

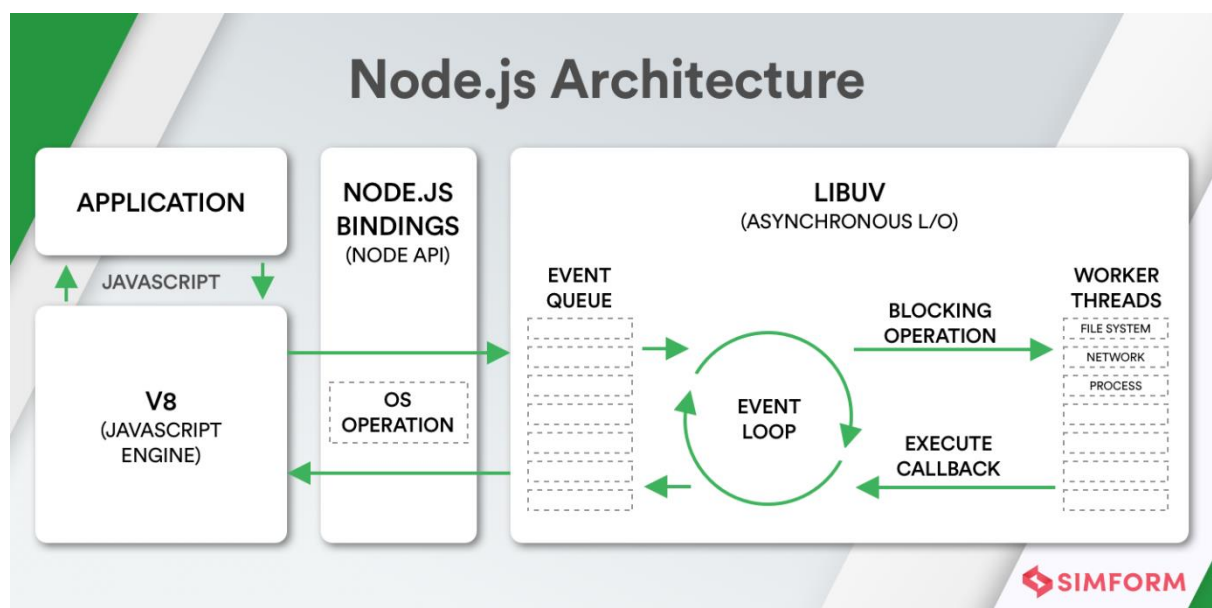
## Project Description:

The Alzheimer's Disease Detection System is an innovative project designed to tackle the challenges presented by Alzheimer's disease (AD), a progressive and irreversible neurological disorder affecting memory, cognition, and behavior. As the most common cause of dementia among older adults, AD is characterized by abnormal protein deposits in the brain, such as amyloid plaques and tau tangles. While the exact cause of AD remains incompletely understood, factors like genetics, environment, and lifestyle are believed to contribute, with age being a significant risk factor, particularly after the age of 65.

The project leverages deep learning models, specifically utilizing Xception, to analyze medical imaging data for the early detection of Alzheimer's disease. By identifying subtle signs before symptoms become severe, the system aims to provide healthcare providers with valuable tools for early intervention and support. Early symptoms may include mild memory loss and difficulty with problem-solving, progressing to severe memory impairment and an inability to perform daily activities. This proactive approach to detection holds the promise of improving outcomes for patients and their families, offering the potential for early treatment and support.

Through the integration of advanced deep learning techniques into the healthcare landscape, the Alzheimer's Disease Detection System seeks to enhance diagnostic capabilities, enabling timely interventions that may ultimately lead to better outcomes and an improved quality of life for those affected by Alzheimer's disease.

## Technical Architecture:



## Project Flow:

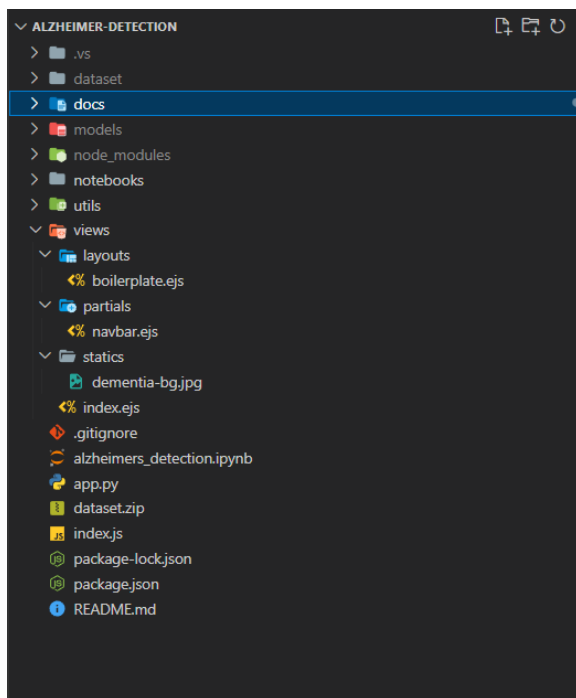
- The user interacts with the UI to choose an image.
- The chosen image is sent through a protocol as a request body
- The flask API receives the image from the client
- The flask API then uses the model to predict and send the prediction as a response
- The predictions are displayed on the website using chartJS run on nodeJS.
- This process enables users to input an image and receive accurate predictions quickly.

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

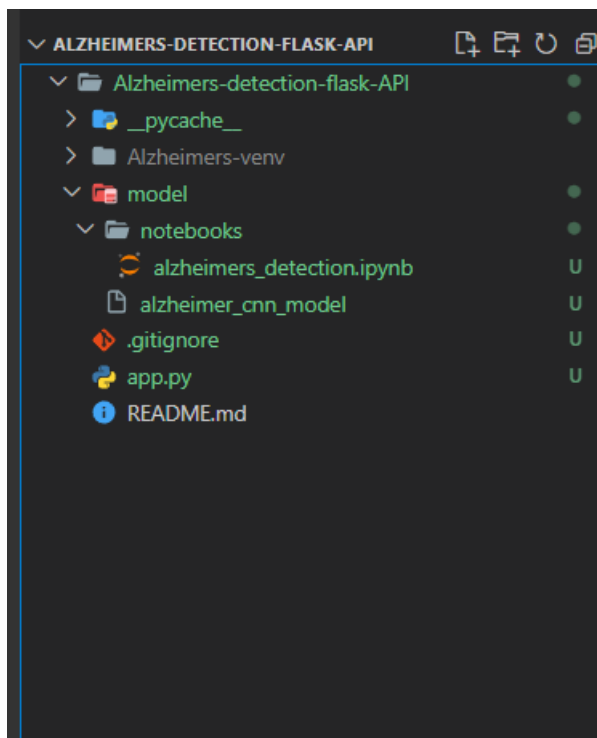
1. Data Collection.
2. Create a Train and Test path.
3. Image Pre-processing.
4. Import the required library
5. Configure ImageDataGenerator class
6. Handling imbalance data
7. Splitting into train-test split
8. Model Building
9. Pre-trained CNN model as a Feature Extractor
10. Creating Sequential layers
11. Configure the Learning Process
12. Train the model
13. Save the Model
14. Test the model
15. Application Building
16. Build a flask API to handle incoming images and run prediction on it
17. Build an expressJS server and ejs template to implement UI
18. Use fetch to the flaskAPI to send image scan and get predictions
19. Run the application

## Project Structure:

### 1)NodeJS structure:



### 2)flask API:



## Milestone 1: Data Collection

In the initial phase of our project, we focused on gathering a diverse and representative dataset crucial for training our Alzheimer's detection model. Leveraging reputable open sources such as Kaggle and the UCI repository, we obtained a comprehensive set of brain MRI images. These images, categorized into four distinct types – Mild Demented, Moderate Demented, Non-Demented, and Very Mild Demented – were meticulously organized into subdirectories as outlined in the project structure. This meticulous organization facilitates a streamlined approach to dataset management.

For simplicity, we have made the dataset used in this project accessible through the following link: [Alzheimer's Dataset \(4 classes of Images\)](#).

*Note: To ensure optimal model accuracy, we encourage the inclusion of additional images in the training set.*

```
drive sample_data

from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

To facilitate seamless integration with Google Colab, where our training model will be constructed, we recommend uploading the dataset as a compressed zip file. Follow these steps on Google Colab to achieve this:

1. Open a new notebook on Google Colab.
2. Navigate to the "Files" icon on the left-hand panel.
3. Select "Upload" and choose the zip file containing the dataset.
4. Wait for the upload to conclude, and confirm the file's presence in the "Files" section.

To unzip the file, execute the appropriate command in a code cell.

```
!unzip "/content/drive/MyDrive/Alzheimers detection/dataset.zip" -d "/content"
```

Streaming output truncated to the last 5000 lines.

```
inflating: /content/dataset/train/MildDemented/mildDem21.jpg
inflating: /content/dataset/train/MildDemented/mildDem210.jpg
inflating: /content/dataset/train/MildDemented/mildDem211.jpg
inflating: /content/dataset/train/MildDemented/mildDem212.jpg
inflating: /content/dataset/train/MildDemented/mildDem213.jpg
inflating: /content/dataset/train/MildDemented/mildDem214.jpg
inflating: /content/dataset/train/MildDemented/mildDem215.jpg
inflating: /content/dataset/train/MildDemented/mildDem216.jpg
inflating: /content/dataset/train/MildDemented/mildDem217.jpg
```

## Activity 2: Dataset Preparation

In preparation for building our deep learning model, we focused on creating distinct training and testing datasets. In our project dataset folder, we observed the existence of pre-arranged training and testing folders, streamlining this crucial step. We assigned variables to the respective folder paths, ensuring a seamless transition into the model development phase. This strategic organization not only simplifies dataset management but also adheres to best practices in machine learning model training.

This meticulous approach sets the stage for the subsequent milestones, where the model will undergo training on the enriched dataset, paving the way for robust Alzheimer's disease detection capabilities.

## Activity 1: Importing the libraries

Import the necessary libraries as shown in the image

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
from matplotlib import pyplot as plt
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
```

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.

- **Numpy:** It is for performing mathematical functions

## 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. 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.
- 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\_datagen object to generate

- batches of augmented training data on the fly, as needed by the fit() method of a Keras model.

```
train_datagen=ImageDataGenerator(rescale=1./255)
test_datagen=ImageDataGenerator(rescale=1./255)
```

#### Activity 4: Splitting into train test split

```
test=test_datagen.flow_from_directory("dataset/test",
                                     classes=classNames,
                                     color_mode="grayscale",
                                     class_mode="categorical",
                                     batch_size=32,
                                     keep_aspect_ratio=True)

Found 1279 images belonging to 4 classes.

train=train_datagen.flow_from_directory("dataset/train",
                                       classes=classNames,
                                       color_mode="grayscale",
                                       class_mode="categorical",
                                       batch_size=32,
                                       keep_aspect_ratio=True)
```

Now we are splitting the labels into train-test split in 80:20 ratio & assigning them to train\_data, train\_labels & test\_data, test\_labels. Validation sets of train\_data, val\_data, train\_labels, val\_labels.

## Milestone 3: Model Building

In this stage of our project, the focus shifts to constructing an effective model for Alzheimer's disease detection. Instead of leveraging a pre-trained model like Xception, our custom model is designed using the Keras Sequential API, tailored to the specific requirements of our task. The provided code snippet illustrates the architecture of our Convolutional Neural Network (CNN).

The model begins with a sequence of convolutional layers, each followed by a rectified linear unit (ReLU) activation function for non-linearity. Max-pooling layers are interspersed to down-sample spatial dimensions. To combat overfitting, dropout layers are strategically placed, preventing excessive reliance on specific neurons during training. The flattened layer prepares the data for transition to densely connected layers, introducing higher-level features.

Following the convolutional layers, the model incorporates densely connected layers with varying units and ReLU activation functions. The final layer employs a softmax activation function, indicative of the multi-class classification nature of our task. Notably, the choice of activation functions, kernel sizes, and dropout rates is designed to balance the model's complexity and generalization capabilities.

Subsequently, the model is compiled using categorical crossentropy as the loss function, the Adam optimizer for efficient weight updates, and accuracy as the metric for evaluation. The training process is executed on a GPU, enhancing computational efficiency.

It's essential to acknowledge that this approach provides flexibility for fine-tuning and adapting the model architecture based on performance evaluation. As we progress through subsequent milestones, we anticipate iteratively refining the model and enhancing its ability to accurately detect Alzheimer's disease stages in brain MRI images.

```
model=keras.Sequential([
    layers.Conv2D(filters=16,kernel_size=(4,4),activation='relu',use_bias=True,bias_initializer='random_normal'),
    layers.MaxPool2D(strides=(1,1)),
    layers.Conv2D(filters=32,kernel_size=(4,4),activation='relu',use_bias=True,bias_initializer='random_normal'),
    layers.MaxPool2D(strides=(1,1)),
    layers.Conv2D(filters=64,kernel_size=(4,4),activation='relu',use_bias=True,bias_initializer='random_normal'),
    layers.MaxPool2D(strides=(1,1)),
    layers.Dropout(rate=0.25),
    layers.Flatten(),

    layers.Dense(units=32,use_bias=True,bias_initializer='random_normal',activation='relu'),
    layers.Dropout(rate=0.25),
    layers.Dense(units=16,use_bias=True,bias_initializer='random_normal',activation='relu'),
    layers.Dense(units=8,use_bias=True,bias_initializer='random_normal',activation='relu'),
    layers.Dense(units=4,use_bias=True,bias_initializer='random_normal',activation='softmax')
])
```

```
with tf.device("/GPU:0"):
    model.compile(loss="categorical_crossentropy",optimizer="adam",metrics="accuracy")
```

```
history=0
with tf.device("/GPU:0"):
    history=model.fit(train,steps_per_epoch=161,epochs=5,validation_data=test,validation_steps=40)
```



## Activity 4: Train the model

Now, let us train our model with our image dataset. The model is trained for 30 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. fit functions used to train a deep learning neural network

Arguments:

- Epochs: an integer and number of epochs we want to train our model for.
- 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=0
with tf.device("/GPU:0"):
    history=model.fit(train,steps_per_epoch=161,epochs=5,validation_data=test,validation_steps=40)
```

```
Epoch 1/5
161/161 [=====] - 44s 275ms/step - loss: 0.1732 - accuracy: 0.9459 - val_loss: 4.3481 - val_accuracy: 0.6013
Epoch 2/5
161/161 [=====] - 44s 274ms/step - loss: 0.1687 - accuracy: 0.9490 - val_loss: 5.0139 - val_accuracy: 0.5801
Epoch 3/5
161/161 [=====] - 44s 274ms/step - loss: 0.1615 - accuracy: 0.9518 - val_loss: 4.5455 - val_accuracy: 0.5966
Epoch 4/5
161/161 [=====] - 44s 272ms/step - loss: 0.1627 - accuracy: 0.9520 - val_loss: 4.3783 - val_accuracy: 0.5872
Epoch 5/5
161/161 [=====] - 44s 273ms/step - loss: 0.1588 - accuracy: 0.9529 - val_loss: 4.1747 - val_accuracy: 0.5504
```

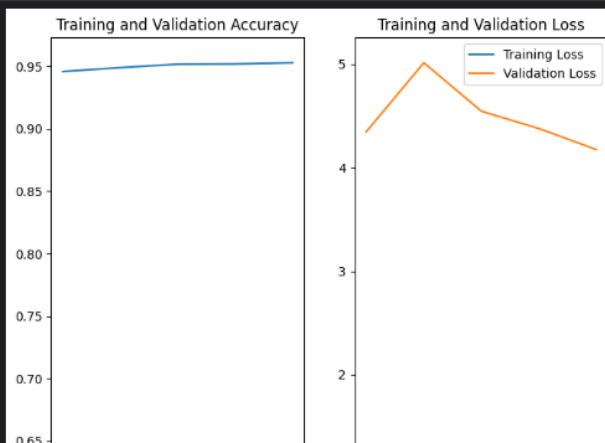
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(5)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



## Activity 5: Save the Model

Out of all the models we tried (CNN, VGG) gave us the best accuracy. So we are saving Xception as our final model

```
model.save("/content/drive/MyDrive/Colab Notebooks/models/alzheimer_detection.h5")
```

The model is saved with .h5 extension as follows

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

## Milestone 4: Application Building

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

This section has the following tasks

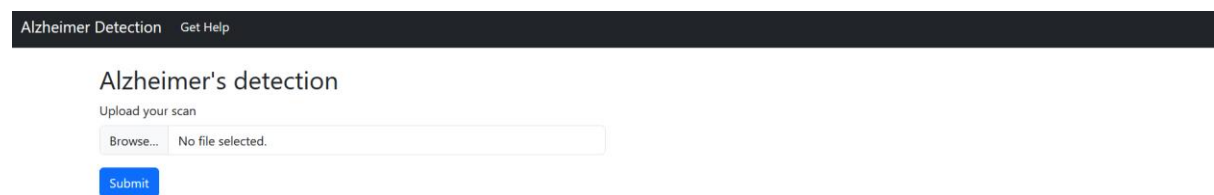
- model prediction using flask
- Building python code

Activity1: html templates:

For this project create one HTML file namely

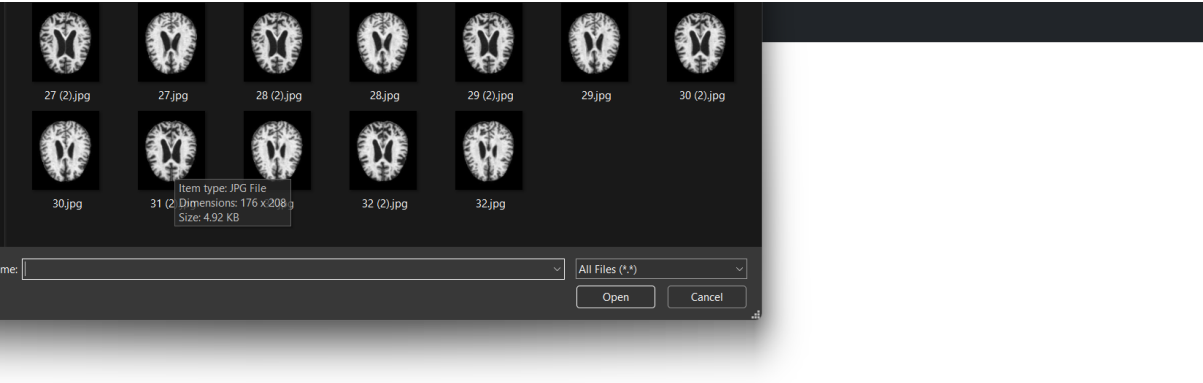
- index.ejs

Let's see how our index.ejs page looks like:



The screenshot shows a web application interface for Alzheimer's detection. At the top, there is a dark navigation bar with the text "Alzheimer Detection" and a link "Get Help". Below the navigation bar, the main heading is "Alzheimer's detection". Underneath the heading, there is a section titled "Upload your scan". This section contains a file upload interface with a "Browse..." button and a text field that says "No file selected.". Below the file upload section, there is a blue "Submit" button.

Output:



Alzheimer Detection    Get Help

### Alzheimer's detection

Upload your scan

Browse...    32 (2).jpg

Submit

### You have moderate dementia

Caring for someone with moderate dementia requires increased patience and adaptability. Establishing a clear routine remains important, but flexibility becomes crucial as cognitive challenges intensify. Providing more hands-on assistance with daily activities becomes necessary, and creating a safe environment becomes paramount to prevent accidents. Communication may become more challenging, requiring simplified language and increased support. Memory aids and engaging activities should be tailored to the individual's evolving abilities. Social interaction remains valuable, though the caregiver may need to take a more active role. Seeking regular guidance from healthcare professionals becomes even more critical, and joining support groups for caregivers can offer additional insights and emotional support. As the caregiving demands escalate, maintaining one's well-being through self-care becomes an even more vital aspect of the caregiving journey.

Alzheimer's Report

Category	Value
mild_demented	0.0
moderate_demented	1.0
non_demented	0.0
very_mild_demented	0.0

## nodeJS server:

```
const express = require("express");
const app = express();
const path = require("path");
const ejsMate = require("ejs-mate");
const methodOverride = require("method-override");
const multer = require("multer");

const storage = multer.memoryStorage();
const upload = multer({ storage: storage });

const modelPath = "file://models/alzheimer_detection/model.json";

app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(methodOverride("_method"));
app.use(express.static(path.join(__dirname, "public")));

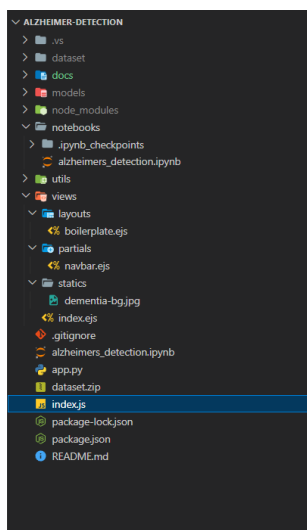
app.engine("ejs", ejsMate);

app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "views"));

app.listen(8080, () => {
  console.log("listening on 8080");
});

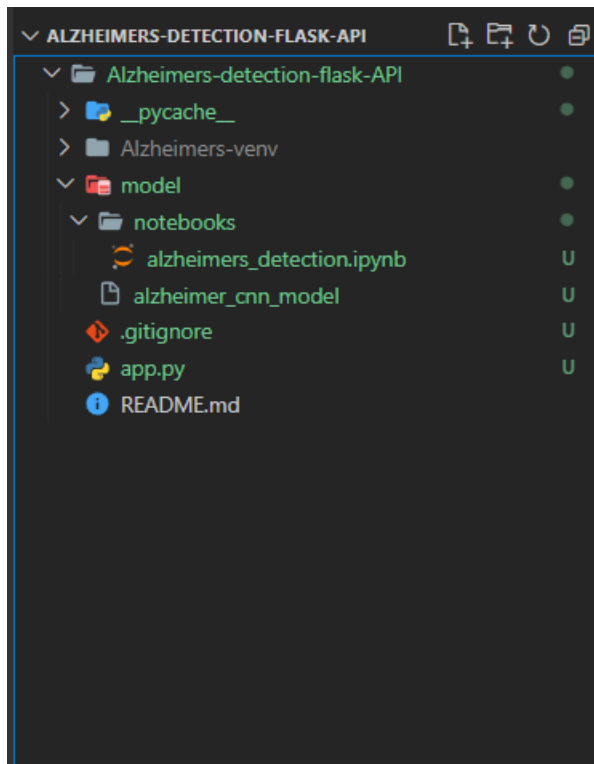
app.get("/", (req, res) => {
  res.render("index");
});
```

## Structure:



## Python flask API:

### Structure:



### Code:

```
from PIL import Image
import io
from flask import Flask, request, jsonify
from flask_cors import CORS
import numpy as np
import tensorflow as tf
MODEL_PATH="model/alzheimer_cnn_model"
app = Flask(__name__)
CORS(app)

# Load your TensorFlow model
# model_path = 'model/alzheimer_detection.h5'
# model = tf.keras.models.load_model(model_path)

# def preprocess_image(image, target_size=(224, 224)):
#     # Load the image, resize it to the target size, and convert it to an
#     array
#     img = tf.keras.preprocessing.image.load_img(image,
# target_size=target_size)
#     img_array = tf.keras.preprocessing.image.img_to_array(img)
#     # Expand the dimensions to match the model's expected input shape
#     img_array = np.expand_dims(img_array, axis=0)
```

```

#     # Preprocess the image (normalize values)
#     return tf.keras.applications.mobilenet.preprocess_input(img_array)

def image_to_buffer(image):
    buffer = io.BytesIO()
    image.save(buffer)
    return buffer.getvalue()

def resize_img(img_buffer, size=(255, 255)):
    try:
        # Ensure the image format is compatible with PIL
        pil_image =
Image.open(io.BytesIO(img_buffer)).resize(size).convert('RGB')
    except Exception as e:
        print(f"Error opening image: {e}")
        return None

    # Convert grayscale to RGB

    numpy_image = tf.keras.preprocessing.image.img_to_array(pil_image)
    tensor = tf.convert_to_tensor([numpy_image], dtype=tf.float32)

    return tensor

def image_to_tensor(image):
    image_buffer=image_to_buffer(image)
    tensor= resize_img(image_buffer,(176,176))
    return tensor

def predict(model_path,input_tensor):
    model = tf.keras.models.load_model(model_path, compile=False)
    predictions=model.predict(input_tensor)
    print(predictions)
    return predictions

@app.route("/",methods=['GET'])
def home():
    return jsonify({'msg':'You will do it'})
@app.route('/get-result', methods=['POST'])
def handle_api():
    image=request.files['image']

    tensor=image_to_tensor(image)
    print(tensor.shape)
    predictions=predict(MODEL_PATH,tensor)[0]
    predictions = [float(value) for value in predictions]
    print(predictions)

```

```

    predictions={
      "mild_demented":predictions[0],
      "moderate_demented":predictions[1],
      "non_demented":predictions[2],
      "very_mild_demented":predictions[3]
    }
    return jsonify({'class': predictions})

if __name__ == '__main__':
    app.run(debug=True, port=5000, host="localhost")

```

### index.ejs (HTML template):

```

<% layout('layouts/boilerplate') -%>

<div class="row">
  <div class="col-6">
    <h2 class="h2">Alzheimer's detection</h2>
    <form id="alzheimerForm" action="/blah">
      <div class="mb-3">
        <label for="imageUpload" class="form-label">Upload your
scan</label>
        <input type="file" class="form-control" id="imageUpload"
name="imageUpload"
placeholder="name@example.com">
      </div>
      <button class="btn btn-primary">Submit</button>
    </form>
    <div class=" mt-3" id="reportSummary">

    </div>
  </div>
  <div class="col-6">
    <canvas id="reportCard">

    </canvas>
  </div>
</div>
<script>
  const form = document.querySelector("#alzheimerForm")
  const imageScan = document.querySelector("#imageUpload")

  const uploadScan = async (file, url) => {
    if (!file) {
      //do something

    }
    return null
  }

```



```

    }
    const formData = new FormData()
    formData.append('image', file)

    const data = fetch(url, {
      method: 'POST',
      body: formData
    })
      .then(res => res.json())
      .then(res => {
        console.log(res)
        return res
      })
      .catch((err) => {
        console.log(`Error: ${err}`)
      })

    return data
  }

  const generateReportPieChart = (labels, data, idSelector, colours) =>
  {
    const reportCard = new Chart(idSelector, {
      type: "bar",
      data: {
        labels: labels,
        datasets: [{
          label: "Alzheimer's Report",
          data: data,
          backgroundColor: colours,
          hoverOffset: 4
        }]
      },
      options: {
        scales: {
          y: {
            beginAtZero: true,
            max: 100,
            ticks: {
              callback: function (value) {
                return value + "%";
              }
            }
          }
        }
      }
    });
    return reportCard;
  }

```

```

};

form.addEventListener('submit', async (evt) => {
  const url = "http://localhost:5000/get-result"
  evt.preventDefault()
  const file = imageScan.files[0]
  let result = await uploadScan(file, url).then(data => { return
data })

  result = result.class
  const keys = Object.keys(result)
  const values = Object.values(result)
  const colours = ["#ee9b00", "#d00000", "#99d98c", "#94d2bd"]
  const resultIndex = values.indexOf(Math.max(...values))
  const reportCard = generateReportPieChart(keys, values,
"reportCard", colours)

  if (result) {
    const text = document.querySelector("#reportSummary")
    const h2 = document.createElement("h2")
    const p = document.createElement("p")
    p.classList.add(["lead", "text-center"])
    h2.classList.add("h2")
    let msg = ""
    switch (keys[resultIndex]) {
      case keys[0]:
        //generate report for mild demented
        msg = "Caring for someone with mild dementia involves
creating a stable routine and using clear communication. Providing support
with daily tasks while ensuring a safe environment is essential. Encouraging
the use of memory aids and engaging in suitable activities helps maintain a
sense of normalcy. Social interaction is important, and patience is key when
adapting to changes in behavior. Seeking professional guidance and joining
support groups for caregivers can provide valuable insights. Lastly,
prioritizing self-care is crucial for effectively navigating the challenges of
caregiving."

        h2.innerText = "You have mild dementia"
        p.innerText = msg
        text.append(h2, p)
        break
      case keys[1]:
        //generate report for moderate demented
        msg = "Caring for someone with moderate dementia
requires increased patience and adaptability. Establishing a clear routine
remains important, but flexibility becomes crucial as cognitive challenges
intensify. Providing more hands-on assistance with daily activities becomes
necessary, and creating a safe environment becomes paramount to prevent
accidents. Communication may become more challenging, requiring simplified
language and increased support. Memory aids and engaging activities should be

```

tailored to the individual's evolving abilities. Social interaction remains valuable, though the caregiver may need to take a more active role. Seeking regular guidance from healthcare professionals becomes even more critical, and joining support groups for caregivers can offer additional insights and emotional support. As the caregiving demands escalate, maintaining one's well-being through self-care becomes an even more vital aspect of the caregiving journey."

```
        h2.innerText = "You have moderate dementia"
        p.innerText = msg
        text.append(h2, p)
        break
    case keys[2]:
        //generate report for non demented
        msg = "You do not have Dementia"
        h2.innerText = "You have moderate dementia"
        p.innerText = msg
        text.append(h2, p)
        break
    case keys[3]:
        //generate report for very mild demented
        msg = "Caring for someone with very mild dementia
involves subtle adjustments to support their cognitive well-being.
Establishing a consistent routine can help create a sense of stability. Clear
and simple communication remains important, along with providing gentle
assistance with daily tasks as needed. Creating a safe environment involves
minimal adjustments at this stage. Memory aids and engaging activities should
be tailored to maintain cognitive stimulation. Social interaction remains
beneficial, and patience is key as the person may occasionally struggle with
memory. Regular check-ins with healthcare professionals help monitor and
manage any changes. Caregivers should still prioritize self-care to navigate
the challenges of supporting someone with very mild dementia effectively."
        h2.innerText = "You have moderate dementia"
        p.innerText = msg
        text.append(h2, p)
        break
    }

    }
    console.log(finalResult)
})
</script>
```

**Links:**

- [Github Repo](#)
- [Youtube demo](#)