AICUP - Project 32

Vehicle Tracking Competition

Amina Enkhbayar - 110021139
Manuel Salgueiro – 110021200

Deep Learning - 深度學習 3C

Introductions and Goals

This semester we've been introduced to the AICUP, a competition that offers different Deep Learning tasks to which participants can join. There was a variety of projects from which to choose from, such as object detection training and testing, also multi-tracking of vehicles among others.

The aim for the final project of our Deep Learning course and these tasks was to successfully submit the results of the training and testing to the competition's website and get the better score against other participants.

In our case, we chose to work on project 32 : ***AI drives the future of mobility: cross-camera multi-objective vehicle tracking competition - Model Group***.

In order to achieve this task, we were presented with a baseline jupyter notebook project by ricky-696, which we took as a template to work with. We will describe our observations and experience with the tools presented in this template, as well as explain the changes made to the original pipeline and our experience when executing it.

Table of Contents

Libraries installed

The Pycocotools package includes the tools in the form of an API accesible for python applications, originally intended for loading, parsing and visualizing annotation on the famous large image dataset COCO. The API is contained in the python file coco.py.

Cython is installed at the same time. Cython_bbox is a Python library designed to accelerate bounding box computations commonly used in object detection tasks. It leverages Cython, a superset of Python that allows compiling Python code into efficient C extensions.

Faiss-cpu and faiss-gpu are libraries that provide implementations of the FAISS (Facebook AI Similarity Search) library for efficient similarity search.

ONNX (Open Neural Network eXchange) isn't actually a library in the traditional sense. It defines a common set of operators (the building blocks of machine learning models) and a file format. Once a model is in the ONNX format, it can be used by various tools and runtime environments that support ONNX, so one is not limited by the original format. Onnxruntime would be the runtime engine for running models in the ONNX format.

Having the TensorFlow libraries already installed in the environment, we install TensorRT, a runtime engine that optimizes pre-trained models for NVIDIA GPUs by removing unnecessary computations, converting the model to a format designed for NVIDIA GPUs and leveraging hardware acceleration features.

The requirements.txt file install several libraries we all use in our usual Data mining projects, plus some specifics like Torchvision, a library specifically made for Computer Vision tasks.

Datasets

AICUP (train) dataset

This dataset is separated in the train and test datasets. The train dataset is composed by the images but also labels, which consists in .txt files that match the images' name. The label contains the coordinates and dimensions of the elements within the images:

class	center_x	center_y	width	height	track_ID
0	0.704687	0.367592	0.032291	0.1	1

The train dataset is downloaded and must be located at our project's root directory as /train.

The test dataset isn't actually used in this pipeline, instead we'll see how the file generate_AICUP_patches.py will generate it.

ReID dataset

ReID stands for Re-Identification, it seems to be a naming standard for person identification datasets, but apparently it's just the AICUP dataset adapted for a ReID format that fast_reid can work with. This conversion happens via /fast_reid/datasets/generate_AICUP_patches.py on the AICUP train dataset:

- The ratio for the the train set is 0.8 and 0.2 for the test set.
- It converts the images' size to 720 x 1280 under .bmp (Bitmap) loss-less format. It's also easy to read and write for all OS.
- It patches the labels onto the new .bmp images.
- The patched files are then written onto:
  - /content/AICUP_Manny_32/fast_reid/datasets/AICUP-ReID/bounding_box_train

- /content/AICUP_Manny_32/fast_reid/datasets/AICUP-ReID/bounding_box_test

The images within these two sets are stored under the same directory, with no inner forkings, with a new resolution and file format. This is the structure and file format fast_reid framework will work with.

YoloV7 dataset

The YoloV7 dataset is also the **AICUP dataset** adapted for the **yolo** format that **YoloV7** can work with. This script will generate new images separated in a new train and validation datasets. This conversion happens via /yolov7/tools/**AICUP_to_YOLOv7.py** on the AICUP train dataset:

- The **ratio** for the the train set is **0.8** and **0.2** for the validation set.
- There's no apparent change in the size of images.
- The images are **renamed** according to its corresponding **/label** text file name.

The purpose of the script and the new dataset is to split the AICUP dataset into a train and validation dataset, keeping the same **/images** and **/labels** structure. It would look like this is the structure and naming conventions YoloV7 will work with.

Deep Learning Frameworks and Models

Fast_ReID Person ID Framework

A framework designed for Person Re-Identification algorithms. It offers pre-trained models and tools ready to use in other projects. These models aren't trained relying on a single model architecture, but offers instead implementations of **popular architectures** specialized in person identification:

- **ResNet**: Convolutional Neural Network.
- **OSNet**: Specialized in handling small variation detection.
- **HRNet**: Mantains high-resolution features for better identification.

The pool of available models within the framework is called **Model Zoo**. These models are trained using the architectures mentioned before, along with Re-ID algorithms like:

- **Triplet Loss**: A loss function that encourages the model to embed images of the same person together in the feature space while pushes the ones of different people apart.
- **SphereNet**: Learns discriminative features for person Re-ID.
- **Margin Loss**: Loss function that creates a larger margin between a person's features and those of different people.

In conclusion, Fast-ReID allows you to choose models, that implement different architectures, algorithms and loss functions, to train them on your own datasets. These "**baselines**" are available for download at https://github.com/JDAI-CV/fast-reid/blob/master/MODEL_ZOO.md. Sample from Github:

Method	Pretrained	Rank@1	mAP	mINP	download
BoT(R50)	ImageNet	94.4%	86.1%	59.4%	model
BoT(R50-ibn)	ImageNet	94.9%	87.6%	64.1%	model
BoT(S50)	ImageNet	95.2%	88.7%	66.9%	model
BoT(R101-ibn)	ImageNet	95.4%	88.9%	67.4%	model

Implementation of Fast_ReID

It would seem the Fast-ReID utilities set for this project, despite executed, can't execute successfully, according to the result of the specific cell in the original pipeline, never training past epoch 2, error which is documented in the READ_ME file of the baseline project:

```
...
File "./fast_reid/fastreid/evaluation/reid_evaluation.py", line 107, in evaluate
    cmc, all_AP, all_INP = evaluate_rank(dist, query_pids, gallery_pids, query_camids, gallery_
  File "./fast_reid/fastreid/evaluation/rank.py", line 198, in evaluate_rank
    return evaluate_cy(distmat, q_pids, g_pids, q_camids, g_camids, max_rank, use_metric_cuhk0
  File "rank_cy.pyx", line 20, in rank_cy.evaluate_cy
  File "rank_cy.pyx", line 28, in rank_cy.evaluate_cy
  File "rank_cy.pyx", line 240, in rank_cy.eval_market1501_cy
AssertionError: Error: all query identities do not appear in gallery
```

Figure 1: The lack of two datasets "query" and "gallery" required for evaluation seem to stop the process

This discontinuation is later observed in further steps (Tracking and Creating the submission file for AICUP), as the results of the FastReID training are never dealt with within the code.

YOLOv7

YOLOv7 (You Only Look Once) is a real-time object detection model. It's the 7th iteration of previous YOLO versions. The last version available is YOLOv10, released in 2024.

It can effectively reduce about 40% of parameters, 50% of computation and requires cheaper hardware, as opposed to other computer vision neural networks, and can be trained much faster on small datasets without pre-trained weights. This efficiency is suitable for applications where both real-time performance and accurate object detection is important. The model surpasses all known object detectors in both accuracy and speed in the the range from 5 FPS to 160 FPS, having the highest accuracy of 56.8% AP (Average Precision) at 30+ FPS on GPU V100.

The model relies on CNN (Convolutional Neural Network) for extracting features from images, object localization and measurement of objects within the image, and object classification. It also implements techniques that don't involve CNNs, such as ELAN (Efficient Layer Aggregation Network), focused on optimizing information flow through the network during feature extraction, with the main goal of ensuring deeper networks learn and converge more effectively during training.

The ELAN approach

Gradients are information calculated during backpropagation that deepens the detail handled by Neural Networks. They indicate how much each weight needs to be adjusted to minimize the error between the network's prediction and the actual output. The CNNs may suffer from vanishing gradients as weights propagate through many layers, and specially in deeper layers. Some algorithms, like the ResNet family, try to prevent the gradient loss by introducing a bias that propagates through layers. ELAN tackles this issue by strategically controlling the path lengths of gradients within the network.

- Shuffling: rearranges the feature channels within a layer, preventing gradients from getting stuck in specific channels
- Channel Splitting and Concatenation: it splits inout channels into multiple branches with fewer branches, processing data separately and recombining the features using concatenation, favoring gradient flow and feature diversity.
- Efficient Transition Blocks: these blocks ensure efficiency for proper information flow and resolution changes when transitioning different stages of the network.

YOLOv7 stages

The model follows a set of stages to process the images and identify objects within it:

1. Backbone feature extraction:
   - Images are fed to YOLOv7's backbone network, a CNN architecture.
   - This backbone network **extracts hierarchical features** from the image. These features capture various levels of detail, like edges, textures, shapes and parts.

2. Neck (Feature Fusion):
   - This component takes the feature maps extracted in the previous step.
   - It combines and merges these features from various resolutions, trying to **create new features** that better represent objects, therefore **easier to detect** during object detection.
   - To match the spatial dimensions of different resolutions' features, it uses Up-sampling techniques.

3. Head (Prediction):
   - Takes on the fused feature maps from the Neck stage.
   - Via CNN, it makes two crucial **predictions for each potential object** in the image:
     - **Bounding Box Prediciton**: predicts location and size of the box **tagging the object**.
     - **Class Prediction**: this step **assigns the class label** to the detected object within the tag.

Implementation of YOLOv7

The values for each convolutional layer's parameters, splitting, concatenation and up-sampling, in the case of given pipeline are located in the yaml file /yolov7/cfg/training/**yolov7-AICUP.yaml**:

```
# yolov7 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [32, 3, 1]],  # 0

   [-1, 1, Conv, [64, 3, 2]],  # 1-P1/2
   [-1, 1, Conv, [64, 3, 1]],

   [-1, 1, Conv, [128, 3, 2]],  # 3-P2/4
   [-1, 1, Conv, [64, 1, 1]],
   [-2, 1, Conv, [64, 1, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1]],  # 11
```

Figure 2: Example of Convolutional Layering in YOLOv7

Implementation of the Pipeline

Initial Setup with Conda and VS Code

For the initial implementation phase, we used Conda to create a virtual environment and download the libraries and packages required and used VS Code to edit the files as necessary. However, downloading dependencies became quite a hurdle, with some libraries like faiss-gpu, cython_bbox, and onnxruntime presenting unique challenges that required individual troubleshooting. The process was time-consuming and inefficient, leading us to reconsider our approach, especially as the alternative was readily available and better equipped for training machine learning models.

Switching to Google Colab

Google Colab offered a more streamlined environment for handling model training jobs. Providing the computational resources necessary for our model, but likewise with using our personal machines, this setup was not without its own set of challenges. We encountered issues related to memory limitations and with slow training processes as well, as a result of us having to adapt to those limitations. Additionally, unsaved changes could be lost upon session expiration, adding another burden to our workflow, that we would have to account for.

Errors and Solutions

There are two different frameworks and configuration setups that train a machine learning model.
For FastReID , errors arose such as *`ImportError: cannot import name 'Mapping' from 'collections'`*, which is due to changes in Python 3.10, where `Mapping` was moved from *`collections`* to *`collections.abc`*.
This affected multiple files, including *`train_net.py`*, *`__init__.py`*, *`hooks.py`*, *`evaluation/__init__.py`*, and *`evaluation/testing.py`*.
Another notable error was the *`ModuleNotFoundError: No module named 'torch._six'`*, resulting from the removal of the *`torch._six`* module in recent PyTorch versions. The solution involved replacing it with *`string_classes = str`*.
Whatever error we encounter in the code, they all arose from incongruencies between past and present Python versions or outdated libraries.

This continues with YOLOv7. An *`AttributeError: module 'numpy' has no attribute 'int'`* surfaced due to the deprecation of *`np.int`* in NumPy version 1.20. The solution was to replace *`np.int`* with *`int`*. Other errors included *`NameError: name 'int16' is not defined`* and *`NameError: name 'interp' is not defined`*, both of which required appropriate replacements and imports. These issues were addressed in files such as *`datasets.py`* and *`general.py`*. Additionally, updating the *`AICUP.yaml`* config file was necessary due to structural issues in the dataset that prevented the model from locating it correctly.

Memory and Training Time Constraints

One of the most persistent obstacles was running out of memory while using Torch CUDA. This limitation forced us to use smaller batch sizes, which, in turn, prolonged the training time significantly. The time constraints further compounded these issues, leaving limited opportunities to fix errors, re-train, and re-try.
We initially attempted to use FastReID, but it quickly became apparent that the training time was prohibitive. For instance, training for 20 epochs took approximately 12 hours, and the process ultimately ended in an error (*`AssertionError: Error: all query identities do not appear in gallery`*). For YOLOv7, 10 epochs took around 6.7 hours, and the submitted results were rejected due to an error on our part. Despite our efforts to correct these errors, the extensive training times, coupled with the memory constraints made it impossible to submit revised results within the given timeframe.

As recurring issues would crop up. New errors frequently emerged, often requiring substantial debugging and adjustment. Additionally, as we've mentioned before, the nature of Google Colab sessions posed a challenge, as they would expire and potentially result in the loss of unsaved work, or we would be out of memory, again.

Final Results

Despite our efforts, the final submission was not accepted due to a mistake on our part. In contrast, a trial submission example we uploaded purely to test the submission process was accepted and scored. This file, however, was not our own creation but a sample used to understand the submission mechanics.

Conclusions

Our journey through this project was marked by numerous technical challenges. We faced significant obstacles with dependency management, code inconsistencies, and memory limitations that extended our training times and introduced frequent errors. Despite these obstacles, the process provided valuable insights into the intricacies of model training and the critical importance of thorough preparation and testing.

In the end, while we were unable to successfully submit our final results, the experience underscored the importance of adaptability and persistence in the face of technical challenges. Our trial submission, though not of our own creation, provided a benchmark for understanding the submission process and highlighted areas for improvement in future projects.
Placing 82nd out of 101 final entries, we recognized the importance of comprehensive error handling and resource management in achieving successful model training and deployment.
Despite the setbacks, the experience has been an invaluable learning opportunity, providing insights that will undoubtedly inform our future endeavours in machine learning and model training.

Bibliography

COCO Dataset

http://cocodataset.org/

Fast-ReID Framework

https://github.com/JDAI-CV/fast-reid
https://github.com/JDAI-CV/fast-reid/blob/master/MODEL_ZOO.md

YOLOv7

https://viso.ai/deep-learning/yolov7-guide/
https://arxiv.org/pdf/2207.02696