Project 1 Report

Name: Chidozie Jeffrey Arukwe
MavID: 1001894300

The project attempts to build a Convolutional Neural Network (CNN) which performs classification task on a dataset containing 10 MRI (Magnetic Resonance Imaging) scans of Alzheimer's disease patients and 10 MRI scans of normal patients.

This work uses a 3D CNN approach. This is motivated by the fact that dealing with the individual slices independently in 2D CNNs deliberately discards the depth information which results in poor performance for the intended task [1].

This report will show the steps needed to build a 3D convolutional neural network (CNN) to predict the presence of Alzheimer's disease (AD) in MRI scans. 2D CNNs are commonly used to process RGB images (3 channels). A 3D CNN is simply the 3D equivalent: it takes as input a 3D volume or a sequence of 2D frames (e.g., slices in an MRI scan).

The dataset

It contains 10 T1 MRI scans of Alzheimer's disease patients and 10 T1 MRI scans of normal patients.

The dataset can be found:

```
https://github.com/Jeffreyarukwe/CN/raw/main/CN.zip
https://github.com/Jeffreyarukwe/AD/raw/main/AD.zip
```

Objectives

There are some goals set forth with exploring this dataset, they include:
1. Train a CNN model
2. Using the trained model to do the classification, i.e. given new data points, how accurately can the model predict what class it belongs
3. Report some performance metrics

Setup

Tensorflow Keras library was used for this project.

```
import os
import zipfile
import numpy as np
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
```

Loading data and preprocessing

The files are provided in **Nifti** format with the extension .nii. To read the scans, we use the **Nibabel** package. A threshold between 0.0 and 1055409.2 was used to normalize MRI scans.

To process the data, we do the following:

We first rotate the volumes by 90 degrees, so the orientation is fixed

We scale the values to be between 0 and 1.

We resize width, height and depth.

Here we define several helper functions to process the data. These functions will be used when building training and validation datasets.

Then we read the paths of the MRI scans from the class directories.

```python
# Folder "CN" consist of MRI scans of normal patients
normal_scan_paths = [
    os.path.join(os.getcwd(), "MedData/CN", x)
    for x in os.listdir("MedData/CN")
]
# Folder "AD" consist of MRI scans having Alzheimer's disease
abnormal_scan_paths = [
    os.path.join(os.getcwd(), "MedData/AD", x)
    for x in os.listdir("MedData/AD")
]

print("MRI scans with normal patients: " + str(len(normal_scan_paths)))
print("MRI scans with abnormal patients: " + str(len(abnormal_scan_paths)))
```

```
MRI scans with normal patients: 10
MRI scans with abnormal patients: 10
```

Build train and validation datasets

Read the scans from the class directories and assign labels. Downsample the scans to have shape of 128x128x64. Rescale the raw voxel values to the range 0 to 1. Lastly, split the dataset into train and validation (test) subsets.

```
# Read and process the scans.
# Each scan is resized across height, width, and depth and rescaled.
abnormal_scans = np.array([process_scan(path) for path in abnormal_scan_paths])
normal_scans = np.array([process_scan(path) for path in normal_scan_paths])

# For the MRI scans of AD patients
# assign 1, for the normal ones assign 0.
abnormal_labels = np.array([1 for _ in range(len(abnormal_scans))])
normal_labels = np.array([0 for _ in range(len(normal_scans))])

# Split data in the ratio 70-30 for training and validation.
x_train = np.concatenate((abnormal_scans[:7], normal_scans[:7]), axis=0)
y_train = np.concatenate((abnormal_labels[:7], normal_labels[:7]), axis=0)
x_val = np.concatenate((abnormal_scans[7:], normal_scans[7:]), axis=0)
y_val = np.concatenate((abnormal_labels[7:], normal_labels[7:]), axis=0)
print(
    "Number of samples in train and validation are %d and %d."
    % (x_train.shape[0], x_val.shape[0])
)
```

```
Number of samples in train and validation are 14 and 6.
```

Data augmentation

Data augmentation is a strategy that enables us to significantly increase the diversity of data available for training models, without collecting new data. Data augmentation techniques such as cropping, padding, flipping, rotation, translation, brightness, contrast, color Augmentation, and saturation are commonly used to train large neural networks.

The MRI scans also augmented by rotating at random angles during training. Since the data is stored in rank-3 tensors of shape (samples, height, width, depth), we add a dimension of size 1 at axis 4 to be able to perform 3D convolutions on the data. The new shape is thus (samples, height, width, depth, 1).

```python
import random

from scipy import ndimage


@tf.function
def rotate(volume):
    """Rotate the volume by a few degrees"""

    def scipy_rotate(volume):
        # define some rotation angles
        angles = [-20, -10, -5, 5, 10, 20]
        # pick angles at random
        angle = random.choice(angles)
        # rotate volume
        volume = ndimage.rotate(volume, angle, reshape=False)
        volume[volume < 0] = 0
        volume[volume > 1] = 1
        return volume

    augmented_volume = tf.numpy_function(scipy_rotate, [volume], tf.float32)
    return augmented_volume
```

While defining the train and validation data loader, the training data is passed through and augmentation function which randomly rotates volume at different angles. Note that both training and validation data are already rescaled to have values between 0 and 1.

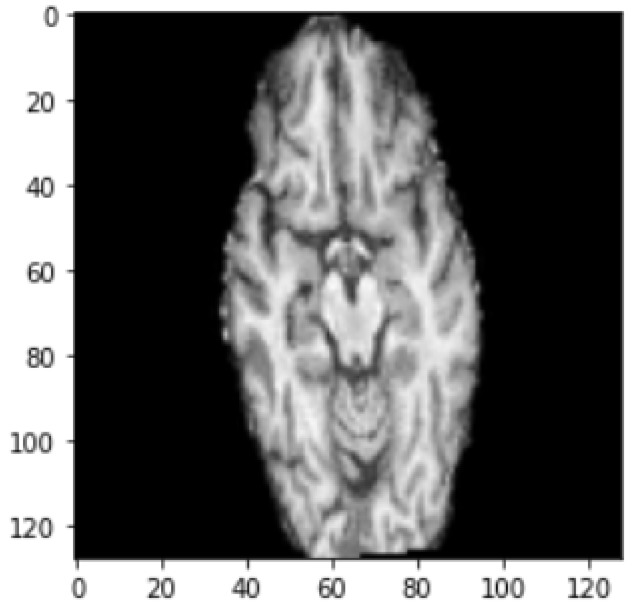
```
# Define data loaders.
train_loader = tf.data.Dataset.from_tensor_slices((x_train, y_train))
validation_loader = tf.data.Dataset.from_tensor_slices((x_val, y_val))

batch_size = 2
# Augment the on the fly during training.
train_dataset = (
    train_loader.shuffle(len(x_train))
    .map(train_preprocessing)
    .batch(batch_size)
    .prefetch(2)
)
# Only rescale.
validation_dataset = (
    validation_loader.shuffle(len(x_val))
    .map(validation_preprocessing)
    .batch(batch_size)
    .prefetch(2)
)
```

Visualization

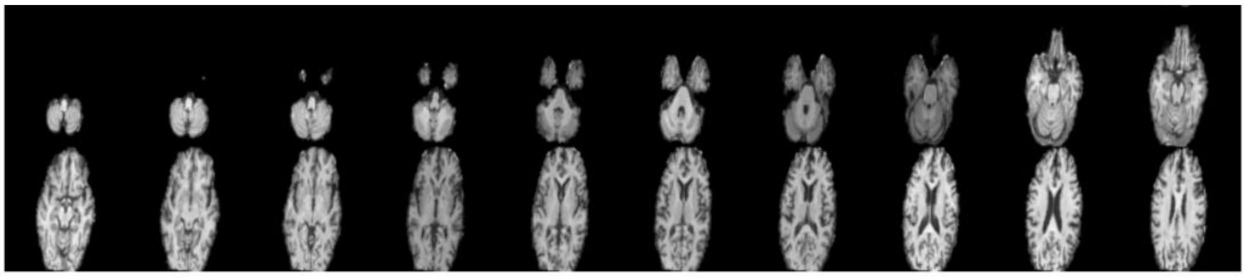
We can now visualize an augmented MRI scan.

```python
import matplotlib.pyplot as plt

data = train_dataset.take(1)
images, labels = list(data)[0]
images = images.numpy()
image = images[0]
print("Dimension of the MRI scan is:", image.shape)
plt.imshow(np.squeeze(image[:, :, 30]), cmap="gray")
```

```
Dimension of the MRI scan is: (128, 128, 64, 1)
<matplotlib.image.AxesImage at 0x7f7753731bd0>
```

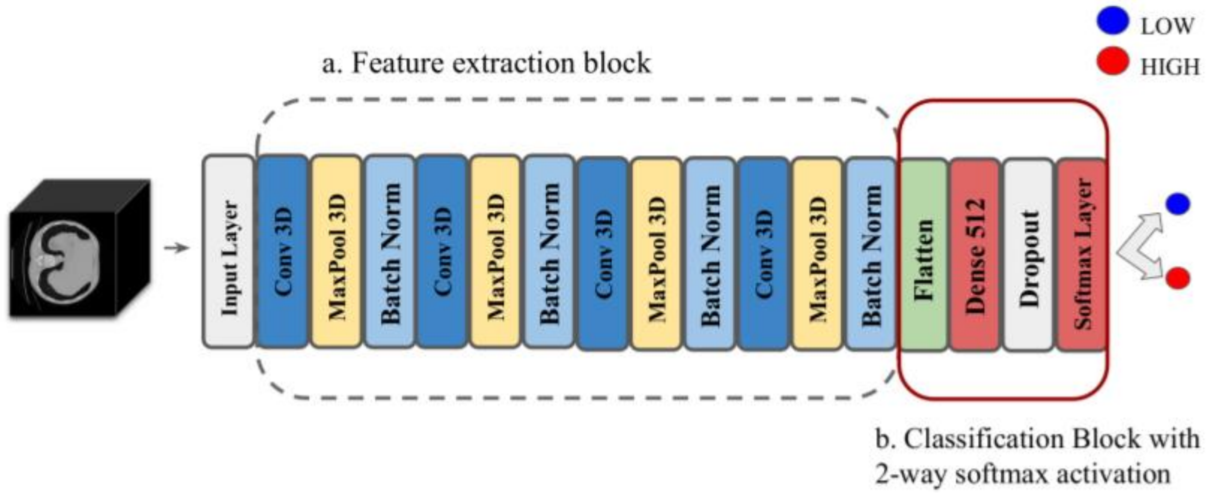


Or Multiple of them:



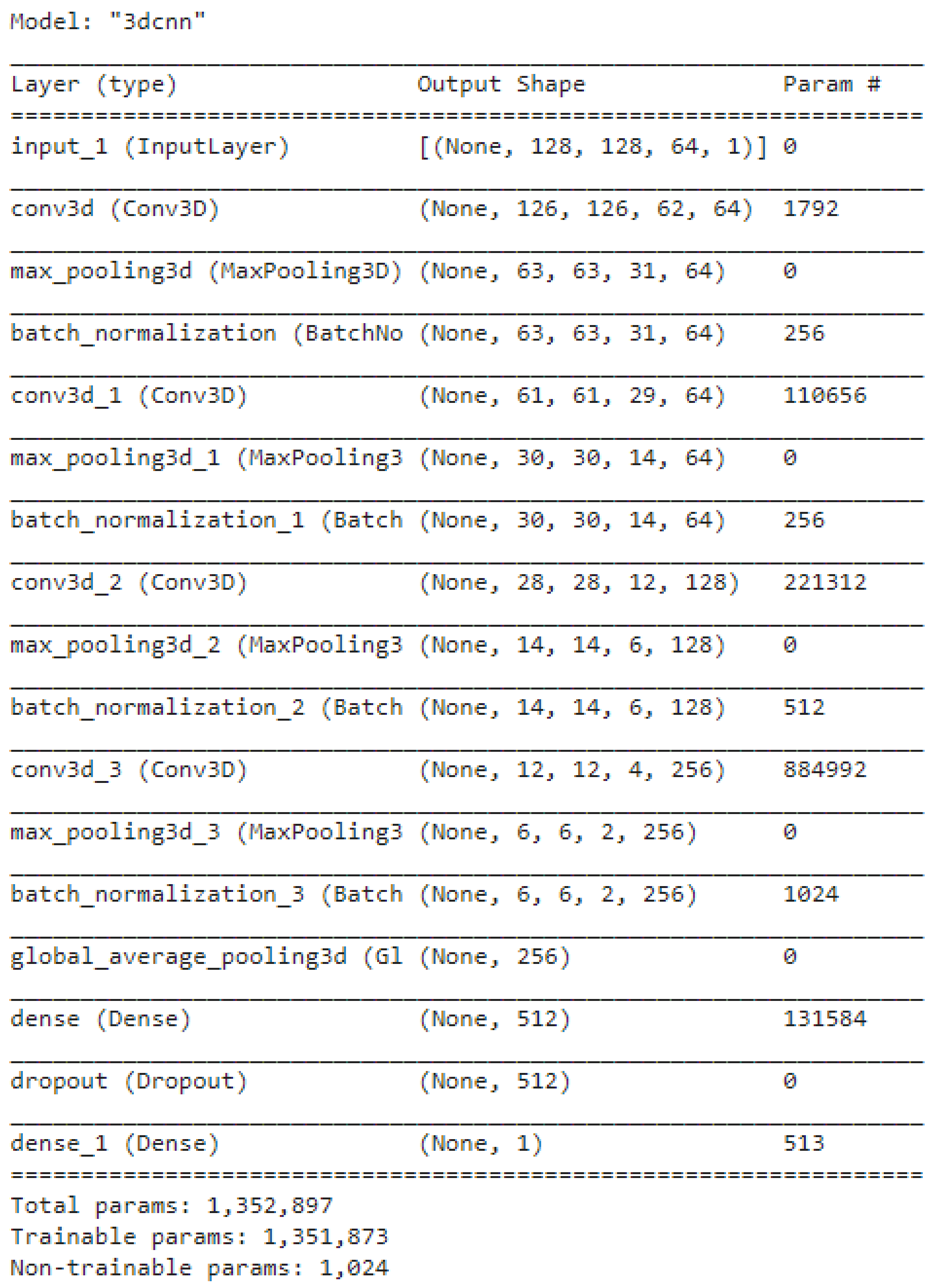
Define a 3D convolutional neural network

To make the model easier to understand, we structure it into blocks. The architecture of the 3D CNN used in this example is based on this paper:



The design is that of a 17-layer 3D CNN which comprises four 3D convolutional (CONV) layers with two layers consisting of 64 filters followed by 128 and 256 filters all with a kernel size of 3×3×3. Each CONV layer is followed by a max-pooling (MAXPOOL) layer with a stride of 2 and ReLU activation which ends with batch normalization (BN) layer. Essentially, the feature extraction block consists of four CONV-MAXPOOL-BN modules. The final output from the feature extraction block is flattened and passed to a fully connected layer with 512 neurons with a dropout set to 30% and fed to a softmax layer for a binary classification.

Below is a summary of the model.



```
Model: "3dcnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 64, 1)] 0         
_________________________________________________________________
conv3d (Conv3D)              (None, 126, 126, 62, 64)  1792      
_________________________________________________________________
max_pooling3d (MaxPooling3D) (None, 63, 63, 31, 64)    0         
_________________________________________________________________
batch_normalization (BatchNo (None, 63, 63, 31, 64)    256       
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 61, 61, 29, 64)    110656    
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 30, 30, 14, 64)    0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 30, 30, 14, 64)    256       
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 28, 28, 12, 128)   221312    
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 14, 14, 6, 128)    0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 6, 128)    512       
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 12, 12, 4, 256)    884992    
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 6, 6, 2, 256)      0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 6, 6, 2, 256)      1024      
_________________________________________________________________
global_average_pooling3d (Gl (None, 256)               0         
_________________________________________________________________
dense (Dense)                (None, 512)               131584    
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513       
=================================================================
Total params: 1,352,897
Trainable params: 1,351,873
Non-trainable params: 1,024
```

Train the model

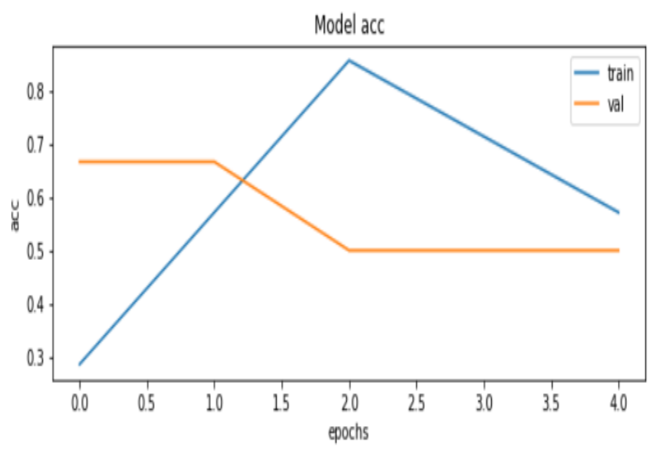
Then we compile and train the model for 25 Epochs.

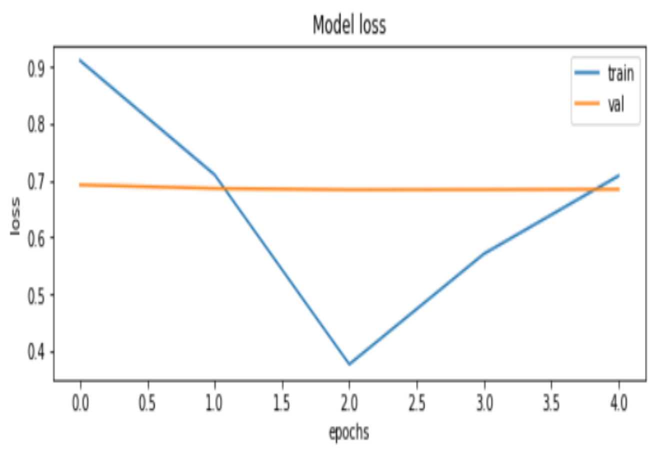
I experimented with the different hyperparameters (e.g., The learning rate of 0.0001 was chosen. Batch size = 2)

```
# Compile model.
initial_learning_rate = 0.0001
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=100000, decay_rate=0.96, staircase=True
)
model.compile(
    loss="binary_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
    metrics=["acc"],
)

# Define callbacks.
checkpoint_cb = keras.callbacks.ModelCheckpoint(
    "3d_image_classification.h5", save_best_only=True
)
early_stopping_cb = keras.callbacks.EarlyStopping(monitor="val_acc", patience=15)

# Train the model, doing validation at the end of each epoch
epochs = 25
model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    shuffle=True,
    verbose=2,
    callbacks=[checkpoint_cb, early_stopping_cb],
)
```

Visualizing Model Performance

Here the model accuracy and loss for the training and the validation sets are plotted. Since the validation set is class-balanced, accuracy provides an unbiased representation of the model's performance. The graphs below show the accuracy and model loss for the both training and test sets.



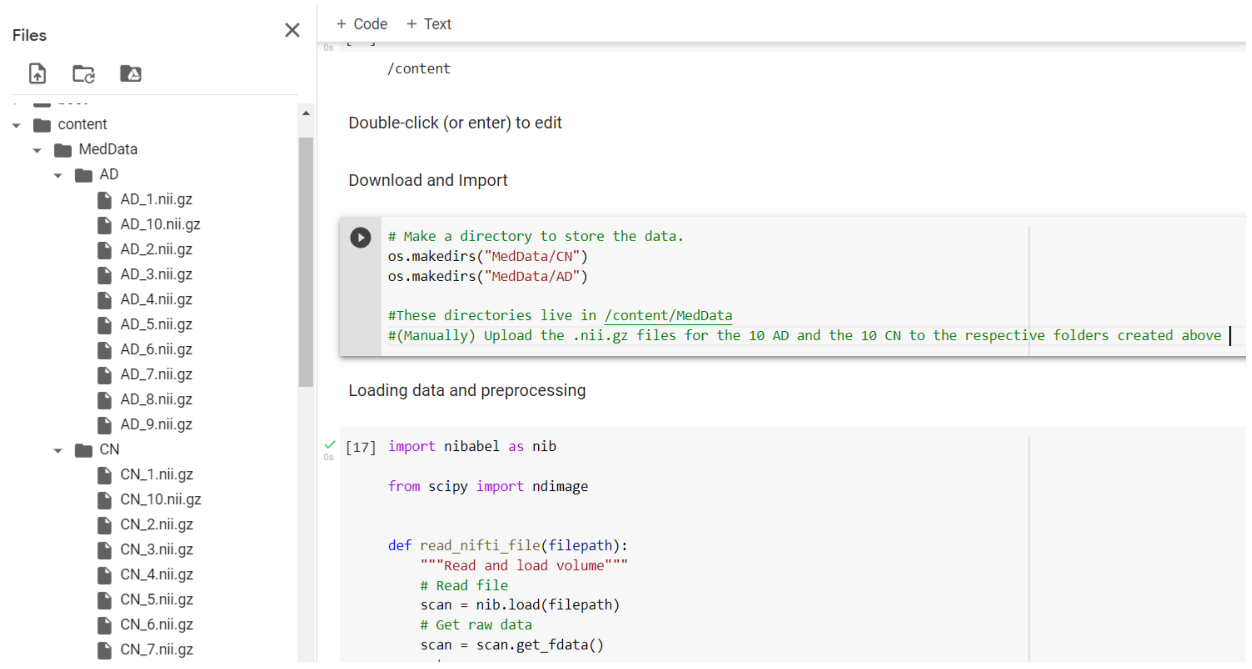


Limitations:

- I did not have a GPU to train for longer epochs to improve the accuracy performance on out of sample data.
- Future work can train for longer epochs (e.g. 100 epochs) for a better performance.

Appendix

Below is the view of the dataset uploaded to their respective folders (Google Colab).



```
/content
```

Double-click (or enter) to edit

Download and Import

```
# Make a directory to store the data.
os.makedirs("MedData/CN")
os.makedirs("MedData/AD")

#These directories live in /content/MedData
#(Manually) Upload the .nii.gz files for the 10 AD and the 10 CN to the respective folders created above
```

Loading data and preprocessing

```
[17] import nibabel as nib

     from scipy import ndimage


     def read_nifti_file(filepath):
         """Read and load volume"""
         # Read file
         scan = nib.load(filepath)
         # Get raw data
         scan = scan.get_fdata()
```

Reference

1. Zunair, H., Rahman, A., Mohammed, N., & Cohen, J. P. (2020, October). Uniformizing techniques to process CT scans with 3D CNNs for tuberculosis prediction. In *International Workshop on PRedictive Intelligence In MEdicine* (pp. 156-168). Springer, Cham.