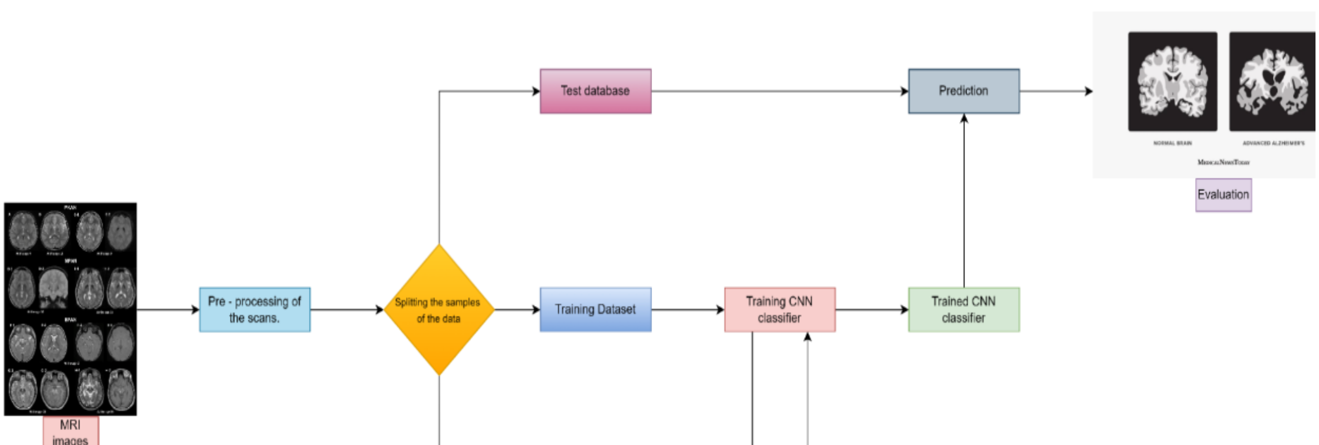
Cognitive Care: Early Intervention for Alzheimer's Disease

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.



Project Flow –

- The user interacts with the UI to choose an image.
- The chosen 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.
- 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.

Data Collection –

1) Dataset Download – https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images

2)Creating a testing and training path

```
In [9]: from tensorflow.keras.layers import Dense, Flatten, Input, Dropout
        from tensorflow.keras.models import Model
        from tensorflow.keras.preprocessing import image
        from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
        from tensorflow.keras.applications.xception import Xception
        import numpy as np
```

```
In [10]: train_path = "C://Users//Chetan 696//Desktop//Alzheimers//Combined Dataset//train"
         test_path = "C://Users//Chetan 696//Desktop//Alzheimers//Combined Dataset//test"
```

3)Image Pre-processing.

- **Import the required library** – In Python, the primary library for working with image data is TensorFlow, and its Keras API provides useful tools for image pre-processing.
- **Configure ImageDataGenerator class** – ImageDataGenerator is a powerful tool in Keras for augmenting and pre-processing image data. It helps generate batches of augmented images during training. Here's an example of configuring ImageDataGenerator
- **Handling imbalance data** – Imbalanced data can lead to biased models. If your classes are unevenly distributed, consider using techniques like class weights or oversampling/undersampling. That is why through data augmentation we have created a balanced dataset.
- **Splitting into train-test split** –To ensure your model generalizes well, split your dataset into training and testing sets. This helps evaluate the model's performance on unseen data. Use train_test_split from scikit-learn: , images' is your input data, 'labels' are the corresponding labels, and the 'test_size' parameter determines the proportion of the dataset used for testing.

```
In [11]: from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
IMG_SIZE=180
IMAGE_SIZE=[180, 180]
DIM=(IMG_SIZE, IMG_SIZE)
ZOOM=[.99, 1.01]
BRIGHT_RANGE=[0.8, 1.2]
HORZ_FLIP=True
FILL_MODE="constant"
DATA_FORMAT="channels_last"
WORK_DIR="C://Users//Chetan 696//Desktop//Alzheimers//Combined Dataset//train"
work_dr = IDG(rescale=1./255, brightness_range=BRIGHT_RANGE, zoom_range=ZOOM, data_format=DATA_FORMAT, fill_mode=FILL_MODE, hori
train_data_gen = work_dr.flow_from_directory(directory=WORK_DIR, target_size=DIM, batch_size=6500, shuffle=False)

Found 10240 images belonging to 4 classes.
```

4)Model Building

- **Pre-trained CNN model as a Feature Extractor –** TensorFlow and Keras provide access to several pre-trained CNN models such as Xception, VGG16, ResNet, and MobileNet. An Xception model based on the project's requirements. Importing the model and removing the top layers (fully connected layers) to use it as a feature extractor:
- **Creating Sequential layers –** Building our own model with additional layers
    - **Dropout (0.5):** Dropout is a regularization technique that randomly sets a fraction of input units to zero during training, which helps prevent overfitting. In this case, it's applied after the xception_model layer.
    - **GlobalAveragePooling2D ():** This layer performs global average pooling on the spatial dimensions of the input. It reduces each spatial dimension (height and width) to 1 by taking the average, resulting in a fixed-size output regardless of the input size.
    - **Flatten ():** This layer is used to flatten the input. It transforms the output of the previous layer (which is the result of global average pooling in this case) into a one-dimensional array, suitable for feeding into a densely connected layer.
    - **Dense (512, activation='relu'):** A densely connected layer with 512 units and ReLU activation function. This layer learns complex patterns in the data.
    - **BatchNormalization ():** Batch normalization normalizes the activations of a layer, helping to stabilize and accelerate the training process.
    - **Dropout (0.5):** Another dropout layer for regularization.
    - **Dense (256, activation='relu'), BatchNormalization (), Dropout (0.5):** Similar structure to the previous dense layer, batch normalization, and dropout layer. These layers further extract and refine features from the data.
    - **Dense (128, activation='relu'), BatchNormalization (), Dropout (0.5):** Another set of dense, batch normalization, and dropout layers.
    - **Dense (64, activation='relu'), Dropout (0.5), BatchNormalization ():** Similar to the previous dense layer, dropout, and batch normalization layers.
    - **Dense (4, activation='softmax'):** The final dense layer with 4 units and a softmax activation function, which is suitable for multi-class classification problems. The output represents the probability distribution over the classes.

- **Configure the Learning Process –** Compiling the model by specifying the loss function, using optimizers such as adam and rmsprop and metrics.
    - 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. Here we are using Adam optimizer.
- **Train the model –** Training the model using the training data generator by ImageDataGenerator for 50 epochs.
- **Save the Model –** We save the model with a .h5 extension. A h5 data file saved as Hierarchal Data Format (HDF) contains multidimensional arrays of data.
- **Test the model –** In the process of development evaluation of the model is important. By taking an image as an input and checking the results.

In [12]:
```
train_data,train_labels=train_data_gen.next()
```

In [13]:
```
print(train_data.shape,train_labels.shape)
```

```
(6500, 180, 180, 3) (6500, 4)
```

In [14]:
```
from sklearn.model_selection import 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)
```

In [15]:
```
from keras.applications import Xception

IMAGE_SIZE = [180, 180]
input_shape = tuple(IMAGE_SIZE + [3])

xception_model = Xception(input_shape=input_shape, include_top=False, weights='imagenet')
```

In [16]:
```
for layer in xception_model.layers:
    layer.trainable = False
```

```
In [17]: from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SeparableConv2D, BatchNormalization, GlobalAveragePooling2D

custom_inception_model = Sequential([
    xception_model,
    Dropout(0.5),
    GlobalAveragePooling2D(),
    Flatten(),
    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="inception_cnn_model")
```

```
In [18]: import tensorflow as tf
METRICS = [
    tf.keras.metrics.CategoricalAccuracy(name='acc'),
    tf.keras.metrics.AUC(name='auc')
]

custom_inception_model.compile(optimizer='rmsprop', loss=tf.losses.CategoricalCrossentropy(), metrics=METRICS)
```

```
Epoch 1/30
130/130 [==============================] - 237s 2s/step - loss: 1.0728 - acc: 0.5712 - auc: 0.8101 - val_loss: 0.5897 - val_ac
c: 0.7904 - val_auc: 0.9412
Epoch 2/30
130/130 [==============================] - 223s 2s/step - loss: 0.5197 - acc: 0.7937 - auc: 0.9524 - val_loss: 0.4227 - val_ac
c: 0.8481 - val_auc: 0.9709
Epoch 3/30
130/130 [==============================] - 221s 2s/step - loss: 0.4067 - acc: 0.8409 - auc: 0.9695 - val_loss: 0.2271 - val_ac
c: 0.9038 - val_auc: 0.9904
Epoch 4/30
130/130 [==============================] - 220s 2s/step - loss: 0.3412 - acc: 0.8656 - auc: 0.9777 - val_loss: 0.2258 - val_ac
c: 0.9125 - val_auc: 0.9898
Epoch 5/30
130/130 [==============================] - 221s 2s/step - loss: 0.3254 - acc: 0.8716 - auc: 0.9798 - val_loss: 0.2840 - val_ac
c: 0.8808 - val_auc: 0.9851
Epoch 6/30
130/130 [==============================] - 222s 2s/step - loss: 0.3062 - acc: 0.8837 - auc: 0.9818 - val_loss: 0.2463 - val_ac
c: 0.9058 - val_auc: 0.9881
Epoch 7/30
130/130 [==============================] - 221s 2s/step - loss: 0.2900 - acc: 0.8947 - auc: 0.9837 - val_loss: 0.1909 - val_ac
c: 0.9250 - val_auc: 0.9921
Epoch 8/30
130/130 [==============================] - 269s 2s/step - loss: 0.2885 - acc: 0.8974 - auc: 0.9837 - val_loss: 0.1980 - val_ac
c: 0.9250 - val_auc: 0.9918
Epoch 9/30
130/130 [==============================] - 343s 3s/step - loss: 0.2700 - acc: 0.9014 - auc: 0.9856 - val_loss: 0.1797 - val_ac
c: 0.9346 - val_auc: 0.9939
Epoch 10/30
130/130 [==============================] - 332s 3s/step - loss: 0.2731 - acc: 0.8981 - auc: 0.9856 - val_loss: 0.2854 - val_ac
c: 0.8788 - val_auc: 0.9844
Epoch 11/30
130/130 [==============================] - 277s 2s/step - loss: 0.2733 - acc: 0.8988 - auc: 0.9853 - val_loss: 0.2028 - val_ac
c: 0.9250 - val_auc: 0.9921
Epoch 12/30
130/130 [==============================] - 252s 2s/step - loss: 0.2495 - acc: 0.9096 - auc: 0.9875 - val_loss: 0.1570 - val_ac
c: 0.9337 - val_auc: 0.9953
Epoch 13/30
130/130 [==============================] - 241s 2s/step - loss: 0.2558 - acc: 0.9007 - auc: 0.9870 - val_loss: 0.2161 - val_ac
c: 0.9077 - val_auc: 0.9906
Epoch 14/30
130/130 [==============================] - 220s 2s/step - loss: 0.2529 - acc: 0.9058 - auc: 0.9876 - val_loss: 0.1876 - val_ac
c: 0.9269 - val_auc: 0.9929
Epoch 15/30
130/130 [==============================] - 240s 2s/step - loss: 0.2347 - acc: 0.9094 - auc: 0.9890 - val_loss: 0.2738 - val_ac
c: 0.9048 - val_auc: 0.9856
Epoch 16/30
130/130 [==============================] - 249s 2s/step - loss: 0.2281 - acc: 0.9149 - auc: 0.9895 - val_loss: 0.2123 - val_ac
c: 0.9212 - val_auc: 0.9907
Epoch 17/30
130/130 [==============================] - 249s 2s/step - loss: 0.2281 - acc: 0.9123 - auc: 0.9890 - val_loss: 0.2993 - val_ac
c: 0.8808 - val_auc: 0.9842
Epoch 18/30
130/130 [==============================] - 246s 2s/step - loss: 0.2090 - acc: 0.9190 - auc: 0.9911 - val_loss: 0.2067 - val_ac
c: 0.9192 - val_auc: 0.9906
Epoch 19/30
130/130 [==============================] - 248s 2s/step - loss: 0.2235 - acc: 0.9171 - auc: 0.9900 - val_loss: 0.2335 - val_ac
c: 0.9135 - val_auc: 0.9902
Epoch 20/30
130/130 [==============================] - 250s 2s/step - loss: 0.2350 - acc: 0.9118 - auc: 0.9884 - val_loss: 0.1431 - val_ac
c: 0.9442 - val_auc: 0.9960
```

```
Epoch 21/30
130/130 [==============================] - 236s 2s/step - loss: 0.2161 - acc: 0.9163 - auc: 0.9906 - val_loss: 0.1504 - val_ac
c: 0.9423 - val_auc: 0.9956
Epoch 22/30
130/130 [==============================] - 237s 2s/step - loss: 0.2186 - acc: 0.9187 - auc: 0.9897 - val_loss: 0.1814 - val_ac
c: 0.9250 - val_auc: 0.9937
Epoch 23/30
130/130 [==============================] - 246s 2s/step - loss: 0.2092 - acc: 0.9262 - auc: 0.9910 - val_loss: 0.1308 - val_ac
c: 0.9442 - val_auc: 0.9966
Epoch 24/30
130/130 [==============================] - 251s 2s/step - loss: 0.2143 - acc: 0.9216 - auc: 0.9904 - val_loss: 0.1562 - val_ac
c: 0.9288 - val_auc: 0.9953
Epoch 25/30
130/130 [==============================] - 251s 2s/step - loss: 0.2030 - acc: 0.9250 - auc: 0.9909 - val_loss: 0.1662 - val_ac
c: 0.9327 - val_auc: 0.9945
Epoch 26/30
130/130 [==============================] - 252s 2s/step - loss: 0.2224 - acc: 0.9185 - auc: 0.9897 - val_loss: 0.1892 - val_ac
c: 0.9240 - val_auc: 0.9932
Epoch 27/30
130/130 [==============================] - 246s 2s/step - loss: 0.2234 - acc: 0.9171 - auc: 0.9892 - val_loss: 0.1495 - val_ac
c: 0.9404 - val_auc: 0.9957
Epoch 28/30
130/130 [==============================] - 247s 2s/step - loss: 0.2194 - acc: 0.9137 - auc: 0.9904 - val_loss: 0.1832 - val_ac
c: 0.9288 - val_auc: 0.9922
Epoch 29/30
130/130 [==============================] - 240s 2s/step - loss: 0.1999 - acc: 0.9264 - auc: 0.9911 - val_loss: 0.1511 - val_ac
c: 0.9346 - val_auc: 0.9951
Epoch 30/30
130/130 [==============================] - 139s 1s/step - loss: 0.1927 - acc: 0.9286 - auc: 0.9917 - val_loss: 0.1379 - val_ac
c: 0.9452 - val_auc: 0.9962
```

```
custom_inception_model.save("Xception_model_image.h5")
```

50 Epoch –

```
history=custom_inception_model.fit(train_data,train_labels,validation_data=(val_data,val_labels), epochs=50)
```

```
Epoch 1/50
84/84 [==============================] - 87s 999ms/step - loss: 1.1858 - acc: 0.5199 - auc: 0.7643 - val_loss: 0.5679 - val_ac
c: 0.7988 - val_auc: 0.9552
```

```
Epoch 50/50
84/84 [==============================] - 84s 1s/step - loss: 0.1723 - acc: 0.9425 - auc: 0.9933 - val_loss: 0.1581 - val_acc:
0.9429 - val_auc: 0.9940
```

Val_Acc = 99.4%

30 Epoch –

```
history=custom_inception_model.fit(train_data,train_labels,validation_data=(val_data,val_labels), epochs=30)
```

```
Epoch 1/30
130/130 [==============================] - 149s 1s/step - loss: 1.0240 - acc: 0.6002 - auc: 0.8284 - val_loss: 0.5805 - val_ac
c: 0.7923 - val_auc: 0.9429

Epoch 30/30
130/130 [==============================] - 241s 2s/step - loss: 0.1967 - acc: 0.9240 - auc: 0.9918 - val_loss: 0.1481 - val_ac
c: 0.9365 - val_auc: 0.9958
```

Final Val_Acc = 99.58%

Saving The Model –

```
custom_inception_model.save("Xception_model.h5")
```

```
C:\Users\Chetan 696\anaconda3\Lib\site-packages\keras\src\engine\training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

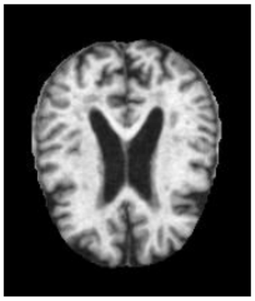
Testing The Model –

```
import numpy as np
from keras.preprocessing import image
from keras.applications.xception import Xception, preprocess_input, decode_predictions
import numpy as np
from keras.preprocessing import image
from keras.models import load_model
```

```
model_path = "C://Users//Chetan 696//Desktop//Alzheimers//Xception_model_image.h5"
model = load_model(model_path)
```

Mild Demented –

```
from IPython.display import display
from PIL import Image

# Use the file uploader to select an image
uploaded_image_path = "C://Users//Chetan 696//Desktop//Alzheimers//Alzheimer_s Dataset//test//MildDemented//28.jpg"
img = Image.open(uploaded_image_path)
display(img)
```

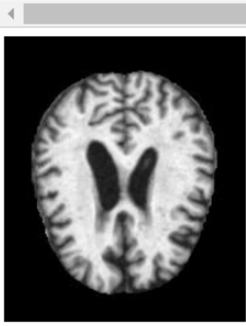


```python
from PIL import Image
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input
import numpy as np

# Load the image
img_path = "C://Users//Chetan 696//Desktop//Alzheimers//Alzheimer_s Dataset//test//MildDemented//28.jpg"
img = Image.open(img_path)

# Convert the image to RGB if it's grayscale
img = img.convert('RGB')

# Resize the image to match the model's input shape (180, 180)
img = img.resize((180, 180))

# Convert the image to array and preprocess it
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
img_data = preprocess_input(x)

# Now you can use img_data with your model
output = np.argmax(model.predict(img_data), axis=1)

class_labels = ["MildDemented", "ModerateDemented", "NonDemented", "VeryMildDemented"]
predicted_class = class_labels[output[0]]

print("Predicted class:", predicted_class)
```

```
1/1 [==============================] - 0s 63ms/step
Predicted class: MildDemented
```

Moderate Demented –

```python
from IPython.display import display
from PIL import Image

# Use the file uploader to select an image
uploaded_image_path = "C://Users//Chetan 696//Desktop//Alzheimers//Alzheimer_s Dataset//test//ModerateDemented//28.jpg"
img = Image.open(uploaded_image_path)
display(img)
```

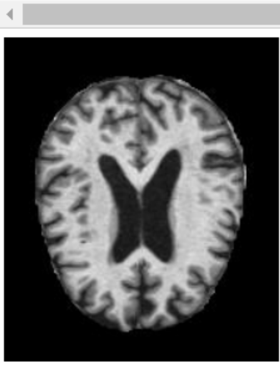


```python
from PIL import Image
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input
import numpy as np

# Load the image
img_path = "C://Users//Chetan 696//Desktop//Alzheimers//Alzheimer_s Dataset//test//ModerateDemented//28.jpg"
img = Image.open(img_path)

# Convert the image to RGB if it's grayscale
img = img.convert('RGB')

# Resize the image to match the model's input shape (180, 180)
img = img.resize((180, 180))

# Convert the image to array and preprocess it
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
img_data = preprocess_input(x)

# Now you can use img_data with your model
output = np.argmax(model.predict(img_data), axis=1)

class_labels = ["MildDemented", "ModerateDemented", "NonDemented", "VeryMildDemented"]
predicted_class = class_labels[output[0]]

print("Predicted class:", predicted_class)
```

```
1/1 [==============================] - 0s 64ms/step
Predicted class: ModerateDemented
```

Non-Demented –

```python
from IPython.display import display
from PIL import Image

# Use the file uploader to select an image
uploaded_image_path = "C://Users//Chetan 696//Desktop//Alzheimers//Alzheimer_s Dataset//test//NonDemented//26 (75).jpg"
img = Image.open(uploaded_image_path)
display(img)
```

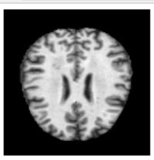


```python
from PIL import Image
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input
import numpy as np

# Load the image
img_path = "C://Users//Chetan 696//Desktop//Alzheimers//Alzheimer_s Dataset//test//NonDemented//26 (75).jpg"
img = Image.open(img_path)

# Convert the image to RGB if it's grayscale
img = img.convert('RGB')

# Resize the image to match the model's input shape (180, 180)
img = img.resize((180, 180))

# Convert the image to array and preprocess it
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
img_data = preprocess_input(x)

# Now you can use img_data with your model
output = np.argmax(model.predict(img_data), axis=1)

class_labels = ["MildDemented", "ModerateDemented", "NonDemented", "VeryMildDemented"]
predicted_class = class_labels[output[0]]

print("Predicted class:", predicted_class)
```

```
1/1 [==============================] - 0s 60ms/step
Predicted class: NonDemented
```

Very Mild Demented –

```python
from IPython.display import display
from PIL import Image

# Use the file uploader to select an image
uploaded_image_path = "C://Users//Chetan 696//Desktop//Alzheimers//Combined Dataset//test//Very Mild Impairment//30 (12).jpg"
img = Image.open(uploaded_image_path)
display(img)
```



```python
from PIL import Image
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input
import numpy as np

# Load the image
img_path = "C://Users//Chetan 696//Desktop//Alzheimers//Combined Dataset//test//Very Mild Impairment//30 (12).jpg"
img = Image.open(img_path)

# Convert the image to RGB if it's grayscale
img = img.convert('RGB')

# Resize the image to match the model's input shape (180, 180)
img = img.resize((180, 180))

# Convert the image to array and preprocess it
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
img_data = preprocess_input(x)

# Now you can use img_data with your model
output = np.argmax(model.predict(img_data), axis=1)

class_labels = ["VeryMildDemented", "ModerateDemented", "NonDemented", "MildDemented"]
predicted_class = class_labels[output[0]]

print("Predicted class:", predicted_class)
```

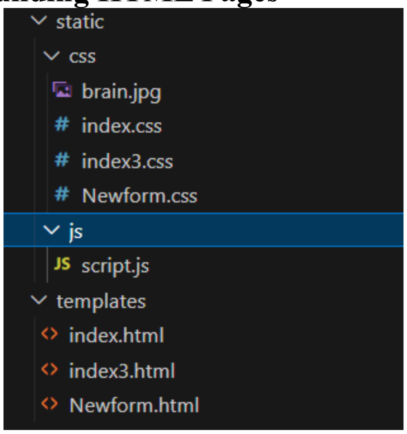
```
1/1 [==============================] - 1s 686ms/step
Predicted class: VeryMildDemented
```

Application Building –

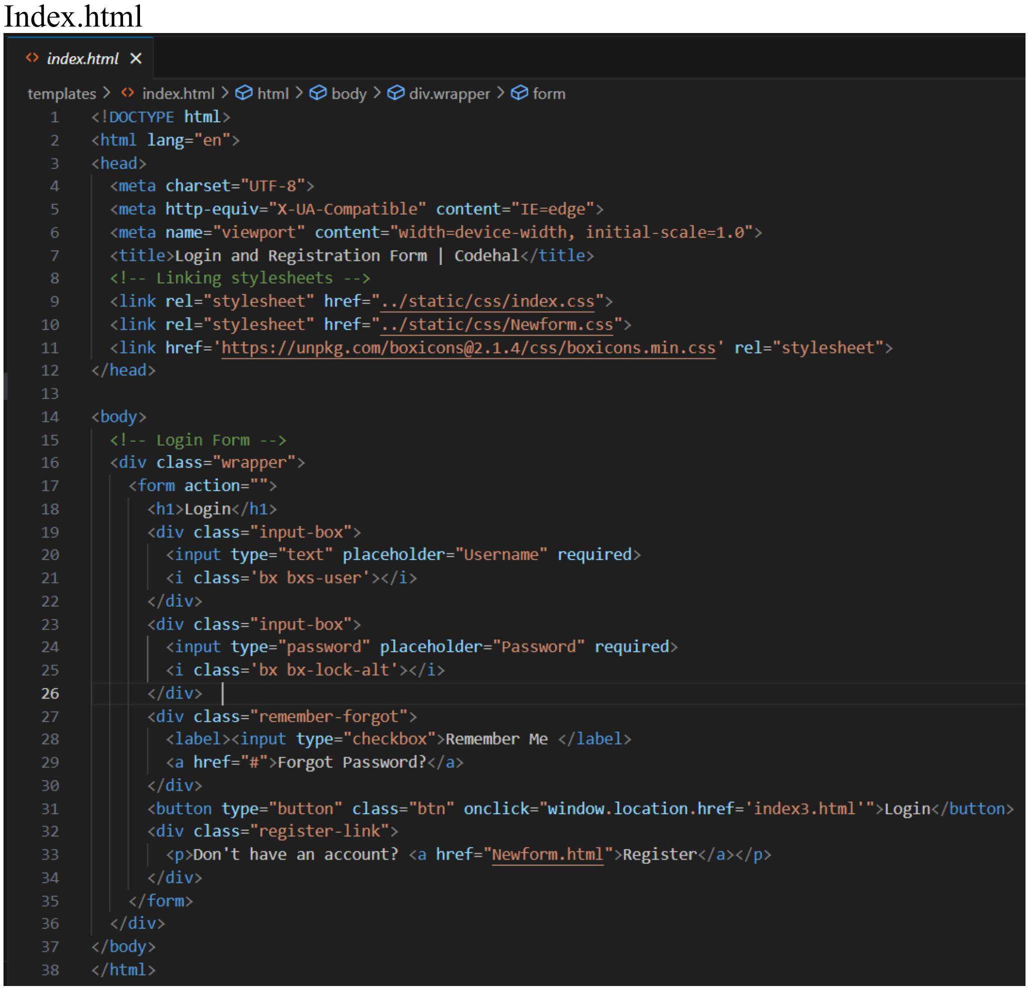
We have built a web application that is integrated to the model we built. An UI is provided for the users where they have to upload an image (MRI Scan) for prediction. The image is then run through the saved model and the predicted output is shown in the UI.

Steps –

1. **Building HTML Pages**



- Index.html



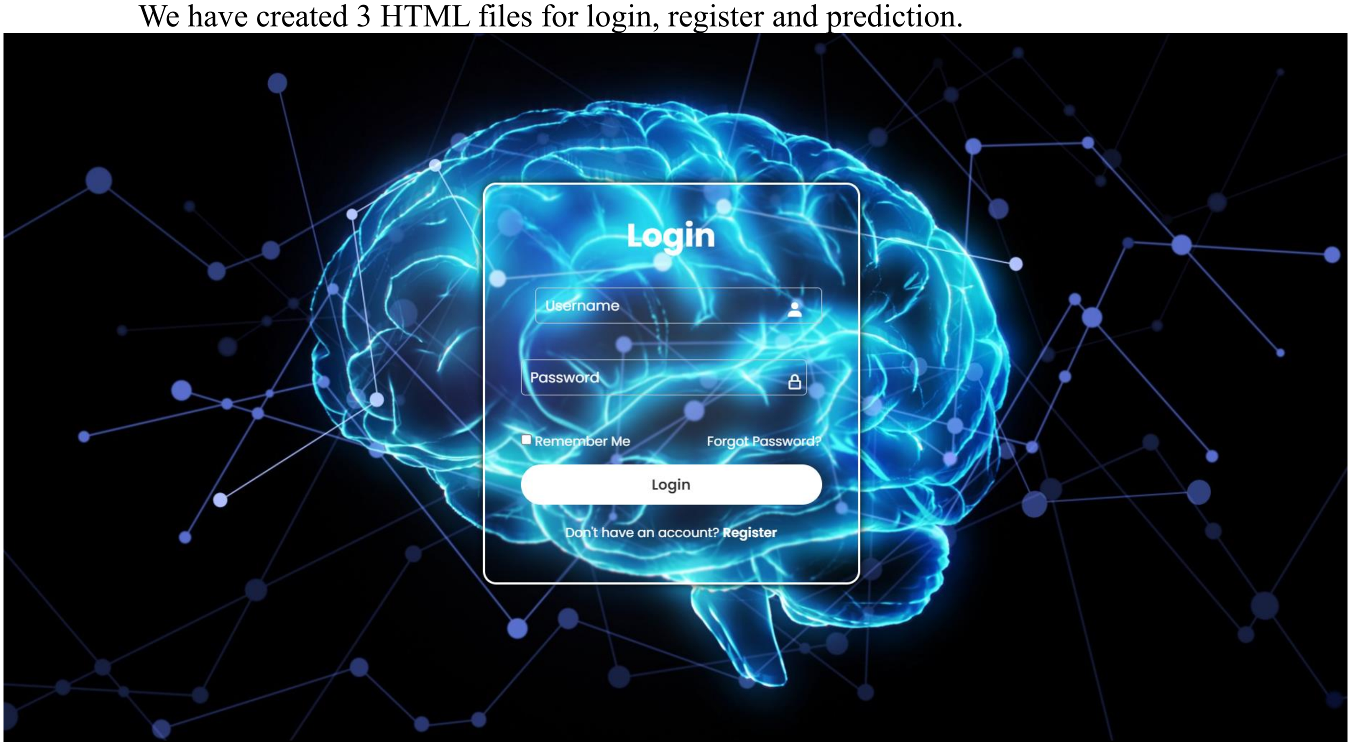
```
index.html ×
templates > index.html > html > body > div.wrapper > form
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login and Registration Form | Codehal</title>
  <!-- Linking stylesheets -->
  <link rel="stylesheet" href="../static/css/index.css">
  <link rel="stylesheet" href="../static/css/Newform.css">
  <link href='https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css' rel="stylesheet">
</head>

<body>
  <!-- Login Form -->
  <div class="wrapper">
    <form action="">
      <h1>Login</h1>
      <div class="input-box">
        <input type="text" placeholder="Username" required>
        <i class='bx bxs-user'></i>
      </div>
      <div class="input-box">
        <input type="password" placeholder="Password" required>
        <i class='bx bx-lock-alt'></i>
      </div>
      <div class="remember-forgot">
        <label><input type="checkbox">Remember Me </label>
        <a href="#">Forgot Password?</a>
      </div>
      <button type="button" class="btn" onclick="window.location.href='index3.html'">Login</button>
      <div class="register-link">
        <p>Don't have an account? <a href="Newform.html">Register</a></p>
      </div>
    </form>
  </div>
</body>
</html>
```

CSS Code –

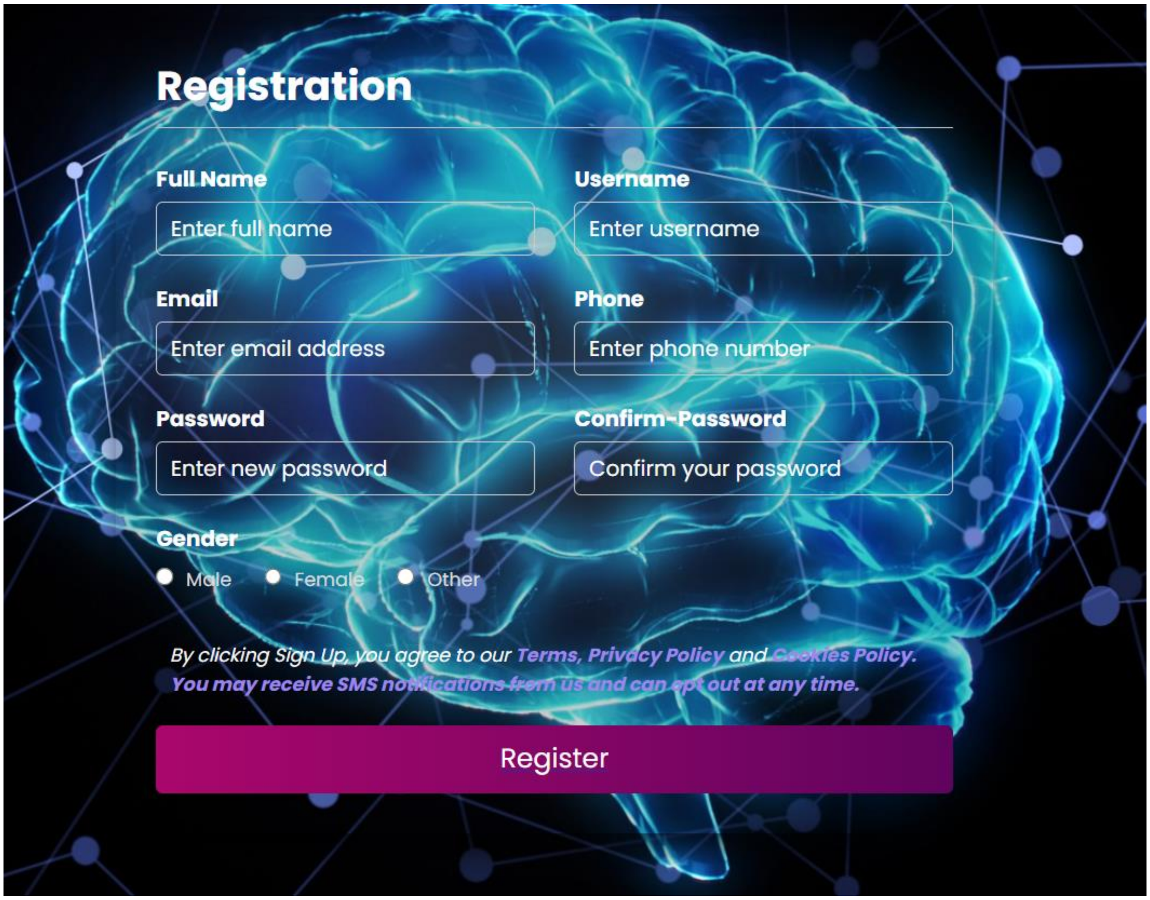
```
index.css ×
static > css > # index.css > *
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: "Poppins", sans-serif;
}

body {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  background: url("https://altoida.com/wp-content/uploads/2022/06/shutterstock_1638621172.jpg") no-repeat;
  background-size: cover;
  background-position: center;
}

.wrapper {
    width: 420px;
    background: transparent;
    border: 3px solid white;
    backdrop-filter: blur(1px);
    box-shadow: 0 0 10px rgba(0,0,0.2);
    color: #fff;
    border-radius: 15px;
    padding: 30px 40px;
  }

  .wrapper h1 {
    font-size: 36px;
    text-align: center;
  }

  .wrapper .input-box {
    position: relative;
    width: 100%;
    height: 50px;
    margin: 30px 0;
  }
```

2. Building Python Code

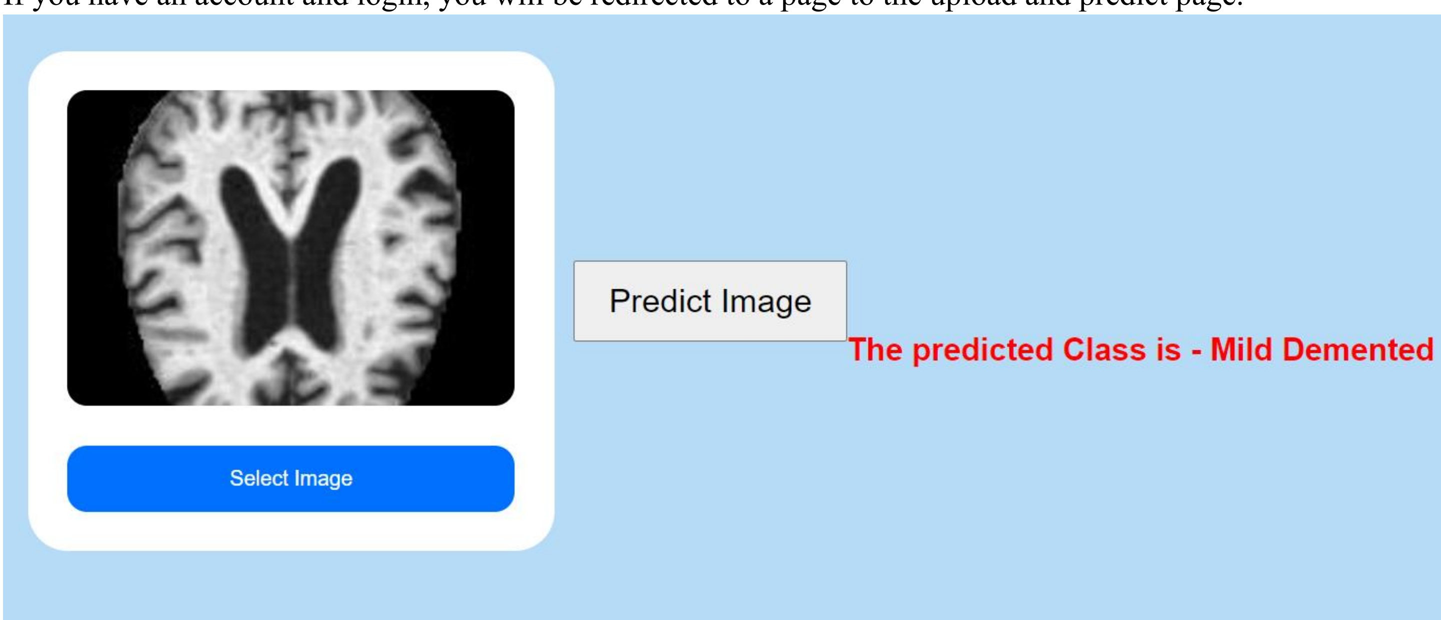
We have created 3 HTML files for login, register and prediction.

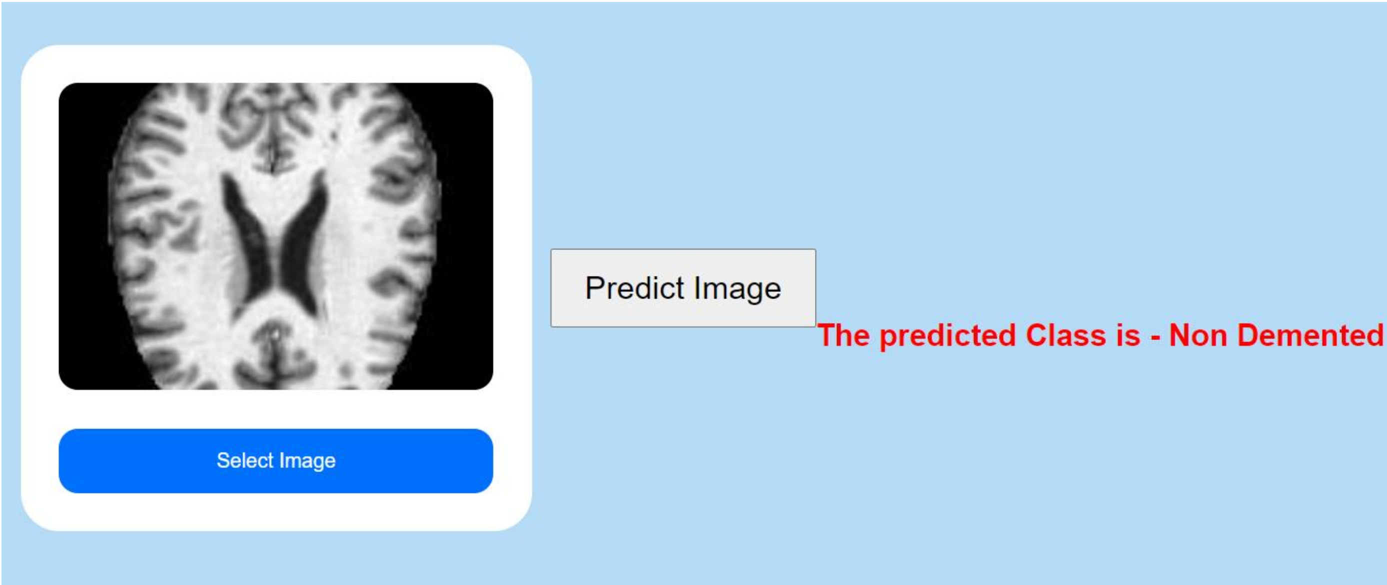


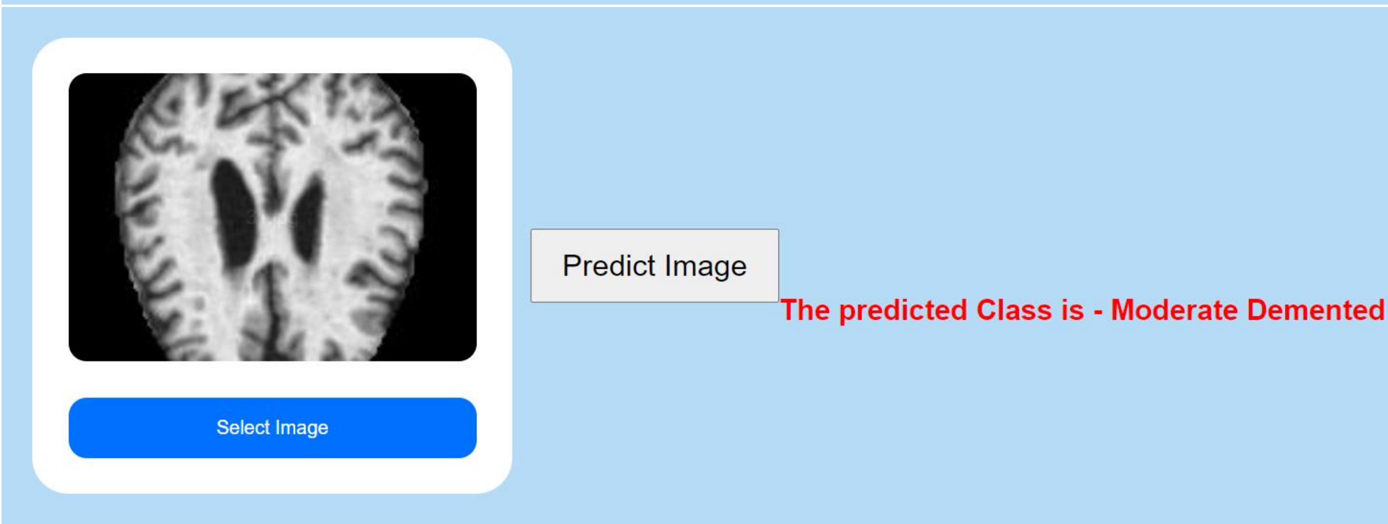
If you don't have an account, you will be redirected to a register account page.

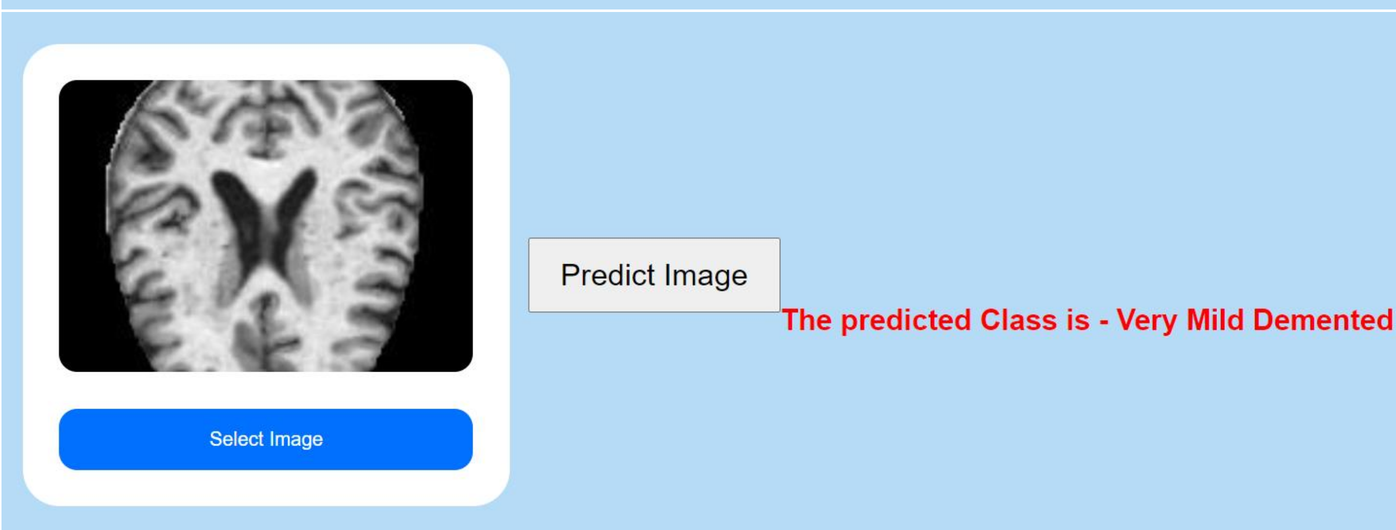


If you have an account and login, you will be redirected to a page to the upload and predict page.




Predict Image
The predicted Class is - Non Demented
Select Image


Predict Image
The predicted Class is - Moderate Demented
Select Image


Predict Image
The predicted Class is - Very Mild Demented
Select Image