CSE4288 Introduction to Machine Learning
Team Project Fall 2024
Group 9
Model Development

Team Members:

1- Eren Duyuk 150120509
2- Ufuk Acar 150121071
3- Yusuf Demir 150120032
4- Emir Uyar 150120007
5- Muhammed Hayta 150121068

Introduction

Convolutional Neural Networks (CNNs) have emerged as a dominant approach for tackling image recognition challenges due to their ability to automatically extract hierarchical features from raw data. This report details the development and evaluation of a CNN-based image classifier aimed at categorizing images into ten distinct categories, such as animals, objects, and natural elements. The project leverages the capabilities of TensorFlow/Keras for model implementation and optimization. It addresses various challenges commonly encountered in machine learning projects, including dataset inconsistencies, overfitting, and computational limitations, providing robust solutions to each.

1. Summary of Model

The provided project involves a convolutional neural network (CNN) for image classification. Below are the key details:

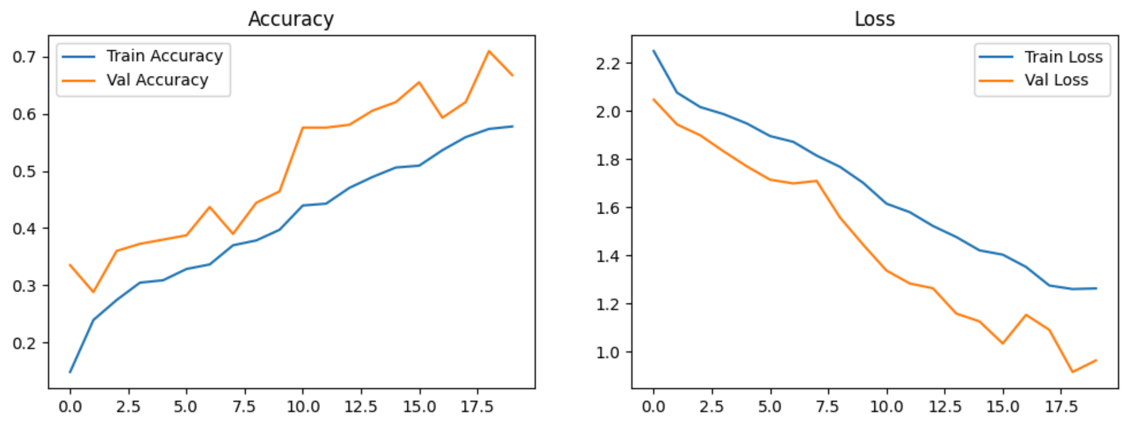
1. **Model Architecture**:
    - The model uses TensorFlow/Keras to define a deep learning structure optimized for image classification. Layers include Conv2D, MaxPooling2D, Flatten, Dense, and Dropout, which collectively allow for feature extraction and classification.
    - It is designed to classify images into 10 distinct categories, such as bird, car, cloud, dog, and more, representing a diverse set of real-world objects.
    - Input images are standardized by resizing them to 64x64 pixels, ensuring uniformity across the dataset for effective training.

2. **Data Handling**:
    - The project employs custom functions to load and preprocess data. This involves reading image files, associating them with corresponding labels, and converting them into tensors for model consumption.
    - Data augmentation is applied using TensorFlow's ImageDataGenerator, which includes transformations like rotation, flipping, and scaling to artificially expand the dataset and enhance model robustness.
    - The dataset is split into training and testing subsets using train_test_split, allowing for separate evaluation of model performance.

3. **Training Configuration**:

- Training is configured with a batch size of 32 and a total of 20 epochs, striking a balance between computational efficiency and learning depth.
- Cross-entropy loss is used as the loss function, and the Adam optimizer is employed to update model weights during training. This choice ensures a fast and adaptive learning process.

4. **Result**:
    - The accuracy and loss plots show that both training and validation performance improve over epochs. Accuracy increases steadily, while loss decreases consistently, indicating effective learning. Minor fluctuations in validation accuracy suggest slight variability, but overall, the model generalizes well without significant overfitting.



2. Challenges Encountered and Solutions

- **Challenge 1**: Dataset inconsistency, such as mismatched or missing image-label pairs.
    - **Solution**: Implemented error handling during data loading to log and skip problematic entries, maintaining the integrity of the dataset.

- **Challenge 2**: Overfitting due to a limited number of training samples.
    - **Solution**: Utilized data augmentation techniques to create more diverse training examples. Dropout layers were also added to the model architecture to reduce reliance on specific neurons.

- **Challenge 3**: Prolonged training times for high-resolution images.
    - **Solution**: Downscaled images to 64x64 pixels, balancing resolution and computational requirements. Additionally, the batch size was optimized to improve GPU utilization.

- **Challenge 4**: Difficulty in visualizing training progress and debugging.
    - **Solution**: Added callbacks such as ModelCheckpoint and EarlyStopping to monitor and log training progress dynamically. Visualized loss and accuracy trends using Matplotlib for better insight.

3. Code Documentation

File: Model.py

- The Model.py file includes a comprehensive implementation of the CNN-based image classifier. Below is a detailed breakdown of its functionality:
    1. **Importing Libraries**: Essential libraries such as TensorFlow/Keras, numpy, PIL, and matplotlib are imported for model building, data preprocessing, and result visualization.

    2. **Configuration**: Key parameters such as categories, img_size, batch_size, and epochs are defined to configure the dataset and model.

    3. **Data Loading**: The load_data() function:
        - Loads images and labels from specified directories.
        - Parses bounding box information for cropping images.
        - Resizes cropped images to 64x64 pixels and normalizes them.
        - Handles exceptions to ensure smooth data processing.

    4. **Data Preprocessing**: Includes normalization and one-hot encoding of labels, and splits the dataset into training, validation, and test subsets using train_test_split.

5. **Data Augmentation**: Employs ImageDataGenerator to perform data augmentation (rotation, translation, scaling, and flipping) for the training set, improving model robustness.

6. **Model Definition**:
   - A CNN is defined with three convolutional layers, each followed by a max-pooling layer for feature extraction.
   - Fully connected (Dense) layers are added for classification, including a softmax output layer for multi-class prediction.

7. **Model Training**: The model is compiled with the Adam optimizer and categorical cross-entropy loss. The fit() function trains the model over 20 epochs with validation data.

8. **Evaluation**: The model is evaluated on the test set, and performance metrics such as accuracy and a classification report are generated to assess its effectiveness.

9. **Model Saving**: The trained model is saved as hand_drawn_classifier.keras for later use.

10. **Visualization**: Training history (accuracy and loss trends) is plotted using matplotlib to analyze the model's learning progress.

File: requirements.txt

- This file enumerates all dependencies, including TensorFlow, numpy, and matplotlib. It ensures reproducibility and simplifies environment setup for other users or collaborators.