Parking Spot Detection System

This is a system for detecting available parking spots using a Convolutional Neural Network (CNN). The system consists of two main components: a Python script for training the CNN model and another script for applying the trained model to real-time video feeds or images to determine parking spot occupancy.

Project File Structure

Based on your directory listing, here's an overview of the project files:

```
project/
├── clf-data/                  # Directory for training data (empty/occupied parking spot images)
├── cnn_parking_model.h5       # The trained CNN model
├── create_parking_mask.py     # Script to create or refine parking spot masks
├── input.jpg                  # Example input image for detection
├── main.py                    # The main parking spot detection script (formerly parking_detection.py)
├── mask_1920_1080.png         # The primary mask image defining parking spot locations
├── mask_crop.png              # Potentially an intermediate or cropped mask file
├── model.ipynb                # Jupyter Notebook for model experimentation or development
├── output/                    # Directory for output files (e.g., parking_status.csv)
├── report.docx                # A word document for project reports or documentation
├── requirements.txt           # Lists Python dependencies and their versions
├── util.py                    # Utility functions, including get_parking_spots_bboxes
└── __pycache__/               # Python's internal cache directory
```

1. Model Training Script (cnn_training.py)

This script is responsible for training a deep learning model to classify whether a parking spot is occupied or empty. It leverages the MobileNetV2 architecture for efficient image classification.

Purpose

The primary goal of this script is to train a binary classification model that can distinguish between "occupied" and "empty" parking spots from image crops.

Key Components and How it Works

- **ImageDataGenerator**: This Keras utility is used for data augmentation and efficient loading of images from directories. It applies various transformations (rotation, zoom, shifts, brightness, horizontal flip) to the training images, which helps in making the model more robust and less prone to overfitting. It also handles image resizing.
- **MobileNetV2**: A pre-trained convolutional neural network known for its efficiency and good performance on mobile and embedded vision applications.
    - weights='imagenet': Initializes the model with weights pre-trained on the ImageNet dataset, allowing for transfer learning.
    - include_top=False: Excludes the top classification layer of MobileNetV2, as we will add our own custom classification layers for our specific task.
    - input_shape=(128, 128, 3): Defines the expected input image size (width, height, channels).
- **Transfer Learning**: The base_model.trainable = False line freezes the weights of the pre-trained MobileNetV2. This means only the newly added layers will be trained, significantly speeding up training and requiring less data.
- **Custom Classification Head**:
    - GlobalAveragePooling2D(): Reduces the spatial dimensions of the feature maps from the base model, preparing them for the dense layers.
    - Dropout(0.3): A regularization technique that randomly sets a fraction of input units to 0 at each update during training, which helps prevent overfitting.
    - Dense(64, activation='relu'): A fully connected layer with ReLU activation.
    - Dense(1, activation='sigmoid'): The output layer. Since it's a binary classification problem (occupied/empty), a single neuron with a sigmoid activation function is used. The sigmoid outputs a probability between 0 and 1.
- **Model Compilation**:
    - optimizer=Adam(learning_rate=1e-4): Uses the Adam optimizer with a small learning rate, suitable for fine-tuning.
    - loss='binary_crossentropy': The standard loss function for binary classification problems.
    - metrics=['accuracy']: Monitors the accuracy during training.
- **Model Training (model.fit)**: The model is trained using the train_generator and validated with val_generator for a specified number of epochs.
- **Model Saving**: After training, the model is saved to cnn_parking_model.h5 for later use.

How to Use

1. **Data Preparation**: Create a directory structure like this:
   clf-data/
   ├── empty/
   │   ├── 00000000_00000161.jpg
   │   ├── 00000000_00000164.jpg
   │   └── ...
   └── not_empty/
       ├── 00000000_00000003.jpg
       ├── 00000000_00000005.jpg
       └── ...

   Place images of empty parking spots in the empty folder and occupied parking spots in the occupied folder.
2. **Run the Script**: Execute the cnn_training.py script (or the relevant cells in model.ipynb). It will load the data, train the model, and save it.

   cmd
   python cnn_training.py

2. Parking Spot Detection Script (main.py)

This main.py uses the trained CNN model to analyze video frames or static images, identify parking spots based on a mask, and determine their occupancy status.

Purpose

To provide a real-time visual representation of parking spot availability and generate a summary CSV report.

Key Functions and How it Works

- **predict_spot_cnn(spot_img)**:
  - Takes an image crop of a single parking spot as input.
  - Resizes the image to (128, 128), normalizes pixel values to [0, 1], and adds a batch dimension.
  - Uses the loaded CNN model (model.predict) to get a prediction (probability).
  - Returns True if the spot is predicted to be occupied (probability > 0.5), False otherwise.
- **process_frame_with_cnn(frame, spots, spots_status)**:
  - Iterates through each defined parking spot (spots).

    - For each spot, it crops the corresponding region from the current frame.
    - Calls predict_spot_cnn to get the occupancy status for the cropped spot.
    - Updates spots_status array.
    - Draws a rectangle around each parking spot on the frame: green for empty, red for occupied.
    - Displays the total available spots count on the frame.
    - Resizes and displays the frame in a cv2.imshow window.
- **main(mask_path, input_path)**:
    - **Mask File**: Loads a grayscale mask image. This mask defines the precise locations of parking spots. Each connected component in the mask represents a parking spot.
    - **get_parking_spots_bboxes**: This function (from util.py) takes the connected components from the mask and extracts the bounding box coordinates (x, y, w, h) for each parking spot.
    - **Video/Image Processing**:
        - If input_path is a video file, it opens the video using cv2.VideoCapture. It then processes frames at a step interval to reduce computational load.
        - If input_path is an image file, it processes the single image.
        - Press 'q' to quit the video processing loop.
    - **CSV Output**: After processing (either the video finishes or the image is processed), it calculates the total, occupied, and available slots and saves this information to a CSV file named parking_status.csv in the output directory.

How to Use

1. **Trained Model**: Ensure you have the cnn_parking_model.h5 file in the same directory as this script (or provide the correct path).
2. **Mask File**: You need a binary mask image (e.g., mask_1920_1080.png) where each parking spot is represented by a distinct white region on a black background. This mask is crucial for defining the areas to be analyzed.
3. **util.py**: This script depends on util.py which contains the get_parking_spots_bboxes function.
4. **Input Video/Image**: Provide the path to your input video (e.g., .mp4, .avi) or image (e.g., .jpg, .png).
5. **Create Output Directory**: Make sure there's an output directory in the same location where you run the script, as the CSV file will be saved there.
6. **Run the Script**:
   cmd python main.py
   You might need to adjust the mask_path and input_path variables in the if __name__ == "__main__": block within main.py to match your file locations.

3. Other Project Files

- **create_parking_mask.py**: This script is used to manually or semi-automatically generate the mask_1920_1080.png (or mask_crop.png) file by allowing a user to define parking spot regions on an image. This is a crucial preliminary step for defining the areas the detection system will monitor.
- **model.ipynb**: A Jupyter Notebook file, commonly used for interactive development, experimentation, and presenting machine learning workflows. It might contain the code for training the CNN model, data exploration, or model evaluation.
- **requirements.txt**: This file lists all the Python packages and their exact versions required to run the project. It ensures that anyone setting up the project can install the correct dependencies.

Prerequisites

Before running these scripts, ensure you have the following Python libraries installed with the specified versions. You can install them using the requirements.txt file:

- numpy==1.26.4
- opencv-python==4.11.0.86
- tensorflow==2.18.0
- tensorflow_intel==2.18.0
- tensorflow_object_detection_api==0.1.1
- Pillow (often installed as a dependency of tensorflow or opencv-python, but good to check)

To install all dependencies from requirements.txt:

pip install -r requirements.txt

Conclusion

This system provides a robust way to monitor parking spot availability using deep learning. The training script allows you to create a specialized model, while the detection script applies this model to real-world scenarios, offering both visual feedback and a data summary. The additional files indicate a well-structured project with tools for mask creation, interactive development, and formal reporting.