README

1. Power Consumption Measurement

The measurement of energy consumption in this study relies on the `emissions_tracker.py` file provided by CodeCarbon. Specifically, the method `def _do_measurements(self) -> float:` is utilized to calculate the total energy consumed since the beginning of the process. The returned value, `total_energy_kwh`, represents the cumulative energy usage in kilowatt-hours (kWh) for all hardware components involved.

For more details, refer to the official CodeCarbon GitHub repository: `https://github.com/mlco2/codecarbon`.

2. ImageNet Training

This project focuses on training a deep learning model on the ImageNet dataset while evaluating both its accuracy and power consumption. The dataset is managed using the `ImageNetDataset` class from `imagenet.py`, which organizes the data into a training-validation split and applies preprocessing transformations like resizing, cropping, and normalization.

In `main.py`, we use a pre-trained `gcvit_small` model from Hugging Face's Transformers library. The model is fine-tuned on ImageNet data, with modifications to the final classification layer to match the dataset's requirements. The model is downloaded directly from Hugging Face's model repository, ensuring seamless integration and flexibility. More details on Hugging Face models can be found at https://huggingface.co/models.

To monitor energy consumption, the CodeCarbon library is used. The `EmissionsTracker` class tracks the total energy used (in kilowatt-hours) throughout the training process, allowing an analysis of power efficiency alongside performance metrics. The results, including training and validation accuracy, loss, and power usage, are logged into `result.xlsx`. The best model is saved as `best.pt`, and the final state is stored as `last.pt`.

To replicate the experiment, prepare the ImageNet dataset in the expected directory structure, install required dependencies, and run `main.py`. Hugging Face's model repository simplifies access to pre-trained models, improving both performance and adaptability in deep learning workflows.

3. MMSegmentation for Semantic Segmentation

This project is based on the MMSegmentation framework to evaluate various models in semantic segmentation tasks, aiming to analyze the trade-off between performance and energy efficiency. The experiments use classic datasets such as Cityscapes and evaluate models including MobileNet, PSANet, DeepLabV3, APCNet, and the SETR series (mla, naive, and pup). All training and inference tasks are implemented using the MMSegmentation framework. Initially, models provided by MMSegmentation are loaded and adjusted based on specific task requirements.

The selection and invocation of models follow the standard process of the MMSegmentation framework. This is achieved by specifying the model path under `segmentation` `mmsegmentation`
`configs` . Models can be run by appending the corresponding configuration file path after the `mmsegmentation\tools\train.py` script. Batch size, training iterations, and other parameters are adjusted during the experiments to ensure fair comparisons across different models.

Energy consumption measurement is conducted using the CodeCarbon library, which records the total power usage of GPUs and CPUs during training in kilowatt-hours (kWh). Additionally, model parameters such as FLOPs and Params can be obtained using the `mmsegmentation\tools\analysis_tools\get_flops.py` script to facilitate the final performance and efficiency evaluation. The experiments record metrics such as average accuracy (aAcc), sustainability metrics (SAM), the F-Measure of sustainability (FSM), and the area under the sustainability curve (ASC). These metrics provide critical insights into optimizing the balance between performance and energy efficiency. The modularity and flexibility of the MMSegmentation framework ensure the reproducibility of these experiments.

The MMSegmentation project can be found at https://github.com/open-mmlab/mmsegmentation.

4. MMPose for Animal Pose Estimation

This project is based on the MMPose framework to conduct experiments on animal pose estimation tasks, focusing on the analysis of model complexity, energy consumption, and accuracy performance. The experiments use the AP-10K dataset, a large-scale benchmark for animal pose estimation containing over 10,000 images with high-quality keypoint annotations suitable for keypoint detection tasks.

During the experiments, pre-trained model weights provided by MMPose are loaded, and model configurations are adjusted based on task requirements. This includes setting the input image size, configuring the output of the keypoint detection head, and optimizing training parameters. All training and evaluation tasks are conducted by invoking the `mmpose\tools\train.py` script, specifying the corresponding model configuration file path. The configuration files are located under `mmpose\configs` and can be selected based on the specific task requirements.

Energy consumption is monitored using the CodeCarbon library, which records the total power usage of GPUs and CPUs during training in kilowatt-hours (kWh). The performance metrics include average keypoint accuracy (aAcc), floating-point operations (FLOPs), and the number of parameters (Params). Sustainability metrics such as SAM, FSM, and ASC are also computed to evaluate the trade-off between performance and energy efficiency.

Additionally, the `mmpose\tools\analysis_tools\get_flops.py` script can be used to analyze model complexity (e.g., FLOPs and Params), providing critical insights into understanding the balance between energy consumption and accuracy. The modularity and usability of MMPose make it well-suited for a wide range of animal pose estimation tasks.

The MMPose project can be found at https://github.com/open-mmlab/mmpose.