A PROJECT REPORT

ON

ML/DL techniques for Alzheimer's disease (AD) early diagnosis and mapping progression

Submitted In Partial Fulfillment of the Requirement for The Award of

Post Graduate Diploma in Artificial Intelligence (PG-DAI)

Under the Guidance of

MR. RAVI PAYAL
(Joint Director)
(Program Coordinator)

MR. DOMINIC IRUDHAYARAJ
(Project Guide)

Submitted By:

Deepam Kalekar, Kiran Malape,
Sudhansh Mehta (PG-DAI)

CONTENTS

CDAC, B-30, Institutional Area, Sector-62

Noida (Uttar Pradesh)-201307

CERTIFICATE

CDAC, NOIDA

This is to certify that Report entitled **"ML/DL techniques for Alzheimer's disease (AD) early diagnosis and mapping progression"** which is submitted by Deepam Kalekar, Kiran Malape & Sudhansh Mehta in partial fulfillment of the requirement for the award of **Post Graduate Diploma in Artificial Intelligence** (PG-DAI) to **CDAC, Noida** is a record of the candidates own work carried out by them under my supervision.

The documentation embodies results of original work, and studies are carried out by the student themselves and the contents of the report do not from the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

MR. DOMINIC IRUDHAYARAJ

(Project Guide)

ACKNOWLEDGEMENT

I would like to express my best sense of gratitude & endeavor with respect to Mr. Dominic Irudhayaraj for suggesting the problems scholarly guidance and expert supervision during the course of this project. Special thanks to Mr. Ravi Payal (Program Coordinator).

Deepam Kalekar, Kiran Malape &

Sudhansh Mehta (PG-DAI)

ABSTRACT

Millions of people across the world suffer from Alzheimer's disease (a neurodegenerative disorder of the brain) severely affecting their quality of life. The problem of waiting queues for getting medical imaging scans (structural brain MRI scans) done is only exacerbated by even lesser ratio of qualified technicians and Neuroradiologists who generate the technical report. Towards that end the current work tries to alleviate the problem of patients by developing a toolkit where in only the scans need be fed in to the models, which provide a classification of the different stages of cognitive health and the prognostic prediction/risk of patient declining into full-blown AD.

MOTIVATION

- Alzheimer's Disease (AD) is a neurodegenerative disorder of the brain and the leading cause of Dementia.
- Worldwide 55 million people are affected by AD in which India is contributing around 8.7%.
- In India there are a lot of people who are not taking any treatment for this by terming it as normal aging process.
- A lack of adequate diagnostic centers and specialized medical personnel like technicians and Neuroradiologists vis-à-vis a developing country with huge population like India.
- Need to automate the early diagnosis of AD:
    1. Start an early treatment plan, which can alleviate the symptoms, and the neurologist can personalize the treatment and care.
    2. Optimize the clinical trial and reduce time and cost.

Convolution Neural Networks(CNN's) are inspired by biological processes in living beings and the layout of the CNN's resembles the layout of visual cortex in animals. Convolution Neural Networks deliver state of the art results in Image Processing and Video Processing leading to diverse applications in fields such as medical image analysis(in Radiology), predicting the steering angle in self-driving cars (Autonomous Driving Vehicles) to name a few.

This work leverages the power of CNNs on structural MRI scan data to measure the atrophy of the brain caused due to Alzheimer's disease. The project is broadly divided into two stages:

- Classification: A classification of patients into following classes:
    1. Cognitively normal (CN)
    2. Early Mild Cognitive Impairment (EMCI)
    3. Mild Cognitive Impairment (MCI)
    4. AD (a full-blown Alzheimer's disease)
- Prognostic Prediction: probability of conversion of a patient from mild cognitive impairment (MCI) to a full-blown AD.

DATASET DESCRIPTION

All models are trained on structural MRI scans from the ADNI (Alzheimer's Disease Neuroimaging Initiative) database, which contains neuroimages assigned one of three labels: control (CN), representing no disorder; Early Mild Cognitive Impairment (EMCI); mild cognitive impairment (MCI); and Alzheimer's disease (AD).

CLASSIFICATION – PATIENT'S STAGE OF COGNITIVE HEALTH

- CNN's being state of the art in image classification, different CNN architectures from basic to advanced transfer learning architectures were deployed for classification of different stages of cognitive health (CN,EMCI, MCI & AD).

BASIC CNN MODEL ARCHITECTURE

```
model = keras.Sequential([

    keras.Input(shape=train_data.shape[1:]),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation="relu"),

    layers.Dropout(0.5),
    layers.Dense(4, activation="softmax")
```

TRANSFER LEARNING APPROACH

- To use the widely popular CNN architectures the concept of Transfer learning comes into play. The idea is to freeze the early convolutional layers of the network and only train the last few layers(fine tuning to the current application) which make a prediction.
- Various transfer learning architectures were applied to the problem of Cognitive Health State classification out of which densenet-169 gave best accuracy.

DENSENET-169 MODEL ARCHITECTURE

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet169 (Functional)     (None, 7, 7, 1664)        12642880
_________________________________________________________________
dropout (Dropout)            (None, 7, 7, 1664)        0
_________________________________________________________________
flatten (Flatten)            (None, 81536)             0
_________________________________________________________________
batch_normalization (BatchNo (None, 81536)             326144
_________________________________________________________________
dense (Dense)                (None, 2048)              166987776
_________________________________________________________________
batch_normalization_1 (Batch (None, 2048)              8192
_________________________________________________________________
activation (Activation)      (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
activation_1 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 4100
=================================================================
Total params: 182,071,364
Trainable params: 169,259,268
Non-trainable params: 12,812,096
_________________________________________________________________
```

SOURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import skimage.io
import os
import tqdm
import glob
import tensorflow
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.color import grey2rgb
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout, Flatten, Dense, Activation, MaxPool2D, Conv2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications.densenet import DenseNet169
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
"""### IMPORT / VIEWING / PREPROCESSING DATASET > `DATA AUGMENTATION`"""
train_datagen = ImageDataGenerator(rescale = 1./255, rotation_range=30,
zoom_range=0.2, horizontal_flip=True, vertical_flip=True, validation_split = 0.2)
valid_datagen = ImageDataGenerator(rescale = 1./255, validation_split = 0.2)
test_datagen  = ImageDataGenerator(rescale = 1./255)

train_dataset    =  train_datagen.flow_from_directory(directory  =  '../input/alzheimers-
dataset-4-class-of-images/Alzheimer_s Dataset/train',target_size = (224,224),
class_mode = 'categorical', subset = 'training', batch_size = 128)

valid_dataset  =  valid_datagen.flow_from_directory(directory  =  '../input/alzheimers-
dataset-4-class-of-images/Alzheimer_s Dataset/train', target_size = (224,224),
class_mode = 'categorical',subset = 'validation',batch_size = 128)



fig, ax = plt.subplots(nrows = 1, ncols = 5, figsize=(20,20))
for i in tqdm(range(0,5)):
    rand1 = np.random.randint(len(train_dataset))
    rand2 = np.random.randint(100)
    ax[i].imshow(train_dataset[rand1][0][rand2])
    ax[i].axis('off')
    a = train_dataset[rand1][1][rand2]
    if a[0] == 1:
        ax[i].set_title('Mild Dementia')
    elif a[1] == 1:
        ax[i].set_title('Moderate Dementia')
    elif a[2] == 1:
        ax[i].set_title('Non Demetia')
    elif a[3] == 1:
        ax[i].set_title('Very Mild Dementia')
```

```
"""### MODEL BUILDING"""
# Model Initialization
base_model = DenseNet169(input_shape=(224,224,3), include_top=False,
weights="imagenet")

# Freezing Layers
for layer in base_model.layers:
    layer.trainable=False

# Building Model
model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(2048,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1024,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(4,activation='softmax'))

# Model Compile
OPT    = tensorflow.keras.optimizers.Adam(lr=0.001)
model.compile(loss='categorical_crossentropy',metrics=[tensorflow.keras.metrics.AUC(n
ame = 'auc')],optimizer=OPT)
```

```
# Defining Callbacks
filepath = './best_weights.hdf5'
earlystopping = EarlyStopping(monitor = 'val_auc', mode = 'max' , patience = 15,
verbose = 1)

checkpoint    = ModelCheckpoint(filepath,monitor = 'val_auc',
mode='max', save_best_only=True, verbose = 1)

callback_list = [earlystopping, checkpoint]

model_history=model.fit(train_dataset,
                        validation_data=valid_dataset,
                        epochs = 500,
                        callbacks = callback_list,
                        verbose = 1)

"""### MODEL EVALUATION"""
# Summarize history for loss
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()
```

```
# Summarize history for loss
plt.plot(model_history.history['auc'])
plt.plot(model_history.history['val_auc'])
plt.title('Model AUC')
plt.ylabel('AUC')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()

# Test Data
test_dataset = test_datagen.flow_from_directory(directory = '../input/alzheimers-dataset-4-class-of-images/Alzheimer_s Dataset/test',
 target_size = (224,224),class_mode = 'categorical',batch_size = 128)

# Evaluating Loss and AUC
model.evaluate(test_dataset)

# Test Case 1: Non-Dementia
dic = test_dataset.class_indices
idc = {k:v for v, k in dic.items()}
img = load_img('../input/alzheimers-dataset-4-class-of-images/Alzheimer_s Dataset/test/NonDemented/26 (100).jpg', target_size = (224,224,3))
img = img_to_array(img)
img = img/255
imshow(img)
plt.axis('off')
img = np.expand_dims(img,axis=0)
answer = model.predict_classes(img)
probability = round(np.max(model.predict_proba(img)*100),2)
print(probability, '% chances are there that the image is',idc[answer[0]])
```

PROGNOSTIC PREDICTION

- Deep convolutional neural networks augmented with a recurrent LSTM mechanism offer a powerful solution for predicting prognoses of Alzheimer’s disease(AD) (probability of conversion of Mild Cognitive impairment to full-blown AD) in patients based on structural MRI scans.
- Project develops and trains a deep convolutional LSTM neural network on structural MRI neuroimage data of Alzheimer’s patients to yield predictive prognoses of future disease progression for individual patients based on previous MRI sequencing. The model takes in a sequence of MRI neuroimages and yields the likelihood of conversion from mild cognitive impairment (MCI) to Alzheimer's disease (AD) as a prognostic prediction.

MODEL ARCHITECTURE

The prediction network combines a convolutional neural network (CNN), which compresses the neuroimages by extracting learned features; and a long short-term memory (LSTM) cell, which combines the extracted features with those of previously inputted MRI scans for each patient. The output of the LSTM is fed through a single fully connected layer, to translate the multidimensional LSTM output into a single probability between 0 and 1. This network will be trained to produce the probability that a patient will develop Alzheimer's within the next five years, weighed against a loss function that aggregates diagnoses for individual patients over time.

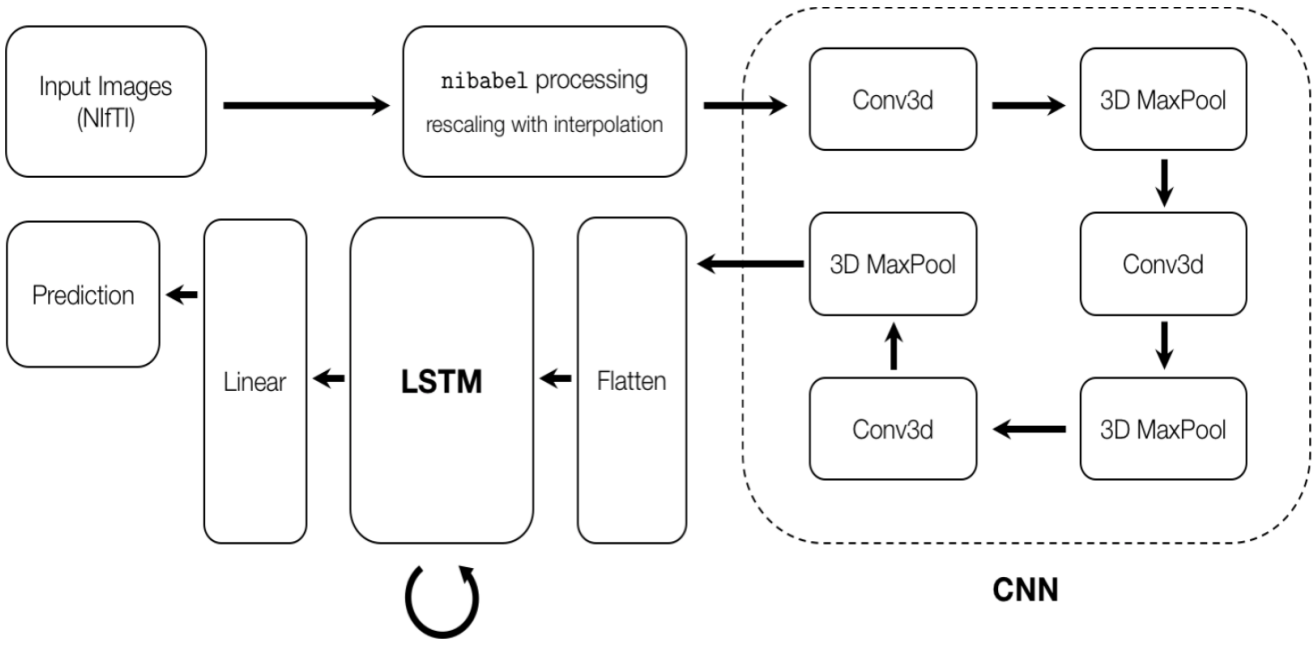


Fig. The above figure denotes the architecture. CNNs are used for feature extraction from the images and converting into a 1-D tensor, which is fed into the LSTM. LSTM aggregates features from different time stamps of the patient diagnostic history to give a prognostic prediction of Alzheimer's progression (probability of conversion from MCI to AD).

SOURCE CODE

```
from google.colab import drive
drive.mount('/content/drive')
import time
import math
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data.sampler import SubsetRandomSampler

# To unpack ADNI data
import pickle
import random
!cp /content/drive/MyDrive/Alzheimer_project/network.py /content
!cp /content/drive/MyDrive/Alzheimer_project/data_loader.py /content

# Import network
import sys
from network import Network
from data_loader import MRIData
import argparse
```

```
"""parser = argparse.ArgumentParser(description='Train and validate network.')
parser.add_argument('--disable-cuda', action='store_true', default=False,
                    help='Disable CUDA')
args = parser.parse_args()
args.device = None
print(args.disable_cuda)
if torch.cuda.is_available():
    print("Using CUDA. : )")
    # torch.set_default_tensor_type('torch.cuda.FloatTensor')
    args.device = torch.device('cuda')
else:
    print("We aren't using CUDA.")
    args.device = torch.device('cpu')
#For reproducibility for testing purposes. Delete during actual training.
torch.manual_seed(1)
random.seed(1)
# Hyperparameters
BATCH_SIZE = 5
# Dimensionality of the data outputted by the LSTM,
# forwarded to the final dense layer.
LSTM_output_size = 16
input_size = 1 # Size of the processed MRI scans fed into the CNN.
output_dimension = 2 # the number of predictions the model will make
```

```
# 2 used for binary prediction for each image.

# update the splicing used in train()

learning_rate = 0.1

training_epochs = 1

# The size of images passed, as a tuple

data_shape = (200,200,150)

# Other hyperparameters unlisted: the depth of the model, the kernel size, the padding, the channel restriction.

!cp /content/drive/MyDrive/Data/Combined_MRI_List.pkl /content

## Import Data

MRI_images_list = pickle.load(open("Combined_MRI_List.pkl", "rb"))

random.shuffle(MRI_images_list)

train_size = int(0.7 * len(MRI_images_list))

#EDITING FOR CHECKING TEST FUNCTION

# Split list

training_list = MRI_images_list[:train_size]

test_list =  MRI_images_list[train_size:]

DATA_ROOT_DIR = '/content/drive/MyDrive'

train_dataset = MRIData(DATA_ROOT_DIR, training_list)

test_dataset = MRIData(DATA_ROOT_DIR, test_list)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)

test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True)

training_data = train_loader

test_data = test_loader
```

```
## Define Model

model=Network(input_size,data_shape,output_dimension).to(torch.device('cuda'))#torch.device('cuda')

loss_function = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=learning_rate)


## Training Function

def train(model,training_data,optimizer,criterion):
    """ takes (model, training data, optimizer, loss function)"""
    torch.set_grad_enabled(True)
    # Activate training mode
    model.train()
    # Initialize the per epoch loss
    epoch_loss = 0
    epoch_length = len(training_data)
    for i, patient_data in enumerate(training_data):
        #if i % (math.floor(epoch_length / 5) + 1) == 0: print(f"\t\tTraining Progress:{i / len(training_data) * 100}%")
        # Clear gradients
        model.zero_grad()
        torch.cuda.empty_cache() # Clear CUDA memory
        batch_loss=torch.tensor(0.0, requires_grad=True).to(torch.device('cuda'))
        # Clear the LSTM hidden state after each patient
        model.hidden = model.init_hidden()
        # Get the MRI's and classifications for the current patient
        patient_markers = patient_data['num_images']
```

```
        patient_MRIs = patient_data["images"].to(torch.device('cuda'))
        patient_classifications = patient_data["label"]
        print("Patient batch classes ", patient_classifications)
        for x in range(len(patient_MRIs)):
            try:
                # Clear hidden states to give each patient a clean slate
                model.hidden = model.init_hidden()
                single_patient_MRIs = patient_MRIs[x][:patient_markers[x]].view(-1,1,data_shape[0],data_shape[1],data_shape[2])
                patient_diagnosis = patient_classifications[x]
                patient_endstate = torch.ones(single_patient_MRIs.size(0)) * patient_diagnosis
                patient_endstate = patient_endstate.long().to(torch.device('cuda'))
                out = model(single_patient_MRIs)
                if len(out.shape)==1:
                    out = out[None,...] # In the case of a single input, we need padding
                print("model predictions are ",out)
                print("patient endstate is ",patient_endstate)
                model_predictions = out
                loss = criterion(model_predictions, patient_endstate)
                batch_loss += loss
            except Exception as e:
                print("EXCEPTION CAUGHT:",e)
        batch_loss.retain_grad()
        batch_loss.backward()
        print("batch loss is",batch_loss)
```

```
        optimizer.step()
        epoch_loss += batch_loss
    if epoch_length == 0: epoch_length = 0.000001
    return epoch_loss / epoch_length


## Testing Function
def test(model, test_data, criterion):
    """takes (model, test_data, loss function) and returns the epoch loss."""
    torch.set_grad_enabled(False)
    model.eval()
    # Initialize the per epoch loss
    epoch_loss = 0
    epoch_length = len(test_data)
    for i, patient_data in enumerate(test_data):
        #if i % (math.floor(epoch_length / 5) + 1) == 0: print(f"\t\tTraining Progress:{i / len(training_data) * 100}%")
        # Clear gradients
        model.zero_grad()
        torch.cuda.empty_cache() # Clear CUDA memory
        batch_loss=torch.tensor(0.0).to(torch.device('cuda'))
        # Clear the LSTM hidden state after each patient
        model.hidden = model.init_hidden()
        # Get the MRI's and classifications for the current patient
        patient_markers = patient_data['num_images']
        patient_MRIs = patient_data["images"].to(torch.device('cuda'))
        patient_classifications = patient_data["label"]
```

```
        print("Patient batch classes ", patient_classifications)
        for x in range(len(patient_MRIs)):
            try:
                # Clear hidden states to give each patient a clean slate
                model.hidden = model.init_hidden()
                single_patient_MRIs = patient_MRIs[x][:patient_markers[x]].view(-
1,1,data_shape[0],data_shape[1],data_shape[2])
                patient_diagnosis = patient_classifications[x]
                patient_endstate=torch.ones(single_patient_MRIs.size(0))                *
patient_diagnosis
                patient_endstate = patient_endstate.long().to(torch.device('cuda'))
                out = model(single_patient_MRIs)
                if len(out.shape)==1:
                    out = out[None,...] # In the case of a single input, we need padding
                print("model predictions are ",out)
                print("patient endstate is ",patient_endstate)
                model_predictions = out
                loss = criterion(model_predictions, patient_endstate)
                batch_loss += loss
                print("Current test loss ",loss)
            except Exception as e:
                print("EXCEPTION CAUGHT:",e)
        epoch_loss += batch_loss
        print("batch loss is",batch_loss)
        print("total loss is",epoch_loss)
```

```
    if epoch_length == 0: epoch_length = 0.000001
    return epoch_loss / epoch_length
torch.set_grad_enabled(True)
# Perform training and measure test accuracy. Save best performing model.
best_test_accuracy = float('inf')
for epoch in range(training_epochs):
    start_time = time.time()
    train_loss = train(model, training_data, optimizer, loss_function)
    test_loss = test(model, test_data, loss_function)
    end_time = time.time()
    epoch_mins = math.floor((end_time-start_time)/60)
    epoch_secs = math.floor((end_time-start_time)%60)
    print(f"Hurrah! Epoch {epoch + 1}/{training_epochs} concludes. | Time: {epoch_mins}m {epoch_secs}s")
    print(f"\tTrain Loss: {train_loss:.3f}| Train Perplexity: {math.exp(train_loss):7.3f}")
    print(f"\tTest Loss: {test_loss:.3f}| Test Perplexity: {math.exp(test_loss):7.3f}")
    if test_loss<best_test_accuracy:
        print("...that was our best test accuracy yet!")
        best_test_accuracy=test_loss
        torch.save(model.state_dict(),'ad-model.pt')
```

RESULTS - CLASSIFICATION

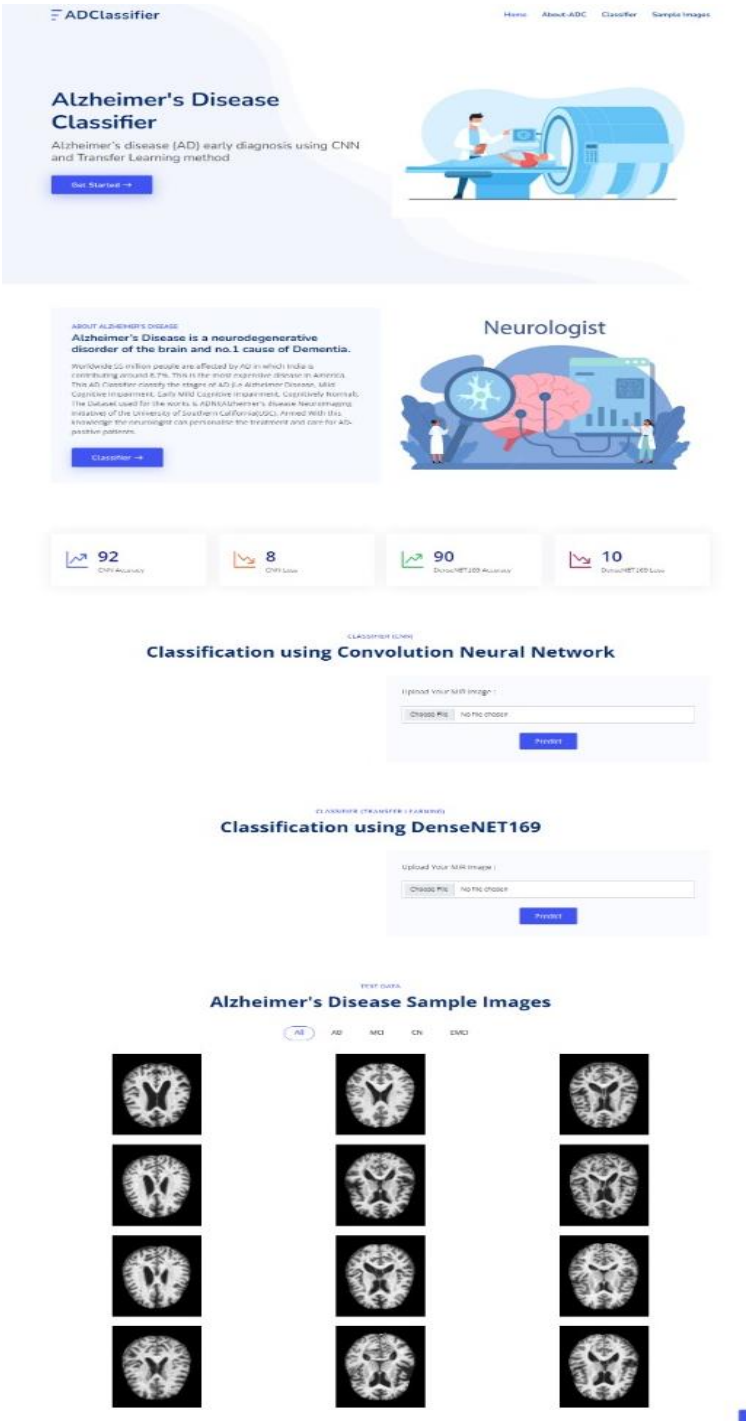
Architecture	Accuracy
Basic CNN	94%
Transfer Learning CNN – Densenet-169	90%

RESULTS – PROGNOSTIC PREDICTION – MCI to AD

```
Hurrah! Epoch 1/1 concludes. | Time: 68m 4s
        Train Loss: 22.309| Train Perplexity: 4881831980.536
        Test Loss: 5.483| Test Perplexity: 240.491
...that was our best test accuracy yet!
```

DEPLOYMENT

- An end product (web app) for the prospective clients is developed using Flask Web application development. The client can upload his structural MRI scans to get all the results.
- Prospective Clients:
    1. Neuroradiologists
    2. Neurosurgeons
    3. Patient

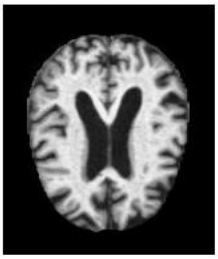


SAMPLE OUPUT ON FLASK

CLASSIFIER (TRANSFER LEARNING)

Classification using DenseNET169

Alzheimer Disease



Upload Your MIR Image :

Choose File | No file chosen

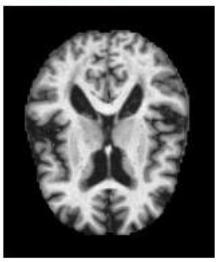
Predict

Predicted AD Class for MRI is: Alzheimer Disease

CLASSIFIER (CNN)

Classification using Convolution Neural Network

Cognitively Normal



Upload Your MIR Image :

Choose File | No file chosen

Predict

Predicted AD Class for MRI is: Cognitively Normal

CONCLUSION & FUTURE SCOPE

- Owing to the dearth of specialized medical personnel & diagnostic centers in a developing country like India many a times marginalized communities are unable to get the proper medical attention required in as serious disease like Alzheimer's. Our current work provides a one stop catering to this segment of population where the patients can just upload their scans on the web app and get fast results & analysis of their cognitive health status. Apart from this the work also eases the workload of the medical personnel by providing them with a base status of the patients.
- The current work only focuses on structural MRI scans for assessing the cognitive health of the patients. Further study can include use of PET imaging biomarkers (Aβ-PET and tau PET).

REFERENCES

- Osborn's Brain: Imaging, Pathology, and Anatomy - Anne G. Osborn, Garry L. Hedlund, and Karen L Salzman, MD.
- Detecting the Stages of Alzheimer’s Disease with Pre-trained Deep Learning Architectures - Serkan Sava.
- Dynamic Prediction of Alzheimer’s Disease Progression Using Features of Multiple Longitudinal Outcomes and Time-to-Event Data - Sheng Luo.
- Deep Learning in Alzheimer's Disease: Diagnostic Classification and Prognostic Prediction Using Neuroimaging Data - Taeho Jo, Kwangsik Nho, Andrew J. Saykin.