DL最終課題_レポート

氏名：出水小春
所属：東京大学 総合文化研究科 生命環境系 M1

課題内容：実装での工夫点を書く。
GitHub参照リンク: https://github.com/small-spring/dl_competition.git

提出したコードについて

.devcontainer フォルダでDockerの環境構築

- dockerfile
  - NVIDIAが提供するDocker imageをベースにした。
  - 基盤的なソフトを`apt-install` したのち、 `requirements.txt`をコピーして`pip`でインストールする。
- docker-compose.yml
  - 環境変数 `PYTHONPATH=/workspace:/home`と指定することで、パッケージを読むように設定。
- devcontainer.json
  - VSCodeの必要な拡張機能をロードするように設定
  環境
- Windows 11 Home Edition,
- CPU RAM: 32GB,
- GPU: NVIDIA GeForce RTX 4060 Ti 8GB
- 全GPUとwslconfigによりCPU RAMを20GB割当。swapはなし。

yamhit氏のコードを参考にした変更

- https://github.com/yamhit/dl_lecture_competition_pub/tree/event-camera-competition
- バッチのシャッフルをtrueに
- schedulerの導入
  - `initial_learning_rate`を0.01から0.001に
  - `torch.optim.lr_scheduler.StepLR` link
    - 学習率を3エポックごとにgamma=0.5倍に
    - 最終的に10epochs回したので、0.001→0.0005→0.00025
- 損失関数の変更

    - 重み0.1 に `smoothness_weight` を追加。予測したflowの変化が小さいと損失が小さくなるように設定。
- epochごとにモデルを保存
    - 手元で動作が不安定で、epochの途中で止まってしまう動作が多く入ったが、bestなモデルを保存しておくことでスムーズに再開できた。
- （自分が改変した部分）途中で止まるなど動作が不安定だったので、batchごとにlossが小さいものを選んで保存するようにした。

提出したコードには反映できなかった、やりたかったこと

教師なし・コントラスト最大化によるOptical Flow推定手法の適用

- 以下の論文で述べられている、eventだけからoptical flowを推定する教師なしの手法を適応しようとしたが、間に合わず、途中で挫折した。（できればここに、何かの深層学習手法を組み合わせるつもりだった。）
    - Shiba, S., Klose, Y., Aoki, Y. & Gallego, G. Secrets of event-based optical flow, depth and ego-motion estimation by Contrast Maximization. *IEEE Trans. Pattern Anal. Mach. Intell.* **PP**, 1–18 (2024).
    - 環境に対して視点が動くような映像であると仮定し、輝点のワープを推定する手法。
- この手法の特徴は、コントラスト最大化において損失関数が不適切な結果に収束してしまうのを防ぐ点である」
    - 損失関数を、複数の時点を基点として計算する(Multi-reference focus loss; 多時刻参照ワープ)
    - 速度（flow）が位置依存的だけではなく、時間依存でも変化すると仮定を変更し、occlusion（遮蔽）を乗り越えられるようにした。
    - 画面を複数通りに分割することで、大域的なoptical flow → 局所的なoptical flowの順に推定する(Multi-scale Approach)
- 公開されているGitHub上のコードでは、requirementが不完全であったため、書き出し、Dockerで環境構築した。基本となる実行はでき、論文の図を出力した。他、コードをある程度読み、DSECデータセットを適応させようとしたところで時間が尽きた。
- コード: https://github.com/small-spring/event_based_optical_flow （不完全であるため非公開リポジトリ）

MEGをローカルでやろうとした

- しばらく取り組んだが、メモリが慢性的に不足し、安定して動作しなかったのでEvent-Cameraに切り替えた。
- swapを割り当てると非常に重くなった。

反省

- 前提の構築に多大に時間を割き、しかもやろうとしたことの難易度が自分の実力にあっていなかった。
- まずepochごとにbest modelが保存され、途中で学習が止まっても再開できる体制を整え、シンプルなモデルである程度回してベンチマークを取るべきだった。（これを知るよい経験にはなった。）