Report

Alzheimer's Disease Detection

AMIRA ramzi (3cs2)

DRAA Abdel-Illah (3cs1)

Project file is available through google colab link (access with estin gmail account), or through ipynb file attached to the email.

Alzheimer's Disease Detection Project Report

Problem Description and Dataset Overview

Problem Description

Alzheimer's disease is a progressive neurological disorder that affects memory, thinking, and behavior. Early diagnosis can significantly improve the management and care for affected individuals. This project aims to develop a deep learning model capable of classifying brain images into categories associated with different stages of Alzheimer's disease.

Dataset Overview

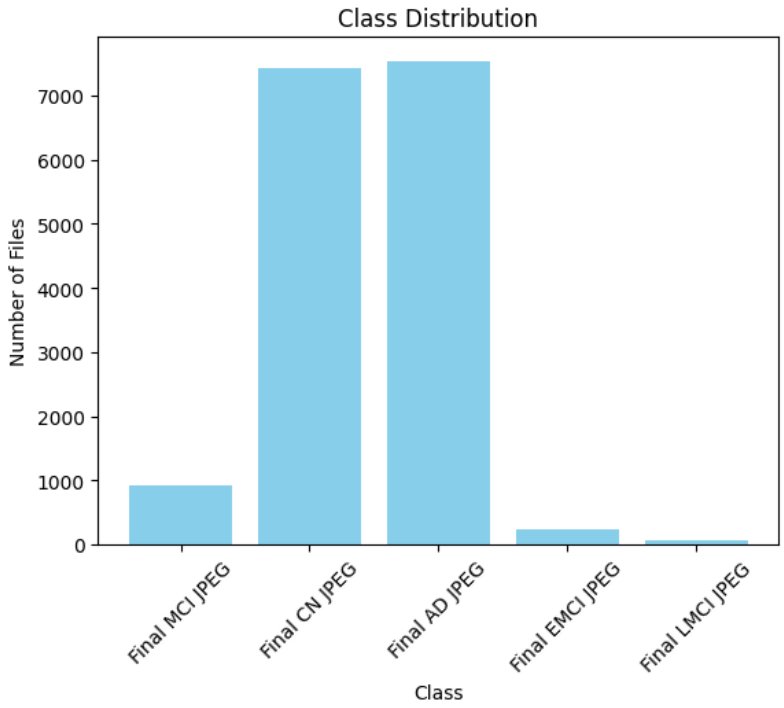
The dataset used for this project was organized into five classes:

- Final AD JPEG (Alzheimer's Disease)
- Final CN JPEG (Cognitively Normal)
- Final EMCI JPEG (Early Mild Cognitive Impairment)
- Final LMCI JPEG (Late Mild Cognitive Impairment)
- Final MCI JPEG (Mild Cognitive Impairment)

Initial Class Distribution

The dataset was initially imbalanced, with significant differences in the number of images per class:

```
Class Counts: {'Final MCI JPEG': 922, 'Final CN JPEG': 7430, 'Final AD JPEG': 7536, 'Final EMCI JPEG': 240, 'Final LMCI JPEG': 72}
```



Methodology and Approach

Data Preprocessing

Handling Class Imbalance

To address the class imbalance, data augmentation was applied to minority classes to increase their sample sizes. This involved generating synthetic variations of existing images through transformations such as rotation, width and height shifts, shear, zoom, and horizontal flipping. The augmentation process was implemented using the ImageDataGenerator class from TensorFlow.

Augmentation Process

- Augmentation was applied only to classes with fewer than 1,000 images.
- Each augmented image was saved in a separate directory corresponding to its class.
- For each class, augmentation was performed until the total number of images reached at least 1,000.

Data Augmentation Details

The following transformations were applied during augmentation:

- **Rescaling**: Pixel values were normalized to the range [0, 1] by dividing by 255.
- **Rotation**: Images were randomly rotated within a range of ±30 degrees.
- **Width and Height Shifts**: Random horizontal and vertical shifts up to 20% of the image dimensions.
- **Shear**: Shearing transformations up to 15%.
- **Zoom**: Random zooming in or out by up to 20%.
- **Horizontal Flip**: Random horizontal flipping to introduce variability.

Class Weight Calculation

Class weights were computed using the compute_class_weight function from sklearn to address imbalance during training. These weights were incorporated into the training process by passing them as a parameter to the “fit” method of the model. Specifically, the class_weight argument was set to a dictionary where each class was associated with its corresponding weight. This adjustment ensured that the loss function penalized errors for underrepresented classes more heavily, effectively balancing their contribution to the gradient updates during optimization.

Final Dataset Preparation

After augmentation, the final dataset was split into training and testing sets:

- Training data for "Final AD JPEG" and "Final CN JPEG" classes was directly used without augmentation.
- Augmented data for "Final EMCI JPEG," "Final LMCI JPEG," and "Final MCI JPEG" classes was split into training (80%) and testing (20%) sets.
- Final class counts:
    - **Train**: {'Final MCI JPEG': 2012, 'Final CN JPEG': 7430, 'Final AD JPEG': 7536, 'Final EMCI JPEG': 18265, 'Final LMCI JPEG': 17849}
    - **Test**: {'Final MCI JPEG': 504, 'Final CN JPEG': 1220, 'Final AD JPEG': 810, 'Final EMCI JPEG': 4567, 'Final LMCI JPEG': 4463}

Data Loading

Data loading was accomplished using the ImageDataGenerator class, which enabled efficient preprocessing of images. The pixel values were normalized to the range [0, 1] by applying rescaling during the data generation process. Separate generators were configured for the training and testing datasets to ensure consistency and proper handling of the respective data splits.

```
# Data generators for training and testing
train_datagen = ImageDataGenerator(rescale=1.0/255)
test_datagen = ImageDataGenerator(rescale=1.0/255)

# Load training data
train_generator = train_datagen.flow_from_directory(
    "final_dataset/train",
    target_size=(250, 250),
    batch_size=32,
    class_mode="categorical"
)

# Load testing data
test_generator = test_datagen.flow_from_directory(
    "final_dataset/test",
    target_size=(250, 250),
    batch_size=32,
    class_mode="categorical"
)
```

Model Architecture Description

The model was designed as a convolutional neural network (CNN) using the TensorFlow Keras API. The architecture is as follows:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout

# Define the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(250, 250, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
```

```
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(5, activation='softmax')  # 5 classes in the dataset
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Architecture Explanation

a. Hyperparameter Selection

1. **Conv2D Layers**:
   - **Filter Sizes**: Filters of sizes (32, 64, and 128) were used to progressively extract spatial hierarchies. Lower filter sizes capture basic features like edges, while higher filter sizes capture more complex patterns.
   - **Kernel Size (3,3)**: A small kernel size helps detect fine details in the images, which is crucial for distinguishing between subtle differences in Alzheimer's stages.
   - **Activation Function (ReLU)**: ReLU is computationally efficient and helps prevent the vanishing gradient problem during backpropagation.
2. **Pooling Layers**:
   - **MaxPooling2D**: This operation reduces the spatial dimensions, ensuring computational efficiency and mitigating overfitting by focusing on dominant features.
   - **Pool Size (2,2)**: A common choice for downsampling, this halves the spatial dimensions, balancing feature retention and dimensionality reduction.
3. **Dense Layers**:
   - **128 Units**: This fully connected layer captures high-level abstractions from the feature maps.
   - **Softmax Output Layer**: Outputs probabilities for the 5 classes, ensuring a normalized sum of probabilities.
4. **Dropout**:
   - **Rate (0.5)**: A dropout rate of 50% was selected to effectively prevent overfitting by randomly deactivating half the neurons during each training iteration.
5. **Compilation Parameters**:
   - **Optimizer (Adam)**: Adam combines the benefits of RMSProp and SGD with adaptive learning rates, making it robust for complex datasets.
   - **Loss Function (Categorical Crossentropy)**: This is ideal for multi-class classification tasks.

b. Layer-by-Layer Explanation

1. **Conv2D Layers**:
   - The first convolutional layer extracts basic patterns like edges and textures.

    - Subsequent layers learn increasingly complex patterns, such as shapes and region-specific features, critical for Alzheimer's stage differentiation.
2. **MaxPooling2D Layers**:
    - Each pooling layer reduces the spatial dimensions, enabling the network to focus on the most salient features while reducing computational load.
3. **Flatten Layer**:
    - Converts the 3D feature maps into a 1D vector, preparing the data for the fully connected layers.
4. **Dense Layer**:
    - The 128-unit dense layer learns high-level representations, aiding in accurate classification.
5. **Dropout Layer**:
    - Regularization via dropout ensures that the model generalizes well on unseen data, reducing the risk of overfitting.
6. **Output Dense Layer**:
    - The softmax activation ensures that the network outputs a probability distribution across the five classes, facilitating clear predictions. **Conv2D Layers**: Three convolutional layers extract spatial features from the input images with increasing filter sizes (32, 64, and 128) for hierarchical feature extraction.

Implementation Details

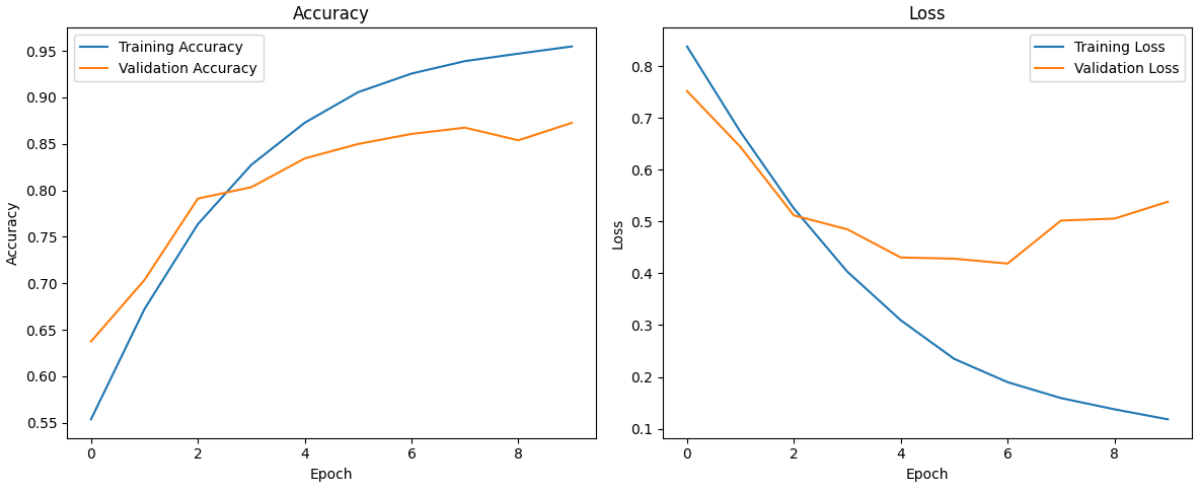
The model was trained for 10 epochs with the following training history:

```
Epoch 1/10
accuracy: 0.5186 - loss: 0.9207 - val_accuracy: 0.6374 - val_loss: 0.7522
...
Epoch 10/10
accuracy: 0.9550 - loss: 0.1188 - val_accuracy: 0.8726 - val_loss: 0.5384
```

Class weights were passed to the fit method to address class imbalance. Data augmentation was applied during training, and validation accuracy consistently improved across epochs.

Results and Analysis

- **Performance Metrics**:
    - Final training accuracy: 95.50%
    - Final validation accuracy: 87.26%

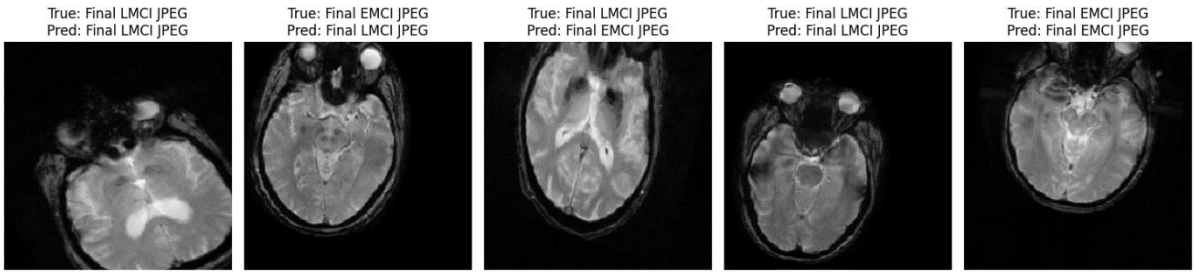


- The model performed well on the validation and test sets, indicating good generalization.
- **Insights**:
    - The use of class weights and data augmentation mitigated class imbalance effectively.
    - Minor overfitting was observed in later epochs, which could be addressed by early stopping or regularization.

Conclusions and Future Work

Conclusions

This project successfully demonstrated the application of deep learning for classifying brain images into Alzheimer's disease stages. The model achieved high accuracy, proving its potential for aiding in medical diagnoses.



Future Work

- Increase dataset size by incorporating additional data sources.
- Explore advanced architectures like ResNet or EfficientNet for further performance improvements.
- Deploy the model as a web-based diagnostic tool with real-time inference capabilities.

References

- TensorFlow Documentation: https://www.tensorflow.org
- Keras API Reference: https://keras.io
- ImageDataGenerator Guide: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator