

Classification of Alzheimer's Disease Stage Diagnosis: A Benchmark Analysis of Convolutional Neural Network (CNN) Application Utilizing MRI Images

✓ Read the data in, and conducted an exploratory data analysis

First, we will set up this notebook so that it will display multiple outputs for each cell if needed, as well as load the necessary libraries.

```
from IPython.core.interactiveshell import InteractiveShell
from IPython.display import display, HTML
import pandas as pd

InteractiveShell.ast_node_interactivity = "all"

#####
#
#   Set Screen Attributes
#
#####
# set cell width
display(HTML("<style>.container { width:100% !important; }</style>"))

# set cell output window height
display(HTML("<style>div.output_scroll { height: 160em; } </style>"))

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
import shutil
import os
from google.colab import files
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#pip install opencv-python-headless
```

```
#pip install opencv-python
```

```
import numpy as np
import cv2
```

```
#pip install tensorflow
```

```
pip install keras_tuner
```

```
Collecting keras_tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages (from keras_tuner) (3.8.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from keras_tuner) (24.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from keras_tuner) (2.32.3)
Collecting kt-legacy (from keras_tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from keras->keras_tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from keras->keras_tuner) (1.26.4)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras->keras_tuner) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras->keras_tuner) (0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (from keras->keras_tuner) (3.12.1)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras->keras_tuner) (0.14.1)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packages (from keras->keras_tuner) (0.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->keras_tuner) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->keras_tuner) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->keras_tuner) (2.3.0)
```

```
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->keras_tuner) (2025.1.31)
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.11/dist-packages (from optree->keras->keras_tuner) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras->keras_tuner) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras->keras_tuner) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras->keras_tuner) (0
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
----- 129.1/129.1 kB 5.2 MB/s eta 0:00:00
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Installing collected packages: kt-legacy, keras_tuner
Successfully installed keras_tuner-1.4.7 kt-legacy-1.0.5
```

```
import tensorflow as tf
import keras
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import keras_tuner as kt
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Input, GlobalAveragePooling2D
from tensorflow.keras import backend as K
from tensorflow.keras.metrics import AUC, Precision, Recall
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import KFold
from tensorflow.keras import layers
from sklearn.model_selection import StratifiedKFold
from sklearn.utils import shuffle
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.applications import InceptionV3, VGG16
from tensorflow.keras.applications.inception_v3 import preprocess_input
```

Load the data:

README INFO:

dataset_info: features:
<ul style="list-style-type: none">name: image dtype: imagename: label dtype: class_label:
names:
<ul style="list-style-type: none">'0': Mild_Demented'1': Moderate_Demented'2': Non_Demented'3': Very_Mild_Demented
splits:
<ul style="list-style-type: none">name: train num_bytes: 22560791.2 num_examples: 5120name: test num_bytes: 5637447.08 num_examples: 1280 download_size: 28289848 dataset_size: 28198238.28 license: apache-2.0
task_categories:
<ul style="list-style-type: none">image-classification language:en tags:medical pretty_name: Alzheimer_MRI Disease Classification Dataset size_categories:1K<n<10K

Alzheimer_MRI Disease Classification Dataset

The Falah/Alzheimer_MRI Disease Classification dataset is a valuable resource for researchers and health medicine applications. This dataset focuses on the classification of Alzheimer's disease based on MRI scans. The dataset consists of brain MRI images labeled into four categories:

- '0': Mild_Demented
- '1': Moderate_Demented
- '2': Non_Demented
- '3': Very_Mild_Demented

Dataset Information

- Train split:
 - Name: train
 - Number of bytes: 22,560,791.2
 - Number of examples: 5,120
- Test split:
 - Name: test
 - Number of bytes: 5,637,447.08
 - Number of examples: 1,280
- Download size: 28,289,848 bytes
- Dataset size: 28,198,238.28 bytes

Citation

If you use this dataset in your research or health medicine applications, we kindly request that you cite the following publication:

```
@dataset{alzheimer_mri_dataset,
  author = {Falah.G.Salieh},
  title = {Alzheimer MRI Dataset},
  year = {2023},
  publisher = {Hugging Face},
  version = {1.0},
  url = {https://huggingface.co/datasets/Falah/Alzheimer_MRI}
}
```

Usage Example

Here's an example of how to load the dataset using the Hugging Face library:

```
from datasets import load_dataset

# Load the Falah/Alzheimer_MRI dataset
dataset = load_dataset('Falah/Alzheimer_MRI', split='train')

# Print the number of examples and the first few samples
print("Number of examples:", len(dataset))
print("Sample data:")
for example in dataset[:5]:
    print(example)

def disease_label_from_category(category):
    labels = {0: "Mild_Demented", 1: "Moderate_Demented", 2: "Non_Demented", 3: "Very_Mild_Demented"}
    return labels.get(category, "Unknown")

os.getcwd()

#set working Directory
#os.chdir('/Users/mhurt/Documents/MSDS458/Course Project')

➦ 'C:\\Users\\mhurt\\Documents\\MSDS458\\Course Project'

#pip install shutil

# Move kaggle.json to the right location
#shutil.move("kaggle.json", os.path.expanduser("~/kaggle/"))

# Set proper permissions
#os.chmod(os.path.expanduser("~/kaggle/kaggle.json"), 600)

!kaggle datasets download -d borhanitrash/alzheimer-mri-disease-classification-dataset --unzip

➦ Dataset URL: https://www.kaggle.com/datasets/borhanitrash/alzheimer-mri-disease-classification-dataset
License(s): apache-2.0
Downloading alzheimer-mri-disease-classification-dataset.zip to C:\\Users\\mhurt\\Documents\\MSDS458\\Course Project
```


```
0%|          | 0.00/26.0M [00:00<?, ?B/s]
4%|3         | 1.00M/26.0M [00:00<00:04, 5.44MB/s]
15%|#5        | 4.00M/26.0M [00:00<00:01, 14.5MB/s]
27%|##6       | 7.00M/26.0M [00:00<00:01, 19.4MB/s]
39%|###8      | 10.0M/26.0M [00:00<00:00, 20.8MB/s]
54%|####3     | 14.0M/26.0M [00:00<00:00, 22.4MB/s]
69%|#####9   | 18.0M/26.0M [00:00<00:00, 25.0MB/s]
81%|#####   | 21.0M/26.0M [00:01<00:00, 22.3MB/s]
92%|#####2   | 24.0M/26.0M [00:01<00:00, 22.5MB/s]
100%|#####  | 26.0M/26.0M [00:01<00:00, 21.5MB/s]

# Define source and destination paths
testdata_path = "C:/Users/mhurt/Documents/MSDS458/Course Project/Alzheimer MRI Disease Classification Dataset/Data/test-00000-of-00001-44110b9df98
traindata_path = "C:/Users/mhurt/Documents/MSDS458/Course Project/Alzheimer MRI Disease Classification Dataset/Data/train-00000-of-00001-c08a401c5
destination_path = os.getcwd()

# Move the file
shutil.move(testdata_path, destination_path)
shutil.move(traindata_path, destination_path)
```

 'C:\\Users\\mhurt\\Documents\\MSDS458\\Course Project\\test-00000-of-00001-44110b9df98c5585.parquet'
'C:\\Users\\mhurt\\Documents\\MSDS458\\Course Project\\train-00000-of-00001-c08a401c53fe5312.parquet'

uploaded = files.upload()

 Choose Files 2 files

- **test-00000-of-00001-44110b9df98c5585.parquet**(n/a) - 5645961 bytes, last modified: 2/9/2025 - 100% done
- **train-00000-of-00001-c08a401c53fe5312.parquet**(n/a) - 22643887 bytes, last modified: 2/9/2025 - 100% done

Saving test-00000-of-00001-44110b9df98c5585.parquet to test-00000-of-00001-44110b9df98c5585.parquet
Saving train-00000-of-00001-c08a401c53fe5312.parquet to train-00000-of-00001-c08a401c53fe5312.parquet

```
test_data = "test-00000-of-00001-44110b9df98c5585.parquet"
train_data = "train-00000-of-00001-c08a401c53fe5312.parquet"
```

```
test_data = pd.read_parquet(test_data)
train_data = pd.read_parquet(train_data)
```

```
test_data.head() # Display the first few rows
```

```
train_data.head()
```






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





	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\\xd8\\xff\\xe0\\x00\\x10JFIF\\x00\\x...	3	
3	{'bytes': b'\\xff\\xd8\\xff\\xe0\\x00\\x10JFIF\\x00\\x...	3	
4	{'bytes': b'\\xff\\xd8\\xff\\xe0\\x00\\x10JFIF\\x00\\x...	2	

Next steps:

Generate code with train_data

 View recommended plots

New interactive sheet

Generate code with train_data

 View recommended plots

Investigation of Data, Missingness, and Outliers in Testing and Training Data



Test data:

```
# find null counts, percentage of null values, and column type
null_count = test_data.isnull().sum()
null_percentage = test_data.isnull().sum() * 100 / len(test_data)
column_type = test_data.dtypes

# show null counts, percentage of null values, and column type for columns with more than one Null value
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Missing Count', 'Percentage Missing', 'Column Type'])
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Missing', ascending=False)
null_summary_only_missing
```

Missing Count	Percentage Missing	Column Type
---------------	--------------------	-------------

```
def dict_to_image(image_dict):
    if isinstance(image_dict, dict) and 'bytes' in image_dict:
        byte_string = image_dict['bytes']
        nparr = np.frombuffer(byte_string, np.uint8)
        img = cv2.imdecode(nparr, cv2.IMREAD_GRAYSCALE)
        return img
    else:
        raise TypeError(f"Expected dictionary with 'bytes' key, got {type(image_dict)}")

test_data['img_arr'] = test_data['image'].apply(dict_to_image)
test_data.drop("image", axis=1, inplace=True)
test_data.head()

train_data['img_arr'] = train_data['image'].apply(dict_to_image)
train_data.drop("image", axis=1, inplace=True)
train_data
```

	label	img_arr
0	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
	label	img_arr
0	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
...
5115	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5116	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5117	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5118	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5119	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

5120 rows × 2 columns

Next steps: [Generate code with train_data](#) [View recommended plots](#) [New interactive sheet](#)

```
# Check the type of the 'label' column
print(f"Original type of 'label' column: {test_data['label'].dtype}")

# Convert the 'label' column to numerical values
test_data['label'] = pd.to_numeric(test_data['label'], errors='coerce') # Convert to numeric, 'coerce' will turn errors into NaN
```

```
# Check the type again after conversion
print(f"New type of 'label' column: {test_data['label'].dtype}")
```

```
# Optionally, check the updated DataFrame
print(test_data)
```

```
Original type of 'label' column: int64
New type of 'label' column: int64
```

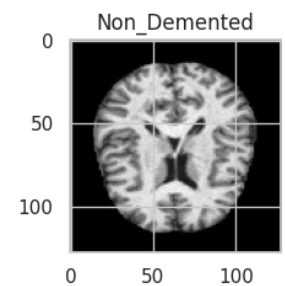
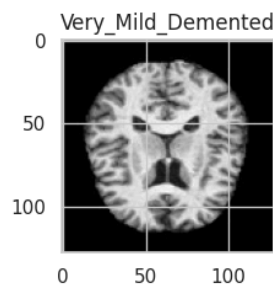
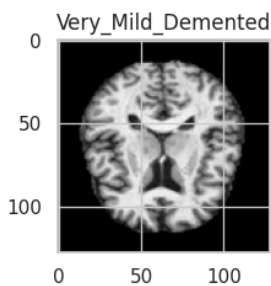
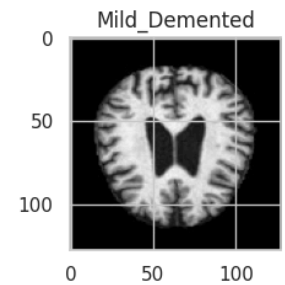
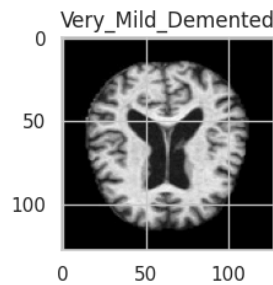
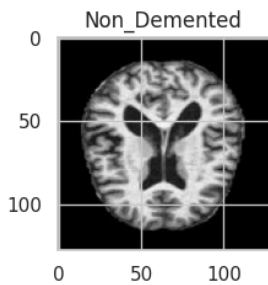
	label	img_arr
0	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
...
1275	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1276	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1277	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1278	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1279	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

[1280 rows x 2 columns]

```
# Check we can actually render the image and that it looks reasonable
fig, ax = plt.subplots(2, 3, figsize=(15, 5))
axs = ax.flatten()
```

```
for axes in axs:
    rand = np.random.randint(0, len(test_data))
    axes.imshow(test_data.iloc[rand]['img_arr'], cmap="gray")
    axes.set_title(disease_label_from_category(test_data.iloc[rand]['label']))
plt.tight_layout()
plt.show()
```

```
<matplotlib.image.AxesImage at 0x79c10c943410>
Text(0.5, 1.0, 'Non_Demented')
<matplotlib.image.AxesImage at 0x79c10c8e7f90>
Text(0.5, 1.0, 'Very_Mild_Demented')
<matplotlib.image.AxesImage at 0x79c10c943f90>
Text(0.5, 1.0, 'Mild_Demented')
<matplotlib.image.AxesImage at 0x79c10c959ed0>
Text(0.5, 1.0, 'Very_Mild_Demented')
<matplotlib.image.AxesImage at 0x79c10c95bad0>
Text(0.5, 1.0, 'Very_Mild_Demented')
<matplotlib.image.AxesImage at 0x79c10c76d310>
Text(0.5, 1.0, 'Non_Demented')
```



```
# Map the numeric labels to disease categories
#test_data['label'] = test_data['label'].map(disease_label_from_category)

# Count the occurrences of each label
test_label_counts = test_data['label'].value_counts().sort_index()
```

```

# Sort the test_label_counts by value in increasing order
test_label_counts = test_label_counts.sort_values()

# Set up the aesthetics using Seaborn
sns.set(style="whitegrid") # White background with gridlines

# Create a dictionary for custom label names
custom_labels = {
    'Mild_Demented': "Mild Dementia",
    'Moderate_Demented': "Moderate Dementia",
    'Non_Demented': "No Dementia",
    'Very_Mild_Demented': "Very Mild Dementia"
}

# Create the bar plot
plt.figure(figsize=(8, 6)) # Set the figure size
ax = sns.barplot(x=test_label_counts.index, y=test_label_counts.values, palette='viridis', legend=False, hue=test_label_counts.index)

# Set labels and title
ax.set_xlabel("Alzheimer's Disease Stage Labels", fontsize=14, fontweight='bold', labelpad=15)
ax.set_ylabel("Frequency", fontsize=14, fontweight='bold', labelpad=15)
ax.set_title("Table 2. Frequency of Test Data Labels", fontsize=16, fontweight='bold')

# Customize the tick marks and labels
# Replace the x-tick labels with custom labels
ax.set_xticklabels([custom_labels.get(label, label) for label in test_label_counts.index], ha='right', fontsize=12)

ax.set_yticklabels(ax.get_yticklabels(), fontsize=12)

# Add value annotations (frequency) on top of each bar
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 0.1, # Positioning the text on top of the bar
           f'{height:.0f}', # Value format
           ha='center', va='bottom', fontsize=12, color='black')

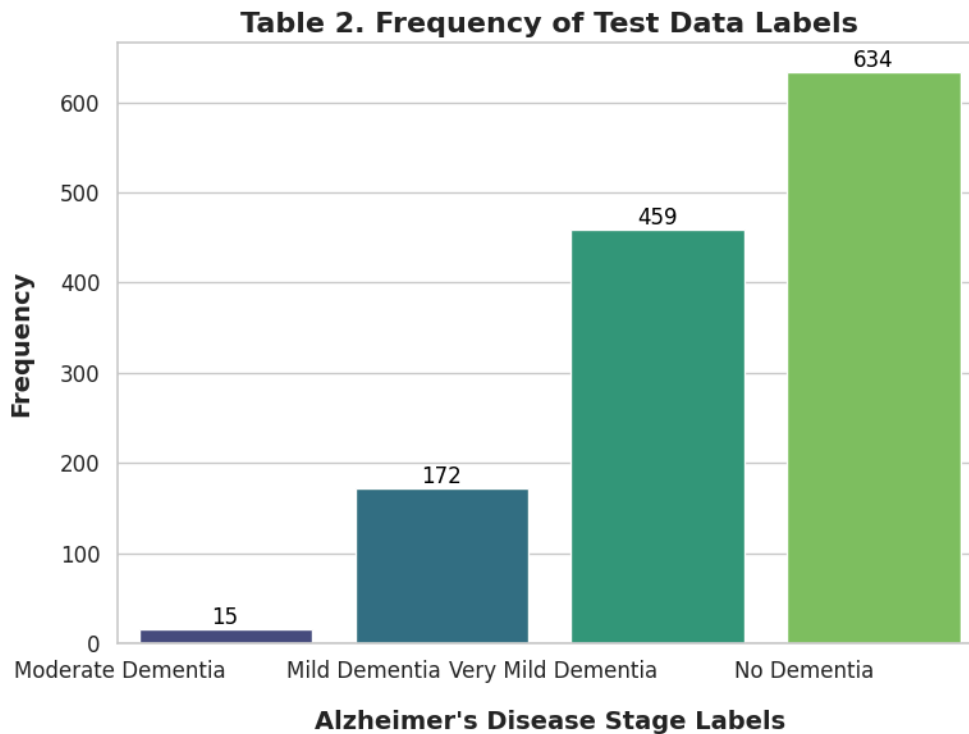
# Display the plot
plt.tight_layout()
# plt.savefig('test_data_label_freqs.png', format='png', dpi=300) # Save with 300 DPI for better resolution
plt.show()

```

```

<Figure size 800x600 with 0 Axes>
Text(0.5, 0, "Alzheimer's Disease Stage Labels")
Text(0, 0.5, 'Frequency')
Text(0.5, 1.0, 'Table 2. Frequency of Test Data Labels')
<ipython-input-25-b672e87e339d>:23: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks()
  ax.set_xticklabels([custom_labels.get(label, label) for label in test_label_counts.index], ha='right', fontsize=12)
[Text(0, 0, 'Moderate Dementia'),
 Text(1, 0, 'Mild Dementia'),
 Text(2, 0, 'Very Mild Dementia'),
 Text(3, 0, 'No Dementia')]
<ipython-input-25-b672e87e339d>:25: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks()
  ax.set_yticklabels(ax.get_yticklabels(), fontsize=12)
[Text(0, 0.0, '0'),
 Text(0, 100.0, '100'),
 Text(0, 200.0, '200'),
 Text(0, 300.0, '300'),
 Text(0, 400.0, '400'),
 Text(0, 500.0, '500'),
 Text(0, 600.0, '600'),
 Text(0, 700.0, '700')]
Text(0.0, 15.1, '15')
Text(1.0, 172.1, '172')
Text(2.0, 459.1, '459')
Text(3.0, 634.1, '634')

```



Train data:

```

# find null counts, percentage of null values, and column type
null_count = train_data.isnull().sum()
null_percentage = train_data.isnull().sum() * 100 / len(train_data)
column_type = train_data.dtypes

# show null counts, percentage of null values, and column type for columns with more than one Null value
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Missing Count', 'Percentage Missing', 'Column Type'])
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Missing', ascending=False)
null_summary_only_missing

```

```

Missing Count  Percentage Missing  Column Type

```

```

# Check the type of the 'label' column
print(f"Original type of 'label' column: {train_data['label'].dtype}")

```



```
# Convert the 'label' column to numerical values
test_data['label'] = pd.to_numeric(train_data['label'], errors='coerce') # Convert to numeric, 'coerce' will turn errors into NaN

# Check the type again after conversion
print(f"New type of 'label' column: {train_data['label'].dtype}")

# Optionally, check the updated DataFrame
print(train_data)
```

```
Original type of 'label' column: int64
New type of 'label' column: int64
```

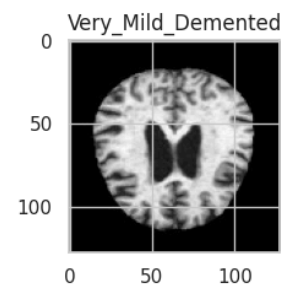
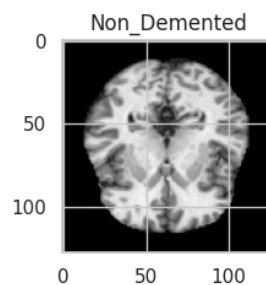
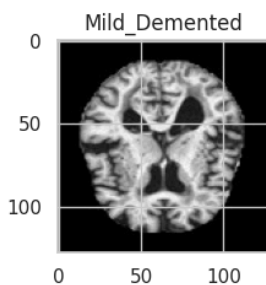
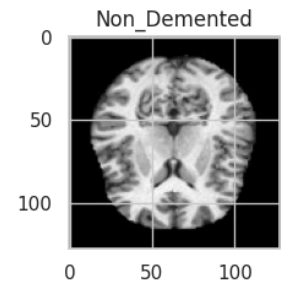
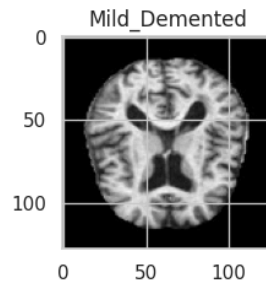
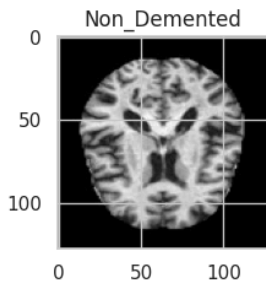
	label	img_arr
0	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
...
5115	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5116	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5117	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5118	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5119	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

[5120 rows x 2 columns]

```
# Check we can actually render the image and that it looks reasonable
fig, ax = plt.subplots(2, 3, figsize=(15, 5))
axs = ax.flatten()
```

```
for axes in axs:
    rand = np.random.randint(0, len(train_data))
    axes.imshow(train_data.iloc[rand]['img_arr'], cmap="gray")
    axes.set_title(disease_label_from_category(train_data.iloc[rand]['label']))
plt.tight_layout()
plt.show()
```

```
<matplotlib.image.AxesImage at 0x79c10c62d410>
Text(0.5, 1.0, 'Non_Demented')
<matplotlib.image.AxesImage at 0x79c10c77bdd0>
Text(0.5, 1.0, 'Mild_Demented')
<matplotlib.image.AxesImage at 0x79c10c5dc0d0>
Text(0.5, 1.0, 'Non_Demented')
<matplotlib.image.AxesImage at 0x79c10c56e310>
Text(0.5, 1.0, 'Mild_Demented')
<matplotlib.image.AxesImage at 0x79c10c61a1d0>
Text(0.5, 1.0, 'Non_Demented')
<matplotlib.image.AxesImage at 0x79c10c467d90>
Text(0.5, 1.0, 'Very_Mild_Demented')
```



```
# Map the numeric labels to disease categories
train_data['label'] = train_data['label'].map(disease_label_from_category)

# Count the occurrences of each label
train_label_counts = train_data['label'].value_counts().sort_index()
```

```
# Sort the test_label_counts by value in increasing order
train_label_counts = train_label_counts.sort_values()

# Set up the aesthetics using Seaborn
sns.set(style="whitegrid") # White background with gridlines

# Create the bar plot
plt.figure(figsize=(8, 6)) # Set the figure size
ax = sns.barplot(x=train_label_counts.index, y=train_label_counts.values, palette='viridis', legend=False, hue=train_label_counts.index)

# Set labels and title
ax.set_xlabel("Alzheimer's Disease Stage Labels", fontsize=14, fontweight='bold', labelpad=15)
ax.set_ylabel("Frequency", fontsize=14, fontweight='bold', labelpad=15)
ax.set_title("Table 1. Frequency of Train Data Labels", fontsize=16, fontweight='bold')

# Customize the tick marks and labels
# Replace the x-tick labels with custom labels
ax.set_xticklabels([custom_labels.get(label, label) for label in train_label_counts.index], ha='right', fontsize=12)
ax.set_yticklabels(ax.get_yticklabels(), fontsize=12)

# Add value annotations (frequency) on top of each bar
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 0.1, # Positioning the text on top of the bar
           f'{height:.0f}', # Value format
           ha='center', va='bottom', fontsize=12, color='black')

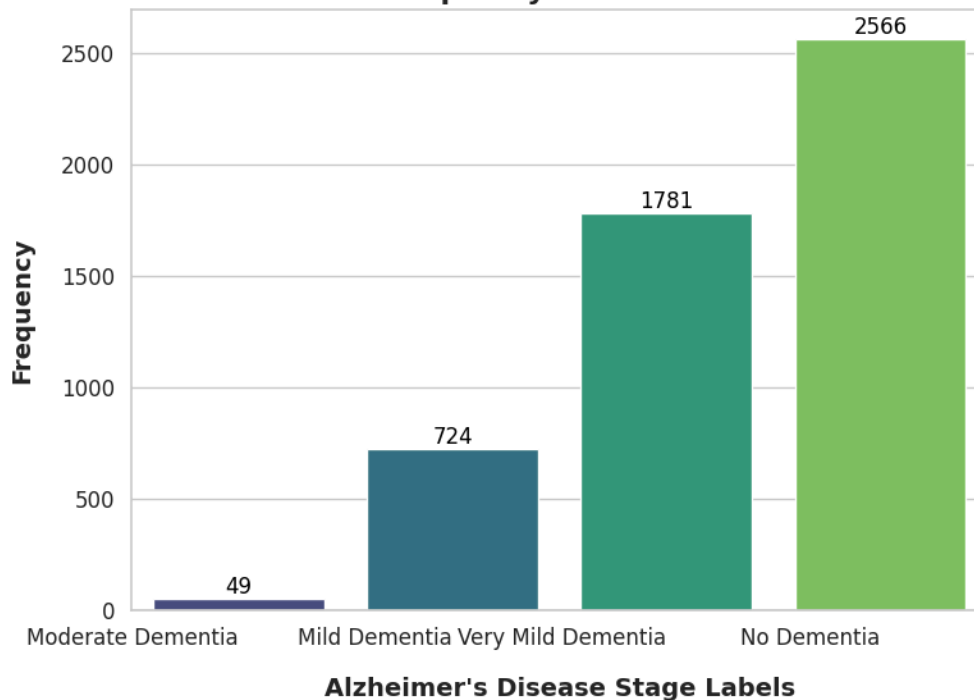
# Display the plot
plt.tight_layout()
#plt.savefig('train_data_label_freqs.png', format='png', dpi=300) # Save with 300 DPI for better resolution
plt.show()
```

```

<Figure size 800x600 with 0 Axes>
Text(0.5, 0, "Alzheimer's Disease Stage Labels")
Text(0, 0.5, 'Frequency')
Text(0.5, 1.0, 'Table 1. Frequency of Train Data Labels')
<ipython-input-30-90c197e70281>:15: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks()
  ax.set_xticklabels([custom_labels.get(label, label) for label in train_label_counts.index], ha='right', fontsize=12)
[Text(0, 0, 'Moderate Dementia'),
 Text(1, 0, 'Mild Dementia'),
 Text(2, 0, 'Very Mild Dementia'),
 Text(3, 0, 'No Dementia')]
<ipython-input-30-90c197e70281>:16: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks()
  ax.set_yticklabels(ax.get_yticklabels(), fontsize=12)
[Text(0, 0.0, '0'),
 Text(0, 500.0, '500'),
 Text(0, 1000.0, '1000'),
 Text(0, 1500.0, '1500'),
 Text(0, 2000.0, '2000'),
 Text(0, 2500.0, '2500'),
 Text(0, 3000.0, '3000')]
Text(0.0, 49.1, '49')
Text(1.0, 724.1, '724')
Text(2.0, 1781.1, '1781')
Text(3.0, 2566.1, '2566')

```

Table 1. Frequency of Train Data Labels



Experiment 1a:

Building a Simple CNN for MRI Image Classification: A Foundational Approach

Prepare the data for CNN

```

# Access the first image: TEST DATA
image_array = test_data.iloc[0]['img_arr']

# Check the number of dimensions in the image array
if len(image_array.shape) == 2:
    # Grayscale image (height, width)
    height, width = image_array.shape
elif len(image_array.shape) == 3:

```

```

# RGB image (height, width, channels)
height, width, channels = image_array.shape
print(f"Channels: {channels}") # This will print the number of color channels (typically 3 for RGB)
else:
    raise ValueError("Unexpected image shape")

print(f"Image Size: {height}x{width} pixels")

↗ Image Size: 128x128 pixels

# Access the first image: TRAIN DATA
image_array = train_data.iloc[0]['img_arr']

# Check the number of dimensions in the image array
if len(image_array.shape) == 2:
    # Grayscale image (height, width)
    height, width = image_array.shape
elif len(image_array.shape) == 3:
    # RGB image (height, width, channels)
    height, width, channels = image_array.shape
    print(f"Channels: {channels}") # This will print the number of color channels (typically 3 for RGB)
else:
    raise ValueError("Unexpected image shape")

print(f"Image Size: {height}x{width} pixels and {channels} channels.")

↗ Image Size: 128x128 pixels and 3 channels.

X_train = np.stack(train_data['img_arr'].values) # Convert list of arrays into a single 4D NumPy array
X_test = np.stack(test_data['img_arr'].values) # Convert list of arrays into a single 4D NumPy array

# Ensure the shape is (num_samples, 128, 128, 1) for grayscale images
if X_train.ndim == 3:
    X_train = np.expand_dims(X_train, axis=-1)
if X_test.ndim == 3:
    X_test = np.expand_dims(X_test, axis=-1)

# Normalize images to range [0,1] if necessary
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(train_data['label']) # Convert labels to integers
y_test_encoded = label_encoder.fit_transform(test_data['label']) # Convert labels to integers

# Check the unique mapping (useful for debugging)
print(dict(zip(label_encoder.classes_, range(len(label_encoder.classes_)))))

# Convert to one-hot encoding
num_classes = len(np.unique(y_train_encoded))
y_train_one_hot = to_categorical(y_train_encoded, num_classes=num_classes)
y_test_one_hot = to_categorical(y_test_encoded, num_classes=num_classes)

↗ {0: 0, 1: 1, 2: 2, 3: 3}

# Split training data into training and validation sets (80% train, 30% val)
X_train_final, X_val, y_train_final, y_val = train_test_split(X_train, y_train_one_hot, test_size=0.3, random_state=42, stratify=y_train_encoded)

# Print final shapes
print(f"Training set: {X_train_final.shape}, {y_train_final.shape}")
print(f"Validation set: {X_val.shape}, {y_val.shape}")
print(f"Test set: {X_test.shape}, {y_test_one_hot.shape}")

↗ Training set: (3584, 128, 128, 1), (3584, 4)
Validation set: (1536, 128, 128, 1), (1536, 4)
Test set: (1280, 128, 128, 1), (1280, 4)

```

▼ Set Initial Parameters

Final Structure:

1. Training Set: X_train_final, y_train_final
2. Validation Set: X_val, y_val
3. Test Set: X_test, y_test_one_hot

```
# Clear session to avoid memory issues
K.clear_session()
```

```
metrics=['accuracy', AUC(), Precision(), Recall()]
```

```
# Define the callbacks
early_stopping = EarlyStopping(
    monitor='val_loss', # Monitor validation loss
    patience=10, # Stop training if val_loss does not improve for 5 consecutive epochs
    restore_best_weights=True # Restore the best model weights after stopping
)
```

```
model_checkpoint = ModelCheckpoint(
    'best_model.keras', # Save the best model
    monitor='val_accuracy', # Track validation accuracy
    save_best_only=True, # Only save models that improve on validation accuracy
    mode='max', # Higher validation accuracy is better
    verbose=1
)
```

```
model = Sequential([
    # Input Layer
    Conv2D(32, (3,3), activation='leaky_relu', padding='same', input_shape=(128, 128, 1)),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    # Second Conv Block
    Conv2D(64, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    Conv2D(64, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    # Third Conv Block
    Conv2D(128, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    Conv2D(128, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    # Fourth Conv Block
    Conv2D(256, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    Conv2D(256, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    # Flatten layer
    Flatten(),

    # Fully Connected Layers
    Dense(512, activation='leaky_relu'),
    Dropout(0.3),
    Dense(256, activation='leaky_relu'),
    Dropout(0.3),

    # Output Layer
    Dense(4, activation='softmax') # 4 classes
])
```

```
#CNN 1 at learning_rate=0.0001: 10 epochs, 64 batch size, dropout layers set to 0.5 w/ relu. accuracy was 78% train, 52% test
#CNN 8 at learning_rate=0.0001: 10 epochs, 128 batch size, dropout layers set to 0.5 w/ leaky relu . accuracy was 87% train, 43% test
#CNN 7 at learning_rate=0.0001: 10 epochs, 64 batch size, dropout layers set to 0.5 w/ leaky relu. accuracy was 92% train, 69% test
#CNN 9 at learning_rate=0.0001: 10 epochs, 128 batch size, dropout layers set to 0.3 w/ leaky relu . accuracy was 98% train, 58% test
#CNN 5 at learning_rate 0.0001: 10 epochs, 64 batch size, dropout layers set to 0.3, w/ leaky_relu. accuracy 98% train, 68% test
optimizer = Adam(learning_rate=0.0001)
```

```
#CNN 4 (need to redo) at learning_rate=0.001: 10 epochs, 64 batch size, dropout layers set to 0.5 w/ relu. accuracy was % train, % test
#CNN 6 at learning_rate 0.001: 10 epochs, 64 batch size, dropout layers set to 0.3 w/ leaky_relu. accuracy 54% train, 19% test
#optimizer = Adam(learning_rate=0.001)
```

```
#CNN 2 (need to redo) at learning_rate=0.01: 10 epochs, 64 batch size, dropout layers set to 0.5 and relu. accuracy was % train, % test#
#optimizer = Adam(learning_rate=0.01)

#CNN 10 at learning rate 0.01 and momentum is 0.0: 10 epochs, 64 batch size, dropout layers set to 0.3 w/ leaky relu. accuracy was 52% train, 36% te
#CNN 11 at learning rate 0.01, momentum = 0.9: NO
#CNN 12 at learning rate 0.1, momentum = 0.9: NO
#CNN 13 at learning rate 0.005, momentum = 0.95: No
#CNN 14 at learning rate 0.001, momentum = 0.5: No
#CNN 15 at learning rate 0.1, momentum = 0.6:
#CNN 16 at learning rate 0.001, momentum = 0.99:
#CNN 17 at learning rate 0.2, momentum = 0.3:
#optimizer = SGD(learning_rate=0.001, momentum = 0.5)

# Compile model
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=metrics)

# Model Summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	320
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 64)	256
conv2d_2 (Conv2D)	(None, 64, 64, 64)	36,928
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73,856
batch_normalization_3 (BatchNormalization)	(None, 32, 32, 128)	512
conv2d_4 (Conv2D)	(None, 32, 32, 128)	147,584
batch_normalization_4 (BatchNormalization)	(None, 32, 32, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_5 (Conv2D)	(None, 16, 16, 256)	295,168
batch_normalization_5 (BatchNormalization)	(None, 16, 16, 256)	1,024
conv2d_6 (Conv2D)	(None, 16, 16, 256)	590,080
batch_normalization_6 (BatchNormalization)	(None, 16, 16, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 256)	0
conv2d_7 (Conv2D)	(None, 8, 8, 512)	1,180,160
batch_normalization_7 (BatchNormalization)	(None, 8, 8, 512)	2,048
conv2d_8 (Conv2D)	(None, 8, 8, 512)	2,359,808
batch_normalization_8 (BatchNormalization)	(None, 8, 8, 512)	2,048
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 512)	4,194,816
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 4)	1,028

› First set of ablation and optimization experiments: relu activation function

[] ↳ 17 cells hidden

› Major changes to this next set of experiments: 0.3 dropout layers, and leaky relu activation function

[] ↳ 8 cells hidden

› Testing dense layers.

[] ↳ 22 cells hidden

> Major changes to this third set of experiments: 0.3 dropout layers, leaky relu activation function, SGD optimizer

[] ↳ 12 cells hidden

✓ This next section explored simpler architectures (i.e., less blocks) compared to the previous four conv block model:

✓ Two conv blocks. As a result of previous experiments, the activation function selected was leaky relu, Adam optimizer set at a learning rate of 0.0001, and with a loss function of categorical cross-entropy

```
def build_model(hp):
    model = Sequential()

    # Explicitly define the input shape
    model.add(Input(shape=(128, 128, 1)))

    # First Conv Block
    model.add(Conv2D(hp.Choice('filters_1', [32, 64]), (3,3), activation='leaky_relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2,2)))

    # Second Conv Block
    model.add(Conv2D(hp.Choice('filters_2', [64, 128, 256]), (3,3), activation='leaky_relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(hp.Choice('filters_2', [64, 128, 256]), (3,3), activation='leaky_relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2,2)))

    # Flatten layer
    model.add(Flatten())

    # Fully Connected Layers
    model.add(Dense(hp.Int('dense_units_1', 256, 512, step=128), activation='leaky_relu'))
    model.add(Dropout(hp.Choice('dropout_1', [0.3])))
    model.add(Dense(hp.Int('dense_units_2', 128, 256, step=64), activation='leaky_relu'))
    model.add(Dropout(hp.Choice('dropout_2', [0.3])))

    # Output Layer
    model.add(Dense(4, activation='softmax')) # 4 classes

    # Compile Model
    model.compile(
        optimizer=Adam(learning_rate=hp.Choice('learning_rate', [1e-4])),
        loss='categorical_crossentropy',
        metrics=metrics
    )

    return model

# Keras Tuner - Random Search
tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=4, # Number of different models to test
    executions_per_trial=1, # Number of times each model is trained
    directory='tuner_results',
    project_name='cnn_tuning'
)

# Run the search
tuner.search(X_train_final, y_train_final, epochs=8, validation_data=(X_val, y_val), verbose=1)
```

```
➡ Trial 4 Complete [00h 13m 16s]
val_accuracy: 0.900390625

Best val_accuracy So Far: 0.919921875
Total elapsed time: 02h 49m 37s
```



```
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(best_hps.values)
```

```
{'filters_1': 32, 'filters_2': 128, 'dense_units_1': 512, 'dropout_1': 0.3, 'dense_units_2': 192, 'dropout_2': 0.3, 'learning_rate': 0.0001}
```

```
model = Sequential([
    # Input Layer (explicitly define input size)
    Input(shape=(128, 128, 1)), # This is the correct way to define the input shape

    # Input Layer
    Conv2D(32, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    # Second Conv Block
    Conv2D(128, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    Conv2D(128, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    # Flatten layer
    Flatten(),

    # Fully Connected Layers
    Dense(512, activation='leaky_relu'),
    Dropout(0.3),
    Dense(192, activation='leaky_relu'),
    Dropout(0.3),

    # Output Layer
    Dense(4, activation='softmax') # 4 classes
])
```

```
best_model = tuner.hypermodel.build(best_hps)
best_model.fit(X_train_final, y_train_final, epochs=8,
               validation_data=(X_val, y_val),
               batch_size= 32, # Adjust batch size based on available memory
               verbose=1, # Shows progress during training,
               callbacks=[early_stopping, model_checkpoint] )
```

```
Epoch 1/8
112/112 ————— 0s 2s/step - accuracy: 0.4675 - auc: 0.8332 - loss: 7.0675 - precision: 0.6583 - recall: 0.6520
Epoch 1: val_accuracy improved from -inf to 0.50130, saving model to best_model.keras
112/112 ————— 230s 2s/step - accuracy: 0.4677 - auc: 0.8329 - loss: 7.0457 - precision: 0.6577 - recall: 0.6514 - val_accuracy:
Epoch 2/8
112/112 ————— 0s 2s/step - accuracy: 0.5565 - auc: 0.7855 - loss: 2.3870 - precision: 0.5610 - recall: 0.5486
Epoch 2: val_accuracy did not improve from 0.50130
112/112 ————— 244s 2s/step - accuracy: 0.5567 - auc: 0.7856 - loss: 2.3856 - precision: 0.5612 - recall: 0.5488 - val_accuracy:
Epoch 3/8
112/112 ————— 0s 2s/step - accuracy: 0.6555 - auc: 0.8553 - loss: 1.6222 - precision: 0.6597 - recall: 0.6520
Epoch 3: val_accuracy did not improve from 0.50130
112/112 ————— 246s 2s/step - accuracy: 0.6555 - auc: 0.8553 - loss: 1.6225 - precision: 0.6596 - recall: 0.6520 - val_accuracy:
Epoch 4/8
112/112 ————— 0s 2s/step - accuracy: 0.7156 - auc: 0.8871 - loss: 1.3521 - precision: 0.7175 - recall: 0.7118
Epoch 4: val_accuracy improved from 0.50130 to 0.54297, saving model to best_model.keras
112/112 ————— 256s 2s/step - accuracy: 0.7156 - auc: 0.8871 - loss: 1.3523 - precision: 0.7175 - recall: 0.7118 - val_accuracy:
Epoch 5/8
112/112 ————— 0s 2s/step - accuracy: 0.7576 - auc: 0.9216 - loss: 1.0204 - precision: 0.7597 - recall: 0.7535
Epoch 5: val_accuracy improved from 0.54297 to 0.68034, saving model to best_model.keras
112/112 ————— 248s 2s/step - accuracy: 0.7576 - auc: 0.9216 - loss: 1.0207 - precision: 0.7597 - recall: 0.7536 - val_accuracy:
Epoch 6/8
112/112 ————— 0s 2s/step - accuracy: 0.7752 - auc: 0.9216 - loss: 1.0653 - precision: 0.7781 - recall: 0.7724
Epoch 6: val_accuracy improved from 0.68034 to 0.87956, saving model to best_model.keras
112/112 ————— 281s 3s/step - accuracy: 0.7754 - auc: 0.9217 - loss: 1.0644 - precision: 0.7782 - recall: 0.7726 - val_accuracy:
Epoch 7/8
112/112 ————— 0s 2s/step - accuracy: 0.8456 - auc: 0.9529 - loss: 0.7351 - precision: 0.8476 - recall: 0.8449
Epoch 7: val_accuracy did not improve from 0.87956
112/112 ————— 273s 2s/step - accuracy: 0.8456 - auc: 0.9529 - loss: 0.7347 - precision: 0.8476 - recall: 0.8450 - val_accuracy:
Epoch 8/8
112/112 ————— 0s 2s/step - accuracy: 0.8890 - auc: 0.9717 - loss: 0.4783 - precision: 0.8896 - recall: 0.8867
Epoch 8: val_accuracy did not improve from 0.87956
112/112 ————— 270s 2s/step - accuracy: 0.8891 - auc: 0.9718 - loss: 0.4780 - precision: 0.8897 - recall: 0.8868 - val_accuracy:
<keras.src.callbacks.history.History at 0x1d7b0214e90>
```

```
test_acc = best_model.evaluate(X_test, y_test_one_hot, verbose=1)
print(f"Test Accuracy: {test_acc}")
```

40/40 11s 264ms/step - accuracy: 0.8566 - auc: 0.9680 - loss: 0.4433 - precision: 0.8670 - recall: 0.8540
 Test Accuracy: [0.4179472327232361, 0.8617187738418579, 0.9713057279586792, 0.8686708807945251, 0.8578125238418579]

```
K.clear_session()
```

```
# Build the model
```

```
model = Sequential([

    # Input Layer (explicitly define input size)
    Input(shape=(128, 128, 1)), # This is the correct way to define the input shape
```

```
    # First Conv Block
```

```
    Conv2D(32, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
```

```
    # Second Conv Block
```

```
    Conv2D(128, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    Conv2D(128, (3,3), activation='leaky_relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
```

```
    # Flatten layer
```

```
    Flatten(),
```

```
    # Fully Connected Layers
```

```
    Dense(512, activation='leaky_relu'),
    Dropout(0.3),
    Dense(192, activation='leaky_relu'),
    Dropout(0.3),
```

```
    # Output Layer
```

```
    Dense(4, activation='softmax') # 4 classes
```

```
])
```

```
# Compile the model
```

```
model.compile(
    optimizer=Adam(learning_rate=0.0001), # Set your preferred learning rate
    loss='categorical_crossentropy', # Appropriate loss function for multi-class classification
    metrics=['accuracy'] # Track accuracy as a metric during training
)
```

```
# You can now train the model with model.fit()
```

```
model.summary()
```

 Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	320
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 128)	36,992
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 128)	512
conv2d_2 (Conv2D)	(None, 64, 64, 128)	147,584
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67,109,376
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 192)	98,496
dropout_1 (Dropout)	(None, 192)	0
dense_2 (Dense)	(None, 4)	772

Total params: 67,394,692 (257.09 MB)

Trainable params: 67,394,692 (257.09 MB)

```
model.fit(X_train_final, y_train_final, epochs=8,
        validation_data=(X_val, y_val),
        batch_size= 32, # Adjust batch size based on available memory
        verbose=1, # Shows progress during training,
        callbacks=[early_stopping, model_checkpoint] )
```

 Epoch 1/8
112/112 ————— 0s 52ms/step - accuracy: 0.4759 - loss: 7.2744
Epoch 1: val_accuracy improved from -inf to 0.50130, saving model to best_model.keras
112/112 ————— 26s 119ms/step - accuracy: 0.4762 - loss: 7.2481 - val_accuracy: 0.5013 - val_loss: 7.5727
Epoch 2/8
112/112 ————— 0s 53ms/step - accuracy: 0.5700 - loss: 2.2380
Epoch 2: val_accuracy did not improve from 0.50130
112/112 ————— 25s 61ms/step - accuracy: 0.5701 - loss: 2.2384 - val_accuracy: 0.5013 - val_loss: 6.7303
Epoch 3/8
112/112 ————— 0s 53ms/step - accuracy: 0.6643 - loss: 1.5907
Epoch 3: val_accuracy improved from 0.50130 to 0.50260, saving model to best_model.keras
112/112 ————— 20s 151ms/step - accuracy: 0.6644 - loss: 1.5896 - val_accuracy: 0.5026 - val_loss: 5.5571
Epoch 4/8
112/112 ————— 0s 53ms/step - accuracy: 0.7302 - loss: 1.1628
Epoch 4: val_accuracy improved from 0.50260 to 0.57682, saving model to best_model.keras
112/112 ————— 16s 110ms/step - accuracy: 0.7303 - loss: 1.1629 - val_accuracy: 0.5768 - val_loss: 1.4135
Epoch 5/8
112/112 ————— 0s 55ms/step - accuracy: 0.7873 - loss: 0.9833
Epoch 5: val_accuracy improved from 0.57682 to 0.75716, saving model to best_model.keras
112/112 ————— 14s 124ms/step - accuracy: 0.7873 - loss: 0.9825 - val_accuracy: 0.7572 - val_loss: 0.7225
Epoch 6/8
112/112 ————— 0s 53ms/step - accuracy: 0.8228 - loss: 0.7966
Epoch 6: val_accuracy improved from 0.75716 to 0.83984, saving model to best_model.keras
112/112 ————— 14s 123ms/step - accuracy: 0.8228 - loss: 0.7968 - val_accuracy: 0.8398 - val_loss: 0.5592
Epoch 7/8
112/112 ————— 0s 52ms/step - accuracy: 0.8605 - loss: 0.6228
Epoch 7: val_accuracy did not improve from 0.83984
112/112 ————— 13s 58ms/step - accuracy: 0.8605 - loss: 0.6229 - val_accuracy: 0.8184 - val_loss: 0.5801
Epoch 8/8
112/112 ————— 0s 52ms/step - accuracy: 0.9080 - loss: 0.3849
Epoch 8: val_accuracy improved from 0.83984 to 0.90169, saving model to best_model.keras
112/112 ————— 18s 128ms/step - accuracy: 0.9080 - loss: 0.3848 - val_accuracy: 0.9017 - val_loss: 0.3498
<keras.src.callbacks.history.History at 0x7cc1331a0bd0>

```
test_acc = model.evaluate(X_test, y_test_one_hot, verbose=1)
print(f"Test Accuracy: {test_acc}")
```

 40/40 ————— 1s 15ms/step - accuracy: 0.4092 - loss: 7.0759
Test Accuracy: [7.220282554626465, 0.40546876192092896]

> LOOCV

[] ↳ 5 cells hidden

> Stratified 5 Kfold CV:

[] ↳ 1 cell hidden

> 5 Kfold CV:

[] ↳ 1 cell hidden

> Three conv blocks. As a result of previous experiments, the activation function selected was leaky relu, Adam optimizer set at a learning rate of 0.0001, and with a loss function of categorical cross-entropy. Filter 1 size was 32, and Filter 2 was kept at 128, both with filter sizes of (3,3). For the dropout layers, the previously identified values were included in the search range in this next randomsearch

[] ↳ 6 cells hidden

> Stratified 5 Kfold CV:

[] ↳ 1 cell hidden

> 5 K-fold CV:

[] ↳ 1 cell hidden

Experiment 1b:

✓ Impact of Data Augmentation on MRI Image Classification Using the Best CNN from Experiment 1a

```
datagen = ImageDataGenerator(
    #preprocessing_function=adjust_brightness_and_contrast, # Apply custom brightness and contrast adjustment
    width_shift_range=0.3, # Shift width by 20%
    height_shift_range=0.3, # Shift height by 20%
    zoom_range=0.1, # Zoom in/out
    fill_mode='nearest' # Filling strategy for empty pixels
)

#X_train, X_test, y_train_one_hot, y_test_one_hot

# Augment images
augmented_images = []
for img_array in train_data['img_arr'].values:
    img_array = np.expand_dims(img_array, axis=0) # Add batch dimension (1, 128, 128, 1)
    img_array = np.expand_dims(img_array, axis=-1)
    augmented_img = next(datagen.flow(img_array, batch_size=1))[0] # Apply augmentation
    augmented_images.append(augmented_img)

# Stack images into a 4D NumPy array
X_train_augmented = np.stack(augmented_images) # Convert list of arrays into a single 4D NumPy array

# Ensure the shape is (num_samples, 128, 128, 1) for grayscale images
if X_train_augmented.ndim == 3:
    # X_train_augmented = np.expand_dims(X_train_augmented, axis=-1)
```

```
# Normalize images to range [0,1] if necessary
X_train_augmented = X_train_augmented.astype('float32') / 255.0

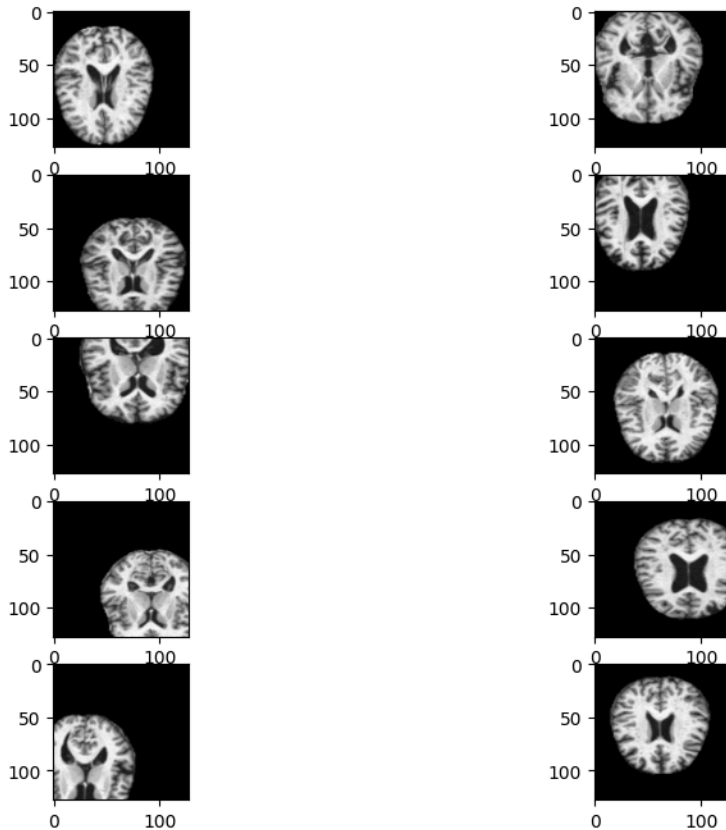
# Define how many images to display
num_images = 10 # Display first 10 images

# Create a figure with subplots (5 rows, 2 columns)
fig, ax = plt.subplots(5, 2, figsize=(10, 8))

# Flatten the axes array for easy iteration
axs = ax.flatten()

# Loop through the first few augmented images and display them
for i in range(num_images):
    axs[i].imshow(X_train_augmented[i].squeeze(), cmap="gray") # Squeeze to remove singleton dims
```

```
<matplotlib.image.AxesImage at 0x7a6f8a53bf90>
<matplotlib.image.AxesImage at 0x7a6f8a51c0d0>
<matplotlib.image.AxesImage at 0x7a6f8a619e10>
<matplotlib.image.AxesImage at 0x7a6f8a50bed0>
<matplotlib.image.AxesImage at 0x7a6f8a59a850>
<matplotlib.image.AxesImage at 0x7a6f8a508750>
<matplotlib.image.AxesImage at 0x7a6f8a5aaa90>
<matplotlib.image.AxesImage at 0x7a6f8a5abe10>
<matplotlib.image.AxesImage at 0x7a6f8a5a9650>
<matplotlib.image.AxesImage at 0x7a6f8a6d5350>
```



```
# Split training data into training and validation sets (80% train, 30% val)
X_train_aug, X_val_aug, y_train_aug, y_val_aug = train_test_split(X_train_augmented, y_train_one_hot, test_size=0.3, random_state=42, stratify=y_t

K.clear_session()

# Compile model
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=metrics)

# Model Summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	320
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 128)	36,992
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 128)	512
conv2d_2 (Conv2D)	(None, 64, 64, 128)	147,584
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67,109,376
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 192)	98,496
dropout_1 (Dropout)	(None, 192)	0
dense_2 (Dense)	(None, 4)	772

Total params: 67,394,692 (257.09 MB)
Total estimated size: 67,394,692 (257.09 MB)

```
history = cnn_aug = model.fit(
    X_train_aug, y_train_aug, # Training data
    validation_data=(X_val_aug, y_val_aug), # Validation data
    epochs=10, # Number of training cycles
    batch_size=32, # Adjust batch size based on available memory
    verbose=1, # Shows progress during training
    callbacks=[early_stopping, model_checkpoint] )
```

```
Epoch 1/10
112/112 ━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.4268 - loss: 5.1174
Epoch 1: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 18s 73ms/step - accuracy: 0.4269 - loss: 5.1043 - val_accuracy: 0.0098 - val_loss: 9.5622
Epoch 2/10
112/112 ━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.5062 - loss: 2.0678
Epoch 2: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 14s 57ms/step - accuracy: 0.5061 - loss: 2.0674 - val_accuracy: 0.0098 - val_loss: 32.3982
Epoch 3/10
112/112 ━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.5267 - loss: 1.5279
Epoch 3: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 6s 57ms/step - accuracy: 0.5267 - loss: 1.5281 - val_accuracy: 0.0104 - val_loss: 22.3801
Epoch 4/10
112/112 ━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.5412 - loss: 1.4891
Epoch 4: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 11s 61ms/step - accuracy: 0.5412 - loss: 1.4891 - val_accuracy: 0.2435 - val_loss: 6.1661
Epoch 5/10
112/112 ━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.5684 - loss: 1.3544
Epoch 5: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 11s 67ms/step - accuracy: 0.5684 - loss: 1.3545 - val_accuracy: 0.5065 - val_loss: 2.2722
Epoch 6/10
112/112 ━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.6028 - loss: 1.2158
Epoch 6: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 7s 62ms/step - accuracy: 0.6028 - loss: 1.2160 - val_accuracy: 0.4811 - val_loss: 1.5266
Epoch 7/10
112/112 ━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.6351 - loss: 1.0819
Epoch 7: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 7s 61ms/step - accuracy: 0.6351 - loss: 1.0817 - val_accuracy: 0.5098 - val_loss: 1.2694
Epoch 8/10
112/112 ━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.6728 - loss: 0.9799
Epoch 8: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 10s 59ms/step - accuracy: 0.6728 - loss: 0.9800 - val_accuracy: 0.5039 - val_loss: 1.4524
Epoch 9/10
112/112 ━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.7418 - loss: 0.7494
Epoch 9: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 7s 59ms/step - accuracy: 0.7417 - loss: 0.7502 - val_accuracy: 0.4915 - val_loss: 1.6549
Epoch 10/10
112/112 ━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.7704 - loss: 0.7040
Epoch 10: val_accuracy did not improve from 0.90169
112/112 ━━━━━━━━━━━ 7s 60ms/step - accuracy: 0.7704 - loss: 0.7044 - val_accuracy: 0.4720 - val_loss: 1.7682
```

Extract training history

```

# Extract training history
epochs = range(1, len(cnn_aug.history['accuracy']) + 1)

# Plot Accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, cnn_aug.history['accuracy'], label='Training Accuracy')
plt.plot(epochs, cnn_aug.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')

# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(epochs, cnn_aug.history['loss'], label='Training Loss')
plt.plot(epochs, cnn_aug.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training vs Validation Loss')

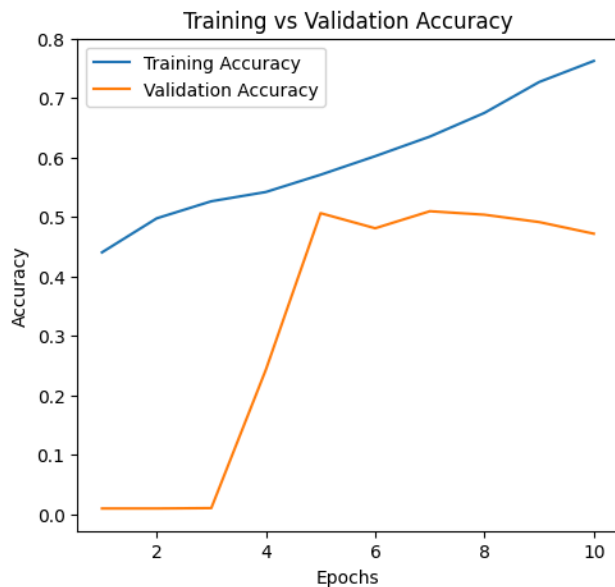
plt.show()

```

```

<Figure size 1200x500 with 0 Axes>
<Axes: >
[<matplotlib.lines.Line2D at 0x7cc08e3c6bd0>]
[<matplotlib.lines.Line2D at 0x7cc082b05150>]
Text(0.5, 0, 'Epochs')
Text(0, 0.5, 'Accuracy')
<matplotlib.legend.Legend at 0x7cc082b3e650>
Text(0.5, 1.0, 'Training vs Validation Accuracy')
<Axes: >
[<matplotlib.lines.Line2D at 0x7cc082156b90>]
[<matplotlib.lines.Line2D at 0x7cc082164e90>]
Text(0.5, 0, 'Epochs')
Text(0, 0.5, 'Loss')
<matplotlib.legend.Legend at 0x7cc0b1965810>
Text(0.5, 1.0, 'Training vs Validation Loss')

```



```

test_acc = model.evaluate(X_test, y_test_one_hot, verbose=1)
print(f"Test Accuracy: {test_acc}")

```

```

40/40 ————— 1s 18ms/step - accuracy: 0.3565 - loss: 1.6716
Test Accuracy: [1.6850637197494507, 0.35468751192092896]

```

Experiment 2a:

Transfer Learning for Alzheimer's Stage Classification: Leveraging CNNs with MRI

Imaging

▼ InceptionV3 Prep: Convert to RGB

```
# Convert grayscale to 3 channels by duplicating the single channel
#X_train_rgb = np.repeat(X_train, 3, axis=-1)
#X_test_rgb = np.repeat(X_test, 3, axis=-1)

# Preprocess for InceptionV3
#X_train_rgb = preprocess_input(X_train_rgb)
#X_test_rgb = preprocess_input(X_test_rgb)

# Convert list of image arrays into a single NumPy array
X_train_rgb = np.stack(train_data['img_arr'].values).astype('float32')
X_test_rgb = np.stack(test_data['img_arr'].values).astype('float32')

# Ensure the shape is (num_samples, 128, 128, 3) for RGB images
if X_train_rgb.ndim == 3:
    X_train_rgb = np.expand_dims(X_train_rgb, axis=-1) # Add channel dimension if missing
if X_test_rgb.ndim == 3:
    X_test_rgb = np.expand_dims(X_test_rgb, axis=-1)

# Handle grayscale images by converting to 3 channels if necessary
if X_train_rgb.shape[-1] == 1:
    X_train_rgb = np.repeat(X_train_rgb, 3, axis=-1)
if X_test_rgb.shape[-1] == 1:
    X_test_rgb = np.repeat(X_test_rgb, 3, axis=-1)

# Compute mean and standard deviation (channel-wise across all images)
mean = np.mean(X_train_rgb, axis=(0, 1, 2), keepdims=True) # Compute mean per channel
std = np.std(X_train_rgb, axis=(0, 1, 2), keepdims=True) # Compute std per channel

mean_test = np.mean(X_test_rgb, axis=(0, 1, 2), keepdims=True) # Compute mean per channel
std_test = np.std(X_test_rgb, axis=(0, 1, 2), keepdims=True) # Compute std per channel

# Apply standardization
X_train_rgb = (X_train_rgb - mean) / std
X_test_rgb = (X_test_rgb - mean_test) / std_test # Use training set mean & std for test data

X_train_rgb.shape, X_test_rgb.shape

((5120, 128, 128, 3), (1280, 128, 128, 3))

# Convert list of image arrays into a single 4D NumPy array
#X_train_rgb = np.stack(train_data['img_arr'].values).astype('float32') / 255.0 # Normalize while stacking
#X_test_rgb = np.stack(test_data['img_arr'].values).astype('float32') / 255.0 # Normalize while stacking

# Ensure the shape is (num_samples, 128, 128, 3) for RGB images
#if X_train_rgb.ndim == 3:
#    X_train_rgb = np.expand_dims(X_train_rgb, axis=-1) # Add channel dimension if missing
#if X_test_rgb.ndim == 3:
#    X_test_rgb = np.expand_dims(X_test_rgb, axis=-1)

# Handle grayscale images by converting to 3 channels if necessary
#if X_train_rgb.shape[-1] == 1:
#    X_train_rgb = np.repeat(X_train_rgb, 3, axis=-1)
#if X_test_rgb.shape[-1] == 1:
#    X_test_rgb = np.repeat(X_test_rgb, 3, axis=-1)

#X_train_rgb.shape
#X_test_rgb.shape

(5120, 128, 128, 3)
(1280, 128, 128, 3)

# Create a custom model based on InceptionV3
def build_inception_model():
    # Set up input tensor for 128x128 images with 3 channels
    #input_tensor = Input(shape=(128, 128, 3))
```



```

# Load InceptionV3 without the top layer and with pre-trained weights
base_model = InceptionV3(
    weights='imagenet',
    include_top=False,
    input_shape=(128, 128, 3)
)

# Freeze the base model layers
base_model.trainable = False

# Add custom layers for classification
x = base_model.output
#x = MaxPooling2D((2, 2))(x)
x = GlobalAveragePooling2D()(x) # Efficient pooling instead of Flatten
#x = Flatten()(x)
x = Dense(512, kernel_initializer='he_normal')(x)
x = BatchNormalization()(x) # Normalize activations
x = tf.keras.layers.Activation('swish')(x)
x = Dropout(0.4)(x)
x = Dense(256, kernel_initializer='he_normal')(x)
x = BatchNormalization()(x) # Normalize activations
x = tf.keras.layers.Activation('swish')(x)
x = Dropout(0.4)(x)
predictions = Dense(4, activation='softmax')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

return model

# Split training data into training and validation sets (70% train, 30% val)
X_train_inc, X_val_inc, y_train_inc, y_val_inc = train_test_split(X_train_rgb, y_train_one_hot, test_size=0.3, random_state=42, stratify=y_train_e

```

› Leaky ReLu

[] ↪ 3 cells hidden

✓ Swish lr = 0.00001

```

K.clear_session()

# Build model
model = build_inception_model()
print("Model created successfully")

# Callbacks
checkpoint = ModelCheckpoint(
    'inception_mri_model.keras',
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=1
)

early_stopping = EarlyStopping(
    monitor='val_loss', # Monitor validation loss
    patience=5, # Stop training if val_loss does not improve for 5 consecutive epochs
    restore_best_weights=True # Restore the best model weights after stopping
)

# Train the model
history = model.fit(
    X_train_inc,
    y_train_inc,
    batch_size=32,
    epochs=25,
    validation_data=(X_val_inc, y_val_inc),
    callbacks=[checkpoint, early_stopping]
)

```

```
# Load the best model before evaluation
best_model = load_model('inception_mri_model.keras')
```

```
# Evaluate the model
test_loss, test_acc = best_model.evaluate(X_test_rgb, y_test_one_hot)
print(f"Test accuracy: {test_acc}; Test loss: {test_loss}")
```

```
Epoch 12/25
110/112 ————— 0s 25ms/step - accuracy: 0.5268 - loss: 1.1193
Epoch 12: val_accuracy improved from 0.59245 to 0.60221, saving model to inception_mri_model.keras
112/112 ————— 30s 274ms/step - accuracy: 0.5268 - loss: 1.1198 - val_accuracy: 0.6022 - val_loss: 1.0153
Epoch 13/25
112/112 ————— 0s 26ms/step - accuracy: 0.5137 - loss: 1.1437
Epoch 13: val_accuracy improved from 0.60221 to 0.60807, saving model to inception_mri_model.keras
112/112 ————— 31s 277ms/step - accuracy: 0.5138 - loss: 1.1435 - val_accuracy: 0.6081 - val_loss: 0.9988
Epoch 14/25
110/112 ————— 0s 32ms/step - accuracy: 0.5398 - loss: 1.0968
Epoch 14: val_accuracy improved from 0.60807 to 0.60872, saving model to inception_mri_model.keras
112/112 ————— 41s 282ms/step - accuracy: 0.5399 - loss: 1.0970 - val_accuracy: 0.6087 - val_loss: 0.9930
Epoch 15/25
111/112 ————— 0s 26ms/step - accuracy: 0.5498 - loss: 1.0844
Epoch 15: val_accuracy improved from 0.60872 to 0.61133, saving model to inception_mri_model.keras
112/112 ————— 40s 274ms/step - accuracy: 0.5498 - loss: 1.0844 - val_accuracy: 0.6113 - val_loss: 0.9830
Epoch 16/25
111/112 ————— 0s 27ms/step - accuracy: 0.5576 - loss: 1.0646
Epoch 16: val_accuracy did not improve from 0.61133
112/112 ————— 15s 41ms/step - accuracy: 0.5580 - loss: 1.0643 - val_accuracy: 0.6113 - val_loss: 0.9721
Epoch 17/25
111/112 ————— 0s 31ms/step - accuracy: 0.5722 - loss: 1.0533
Epoch 17: val_accuracy improved from 0.61133 to 0.61589, saving model to inception_mri_model.keras
112/112 ————— 30s 264ms/step - accuracy: 0.5723 - loss: 1.0531 - val_accuracy: 0.6159 - val_loss: 0.9592
Epoch 18/25
110/112 ————— 0s 27ms/step - accuracy: 0.5797 - loss: 1.0198
Epoch 18: val_accuracy improved from 0.61589 to 0.62826, saving model to inception_mri_model.keras
112/112 ————— 43s 284ms/step - accuracy: 0.5796 - loss: 1.0200 - val_accuracy: 0.6283 - val_loss: 0.9416
Epoch 19/25
112/112 ————— 0s 25ms/step - accuracy: 0.5952 - loss: 1.0130
Epoch 19: val_accuracy improved from 0.62826 to 0.63086, saving model to inception_mri_model.keras
112/112 ————— 39s 270ms/step - accuracy: 0.5951 - loss: 1.0131 - val_accuracy: 0.6309 - val_loss: 0.9426
Epoch 20/25
110/112 ————— 0s 25ms/step - accuracy: 0.5797 - loss: 1.0049
Epoch 20: val_accuracy improved from 0.63086 to 0.63281, saving model to inception_mri_model.keras
```