Meteval (MetSnap): Technická dokumentace

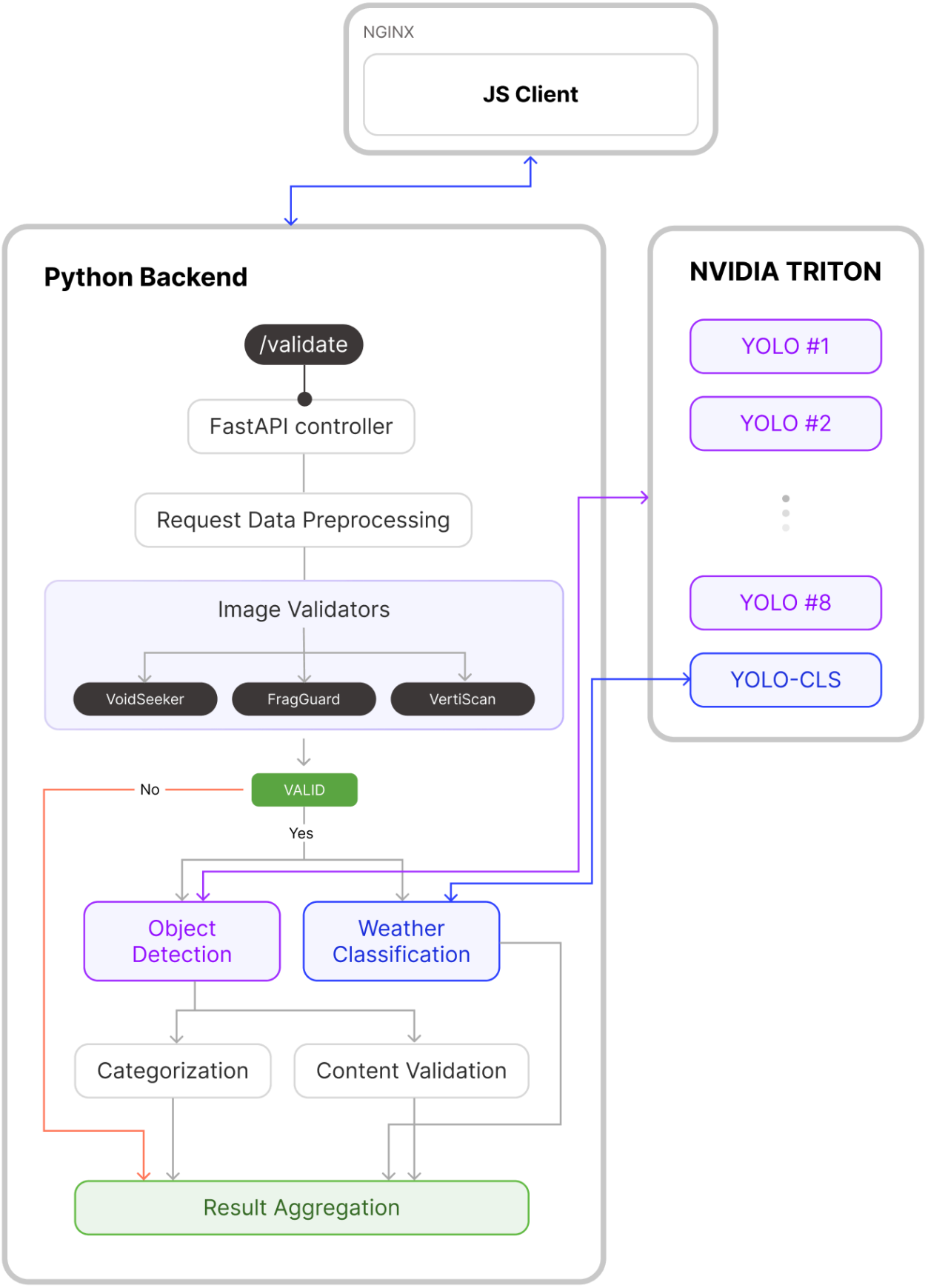
Meteval (MetSnap) je aplikace skládající se z objektově orientované backendové aplikace napsané v Pythonu, klientské aplikace pro zobrazování výsledků a interakce s uživateli psané v JavaScriptu a NVIDIA Triton Inference serveru pro hostování modelů strojového učení. Je navržena pro demonstrativní účely navržené analýzy a validace obrázků pomocí technik strojového učení v rámci bakalářské práce. Tato dokumentace poskytuje přehled o architektuře aplikace, jejích hlavních modulech a o tom, jak tyto moduly spolupracují mezi sebou, aby dosáhly požadované funkcionality.

Struktura projektu

Struktura adresářů:

- `src/`: Hlavní složka se zdrojovými kódy aplikace, včetně modulů pro API, zpracování dat, klasifikaci a validaci.
- `data_types/`: Definice datových typů použitých v projektu.
- `dev/`: Skripty a nástroje pro vývoj, testování a trénink modelů.
- `docker/`: Docker skripty a konfigurace pro deployment aplikace.
- `client/`: Klientská aplikace pro interakci s uživateli a vizualizaci výsledků.
- `nginx/`: Konfigurace pro Nginx, který může sloužit jako reverse proxy server pro aplikaci.
- `triton/`: Složka obsahující složku s repozitářem modelů pro NVIDIA Triton Inference Server včetně `triton_client.py` modulu obsahující třídu pro komunikaci se serverem. Obsahuje také spouštěcí soubor spolu s Dockerfile předpisem kontejneru, ve kterém běží.
- `train_cfgs/`: Obsahuje konfigurační soubory pro tréninky modelů
- `assets/`: Složka s dvěma testovacíma obrázkama

Architektura celé aplikace



Hlavní backendová python aplikace

Klíčové soubory:

- `main.py`: Hlavní spouštěcí skript pro FastAPI server.
- `Dockerfile` a `docker-compose.yml`: Docker konfigurace pro sestavení a spuštění kontejnerů aplikace.
- `train.py`: Skript pro trénink modelů.
- `settings.toml`: Konfigurační soubor pro nastavení parametrů aplikace.

Moduly a komponenty:

- `src/api/`: Obsahuje komponenty pro zpracování API požadavků.
    - `api_call_processor.py`: Abstraktní třída, poskytující základní strukturu pro zpracování API volání.
    - `api_call_img_processor.py`: Modul pro zpracování dat přijatých přes API, který integruje metody pro detekci a klasifikaci a vrací odpověď ve formě obrázku s vyznačenými detekovanými objekty a výsledky validace a klasifikace v hlavičce.
    - `api_call_json_processor.py`: Modul pro zpracování dat přijatých přes API, který integruje metody pro detekci a klasifikaci a vrací odpověď ve formě JSON objektu.
- `src/data_preprocessors/`:
    - `yolo_data_preprocessor.py`: Příprava dat pro trénink YOLOv8 detektoru, včetně normalizace a augmentace obrazu.
- `src/image_checkers/`:
    - `image_checker.py`: Třída určená k ověření integrity a formátu obrázkových souborů.
- `src/image_classifiers/`: Obsahuje třídy pro klasifikaci obrázků.
    - `base_image_classifier.py`: Základní třída pro všechny klasifikační modely.
    - `categorizer/`: Moduly specificky zaměřené na kategorizaci obsahu obrázků.
        - `image_categorization_postrocessor.py`: Zpracovává výstupy klasifikačního modelu.
        - `image_categorizer.py`: Používá extrahované objekty z detekčních YOLOv8 modelů a predikuje kategorii scenérie pomocí RMLP modelu.
    - `classifiers/`: Obsahuje specifické klasifikační modely pro různé účely.
        - `binary`, `multilabel`, `singlelabel`: Klasifikace dle různých kategorií a typů podle konfiguračních souborů.
- `datasets/`: Třídy pro načítání a přípravu datových sad pro trénink a validaci modelů.

- `src/image_content_recogniser/`:
  - `image_content_recogniser.py`: Hlavní třída pro rozpoznávání obsahu v obrázcích, koordinuje detekci objektů, klasifikaci a validaci.
- `src/image_properties_validators/`: Moduly pro validaci poškození obrázků.
  - `image_properties_validator.py`: Koordinuje validátory poškození obrázků.
  - `validators/`: Konkrétní implementace validátorů.
    - `no_image_validator.py`: Detekuje obrázky s nízkým obsahem informací (např. jednobarevné plochy) pomocí algoritmu VoidSeeker.
    - `partial_download_validator.py`: Kontroluje, zda obrázek nebyl pouze částečně stažen pomocí algoritmu FragGuard.
    - `vertical_corruption_validator.py`: Hledá vertikální poškození v obrázcích pomocí algoritmu VertiScan.
- `src/object_detection/`: Třídy pro detekci objektů.
  - `image_object_detector.py`: Abstraktní základ pro detektory objektů.
  - `image_object_detector_api_triton.py`: Implementace detektoru s využitím NVIDIA Triton Inference Serveru.
  - `image_object_detector_local.py`: Implementace detektoru objektů bez potřeby externích služeb.
  - `image_object_detection_postrocessor.py`: Zpracování detekovaných objektů, úpravy a filtrace výstupů.
- `src/profiler.py`:
  - `profiler.py`: Nástroj pro měření a zaznamenávání výkonnostních metrik během operací aplikace.
- `src/settings.py`:
  - `settings.py`: Centrální konfigurační soubor pro nastavení parametrů a preferencí systému.
- `src/utils.py`:
  - `utils.py`: Pomocné funkce pro běžné úlohy napříč aplikací, jako je manipulace s daty nebo logování.

API koncové body

Aplikace definuje následující čtyři API koncové body:

- `/validator/detect-img`: Zpracovává vstupní obrázky a vrací zpracovaný obrázek s informacemi o validaci a klasifikaci v hlavičce odpovědi. Je řízen třídou `Api_Call_Img_Processor`.

- `/validator/detect-json`: Zpracovává vstupní obrázky a vrací JSON objekt obsahující informace o validaci a klasifikace. Je řízen třídou `Api_Call_Json_Processor`.

- `/healthz`: Používá se v prostředích Kubernetes k monitorování zdraví aplikace, vrací HTTP statusový kód 200, pokud je aplikace funkční, a 500 v opačném případě.

FastAPI automaticky generuje dokumentaci API SwaggerUI, která je přístupná na cestě `/validator/routesz`.

Průchod aplikací

1. Příjem dat prostřednictvím API
    - Uživatelský požadavek je přijat API endpointem API Controlleru obsahujícím FastAPI router.

2. Zpracování dat
    - Rozpoznání datového typu: Na základě typu endpointu a přijatých dat jsou data předána odpovídající zpracovatelské třídě v `src/api`.
    - Pro výsledky ve formě obrázku: `api_call_img_processor.py`
    - Pro výsledky ve formě JSON objektu: `api_call_json_processor.py`
    - Následuje předzpracování dat, při kterém jsou obrázky normalizovány a připraveny pro detekci

3. Detekce a klasifikace
    - Probíhá ve třídě `ImageContentRecogniser`
    - Validace poškození obrázku probíhá pomocí speciální validátorů `no_image_validator`, `partial_download_validator`, `vertical_corruption_validator`, které kontrolují příslušné typy poškození obrázků
    - Detekce objektů probíhá tak, že obrázky jsou zpracovány pomocí předem trénovaných modelů YOLOv8s. Výsledky obsahují informace o lokalizaci a typu detekovaných objektů.
    - Na základě detekovaných objektů a dalších atributů obrázku jsou data klasifikována do příslušných kategorií - typ scény, počasí.
    - Výstupy z detekce a klasifikace jsou dále zpracovány pro odstranění redundancí, filtrování nerelevantních dat s nízkou úrovní jistoty

4. Formátování a odeslání výsledků
    - Výstupy z procesů jsou transformovány do finálního formátu odpovědí:
        - JSON struktura obsahující výsledky detekce, klasifikace a validace.
        - Obrázek s nakreslenými rámečky detekovaných objektů a výsledky validací zapsané v hlavičce odpovědi

    - Formátovaná odpověď je odeslána zpět klientovi prostřednictvím HTTP odpovědi. Struktura odpovědi zahrnuje status kód. V případě obrázkové odpovědi se jedná o „streaming“ odpověď.

5. Logování a monitoring
    - Během celého procesu jsou důležité události logovány pro účely debugování a monitoringu. Logy obsahují informace o průběhu zpracování, chybách a výkonu systému.

Technologie a knihovny použité v projektu

Projekt využívá širokou škálu technologií a knihoven k zajištění funkcionality, výkonu a škálovatelnosti systému. Zde je detailní přehled důležitých technologií:

1. Python:
    - Verze: >=3.10.0, <3.12.0
    - Python je základním jazykem, ve kterém je projekt napsán

2. PyTorch:
    - Verze: torch=2.0.1, torchvision=0.15.2, torchaudio=2.0.2
    - PyTorch je jedna z nejdůležitějších knihoven využívaná modely hlubokého učením. Umožňuje efektivní výpočty na tenzorech a automatické diferencování.
3. FastAPI:
    - Verze: 0.100.0
    - FastAPI je moderní webový framework pro stavbu API s automatickou validací vstupů a generováním dokumentace.
4. NumPy:
    - Verze: 1.23.2
    - NumPy je základní knihovna pro numerické výpočty v Pythonu poskytující podporu pro velké multidimenzionální pole a matice spolu s širokou škálou matematických funkcí.
5. Pandas:
    - Verze: 2.0.2
    - Pandas je knihovna poskytující vysokovýkonné, snadno použitelné datové struktury a nástroje pro analýzu dat v Pythonu.
6. SciPy:
    - Verze: 1.10.1
    - SciPy je knihovna používaná pro vědecké a technické výpočty, která obsahuje moduly pro optimalizaci, lineární algebru, integraci, interpolaci, speciální funkce, FFT, zpracování signálu a obrazu, ODE řešiče a další.
7. Scikit-learn:
    - Verze: 1.3.2
    - Scikit-learn je robustní knihovna pro strojové učení v Pythonu, která poskytuje jednoduché a efektivní nástroje pro datovou analýzu a modelování dat.
8. OpenCV:
    - Verze: 4.7.0.72
    - OpenCV (Open Source Computer Vision Library) je knihovna zaměřená na počítačové vidění a strojové učení.

9. Matplotlib:
    - Verze: 3.7.1
    - Matplotlib je knihovna pro tvorbu statických, interaktivních a animovaných vizualizací v Pythonu.
10. Uvicorn:
    - Verze: 0.22.0
    - Uvicorn je lehký ASGI server pro Python, určený pro běh FastAPI aplikací, což zvyšuje jejich asynchronní schopnosti.

- Další Knihovny:
    - `pillow`, `scikit-image`, `albumentations`: Knihovny pro zpracování obrazu.
    - `tqdm`, `parse`, `debugpy`, `orjson`, `pebble`: Nástroje pro zvýšení výkonnosti aplikace, pro vývoje a debugging.
    - `httpx`, `requests-toolbelt`: Pro síťové komunikace a práci s HTTP požadavky.
    - `pytest`, `mypy`, `black`, `pylint`, `jupyter`: Nástroje pro testování, statickou analýzu kódu, formátování kódu a interaktivní programování.

Klientská JavaScript aplikace pro vizualizaci výsledků analýzy a validace

Tento část popisuje funkcionalitu a strukturu klientské aplikace, která slouží pro zobrazování výsledků analýzy a validace obrázků prostřednictvím webového rozhraní.

Struktura Aplikace

1. Konfigurace a inicializace
   Aplikace je inicializována s následujícími konfiguračními parametry:
    - `endpointType`: Typ endpointu, který se používá pro komunikaci se serverem. Výchozí hodnota je `detect-json`.
    - `getApiUrl`: Funkce pro generování URL API na základě aktuálního protokolu, hostname a portu.
2. Zpracování médií
   Aplikace podporuje zpracování obrázkových souborů pomocí těchto metod:
    - `triggerFileSelect`: Aktivuje výběr souborů.
    - `rescaleAndPostImage`: Změní velikost obrázku a pošle ho na server k analýze.
    - `resizeAndSendImage`: Přeškáluje obrázek na nové rozměry a pošle jej na server.

3. Zobrazení výsledků
   Komponenty pro zobrazení výsledků analýzy a validace:

- `showImgInfoWelcome`, `showImgInfoProcessing`, `showImgInfoError`: Funkce pro zobrazení stavů UI během zpracování obrázků.
- `drawBboxes`: Kreslí ohraničující boxy detekovaných objektů na obrázku.
- `generateImg`: Generuje a zobrazuje obrázek s detekovanými objekty.
- `generateODValidators`, `generateCVValidators`, `generateIsValid`: Tyto funkce generují UI komponenty, které zobrazují výsledky validace jako jsou detekce objektů, kontrola obsahu a další.

Technologie a Knihovny

Aplikace využívá následující technologie a knihovny:

- HTML/CSS: Pro strukturu a styl webové aplikace.
- JavaScript: Pro dynamické zpracování na straně klienta.

Hostování klientské aplikace prostřednictvím NGINX

Klientská aplikace je hostována na serveru NGINX, který je konfigurován pro správu statického obsahu a přesměrování API požadavků.

Formát logování

Pro logování je definován formát main_time, který zaznamenává základní informace o příchozích požadavcích včetně IP adresy, uživatele, času, typu požadavku, statusu, odeslaných bytech, refereru, user agentu a času zpracování požadavku.

Server konfigurace

Server naslouchá na portu 3000 a je konfigurován pro obsluhu souborů z kořenového adresáře `/meteval/client`. Maximální velikost těla požadavku je nastavena na 10MB.

Konfigurace endpointů

Pro obsluhu health checks a API požadavků jsou definovány specifické lokace. Příkladem je přesměrování z /validator/healthz na `/healthz` a proxy pro API požadavky na upstream meteval.

Správa chyb

Pro správu chybových stránek je konfigurace NGINX nastavena tak, aby přesměrovala chybové stavy 500, 502, 503 a 504 na stránku /50x.html, která je umístěna v adresáři `/usr/share/nginx/html`.

NVIDIA Triton Inference Server

Triton Inference Server je optimalizovaný server pro nasazení strojového učení, který podporuje efektivní inferenci s využitím GPU i CPU prostředků. Server umožňuje správu více modelů a poskytuje robustní API pro snadnou integraci do produkčních systémů.

Konfigurace Triton Serveru

Spouštěcí skript

- `triton_startup.sh`
  Parametry příkazu:
  `--model-repository`: Adresář s modely, specifikuje se absolutní cesta k modelům na serveru, například `/models/yolo-onnx-cpu`.

  `--http-port=8000`: HTTP port pro příjem inferenčních požadavků od klientů.

  `--grpc-port=8001`: gRPC port pro efektivnější komunikaci v prostředích s vysokým objemem dat.

Dockerfile Konfigurace

- Základní obraz: `nvcr.io/nvidia/tritonserver:22.09-py3` - Tento obraz z NVIDIA GPU Cloud (NGC) obsahuje předinstalovaný Triton server.

- Kopírování souborů:
  - Modely jsou umístěny ve složce `/models` uvnitř Docker kontejneru.
  - Spouštěcí skript `triton_startup.sh` je umístěn v kořenovém adresáři kontejneru.
  - Vstupní bod (ENTRYPOINT): Určuje, že po spuštění kontejneru se má vykonat skript `triton_startup.sh`.

Triton Client (triton_client.py)

- Komunikace:
  - Klient využívá gRPC protokol pro komunikaci s Triton serverem, což umožňuje rychlé a efektivní zpracování dat.
- Konfigurace klienta:
  - Klient je inicializován s URL serveru, která je konfigurována v souboru nastavení (`cfg['TRITON_URL']`).

Seznam modelů běžících na NVIDIA Triton Inference Serveru

NVIDIA Triton Inference Server podporuje souběžné nasazení a provoz několika modelů, což umožňuje rozsáhlé a flexibilní možnosti inferencí v rámci jednoho serveru. Níže je uveden seznam modelů, které jsou aktuálně hostovány na serveru v konfiguraci pro CPU.

YOLO-ONNX-CPU Model Repository
Seznam modelů a jejich účel:

- Anatomical Exposure Detection YOLOv8s Model
- COCO Detection YOLOv8s Model
- Face Detection
- License Plate Detection
- Pipeline
    - Model umožňující inferenci všech jednotlivých detekčních modelů sekvenčně za sebou na jedno API volání, což odstraňuje problém násobné internetové latence, které dochází při API volání inference každého modelu zvlášť.

- Scene-Understanding Detection
- Sky Detection
- Sun Detection
- Text Detection
- Weather Classification

Každý model je uložen ve své vlastní složce spolu s konfiguračním souborem `config.pbtxt`, který specifikuje parametry modelu, vstupní a výstupní formáty a další metadatové informace důležité pro správnou funkci na serveru.