It is technically a sliding dot product or cross-correlation in mathematics. This matters because it influences how weight is determined at a particular index point, which has implications for the indices in the matrix.

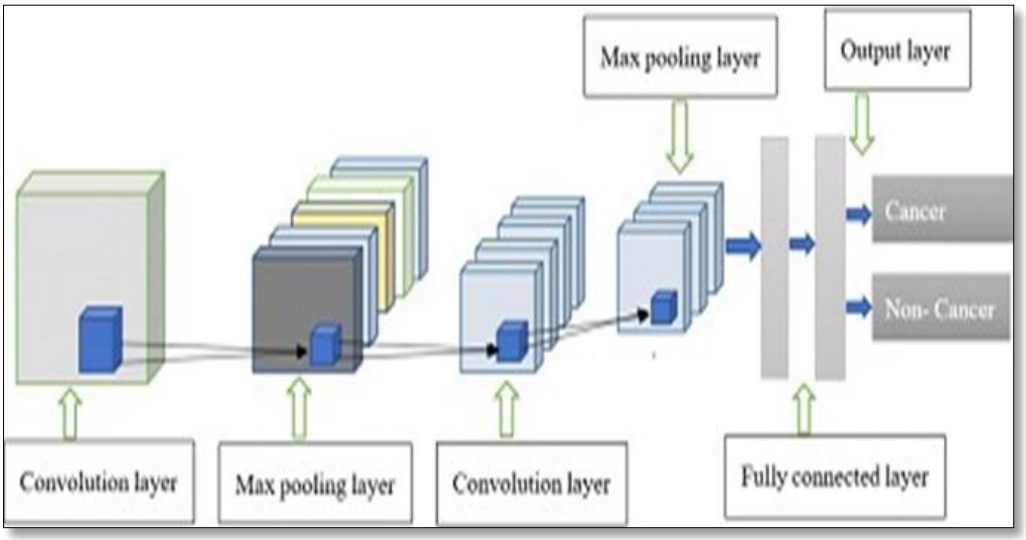


Figure 3.2: CNN Architecture

Architecture

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a ReLU layer , and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

Convolutional Layer

When programming a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a neural network should have the following attributes:

•Convolutional kernels defined by a width and height (hyper-parameters).

•The number of input channels and output channels (hyper-parameter).

•The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable.

Add Convolutional Layer

The convolutional layer is the first and core layer of CNN. It is one of the building blocks of a CNN and is used for extracting important features from the image. In the Convolution operation, the input image will be convolved with the feature detector/filters to get a feature

map. The important role of the feature detector is to extract the features from the image. The group of feature maps is called a feature layer.

In the convolution2D function, we have given arguments we are applying 32 filters of 3x3 matrix are applied, and it does so by performing 2d convolutional operations the input image will be convolved with filters that is we multiply each values of kernel matrix with the input image matrix and then sum it to perform convolution. So, the sum of the element-wise multiplication gives the result for that window.

Activation function defines the output of input or set of inputs or in other terms defines node of the output of node that is given in inputs. They basically decide to deactivate neurons or activate them to get the desired output. Activation function defines node of the output of node that is given in inputs. It decides to deactivate neurons or activate them to get the desired output. ReLU stands for rectified linear unit and is a type of activation function. Mathematically, it is defined as y = max(0, x). It is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better.

Add max pooling layer.

Pooling reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer and number of parameters used in our model. It keeps only the necessary details. Pooling is a technique in CNN that helps us to avoid overfitting of data, spatial invariance, and distortion.

After applying max pooling, we will get another feature map called Pooled Feature Map. The primary operation of Max pooling is to down sample the feature maps. It is an effective way to make the models efficient and increase their predictive power.

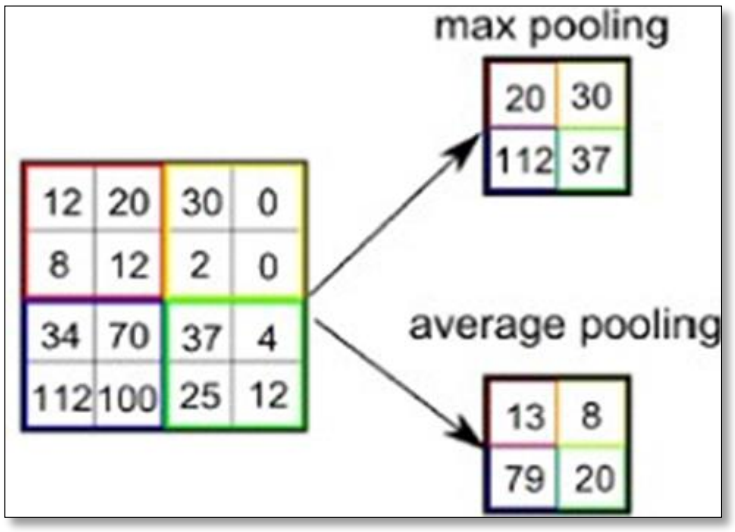


Figure 3.3: Max Pooling Layer

Add Flattening layer.

Now the pooled feature map from the pooling layer will be converted into one single dimension matrix or map, flattening layer converts the multi-dimension matrix to one single dimension layer. Flattens the input so we can introduce a standard Dense layer that will lead us to a single result layer. Before this operation, we have three-dimensional data of width, height, and color of each pixel of the image.

Add Fully connected Layers.

The fundamental goal of a fully connected layer is to take the results of the con- volution and pooling processes and use them to classify the image into a label.

Adding Dense layers

This step is to add a dense layer. Each neuron in a layer receives input from all the neurons present in the previous layer. We flatten the feature map and convert it into a vector or single dimensional array in the Flatten layer. This vector array is fed it as an input to the neural network and applies an activation function, such as sigmoid or other, and returns the output.

Dropout

Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time. This helps prevent overfitting.

3.3.4 Designed CNN Architecture

We first created a 2-Dimensional CNN model and then imported model building libraries. Sequential () function is used to start building models. The input layer receives input images from the dataset and resizes them to size 224*224 before sending them to additional layers. Our network has 12 layers total, 5 of which are convolutional, and 3 of which are used for the maximum pooling operation. Due to its computing efficiency, a ReLu activation function is applied after each convolutional layer. Added is a flatten layer. The addition of a fully connected layer, which connects the current layer to the layer below it, is followed by a sigmoid activation function. Dropout layers are also added to lessen the number of connections between neurons.

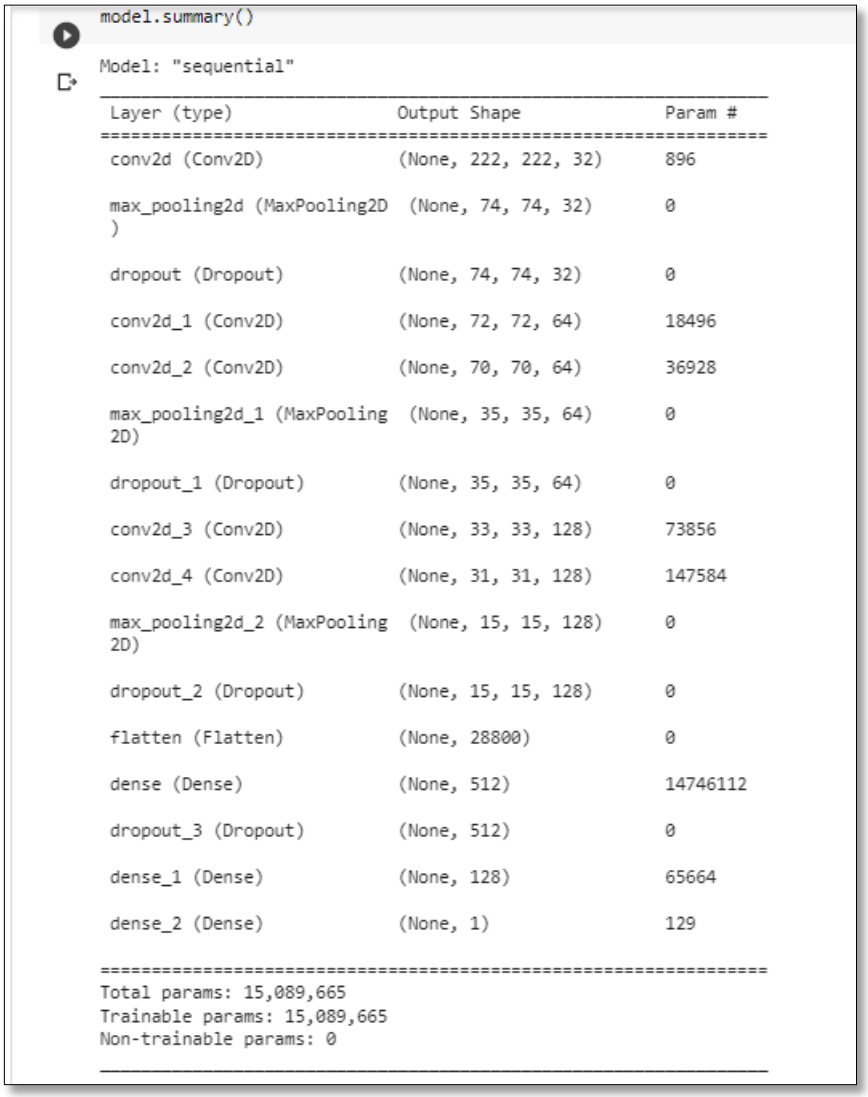
```
model = Sequential()
model.add(Convolution2D (32,(3,3),input_shape=(224,224,3),activation='relu'))
model.add(MaxPooling2D (pool_size=(3,3)))
model.add(Dropout(0.25))
model.add(Convolution2D(64,(3,3),activation='relu'))
model.add(Convolution2D(64,(3,3),activation='relu'))


model.add(MaxPooling2D (pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Convolution2D(128,(3,3),activation='relu'))
model.add(Convolution2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D (pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(units=512,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=128,activation='relu'))
model.add(Dense(1,activation="sigmoid"))

model.summary()
```

Figure 3.4: CNN Layer Architecture added with detailed.



```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 dropout (Dropout)           (None, 74, 74, 32)        0

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 conv2d_2 (Conv2D)           (None, 70, 70, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 35, 35, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 35, 35, 64)        0

 conv2d_3 (Conv2D)           (None, 33, 33, 128)       73856

 conv2d_4 (Conv2D)           (None, 31, 31, 128)       147584

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 128)      0
 2D)

 dropout_2 (Dropout)         (None, 15, 15, 128)       0

 flatten (Flatten)           (None, 28800)             0

 dense (Dense)               (None, 512)               14746112

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 128)               65664

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 15,089,665
Trainable params: 15,089,665
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.5: CNN model summary

3.3.5 Configuring and compiling.

Using Compile () method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics. If there are 2 classes in the output, then loss is binary cross entropy.

3.3.6 Model for Training

A crucial step in developing a deep learning model is model training. Only a trained model will give us the desired results with the highest level of accuracy. Model training is a process for giving the neural networks the right weights for their inputs so that the results they produce after going through each layer are accurate. Send the data to the model so that it can finish training with the training data. Run model. Fit for a specified number of epochs using the training and validation dataset.

epochs: no: of times the model should get trained.

Steps per epoch = number of training images / batch size

validation steps = no of test images / 32

Optimizing and Saving model

Check the predicted value and accuracy. Save trained built model to view prediction and integrate it with UI web application.

3.4 Transfer learning

Transfer learning makes use of previously learned information from a trained model in subsequent tasks and uses it to enhance generalization with new data. The idea is to use a model that has already been successfully trained over a long period of time using a larger dataset. Transfer learning is the process of applying previously learned knowledge and skills to new tasks in order to solve problems. Due to the lack of data, this method is essential in medical image processing.

Transfer learning, a technique for streamlining the training process with previously trained models in a different task, also produces positive results for small datasets and cuts down on time. A pre-trained model is applied to a new task as a foundation. By sifting through a large dataset of data from various domains at the beginning of the training process to extract relatively useful spatial features, it enables models to be trained quickly and accurately. Pretrained models in Keras library are used for transfer learning in computer vision and natural language processing. These models come with preloaded weights and biases, trained on large

datasets for multi-class classification. This saves time and computing power required for training the network from scratch.

In image classification, pretrained models like VGG16 and InceptionV3 can be used as feature extractors by removing their output layer. These models can extract basic features such as edges, curves, shapes, and lines. The models can also be partially trained by freezing some layers and fine-tuning others. This approach can improve the accuracy of the classification task.

3.4.1 VGG-16

Architecture

VGG16 is a convolutional neural network with 16 layers that was developed by the Visual Geometry Group and proposed by A. Zisserman and K. Simonyan for the ImageNet competition. It is known for its simplicity and has become a popular architecture in computer vision. With over 138 million parameters, it has achieved a top-5 accuracy of 92.3% on the ImageNet dataset. VGG16 learns weights and biases in its layers to improve its performance in image classification tasks.

The network architecture consists of 13 convolutional layers followed by 3 dense layers for classification. The number of filters in the convolution layers increases gradually, and max pooling layers are applied at different stages to extract informative features. The input is a 224 x 224 RGB image, and small-sized kernels with a receptive field are used in the convolution layers. Additionally, 1x1 kernels are incorporated to serve as linear input transformation.

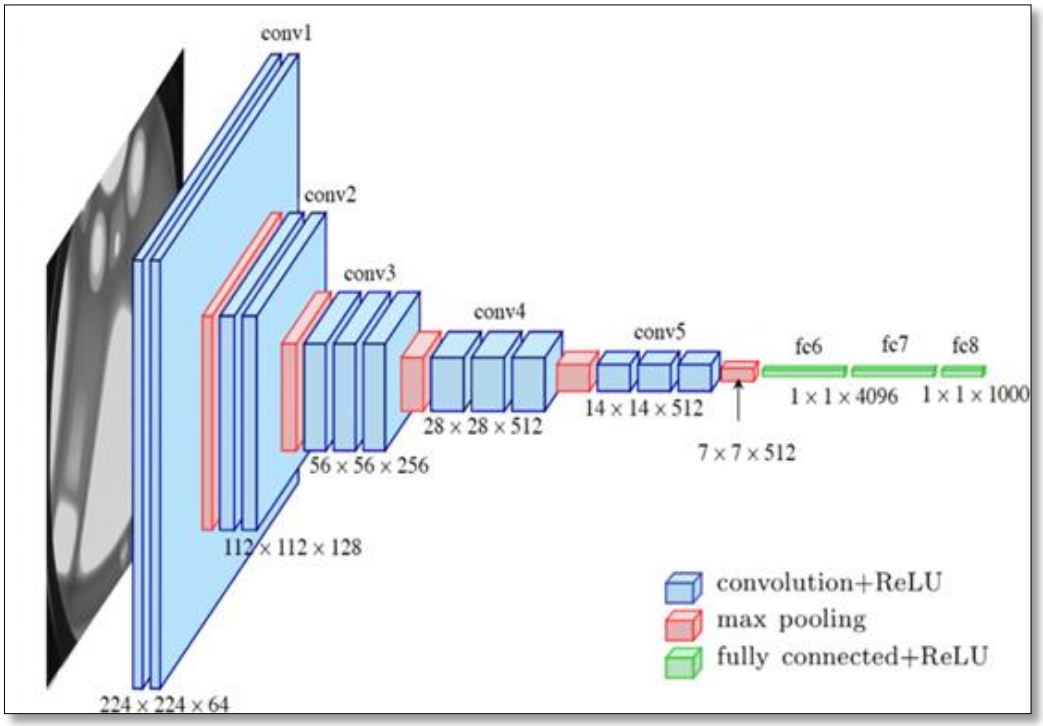


Figure 3.6: VGG-16 architecture

The Vgg 16 model is a deep neural network used for image recognition tasks. The model takes in 224x224x3 pixel images as input and passes them through a stack of convolutional layers. The first two layers are convolutional layers with 3x3 filters, producing an output size of 224x224x64. The output of these layers is then passed through a max pooling layer, which reduces the size of the image to 112x112x64. The second convolutional layer is similar to the first, but uses Rectified Linear Units (ReLU) to speed up training time. The output of this layer is also passed through a max pooling layer with a 2x2 window. After this, five more max pooling layers are used to downsize the image, followed by three fully connected layers with 4096 channels each. The final layer is a softmax layer, which produces a probability distribution over the 1000 possible image classes. This model has achieved state-of-the-art performance on several image recognition benchmarks.

The VGG16 architecture consists of a series of convolutional and pooling layers, followed by fully connected layers and an output layer. The input image is first passed through two convolution layers with 64 filters each, resulting in an output dimension of 224x224x64. Two

more convolution layers with 128 filters are added, followed by a pooling layer that reduces the image size to 56x56x128. Two more convolution layers with 256 filters each are added, followed by another pooling layer that reduces the image size to 28x28x256. Convolution Layer 4 with 512 filters is added, followed by a pooling layer that reduces the image size to 14x14x512. Another pooling layer reduces the image size to 7x7x512, and finally, two fully connected layers with 4090 nodes each are added. The output layer originally had 1000 classes, but it was modified to two classes for binary classification. This was done by adding a new fully connected layer and a sigmoid activation function.

To add the output layer for binary classification, the number of classes in the dependent variable (two) was specified, and a sigmoid activation function was used. The weight initializer and its arguments were chosen based on the specific problem being solved. The choice of weight initializer can affect the performance of the model, and various options such as uniform, normal, and truncated normal can be used. The arguments for the weight initializer can also be adjusted to further customize the model. Overall, the VGG16 architecture provides a powerful tool for image classification tasks and can be adapted to a wide range of applications.

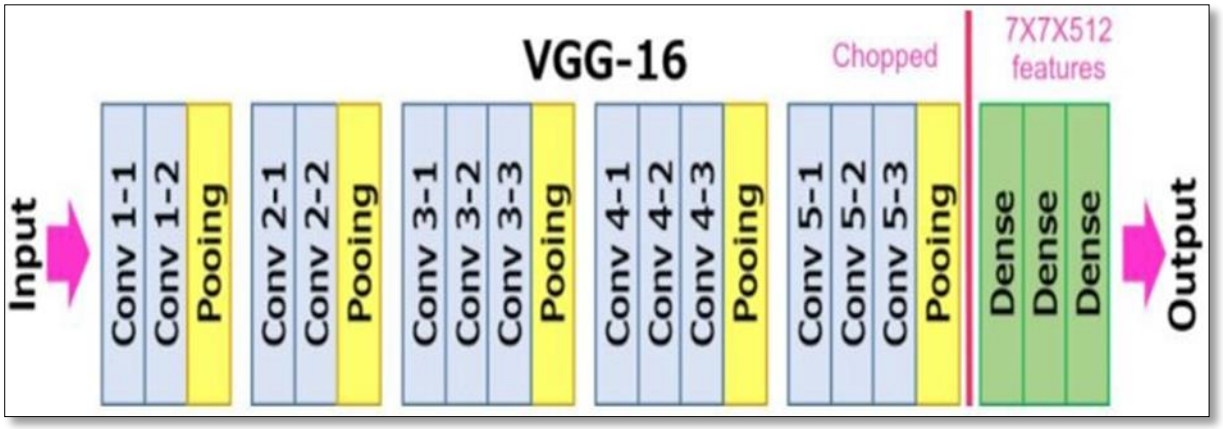
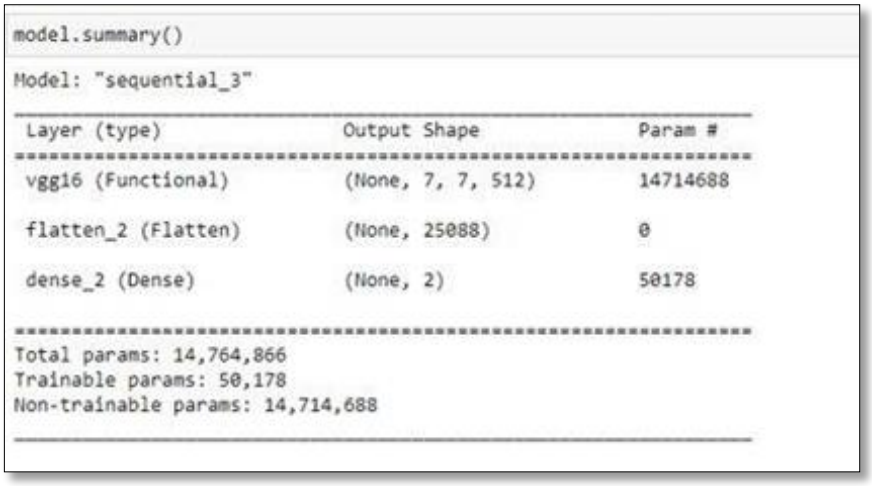


Figure 3.7: VGG-16 layers



```
model.summary()

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_2 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 3.8: VGG-16 model summary.

STEPS FOR IMPLEMENTATION OF TRANSFER LEARNING IN VGG16

Transfer learning implemented in VGG16 using Keras:

1.Instantiate a base model and load pre-trained weights into it.

2. Freeze all layers in the base model by setting trainable = False.

3. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.

4. Train the new layers on your dataset.

Changes made to final layer:

• Flatten the output of our base model to 1 dimension

• Add a fully connected layer of 512 units and ReLU activation

• This time, we will go with a dropout rate of 0.5

• Add a output Dense layer with Sigmoid activation

As a result, we can see that we get 94.8% Validation accuracy for VGG16.

EXAMPLE :

```
from tensorflow.keras.applications.vgg16 import VGG16
base_model=VGG16(input_shape=(224,224,3),include_top=False,weights='imagenet')# leave out of the last fully connected layer

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

```
for layer in base_model.layers:
    layer.trainable=False
```

```
x=layers.Flatten()(base_model.output)
x=layers.Dense(512,activation='relu')(x)

x=layers.Dropout(0.5)(x)
x=layers.Dense(1,activation='sigmoid')(x)
model=tf.keras.models.Model(base_model.input,x)
model.compile(loss="binary_crossentropy",optimizer='adam',metrics=['accuracy'])
```

Figure 3.9: Transfer learning implemented in VGG16 using Keras

3.4.2 Inception V3

Inception v3 is a powerful convolutional neural network designed by Google for image analysis and object detection. It was first introduced during the ImageNet Recognition Challenge and consists of nearly 48 layers. This neural network was trained on a subset of ImageNet dataset that has 1000 classes with input size 299x299.

The Inception v3 model is composed of both symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, dropouts, and fully connected layers. Batch normalization is used extensively throughout the model and applied to activation inputs. The model employs filters of various sizes, including 1x1, 3x3, and 5x5, to extract features of different scales from the input image. To reduce the number of dimensions and computations, a 1x1 filter is added before the 3x3 and 5x5 filters.

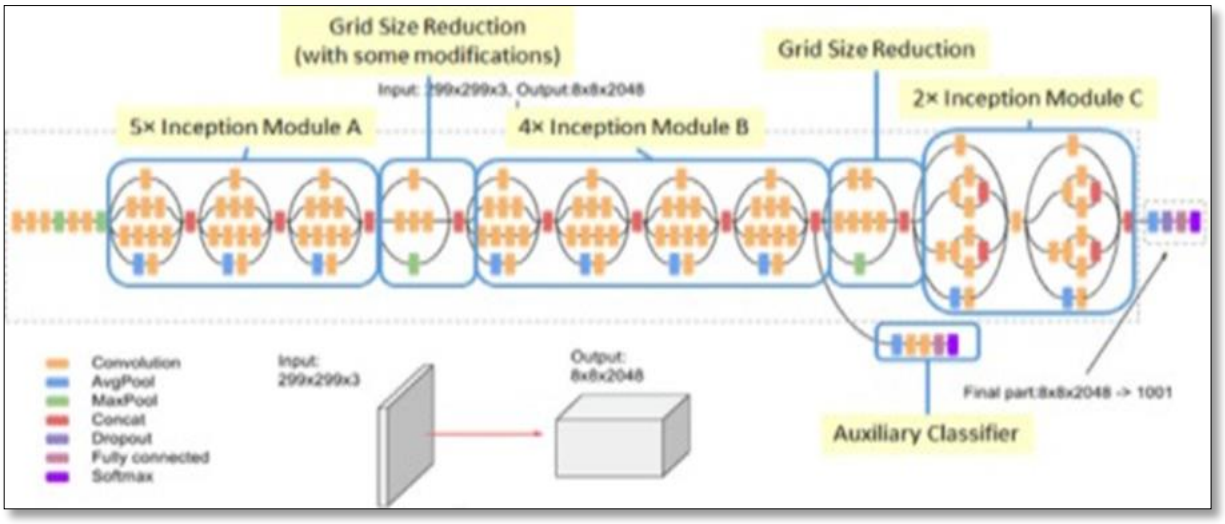


Figure 3.10: InceptionV3 architecture

The Inception v3 model utilizes parallel filters to conduct convolutions at different scales, followed by aggregation into one block. The model employs various operations such as pooling, factorized convolutions, smaller and asymmetric convolutions, and concatenation within each module. The architecture is built progressively with these features.

Inception v3 Architecture

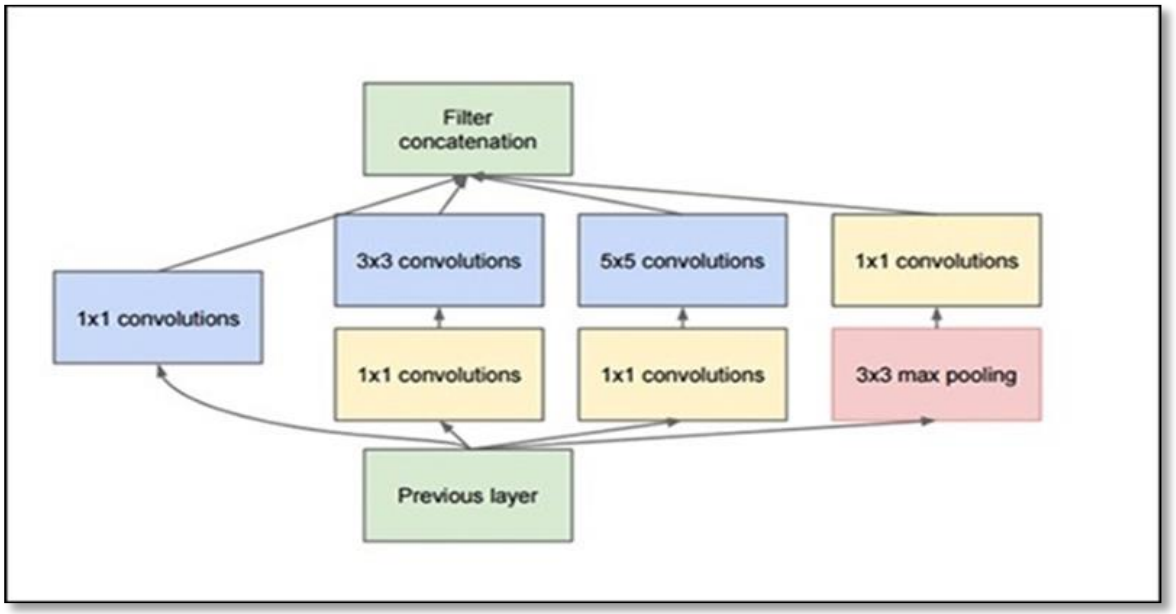


Figure 3.11: Inception module with dimension reduction

Factorizing convolutions

Factorization convolutions are a technique used to improve the efficiency of neural networks by reducing the number of connections and parameters that need to be learned. By using a combination of 3x1 and 1x3 convolutions instead of a single 3x3 convolution, the number of parameters can be reduced by 33%, resulting in faster and more efficient network performance. This technique allows for greater control over the network's efficiency and ensures that it can perform well while using fewer resources.

Smaller convolutions

To speed up training, larger convolutions can be replaced with smaller ones. For instance, a 5x5 filter has 25 parameters, while two 3x3 filters have only 18 parameters (3x3+3x3). In this approach, a 3x3 convolution layer is placed in the middle, followed by a fully-connected layer. This technique can reduce the computational cost of training without compromising performance.

Asymmetric convolutions

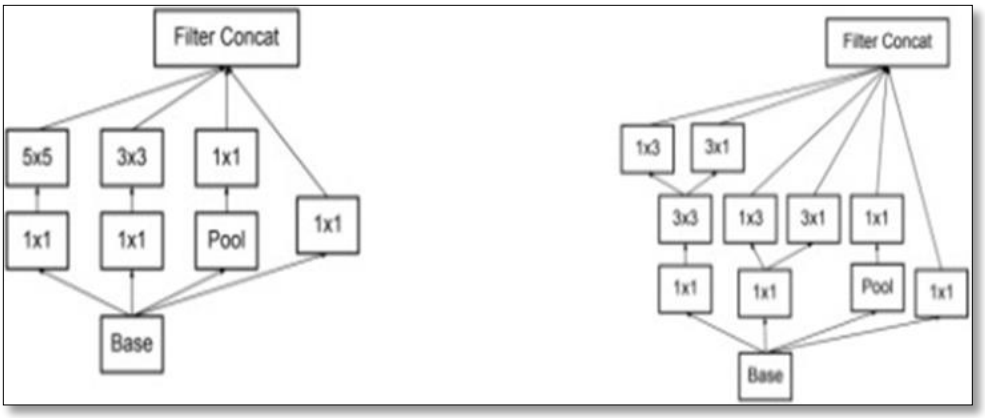


Figure 3.12: Asymmetric convolutions

Replacing a 3×3 convolution with a 1×3 convolution followed by a 3×1 convolution is possible. However, replacing a 3×3 convolution with a 2×2 convolution would slightly increase the number of parameters compared to the asymmetric convolution approach.

Auxiliary classifier

An auxiliary classifier is a small convolutional neural network (CNN) that is inserted between layers during training. Its loss is added to the main network loss, which helps to regularize the training process. In Inception v3, the auxiliary classifier serves as a regularization technique to prevent overfitting and improve the generalization performance of the model.

Grid size reduction

Grid size reduction is usually done by pooling operations.

STEPS FOR IMPLEMENTATION OF TRANSFER LEARNING

Transfer learning implemented using Keras:

1.Instantiate a base model and load pre-trained weights into it.

2.Freeze all layers in the base model by setting trainable = False.

3.Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.

4.Train the new layers on your dataset.

```
base_model=tf.keras.applications.inception_v3.InceptionV3(
    include_top=False,
    weights='imagenet',
    input_tensor=None,
    input_shape=(299,299,3),
    pooling=None,
    classes=1000,
    classifier_activation='softmax'
)
for layer in base_model.layers:
    layer.trainable=False

x=layers.Flatten()(base_model.output)
x=layers.Dense(512,activation='relu')(x)

x=layers.Dropout(0.5)(x)
x=layers.Dense(1,activation='sigmoid')(x)
model=tf.keras.models.Model(base_model.input,x)
```

Figure 3.13: Transfer learning implemented in Inception V3 using Keras

3.5 Hyperparameter tuning.

Hyperparameter tuning is a crucial process in optimizing the performance of a network. By adjusting the network architecture and hyperparameter values, the best set of parameters can be obtained. This involves running data through the model using different operations and comparing the predicted results with the actual values to evaluate accuracy. The parameters are then adjusted until the optimal values are achieved. Overall, hyperparameter tuning involves a careful and iterative process of testing and adjusting various parameters to maximize the performance of the network.

There are various ways to optimize the performance of a convolutional neural network (CNN). One approach is to modify the number of convolutional blocks and input and output parameters in the network, as well as the kernel size and other arguments. Another technique is to use dropout, a regularization method that can reduce overfitting and improve generalization by randomly removing a fixed number of neurons from the neural network.

The number of epochs is another important hyperparameter that can be adjusted to improve CNN accuracy. The number of epochs represents the number of times the entire training data is fed to the network during training. By varying the number of epochs, we can achieve better accuracy.

Activation functions are also essential for CNNs because they introduce nonlinearity to the model. Sigmoid is often used in the output layer for binary predictions. In addition, image augmentation is a technique that can artificially increase the number of images in the training set by applying random transformations to existing images.

The batch size is another hyperparameter that can affect CNN performance. The batch size defines the number of samples that will be propagated through the network before the weights are updated. Larger batch sizes require more memory space but can result in faster training because weights are updated less frequently. Mini batches are commonly used in training because they strike a balance between training speed and memory requirements. Adjusting the learning rate is a crucial parameter in image training, as it governs the impact of weight and bias updates during the learning process. A lower learning rate may lead to a slower rate of convergence but can also result in a smoother learning trajectory.

Early Stopping and Callback

Callbacks are functions that can be applied to monitor the performance of a model during training. One common use case is to save the model with its weights at a specific point when an improvement is observed in the parameters. It is generally recommended to save the model weights only when there is an improvement.

In TensorFlow, callbacks can be provided to the fit() function via the "callbacks" argument. The "monitor" parameter can be used to specify the performance measure to track in order to end training. For example, we can create a callback that stops training early once a certain value for the validation loss is reached.

To save the best model observed during training, we can use a callback that monitors the factor we are interested in, such as the validation loss. We can also set a delay for triggering the callback, by specifying the number of epochs we would like to see no improvement before stopping the training. This can be done using the "patience" parameter.

One popular callback is Early Stopping, which stops training when a monitored quantity has stopped improving. This allows us to specify a large number of epochs for training and stop training once the model performance stops improving on the validation dataset.

In our project, we track the loss on the validation set during training and use it to determine when to stop training the model. The goal is to achieve accuracy without overfitting.

3.6 Evaluation Parameters

Confusion matrix displays the number of true positive, true negative, false positive and false negatives. We have used these parameters for evaluating the performance.

True positive (TP) : It is the correct classification of the, positive samples which are predicted accurately as a positive label.

E.g.: If an image contains cancerous(Malignant) cells and the model classifies the presence of cancer successfully.

True negative (TN): Negative samples which are correctly predicted as a negative label.

E.g.: If there is no cancer present in image the model predicts it non-cancer False

Negative (FN): positive samples which are predicted incorrectly as a negative label.

E.g. : There is no cancer in the image but the model predicts it contain cancer.

False positive (FP): negative samples which are incorrectly predicted as a positive label.

E.g.: If image does have cancerous cells but the model classifies it non-cancerous.

Accuracy: It is used for performance evaluation of the classifier. It is the ratio of number of Correct predictions by total number of predictions made.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

3.7 Webpage building for prediction.

The objective of this project is to create a proof-of-concept application that allows medical professionals to upload CT scans and receive predictions from a deep learning model. To achieve this, a web application was developed using React JS, with a Python-based back end built on the Flask micro-framework.

The user interface of the web app prompts the user to upload a CT scan image of their choice, which is then processed by the application and sent to the predictive model for analysis. The output of the model is then presented to the user in a format that is easily comprehensible and provides meaningful insights.

The first step involved designing the front-end of the web application, which included implementing an image upload function using React JS. Next, the web page was integrated with the predictive model using Flask.

Finally, the Python code for prediction was executed, and the resulting predictions were displayed on the user interface for the uploaded CT scan image.

Chapter 4

RESULTS AND DISCUSSION

VGG-16 model results.

• VGG-16 Val accuracy: 94.8%

• VGG-16 Training accuracy: 95.0%

This is a plot of accuracy on the training and validation datasets over training epochs. The training and validation accuracy of the fine-tuned VGG16 model is observed that the training and validation results have similar convergence rate. This model does not overfit.

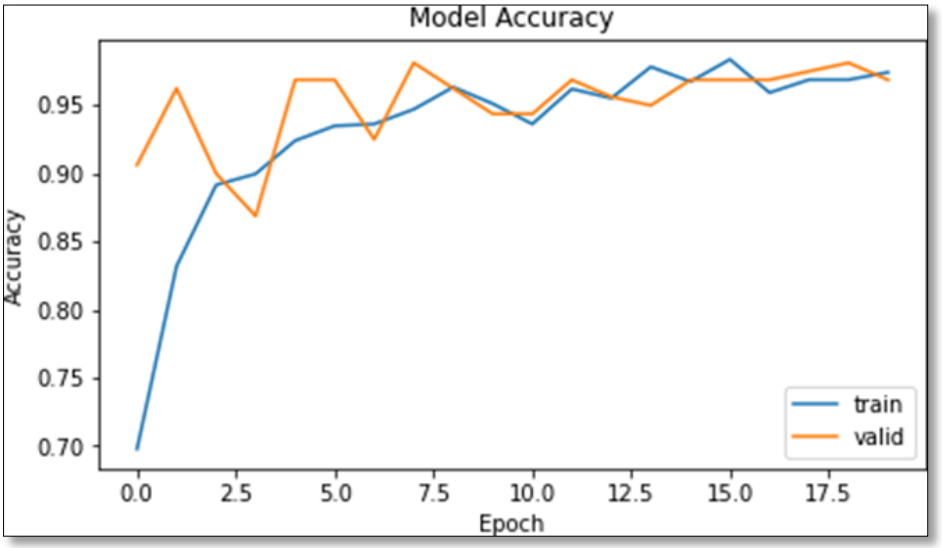


Figure 4.1: VGG-16 Model accuracy.

This a plot of loss on the training and validation dataset over training epochs.

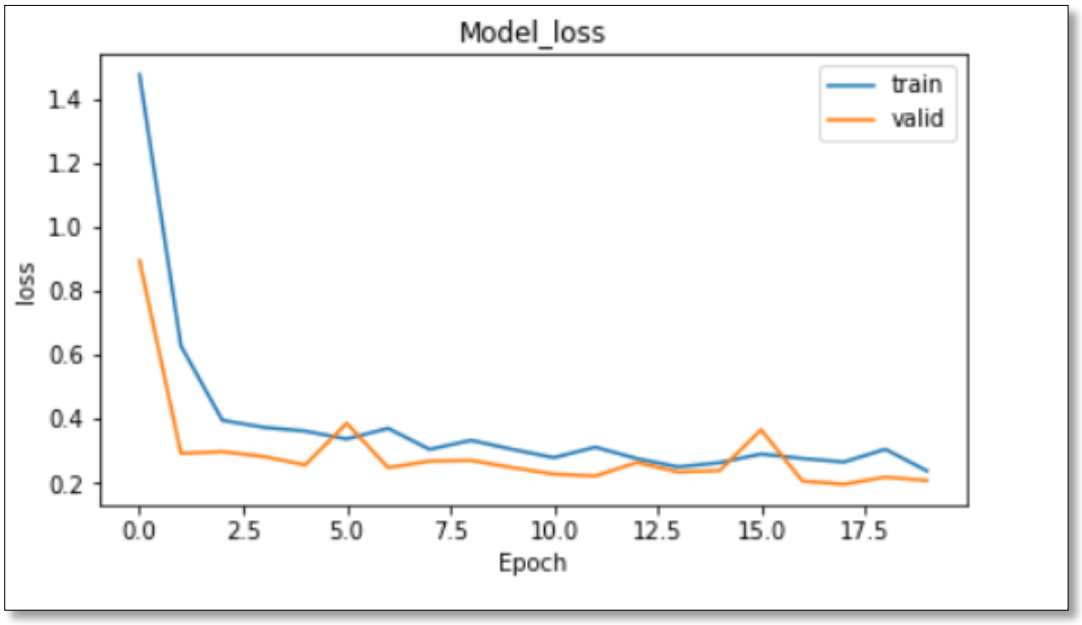


Figure 4.2: VGG-16 Model loss.

The experimental results prove that the transfer learning improves the performance.

Inception V3 model results

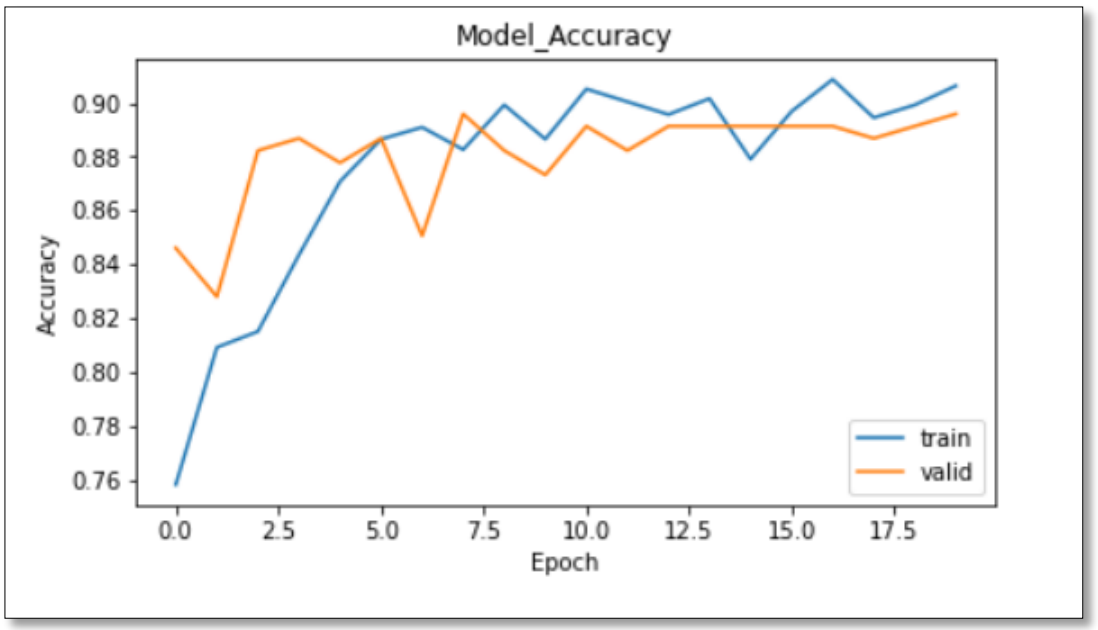


Figure 4.3: InceptionV3 Model accuracy

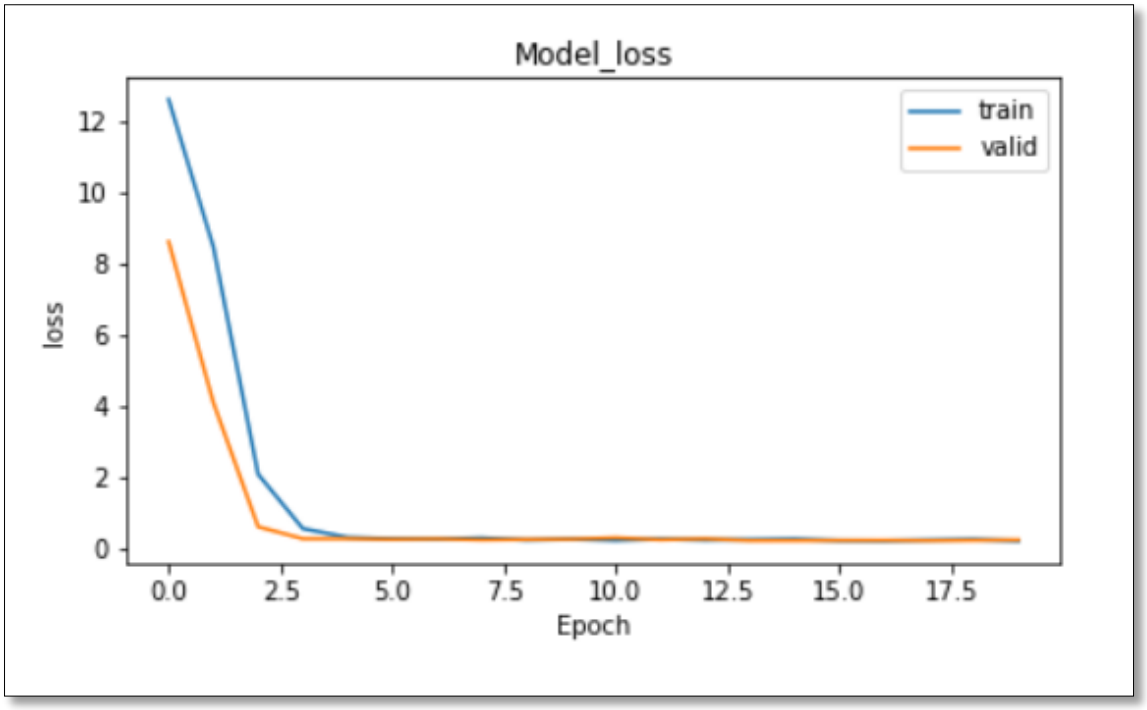


Figure 4.4: InceptionV3 Model loss

- This is a plot of accuracy on the training and validation dataset over training epochs.
- After the end of the training Inception model after 20 epochs, Val Accuracy is 89.59%.
- From the plot of loss, we can see that the model has comparable performance on both train and validation dataset.

4.1 Confusion Matrix

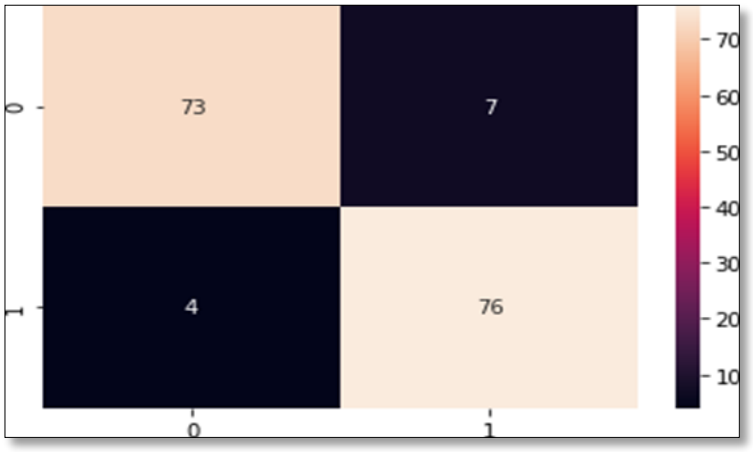


Figure 4.5: InceptionV3 Confusion Matrix

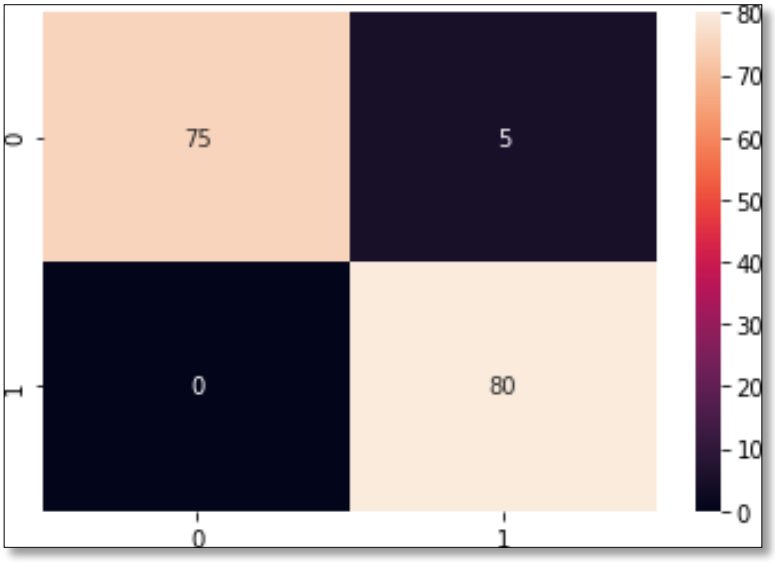


Figure 4.6: VGG-16 Confusion Matrix

The evaluation of two models, VGG-16 and InceptionV3, was done using a confusion matrix containing 160 images, 80 for each class (Malignant and Benign). The black color in the matrix indicates misclassified images, while the other color represents correctly classified images.

The number of true negatives (TN) for VGG-16 is 75, indicating that 75 Benign images were correctly classified as negative. The number of true negatives for InceptionV3 is 73, indicating that 73 Benign images were correctly classified as negative.

The number of true positive (TP) for VGG-16 is 80, indicating that 80 Malignant images were correctly classified as positive. The number of true positives for InceptionV3 is 76, indicating that 76 Malignant images were correctly classified as positive.

The number of false negatives (FN) for VGG-16 is 5, indicating that 5 Benign images were incorrectly classified as negative. The number of false negatives for InceptionV3 is 7, indicating that 7 Benign images were incorrectly classified as negative.

Finally, the number of false positives (FP) for VGG-16 is 0, indicating that Malignant images were incorrectly classified as positive. The number of false positives for InceptionV3 is 4, indicating that 4 Malignant images were incorrectly classified as positive.

Overall, the VGG-16 model performed well in distinguishing between CT scans with cancer nodules and those without, as it had a higher number of true positives and true negatives and a lower number of false positives and false negatives than InceptionV3.

4.2 Prediction Results

We begin by loading our pre-trained models and importing the required libraries. Next, we load an image from the test set and resize it. The image is then converted to a 3D NumPy array, and the pixel values are scaled and converted to the array format.

To pass the image to the predict function, we use the expand_dims () function to expand the shape of the array by inserting a new axis at the desired position. The predict function is then used to classify the image, and the result is saved in the variable pred.

Finally, we print the result of the prediction. Since the input image was a Malignant case, we get the output label as a Malignant case, denoted by label 1.

```
[ ]
img=image.load_img('/content/drive/MyDrive/R_Data/Split_data_CNN/validation/Malignant cases/Malignant case (212).jpg',
                   color_mode='rgb',target_size=(224,224))

[ ] x=image.img_to_array(img)
x=np.expand_dims(x,axis=0)
x.shape

(1, 224, 224, 3)

pred=model.predict(x)
pred

array([[1.]], dtype=float32)

class_names=["Bengin_case","Malignant_case",]
prediction=class_names[int(pred[0][0])]
print(pred[0][0])
print(prediction)

1.0
Malignant_case
```

Figure 4.7: Final prediction

The uploaded CT scan image is analyzed by the model built and the predicted diagnostic results is displayed in webpage as shown.

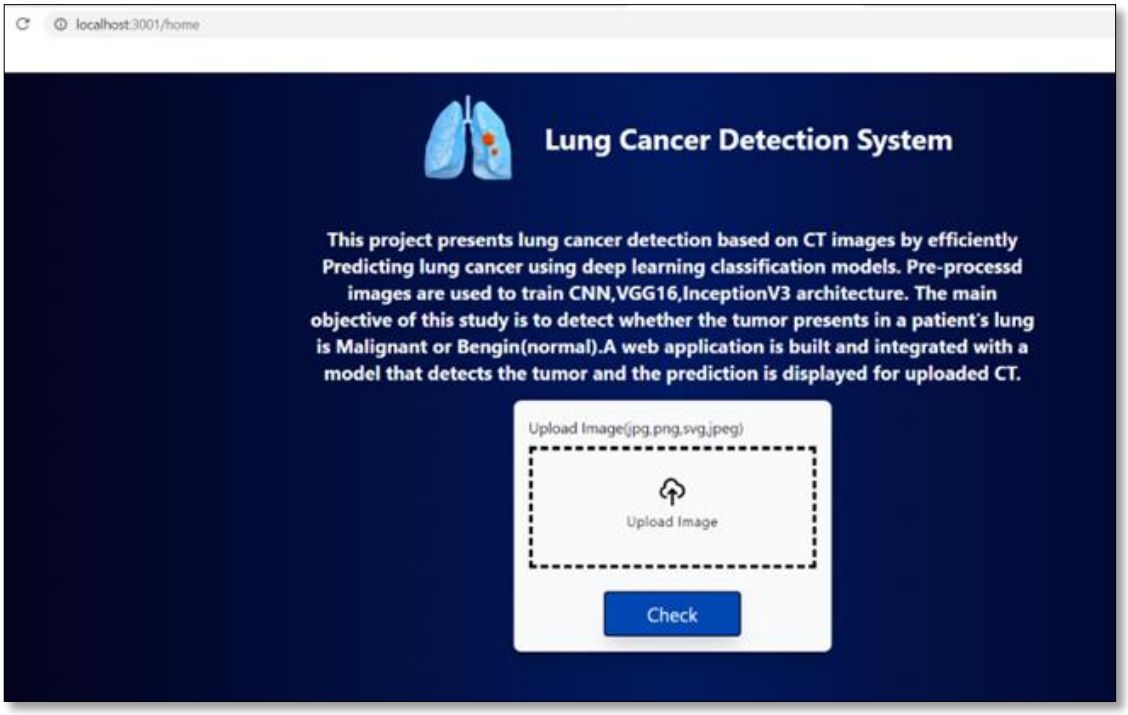


Figure 4.8: The user interface for the uploaded CT scan image.

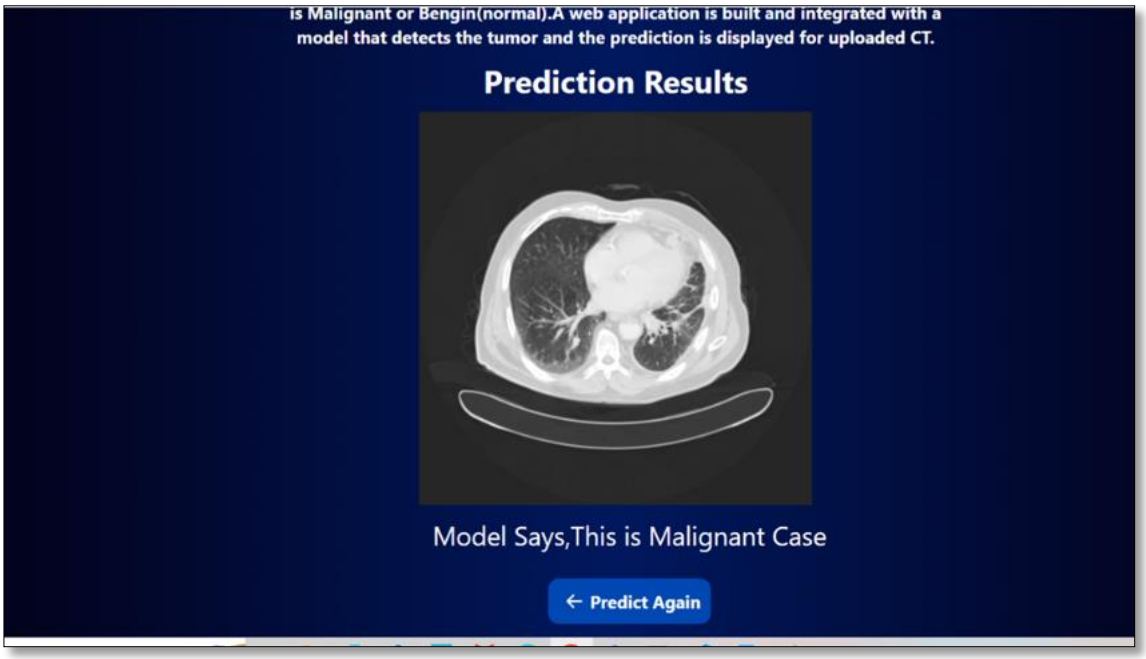


Figure 4.9: Prediction for a given user input.

Chapter 5

CONCLUSION

This project's work is based on deep learning methods for CT scan-based lung cancer detection. The proposed method is based on Transfer Learning models and Convolutional Neural Networks (CNN) models, specifically VGG16 and Inception V3. To improve the validation accuracy, transfer learning was used, along with image pre-processing and hyperparameter tuning. When predicting the output, the VGG16 model performed more accurately, with a 95.0% accuracy rate. However, compared to VGG16, the Inception V3 model trained more quickly. Since the dataset affects which is better, it is difficult to say. For instance, VGG16 can handle smaller images with less detailed information well, whereas Inception V3 can handle larger images with more detailed information well. A website was developed for users to predict cancer based on the uploaded CT scan in order to make the method user-friendly. These findings can be used by radiologists and physicians to treat patients more effectively.

Overall, this project shows how deep learning methods could be used to increase the precision of lung cancer detection using CT scans, which could significantly impact the outcomes of patients.

Bibliography

1. Karthikeyan, M. V. Et Al. “Lung Cancer Detection Using CT SCAN Images.” International Journal Of Advanced Research In Science, Communication And Technology (2021): N. Pag.

2. Ms. Swathi Velugoti, Ms. Revuri Harshini Reddy, Ms. Sadiya Tarannum , Mr. Sama Tharun Kumar ReddyLung Nodule Detection and Classification using Image Processing Techniques Department of Computer Science and Engineering, Guru Nanak Institutions Technical Campus Hyderabad – Telangana, India
3. Rotem Golan, Christian Jacob, J"org Denzinger, "Lung Nodule Detection in CT Images using Deep Convolutional Neural Networks", 978-1-5090-0620- 5/16
4. "Detection Of Lung Cancer From CT Image Using Image Processing And Neural Network." 2015 International Conference On Electrical Engineering And Information Communication Technology (ICEEICT): 1-6
5. by BA Skourt · 2018 · Cited by 221 — In this work, we propose a lung CT image segmentation using the U-net architecture, one of the most used architectures in deep
6. Ahmed, T., Parvin, M. S., Haque, M. R., Uddin, M. S., et al. (2020). Lung cancer detection using ct image based on 3d convolutional neural network. Journal of Computer and Communications, 8(03), 35.
7. Md. Badrul Alam Miah, Mohammad Abu Yousuf, "Detection of Lung Cancer from CT Image Using Image Processing and Neural Network", 2nd Int'l Con- ference on Electrical Engineering and Information Communication Technol- ogy (ICEEICT)2015 Jahangirnagar University, Dhaka-1342, Bangladesh, 21-23 May 2015, 978-1-4673-6676-2/1
8. Taolin Jin, Hui Cui, Shan Zeng, Xiuying Wang, "Learning deep spatial lung features by 3D convolutional neural network for early cancer detection" 978- 1-5386-2839-3/17/
9. by MS AL-Huseiny · 2021 · Cited by 8 — The use of computer algorithms has gained momentum in filling/assisting roles of specialists especially in early diagnosis scenarios.
10. Image Data Augmentation (Volume 7, Issue 3 - V7I3-1606). Available online at: https://www.ijariit.com. Lung Cancer Detection using Image Processing and CNN. Anubha Gupta.
11. Ahmed, T., Parvin, M. S., Haque, M. R., Uddin, M. S., et al. (2020). Lung cancer detection using ct image based on 3d convolutional neural network. Journal of Computer and Communications, 8(03), 35.
12. https://www.kaggle.com/datasets/hamdallak/the-iqothnccd-lung-cancer-dataset

APPENDICES

CNN

```
import os

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

"""**Exploratory Data Analysis (EDA)**"""

import warnings
warnings.filterwarnings('ignore')
# Get all the paths
data_dir_list = os.listdir('/content/drive/MyDrive/R_Data')
print(data_dir_list)
path, dirs, files = next(os.walk("/content/drive/MyDrive/R_Data"))
file_count = len(files)
#print(file_count)

# Make new base directory
original_dataset_dir = '/content/drive/MyDrive/R_Data'
base_dir = '/content/drive/MyDrive/R_Data/Split_data_CNN'
os.mkdir(base_dir)

#create two folders (train and validation)
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)

validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)

#Under train folder create two folders

train_cloud_dir = os.path.join(train_dir, 'Bengin cases')
os.mkdir(train_cloud_dir)

train_foggy_dir = os.path.join(train_dir, 'Malignant cases')
os.mkdir(train_foggy_dir)

#Under validation folder create two folders

validation_cloud_dir = os.path.join(validation_dir, 'Bengin cases')
os.mkdir(validation_cloud_dir)

validation_foggy_dir = os.path.join(validation_dir, 'Malignant cases')
os.mkdir(validation_foggy_dir)
```

```
os.mkdir(validation_foggy_dir)

import os
import random
from shutil import copyfile

def split_data(SOURCE, TRAINING, VALIDATION, SPLIT_SIZE):
    files = []
    for filename in os.listdir(SOURCE):
        file = SOURCE + filename
        if os.path.getsize(file) > 0:
            files.append(filename)
        else:
            print(filename + " is zero length, so ignoring.")

```

```
    training_length = int(len(files) * SPLIT_SIZE)
    valid_length = int(len(files) - training_length)
    shuffled_set = random.sample(files, len(files))
    training_set = shuffled_set[0:training_length]
    valid_set = shuffled_set[training_length:]

    for filename in training_set:
        this_file = SOURCE + filename
        destination = TRAINING + filename
        copyfile(this_file, destination)

    for filename in valid_set:
        this_file = SOURCE + filename
        destination = VALIDATION + filename
        copyfile(this_file, destination)

BENGIN_SOURCE_DIR = '/content/drive/MyDrive/R_Data/Bengin cases/'
TRAINING_BENGIN_DIR = '/content/drive/MyDrive/R_Data/Split_data_CNN/train/Bengin cases/'
VALID_BENGIN_DIR = '/content/drive/MyDrive/R_Data/Split_data_CNN/validation/Bengin cases/'

MALIGNANT_SOURCE_DIR = '/content/drive/MyDrive/R_Data/Malignant cases/'
TRAINING_MALIGNANT_DIR = '/content/drive/MyDrive/R_Data/Split_data_CNN/train/Malignant cases/'
VALID_MALIGNANT_DIR = '/content/drive/MyDrive/R_Data/Split_data_CNN/validation/Malignant cases/'

import os
import random
from shutil import copyfile

split_size = .85

split_data(BENGIN_SOURCE_DIR, TRAINING_BENGIN_DIR, VALID_BENGIN_DIR, split_size)
split_data(MALIGNANT_SOURCE_DIR, TRAINING_MALIGNANT_DIR, VALID_MALIGNANT_DIR, split_size)

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.image import imread
import pathlib

image_folder = ['Bengin cases', 'Malignant cases']
nimgs = {}
for i in image_folder:
    nimages = len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_CNN/train/'+i+'/'))
    nimgs[i]=nimages
plt.figure(figsize=(9, 6))
plt.bar(range(len(nimgs)), list(nimgs.values()), align='center')
plt.xticks(range(len(nimgs)), list(nimgs.keys()))
```

```
plt.xticks(range(len(nimgs)), list(nimgs.keys()))
plt.title('Distribution of different classes in Training Dataset')
plt.show()

for i in ['Bengin cases', 'Malignant cases']:
    print('Training {} images are: '.format(i)+str(len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_CNN/train/'+i+'/'))))

for i in ['Bengin cases', 'Malignant cases']:
    print('Testing {} images are: '.format(i)+str(len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_CNN/validation/'+i+'/'))))

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.image import imread
import pathlib

image_folder = ['Bengin cases', 'Malignant cases']
nimgs = {}
for i in image_folder:
    nimages = len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_CNN/validation/'+i+'/'))
    nimgs[i]=nimages
plt.figure(figsize=(9, 6))
plt.bar(range(len(nimgs)), list(nimgs.values()), align='center')
plt.xticks(range(len(nimgs)), list(nimgs.keys()))
plt.title('Distribution of different classes in Testing Dataset')
plt.show()
```

```
!pip install tensorflow==2.7.0

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import sequential
from tensorflow.keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import Flatten

import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=8,
                                   zoom_range=0.15,
                                   width_shift_range=0.15,
                                   height_shift_range=0.15,
                                   shear_range=0.10,
                                   horizontal_flip=True,
                                   brightness_range=[0.8,1.2],
                                   channel_shift_range=0.15,
                                   fill_mode="nearest")

test_datagen = ImageDataGenerator(rescale=1./255)

x_train = train_datagen.flow_from_directory('/content/drive/MyDrive/R_Data/Split_data_CNN/train',target_size=
(224,224),batch_size=32,class_mode='binary')

x_test = test_datagen.flow_from_directory('/content/drive/MyDrive/R_Data/Split_data_CNN/validation',target_size=
(224,224),batch_size=32,class_mode='binary',shuffle =False)

print(x_train.class_indices)
print(x_test.class_indices)

from keras.layers.convolutional import Convolution2DTranspose
from keras.models import Sequential
from IPython.core import history
from tensorflow.python import metrics
```

```
model = Sequential()
model.add(Convolution2D (32,(3,3),input_shape=(224,224,3),activation='relu'))
model.add(MaxPooling2D (pool_size=(3,3)))
model.add(Dropout(0.25))
model.add(Convolution2D(64,(3,3),activation='relu'))
model.add(Convolution2D(64,(3,3),activation='relu'))


model.add(MaxPooling2D (pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Convolution2D(128,(3,3),activation='relu'))
model.add(Convolution2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D (pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(units=512,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=128,activation='relu'))
model.add(Dense(1,activation="sigmoid"))

model.summary()
```

```
model.compile(loss="binary_crossentropy",optimizer='adam',metrics=['accuracy'])

TRAIN_COUNT =len(x_train.filepaths)
TEST_COUNT=len(x_test.filepaths)
TRAIN_STEPS_PER_EPOCH= round(TRAIN_COUNT/32)
VAL_STEPS_PER_EPOCH = round(TEST_COUNT/32)

print(f"TRAIN_STEPS_PER_EPOCH:{TRAIN_STEPS_PER_EPOCH}")
print(f"VAL_STEPS_PER_EPOCH:{VAL_STEPS_PER_EPOCH}")

from tensorflow.python.keras.callbacks import EarlyStopping,ModelCheckpoint
EARLY_STOP_PATIENCE =5

cb_early_stopper =EarlyStopping(monitor='val_losss',patience=EARLY_STOP_PATIENCE)
cb_checkpointer = ModelCheckpoint(filepath='FINAL_CNN.hdf5',monitor='val_loss',save_best_only=True,mode='auto')
callbacks=[cb_checkpointer,cb_early_stopper]
history=model.fit_generator(x_train,steps_per_epoch=TRAIN_STEPS_PER_EPOCH,epochs=20,
                            validation_data=x_test,validation_steps=VAL_STEPS_PER_EPOCH)

model.save("FINAL_CNN.hdf5")

keys=history.history.keys()
print(keys)

plt.figure(1,figsize=(15,8))
plt.subplot(221)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model_Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','valid'])

plt.figure(1,figsize=(15,8))
plt.subplot(222)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model_loss')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['train','valid'])
plt.show()

from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix

y_pred=model.predict(x_test)

from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay


threshold=0.5
y_pred_threshold=[(1 if val>threshold else 0) for val in y_pred]
```

```
threshold=0.5
y_pred_threshold=[(1 if val>threshold else 0) for val in y_pred]
y_true=x_test.labels.tolist()
print(confusion_matrix(y_true,y_pred_threshold))

print(accuracy_score(y_true,y_pred_threshold))

sns.heatmap(confusion_matrix(y_true,y_pred_threshold),annot=True)

from tensorflow import keras
from tensorflow.keras import layers
from keras.models import load_model
model=load_model('/content/FINAL_CNN.hdf5')

```

```
270
271 from keras.preprocessing import image
272 import cv2
273 import numpy as np
274 import PIL
275 from PIL import Image
276
277 img=image.load_img('/content/drive/MyDrive/R_Data/Split_data_CNN/validation/Malignant cases/Malignant case (212).jpg',
278                    color_mode='rgb',target_size=(224,224))
279
280 x=image.img_to_array(img)
281 x=np.expand_dims(x,axis=0)
282 x.shape
283
284 pred=model.predict(x)
285 pred
286
287 class_names=["Bengin_case","Malignant_case",]
288 prediction=class_names[int(pred[0][0])]
289 print(pred[0][0])
290 print(prediction)
291
```

VGG 16

```
8 """
9
10 import os
11
12 from google.colab import drive
13 drive.mount('/content/drive')
14
15 """**Exploratory Data Analysis (EDA)**"""
16
17 import warnings
18 warnings.filterwarnings('ignore')
19 # Get all the paths
20 data_dir_list = os.listdir('/content/drive/MyDrive/R_Data')
21 print(data_dir_list)
22 path, dirs, files = next(os.walk("/content/drive/MyDrive/R_Data"))
23 file_count = len(files)
24 #print(file_count)
25
26 # Make new base directory
27 original_dataset_dir = '/content/drive/MyDrive/R_Data'
28 base_dir = '/content/drive/MyDrive/R_Data/Split_data_vgg16_o/'
29 os.mkdir(base_dir)
30
31 #create two folders (train and validation)
32 train_dir = os.path.join(base_dir, 'train')
33 os.mkdir(train_dir)
34
35 validation_dir = os.path.join(base_dir, 'validation')
36 os.mkdir(validation_dir)
37
38 #Under train folder create two folders
39
40 train_cloud_dir = os.path.join(train_dir, 'Bengin cases')
41 os.mkdir(train_cloud_dir)
42
43 train_foggy_dir = os.path.join(train_dir, 'Malignant cases')
44 os.mkdir(train_foggy_dir)
45
46 #Under validation folder create five folders
47
48 validation_cloud_dir = os.path.join(validation_dir, 'Bengin cases')
49 os.mkdir(validation_cloud_dir)
50
51 validation_foggy_dir = os.path.join(validation_dir, 'Malignant cases')
52 os.mkdir(validation_foggy_dir)
53
```

```
import os
import random
from shutil import copyfile

split_size = .70

split_data(BENGIN_SOURCE_DIR, TRAINING_BENGIN_DIR, VALID_BENGIN_DIR, split_size)
split_data(MALIGNANT_SOURCE_DIR, TRAINING_MALIGNANT_DIR, VALID_MALIGNANT_DIR, split_size)

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.image import imread
import pathlib

image_folder = ['Bengin cases', 'Malignant cases']
nimgs = {}
for i in image_folder:
    nimages = len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_vgg16_o/train/'+i+'/'))
    nimgs[i]=nimages
plt.figure(figsize=(9, 6))
plt.bar(range(len(nimgs)), list(nimgs.values()), align='center')
plt.xticks(range(len(nimgs)), list(nimgs.keys()))
plt.title('Distribution of different classes in Training Dataset')
plt.show()

for i in ['Bengin cases', 'Malignant cases']:
    print('Training {} images are: '.format(i)+str(len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_vgg16_o/train/'+i+'/'))))

for i in ['Bengin cases', 'Malignant cases']:
    print('Testing {} images are: '.format(i)+str(len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_vgg16_o/validation/'+i+'/'))))

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.image import imread
import pathlib

image_folder = ['Bengin cases', 'Malignant cases']
nimgs = {}
for i in image_folder:
    nimages = len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_vgg16_o/validation/'+i+'/'))
    nimgs[i]=nimages
plt.figure(figsize=(9, 6))
plt.bar(range(len(nimgs)), list(nimgs.values()), align='center')
plt.xticks(range(len(nimgs)), list(nimgs.keys()))
plt.title('Distribution of different classes in Testing Dataset')
plt.show()

!pip install tensorflow==2.7.0

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import sequential
from tensorflow.keras.layers import Dense
```

```
from keras.models import sequential
from tensorflow.keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import Flatten

import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=8,
                                   zoom_range=0.15,
                                   width_shift_range=0.15,
                                   height_shift_range=0.15,
                                   shear_range=0.10,
                                   horizontal_flip=True,
                                   brightness_range=[0.8,1.2],
                                   channel_shift_range=0.15,
                                   fill_mode="nearest")

```

```
                                    fill_mode="nearest")

test_datagen = ImageDataGenerator(rescale=1./255)

x_train = train_datagen.flow_from_directory('/content/drive/MyDrive/R_Data/Split_data_vgg16_o/train',target_size=
(224,224),batch_size=32,class_mode='binary')

x_test = test_datagen.flow_from_directory('/content/drive/MyDrive/R_Data/Split_data_vgg16_o/validation',target_size=
(224,224),batch_size=32,class_mode='binary')

print(x_train.class_indices)
print(x_test.class_indices)

from keras.layers.convolutional import Convolution2DTranspose
from keras.models import Sequential

from tensorflow.keras.applications.vgg16 import VGG16
base_model=VGG16(input_shape=(224,224,3),include_top=False,weights='imagenet')# leave out of the last fully connected layer

for layer in base_model.layers:
    layer.trainable=False

x=layers.Flatten()(base_model.output)
x=layers.Dense(512,activation='relu')(x)

x=layers.Dropout(0.5)(x)
x=layers.Dense(1,activation='sigmoid')(x)
model=tf.keras.models.Model(base_model.input,x)
model.compile(loss="binary_crossentropy",optimizer='adam',metrics=['accuracy'])

TRAIN_COUNT =len(x_train.filepaths)
TEST_COUNT=len(x_test.filepaths)
TRAIN_STEPS_PER_EPOCH= round(TRAIN_COUNT/32)
VAL_STEPS_PER_EPOCH = round(TEST_COUNT/32)

print(f"TRAIN_STEPS_PER_EPOCH:{TRAIN_STEPS_PER_EPOCH}")
print(f"VAL_STEPS_PER_EPOCH:{VAL_STEPS_PER_EPOCH}")

from tensorflow.python.keras.callbacks import EarlyStopping,ModelCheckpoint
EARLY_STOP_PATIENCE =5

cb_early_stopper =EarlyStopping(monitor='val_losss',patience=EARLY_STOP_PATIENCE)
cb_checkpointer = ModelCheckpoint(filepath='best.hdf5',monitor='val_loss',save_best_only=True,mode='auto')
callbacks=[cb_checkpointer,cb_early_stopper]
history=model.fit_generator(x_train,steps_per_epoch=TRAIN_STEPS_PER_EPOCH,epochs=20,
                            validation_data=x_test,validation_steps=VAL_STEPS_PER_EPOCH)

from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

y_pred=model.predict(x_test)
threshold=0.5
y_pred_threshold=[(1 if val>threshold else 0) for val in y_pred]
y_true=x_test.labels.tolist()
print(confusion_matrix(y_true,y_pred_threshold))
print(accuracy_score(y_true,y_pred_threshold))
sns.heatmap(confusion_matrix(y_true,y_pred_threshold),annot=True)

y_pred = model.predict(x_test, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
#y_true=x_test.labels.tolist()
```

```
225
226
227
228 print(classification_report(y_true, y_pred_bool))
229
230 print(confusion_matrix(y_true, y_pred=y_pred_bool))
231
232 print(classification_report(y_true, y_pred_bool))
233
234 keys=history.history.keys()
235 print(keys)
236
237 plt.figure(1,figsize=(15,8))
238 plt.subplot(221)
239 plt.plot(history.history['accuracy'])
240 plt.plot(history.history['val_accuracy'])
241 plt.title('Model_Accuracy')
242 plt.ylabel('Accuracy')
243 plt.xlabel('Epoch')
244 plt.legend(['train','valid'])
245
246 plt.figure(1,figsize=(15,8))
247 plt.subplot(222)
248 plt.plot(history.history['loss'])
249 plt.plot(history.history['val_loss'])
250 plt.title('Model_loss')
251 plt.ylabel('loss')
252 plt.xlabel('Epoch')
253 plt.legend(['train','valid'])
254
255 for key in keys:
256   plt.figure(figsize=(15,2))
257   plt.plot(history.history[key])
258   plt.title(key)
259   plt.ylabel(key)
260   plt.xlabel('epoch')
261   plt.legend(['train','valid'])
262
263 model.save("best.hdf5")
264
265 from tensorflow import keras
266 from tensorflow.keras import layers
267 from keras.models import load_model
268 model=load_model('/content/best.hdf5')
269
270 from keras.preprocessing import image
271 import cv2
272 import numpy as np
273 import PIL
274 from PIL import Image
275
276 img = image.load_img('/content/drive/MyDrive/Data/Malignant cases/Malignant case (104).jpg',
277                      target_size=(224,224))
278
279 x=image.img_to_array(img)
280 x=np.expand_dims(x,axis=0)
281 x.shape
282
283 pred = model.predict(x)
284 pred
285
286 class_names=["Bengin cases","Malignant cases"]
287 prediction = class_names[int(pred[0][0])]
288 print(pred[0][0])
289 print(prediction)
```

```
290
291 from flask import Flask, request, jsonify
292 from tensorflow.keras.preprocessing import image
293 import numpy as np
294
295 # Load the saved model
296 model = load_model('/content/best.hdf5')
297
298 # Define a Flask app
299 app = Flask(__name__)
300
301 # Define a route for predicting the uploaded images
302 @app.route('/predict', methods=['POST'])
303 def predict():
304     # Get the uploaded file from the request
305     file = request.files['image']
306
307     # Load the image and preprocess it
308     img = image.load_img(file, target_size=(224, 224))
309     x = image.img_to_array(img)
310     x = np.expand_dims(x, axis=0)
311     x = preprocess_input(x)
312
313     # Make a prediction
314     preds = model.predict(x)
315
316     # Get the class label
317     label = decode_predictions(preds, top=1)[0][0][1]
318
319     # Return the prediction result
320     return jsonify({'class': label})
321
322 if __name__ == '__main__':
323     app.run(debug=True)
```

Inception V3

```
88
89 import os
90 import random
91 from shutil import copyfile
92
93 split_size = .80
94
95 split_data(BENGIN_SOURCE_DIR, TRAINING_BENGIN_DIR, VALID_BENGIN_DIR, split_size)
96 split_data(MALIGNANT_SOURCE_DIR, TRAINING_MALIGNANT_DIR, VALID_MALIGNANT_DIR, split_size)
97
98 import matplotlib.pyplot as plt
99 import seaborn as sns
100 from matplotlib.image import imread
101 import pathlib
102
103 image_folder = ['Bengin cases', 'Malignant cases']
104 nimgs = {}
105 for i in image_folder:
106     nimages = len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_inception_V3_1/train/'+i+'/'))
107     nimgs[i]=nimages
108 plt.figure(figsize=(9, 6))
109 plt.bar(range(len(nimgs)), list(nimgs.values()), align='center')
110 plt.xticks(range(len(nimgs)), list(nimgs.keys()))
111 plt.title('Distribution of different classes in Training Dataset')
112 plt.show()
113
114 for i in ['Bengin cases', 'Malignant cases']:
115     print('Training {} images are:
'.format(i)+str(len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_inception_V3_1/train/'+i+'/'))))
116
117 for i in ['Bengin cases', 'Malignant cases']:
118     print('Testing {} images are:
'.format(i)+str(len(os.listdir('/content/drive/MyDrive/R_Data/Split_data_inception_V3_1/validation/'+i+'/'))))
```

```
!pip install tensorflow==2.7.0

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import sequential
from tensorflow.keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import Flatten

import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=8,
                                   zoom_range=0.15,
                                   width_shift_range=0.15,
                                   height_shift_range=0.15,
                                   shear_range=0.10,
                                   horizontal_flip=True,
                                   brightness_range=[0.8,1.2],
                                   channel_shift_range=0.15,
                                   fill_mode="nearest")

test_datagen = ImageDataGenerator(rescale=1./255)

x_train = train_datagen.flow_from_directory('/content/drive/MyDrive/R_Data/Split_data_inception_V3_1/train',target_size=
(299,299),batch_size=32,class_mode='binary')

x_test = test_datagen.flow_from_directory('/content/drive/MyDrive/R_Data/Split_data_inception_V3_1/validation',target_size=
(299,299),batch_size=32,class_mode='binary')

print(x_train.class_indices)
print(x_test.class_indices)

import tensorflow.python.keras.applications

from tensorflow.keras.applications.inception_v3 import InceptionV3

from keras.models import Sequential

model = Sequential()
model.add(Convolution2D(32, (3, 3), input_shape=(224, 224, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(units=512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
model.summary()

base_model=tf.keras.applications.inception_v3.InceptionV3(
    include_top=False,
    weights='imagenet',
    input_tensor=None,
    input_shape=(299,299,3),
    pooling=None,
    classes=1000,
    classifier_activation='softmax'
)
for layer in base_model.layers:
    layer.trainable=False

x=layers.Flatten()(base_model.output)
x=layers.Dense(512,activation='relu')(x)

x=layers.Dropout(0.5)(x)
x=layers.Dense(1,activation='sigmoid')(x)
model=tf.keras.models.Model(base_model.input,x)

model.compile(loss="binary_crossentropy",optimizer='adam',metrics=['accuracy'])

TRAIN_COUNT =len(x_train.filepaths)
TEST_COUNT=len(x_test.filepaths)
TRAIN_STEPS_PER_EPOCH= round(TRAIN_COUNT/32)
VAL_STEPS_PER_EPOCH = round(TEST_COUNT/32)

print(f"TRAIN_STEPS_PER_EPOCH:{TRAIN_STEPS_PER_EPOCH}")
print(f"VAL_STEPS_PER_EPOCH:{VAL_STEPS_PER_EPOCH}")

from tensorflow.python.keras.callbacks import EarlyStopping,ModelCheckpoint
EARLY_STOP_PATIENCE =5

cb_early_stopper =EarlyStopping(monitor='val_losss',patience=EARLY_STOP_PATIENCE)
cb_checkpointer = ModelCheckpoint(filepath='Inception_v3.hdf5',monitor='val_loss',save_best_only=True,mode='auto')
callbacks=[cb_checkpointer,cb_early_stopper]
history=model.fit_generator(x_train,steps_per_epoch=TRAIN_STEPS_PER_EPOCH,epochs=20,
                            validation_data=x_test,validation_steps=VAL_STEPS_PER_EPOCH)

model.save("/content/drive/MyDrive/Data_O/OOO/Cancer/Inception_v3.hdf5")

from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix

y_pred=model.predict(x_test)
threshold=0.5
y_pred_threshold=[(1 if val>threshold else 0) for val in y_pred]
y_true=x_test.labels.tolist()
print(confusion_matrix(y_true,y_pred_threshold))
print(accuracy_score(y_true,y_pred_threshold))
sns.heatmap(confusion_matrix(y_true,y_pred_threshold),annot=True)

y_pred = model.predict(x_test, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
#y_true=x_test.labels.tolist()



print(classification_report(y_true, y_pred_bool))

print(confusion_matrix(y_true, y_pred=y_pred_bool))

keys=history.history.keys()
print(keys)

plt.figure(1,figsize=(15,8))
plt.subplot(221)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model_Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','valid'])

```

```
plt.figure(1,figsize=(15,8))
plt.subplot(221)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model_Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','valid'])

plt.figure(1,figsize=(15,8))
plt.subplot(222)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model_loss')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['train','valid'])

plt.show()

for key in keys:
  plt.figure(figsize=(15,2))
  plt.plot(history.history[key])
  plt.title(key)
  plt.ylabel(key)
  plt.xlabel('epoch')
  plt.legend(['train','valid'])

plt.show()

from tensorflow import keras
from tensorflow.keras import layers
from keras.models import load_model
model=load_model('/content/drive/MyDrive/Data_O/OOO/Cancer/Inception_v3.hdf5')

from keras.preprocessing import image
import cv2
import numpy as np
import PIL
from PIL import Image

img = image.load_img('/content/drive/MyDrive/Data_O/OOO/Cancer/Split_data_inception_V3/validation/Malignant cases/Malignant case
(170).jpg',target_size=(299,299))

x=image.img_to_array(img)
x=np.expand_dims(x,axis=0)
x.shape

pred = model.predict(x)
pred

class_names=["Bengin cases","Malignant cases"]
prediction = class_names[int(pred[0][0])]
print(pred[0][0])
print(prediction)

```