Description

---

1. Descriere și scurtă EDA a datelor

Descrierea Datelor

Setul de date este conceput pentru o sarcină de **Visual Question Answering (VQA)**. Datele sunt stocate în format **JSON Lines (.jsonl)**, unde fiecare linie reprezintă un singru exemplu de antrenament/validare.

Din clasa `VQADataset`, putem deduce structura fiecărui obiect JSON:

- `'image_path'`: O cale (string) către fișierul imagine de pe disc.
- `'question'`: Întrebarea (string) referitoare la imagine.
- `'answer'`: Răspunsul corect (string) la întrebare.

Scurtă Analiză Exploratorie a Datelor (EDA)

Codul nu efectuează o EDA complexă (cum ar fi analiza distribuției lungimii răspunsurilor sau vizualizarea imaginilor). Totuși, implementează un pas esențial de **curățare și validare a datelor** în timpul încărcării (`VQADataset.__init__`):

1. **Verificarea Existenței Fișierelor:** Scriptul verifică dacă imaginea specificată de `image_path` există fizic pe disc folosind `os.path.exists()`.
2. **Omiterea Datelor Lipsă:** Dacă o imagine lipsește, intrarea respectivă este omisă din setul de date, iar un contor (`skipped`) înregistrează și raportează numărul total de exemple care nu au putut fi încărcate.
3. **Gestionarea Erorilor la Citire:** În `__getitem__`, există un bloc `try...except` care prinde erorile ce pot apărea la deschiderea unei imagini (de ex., fișier corupt).

---

2. Descriere Algoritm Inteligent

Algoritmul Ales

Algoritmul principal este **Moondream2** (`MODEL_ID = "vikhyatk/moondream2"`). Acesta este un model de limbaj vizual (Vision Language Model - VLM) pre-antrenat, de mici dimensiuni, capabil să răspundă la întrebări despre imagini.

Este încărcat folosind biblioteca `transformers` de la Hugging Face.

Motivația Alegerii

1. **Fine-Tuning (Ajustare Fină):** Scriptul nu antrenează un model de la zero. În schimb, aplică **fine-tuning** pe un model puternic, deja pre-antrenat (Moondream2), pentru a-l specializa pe un set de date specific (sugerat de `OUTPUT_DIR` ca fiind legat de "crosswalks" - treceri de pietoni). Această abordare este mult mai eficientă din punct de vedere computațional.
2. **Eficiența Parametrilor (PEFT):** Se aplică o strategie specifică de antrenament: **encoder-ul vizual este înghețat** (`model.vision_encoder.requires_grad_(False)`). Doar parametrii modelului de limbaj (`model.text_model`) sunt actualizați. Acest lucru reduce drastic numărul de parametri antrenabili (afișați în consolă ca **1,419,333,637** din totalul de **1,867,982,709**), scăzând cerințele de memorie (VRAM) și accelerând antrenamentul.
3. **Model Causal:** Modelul este încărcat ca `AutoModelForCausalLM` , ceea ce înseamnă că este antrenat să prezică următorul token (cuvânt/sub-cuvânt) dintr-o secvență.

Librăriile Folosite

- `torch` **(PyTorch):** Biblioteca de bază pentru deep learning, folosită pentru definirea modelelor, calcularea pierderii (loss) și optimizare.
- `transformers` **(Hugging Face):** Folosită pentru a descărca și utiliza modelul pre-antrenat `Moondream2` (`AutoModelForCausalLM`) și tokenizer-ul asociat (`AutoTokenizer`).
- `PIL` **(Pillow):** Folosită pentru a încărca și procesa imaginile (`Image.open`).
- `json` și `os` **:** Utilitare standard Python pentru citirea fișierelor `.jsonl` și verificarea căilor către fișiere.

---

3. Descriere Metodologie Experimentală și Rezultate

Împărțirea Setului de Date

Setul de date este **deja împărțit** în două fișiere distincte, definite de constantele:

- **Set de Antrenament (Train):** `TRAIN_FILE = "train.jsonl"`
- **Set de Validare (Validation):** `VALID_FILE = "valid.jsonl"`

Scriptul *nu* creează un set de testare (`test`) separat; folosește setul de validare pentru a monitoriza performanța modelului după fiecare epocă de antrenament.

Metricile Folosite pentru Evaluare

Metrica principală utilizată atât pentru antrenament, cât și pentru validare, este **Cross-Entropy Loss** (`torch.nn.CrossEntropyLoss`).

Un detaliu metodologic crucial (implementat în bucla de antrenament și validare) este **masking-ul (mascarea) pierderii**:

1. Modelul primește ca intrare atât imaginea, cât și textul formatat: `"\n\nQuestion: [ÎNTREBARE]\n\nAnswer: [RĂSPUNS]"` .
2. Pentru a forța modelul să învețe doar să genereze răspunsul, pierderea (loss) este calculată **exclusiv pe token-urile care corespund răspunsului (`[RĂSPUNS]`)**.
3. Token-urile întrebării (`[ÎNTREBARE]`) și ale prompt-ului sunt ignorate în calculul pierderii prin setarea etichetelor (labels) corespunzătoare la `-100` , valoare ignorată de `CrossEntropyLoss` .

Hiperparametrii Folosiți

Următorii hiperparametri sunt definiți explicit în script (și confirmați de rularea a doua):

- **Model:** `vikhyatk/moondream2`
- **Număr de Epoci:** `EPOCHS = 5`
- **Rata de Învățare (Learning Rate):** `LEARNING_RATE = 3e-5`
- **Dimensiunea Lotului (Batch Size):** `BATCH_SIZE = 2`

- **Optimizator:** `AdamW` (aplicat doar pe parametrii antrenabili).
- **Gradient Clipping:** `max_norm=1.0` (folosit pentru a preveni "explozia" gradienților și a stabiliza antrenamentul).
- **Precizie:** `torch.bfloat16` (folosită la încărcarea modelului pentru a reduce consumul de memorie).

Rezultate Obținute

Scriptul este conceput să ruleze antrenamentul și să raporteze rezultatele în consolă.

- **Rezultate Cantitative:** După fiecare epocă, scriptul afișează **media pierderii de antrenament** (`Avg Loss` pentru training) și **media pierderii de validare** (`Avg Loss` pentru validation). Acestea sunt singurele metrici de performanță calculate. (Vezi analiza detaliată în secțiunea 4).
- **Rezultate Calitative:** Scriptul **nu** include o secțiune de inferență sau testare calitativă. Prin urmare, **nu generează exemple de răspunsuri corecte sau greșite** prezise de modelul ajustat.

Produsul final (rezultatul) principal al rulării acestui script este **modelul fine-tunat** (doar componentele textuale, deoarece cele vizuale au fost înghețate), salvat în directorul `OUTPUT_DIR` (`./moondream_crosswalks_finetuned`).

---

4. Analiza Log-urilor de Antrenament (Datele Furnizate)

Log-urile din cele două rulări (în special a doua, completă, de 5 epoci) oferă o imagine clară a performanței modelului și scot la iveală un fenomen important: **overfitting-ul**.

Comparația Loss Antrenament vs. Validare (Rularea 2)

Să analizăm progresia pierderii (loss) de-a lungul celor 5 epoci din a doua rulare:

Epoca	Training Avg Loss	Validation Avg Loss	Observație
1	1.9113	1.7609	Ambele scad, modelul învață.
2	1.3526	**1.7252**	**Punctul optim.** Loss-ul de validare este cel mai mic.
3	0.9645	1.7986	**Overfitting.** Loss-ul de antrenament scade, dar cel de validare crește.
4	0.6009	1.9750	Overfitting-ul se accentuează.
5	0.3540	2.0121	Modelul este puternic supra-ajustat.

Interpretarea Datelor

1. **Loss-ul de Antrenament (Training Loss):** Această valoare **scade constant** de la `1.91` la `0.35`. Acest lucru este de așteptat și indică faptul că modelul învață cu succes să potrivească datele din setul de antrenament (`train.jsonl`).
2. **Loss-ul de Validare (Validation Loss):** Această valoare este crucială deoarece măsoară performanța modelului pe date "noi" (pe care nu s-a antrenat).
   - Scade de la Epoca 1 la Epoca 2 (de la `1.7609` la `1.7252`). Acesta este momentul în care modelul a atins cea mai bună capacitate de generalizare.
   - **După Epoca 2, loss-ul de validare crește constant** (la `1.79` , `1.97` și `2.01`).

Concluzie: Overfitting (Supra-ajustare)

Datele arată un caz clasic de **overfitting** (supra-ajustare) începând cu Epoca 3.

- **Ce se întâmplă?** Modelul devine atât de bun la "memorat" exemplele de antrenament, încât începe să învețe și zgomotul sau particularitățile acelor date. Când este expus la date noi (setul de validare), performanța sa scade, deoarece "regulile" pe care le-a învățat sunt prea specifice și nu se generalizează bine.