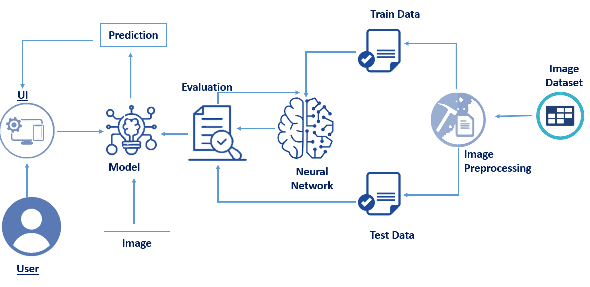
**Pathology Image Analysis of Lung Cancer Prediction Using IBM Watson**

**Intoduction to Project:**

Lung cancer is one of the most killerdiseases in the developing countries and the detection of the cancer at the early tage is a challenge. Analysis and cure of lung malignancy have been one of the greatest difficulties faced by humans over the most recent couple of decades. Early identification of tumor would facilitate in sparing a huge number of lives over the globe consistently. This project presents lung cancer detection based on chest CT images using CNN. In the first stage, lung regions are extracted from the CT images. The extracted images are used to train the CNN architecture. Then, CNN is used to test the patient images. The main objective of this study is to detect whether the tumor present in a patient’s lung is malignant or benign. A web application is built which gives the feasibility of uploading a CT scan image. This app is integrated with a model. The model detects the tumor and the prediction is shown on the UI.  The main objective of this project is to detect whether the tumor present in a patient’s lung is malignant or benign using Convolution Neural Network (CNN). In a study published in Nature Medicine, researchers said that lung cancer caused an estimated 160,000 deaths in 2018, making it the most common cause of cancer death in the US. Lung cancer screenings that use low-dose tomography have been shown to reduce mortality by 20-43 percent, but there are still challenges that result in unclear diagnoses, subsequent unnecessary procedures, and high costs. Radiologists also usually have to look through dozens of 2D images within a single CT scan, and cancer can be hard to spot. Deep learning can offer a viable solution to these problems.

**Technical Architecture:**

**Prerequisites**   **:**

To complete this project you should have the following software  and packages

**Anaconda Navigator :**

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform,  package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupiter notebook and spyder

To install Anaconda navigator and to know how to use Jupyter Notebook a Spyder using Anaconda watch the video: <https://youtu.be/5mDYijMfSzs> .

To build Deep learning models you must require the following packages

**Tensor flow:** TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML powered applications.

**Keras :** Keras leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features:

Consistent, simple and extensible API.

Minimal structure - easy to achieve the result without any frills.

It supports multiple platforms and backends.

It is user friendly framework which runs on both CPU and GPU.

Highly scalability of computation.

Flask: Web frame work used for building  Web applications

Watch the video to Install the necessary Packages: <https://youtu.be/akj3_wTploU>.

**Prior Knowledge**

You should know the following concepts

* Basics of Python
* Convolution Neural Network
* Artificial Neural Network
* OpenCV

**Project Objectives**

By the end of this project you will:

* Get to know about image processing.
* Understand Deep learning Architecture.
* Understand the concepts like Artificial neural network ana Convolution neural network.
* Classify images using a Convolutional Neural Network.
* How to build a web application.

**Project Flow:**

* User interacts with User interface to upload image
* Uploaded image is analyzed by the model which is integrated
* Once model analyses the uploaded image, the prediction is showcased on the UI

To accomplish this, we have to complete all the activities and tasks listed below

* Data Collection.

1. Collect the dataset or Create the dataset

* Data Preprocessing.

1. Import the ImageDataGenerator library

2. Configure ImageDataGenerator class

3. Apply ImageDataGenerator functionality to Trainset and Testset

* Model Building

1. Import the model building Libraries

2. Initializing the model

3. Adding Input Layer

4. Adding Hidden Layer

5. Adding Output Layer

6. Configure the Learning Process

7. Training and testing the model

8. Optimize the Model

9. Save the Model

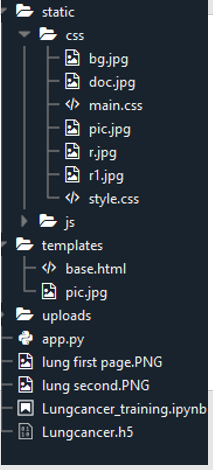
* Application Building

1. Create an HTML file

2. Build Python Code

**Project Structure**

create the folders and place all the required files as shown below



* A model file which has the trained data Lungcancer.h5 file
* A jupyter notebook file to train the machine
* app.py is the server-side scripting file
* and HTML files saved in the templates folder
* Images and js files required for the web page are stored in a static folder.

**Data Collection**

Artificial Intelligence is a data hunger technology, it depends heavily on data, without data, it is impossible for a machine to learn. It is the most crucial aspect that makes algorithm training possible. In Convolutional Neural Networks, as it deals with images, we need training and testing data set. It is the actual data set used to train the model for performing various actions.

**Download The Dataset**

In this section, we will be collecting data for building our project. We will be creating two folders one for training and the other for testing. Images present in the training folder will be used for building the model and the testing images will be used for validating our model.

Down the Dataset from the link given below [Dataset](https://wiki.cancerimagingarchive.net/display/Public/RIDER+Lung+CT)

Follow the instructions given on the above page to download the images. Refer to the reference link given below to organize your downloaded images into train and test folders.

**Image Pre Processing**

Now that we have all the data collected, let us use this data to train the model . before training the model you have to preprocess the images and then feed them on to the model for training. we make use of Keras ImageDataGenerator  class for image preprocessing.

For more info about image preprocessing please click on the below link [data Augmentation](https://blog.roboflow.com/why-preprocess-augment/)

Image Pre-processing includes the following main tasks

* Import ImageDataGenerator Library.
* Configure ImageDataGenerator Class.
* Applying ImageDataGenerator functionality to the trainset and test set.

Note: The ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.

Open Jupyter notebook and create a new python file, name it Lungcancer\_training and save it in the project folder. To know more about the usage  of the Jupyter notebook watch the video given in the pre-requisites section

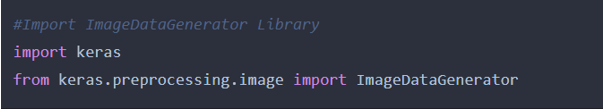
**Importing The ImageDataGenerator Library**

Open created jupyter notebook and  start writing the program

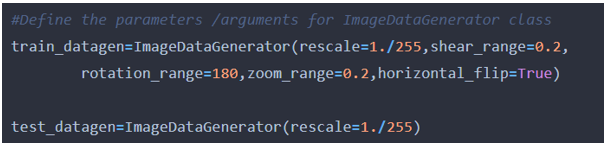
The first step is usually importing the libraries that will be needed in the program.

Import Keras library from that library import the ImageDataGenerator Library to your Python script:

After each code block in this tutorial, you should type ALT + ENTER to run the code and move into a new code block within your notebook.



Define the parameters /arguments for ImageDataGenerator class



Here the arguments which we are given inside the image data generator class is like, rescale, shear\_range, rotation range of the image, and zoom range that we can consider for the image, etc.

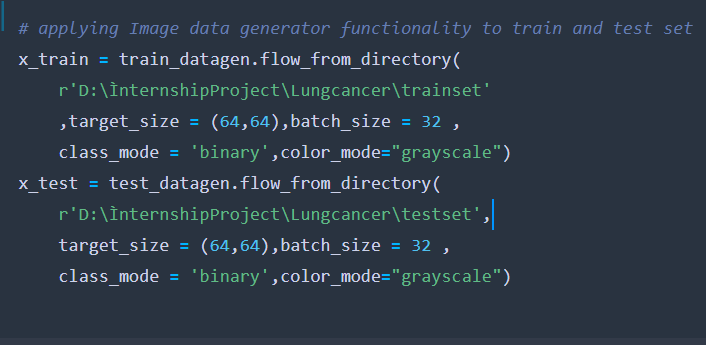
* Image shifts via the width\_shift\_range and height\_shift\_range arguments.
* The image flips via the horizontal\_flip and vertical\_flip arguments.
* The image rotates via the rotation\_range argument

**Applying ImageDataGenerator Functionality To Trainset And Testset**

From the above step, we have initialized the preprocessing features, now let's apply those features to the dataset collected.

The ImageDataGenerator class has three methods flow ( ), flow\_from\_directory ( ), and flow\_from\_dataframe ( ) to read the images from a big NumPy array and folders containing images.

Apply flow\_from\_directory ( ) method for Train folder and test Folder



* The directory must be set to the path where your training folders are present.
* The target\_size is the size of your input images, every image will be resized to this size.
* batch\_size: No. of images to be yielded from the generator per batch.
* “batch\_size” in both train and test generators is to some number that divides your total number of images in your train set and train set respectively.
* class\_mode: Set “binary” if you have only two classes to predict, if not set to “categorical”.

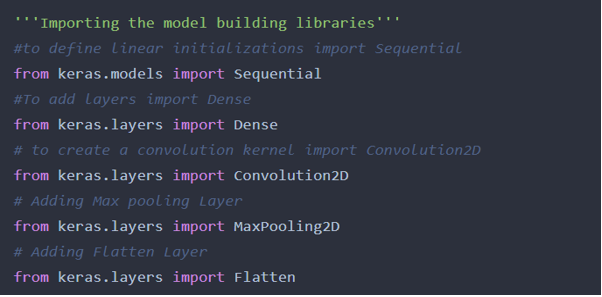
**Model Building**

We are ready with the augmented and pre-processed image data, Lets begin our model building, this activity includes the following steps

* Import the model building Libraries
* Initializing the model
* Adding CNN Layers
* Adding Hidden Layer
* Adding Output Layer
* Configure the Learning Process
* Training and testing the model
* Saving the model

**Import Libraries**

This is a very crucial step in our deep learning model building process. We have to define how our model will look and that requires.



**Initializing The Model**

Keras has 2 ways to define a neural network:

* Sequential
* Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add () method. Now, will initialize our model.



**Add CNN Layers**

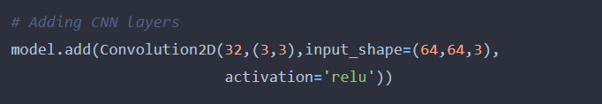
We will be adding three layers for CNN

* Convolution layer
* Pooling layer
* Flattening layer

**Adding 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.

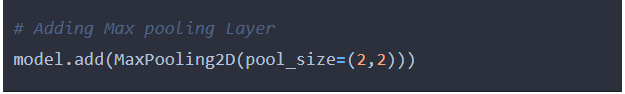
In the Convolution operation, the input image will be convolved with the feature detector/filters to get a feature map. The important role of the feature detector is to extract the features from the image. The group of feature maps is called a feature layer.



* In the convolution2D function, we have given arguments like, 32,(3,3), which means we are applying 32 filters of 3x3 matrix filter, and input\_shape is the input image shape with RGB, here 64x64 is the size and 3 represent the channel, RGB colour images.
* And 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.

**Adding Pooling Layer**

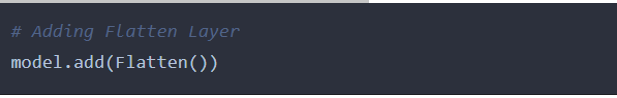
Pooling reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer. It keeps only the necessary details. Pooling is a technique in CNN that helps us to avoid overfitting of data, spatial invariance, and distortion.  After applying max-pooling we will get another feature map called Pooled Feature Map. Now will apply the pooling technique.



In the above code pool\_size is the pooling filter or kernel size.

**Adding Flatten 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.



**Adding Dense Layers**

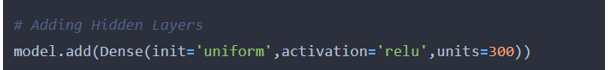
Now, let's add Dense Layers to know more about dense layers click below

[Denselayers](https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/)

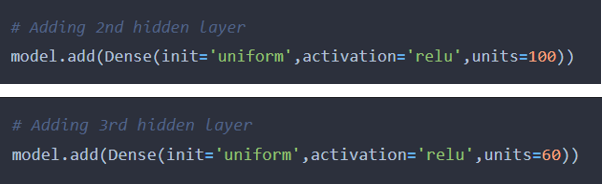
The name suggests that layers are fully connected (dense) by the neurons in a network layer. Each neuron in a layer receives input from all the neurons present in the previous layer. Dense is used to add the layers.

**Adding Hidden layers**

This step is to add a dense layer (hidden 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.

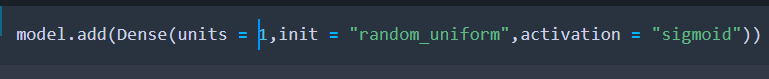


* init is the weight initialization; function which sets all the weights and biases of a network to values suitable as a starting point for training.
* units/ output\_dim, which denote is the number of neurons in the hidden layer.
* The activation function basically decides to deactivate neurons or activate them to get the desired output. It also performs a nonlinear transformation on the input to get better results on a complex neural network.
* You can add many hidden layers, in our project we are added more two hidden layers. The 2nd hidden layer with 100 neurons and 3rd hidden layer with 60 neurons.



**Adding the output layer**

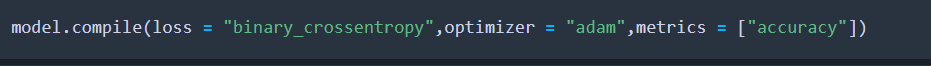
This step is to add a dense layer (output layer) where you will be specifying the number of classes your dependent variable has(in our case 1), activation function, and weight initializer as the arguments. We use the add () method to add dense layers.



**Note:** if you have only one or two class in output layer , assign “units= 1” and “activation = sigmoid”. If you have more than two classes (let's assume 3 ) then assign  “units= 3” and “activation = softmax”.

**Configuring The Learning Process**

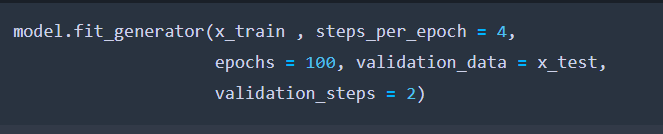
With both the training data defined and model defined, it's time to configure the learning process. This is accomplished with a call to the compile () method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics.



**Note:** In our project, we have 2 classes in the output, so the loss is binary\_crossentropy.

If you have more than two classes in output put “loss = categorical\_cross entropy”.

**Training The Model**



At this point, we have training data and a fully configured neural network to train with loaded data. All that is left is to pass the data to the model for the training process to commence, a process that is completed by iterating on the training data. Training begins by calling the fit ( ) method.

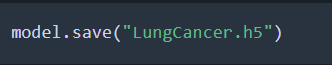
The arguments are the batch size as you are using “adam” (batch gradient descent and epochs: no: of times the model should get trained).

steps\_perepoch = number of training images/batch size

validation steps = no:of test images/32

**Save The Model**

Your model is to be saved for future purposes. This saved model can also be integrated with an android application or web application in order to predict something.



**Application Building**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has uploaded an image. The uploaded image is given to the saved model and prediction is showcased on the UI.

This section has the following tasks

* Building HTML Pages
* Building server-side scrip

**Build HTML Code**

For more information regarding HTML

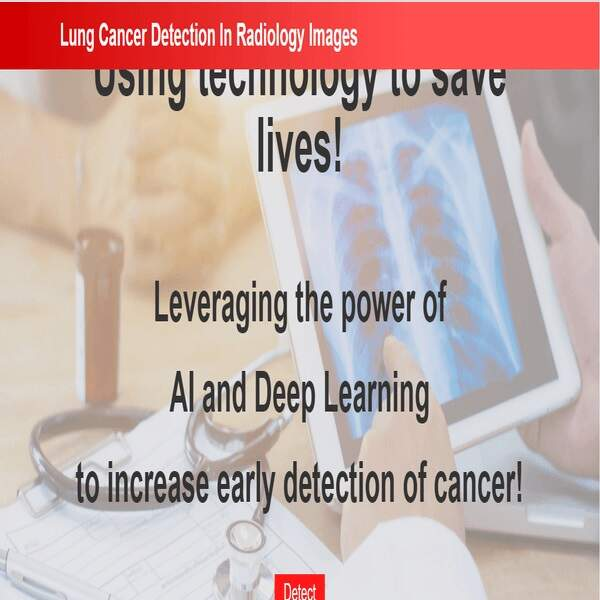
https://www.w3schools.com/html/

In our project, we have 1 file, which is

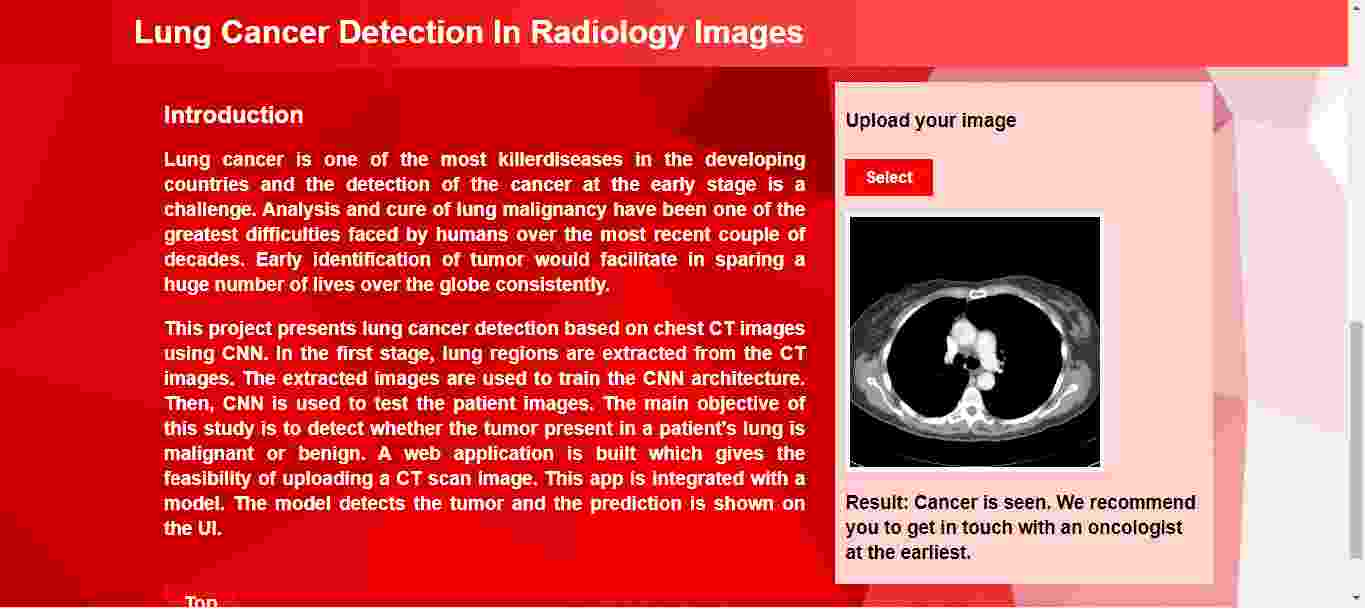
base.html

Please download the HTML files from the link Given

Let's look at our  Web page



Detect Pages Looks Like:



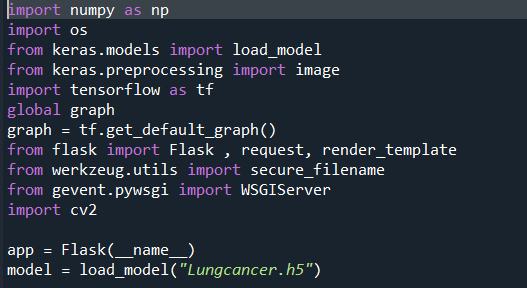
Flask app reference :

<https://www.youtube.com/watch?v=lj4I_CvBnt0>

**Build Python Code**

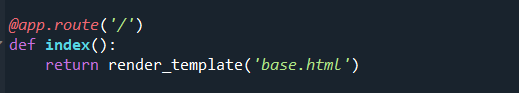
We will be using python for server-side scripting. Let’s see step by step process for writing backend code. Create a python file using spyder and name it as app.py

Importing Libraries

Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (\_\_name\_\_) as an argument

Load all the necessary libraries and files used for detection disease from the uploaded image

**Render The HTML Page**



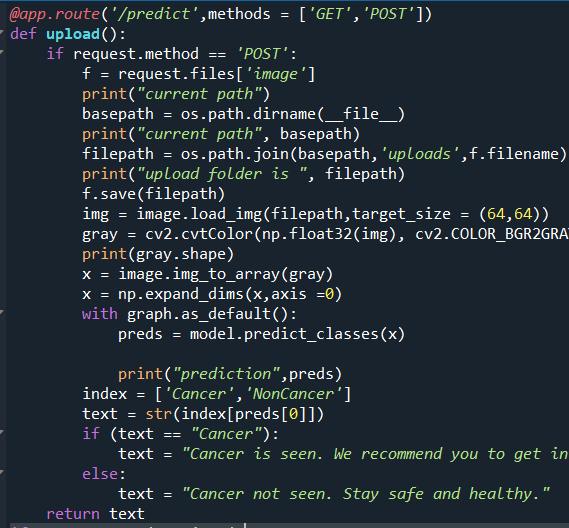
Here we will be using declared constructor to route to the HTML page which we have created earlier. in place of HTML use base.html

**Bind the URL to perform some action**

In the below function,  making retrieving the image uploaded from the web page

Saving the uploaded image in the Uploads folder

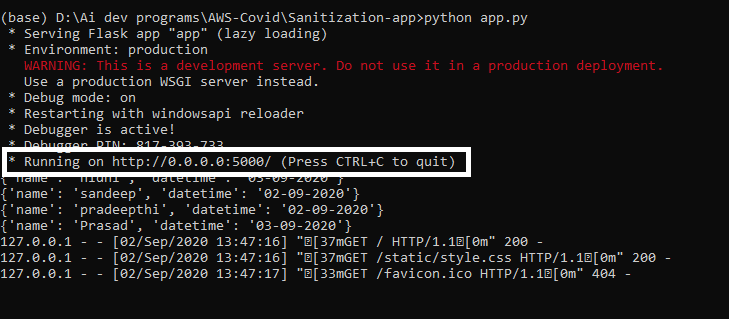
Loading the saved file and analyzing it using a model built

returning the prediction on to the page



**Run The App**

* Open anaconda prompt/command  from the start menu
* Navigate to the folder where your app.py resides
* Now type “python app.py” command
* It will show the local host where your app is running.
* Navigate to the localhost where you can view your web page



**Train The Model On IBM**

In this milestone, you will learn how to build DeepLearning Model Using the IBM cloud.

**Register For IBM Cloud**

To complete this project you must have an IBM account and open weather Account

**IBM Account:**

Please click [here](https://cloud.ibm.com/registration) to register for IBM

Please click [here](https://cloud.ibm.com/login) to log in to IBM Account

Watch the below video to register and login into your IBM account<https://youtu.be/4y_zD-0Q3F8>

**Train The Model On IBM Watson**

Please watch the below video to train the model on IBM  and integrate it with the flask Application <https://youtu.be/BzouqMGJ41k> .