National University of Computer and Emerging Sciences

Car Snap (Damaged Car Parts Classification)

Name	Roll No.
Sayyeda Fatima Masood	21L-7571
Abdullah Naeem	21L-1794
Aatiqa Hussain	21L-5397
Abdul Aziz	21L-5380

Role of Each member

Name	Role
Sayyeda Fatima Masood	Initial Classification module (Damaged/Not Damaged)
Abdullah Naeem (Lead)	Damaged Parts Classification + UI + Integration + Blockchain
Aatiqa Hussain	Initial Classification module (Damaged/Not Damaged)
Abdul Aziz	Damaged Parts Classification

Advanced Image Classification System

Introduction

This project aims to develop a sophisticated image classification system using deep learning, specifically focusing on convolutional neural networks (CNNs). The system integrates TensorFlow for model development, utilizes Django with SQLite for backend operations, and incorporates blockchain technology for secure and transparent transactions.

Methodology

Data Collection and Preprocessing

Collection:

The dataset consists of images across various categories, sourced to ensure a wide representation of features within each category. Each category's images are stored in separate subdirectories, facilitating automated label extraction during loading.

Preprocessing:

- **Loading**: TensorFlow's **image_dataset_from_directory** is employed to load and label images automatically. This function streamlines the preparation by handling directory structures and assigning labels based on subfolder names.
- **Resizing and Normalization**: Each image is resized to 256x256 pixels to ensure uniformity. Pixel values are normalized to the range [0, 1] by dividing by 255. This normalization helps in accelerating the convergence during training by keeping feature magnitudes at a similar scale.

CNN Model Design

Architecture:

The CNN comprises several layers designed to extract spatial hierarchies of features:

- **Convolutional Layers**: Three convolutional layers with 256, 128, and 32 filters respectively, each followed by a ReLU activation function to introduce non-linearity.
- **Pooling Layers**: MaxPooling layers follow each convolutional layer to reduce spatial dimensions, thus reducing the number of parameters and computation in the network.

- **Flattening and Dense Layers**: The flattened output is processed through dense layers, culminating in a softmax output layer that classifies the images into one of the categories based on the highest probability.

Compilation:

- **Optimizer**: Adam optimizer is chosen for its adaptive learning rate capabilities, which helps in quicker convergence.
- **Loss Function**: Sparse categorical cross entropy is used as it is suitable for multi-class classification where each class is represented as a unique integer.

Training and Validation

- **Setup**: The model is trained for multiple epochs using batch sizes that fit the computational capacity. A validation split is provided to monitor overfitting during training.
- **Callbacks**: TensorFlow's TensorBoard is used for logging training metrics and validation metrics, providing a visual interface to monitor the training process and make adjustments if needed.

Evaluation and Metrics

- **Test Dataset**: Separate test data is used to evaluate the model after training, ensuring the assessment is unbiased.
- **Metrics Computed**:
  - Accuracy and loss to get an overview of performance.
  - Confusion matrix to understand misclassifications.
  - Precision, Recall, and F1-Score for detailed performance analysis across classes.

Blockchain and Backend Integration

- **Django with SQLite**: Django serves as the backend framework, managing interactions with a SQLite database that stores user and model data. RESTful APIs are developed using Django for seamless interaction between the frontend and the model.
- **Blockchain Technology**:
  - **Metamask**: Used for handling secure user authentication and transactions via Ethereum blockchain. It allows users to manage blockchain transactions directly from their browsers.
  - **Ganache**: Provides a personal Ethereum blockchain to simulate blockchain scenarios during development and testing, facilitating smart contract deployment and testing without real-world costs.

Deployment and Operational Use

- **Model Saving and Loading**: The trained model is saved in HDF5 format, allowing it to be loaded and utilized within the Django application for real-time image classification tasks.
- **Security and Transactions**: Blockchain integration ensures that all transactions related to premium model features are recorded on the Ethereum blockchain, providing transparency and security.

Conclusion

This project exemplifies the integration of advanced machine learning techniques with blockchain technology to create a secure, reliable, and highly functional image classification system. It combines TensorFlow's powerful machine learning capabilities with Django's robust backend framework and blockchain's security features to provide a comprehensive solution for image-based classification tasks.

Damaged Parts Classification Using Image Recognition

Objective:
The aim of this project is to develop an image classification model capable of identifying and classifying damaged parts from images, leveraging deep learning techniques.

Tools and Libraries Used:
- Python
- TensorFlow (a deep learning framework)
- Keras (part of TensorFlow for high-level neural networks API)
- OpenCV (for image processing)
- Matplotlib (for plotting and visualization)
- Scikit-learn (for splitting the data into training and testing sets)

Methodology:

1. Data Preparation:
- Images of various parts, both damaged and undamaged, were collected and labeled accordingly.
- The TensorFlow function `image_dataset_from_directory` was used to load and automatically label the images based on their directory structure.
- All images were resized to a uniform dimension (256x256 pixels) to ensure consistency in input data.

2. Model Building:

- A convolutional neural network (CNN) was constructed using TensorFlow and Keras to process the image data effectively.
- The architecture included several convolutional layers with ReLU activations and pooling layers to extract significant features from the images.
- The network concluded with a dense layer equipped with softmax activation to classify the images into categories of damaged and undamaged parts.

3. Training:

- The model was trained using the prepared dataset with a specific batch size and shuffle settings to optimize learning and ensure randomness.
- The loss function was carefully chosen to minimize the error between the predicted categories and the actual labels during training.

4. Evaluation:

- Precision, recall, and accuracy metrics were used to evaluate the model's effectiveness on a separate test set, ensuring the model's ability to generalize to new data.
- The results provided insights into the model's strengths and potential areas for improvement in classifying damaged parts.

5. Testing:

- Post-training, the model was tested with new images to assess its real-world applicability.
- The model processed these images by resizing and normalizing them before making predictions about their condition (damaged or undamaged).

6. Deployment:

The trained model was saved to a file, making it available for deployment in practical applications, such as automated inspection systems in manufacturing or quality control environments.

Results:

The model achieved commendable precision, recall, and accuracy metrics, demonstrating its capability to accurately classify damaged parts in various contexts.
Specific results and metrics are detailed in the notebook, showcasing the model's performance on both training and testing datasets.

Binary Classification of Images Using Deep Learning

Objective:

The project aims to develop a binary classification model to distinguish between two distinct classes of images using TensorFlow and deep learning techniques.

Tools and Libraries Used:

- Python
- TensorFlow (a deep learning framework)
- Keras (part of TensorFlow for high-level neural networks API)
- OpenCV (for image processing)
- Matplotlib (for plotting and visualization)
- Pandas (for data manipulation)
- Scikit-learn (for model evaluation)
- Pickle (for saving model weights)

Methodology:

1. Data Preparation:

- The dataset, organized in directories per class, was loaded using TensorFlow's
- `image_dataset_from_directory` function.
- Images were resized to 256x256 pixels to maintain uniformity across the dataset.
- Data augmentation techniques such as random flipping and rotation might be implied to improve model robustness (based on common practices, not explicitly detailed in the notebook).

2. Model Building:

- A Convolutional Neural Network (CNN) was constructed using TensorFlow and Keras.
- The CNN architecture likely included several convolutional and pooling layers to extract features, followed by dense layers for classification.
- The model employed a softmax activation function in the output layer to output probabilities of the two classes.

3. Training:

- The model was trained with images labeled according to their respective classes.
- Model training details such as epochs and optimizer choice are essential but were not explicitly mentioned in the extracted content.

4. Evaluation:

- Model performance was evaluated using precision, recall, and accuracy metrics.
- Specific performance metrics indicate how well the model managed to classify images from a test set.

5. Testing:

- Individual test images were processed and classified using the trained model to demonstrate its practical application.
- Predictions were visually and programmatically verified by loading images, resizing them, and using the model to predict their classes.

6. Deployment:

The trained model was saved for deployment, indicating readiness for integration into an application or further development stages.

Results:

The model achieved certain levels of precision, recall, and accuracy, suggesting its effectiveness in classifying images into two categories.
Specific values for these metrics are detailed in the notebook, reflecting the model's practical performance.