Lista zadań poza skalę

Ostateczny termin oddania: 6.02.2026, godz. 23:59

Za wykonanie listy można otrzymać **łącznie 9 punktów**, które **nie wliczają się do maksimum punktów z przedmiotu**. Z tej puli 2 punkty zostaną doliczone do punktów za ćwiczenia, natomiast 7 punktów pochodzi z puli zadań bonusowych (na wyższą ocenę).

Można oddać **część zadania** – nie ma obowiązku realizowania całości.

Postaraj się wykonać zadanie **samodzielnie**, to znaczy bez korzystania z narzędzi generujących kod. Zachęcam natomiast do korzystania z dokumentacji, blogów, tutoriali itp.

Jeśli w trakcie realizacji któregokolwiek etapu pojawią się problemy lub pytania, prowadzący chętnie pomogą i odpowiedzą na nie podczas laboratoriów lub mailowo. W szczególności śmiało pytajcie o część z PyTorch. Jesteśmy świadomi, że jesteście przed lub w trakcie przedmiotów z zakresu ML/DL, dlatego zadawanie pytań ani korzystanie z naszej pomocy **nie wiąże się z utratą punktów**. Zależy nam przede wszystkim na tym, żebyście „dobrnęli do końca" :)

Zadanie projektowe: segmentacja binarna obrazów

Cel projektu

Celem zadania jest przejście przez pełny pipeline ML / DL na przykładzie segmentacji binarnej obrazów. Projekt ma pokazać zastosowanie różnych narzędzi do pracy ML /DL, dlatego spróbuj używać różnych narzędzi i bibliotek poznanych podczas wykładu. Ich wybór zależy od Ciebie.

W ramach projektu będziesz miał okazję:

- pracować w repozytorium GitHub i regularnie commitować zmiany,
- eksplorować i wizualizować dane (matplotlib),
- pobrać i przygotować dane (NumPy, pandas),
- zbudować modele bazowe (scikit-learn),
- zaimplementować prostą sieć neuronową (PyTorch),
- użyć gotowego modelu segmentacyjnego (Hugging Face),
- logować metryki i obrazy (TensorBoard),
- porównać wszystkie podejścia na tych samych danych testowych.

Zbiór danych

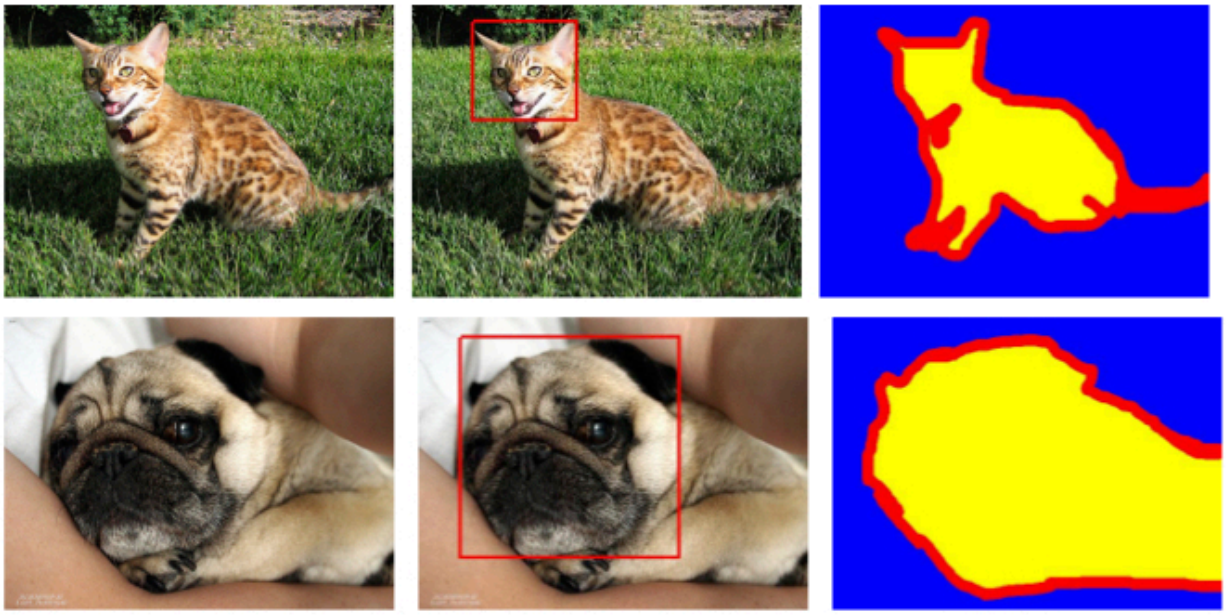
Zbiorem danych, na którym będziesz pracować, jest **Oxford-IIIT Pet**. Dataset składa się z:

- wielu obrazów RGB przedstawiających zwierzęta (psy i koty),
- odpowiadających im masek segmentacyjnych.

Segmentacja to zadanie, które polega na zaznaczeniu obszaru, na którym występuje obiekt, czyli na przypisaniu każdemu pikselowi obrazu jednej z dwóch klas:

- 1 – obiekt (zwierzę),
- 0 – tło.

Przykład danych ze zbioru danych:



Zadanie

Repozytorium GitHub (1pkt - punkt na wyższą ocenę)

Na początku projektu załóż repozytorium na GitHubie. W trakcie pracy:

- wykonaj minimum 4 sensowne commity
- nie wrzucaj do repozytorium zapisanych modeli ani logów TensorBoard, tworząc i poprawnie konfigurując plik `.gitignore`.

Repozytorium powinno zawierać kod (pliki py, notebooki ect.) oraz instrukcję uruchomienia projektu.

Eksploracja danych (1pkt - punkt z laboratorium)

Dane pobierz z Hugging Face lub Kaggle. Następnie przeprowadź eksplorację danych, sprawdzając:

- liczbę próbek w zbiorze,
- rozmiary obrazów (wszystkie mają taki sam rozmiar? A może rozmiary obrazków są różne?),
- typ masek (czy zawierają tylko 0 i 1, czy również inne wartości?),
- liczbę klas w maskach.

Policz pokrycie obrazu przez maskę, czyli procent pikseli należących do obiektu, i zwizualizuj ten rozkład na histogramie. Sprawdź również jasność obrazów (np. jako średnią wartość pikseli) i krótko skomentuj jej rozkład.

Na koniec wyświetl kilka (np 7) przykładowych obrazów wraz z odpowiadającymi im maskami.

Przygotowanie danych (1pkt - punkt z laboratorium)

Maski należy sprowadzić do segmentacji binarnej, tak aby zawierały wyłącznie wartości 0 (tło) i 1 (obiekt). W tym celu zamień inne występujące wartości na zera i jedynki. Następnie zmień rozmiar obrazów i masek do wspólnego rozmiaru, na przykład 128 × 128.

Dane należy przygotować w taki sposób, aby możliwa była klasyfikacja pikselowa. Oznacza to, że:

- każdy piksel będzie osobną próbką,
- cechy piksela to jego wartości RGB,
- etykietą jest wartość maski w tym pikselu.

Jeżeli zbiór danych okaże się zbyt duży, można go zmniejszyć, zachowując pierwotne proporcje klas.

Całość podziel na:

- zbiór treningowy (60%),
- zbiór testowy (30%),
- zbiór walidacyjny (10%).

Od tego momentu wszystkie modele muszą korzystać z tego samego podziału danych, czyli wszystkie trenujemy i testujemy na tych samych zbiorach.

Modele bazowe (1pkt - punkt na wyższą ocenę)

Zbuduj naiwne modele segmentacji pikselowej, w których pojedynczy piksel jest jedną próbką wejściową. Modele powinny otrzymywać na wejściu wektor RGB o rozmiarze (3,) i zwracać predykcję klasy piksela.

Użyj *GradientBoostingClassifier* i *RandomForestClassifier* (lub dwóch innych modeli według uznania).

Spróbuj poprawić jakość modeli poprzez dobór hiperparametrów lub inne poznane techniki. Napisz jak radził sobie Twój model przed poprawą parametrów i po niej.

Modele oceń na zbiorze testowym, licząc accuracy oraz IoU (Intersection over Union). Następnie złóż predykcje z powrotem w obrazy i pokaż kilka przykładów w postaci trójek: obraz, maska prawdziwa, maska przewidziana. Opisz, gdzie modele działały dobrze, a gdzie popełniały największe błędy.

Sieć neuronowa MLP, Dataset, Dataloader (1pkt - punkt na wyższą ocenę)

W tej części należy wykonać segmentację pikselową przy użyciu prostej sieci neuronowej MLP. Sieć otrzymuje pojedynczy piksel opisany trzema wartościami RGB i zwraca prawdopodobieństwo przynależności do klasy „obiekt”.

Architektura sieci jest dana, ale możesz ją zmienić wg uznania:

```
class PixelMLP(nn.Module):
    def __init__(self):
        super().__init__()
```

```
        self.net = nn.Sequential(
            nn.Linear(3, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.net(x)
```

Napisz własny dataset i dataloader. Jeśli wiesz jak, to nie czytaj dalej tego akapitu, tylko napisz samodzielnie :)

Dataset składa się z wielu obrazów, dlatego klasa `Dataset` musi wiedzieć:

- z którego obrazu pochodzi dany piksel,
- jaki jest jego indeks w obrazie,
- jak na tej podstawie pobrać właściwy piksel i etykietę z maski.

Przykładowo:

- `__len__()` może zwracać łączną liczbę pikseli we wszystkich obrazach,
- `__getitem__(idx)` powinno:
  - zamienić `idx` na (id_obrazka, id_piksela),
  - pobrać piksel o kształcie (3,),
  - pobrać etykietę o kształcie (1,).

Po użyciu DataLoader:

- batch danych wejściowych ma kształt (batch_size, 3),
- batch etykiet ma kształt (batch_size, 1).

Trening (1pkt - punkt na wyższą ocenę)

Ponieważ jest to segmentacja binarna, w treningu należy użyć *Binary Cross-Entropy* jako funkcji straty. Można użyć:

- *BCELoss* (jeśli wyjście przechodzi przez sigmoid),
- albo *BCEWithLogitsLoss* (po usunięciu sigmoidu z sieci).

Napisz pętlę treningową.

Na podstawie wyników na zbiorze walidacyjnym zapisz najlepszy model, wybierając jako kryterium IoU lub accuracy i krótko uzasadniając ten wybór.

Tensorboard (1pkt - punkt na wyższą ocenę)

W trakcie treningu i walidacji należy logować do TensorBoard:

- loss,
- accuracy,
- IoU.

Gotowy model segmentacyjny (1pkt - punkt na wyższą ocenę)

Pobierz dowolny pretrenowany model segmentacyjny wytrenowany na zbiorze Oxford-IIIT Pet (lista dostępna na stronie https://huggingface.co/datasets/timm/oxford-iiit-pet w zakładce *Models trained or fine-tuned on*…). Wykonaj inferencję na wybranych obrazach i porównaj uzyskane maski z wynikami modeli bazowych oraz MLP.

Uwaga! Wejściem do tej sieci jest najprawdopodobniej cały obraz, a nie pojedyncze piksele.

Końcowa ocena (1pkt - punkt na wyższą ocenę)

Dla wszystkich modeli (sklearn, MLP, model z Hugging Face) policz na zbiorze testowym accuracy oraz IoU. Złóż predykcje w obrazy i pokaż kilka trójek: obraz, maska prawdziwa, maska przewidziana.

Na koniec odpowiedz:

- który model poradził sobie najlepiej,
- gdzie poszczególne modele popełniały błędy,
- jaki jest stosunek TP, TN, FP i FN dla każdego z modeli
- mapy błędów na obrazkach (w zależności od przynależności do TP, TN, FP, FN).