A picture containing table, elephant, clothing

Description automatically generated

Team Image-ination

*presents*

Digital Image Processing for Breast Cancer Screening

Oracle Consulting

1.0

# Document Control

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Description |
| 26/10/2021 | Mohammad Arshad | V1.0 |  |
|  |  |  |  |

**Note:** This style can be used to write usage notes

It’s hidden for printing and can be removed in 5 seconds from all document by using “select all” on the style menu for this style and then deleting.

Contents

[Document Control 2](#_Toc86338442)

[1.Introduction 4](#_Toc86338443)

[2. Solution and Architecture 5](#_Toc86338444)

[3. Deep Learning Model 6](#_Toc86338445)

[3.1. Model Architecture with ResNet50 6](#_Toc86338446)

[3.2. Image Augmentation 7](#_Toc86338447)

[3.3. Hyper parameter settings 7](#_Toc86338448)

[3.4. Training Results 7](#_Toc86338449)

[3.5. Test Results 7](#_Toc86338450)

[4. User Interface 9](#_Toc86338451)

[5. Pre-Requisites and Skills required 11](#_Toc86338452)

[5.1. Perquisites. 11](#_Toc86338453)

[5.2. Technology Used 11](#_Toc86338454)

[6. Python Flask Webservice 12](#_Toc86338455)

[7. Code Setup and File details 13](#_Toc86338456)

# 1.Introduction

Capturing, processing and analysing Images is a widely prevalent practice across multiple industries. For eg. In the Retail Industry you can use it to track and replenish empty shelves. There are lots of uses in security and traffic management as well where images are analysed to identify security or traffic rule breaches.

Our use case is specific to the Healthcare Industry and our code can be used to read MRI images to detect the possibility of cancerous cells. Typically, this is done manually which makes it subject to the pitfalls of human errors, delays, and individual perceptions.

The algorithms in our model are trained with historical data which is nothing but lots of past images where the results or diagnosis for a specific disease is already known so the model learns to read the images from these training images and can then apply this learning to read new images and provide a classified output related to diagnosis of a given disease, in our case, Breast Cancer.

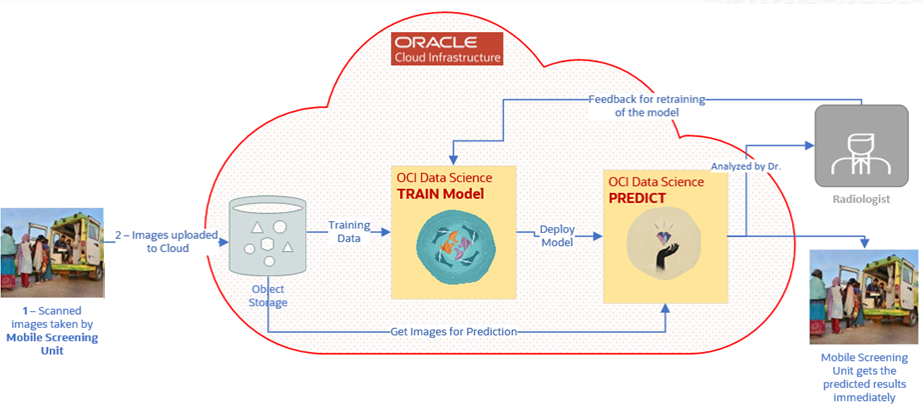
Breast Cancer is the one of most prevailing and undetected cancers among women and in places with limited or expensive medical facilities and doctors, a solution like this can automate part of the process and early diagnosis could reduce the risk of the disease advancing and subsequently, of poor outcomes.

Note that this is primarily used for first stage screening and we are not trying to replace the expertise of doctors and radiologists but leveraging the power of artificial intelligence to provide the initial opinion almost immediately with a probability or confidence score which helps determine if it needs to be sent to a Radiologist for review and further analysis where the patient could also be asked for go for the next level examinations and tests to confirm the diagnosis.

# Solution and Architecture

Our solution runs on Oracle Platforms like Object Storage where the images are uploaded, ODI Data Science platform is used to build and train the models. A neural network model is developed and trained on mammography screening images. The AI model is being created using Deep Learning approach **extending** the ResNet Architecture. A neural network model is developed and trained on mammography screening images and exposed as flask webservice. An UI is developed on Oracle Content Experience, which makes call to Python Flask webservice hosted on an Oracle Linux VM Instance

Autonomous Data warehouse stores the outcomes for further analysis and can optionally be combined with patient information to run Analytics on Oracle Analytics Cloud.



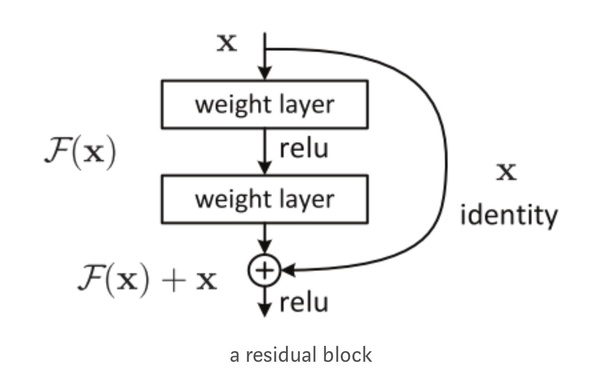
# Deep Learning Model

## Model Architecture with ResNet50

An AI model is created using the Deep Learning approach with extending the ResNet Architecture. This is a Convolutional Neural Network (CNN) Architecture published in 2015. ResNet-50 is a deep residual network. The “50” refers to the number of layers it has. It’s a subclass of convolutional neural networks, with ResNet being most popularly used for image classification.

The main innovation of ResNet is the skip connection. Without adjustments, deep networks often suffer from vanishing gradients, i.e., as the model backpropagates, the gradient gets smaller and smaller. Tiny gradients can make learning intractable.

The skip connection in the diagram below is labelled “identity.” It allows the network to learn the identity function, which allows it to pass the input through the block without passing through the other weight layers.



This allows you to stack additional layers and build a deeper network, offsetting the vanishing gradient by allowing your network to skip through layers that are less relevant in training.

## Image Augmentation

This is the technique to increase the dataset size. In total we had only 433 sample images including positive and negative cases with unbalanced dataset with 10% positive and 90% negative in the training data set. By doing a 4x data augmentation for minor (positive) rotating images at an angle of 70 degrees and making it equal distribution generates around 750 images in total with 45-55 positive to negative ratio. This improves the overall accuracy, precision, recall and f1 score.

## Hyper parameter settings

Below Hyper parameters settings were used

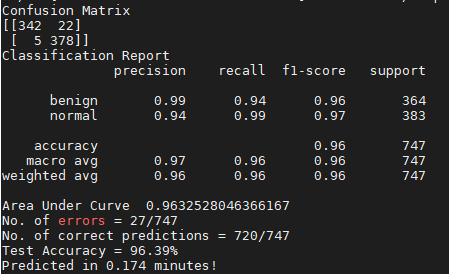
1. No of epocs - around 250.
2. Decreasing learning rate based on epocs.
3. Flip images to 20 degrees while training.
4. Image size is 256
5. Loss function as **binary\_crossentropy**
6. Optimizer **sgd**
7. Additional drop out of .5 in last two layers to prevent any overfitting
8. 80% training and 20% validation

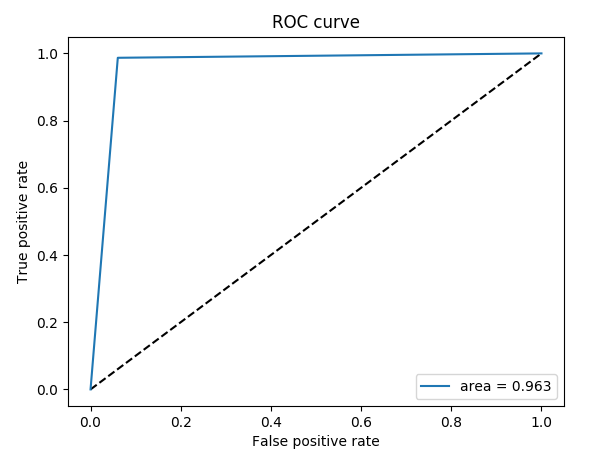
## Training Results

Training gets converged at 238 epoc with 98% train accuracy and 94% validation accuracy, but results are based on a fairly small dataset. This figure could change when we train and test the model with a larger dataset.

## Test Results

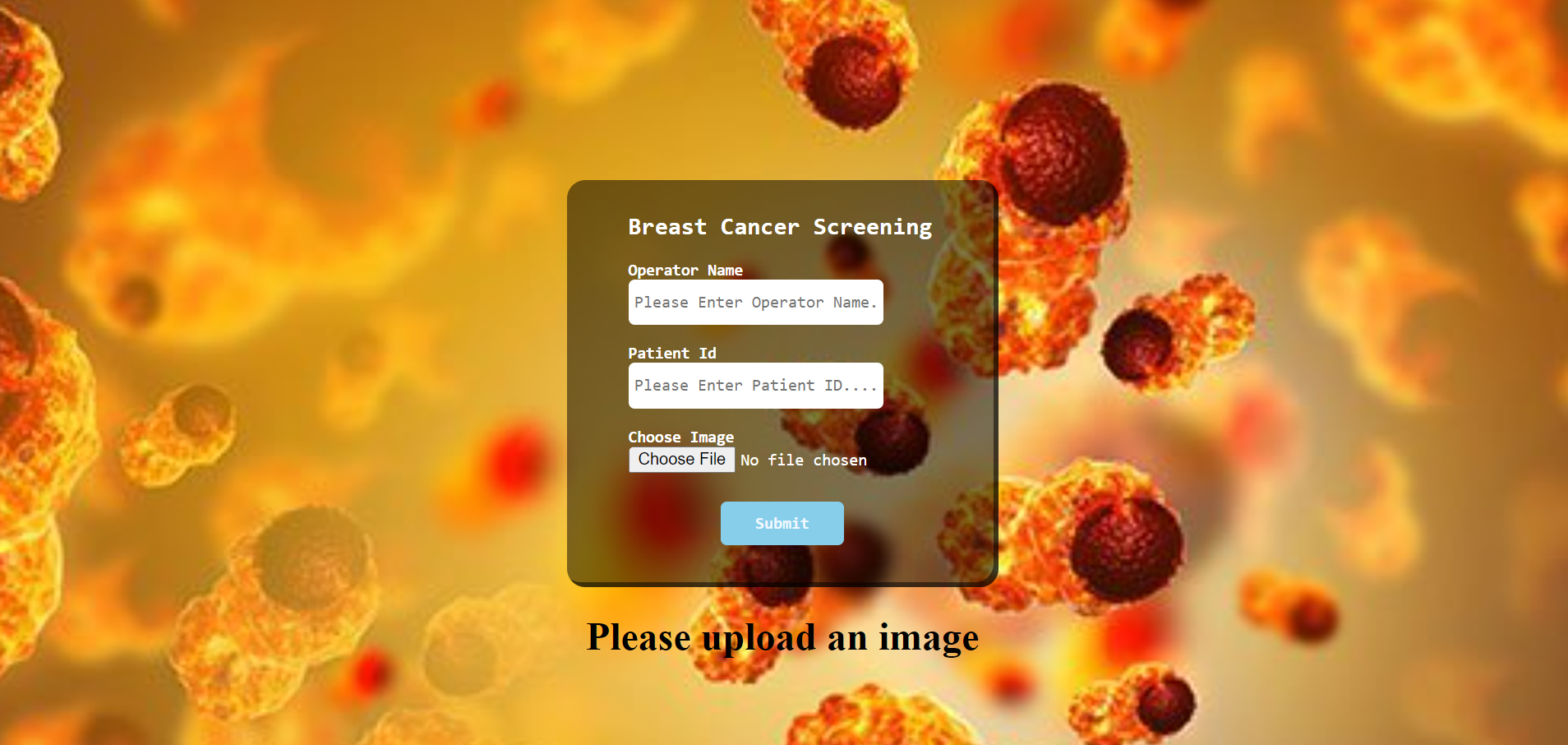
The Model, when tested on same dataset on which it was trained gave the following results



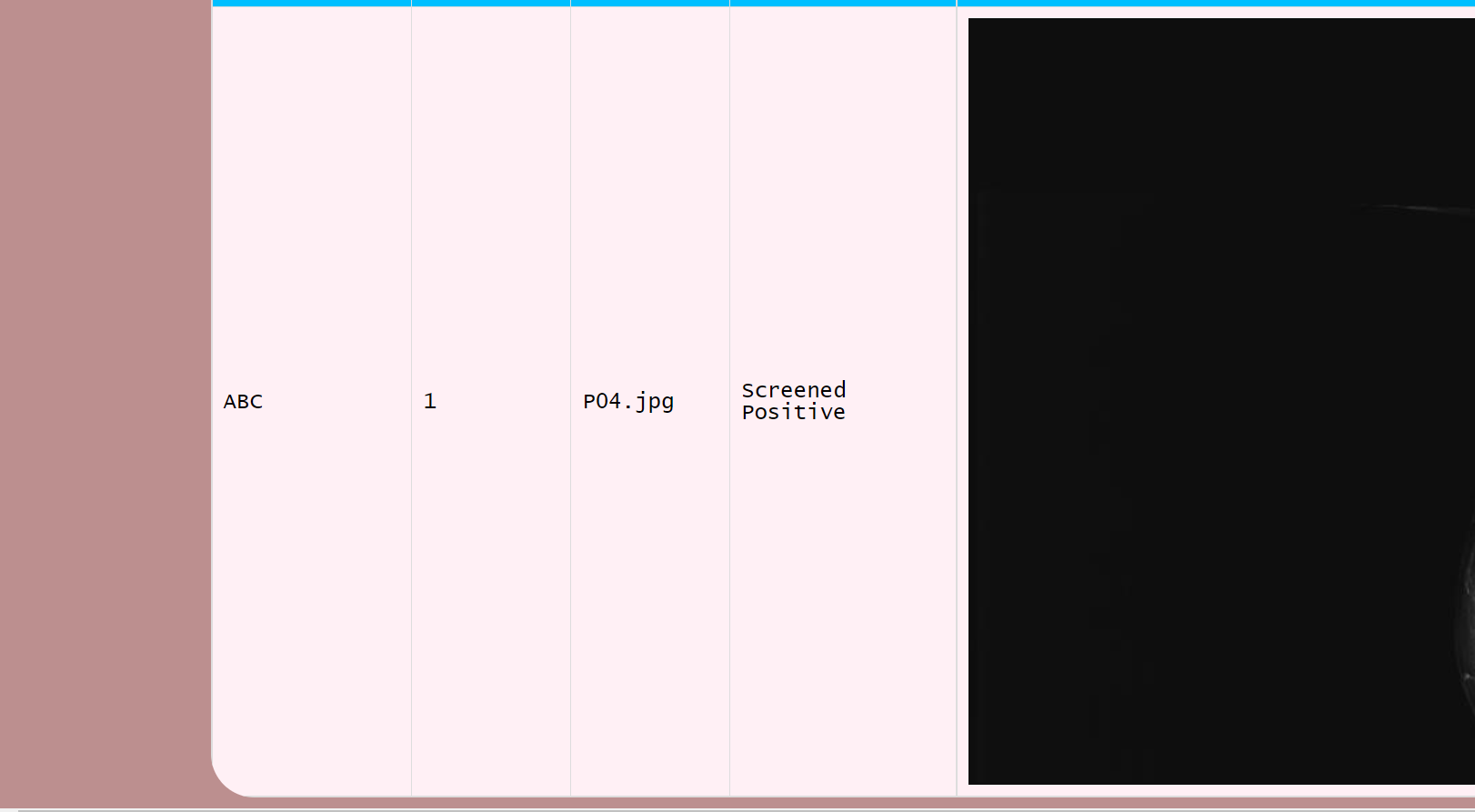


# User Interface

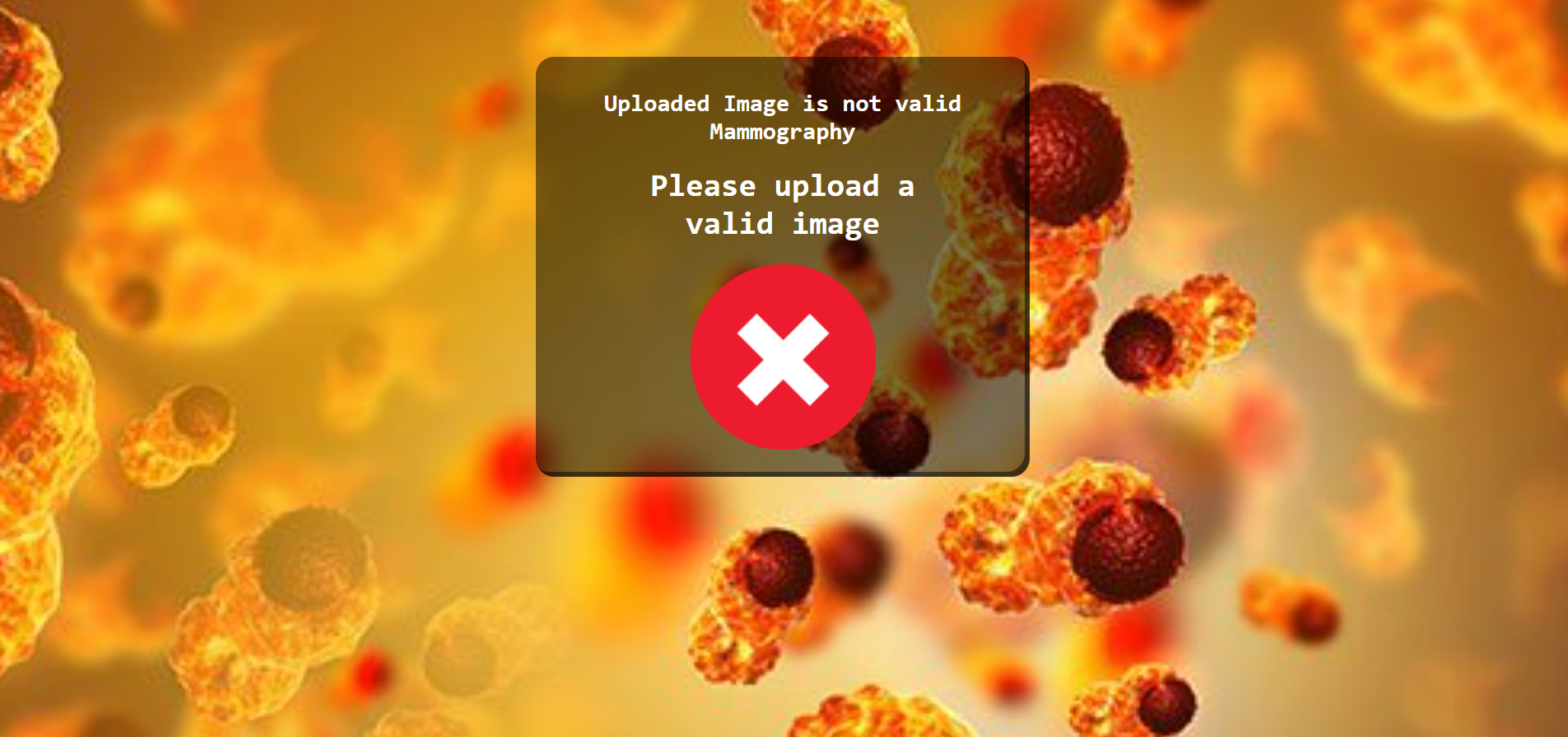
The User Interface is made accessible to the operator. The Operator uploads the image from their local system as below



On clicking the Submit button, it sends the image to the AI Model and gets inference. The results are sent back to the Operator on the UI Screen.



If the operator uploads an image which is not a valid mammography image, model detects it as anomaly and is directed to an error page



# Pre-Requisites and Skills required

## 

## Technology Used

1. Tensorflow
2. Convolutional Neural Network
3. Python Flask REST API
4. Python

## Pre-requisite Skills Required

1. OCI Data Science.
2. ADW instance
3. Python data science packages such as pandas and numpy
4. Deep learning framework Tensorflow
5. Keras
6. Python flask REST Webservice

# Python Flask Webservice

The model is being exposed via python Webservice hosted on Linux instance.

[http:// <hostname>/upload](http://140.238.164.47/upload)

When a new image is uploaded, it takes 3 inputs -PatientId, Operator and the Image upload. The image is uploaded and creates a folder with operator name. The image is archives in the folder image. This method passes the image to the AI model and then after making an inference, returns the inference result as well base64 encoding of the image to the UI.

# Code Setup and File details

* 1. We need to create two environments, one for training and other for inference using **conda create –n train pip python=3.6** and **conda create –n inference pip python=3.6**
  2. Activate the environments with **source activate train** and **source activate inference**
  3. There are 2 files- **requirements-inference.txt** and **requirements-train.txt**
  4. Run to execute both files using **pip install –r requirements-inference.txt** and **pip install –r requirements-train.txt** to install all dependencies in the respective environment.
  5. **train\_model.py :** This is used to train the model. Execute below command to train the model.

***python train\_model.py***

* 1. **test\_model.py:** This will be used to test the model performance. Execute below command to test the model.

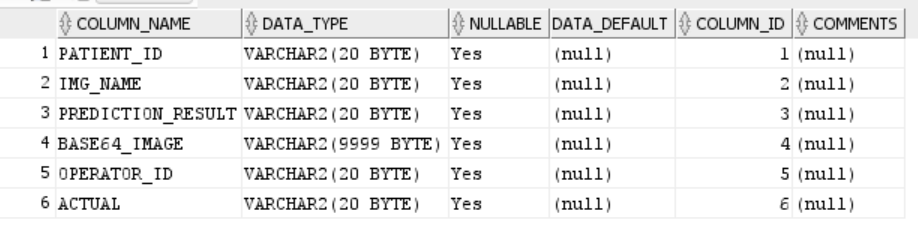
***python test\_model.py***

* 1. **server.py**: This is the python webserver which is to be hit from the UI, execute below command to start the server.

***python server.py***

this starts the UI at port 5001 and REST API can be accessed at <http://localhost:5001/upload>

* 1. **inference.html:** This the first HTML file which needs to open for uploading an image for getting the inference. Include the Operator Name , Patient ID and upload Image. It will infere and respond as “Screened Positive” or “Screened Negative”
  2. On each inference it saves the result in ADW database provided we have updated the credentials in **server.py**
  3. Data gets saved below table format:



* 1. **upload:** This is just a folder where images are uploaded temporarily and deleted later on once inference is done
  2. **test:** Test folder contain some sample images to test
  3. **dataset:** Contain the training dataset.
  4. **trained\_models:** This folder contains the trained model.
  5. **templates**: This contains static html page