Nafith

Nafith

Machine Learning Engineer

Machine LearningTask

by: Basel Mather

Nafith Task

Welcome to the ML Engineer Task Solution at Nafith Logistics International! 🚀

Table of Contents 📚

Introduction 📝

This repository contains the solution for a machine learning task provided by Nafith Logistics International. The task involves developing a Convolutional Neural Network (CNN) classifier from scratch or using a pre-trained model, training it on a provided dataset, and then applying it to classify objects in a video.

Getting Started 🏁

To get started with this project, follow these steps:

1. Clone the repository using `git clone <https://github.com/baselhusam/Nafith.git`>.
2. Install the required packages using `pip install -r requirements.txt`.
   Now you're ready to run the scripts!

How to Run the Scripts 🖥️

Here's how you can run each script:

- `read_and_split.py`: Run using `python3 read_and_split.py --img_size 224`. If you don't parse an argument for `img_size`, it will be 224 by default.
- `train.py`: Run using `python3 train.py --epochs 100 --learning_rate 0.001 --is_transfer_learning`. The number of epochs is 100 by default, and the learning rate is 0.001 by default. If `is_transfer_learning` is given as an argument, it will train the VGG16 pre-trained model; otherwise, it will train the custom model.
- `test.py`: Run using `python3 test.py --is_transfer_learning`. If `is_transfer_learning` is given as an argument, it will evaluate the VGG16 pre-trained model; otherwise, it will evaluate the custom model.
- `classifier.py`: Run using `python3 classifier.py --video_path demo.mkv`. The `video_path` argument is required for the path of the video for running inference on it.

All Together 🔥

```
# Clone the Repo
git clone <https://github.com/baselhusam/Nafith.git>

# Install the Requirements
pip install -r requirements.txt

# Run the read_and_split.py script
python3 read_and_split.py --img_size 224

# Run the `train.py` script for the custom model
python3 train.py

# Run the `train.py` script for the VGG16 model
python3 train.py --is_transfer_learning

# Run the `test.py` script for the custom model
python3 test.py

# Run the `test.py` scripts for the VGG16 model
python3 test.py --is_transfer_learning
```

```
# Run the `classifier.py` script
python3 classifier.py
```

Evaluation 📊

The evaluation metrics for both models are as follows

Model / Metric	Custom Model	VGG16
Train Set	---	--
Accuracy	0.77	0.9
Precision	0.77	0.89
Recall	0.79	0.89
F1-Score	0.76	0.89
Validation Set	---	---
Accuracy	0.69	0.88
Precision	0.70	0.88
Recall	0.71	0.88
F1-Score	0.69	0.88
Test Set	---	---
Accuracy	0.71	0.85
Precision	0.72	0.85
Recall	0.73	0.84
F1-Score	0.71	0.84

Google Drive Link for all the files 💾

Due to GitHub's file size limit of 100MB, not all files are included in this repository. The missing files can be found in this Google Drive link. The missing files include:

- `assets` Folder: This folder contains numpy arrays and saved history and weights for both models.

- `dataset` Folder: This folder is not included in Google Drive because it’s assumed that you already have it on your machine.

The Link for the Google Dirve is Here

Note: Make sure when you donwload the assets folder and unzip it to the project directory to have the following Folder Structure

```
├── assets
│   ├── Data
│   ├── Model
│   ├── Model_Trans
├── dataset
│   ├── car
│   ├── bus
│   ├── truck
|── Predictions
│   ├── car
│   ├── bus
│   ├── truck
├── read_and_split.py
├── train.py
├── test.py
├── classifier.py
├── utils.py
├── main.ipynb
├── requirements.txt
└── demo.mkv
```

FAQs ❓

If you have any questions about this project, please check out our FAQ section or feel free to open an issue.

More Details for the Solution

Approach

The approach to solving this task involves several steps:

1. **Data Preparation** (`read_and_split.py`): This script reads the images, stores them as numpy arrays, splits them into training, testing, and validation sets, and finally saves these numpy arrays.
2. **Model Training** (`train.py`): This script trains a custom CNN model from scratch or uses transfer learning (VGG16) to classify different types of vehicles (car, bus, and truck).
3. **Model Evaluation** (`test.py`): This script evaluates the trained model (either the custom model or the pre-trained VGG16 model). It displays the confusion matrix for the training, validation, and testing sets and saves the images. It also calculates the Accuracy, Precision, Recall, and F1-Score for these sets.
4. **Classification** (`classifier.py`): This script takes the trained model and applies it to count objects from a video. It displays the count for each class, the frames per second (FPS), the majority prediction for every object, and the prediction for the opened frame.

All these scripts are built using classes defined in `utils.py`, which contains all their functionalities.

Files and Folders

The solution includes several files and folders:

- `read_and_split.py`, `train.py`, `test.py`, `classifier.py`, `utils.py`: These are Python scripts that perform various tasks as described above.
- `main.ipynb`: This Jupyter notebook compares the custom model with the VGG16 pre-trained model using flow_from_directory and X and Y Numpy Arrays approaches. It also compares hard disk size (in MB) and time taken to build the models (in seconds).
- `requirements.txt`: This file lists all required Python packages needed to run the solution.
- `Predictions` Folder: Contains saved images from predictions for each object within its supposed folder.
- `assets` Folder: Contains three main sub-folders:

1. `Data`: Contains numpy arrays.
2. `Model`: Contains saved history and weights for the custom model.
3. `Model_Trans`: Contains saved history and weights for the VGG16 pre-trained model.

Dataset

The dataset provided consists of 2531 images belonging to three different classes: car (1087 images), bus (866 images), and truck (578 images).

Python Version

The solution is implemented in Python 3.11.4.

Future Enhancements

In future iterations of this project, we could consider implementing additional features such as real-time object detection and tracking, improving classification accuracy through advanced techniques like data augmentation or fine-tuning, or expanding the solution to handle more classes of objects.