Alzheimer MRI Disease Classification

Aniakwa Nathan

August 6, 2025

1 Introduction

This document presents a machine learning approach for classifying Alzheimer's disease using MRI data. It includes all code cells from the original Jupyter Notebook, along with their inputs and outputs, covering data loading, preprocessing, model building, handling class imbalance, training, evaluation, and visualization. Markdown sections are summarized for context.

2 Load the Data

The dataset is loaded from a Parquet file containing MRI images and labels. The following code installs dependencies and loads the data, with corresponding outputs.

```
!pip install keras-cv
```

```
Requirement already satisfied: keras-cv in /usr/local/lib/python3.11/dist-packages (0
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: regex in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: tensorflow-datasets in /usr/local/lib/python3.11/dist-
Requirement already satisfied: keras-core in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: kagglehub in /usr/local/lib/python3.11/dist-packages (
...
```

```
!pip install tensorflow-addons
```

```
Requirement already satisfied: tensorflow-addons in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: typeguard<3.0.0,>=2.7 in /usr/local/lib/python3.11/dis
```

```
from PIL import Image
import io
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import keras_cv
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten
    , Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.utils.class_weight import compute_class_weight
```

```
import pandas as pd

parquet_file_path = "/content/train-00000-of-00001-
    c08a401c53fe5312.parquet"

# Load the parquet file into a pandas DataFrame
df = pd.read_parquet(parquet_file_path)

display(df.head())
```

	image	label
0	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...	2
1	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...	0
2	{'bytes': b'\xff\d8\xff\e0\x00\x10JFIF\x00\x...	3
3	{'bytes': b'\xff\d8\xff\e0\x00\x10JFIF\x00\x...	3
4	{'bytes': b'\xff\d8\xff\e0\x00\x10JFIF\x00\x...	2

3 Image Preprocessing

The image data is processed by extracting bytes and decoding them. The notebook outlines preprocessing, splitting data, and building a robust classification model to handle class imbalance. The preprocessing code is not fully shown in the provided notebook snippet, but the imports and data loading set the stage.

4 Model Architecture

The convolutional neural network (CNN) architecture is defined, with the following summary output:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 126, 126, 32)      320
max_pooling2d_9 (MaxPooling2D) (None, 63, 63, 32)      0
conv2d_10 (Conv2D)           (None, 61, 61, 64)        18,496
max_pooling2d_10 (MaxPooling2D) (None, 30, 30, 64)     0
conv2d_11 (Conv2D)           (None, 28, 28, 128)       73,856
max_pooling2d_11 (MaxPooling2D) (None, 14, 14, 128)    0
flatten_3 (Flatten)          (None, 25088)             0
dense_6 (Dense)              (None, 128)               3,211,392
dropout_3 (Dropout)          (None, 128)               0
dense_7 (Dense)              (None, 4)                 516
=================================================================
Total params: 3,304,580 (12.61 MB)
```

```
Trainable params: 3,304,580 (12.61 MB)
Non-trainable params: 0 (0.00 B)
```

5 Handle Class Imbalance

Class imbalance is addressed by computing class weights:

```python
# Calculate class weights
classes = np.unique(y_train)
class_weights = compute_class_weight('balanced', classes=classes,
    y=y_train)

# Convert to dictionary format
class_weight_dict = dict(zip(classes, class_weights))

print("Calculated Class Weights:")
print(class_weight_dict)
```

```
Calculated Class Weights:
{np.int64(0): np.float64(1.7695852534562213), np.int64(1): np.float64(26.482758620689
```

6 Model Training

The model is trained for 15 epochs with class weights:

```python
# Train the model
history = model.fit(
    X_train,
    y_train,
    epochs=15, # Increased epochs
    validation_data=(X_val, y_val),
    class_weight=class_weight_dict
)
```

7 Model Evaluation

The model is evaluated using metrics robust to class imbalance:

```python
from sklearn.metrics import classification_report,
   confusion_matrix
import numpy as np

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Get predictions on the test set
y_pred_probs = model.predict(X_test)
```

```
y_pred = np.argmax(y_pred_probs, axis=1)

# Generate classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Generate confusion matrix
print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
display(conf_matrix)
```

8 Visualization

A confusion matrix heatmap is generated:

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=
    False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

9 Analysis of Evaluation Results

The evaluation results are summarized as follows:

- **Overall Accuracy**: 93.95%, but misleading due to class imbalance.
- **Classification Report**:
  - **Precision**: Low for Class 1 (0.06), high for Classes 2 and 3 (0.99).
  - **Recall**: Low for Class 1 (0.30), high for Class 2 (1.00) and Class 3 (0.98).
  - **F1-score**: Low for Class 1 (0.11), high for Classes 2 (0.99) and 3 (0.98).
- **Confusion Matrix**: Class 1 has significant misclassifications (7/10 instances misclassified as Class 0).

10 Key Insights and Next Steps

The model struggles with the minority class (Class 1) despite class weighting. Suggested improvements include resampling, transfer learning, adjusting class weights, or using specialized loss functions like Focal Loss.