Overview of the Paper and Project

Paper Summary:
We work with a paper named "Geography-->Aware Self-->Supervised Learning," focused on inculcating geographical information into the learning process. It strives to develop machine learning models. An essential idea is that images, with their associated geographical metadata (latitude and longitude), are trained together in a model so they can understand and leverage geographical contexts into different tasks.

Project Objective:
This project involves the implementation of methods described in the paper to devise a machine-->learning model that uses geographical information to improve its performance. Outputs from the project include:
1. A trained model that integrates geographical context into its learning process.
2. Testing the performance of the model through evaluation metrics like cosine similarity and mean squared error.

Project Step-->-->>by-->-->>Step Progress

1. Setting Up the Environment:
-->-->> Folder Structure :
-->-->> Set up a structured project directory containing data, output, and src folders for source code.
-->-->> Data Preparation:
-->-->> Downloaded and placed the "classes_in_imagenet.csv" file into a 'data' folder.

2. Downloading Images:
-->-->> Flickr API Setup:
-->-->> I created a Flickr account and got myself an API keys.
-->-->> Modified the "download_images.py" script to download images using the Flickr API based on queries in the CSV file.
-->-->> Images Downloaded successfully and saved in folder "output/images".

3. Image Preprocessing
-->-->> Preprocess Script:
-->-->> Added "preprocess_images.py" script to preprocess downloaded images and store their paths with metadata into "processed_images.csv".
-->-->> Missing metadata files were handled by skipping the corresponding images.

4. Model Training:
-->-->> Model Training Script:
-->-->> Used "train.py" to train a contrastive learning model (MoCo).
-->-->> Modified the script to not re-->-->>train the model if it has already been trained.
Successfully saved your trained model: "moco_model.pth".

5. Testing the Model:
-->-->> Evaluation Script:
* Used "evaluate.py" to evaluate the model.
-->-->> Calculated measures, such as cosine similarity and MSE, to validate the model.

Current Status and Issues

Progress:

-->-->> The images were downloaded and preprocessed successfully.

-->-->> MoCo model trained and saved.

-->-->> Tried to evaluate the model but had some problems with the model loading and mismatches in the state_dict.

Current Issues:

1. Model Load Errors:

-->-->> Loading state_dict for "LatLonToEmbedding" errors with mismatched keys and sizes.

2. Meta File Not Found:

-->-->> Some images lack metadata files, causing a skip during the preprocessing stage.

3. Evaluate Script Errors:

-->-->> Problems with access to local variables, and providing for the right model used for evaluation.

Folder Structure

```
Geo_Project/
├── data
│   └── classes_in_imagenet.csv
├── notebooks
│   ├── .ipynb_checkpoints
│   ├── evaluation_final.ipynb
│   ├── MoCoV2_Framework.ipynb
│   ├── model_evaluation.ipynb
│   └── visualizations.ipynb
```

```
├── output
│   ├── images/
│   ├── geography_aware_model.pth
│   ├── moco_model.pth
│   ├── processed_images.csv
│   └── valid_images.csv
└── src
    ├── __pycache__
    ├── custom_dataset.py
    ├── custom_model.py
    ├── deploy.py
    ├── download_images.py
    ├── evaluate.py
    ├── geo_model.py
    ├── inspect_csv.py
    ├── main.py
    ├── moco.py
    ├── mocov2.py
    ├── model.py
    ├── preprocess_images.py
    └── train.py
```

Use of Each File

data

- **classes_in_imagenet.csv**: The initial dataset containing image class names and geographical coordinates.

notebooks

- **evaluation_final.ipynb**: Notebook for final evaluation of the model.
- **MoCoV2_Framework.ipynb**: Notebook containing the framework for the MoCoV2 model.
- **model_evaluation.ipynb**: Notebook for evaluating different models and comparing their performance.
- **visualizations.ipynb**: Notebook for visualizing the results and model performance.

output

- **images/**: Directory containing the downloaded and processed images.
- **geography_aware_model.pth**: Saved model weights for the Geography-Aware model.
- **moco_model.pth**: Saved model weights for the MoCo model.
- **processed_images.csv**: CSV file containing metadata of processed images.
- **valid_images.csv**: CSV file containing metadata of valid images.

src

- **custom_dataset.py**: Defines the custom dataset class for loading images and their corresponding labels.
- **custom_model.py**: Contains the implementation of the MoCo model and other custom models.
- **deploy.py**: Script for deploying the trained models.
- **download_images.py**: Script for downloading images based on class names from the CSV file.
- **evaluate.py**: Script for evaluating the performance of trained models.

- **geo_model.py**: Script for training the ResNet model.
- **inspect_csv.py**: Script for inspecting the CSV file and extracting relevant data.
- **main.py**: Main script for executing the project workflow.
- **moco.py**: Implementation of the MoCo model.
- **mocov2.py**: Implementation of the MoCoV2 model.
- **model.py**: General model definitions and utilities.
- **preprocess_images.py**: Script for preprocessing images to ensure they are suitable for training.
- **train.py**: Script for training the MoCo model.

Data Preparation

Initial Dataset
Initially, we had the "classes_in_imagenet.csv" dataset which provided a list of image classes along with their corresponding geographical coordinates. The primary task was to extract the relevant data and prepare it for training the model.

Extracting Coordinates and Downloading Images
1. CSV Inspection:
--> We started by inspecting the "classes_in_imagenet.csv" file to understand the structure and content.
--> Extracted the columns containing the image class names and their respective geographical coordinates.

2. Downloading Images:
--> Developed a script ("download_images.py") to download images based on the extracted class names.

--> Utilized web scraping techniques to gather images from various sources.
--> Ensured that the downloaded images were stored in a structured directory ("output/images/").

3. Data Preprocessing:
--> Preprocessed the images using "preprocess_images.py" to ensure they were suitable for training.
--> Resized the images to a consistent size and applied necessary transformations.
--> Generated "processed_images.csv" and "valid_images.csv" to keep track of valid and processed images.

--> Model Implementation

MoCo (Momentum Contrast) Model
The primary focus was on implementing the MoCo model, a self-->supervised learning algorithm designed to learn visual representations by matching representations of different augmentations of the same image.

1. MoCo Model Definition:
--> Implemented the MoCo model in "moco.py".
--> Used ResNet as the base encoder for the MoCo model.
--> Defined the forward pass to compute query and key features.

2. Training the MoCo Model:
--> Developed a training script ("train.py") to train the MoCo model.
--> Utilized contrastive learning techniques to train the model on the preprocessed images.
--> Saved the trained model as "moco_model.pth".

ResNet Model

1. ResNet as Baseline:

--> Implemented a ResNet model in "geo_model.py" as a baseline for comparison.

--> Trained the ResNet model on the same dataset to evaluate its performance against the MoCo model.

2. Challenges with ResNet:

--> The ResNet model showed limitations in accurately predicting geographical coordinates.

--> Despite hyperparameter tuning, the performance did not meet expectations.

Transition to Random Forest Regressor

Given the suboptimal performance of both MoCo and ResNet models, we explored traditional regression models, specifically the Random Forest Regressor.

1. Random Forest Regressor Implementation:

--> Implemented the Random Forest Regressor in the notebook "model_evaluation.ipynb".

--> Trained the model on the features extracted from the MoCo model.

2. Achieving Optimal Results:

--> The Random Forest Regressor significantly outperformed the previous models.

--> Achieved an R-squared value of 0.8538, indicating robust performance in predicting geographical coordinates.

--> Evaluation

Initial Evaluation

1. MoCo and ResNet Models:

--> Evaluated the performance using standard classification metrics.

--> Encountered issues with metrics handling continuous-->multioutput and binary targets.

2. Challenges:

--> Faced various errors such as import errors, shape mismatches, and metric handling issues.

--> Debugged and resolved these errors by aligning the data shapes and converting targets to correct formats.

Regression Model Evaluation

1. Random Forest Regressor:

--> Evaluated the model using regression metrics such as Mean Squared Error (MSE) and R-->squared.

--> The Random Forest Regressor demonstrated a high R-->squared value, confirming its suitability for the task.

2. Visualization:

--> Plotted the actual vs. predicted values to visualize the model's performance.

--> The plot showed a strong correlation, validating the model's predictions.

--> Challenges and Resolutions

Import Errors

Issue: Difficulty in importing "CustomDataset" and "CustomModel".

Resolution: Ensured correct module paths and file structure.

Model Forward Method
Issue: Incorrect argument handling in "MoCo.forward".
Resolution: Adjusted the method to accept correct input parameters.

Shape Mismatch
Issue: Errors due to inconsistent shapes of input variables.
Resolution: Ensured correct alignment of features and labels by appropriate slicing.

Classification Metrics
Issue: Errors like "Classification metrics can't handle a mix of continuous multioutput and multiclass targets".
Resolution: Converted targets and predictions to correct formats for evaluation.

Regression Model Performance
Issue: Initial poor performance with ensemble models.
Resolution: Switched to the Random Forest Regressor, leading to a significant performance boost.

Future Enhancements

1. Fine-Tuning Hyperparameters:
    --> Further fine-tuning of hyperparameters for the Random Forest model to optimize performance.

2. Data Augmentation:
    --> Implementing advanced data augmentation techniques to enhance model robustness.

3. Hybrid Model:
--> Combining MoCo with other machine learning algorithms in a hybrid model to leverage the strengths of both.

4. Real-time Prediction:
--> Developing a real-time prediction system for geographical coordinates using the trained model.

5. UI Integration:
--> Creating a user-friendly interface to interact with the model and visualize predictions.

6. Deployment:
--> Deploying the model as a web service or mobile application for broader accessibility.

Conclusion

This project successfully implemented a Geography-Aware Self-Supervised Learning model, overcoming numerous challenges from the moco model and ultimately achieving high accuracy in geographical prediction using the Random Forest Regressor. The work done here lays a strong foundation for further enhancements and real-world applications. The journey involved extensive debugging, data preprocessing, and model evaluation, leading to valuable insights and a robust final model.

Running the Project

1. `cd Geo_Project`

2. **Set Up the Python Environment:**
```
python -m venv venv
```

3.
```
source venv/bin/activate  # On Windows, use
`venv\Scripts\activate`
```

4. **Install the Required Packages:**
```
pip install -r requirements.txt
```

5. **Prepare the Data:** Ensure `classes_in_imagenet.csv` is in the `data` directory. Run:
```
python src/download_images.py
```

6. **Preprocess the Images:**
```
python src/preprocess_images.py
```

7. **Train the MoCo Model:**
```
python src/train.py
```

8. **Evaluate the Trained Models:**
```
python src/evaluate.py
```

9. **Use Jupyter Notebooks for Further Analysis:**
```
jupyter notebook notebooks/model_evaluation.ipynb
```