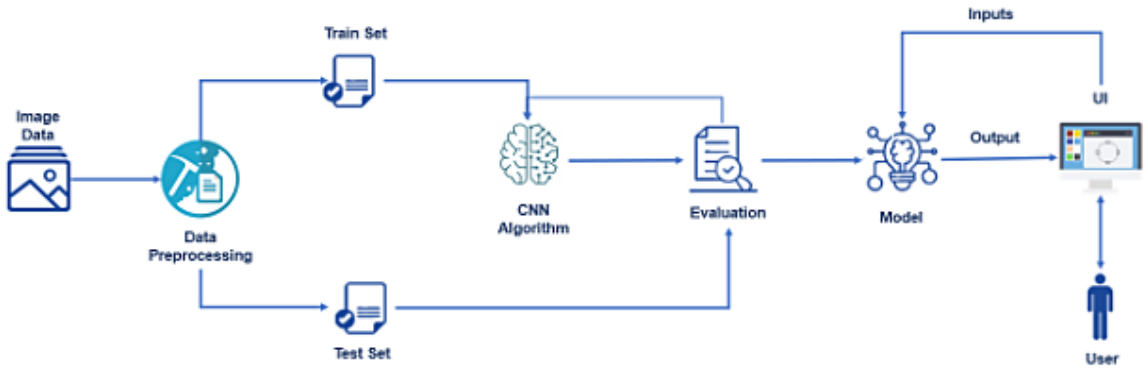
# Pneumonia Detection Using X-Rays Using IBM Watson Studio

**Project Description:**

The risk of pneumonia is immense for many, especially in developing nations where billions face energy poverty and rely on polluting forms of energy. Over 150 million people get infected with pneumonia on an annual basis especially children below 5 years. A patient suffering from Pneumonia takes an X-ray image to the doctor; with them he predicts pneumonia. The results are not just based on seeing the X-ray images, furthermore, tests will be conducted on the patient. The process was time-consuming, but in recent days artificial intelligence helps in predicting pneumonia bypassing the X-ray image. The main objective of this project is to help the doctors to predict the pneumonia disease more accurately using a deep learning model. The objective is not only to help the doctors but also to the patients to precisely predict pneumonia.

**Technical Architecture:**




**Pre requisites:**

**To complete this project, you must require following software's, concepts and packages**

- **Anaconda navigator:**
  - Refer to the link below to download anaconda navigator
  - **Link : https://www.youtube.com/watch?v=5mDYijMfSzs**
- **Python packages:**

  Open anaconda prompt as administrator.

- Type “pip install numpy” and click enter.
- Type “pip install pandas” and click enter.
- Type “pip install matplotlib” and click enter.
- Type “pip install scikit-learn” and click enter.
- Type “pip install Flask” and click enter.

The above steps allow you to install the packages in the anaconda environment

- **Launch Jupyter**
  - Search  for Anaconda Navigator and open Launch Jupyter notebook.

- Then you will be able to see that the jupyter notebook runs on local host:8888.
- To Create a new file Go to New àPython3.The file in jupyter notebook is saved with .ipynb extension.

  - Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

**Project Objectives:**

By the end of this project:

- You’ll be able to understand the problem to classify if it is a regression or a classification kind of problem.
- You will be able to know how to process the images
- You will know the fundamental concepts of Artificial Neural Networks, Convolution

Neural Network

- You will be able to know how to find the accuracy of the model.
- You will be able to build web applications using the Flask framework.

**Project Flow:**

- Download the dataset.
- Classify the dataset into train and test sets.
- Add the neural network layers.
- Load the trained images and fit the model.
- Test the model.
- Save the model and its dependencies.
- Build a Web application using flask that integrates with the model built.

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

- Data Collection.
  - Download the dataset
- Data Preprocessing.
    - Import the Required Libraries.
    - Configure Image Data Generator Class.
    - Apply Image Data Geberator functionality to Train,Test and Validation dataset
- Model Building
    - Initialize the model

    - Add CNN and Dense layers
    - Add Dense Layers
    - Configure the Learning process
    - Fit and Save the model
    - Test the model
- Application Building
    - Create an HTML file
    - Build Python Code
    - Run the App
- Train the model on IBM
    - Register for IBM Cloud
    - Train the model on IBM Watson

**Project Structure:**

Create a Project folder which contains files as shown below

- A python file called app1.py for server side scipting.
- We need the model which is saved and the saved model in this content is **pneumonia.h5**
- Templates folder which contains index.HTML file.
- Static folder which contains css folder which contains styles.css.

**Milestone 1: 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. In this activity lets focus of gathering the dataset.

**Activity1: Download The dataset**

You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository, etc.

You can refer to **https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia** for downloading the dataset.

**Milestone 2: Image Preprocessing**

Image Pre-processing includes the following main tasks

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

**Activity 1: Import Necessary Libraries**

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from Keras

```
# Importing LIBRARIES
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
import os
import numpy as np
import pandas as np
```

**Activity 2: Configure Image Data Generator Class**
**ImageDataGenerator class is instantiated and the configuration for the types of data augmentation**

**There are five main types of data augmentation techniques for image data; specifically:**

- 
- **Image shifts via the width_shift_range and height_shift_range arguments.**
- **Image flips via the horizontal_flip and vertical_flip arguments.**
- **Image rotates  via the rotation_range argument**
- **Image brightness via the brightness_range argument.**
- **Image zooms via the zoom_range argument.**

**An instance of the ImageDataGenerator class can be constructed for train and test.**

```
# Specifying the path of the data(train,test,validaton)

train = 'chest_xray/train'
test = 'chest_xray/test'
val = 'chest_xray/val'
```

```
img_width,img_height= 150,150
input_shape = (img_width,img_height,3)
```

```
#ImageDataGenerator-Generate batches of tensor image data with real-time data augmentation.
#The data will be looped over (in batches).
batch_size = 16
train_datagen = ImageDataGenerator(rescale=1. / 255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1. / 255)
```

**Activity 3:** Apply ImageDataGenerator functionality to Train, Test, and Validation set

```
# Here we import images directly from Directory by using flow_from_directory method.
#flow_from_directory() automatically infers the labels from the directory structure of the folders containing images
train_generator = train_datagen.flow_from_directory(
    train,
    target_size=(img_width, img_height),
    batch_size=16,
    class_mode='binary')

test_generator = test_datagen.flow_from_directory(
    test,
    target_size=(img_width, img_height),
    batch_size=16,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    val,
    target_size=(img_width, img_height),
    batch_size=16,
    class_mode='binary')
```

```
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
```

We can see that for training there are 5216 images belonging to 2 classes and for testing there are 624 images belonging to 2 classes.

**Arguments:**

- directory: Directory where the data is located. If labels is "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data. Default: 32.
- target_size: Size to resize images to after they are read from disk.
- class_mode:
    - ‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).
    - 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
    - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
    - None (no labels).

**Milestone 3: Model Buiding:**

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

**Activity 1: Initialize the Model**

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.

```
model=Sequential()
```

**Activity 2: Add CNN and Dense Layers**

The first layer of the neural network model, the convolution layer will be added. To create a convolution layer, Convolution2D class is used. It takes a number of feature detectors, feature detector size, expected input shape of the image, activation function as arguments. This layer applies feature detectors on the input image and returns a feature map (features from the image).

Activation Function: These are the functions which help us to decide if we need to activate the node or not. These functions introduce non-linearity in the networks.

```
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
```

Add the pooling layer

Max Pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

After the convolution layer, a pooling layer is added. Max pooling layer can be added using

MaxPooling2D class. It takes the pool size as a parameter. Efficient size of the pooling matrix is (2,2). It returns the pooled feature maps. (Note: Any number of convolution layers, pooling and dropout layers can be added)

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Add the flatten layer

The flatten layer is used to convert n-dimensional arrays to 1-dimensional arrays. This 1D array will be given as input to ANN layers.

```
model.add(Flatten())
```

Three dense layers are added which usually takes number of units/neurons. Specifying the activation function, kind of weight initialization is optional.

```
model.add(Dense(output_dim = 40 ,init = 'uniform',activation = 'relu'))
```

```
model.add(Dense(output_dim = 1,activation = 'softmax',init ='uniform'))
```

Note: Any number of convolution, pooling and dense layers can be added according to the data.

```
# The number of filters are 32 and the kernal_size is (3,3)

model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dropout(50))
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(50))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(50))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

Activity 3: Configure the Learning Process

After adding all the required layers, the model is to be compiled. For this step, loss function, optimizer and metrics for evaluation can be passed as arguments.

```
#Here we use RMSPROP optimizer and BINARY_CROSSENTROPY as loss function
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to training phase. Loss function is used to find error or deviation in the learning process. Keras requires loss function during model compilation process.
- Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process

Activity 4: Fit and Save the Model

Fit the neural network model with the train and test set, number of epochs and validation steps.

```
#We Fit the model here using fit_generator as we are dealing with large datasets.
model.fit_generator(
    train_generator,
    steps_per_epoch=5217 // 16,
    epochs=20,
    validation_data=validation_generator,
    validation_steps=17 // 16)
```

Accuracy, Loss: Loss value implies how poorly or well a model behaves after each iteration of optimization. An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage.

The weights are to be saved for future use. The weights are saved in as .h5 file using save().

```
# saving model in H5 format.
model.save('pnemonia.h5')
```

model.summary() can be used to see all parameters and shapes in each layer in our models.

Activity 5: Test the Model

The model is to be tested with different images to know if it is working correctly.

Follow the steps to tset the model

Import the packages and load the saved model

```
from keras.models  import load_model
from keras.preprocessing import image
import numpy as np
```

```
model = load_model('pneumonia.h5')
```

Load the test image, pre-process it and predict

Pre-processing the image includes converting the image to array and resizing according to the model. Give the pre-processed image to the model to know to which class your model belongs to.

```
from skimage.transform import resize

def detect(frame):
    try:
        img = resize(frame,(150,150))
        img = np.expand_dims(img,axis=0)
        prediction = model.predict(img)
        print(prediction)
        prediction = model.predict_classes(img)
        print(prediction)
    except AttributeError:
        print("shape not found")

frame=cv2.imread("test.jpg")
data = detect(frame)
```

```
[[0.98931956]]
[[1]]
```

**Milestone 4 : 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 users where he has to enter the imsge. The test 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 script

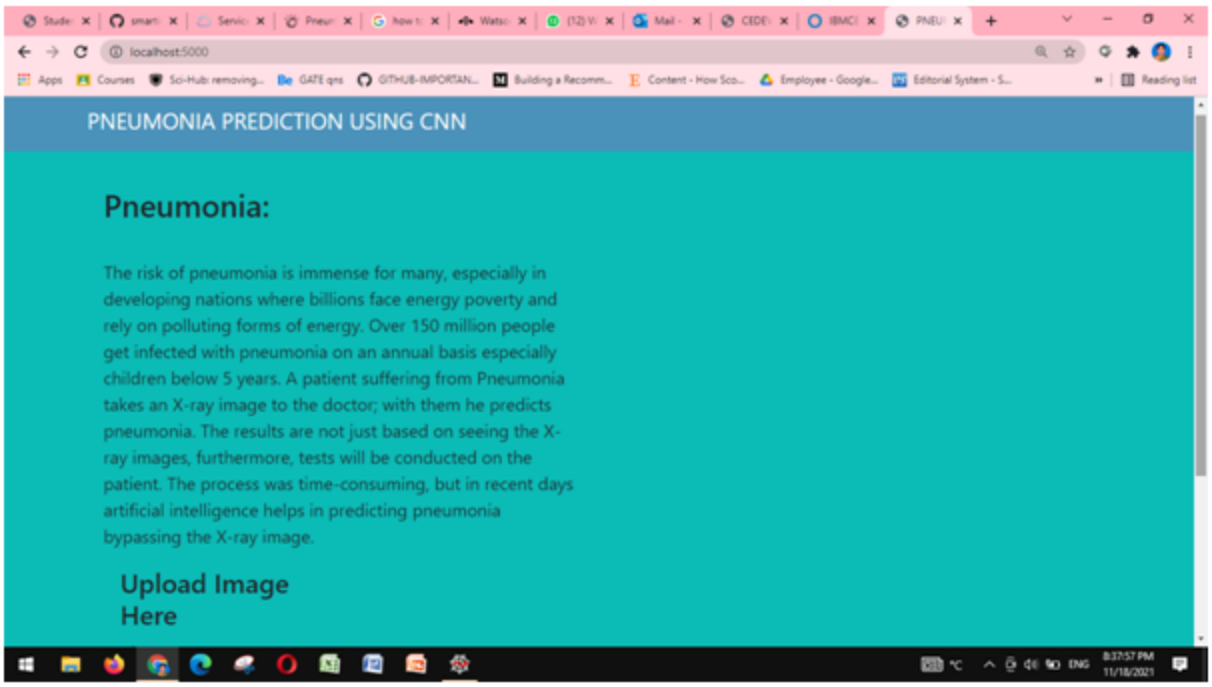
**Activity 1: Build HTML Code**

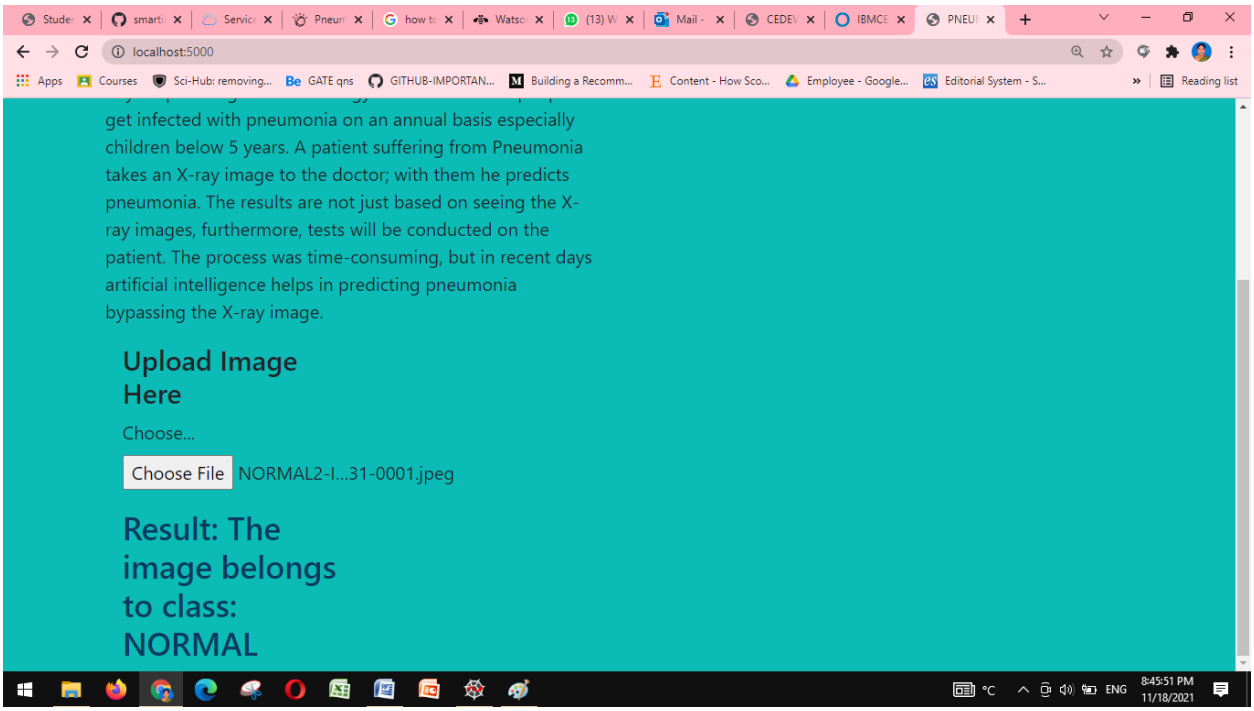
- In this HTML page, we will create the front end part of the web page. In this page we will the test image from the user and Predict whether the person has Pneumonia or Normal.

  For more information regarding HTML
  https://www.w3schools.com/html/
  In our project we have 1 HTML file
  1.Index.html

**Demo.html**

**The html page looks like**






**Activity 2: Main Python Script**

Let us build app1.py flask file which is a web framework written in python for server-side scripting.

**Activity 3: Run the App**

- Open anaconda prompt  from the start menu
- Navigate to the folder where your python script is.
- Now type “python app1.py” command
- Navigate to the localhost where you can view your web page

Then it will run on local host:5000