Dokumentacja Projektu

Tytuł

Automatyczna Klasyfikacja Obrazów Owoców z AutoML

Autorzy

Łukasz Kempa (s21620), Michał Mazurek (s24086)

Linki

GitHub: https://github.com/kempikl/fruits-classification

Hugging Face Spaces: https://huggingface.co/spaces/kempikl/fruit-classifier-app

Uproszczony model: https://github.com/pjamisz/fruits_classificator/tree/main

1. Opis problemu

Krótki opis problemu i kontekstu

Celem projektu jest zbudowanie automatycznego systemu rozpoznawania i klasyfikacji fotografii owoców na 206 różnych kategorii. Rozwiązanie to adresuje wyzwania związane z ręcznym etykietowaniem produktów w branży spożywczej, przyspieszając proces inwentaryzacji i minimalizując błędy ludzkie.

Dlaczego?

Automatyzacja pozwala znacząco obniżyć koszty operacyjne sklepów detalicznych i platform e-commerce, a także zapewnia lepszą jakość danych w systemach logistycznych. Wykorzystanie AutoML umożliwia szybkie prototypowanie modeli bez głębokiej wiedzy inżynierskiej, co jest istotne dla firm o ograniczonych zasobach data science.

Gdzie?

System może być zintegrowany z kasami samoobsługowymi w sklepach stacjonarnych, aplikacjami mobilnymi dla konsumentów oraz zapleczem back-office do automatycznego uzupełniania stanów magazynowych.

Kontekst biznesowy

Wdrożenie przewidziano w średnich i dużych sieciach sklepów owocowo-warzywnych oraz w platformach dostaw „fruits-as-a-service". Użytkownikami końcowymi będą:

- Pracownicy działów inwentaryzacji (szybsze i dokładniejsze liczenie produktów),
- Konsumenci indywidualni (mobilna aplikacja do szybkiego identyfikowania nieznanych gatunków),
- Menedżerowie logistyki (monitoring rotacji i stanów magazynowych w czasie rzeczywistym).

2. Dane

Źródła danych i wiarygodność

Dane pochodzą z publicznego repozytorium Kaggle – zestawu Fruits 360 autorstwa Moltean (2018), zawierającego 103 993 obrazów treningowych i 34 711 testowych.

Wykorzystany został dataset 100 x 100pixeli. W ostatecznej wersji modelu wykorzystano obrazy treningowe, które zostały podzielone w preprocessingu na zbior treningowy i testowy.
Pierwsze trenowane modele opierały się na wszystkich 206 klasach, ostatecznie stworzono również uproszczony model dla 3 klas: Banana, Strawberry, Watermelon.

URL: https://www.kaggle.com/datasets/moltean/fruits/data

Licencja: CC BY-SA 4.0

Opis danych

Obrazy w formacie RGB o rozdzielczości 100×100 px zostały posegregowane na 206 klas odpowiadających gatunkom i odmianom owoców. Przez jednolitą wielkość i kolorystykę tła zestaw idealnie nadaje się do trenowania sieci CNN bez skomplikowanego preprocessingu segmentacyjnego.

Uzasadnienie przydatności

Dane to ogromny dataset z fotografiami owoców, w różnych odmianach, brak w nim danych syntetycznych, dobrze nadaje się do trenowania klasyfikatorów.

Jakość i jednorodność danych (stała rozdzielczość, jednolite tło) pozwalają modelom głębokim skupić się wyłącznie na charakterystycznych cechach owocu.

Statystyki rozkładu próbek na klasę

Na podstawie skryptu python *scripts/class_distribution.py* uzyskano następujące miary opisowe liczby obrazów przypadających na każdą z 206 klas:

- Średnia (mean): 504,8 próbek
- Odchylenie standardowe (std): 156,3
- Minimum (min): 144 próbek
- kwartyl (25%): 450 próbek
- Mediana (50%): 490 próbek

- kwartyl (75%): 666 próbek
- Maksimum (max): 984 próbki

Taki rozkład świadczy o umiarkowanej nierówności liczebności klas – większość klas ma w przybliżeniu 450–666 obrazów, ale są też (np. najbogatsze) sięgające prawie 1 000 oraz słabiej reprezentowane – poniżej 200.

3. Sposób rozwiązania problemu

Wybrany model i uzasadnienie
Stworzono dwa niezależne modele predykcyjne. Jeden klasyfikujący wszystkie 206 klas owoców, wraz z pełną konteneryzacją i wdrożeniem na chmurę. Z powody trudności z wytrenowaniem modelu dla tak złożonego problemu przy brakach sprzętowych, stworzono też alternatywny, uproszczony model dla klasyfikacji 3 klas owoców z wysoką skutecznością.

Etapy realizacji projektu

1. Strukturyzacja projektu z użyciem Kedro: definicja pipelines i katalogu danych.
2. Wczytanie i podział zbioru na część treningową oraz testową.
3. Strojenie hyperparametrów przy pomocy RandomSearch (maks. 50 prób, wczesne zatrzymanie po 5 epochach bez poprawy).
4. Zapis najlepszego modelu do formatu HDF5.
5. Interfejs użytkownika: aplikacja Streamlit udostępniająca funkcjonalność uploadu zdjęcia i wizualizacji predykcji.
6. Konteneryzacja: przygotowanie pliku Dockerfile do łatwego wdrożenia w środowisku produkcyjnym.
7. Wdrożenie produkcyjne w chmurze (Hugging Face Spaces).

Etapy realizacji uproszczonego modelu dla 3 klas

1. Strukturyzacja projektu z użyciem Kedro: definicja pipelines i katalogu danych.
2. Wczytanie ścieżek obrazów i podział zbioru na część treningową oraz testową.
3. Trenowanie modelu za pomocą AutoGluon MultiModal Predictor, automatycznie wybierający najlepsze algorytmy klasyfikacji i hiperparametry
4. Ograniczenie zakresu modelu do klasyfikacji 3 klas spowodowane brakiem możliwośći sprzętowych do wytrenowania klasyfikatora dla wszystkich owoców
5. Stworzenie backendu w FastAPI wystawiającego możliwość predykcji owocu na podstawie obrazu , wraz z testowaniem.
6. Konteneryzacja za pomocą Dockera.
7. Interfejs użytkownika: aplikacja Streamlit udostępniająca funkcjonalność uploadu zdjęcia i wypisania predykcji.

Wyniki ewaluacji modelu
Model dla wszystkich 206 klas z powodu złożoności problemu ma niską skuteczność, co zostało rozwiązane stworzeniem uproszczonego modelu dla 3 klas.

Model uproszczony z racji problemu klasyfikacji dla zaledwie 3 klas owoców osiąga bardzo wysoką skuteczność powyżej 95%, przy rozkładzie prawdopodobieństw około 99%/0,5%/0,5% dla poszczególnych klas.

4. Szczegółowy opis aplikacji

Wykorzystane narzędzia

Język i biblioteki: Python 3.10, TensorFlow 2.11, KerasTuner 1.3.5, Kedro 0.19.x, Streamlit 1.x

Infrastruktura: Docker, Git + GitHub

Struktura kodu

src/fruits_classification/pipelines/training/nodes.py – moduł odpowiedzialny za wczytywanie danych, strojenie modelu oraz zapis najlepszej konfiguracji.

pipeline_registry.py – rejestracja i kolejność uruchamiania etapów pipeline'u.

app.py – kod aplikacji Streamlit, obsługa uploadu, predykcji i wyświetlania wyników.

conf/base/parameters.yml – ścieżki do danych, parametry hyperparametrów, ścieżka wyjściowa modelu.

conf/base/class_names.json – lista nazw klas używana w interfejsie.

Dockerfile – instalacja zależności oraz domyślne polecenie uruchomienia Streamlita.

Odwołania do kodu

Logika treningu: funkcja train_model w nodes.py (linie 1–80).

UI: sekcja "Uploader i predykcja" w app.py (funkcja main()).

Wykorzystane narzędzia dla uproszczonego modelu dla 3 klas

Język i biblioteki: Python, Kedro, AutoGluon (ze wsparciem PyTorch i Transformers) Streamlit FastAPI

Infrastruktura: Docker, Git + GitHub

Struktura kodu

src/fruits/pipelines/data_processing/nodes.py – węzeł odpowiedzialny za wczytanie obrazów, wstawienie ich ścieżek do dataframe, podział na zbiory

src/fruits/pipelines/data_processing/nodes.py

pipeline_registry.py – rejestracja i kolejność uruchamiania etapów pipeline'u.

src/fruits/deploy/frontend/app.py – kod aplikacji Streamlit, obsługa uploadu, predykcji i wyświetlania wyników.

conf/base/parameters.yml – ścieżki do danych, parametry hyperparametrów, ścieżka wyjściowa modelu.

Dockerfile – instalacja zależności oraz domyślne polecenie uruchomienia Streamlita.

Odwołania do kodu

Logika treningu: funkcja train_model w nodes.py (linie 1–80).

UI: sekcja "Uploader i predykcja" w app.py (funkcja main()).

5. Podsumowanie

Co się udało?

- Stworzenie kompletnego, zautomatyzowanego pipeline'u AutoML w środowisku Kedro,
- Uzyskanie modelu o wysokiej efektywności dzięki optymalizacji hyperparametrów,
- Udostępnienie prostego interfejsu Streamlit, który umożliwia użytkownikom końcowym natychmiastowe testowanie modelu.

Problemy i rozwiązania

- Problemy z wytrenowaniem modelu dla wszystkich 206 klas rozwiązano przez stworzenie uproszczonego modelu dla 3 klas.
- Konflikty wersji KerasTuner, AutoGluon – rozwiązano poprzez jednoznaczne określenie wersji w requirements.txt.
- Serializacja tf.data.Dataset w Kedro – zastąpiono domyślny MemoryDataset własnym node'em łączącym dane, co wyeliminowało błędy pamięci.
- Wczesne zatrzymanie – implementacja callbacku EarlyStopping pozwoliła na redukcję czasu treningu bez utraty jakości.

Kierunki rozwoju

- Wytrenowanie modelu dla 206 klas, tak przewidywał z lepszą skutecznością.
- Obracanie, przycinanie i zmiana jasności, by model lepiej radził sobie z różnymi ujęciami.

- Dodanie opcji przesyłania wielu zdjęć naraz, by przyspieszyć testowanie.
- Wdrożenie uproszczonego modelu na chmurę.