final_project_SK

March 15, 2021

1 Aim of project

1.1 - To make a model to predict the stage of Alzheimers using Convolutional Neural Networks

Data is available at https://www.kaggle.com/tourist55/alzheimers-dataset-4-class-of-images

About the data directory. The MRI Images have already been converted to .jpg in the dataset Dataset consists of two files - Training and Testing both containing a total of around ~5000 images each segregated into the severity of Alzheimer's

Classes:

1. MildDemented
2. VeryMildDemented
3. NonDemented
4. ModerateDemeneted

[2]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[3]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout,␣
 ↪Flatten, Dense, Activation, MaxPooling2D, Conv2D, ReLU
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import optimizers
```

[4]:
```
# checking tensorflow version
print(tf.__version__)
```

```
2.3.0
```

1.2 Checking data type

The data is stored as

- Alzheimers_s Dataset/ test/ MildDemented/ VeryMildDemented/ NonDemented/ ModerateDemeneted/ train/ MildDemented/ VeryMildDemented/ NonDemented/ ModerateDemeneted/

[5]:
```
main_data_file_path = '/Users/Susheel/Desktop/BIOF399_final_project/Alzheimer_s␣
↪Dataset/'
train_data_file_path = main_data_file_path + 'train/'
test_data_file_path = main_data_file_path + 'test/'
```

[7]:
```
## Parameters
```

[6]:
```
BATCH_SIZE = 32
IMAGE_SIZE = [176, 208]
```

[7]:
```
### getting the dataset ready
```

[8]:
```
train_dataset  = tf.keras.preprocessing.image_dataset_from_directory(
    directory = train_data_file_path,
    image_size = IMAGE_SIZE,
    seed = 1000,
    subset = 'training',
    batch_size = BATCH_SIZE,
    validation_split = 0.2)

validataion_dataset  = tf.keras.preprocessing.image_dataset_from_directory(
    directory = train_data_file_path,
    image_size = IMAGE_SIZE,
    seed = 1000,
    subset = 'training',
    batch_size = BATCH_SIZE,
    validation_split = 0.2)

test_dataset  = tf.keras.preprocessing.image_dataset_from_directory(
    directory = test_data_file_path,
    image_size = IMAGE_SIZE,
    batch_size = BATCH_SIZE)
```

```
Found 5122 files belonging to 4 classes.
Using 4098 files for training.
Found 5122 files belonging to 4 classes.
Using 4098 files for training.
Found 1279 files belonging to 4 classes.
```

Defining the data classes

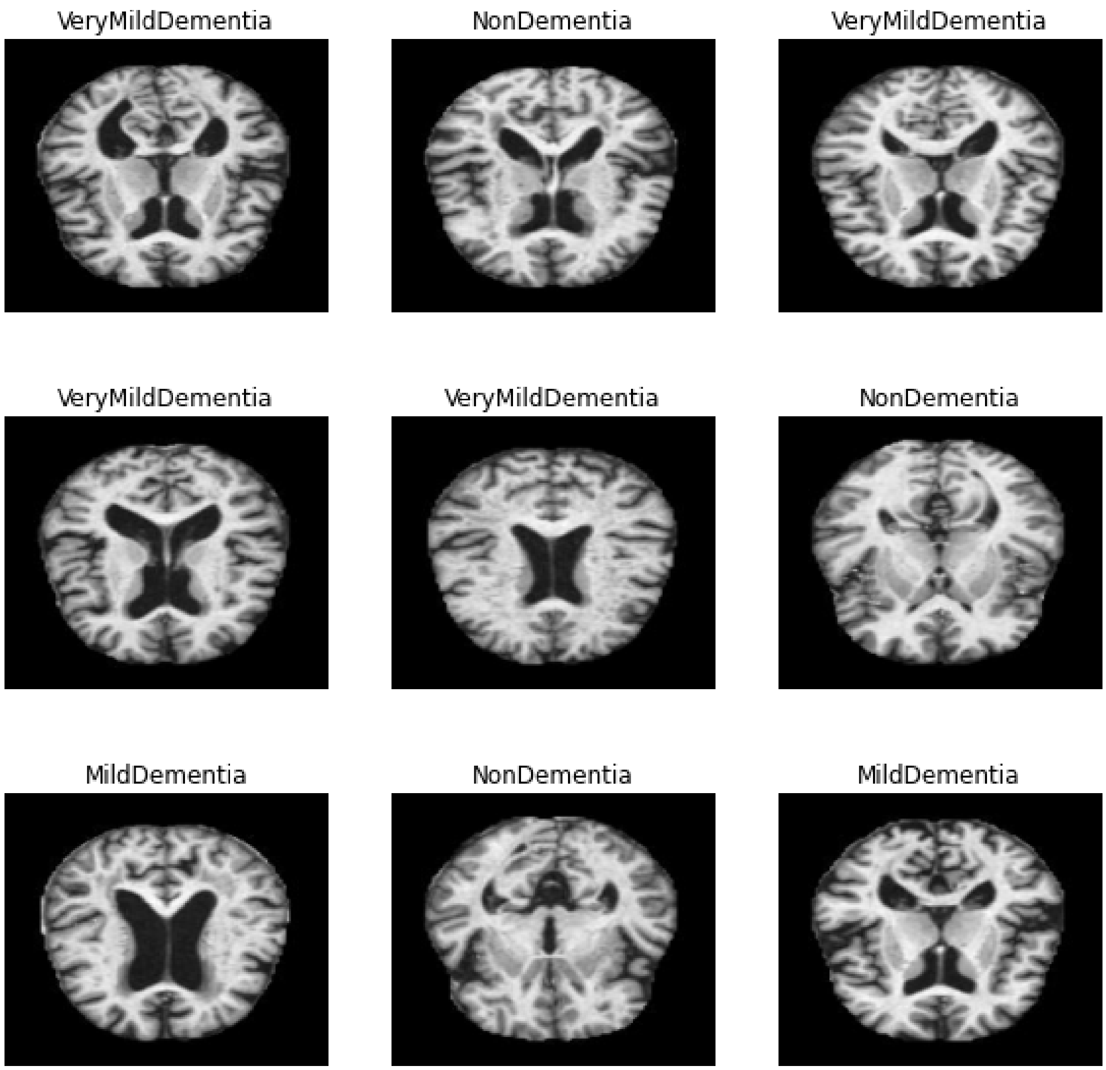
[9]:
```
class_names = ['MildDementia', 'ModerateDementia', 'NonDementia',␣
↪'VeryMildDementia']
train_dataset.class_names = class_names
validataion_dataset.class_names = class_names
```

```
NUM_CLASSES = len(class_names)
```

1.3 Data visualization

[13]:
```
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(train_dataset.class_names[labels[i]])
    plt.axis("off")

plt.savefig('Example_data_type.png')
```



2 Feature Engineering

Because we are working with categorical and noncontinuous data, we want to convert our model into one-hot encodings. One-hot encodings are a way for the model to understand that we're looking at categorial instead of continuous data. Transforming features so that they'll be more understandable is called feature engineering.

[10]:
```
def one_hot_label(image, label):
    label = tf.one_hot(label, NUM_CLASSES)
    return image, label

train_dataset = train_dataset.map(one_hot_label)
validataion_dataset = validataion_dataset.map(one_hot_label)
test_dataset = test_dataset.map(one_hot_label)
```

3 Model Building

[11]:
```
def build_model():

    '''Sequential Model creation'''
    model = Sequential()

    model.add(Conv2D(16,(3,3),padding='same',input_shape =␣
 ↪(IMAGE_SIZE[0],IMAGE_SIZE[1],3),activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding = 'same'))

    model.add(Conv2D(32,(3,3),padding='same',activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding = 'same'))

    model.add(Conv2D(64,(3,3),padding='same',activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding = 'same'))

    model.add(Conv2D(128,(3,3),padding='same',activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding = 'same'))

    model.add(Flatten())
    model.add(Dense(32))
    model.add(ReLU())
    model.add(Dropout(0.25)) # Avoid over-fitting
    model.add(Dense(4))
    model.add(Activation('softmax'))

    return model
```

```python
model = build_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 176, 208, 16)      448       
_________________________________________________________________
batch_normalization (BatchNo (None, 176, 208, 16)      64        
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 88, 104, 16)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 88, 104, 32)       4640      
_________________________________________________________________
batch_normalization_1 (Batch (None, 88, 104, 32)       128       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 44, 52, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 44, 52, 64)        18496     
_________________________________________________________________
batch_normalization_2 (Batch (None, 44, 52, 64)        256       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 26, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 26, 128)       73856     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 11, 13, 128)       0         
_________________________________________________________________
flatten (Flatten)            (None, 18304)             0         
_________________________________________________________________
dense (Dense)                (None, 32)                585760    
_________________________________________________________________
re_lu (ReLU)                 (None, 32)                0         
_________________________________________________________________
dropout (Dropout)            (None, 32)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 132       
_________________________________________________________________
activation (Activation)      (None, 4)                 0         
=================================================================
Total params: 683,780
Trainable params: 683,556
Non-trainable params: 224
_________________________________________________________________
```

[16]:
```
# saving the model summary to a text file
from contextlib import redirect_stdout

with open('modelsummary.txt', 'w') as f:
    with redirect_stdout(f):
        model.summary()
```

[17]:
```
METRICS = [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
           tf.keras.metrics.Precision(name='precision'),
           tf.keras.metrics.Recall(name='recall'),
           tf.keras.metrics.AUC(name='auc')]
```

3.1 Hyperparameters

[18]:
```
num_epochs = 10
initial_learning_rate = 0.001
# optimizer is ADAM
opt = tf.keras.optimizers.Adam(learning_rate=initial_learning_rate, name='Adam')
```

[19]:
```
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=METRICS)
```

3.2 Defining callbacks

[20]:
```
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("alzheimer_model.h5",
                                                    save_best_only=True)

early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=10,
                                                     restore_best_weights=True)
```

4 Model fitting

[21]:
```
history = model.fit(train_dataset,
                    validation_data=validataion_dataset,
                    callbacks=[checkpoint_cb, early_stopping_cb],
                    epochs= num_epochs
                   )
```

```
Epoch 1/10
129/129 [==============================] - 151s 1s/step - loss: 1.4330 -
accuracy: 0.7534 - precision: 0.5135 - recall: 0.2557 - auc: 0.7544 - val_loss:
0.9853 - val_accuracy: 0.7953 - val_precision: 0.7127 - val_recall: 0.3038 -
val_auc: 0.8142
Epoch 2/10
129/129 [==============================] - 148s 1s/step - loss: 0.9891 -
accuracy: 0.7862 - precision: 0.6398 - recall: 0.3316 - auc: 0.8088 - val_loss:
```

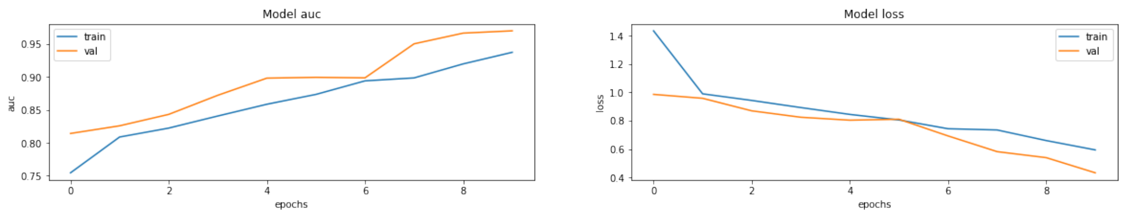
```
0.9576 - val_accuracy: 0.7762 - val_precision: 0.5645 - val_recall: 0.4583 -
val_auc: 0.8258
Epoch 3/10
129/129 [==============================] - 158s 1s/step - loss: 0.9426 -
accuracy: 0.7915 - precision: 0.6613 - recall: 0.3402 - auc: 0.8223 - val_loss:
0.8696 - val_accuracy: 0.7981 - val_precision: 0.6469 - val_recall: 0.4239 -
val_auc: 0.8431
Epoch 4/10
129/129 [==============================] - 184s 1s/step - loss: 0.8926 -
accuracy: 0.7976 - precision: 0.6800 - recall: 0.3599 - auc: 0.8407 - val_loss:
0.8242 - val_accuracy: 0.8087 - val_precision: 0.7137 - val_recall: 0.3924 -
val_auc: 0.8721
Epoch 5/10
129/129 [==============================] - 182s 1s/step - loss: 0.8442 -
accuracy: 0.8049 - precision: 0.7093 - recall: 0.3721 - auc: 0.8585 - val_loss:
0.8034 - val_accuracy: 0.8217 - val_precision: 0.8462 - val_recall: 0.3504 -
val_auc: 0.8982
Epoch 6/10
129/129 [==============================] - 183s 1s/step - loss: 0.8051 -
accuracy: 0.8093 - precision: 0.7156 - recall: 0.3936 - auc: 0.8735 - val_loss:
0.8101 - val_accuracy: 0.8236 - val_precision: 0.9235 - val_recall: 0.3209 -
val_auc: 0.8994
Epoch 7/10
129/129 [==============================] - 194s 2s/step - loss: 0.7435 -
accuracy: 0.8231 - precision: 0.7613 - recall: 0.4258 - auc: 0.8941 - val_loss:
0.6928 - val_accuracy: 0.8062 - val_precision: 0.6469 - val_recall: 0.4954 -
val_auc: 0.8986
Epoch 8/10
129/129 [==============================] - 189s 1s/step - loss: 0.7348 -
accuracy: 0.8297 - precision: 0.7759 - recall: 0.4485 - auc: 0.8985 - val_loss:
0.5820 - val_accuracy: 0.9104 - val_precision: 0.9061 - val_recall: 0.7157 -
val_auc: 0.9503
Epoch 9/10
129/129 [==============================] - 189s 1s/step - loss: 0.6601 -
accuracy: 0.8621 - precision: 0.8296 - recall: 0.5642 - auc: 0.9198 - val_loss:
0.5397 - val_accuracy: 0.9289 - val_precision: 0.9511 - val_recall: 0.7545 -
val_auc: 0.9666
Epoch 10/10
129/129 [==============================] - 204s 2s/step - loss: 0.5940 -
accuracy: 0.8885 - precision: 0.8914 - recall: 0.6308 - auc: 0.9374 - val_loss:
0.4327 - val_accuracy: 0.9197 - val_precision: 0.9185 - val_recall: 0.7450 -
val_auc: 0.9700
```

4.1 Model Visualization

[22]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 3))
ax = ax.ravel()

for i, met in enumerate(['auc', 'loss']):
    ax[i].plot(history.history[met])
    ax[i].plot(history.history['val_' + met])
    ax[i].set_title('Model {}'.format(met))
    ax[i].set_xlabel('epochs')
    ax[i].set_ylabel(met)
    ax[i].legend(['train', 'val'])

plt.savefig("Model_Results.png")
```



4.2 Evaluate the Model

[23]:
```
scores = model.evaluate(test_dataset)
```

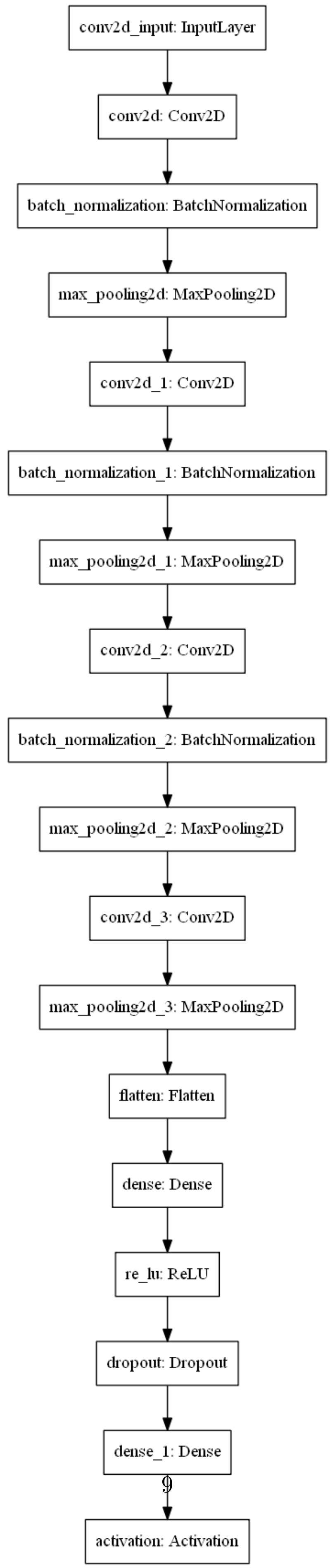
```
40/40 [==============================] - 8s 192ms/step - loss: 1.4601 -
accuracy: 0.7961 - precision: 0.5974 - recall: 0.5661 - auc: 0.8329
```

[26]:
```
%%capture cap
print("Accuracy = ", scores[1])
print("Precision = ", scores[2])
print("Recall = ", scores[3])
print("AUC = ", scores[4])
```

[28]:
```
# saving the scores to a file
with open('modelscores_on_test_dataset.txt', 'w') as f:
    f.write(cap.stdout)
del cap
```

[13]:
```
# Plot the model as a figure
tf.keras.utils.plot_model(model, to_file='model.png')
```

[13]:


conv2d_input: InputLayer
conv2d: Conv2D
batch_normalization: BatchNormalization
max_pooling2d: MaxPooling2D
conv2d_1: Conv2D
batch_normalization_1: BatchNormalization
max_pooling2d_1: MaxPooling2D
conv2d_2: Conv2D
batch_normalization_2: BatchNormalization
max_pooling2d_2: MaxPooling2D
conv2d_3: Conv2D
max_pooling2d_3: MaxPooling2D
flatten: Flatten
dense: Dense
re_lu: ReLU
dropout: Dropout
dense_1: Dense
activation: Activation

```
[ ]:
```