Cognitive Care: Early Intervention for Alzheimer's Disease

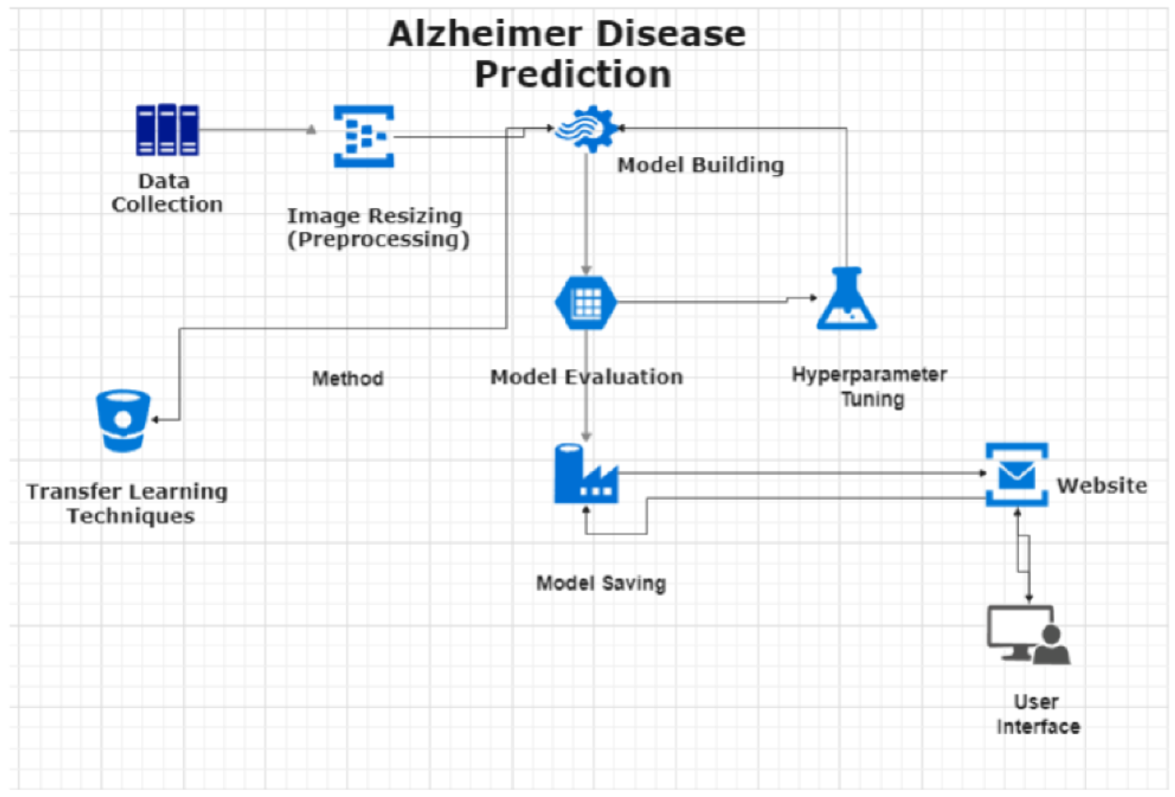
Project Description:

Alzheimer's disease (AD) is a progressive and irreversible neurological disorder that affects the brain, leading to memory loss, cognitive impairment, and changes in behavior and personality. It is the most common cause of dementia among older adults and is characterized by the buildup of abnormal protein deposits in the brain, including amyloid plaques and tau tangles.

The exact cause of Alzheimer's disease is not yet fully understood, but it is believed to be influenced by a combination of genetic, environmental, and lifestyle factors. Age is also a significant risk factor, with the risk of developing the disease increasing significantly after the age of 65. The early symptoms of Alzheimer's disease may include mild memory loss, difficulty with problem-solving, and changes in mood or behavior. As the disease progresses, these symptoms become more severe, with individuals experiencing significant memory loss, difficulty communicating, and a loss of the ability to perform daily activities.

By using deep learning models like Xception to analyze medical imaging data, it may be able to identify early signs of Alzheimer's disease before symptoms become severe. This can help healthcare providers to provide early treatment and support for patients and their families, ultimately leading to better outcomes for all involved. After Prediction of the Stage of disease, the WebApp also recommends the suitable prognosis for the patient based on the Current Disease Progression and Health Circumstances

Technical Architecture:



Project Flow:

- The user uploads an image to the website.
- The uploaded image is processed by a Xception deep learning model.
- The Xception model is integrated with a Flask application.
- The Xception model analyzes the image and generates predictions of the current stage of Alzheimers.
- The predictions are displayed on the Flask UI for the user to see.
- This process enables users to input an image and receive accurate predictions quickly.

* Tasks followed:

1. Data Collection.
    - A Kaggle image dataset consisting of scanned images of the brain is used.
    - Divide it into train and test paths.
2. Image Pre-processing.
    - Import the required libraries.
    - Perform image augmentation.
    - Set input image size as (150,150) for Xception model.
    - Set the suitable batch size, zoom, bright_range, horizontal_flip, fill_mode, data_format and so on.
    - Perform oversampling using SMOTE for balancing the dataset.
3. Model Building
    - Use a customized Xception model.
    - Add layers like dropout, global_average_pooling_2D, Flatten, BatchNormalization and Dense layers to customize the pre-built CNN model of Xception.
    - Among them there will be 4 Dense layers with activation function as ‘relu’ with 512, 256, 128 and 64 filters respectively.
    - The final output layer will be another Dense layer with 4 possible outcomes and a softmax activation function for multi output.
    - Configure the training process, setting the optimizer as ‘Adam’, loss as ‘categorical_crossentropy’ and metrics as ‘Accuracy’, ‘Precision’, ‘Recall’ and ‘AUC’.
    - Train the model using 20 epochs.
    - Save the model.
    - Test and evaluate the model.
4. Application Building
    - Create an HTML file
    - Create CSS files for Styling the Div for the website and for animation
    - Using JavaScript to provide functionality to the Websites Component such as Uploading Images
    - Build Python Flask Code for creating and accessing the prediction website to obtain accurate diagnosis of the Alzheimers disease upon request.
    - Run the application and test its efficiency.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **Deep Learning Concepts:**
    - **CNN**: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
    - **VGG16**:https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615
    - **ResNet-50**: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
    - **Inception-V3**: https://iq.opengenus.org/inception-v3-model-architecture/
    - **Xception**: https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

- **Full Stack Micro Framework**:

Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. **Link:** https://www.youtube.com/watch?v=lj4I_CvBnt0

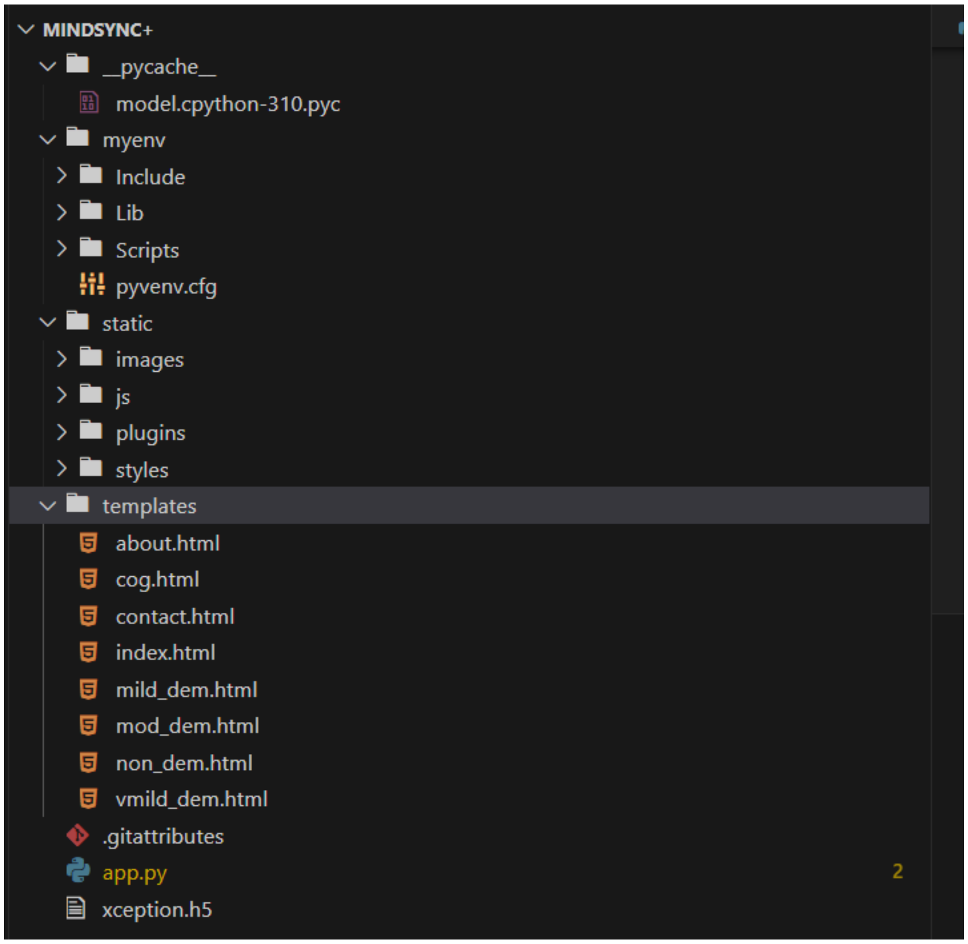
- **Front-End Web Development**:

1. **HTML**: https://www.w3schools.com/html/
2. **CSS:** https://www.w3schools.com/css/default.asp
3. **Javascript:** https://www.w3schools.com/js/default.asp

- **Back-End Web Development:**

1. **Flask:** Flask framework is used for routing purposes and works on the back-end.
2. Python with AI-ML model: Xception model is used for performing predictions.

Project Structure:



```
MINDSYNC+
  __pycache__
    model.cpython-310.pyc
  myenv
    Include
    Lib
    Scripts
    pyvenv.cfg
  static
    images
    js
    plugins
    styles
  templates
    about.html
    cog.html
    contact.html
    index.html
    mild_dem.html
    mod_dem.html
    non_dem.html
    vmild_dem.html
  .gitattributes
  app.py                                                2
  xception.h5
```

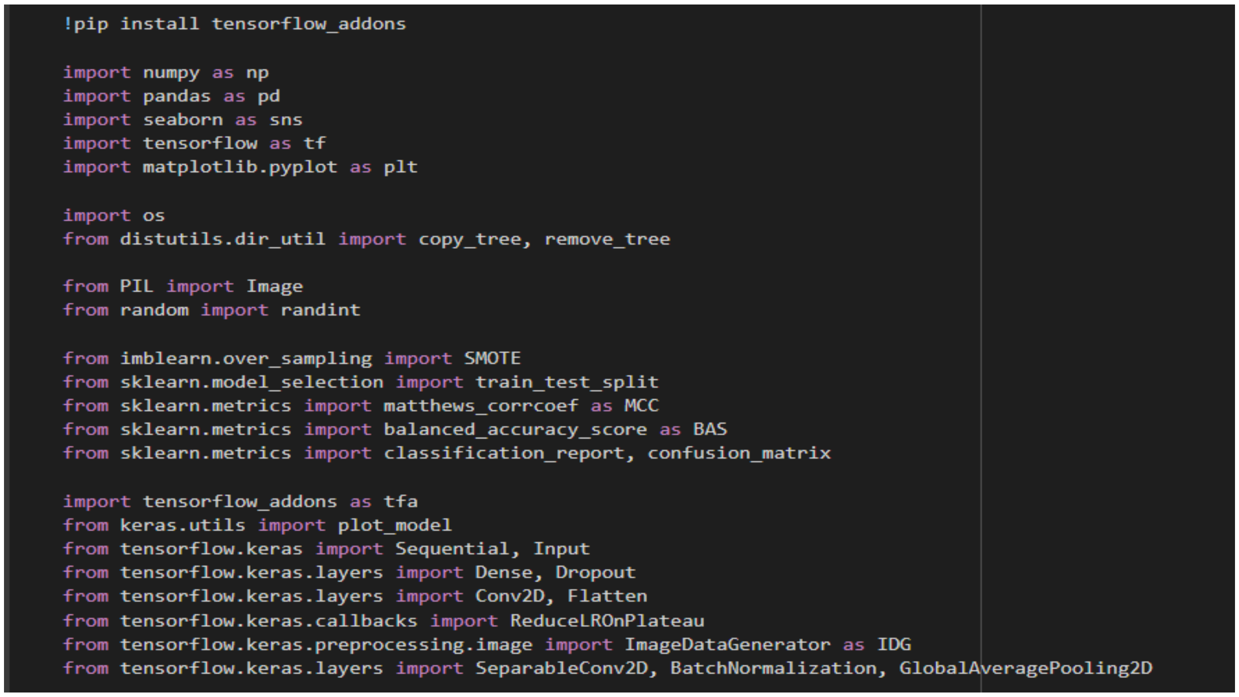
- The xception.h5 contains the model trained in Google Colab python notebook which has a customized **Xception** trained model.
- For building a Flask Application we need HTML pages stored in the **templates** folder, CSS for styling the pages stored in the static folder and a python script **app.py** for server-side scripting.
- There are Four main HTML file in this project which are cog.html, contact.html, about.html, index.html.
- Remaining HTML files are for the result based on the prediction.
- Static contains the style folder which contains the **CSS** file for the HTML page. Each page has static and dynamic CSS.
- In Flask, **myenv** typically refers to the virtual environment or Python environment where you install and manage the dependencies for your Flask web application. It's not a built-in concept in Flask itself but rather a common practice when developing Flask applications (or any Python applications) to isolate your project-specific dependencies from the system-wide Python environment.
- In the app,py file, we are importing modules like Flask, render_template, request, load_model, image and Image from libraries like flask, keras, PIL and tensorflow. We also import the numpy library.
- The render_template() is used for routing the python file to the html pages like index.html, about.html, cog.html and contact.html.
- This is followed by the python code for the customized Xception model for prediction of stage and intensity of Alzheimer’s disease.

Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.
In this project, we are using Kaggle to obtain the dataset of scanned images of the brain. The data consists of MRI images.
The data has four classes of images both in training as well as a testing set

Activity 1: Import the required libraries

Import important libraries like TensorFlow, keras, NumPy, pandas, seaborn, matplotlib, os, PIL, random, imblearn, sklearn and so on. Also import the necessary modules.



```
!pip install tensorflow_addons

import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt

import os
from distutils.dir_util import copy_tree, remove_tree

from PIL import Image
from random import randint

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import matthews_corrcoef as MCC
from sklearn.metrics import balanced_accuracy_score as BAS
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow_addons as tfa
from keras.utils import plot_model
from tensorflow.keras import Sequential, Input
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import Conv2D, Flatten
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
from tensorflow.keras.layers import SeparableConv2D, BatchNormalization, GlobalAveragePooling2D
```

Activity 2: Download the dataset

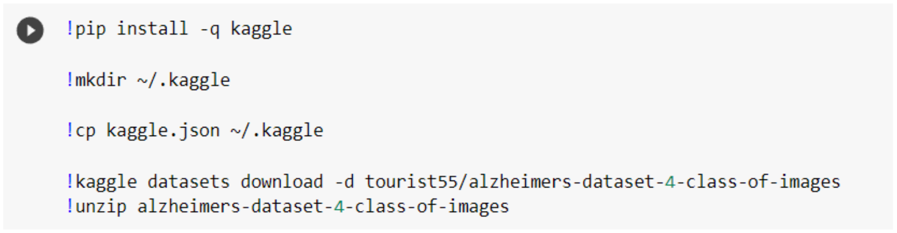
Collect images of brain MRI then split into train and test set. Then within these folders, organize them into subdirectories based on their respective names as shown in the project structure. Create folders of types of Alzheimer.

In this project, we have collected images of 4 types of brain MRI images like ‘MildDemented’, ‘ModerateDemented’, ‘NonDemented’ and ‘VeryMildDemented’ and they are saved in the respective sub directories with their respective names.

You can download the dataset used in this project using the below link Dataset:- Alzheimer's Dataset (4 class of Images) | Kaggle

We are going to build our training model on Google colab. We upload kaggle.json and acquire the respective input image files from Kaggle using their API token link. The zip folder uploaded is unzipped to get folders for different sub-categories of image dataset we need. The following steps are followed to achieve the same:

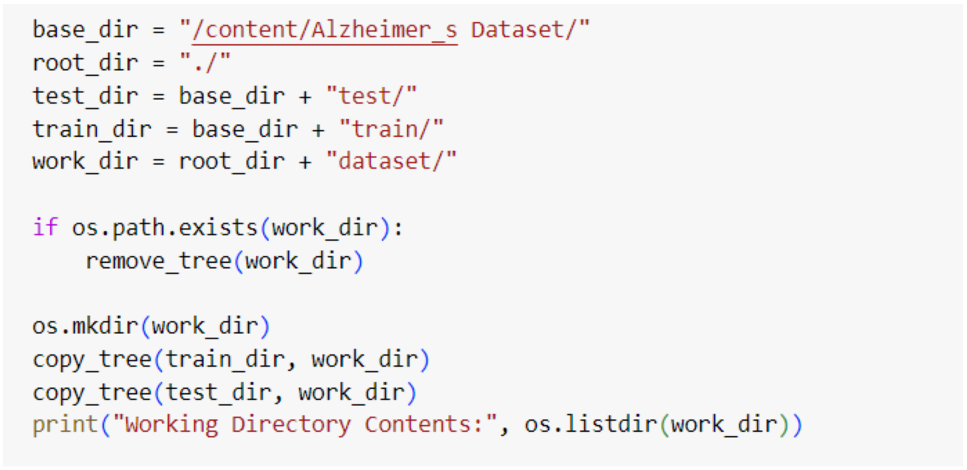
- Open Google Colab and create a new notebook.
- Click on the "Files" icon on the left-hand side of the screen.
- Click on the "Upload" button and select the kaggle.json file from the respective directory.
- Wait for the upload to complete. You should see the file appear in the "Files" section.
- Now use the API token link created for the respective dataset to upload the zip file of the dataset.
- To unzip the file, you can use the following command:



```
!pip install -q kaggle

!mkdir ~/.kaggle

!cp kaggle.json ~/.kaggle

!kaggle datasets download -d tourist55/alzheimers-dataset-4-class-of-images
!unzip alzheimers-dataset-4-class-of-images
```

Activity 3: Create training and testing dataset

To build a Deep Learning model we have to split training and testing data into two separate folders. But In the project dataset folder training and testing folders are presented. So, in this case we just have to assign a variable and pass the folder path to it.



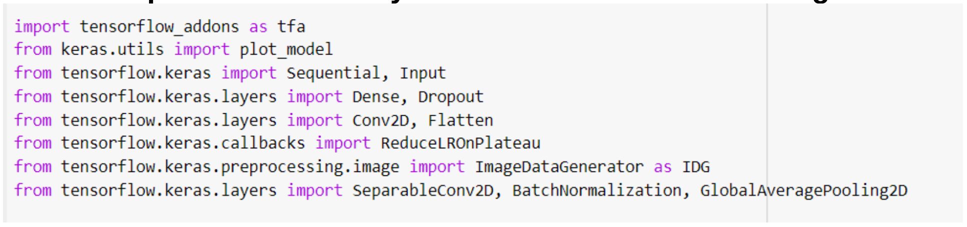
```
base_dir = "/content/Alzheimer_s Dataset/"
root_dir = "./"
test_dir = base_dir + "test/"
train_dir = base_dir + "train/"
work_dir = root_dir + "dataset/"

if os.path.exists(work_dir):
    remove_tree(work_dir)

os.mkdir(work_dir)
copy_tree(train_dir, work_dir)
copy_tree(test_dir, work_dir)
print("Working Directory Contents:", os.listdir(work_dir))
```

Milestone 2: Image Preprocessing

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, zooming etc. The preprocessing steps performed include image augmentation, dataset balancing using SMOTE and so on.

Activity 1: Importing the libraries

Import the necessary libraries as shown in the image



```
import tensorflow_addons as tfa
from keras.utils import plot_model
from tensorflow.keras import Sequential, Input
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import Conv2D, Flatten
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
from tensorflow.keras.layers import SeparableConv2D, BatchNormalization, GlobalAveragePooling2D
```

To understand the above imported libraries:-

- **ImageDataGenerator:** The ImageDataGenerator is a class in the tensorflow.keras.preprocessing.image module that generates batches of augmented image data in real-time during model training.
- **Keras:** Keras is a high-level neural network API written in Python that allows for fast experimentation and prototyping of deep learning models.
- **Dense Layer:** A dense layer in neural networks is a fully connected layer where each neuron in the layer is connected to every neuron in the previous layer, and each connection has a weight associated with it.
- **Flatten Layer:** A flatten layer in neural networks is a layer that reshapes the input tensor into a one-dimensional array, which can then be passed to a fully connected layer.
- **Input Layer:** The input layer in neural networks is the first layer of the network that receives the input data and passes it on to the next layer for further processing.
- **Dropout:** Dropout refers to data, or noise, that's intentionally dropped from a neural network to improve processing and time to results.
- **image:** from tensorflow.keras.preprocessing import image imports the image module from Keras' tensorflow.keras.preprocessing package. This module provides a number of image preprocessing utilities, such as loading images, converting images to arrays, and applying various image transformations.
- **load_img:** load_img is a function provided by the tensorflow.keras.preprocessing.image module that is used to load an image file from the local file system. It takes the file path as input and returns a PIL (Python Imaging Library) image object.
- **Xception:** from tensorflow.keras.applications.xception import Xception imports the Xception class from the tensorflow.keras.applications.xception module, which provides a pre-trained implementation of the Xception convolutional neural network architecture for image classification tasks.
- **Numpy**: It provides support for working with large, multi-dimensional arrays and matrices of numerical data, as well as a wide range of mathematical functions to operate on these arrays
- **tensorflow_addons** as **tfa**: tensorflow_addons is an additional library for TensorFlow that provides extra functionality, operations, and layers not available in the core TensorFlow package. It includes various addons for machine learning and deep learning tasks.
- **keras.utils**: This module provides utility functions, including plot_model, which allows you to generate a plot of the neural network model. This can be helpful for visualizing the architecture of the model.

- **Sequential:** It is a linear stack of layers in Keras. It allows you to create models layer-by-layer in a step-by-step fashion. Each layer has weights that correspond to the layer that follows it in the model.

- **Conv2D**: Conv2D is a 2D convolutional layer in Keras. It performs 2D convolutional operations on the input data, typically used for image processing tasks.

- **ReduceLROnPlateau:** It is a callback in Keras. It monitors a specified metric and reduces the learning rate when the metric has stopped improving. This is helpful for improving training when the model's performance plateaus.

- **SeparableConv2D**: SeparableConv2D is a depthwise separable 2D convolutional layer in Keras. It performs a spatial convolution on each channel of the input data independently and then combines the results.

- **BatchNormalization**: BatchNormalization is a layer in Keras that normalizes and scales the activations of a neural network, reducing internal covariate shift and improving training stability.

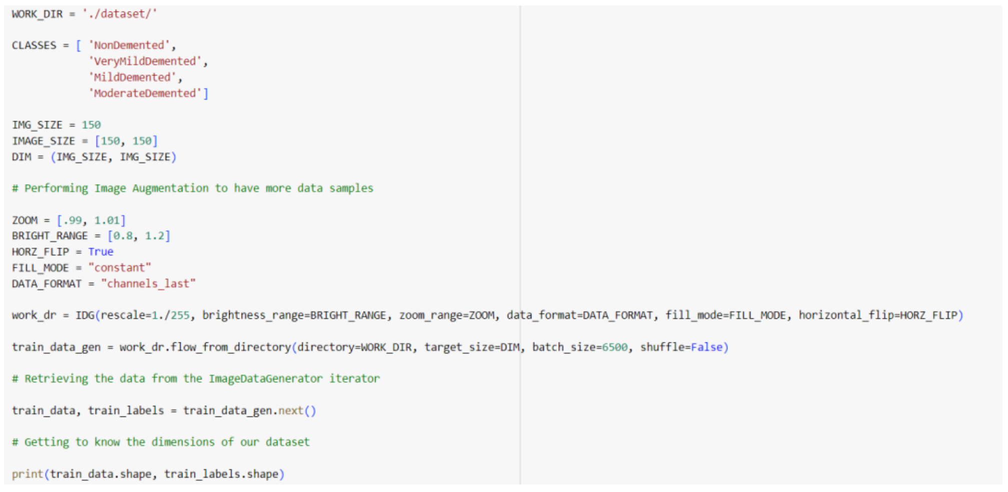
- **GlobalAveragePooling2D:** GlobalAveragePooling2D is a layer in Keras that performs global average pooling on the spatial dimensions of the input data. It reduces each feature map to a single value, effectively creating a global average of the features.

Activity 2: Configure ImageDataGenerator class

The ImageDataGenerator class is part of the tensorflow.keras.preprocessing.image module and is used for generating batches of augmented image data for training a neural network. This additional set of modified images provides additional batches of input for training which improves the capability of the model. The image size is set to (150,150) before being augmented. Here's a breakdown of the parameters used in this example:

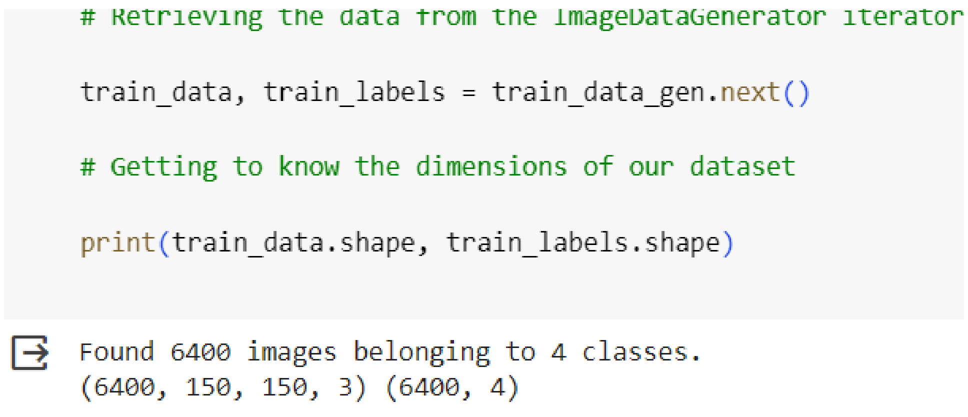
- rescale=1./255: This normalizes the pixel values of the image to the range of [0, 1], which is a common practice in deep learning models.
- zoom_range=[0.99,1.01]: This applies random zooming transformations to the image, which can help the model learn to be more robust to different scales of objects in the image.
- brightness_range=[0.8,1.2]: We can use it to adjust the brightness_range of any image for Data Augmentation.
- horizontal_flip=True: This applies random horizontal flipping to the image, which can help the model learn to be more invariant to the orientation of objects in the image.
- fill_mode='constant': 'fill_mode' is a parameter used to specify the strategy for filling in newly created pixels that might appear after a transformation. When it is 'constant', any newly created pixels outside the boundaries of the original image are filled by a constant value specified by the 'cval' parameter which is 0 by default.
- data_format = 'channels_last': 'data_format' is a parameter that determines the ordering of the dimensions in the input data. It specifies how the data is structured in multi-dimensional arrays.When it is set as 'chennels_last', the order for 2D case is (samples, height, width, channels) and for 3D case is (samples, depth, height, width, channels).
- Also set the batch_size as 6500 and shuffle as False and assign the correct directories for train and test.
- These data augmentation techniques can help increase the diversity and size of the training data, which can improve the performance of the model and reduce overfitting. You can use the train_data_gen object to generate batches of augmented training data on the fly, as needed by the fit() method of a Keras model.

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



```
WORK_DIR = './dataset/'

CLASSES = [ 'NonDemented',
            'VeryMildDemented',
            'MildDemented',
            'ModerateDemented']

IMG_SIZE = 150
IMAGE_SIZE = [150, 150]
DIM = (IMG_SIZE, IMG_SIZE)

# Performing Image Augmentation to have more data samples

ZOOM = [.99, 1.01]
BRIGHT_RANGE = [0.8, 1.2]
HORZ_FLIP = True
FILL_MODE = "constant"
DATA_FORMAT = "channels_last"

work_dr = IDG(rescale=1./255, brightness_range=BRIGHT_RANGE, zoom_range=ZOOM, data_format=DATA_FORMAT, fill_mode=FILL_MODE, horizontal_flip=HORZ_FLIP)

train_data_gen = work_dr.flow_from_directory(directory=WORK_DIR, target_size=DIM, batch_size=6500, shuffle=False)

# Retrieving the data from the ImageDataGenerator iterator

train_data, train_labels = train_data_gen.next()

# Getting to know the dimensions of our dataset

print(train_data.shape, train_labels.shape)
```

Activity 3: Handling imbalance data (SMOTE):



```
# Retrieving the data from the ImageDataGenerator iterator

train_data, train_labels = train_data_gen.next()

# Getting to know the dimensions of our dataset

print(train_data.shape, train_labels.shape)
```

```
Found 6400 images belonging to 4 classes.
(6400, 150, 150, 3) (6400, 4)
```

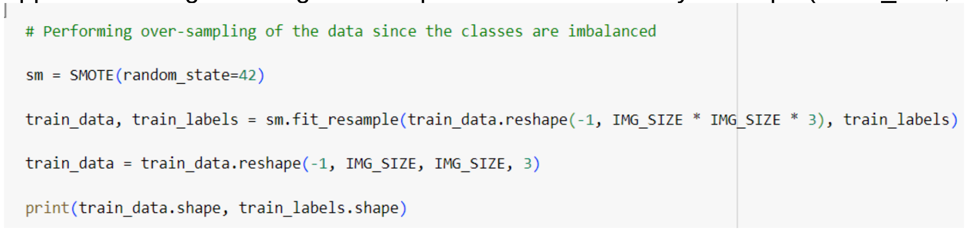
The next() function returns the next item in an iterator.

Checking the data before oversampling. As we can see, there are 5121 total images in the train data and 1279 images in the test data.

The SMOTE() is used to apply the SMOTE oversampling technique to balance the dataset. The random_state value provided is 42.

The reshape(-1, IMG_SIZE*IMG_SIZE*3) is used to reshape the 2D matrix shape of the input image of (150,150,3) dimension to a 1D array of length 150*150*3.

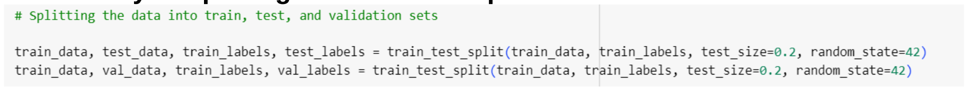
The sm.fit_resample technique is used to apply the SMOTE technique on the reshape train_dataset. After the sampling is applied the images are again reshaped back to a 4D array of shape (batch_size, IMG_size, IMG_size,3).



```
# Performing over-sampling of the data since the classes are imbalanced

sm = SMOTE(random_state=42)

train_data, train_labels = sm.fit_resample(train_data.reshape(-1, IMG_SIZE * IMG_SIZE * 3), train_labels)

train_data = train_data.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

print(train_data.shape, train_labels.shape)
```

After oversampling the data which had a shape of (6400, 150, 150, 3) has now doubled to (12,800, 150, 150, 3). The data has been doubled due to oversampling. As we can observe, from 5121 to 10,242.

Activity 4: Splitting into train test split



```
# Splitting the data into train, test, and validation sets

train_data, test_data, train_labels, test_labels = train_test_split(train_data, train_labels, test_size=0.2, random_state=42)
train_data, val_data, train_labels, val_labels = train_test_split(train_data, train_labels, test_size=0.2, random_state=42)
```

Now we are splitting the labels into train-test split in 80:20 ratio and assigning them to train_data, train_labels and test_data, test_lables. We also do the same for the validation test and assign to train_data, train_labels and val_data, val_

Milestone 3: Model Building

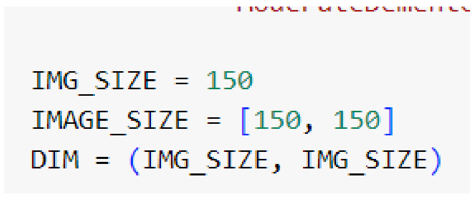
Now it's time to build our model. Let’s use the pre-trained model which is Xception, one of the convolution neural net (CNN) architecture which is considered as a very good model for Image classification. We perform some customization to the pre-built Xception model to make it more accurate and for better training purpose.

Deep understanding on the Xception model is required.

Activity 1: Pre-trained CNN model as a Feature Extractor

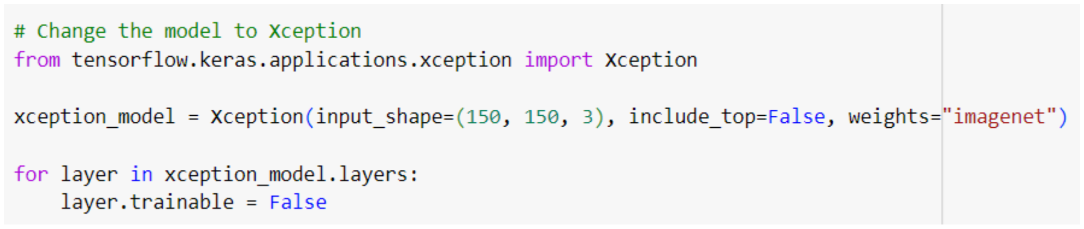
Here, we have considered images of dimension (150,150,3).

Also, we have assigned include_top = False because we are using convolution layer for features extraction and wants to train fully connected layer for our images classification(since it is not the part of Imagenet dataset).



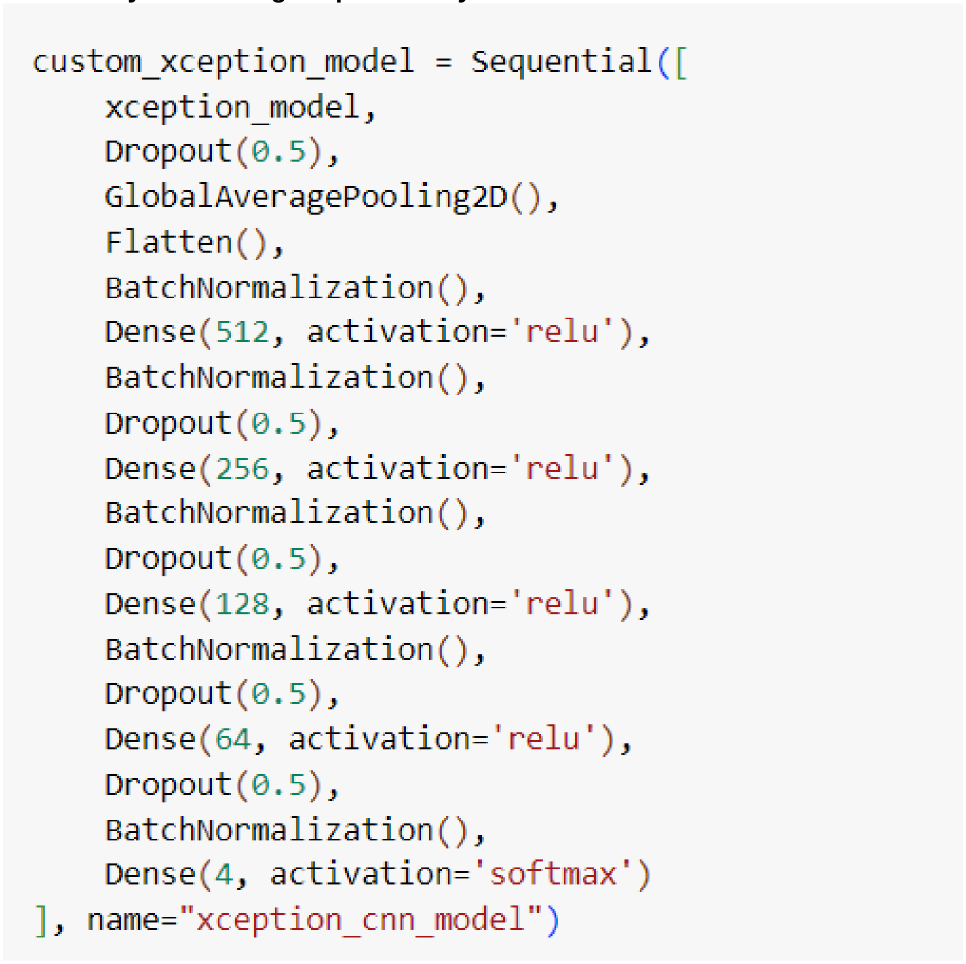
```
IMG_SIZE = 150
IMAGE_SIZE = [150, 150]
DIM = (IMG_SIZE, IMG_SIZE)
```

Different Transfer Learning models are used in our project such as VGG16, ResNet50, InceptionV3 and Xception, and the best model (Xception) is selected as it gives best metrics out of all of them. The image input size of Xception model is 150, 150.



```
# Change the model to Xception
from tensorflow.keras.applications.xception import Xception

xception_model = Xception(input_shape=(150, 150, 3), include_top=False, weights="imagenet")

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

Activity 2: Creating Sequential layers



```
custom_xception_model = Sequential([
    xception_model,
    Dropout(0.5),
    GlobalAveragePooling2D(),
    Flatten(),
    BatchNormalization(),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    BatchNormalization(),
    Dense(4, activation='softmax')
], name="xception_cnn_model")
```

Now we customize the Xception CNN model to make the training process better and more efficient.

For this purpose, we have imported **Sequential** from tensorflow.keras.models. As the name suggests it is used to arrange the Keras layers in a sequential manner.

We first input the pre-built Xception model. Then we add a Dropout layer with parameter as 0.5. Dropout rate is set to 0.5 as a form of regularization technique to prevent overfitting.

Now from layers we are importing **GlobalAveragePooling2D**. GlobalAveragePooling2D accepts an input 4D tensor. It operates the mean on the height and width dimensionalities for all the channels.

Then we add a Flattening layer where a multi-dimensional array can be converted into a one-dimensional array.

SeperableConv2D: This function is separable convolution which is a way to factorize a convolution kernel into twosmaller kernels.

BatchNormalization : Then a Batch Normalization is a normalization technique done between the layers of a Neural Network instead of in the raw data. It serves to speed up training and use higher learning rates, making learning easier.

This is followed by 4 alternating sets of Dense layer, Batch Normalization layer and Dropout layer. These 4 Dense layers have a ‘ReLu’ activation function (Rectified Linear Unit). It has 512, 256, 128 and 64 filters respectively.

Finally add another Dense layer with 4 filters and an activation function ‘softmax’ as the output layer.

ReLu activation function has been taken in the first four dense layers & softmax in the output layer.

A **dense** layer is a deeply connected neural network layer. It is the most common and frequently used layer. Let us create a model object named custom_xception_model with inputs as xception.input and output as dense layer.

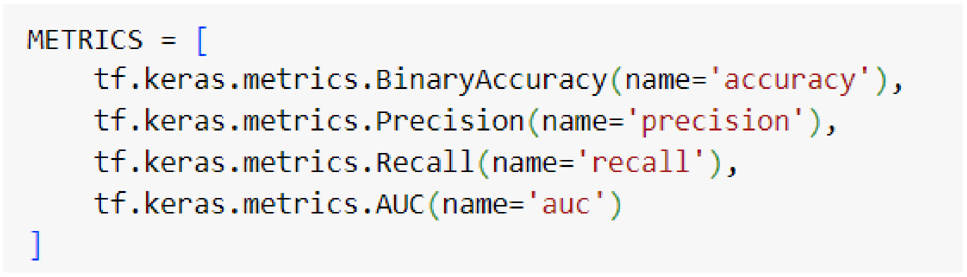
The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities. Understanding the model is a very important phase to properly use it for training and prediction purposes.

Activity 3: Configure the Learning Process

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.

Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. In this case, we are using the 'adam' optimizer.

Metrics are used to evaluate the performance of your model. They provide insights into how well the model is performing but are not used during the training process. In our code, we have defined a set of custom metrics using TensorFlow's Keras metrics, including Binary Accuracy, Precision, Recall, and AUC. These metrics will be used to measure various aspects of the model's performance.

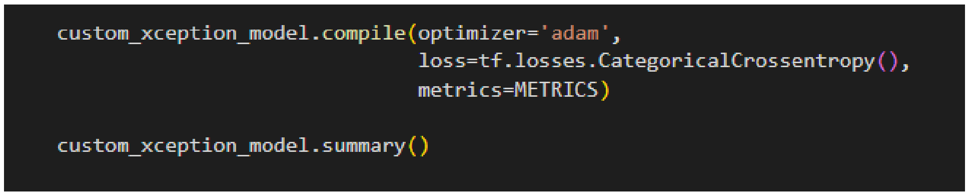


```
METRICS = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall'),
    tf.keras.metrics.AUC(name='auc')
]
```

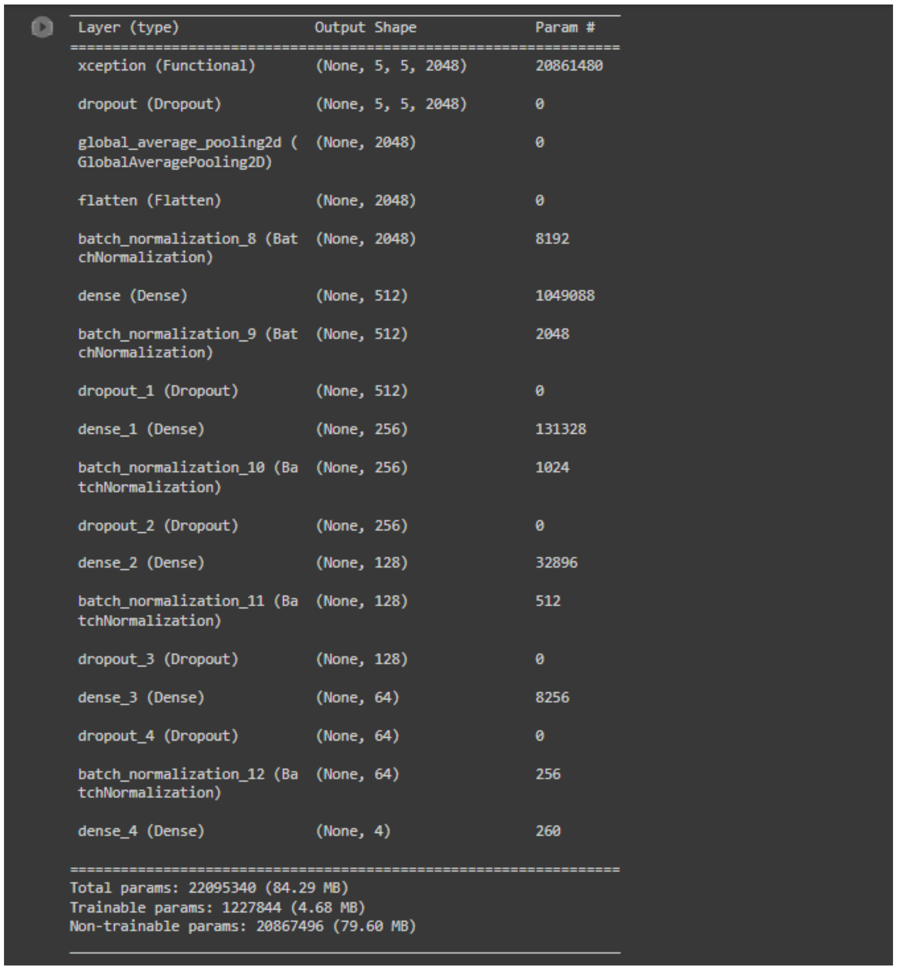
Activity 4: Train the model

Now, we will train our model with our image dataset. The model is trained for 20 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch and training accuracy increases.

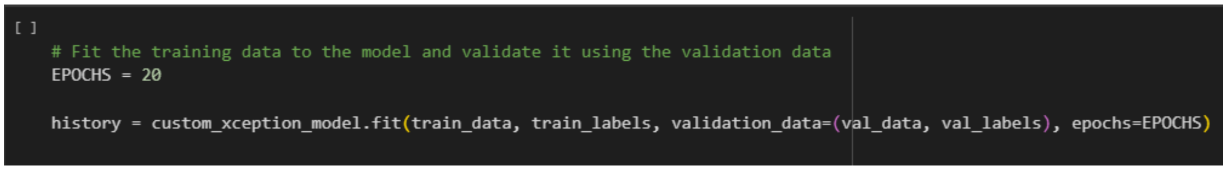
The **fit** function is used to train a deep learning neural network. The custom_xception_model.compile() takes the optimizer, loss and metrics as its parameters whereas the custom_xception_model.fit() takes the training and validation datasets, labels and the number of epochs as its parameters.



```
custom_xception_model.compile(optimizer='adam',
                              loss=tf.losses.CategoricalCrossentropy(),
                              metrics=METRICS)

custom_xception_model.summary()
```



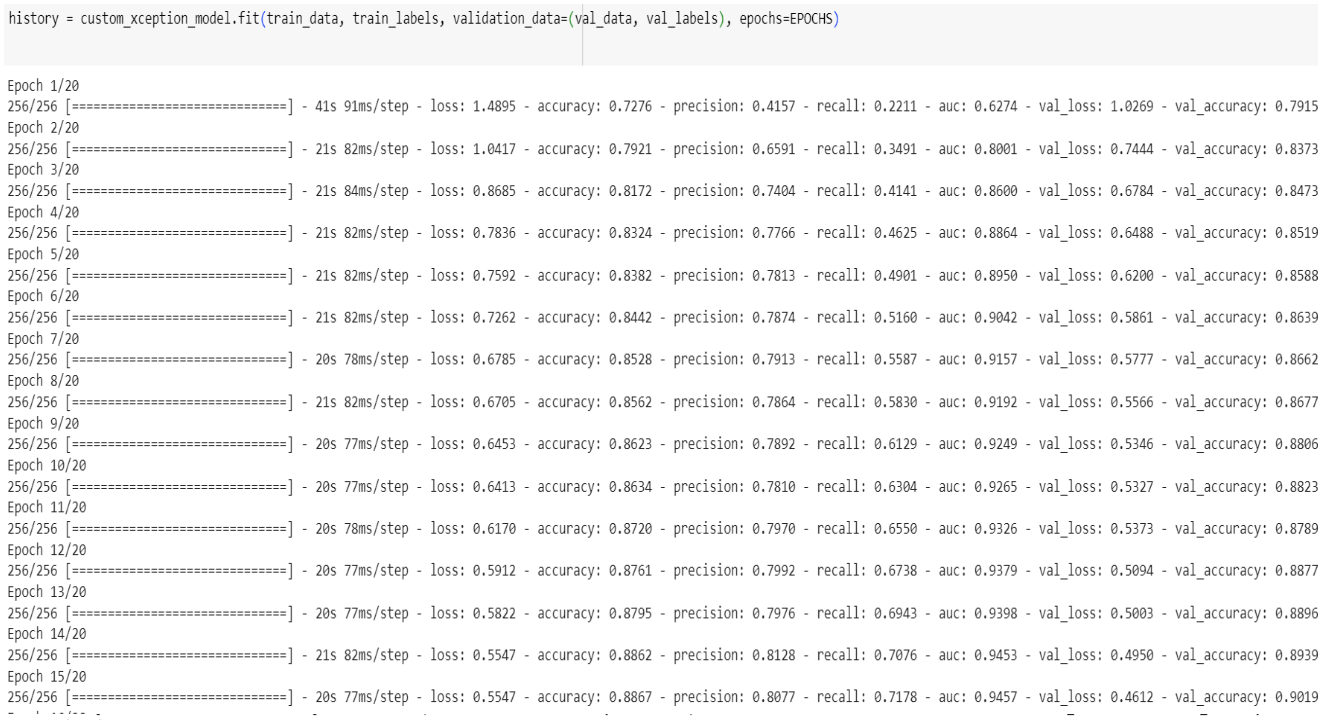
```
Layer (type)                Output Shape              Param #
=================================================================
xception (Functional)       (None, 5, 5, 2048)        20861480

dropout (Dropout)           (None, 5, 5, 2048)        0

global_average_pooling2d (  (None, 2048)              0
GlobalAveragePooling2D)

flatten (Flatten)           (None, 2048)              0

batch_normalization_8 (Bat  (None, 2048)              8192
chNormalization)

dense (Dense)               (None, 512)               1049088

batch_normalization_9 (Bat  (None, 512)               2048
chNormalization)

dropout_1 (Dropout)         (None, 512)               0

dense_1 (Dense)             (None, 256)               131328

batch_normalization_10 (Ba  (None, 256)               1024
tchNormalization)

dropout_2 (Dropout)         (None, 256)               0

dense_2 (Dense)             (None, 128)               32896

batch_normalization_11 (Ba  (None, 128)               512
tchNormalization)

dropout_3 (Dropout)         (None, 128)               0

dense_3 (Dense)             (None, 64)                8256

dropout_4 (Dropout)         (None, 64)                0

batch_normalization_12 (Ba  (None, 64)                256
tchNormalization)

dense_4 (Dense)             (None, 4)                 260

=================================================================
Total params: 22095340 (84.29 MB)
Trainable params: 1227844 (4.68 MB)
Non-trainable params: 20867496 (79.60 MB)
```



```
[ ]
# Fit the training data to the model and validate it using the validation data
EPOCHS = 20

history = custom_xception_model.fit(train_data, train_labels, validation_data=(val_data, val_labels), epochs=EPOCHS)
```

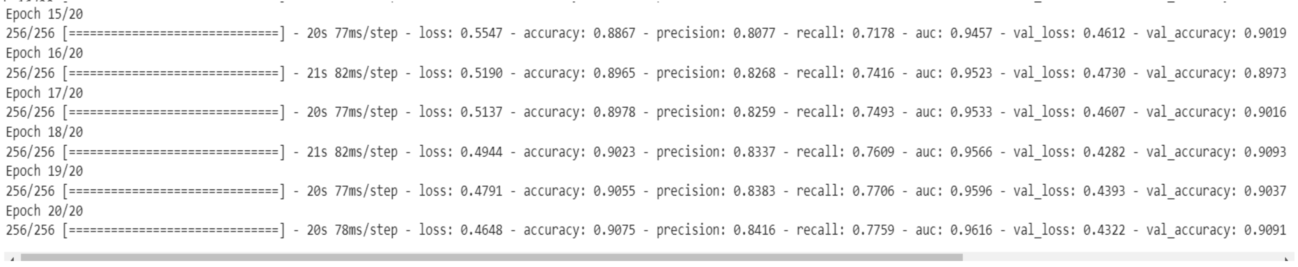
Arguments:

- Epochs: an integer and number of epochs we want to train our model for. (20 epochs is used here)
- validation data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.



```
history = custom_xception_model.fit(train_data, train_labels, validation_data=(val_data, val_labels), epochs=EPOCHS)
```

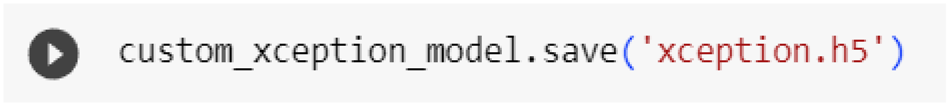
```
Epoch 1/20
256/256 [==============================] - 41s 91ms/step - loss: 1.4895 - accuracy: 0.7276 - precision: 0.4157 - recall: 0.2211 - auc: 0.6274 - val_loss: 1.0269 - val_accuracy: 0.7915
Epoch 2/20
256/256 [==============================] - 21s 82ms/step - loss: 1.0417 - accuracy: 0.7921 - precision: 0.6591 - recall: 0.3491 - auc: 0.8001 - val_loss: 0.7444 - val_accuracy: 0.8373
Epoch 3/20
256/256 [==============================] - 21s 84ms/step - loss: 0.8685 - accuracy: 0.8172 - precision: 0.7404 - recall: 0.4141 - auc: 0.8600 - val_loss: 0.6784 - val_accuracy: 0.8473
Epoch 4/20
256/256 [==============================] - 21s 82ms/step - loss: 0.7836 - accuracy: 0.8324 - precision: 0.7766 - recall: 0.4625 - auc: 0.8864 - val_loss: 0.6488 - val_accuracy: 0.8519
Epoch 5/20
256/256 [==============================] - 21s 82ms/step - loss: 0.7592 - accuracy: 0.8382 - precision: 0.7813 - recall: 0.4901 - auc: 0.8950 - val_loss: 0.6200 - val_accuracy: 0.8588
Epoch 6/20
256/256 [==============================] - 21s 82ms/step - loss: 0.7262 - accuracy: 0.8442 - precision: 0.7874 - recall: 0.5160 - auc: 0.9042 - val_loss: 0.5861 - val_accuracy: 0.8639
Epoch 7/20
256/256 [==============================] - 20s 78ms/step - loss: 0.6785 - accuracy: 0.8528 - precision: 0.7913 - recall: 0.5587 - auc: 0.9157 - val_loss: 0.5777 - val_accuracy: 0.8662
Epoch 8/20
256/256 [==============================] - 21s 82ms/step - loss: 0.6705 - accuracy: 0.8562 - precision: 0.7864 - recall: 0.5830 - auc: 0.9192 - val_loss: 0.5566 - val_accuracy: 0.8677
Epoch 9/20
256/256 [==============================] - 20s 77ms/step - loss: 0.6453 - accuracy: 0.8623 - precision: 0.7892 - recall: 0.6129 - auc: 0.9249 - val_loss: 0.5346 - val_accuracy: 0.8806
Epoch 10/20
256/256 [==============================] - 20s 77ms/step - loss: 0.6413 - accuracy: 0.8634 - precision: 0.7810 - recall: 0.6304 - auc: 0.9265 - val_loss: 0.5327 - val_accuracy: 0.8823
Epoch 11/20
256/256 [==============================] - 20s 78ms/step - loss: 0.6170 - accuracy: 0.8720 - precision: 0.7970 - recall: 0.6550 - auc: 0.9326 - val_loss: 0.5373 - val_accuracy: 0.8789
Epoch 12/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5912 - accuracy: 0.8761 - precision: 0.7992 - recall: 0.6738 - auc: 0.9379 - val_loss: 0.5094 - val_accuracy: 0.8877
Epoch 13/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5822 - accuracy: 0.8795 - precision: 0.7976 - recall: 0.6943 - auc: 0.9398 - val_loss: 0.5003 - val_accuracy: 0.8896
Epoch 14/20
256/256 [==============================] - 21s 82ms/step - loss: 0.5547 - accuracy: 0.8862 - precision: 0.8128 - recall: 0.7076 - auc: 0.9453 - val_loss: 0.4950 - val_accuracy: 0.8939
Epoch 15/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5547 - accuracy: 0.8867 - precision: 0.8077 - recall: 0.7178 - auc: 0.9457 - val_loss: 0.4612 - val_accuracy: 0.9019
```



```
Epoch 15/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5547 - accuracy: 0.8867 - precision: 0.8077 - recall: 0.7178 - auc: 0.9457 - val_loss: 0.4612 - val_accuracy: 0.9019
```

```
Epoch 16/20
256/256 [==============================] - 21s 82ms/step - loss: 0.5190 - accuracy: 0.8965 - precision: 0.8268 - recall: 0.7416 - auc: 0.9523 - val_loss: 0.4730 - val_accuracy: 0.8973
Epoch 17/20
256/256 [==============================] - 20s 77ms/step - loss: 0.5137 - accuracy: 0.8978 - precision: 0.8259 - recall: 0.7493 - auc: 0.9533 - val_loss: 0.4607 - val_accuracy: 0.9016
Epoch 18/20
256/256 [==============================] - 21s 82ms/step - loss: 0.4944 - accuracy: 0.9023 - precision: 0.8337 - recall: 0.7609 - auc: 0.9566 - val_loss: 0.4282 - val_accuracy: 0.9093
Epoch 19/20
256/256 [==============================] - 20s 77ms/step - loss: 0.4791 - accuracy: 0.9055 - precision: 0.8383 - recall: 0.7706 - auc: 0.9596 - val_loss: 0.4393 - val_accuracy: 0.9037
Epoch 20/20
256/256 [==============================] - 20s 78ms/step - loss: 0.4648 - accuracy: 0.9075 - precision: 0.8416 - recall: 0.7759 - auc: 0.9616 - val_loss: 0.4322 - val_accuracy: 0.9091
```

Activity 5: Save the Model



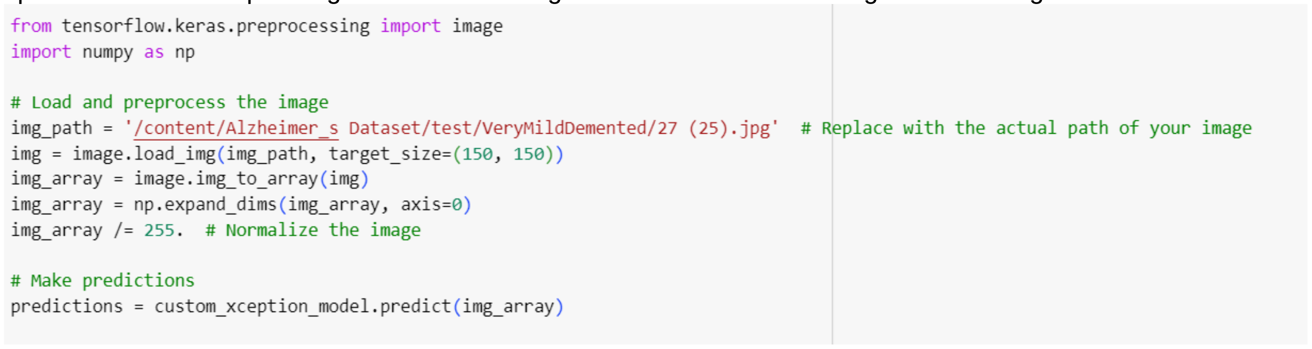
```python
custom_xception_model.save('xception.h5')
```

Out of all the models we tried (VGG16, Resnet50, Inception V3 & Xception) Xception gave us the best training and validation accuracies (about 90%). So, we are saving Xception as our final model.

The model is saved with .h5 extension as follows
A .h5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Activity 6: Testing the model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Use an image to test the model and augment the image.

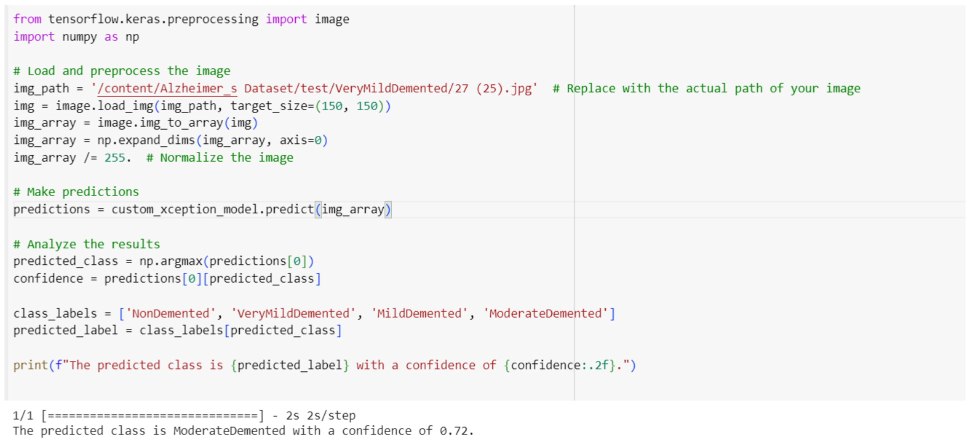


```python
from tensorflow.keras.preprocessing import image
import numpy as np

# Load and preprocess the image
img_path = '/content/Alzheimer_s Dataset/test/VeryMildDemented/27 (25).jpg'  # Replace with the actual path of your image
img = image.load_img(img_path, target_size=(150, 150))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.  # Normalize the image

# Make predictions
predictions = custom_xception_model.predict(img_array)
```

Index positions:

Nondemented = 0
VeryMildDemented = 1
MildDemented = 2
ModerateDemented = 3

Taking an image as input and checking the results



```
from tensorflow.keras.preprocessing import image
import numpy as np

# Load and preprocess the image
img_path = '/content/Alzheimer_s Dataset/test/VeryMildDemented/27 (25).jpg'  # Replace with the actual path of your image
img = image.load_img(img_path, target_size=(150, 150))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.  # Normalize the image

# Make predictions
predictions = custom_xception_model.predict(img_array)

# Analyze the results
predicted_class = np.argmax(predictions[0])
confidence = predictions[0][predicted_class]

class_labels = ['NonDemented', 'VeryMildDemented', 'MildDemented', 'ModerateDemented']
predicted_label = class_labels[predicted_class]

print(f"The predicted class is {predicted_label} with a confidence of {confidence:.2f}.")
```

```
1/1 [==============================] - 2s 2s/step
The predicted class is ModerateDemented with a confidence of 0.72.
```

So, our model will give the index position of the label. In this case, the 3rd index is for Moderate Dementia,which has been predicted correctly.

Milestone 4: Application Building

In this section, we will be building an Alzheimer's Prediction web application called 'MindSync+' that is integrated to the Xception model we built. An User Interface is provided for the users where they must upload the image to get predicted results on the stage and intensity of Alzheimer's. The uploaded image is given as input to the saved modeland the respective prediction is showcased in a different web page with their Key Metrics about how they can deal with the disease.

This section has the following tasks

- Building HTML Pages.
- Building Flask Python Code.

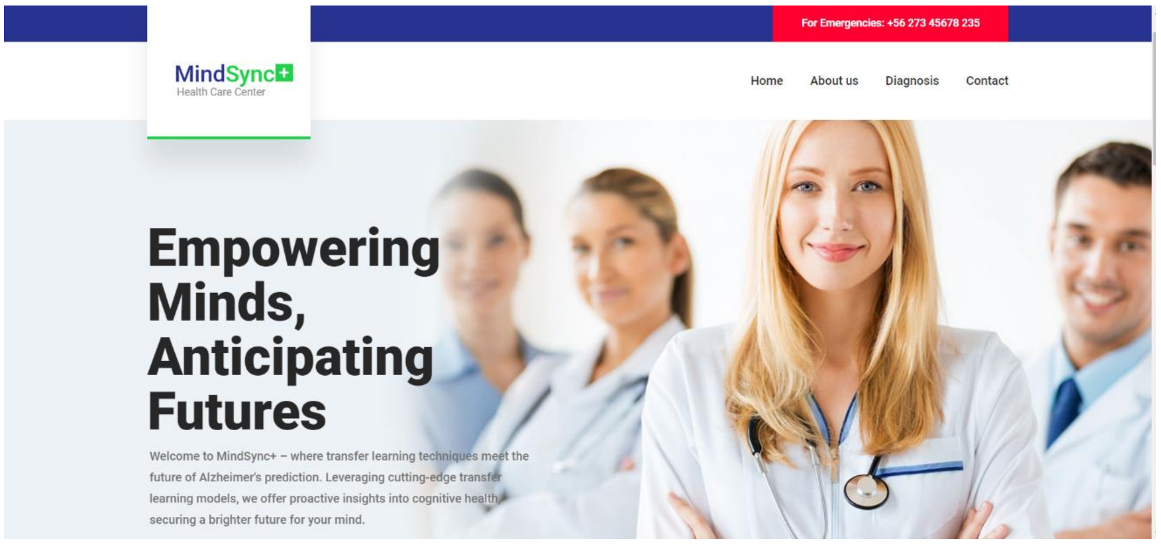
Activity1: Building Html Pages:

For this project create eight HTML files namely

- index.html
- about.html
- cog.html
- contact.html
- mild_dem.html
- mod_dem.html
- non_dem.html
- vmild_dem.html

The last four html pages are output pages. Based on the prediction, the user is redirected to one of these sites.

1. Let's see how our index.html page (Home page) looks like:



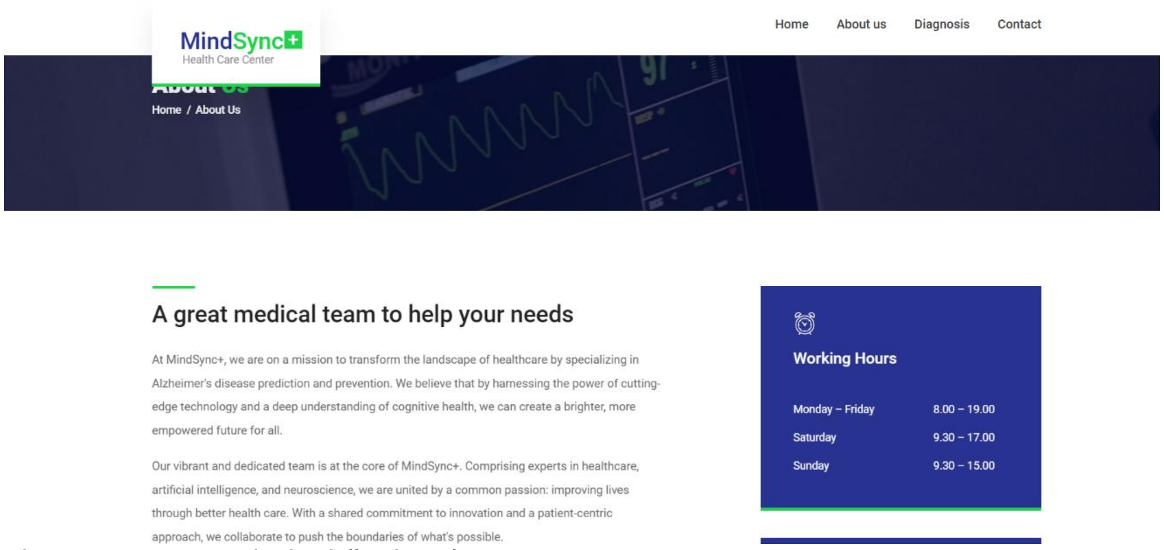
Empowering Minds, Anticipating Futures

Welcome to MindSync+ – where transfer learning techniques meet the future of Alzheimer's prediction. Leveraging cutting-edge transfer learning models, we offer proactive insights into cognitive health, securing a brighter future for your mind.

This is the page is our Landing Page and contains all the information about the service. The page has Links to all the four parts of application in the Dashboard ribbon. This contains all the information

- The timing to call our announcement
- Working days
- Useful Links to important website

2. Let's see how our about.html page (About Us Page) looks like:



A great medical team to help your needs

At MindSync+, we are on a mission to transform the landscape of healthcare by specializing in Alzheimer's disease prediction and prevention. We believe that by harnessing the power of cutting-edge technology and a deep understanding of cognitive health, we can create a brighter, more empowered future for all.

Our vibrant and dedicated team is at the core of MindSync+. Comprising experts in healthcare, artificial intelligence, and neuroscience, we are united by a common passion: improving lives through better health care. With a shared commitment to innovation and a patient-centric approach, we collaborate to push the boundaries of what's possible.

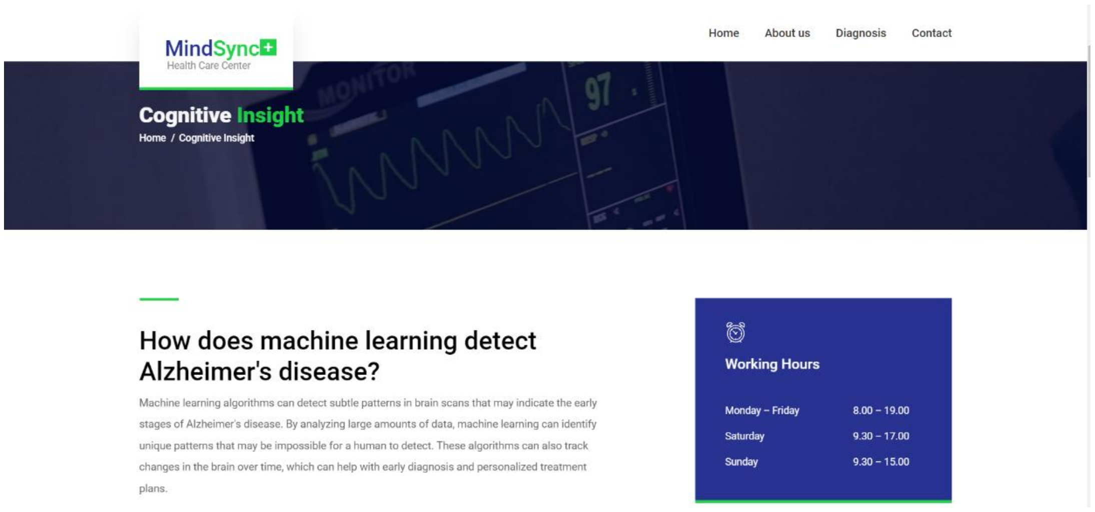
Working Hours

Monday – Friday	8.00 – 19.00
Saturday	9.30 – 17.00
Sunday	9.30 – 15.00

About Us page contain the following elements

- Working Hours of the services
- Information about the team
- Useful links to the articles about Alzheimers

3. Let's see how our cog.html page (Diagnosis Page) looks like:


Home About us Diagnosis Contact

MindSync+ Health Care Center

Cognitive Insight

Home / Cognitive Insight

How does machine learning detect Alzheimer's disease?

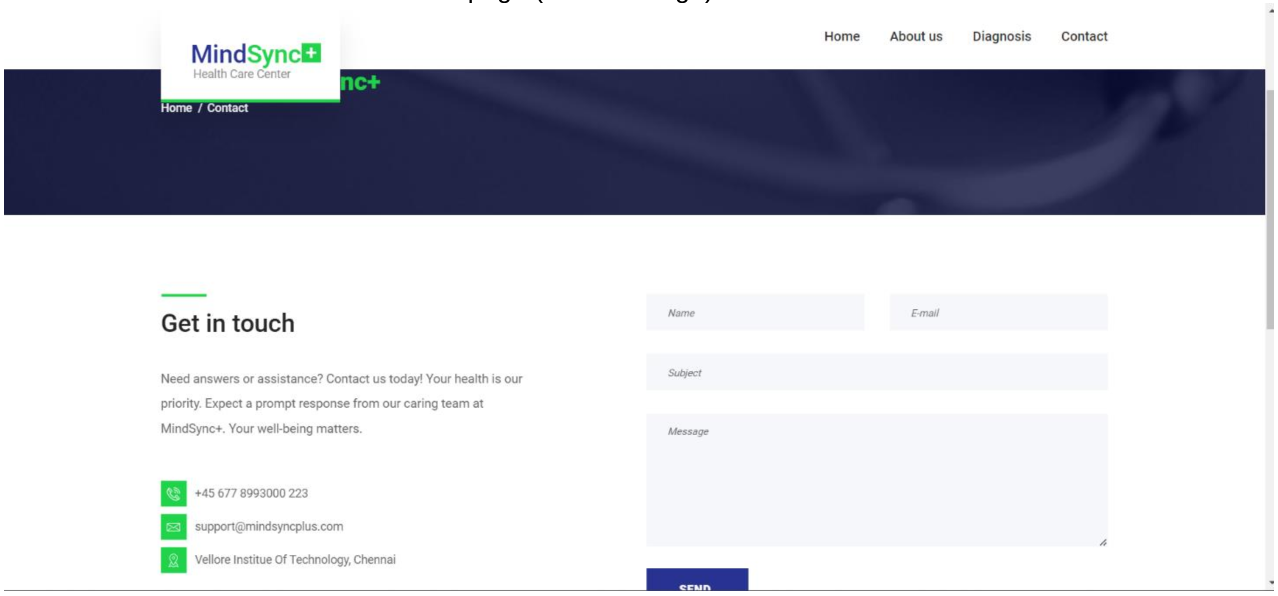
Machine learning algorithms can detect subtle patterns in brain scans that may indicate the early stages of Alzheimer's disease. By analyzing large amounts of data, machine learning can identify unique patterns that may be impossible for a human to detect. These algorithms can also track changes in the brain over time, which can help with early diagnosis and personalized treatment plans.

Working Hours

Monday – Friday	8.00 – 19.00
Saturday	9.30 – 17.00
Sunday	9.30 – 15.00

- The ‘Diagnosis’ page is the page where the prediction process is performed.
- It provides a UI to the user where they can upload a scanned image of the brain by clicking the ‘CHOOSE FILE’ option.
- The uploaded image can be viewed in the ‘Image Preview’ section.
- Then the user should click the ‘Submit’ button.
- The uploaded image is sent to the saved model in the backend, where the prediction takes place.
- Based on the predicted result, the website redirects the user to one of the four result pages, where they can get more information about the current stage of their Alzheimer’s disease and some guidance related to it.

4. Let’s see how our contact.html page (Contact Page) looks like:


Home About us Diagnosis Contact

MindSync+ Health Care Center

Home / Contact

Get in touch

Need answers or assistance? Contact us today! Your health is our priority. Expect a prompt response from our caring team at MindSync+. Your well-being matters.

+45 677 8993000 223

support@mindsyncplus.com

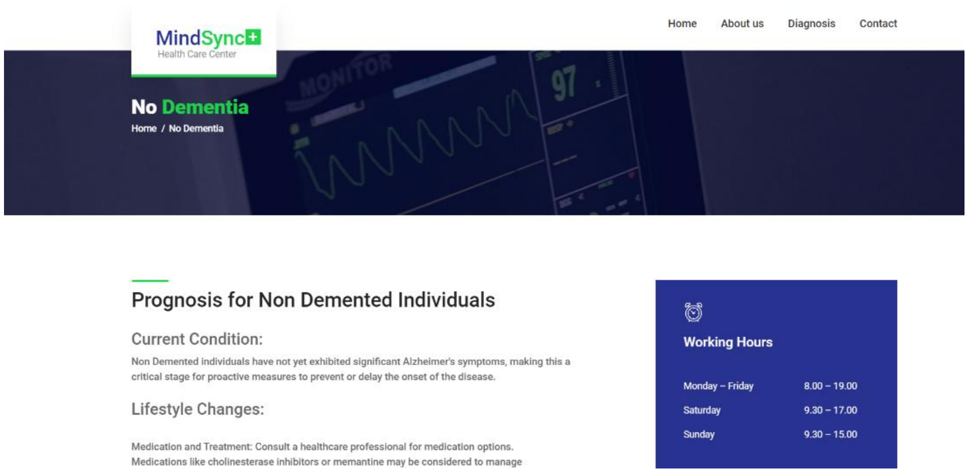
Vellore Institue Of Technology, Chennai

- The Contact page offers a UI to users, where they can provide feedback or register complaints by typing their name, email id, subject of concern, their message and then clicking the ‘SEND’ button.

- They can also use the contact details or email id provided on the website for help or guidance purposes.

Result Pages:

1. NonDemented:



Prognosis for Non Demented Individuals

Current Condition:

Non Demented individuals have not yet exhibited significant Alzheimer's symptoms, making this a critical stage for proactive measures to prevent or delay the onset of the disease.

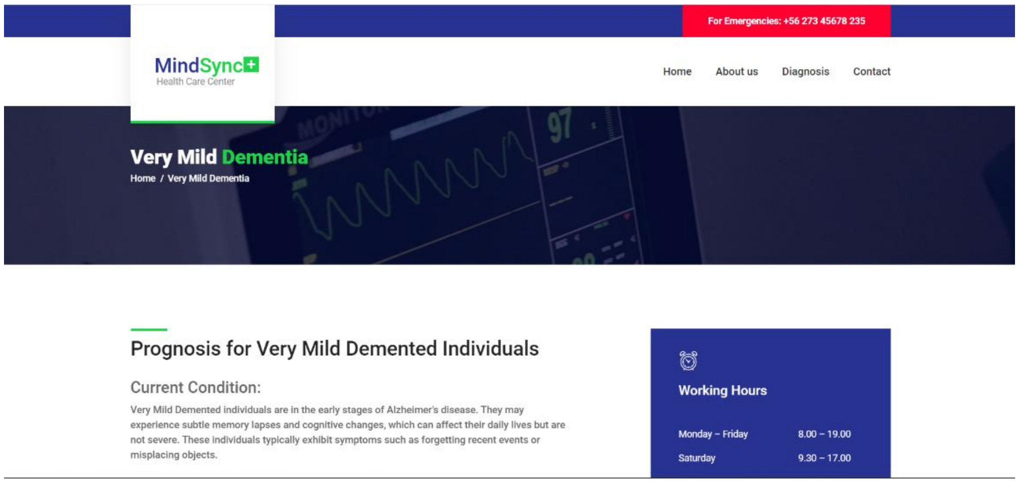
Lifestyle Changes:

Medication and Treatment: Consult a healthcare professional for medication options. Medications like cholinesterase inhibitors or memantine may be considered to manage

Working Hours

Monday – Friday	8.00 – 19.00
Saturday	9.30 – 17.00
Sunday	9.30 – 15.00

2. VeryMildDemented


For Emergencies: +56 273 45678 235
MindSync Health Care Center
Home
About us
Diagnosis
Contact
Very Mild Dementia
Home / Very Mild Dementia

Prognosis for Very Mild Demented Individuals

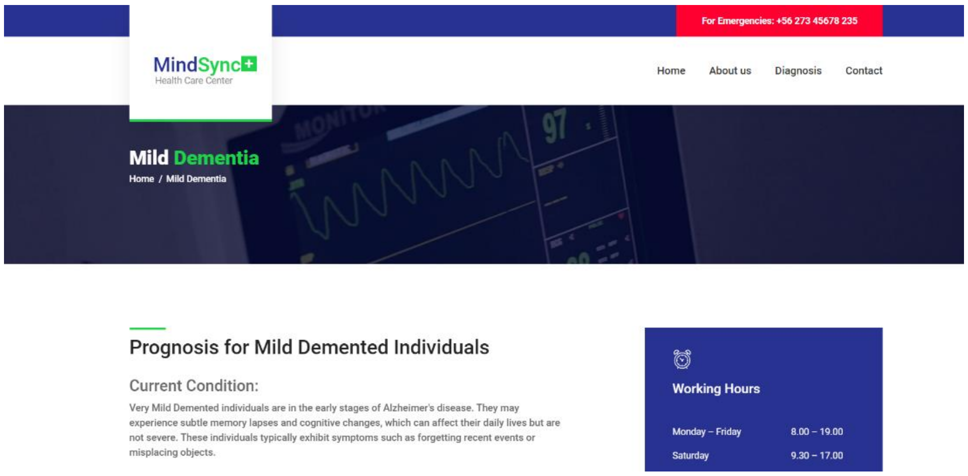
Current Condition:

Very Mild Demented individuals are in the early stages of Alzheimer's disease. They may experience subtle memory lapses and cognitive changes, which can affect their daily lives but are not severe. These individuals typically exhibit symptoms such as forgetting recent events or misplacing objects.

Working Hours

Monday – Friday	8.00 – 19.00
Saturday	9.30 – 17.00

3. MildDemented


For Emergencies: +56 273 45678 235
MindSync Health Care Center
Home
About us
Diagnosis
Contact
Mild Dementia
Home / Mild Dementia

Prognosis for Mild Demented Individuals

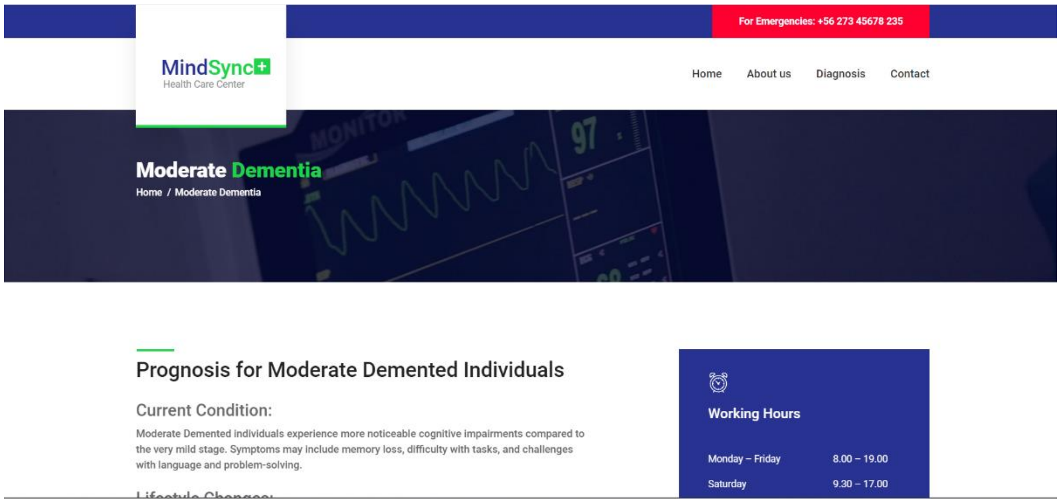
Current Condition:

Very Mild Demented individuals are in the early stages of Alzheimer's disease. They may experience subtle memory lapses and cognitive changes, which can affect their daily lives but are not severe. These individuals typically exhibit symptoms such as forgetting recent events or misplacing objects.

Working Hours

Monday – Friday	8.00 – 19.00
Saturday	9.30 – 17.00

4. ModerateDemented


For Emergencies: +56 273 45678 235

MindSync
Health Care Center

Home About us Diagnosis Contact

Moderate Dementia
Home / Moderate Dementia

Prognosis for Moderate Demented Individuals

Current Condition:

Moderate Demented individuals experience more noticeable cognitive impairments compared to the very mild stage. Symptoms may include memory loss, difficulty with tasks, and challenges with language and problem-solving.

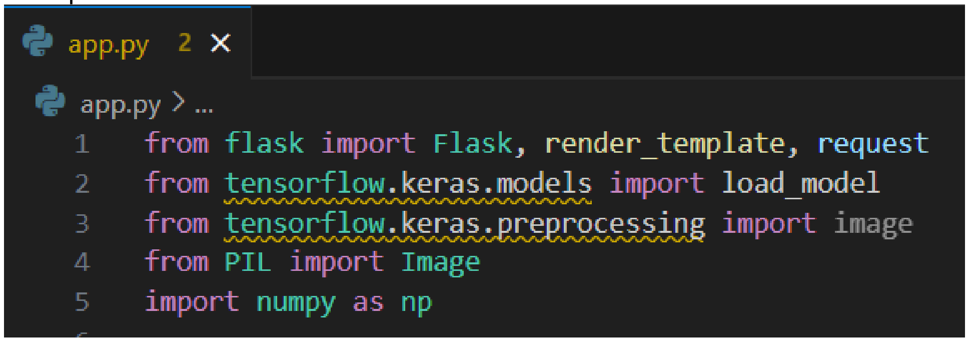
Lifestyle Changes:

Working Hours

Monday – Friday	8.00 – 19.00
Saturday	9.30 – 17.00

Activity 2: Build Python code:

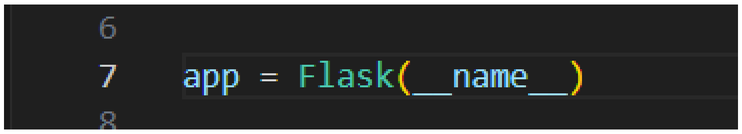
Import the libraries



```
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from PIL import Image
import numpy as np
```

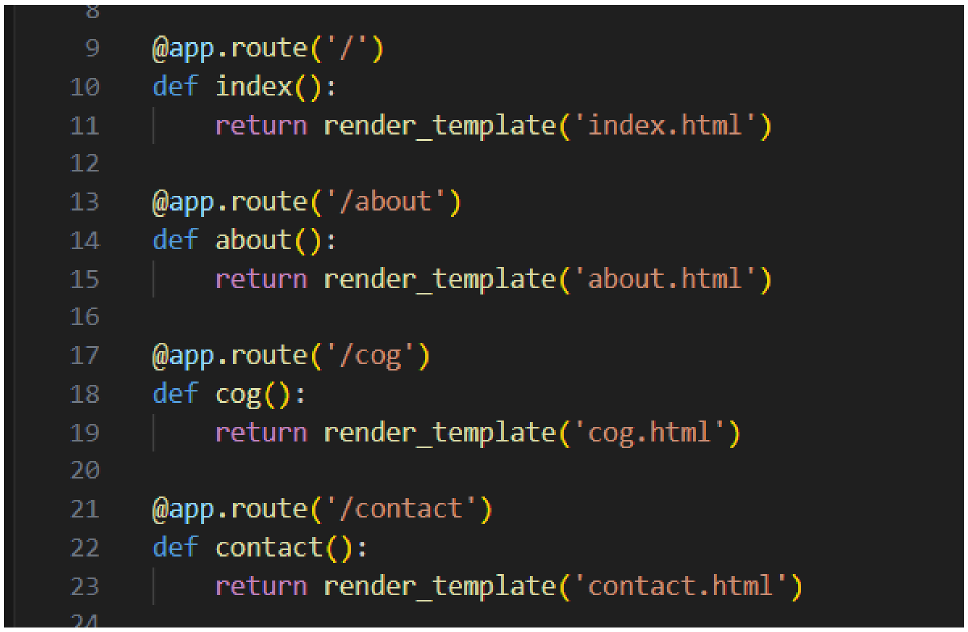
- Import above libraries and modules.

Initializing the flask app



```
app = Flask(__name__)
```

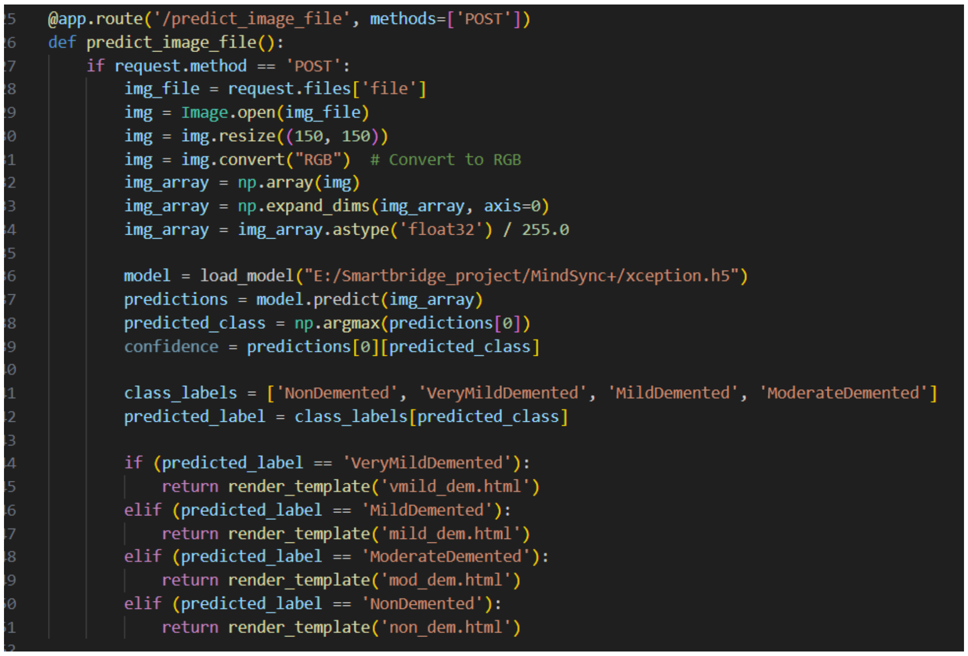
Render HTML pages:



```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/cog')
def cog():
    return render_template('cog.html')

@app.route('/contact')
def contact():
    return render_template('contact.html')
```

- Set the routing pattern of the website for each of the main html pages – index.html, about.html, cog.html and contact.html using the render_template().
- Since the '/' route is bound to the index.html page, when we first open the website, the 'Home Page' is displayed.
- Since the '/about' route is bound to the about.html, when we click on the 'About Us' option we redirect to the 'About Us' page.
- Since the '/cog' route is bound to the cog.html, when we click the 'Diagnosis' option we redirect to the 'Diagnosis' page.

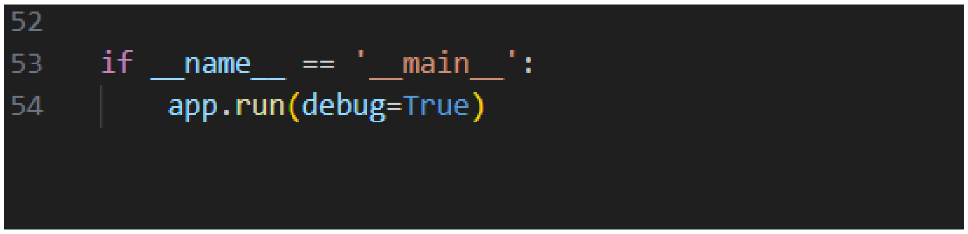
- Since the '/contact' route is bound to the contact.html, when we click the 'Contact' option we redirect to the 'Contact' page.



```
@app.route('/predict_image_file', methods=['POST'])
def predict_image_file():
    if request.method == 'POST':
        img_file = request.files['file']
        img = Image.open(img_file)
        img = img.resize((150, 150))
        img = img.convert("RGB")  # Convert to RGB
        img_array = np.array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array.astype('float32') / 255.0

        model = load_model("E:/Smartbridge_project/MindSync+/xception.h5")
        predictions = model.predict(img_array)
        predicted_class = np.argmax(predictions[0])
        confidence = predictions[0][predicted_class]

        class_labels = ['NonDemented', 'VeryMildDemented', 'MildDemented', 'ModerateDemented']
        predicted_label = class_labels[predicted_class]

        if (predicted_label == 'VeryMildDemented'):
            return render_template('vmild_dem.html')
        elif (predicted_label == 'MildDemented'):
            return render_template('mild_dem.html')
        elif (predicted_label == 'ModerateDemented'):
            return render_template('mod_dem.html')
        elif (predicted_label == 'NonDemented'):
            return render_template('non_dem.html')
```

- The post request is taken, and the files posted in the post method is open and transformations such as resize, rgb coding and Np array createing is done.
- The model is loaded from the Xception.h5 file and the image is predicted, and class of the image is taken in predicted_class
- The Class is used for redirecting the user to the correct page result page using else if page.

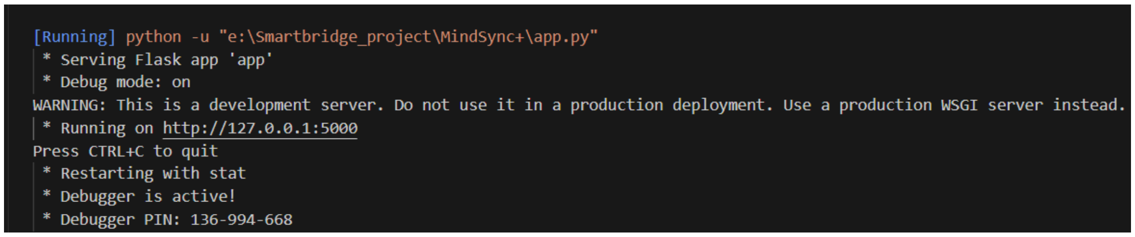
Main Function:



```
if __name__ == '__main__':
    app.run(debug=True)
```

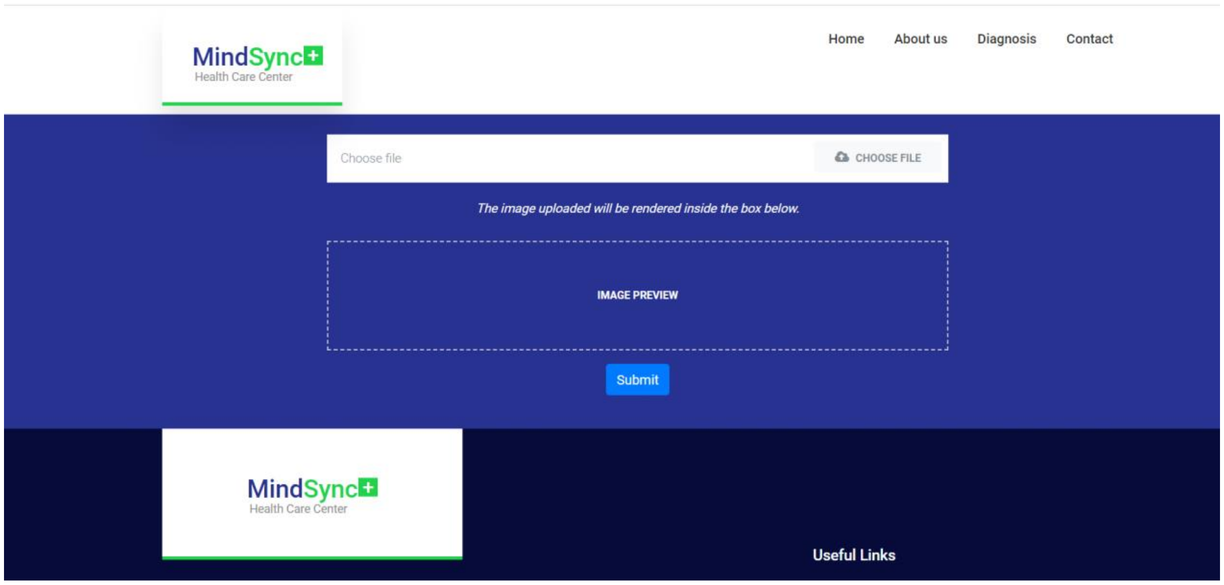
Activity 3: Run the application

- Open VSCode
- Upload all the required files and folders.
- Now click on the ‘run’ button for ‘app.py’.



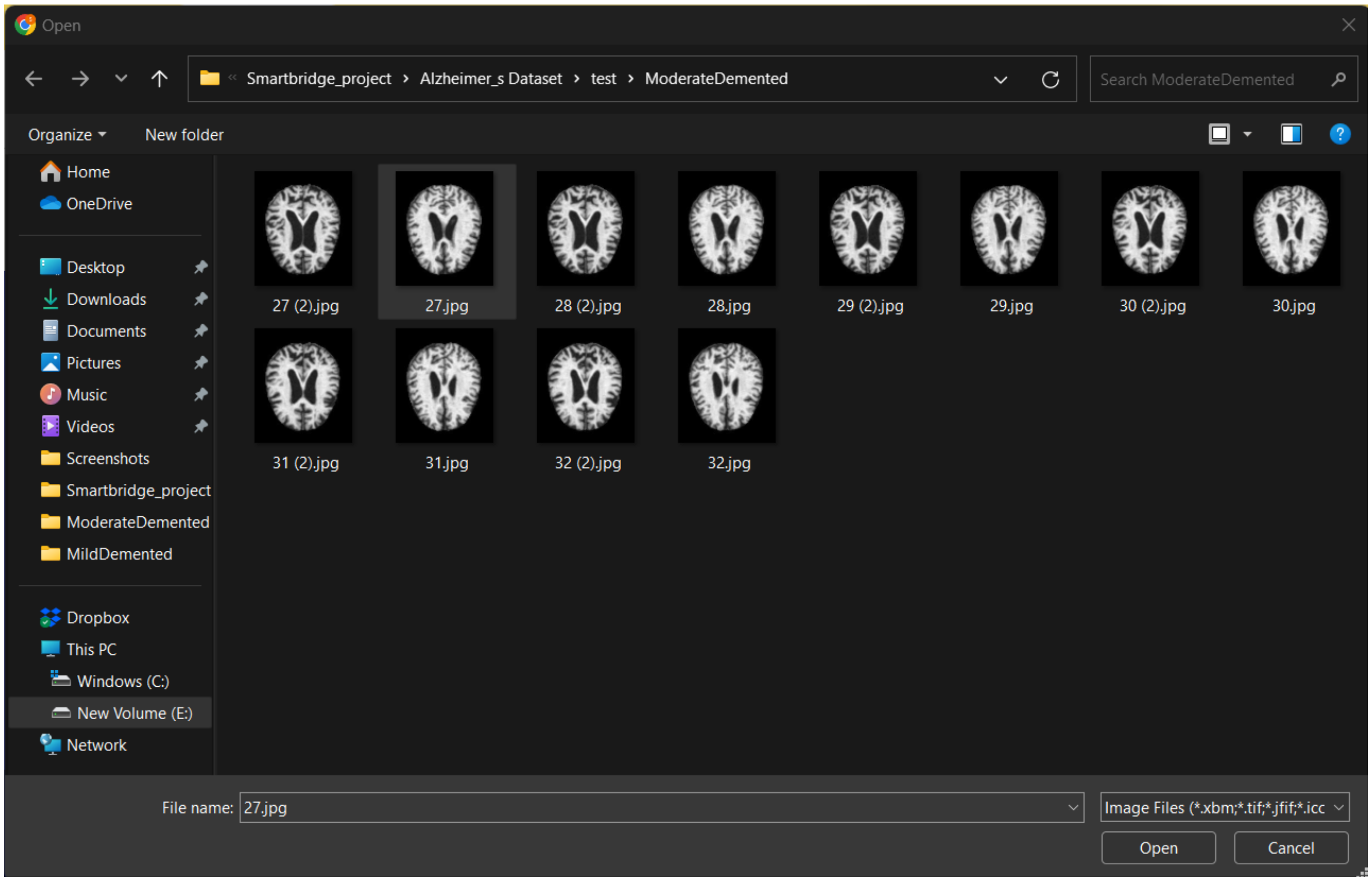
```
[Running] python -u "e:\Smartbridge_project\MindSync+\app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 136-994-668
```

- Visit the website.
- Go to the ‘Diagnosis’ page.
- Click on ‘CHOOSE FILE’ and upload the image that needs to be tested.
- View the preview of the image in the ‘Image Preview’ section.
- Click ‘Submit’ to run the prediction.

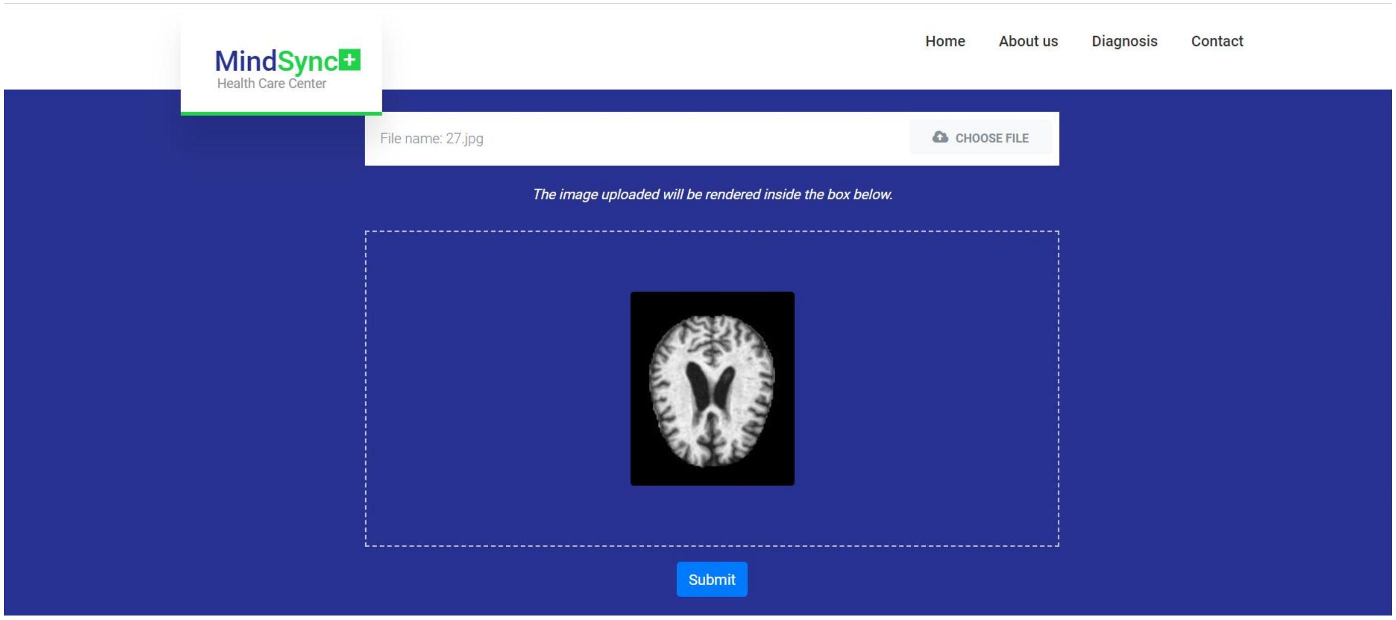


The image uploaded will be rendered inside the box below.

Input:



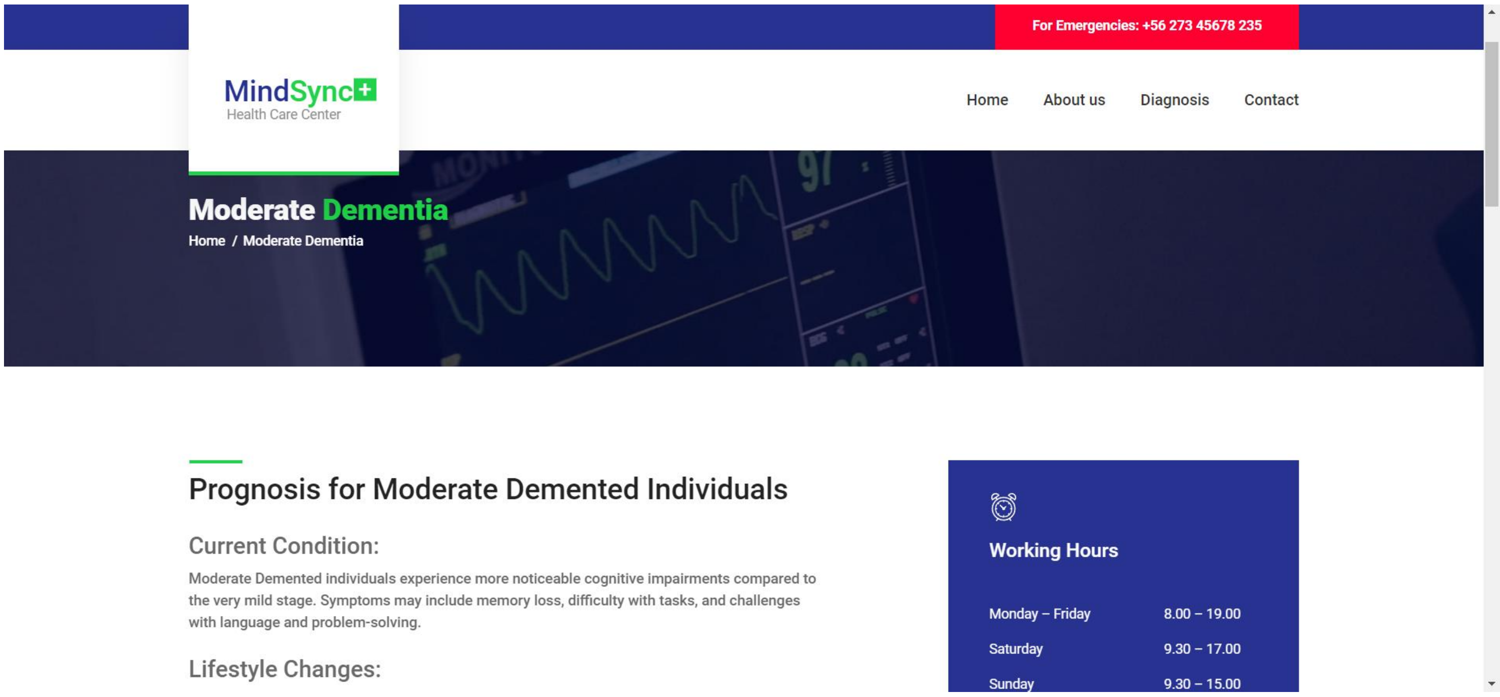
File name: 27.jpg



File name: 27.jpg

The image uploaded will be rendered inside the box below.

Output:



For Emergencies: +56 273 45678 235

MindSync+
Health Care Center

Home About us Diagnosis Contact

Moderate Dementia

Home / Moderate Dementia

Prognosis for Moderate Demented Individuals

Current Condition:

Moderate Demented individuals experience more noticeable cognitive impairments compared to the very mild stage. Symptoms may include memory loss, difficulty with tasks, and challenges with language and problem-solving.

Lifestyle Changes:

Working Hours

Monday – Friday	8.00 – 19.00
Saturday	9.30 – 17.00
Sunday	9.30 – 15.00