Dokumentacja projektu Geoguesser na podstawie danych z dashcamów samochodowych - Analiza i przetwarzanie obrazów, grupa 4

Franciszek Data
Marta Figurska
Alicja Kałuża
Szymon Kolanko
Bartosz Konopka
Filip Krupa
Mikołaj Kuszowski
Maciej Witkowski

1 Podział obowiązków

Imię i nazwisko	Funkcja w zespole
Franciszek Data	Przeniesienie modelu z Tensorflow do Pytorch.
Marta Figurska	Utworzenie i wytrenowanie modelu.
Alicja Kałuża	Przygotowanie i obróbka danych wejściowych.
Szymon Kolanko	Ewaluacja modelu i skrypt.
Bartosz Konopka	Utworzenie i wytrenowanie modelu.
Filip Krupa	Utworzenie wstępnej wersji modelu.
Mikołaj Kuszowski	Przygotowanie i obróbka danych wejściowych.
Maciej Witkowski	Wydobycie cech: rozpoznawanie tablic rejestracyjnych.

Cały zespół uczestniczył również w tworzeniu dokumentacji.

2 Instrukcja wdrożenia

Projekt składa się z czterech plików - dwóch notatników, jednego skryptu oraz pliku z wagami modelu. Plik pythonowy umożliwia przetestowanie wytrenowanego modelu klasyfikującego obrazy lub klatki wideo na podstawie zawartości wizualnej. Instalację można przeprowadzić za pomocą polecenia:

```
pip install torch torchvision pillow opencv-python numpy
```

2.1 Konfiguracja i uruchomienie

1. Ustawienie trybu działania poprzez zmianę zmiennej `input_type`:

```
input_type = 'image'  # lub 'video'
```

2. Wskazanie odpowiednich ścieżek do plików:
   - `test_image_path` – ścieżka do pliku graficznego (dla trybu `'image'`)
   - `test_video_path` – ścieżka do pliku wideo (dla trybu `'video'`)
   - `weights_path` – ścieżka do pliku z wagami modelu (`.pth`)
3. Uruchomienie skryptu:

```
python nazwa_skryptu.py
```

2.2 Opis działania

- **Tryb** `image` – Model wykonuje klasyfikację pojedynczego obrazu i wyświetla najbardziej prawdopodobną klasę wraz z rozkładem procentowym.
- **Tryb** `video` – Model analizuje wybrane klatki z wideo, dokonuje predykcji dla każdej z nich, a wynik końcowy opiera się na średniej z uzyskanych wyników.

3 Przygotowanie danych

3.1 Opis danych

Do realizacji projektu wykorzystano zestaw danych o nazwie "Mapillary Street-level Sequences". Zawiera on ponad 1,6 miliona obrazów z 30 największych miast położonych na sześciu kontynentach. Zestaw był tworzony przez 9 lat i obejmuje wszystkie pory roku. Zdjęcia wykonano w różnych warunkach pogodowych, za pomocą różnych kamer oraz przy zróżnicowanym oświetleniu. Znacząco zwiększa to różnorodność tego zbioru danych.

3.2 Analiza i przygotowanie danych

Dane z wyżej wymienionego zestawu danych zostały pobrane i przeniesione do chmury, aby ułatwić do nich dostęp. W programie tworzona jest lista ścieżek do tych obrazów za pomocą poniższej funkcji.

```
def take_images(folders):
  all_image_paths = []
  i = 0
  for folder in folders.keys():
      print("Sprawdzam folder:", folder)
      for root, _, files in os.walk(folder):
          for file in files:
              if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                  full_path = os.path.join(root, file)
                  all_image_paths.append(full_path)
  return all_image_paths
```

Listing 1: Funkcja grupująca ścieżki do obrazów

Do obsługi zdjęć została stworzona klasa "ImageDataset", która jest własną implementacją zbioru danych w PyTorch. Konstruktor klasy przyjmuje parametry:

1. images_paths - lista ścieżek do obrazów (pobrana za pomocą funkcji "take_images"),
2. targets_dict - słownik, który mapuje ścieżkę do obrazu na odpowiedni kraj,
3. country_to_continent - słownik, który mapuje kraj na kontynent
4. transform - transformacja, która zostanie zastosowana do obrazów

Klasa "ImageDataset"odpowiada za przekształcanie obrazów za pomocą transformacji (np. do wspólnej rozdzielczości) oraz za zwracanie obrazów z odpowiednimi etykietami w postaci krotki: (obraz, kraj, kontynent). Dziedziczy po klasie "Dataset", co pozwala na jej użycie z DataLoaderem w PyTorch. Zawiera również metodę zwracającą liczbę obrazów, co jest wymagane do poprawnego działania DataLoadera.

```
class ImageDataset(Dataset):
    def __init__(self, images_paths, targets_dict,
       country_to_continent, transform=None):
```

```
        self.images_paths = images_paths
        self.transform = transform
        self.targets_dict = targets_dict
        self.country_to_continent = country_to_continent

    def __len__(self):
        return len(self.images_paths)

    def __getitem__(self, idx):
        image_path = self.images_paths[idx]
        image = Image.open(image_path).convert("RGB")
        target = self.targets_dict[image_path.rsplit('/',1)[0]]
        general_label = self.country_to_continent[target]

        if self.transform:
            image = self.transform(image)

        return image, target, general_label
```

Listing 2: Klasa ImageDataset

Do analizy wczytanych danych stworzono funkcję "data_stats". Jako parametr wejściowy przyjmuje obiekt klasy "ImageDataset". Funkcja zlicza etykiety krajów oraz kontynentów, a następnie wyświetla liczbę wszystkich zdjęć w zbiorze, liczbę zdjęć przypadającą na każdy kraj oraz na każdy kontynent.

```
def data_stats(full_train_dataset):
    country_labels = [full_train_dataset.targets_dict[path.rsplit('/'
       , 1)[0]] for path in full_train_dataset.images_paths]
    continent_labels = [country_to_continent[full_train_dataset.
       targets_dict[path.rsplit('/', 1)[0]]]for path in
       full_train_dataset.images_paths]

    country_counts = Counter(country_labels)
    continent_counts = Counter(continent_labels)

    print("Liczba␣zdjec␣treningowych:", len(full_train_dataset))
    print("\n")
    for country, count in country_counts.items():
        print(f"{country}:␣{count}␣obrazow")
    print("\n")
    for continent, count in continent_counts.items():
        print(f"{continent}:␣{count}␣obrazow")
```

Listing 3: Funkcja data_stats

Jej wywołanie zwraca następujące wyniki:

```
Liczba zdjec treningowych: 483601

```

```
netherlands: 11539 obrazow
jordan: 953 obrazow
hungary: 153321 obrazow
india: 12388 obrazow
russia: 77496 obrazow
france: 8480 obrazow
switzerland: 9012 obrazow
kenya: 437 obrazow
canada: 82545 obrazow
norway: 5015 obrazow
usa: 64267 obrazow
brazil: 18989 obrazow
japan: 34823 obrazow
grece: 2466 obrazow
uganda: 1870 obrazow

Europe: 267329 obrazow
Asia: 48164 obrazow
Africa: 2307 obrazow
North America: 146812 obrazow
South America: 18989 obrazow
```

Listing 4: Wynik wywołania funkcji data_stats

Uzyskane wyniki zwróciły naszą uwagę. Rozpiętość liczby przykładów pomiędzy etykietami była znacząca. Dla etykiety "hungary"liczba zdjęć wyniosła 153321, natomiast dla etykiety "kenya"jedynie 437. Taka nierównowaga klas spowodowałaby bias sieci. Model nauczyłby się przewidywać klasy bardziej liczne, ponieważ pozwalałoby to skuteczniej minimalizować funkcję straty. Aby temu zapobiec zostały zaimplementowane dwie techniki:

1. augmentacja danych
2. downsampling

Poniżej przedstawiono funkcję odpowiedzialną za augmentację danych.

```
def take_images_with_augmentation(folders,max_count):
  all_image_paths = []
  i = 0
  country_to_folder = {}
  for folder in folders.keys():
      if not folders[folder] in country_to_folder.keys():
          country_to_folder[folders[folder]] = []
      country_to_folder[folders[folder]].append(folder)

  for country in country_to_folder.keys():
      folder_images = []
      for folder in country_to_folder[country]:
        for root, _, files in os.walk(folder):
            for file in files:
                if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                    full_path = os.path.join(root, file)
                    folder_images.append(full_path)
      if len(folder_images) < max_count:
          num_to_add = max_count - len(folder_images)
          additional_paths = random.choices(folder_images, k=
             num_to_add)
          folder_images.extend(additional_paths)
      all_image_paths.extend(folder_images)

  return all_image_paths
train_transform = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation
       =0.2, hue=0.1),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.ToTensor()
])

files_train_augmentation = take_images_with_augmentation(
   folders_train,max_count)
full_train_dataset_with_augmentation = ImageDataset(
   files_train_augmentation, folders_train, country_to_continent,
   transform=train_transform)
```

Listing 5: Funkcja take_images_with_augmentation

Po augmentacji danych zaimplementowano kod odpowiedzialny za downsampling, który przedstawiono poniżej.

```
country_to_indices = defaultdict(list)

for idx, path in enumerate(full_train_dataset.images_paths):
    country = full_train_dataset.targets_dict[path.rsplit("/", 1)[0]]
    country_to_indices[country].append(idx)

min_count = min(len(indices) for indices in country_to_indices.values
   ())

balanced_indices = []

for indices in country_to_indices.values():
    balanced_indices.extend(random.sample(indices, min_count))

balanced_image_paths = [full_train_dataset.images_paths[i] for i in
   balanced_indices]

balanced_train_dataset = ImageDataset(
    balanced_image_paths,
    targets_dict=full_train_dataset.targets_dict,
    country_to_continent=full_train_dataset.country_to_continent,
    transform=full_train_dataset.transform
)
])

files_train_augmentation = take_images_with_augmentation(
   folders_train,max_count)
full_train_dataset_with_augmentation = ImageDataset(
   files_train_augmentation, folders_train, country_to_continent,
   transform=train_transform)
```

Listing 6: Fragment kodu odpowiedzialny za downsampling

Dzięki zastosowaniu tych technik liczba zdjęć dla poszczególnych klas została wyrównana, co wyeliminowało ryzyko wystąpienia biasu w sieci.

3.3 Przykłady zdjęć ze zbioru danych

Rysunek 1: Mapillary Amsterdam

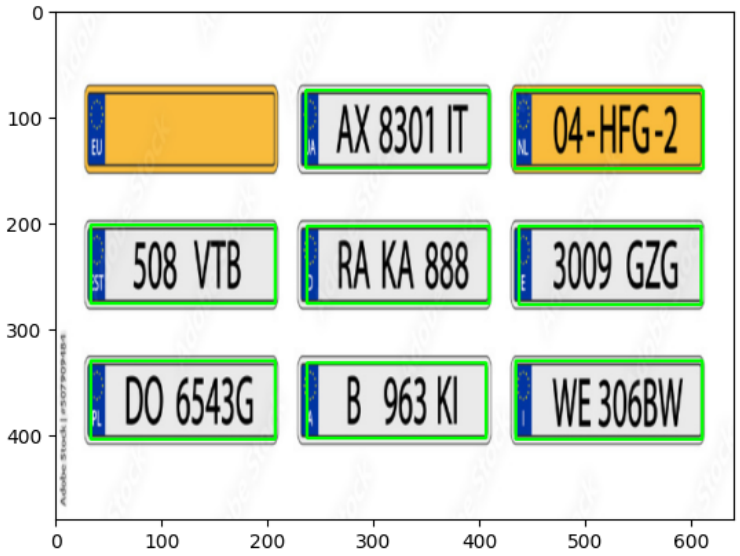
Rysunek 2: Mapillary Nairobi

4 Ekstrakcja cech z obrazu

Pierwotne założenia realizacji projektu przewidywały ekstrakcję cech z pojedynczych klatek obrazu, które umożliwiłyby na późniejszym etapie identyfikację państwa lub regionu, z którego pochodzi zdjęcie. Charakterystyczne informacje, takie jak tablice rejestracyjne, znaki drogowe, cechy środowiska naturalnego czy kierunek jazdy, miały posłużyć jako elementy wektora cech. Po ekstrakcji tych informacji oraz odpowiednim przygotowaniu wektora, miał on stanowić wejście do modelu, który na podstawie dostarczonych danych określiłby lokalizację pochodzenia zdjęcia.

Pierwszym etapem była detekcja tablic rejestracyjnych na obrazie. W tym celu wykorzystano wstępnie wytrenowany model Ultralytics YOLOv8, który został dodatkowo dostrojony na zbiorze danych Hugging Face's License Plate Object Detection. Użyty model jest dostępny na platformie Kaggle.

W celu przetestowania działania modelu przeprowadzono kilka eksperymentów na zdjęciach tablic rejestracyjnych wykonanych w różnych warunkach. Pierwszym, a zarazem najprostszym etapem było sprawdzenie skuteczności detekcji w warunkach idealnych (Rys. 3). W tym przypadku model poradził sobie bez żadnych problemów.



Rysunek 3: Detekcja tablic rejestracyjnych w idealnych warunkach

Następnym etapem było sprawdzenie detekcji na zdjęciu dobrej jakości, przedstawiającym zbliżenie tablicy rejestracyjnej. Jak widać na Rys. 4, tablica została poprawnie rozpoznana, jednak bounding box nie objął jej w całości, prawdopodobnie ze względu na nietypowy kolor.

W przypadku zdjęcia wykonanego w ruchu ulicznym (Rys. 5), mimo dużej liczby elementów na obrazie oraz niewielkich rozmiarów tablic, wstępnie wytrenowany model poprawnie zaznaczył obie widoczne tablice rejestracyjne.

Kolejny test, który teoretycznie mógłby stanowić dla modelu wyzwanie, polegał na detekcji tablicy na klatce z kamerki samochodowej (Rys. 6). Pomimo rozmazanego obrazu i niskiej jakości klatki, model ponownie prawidłowo zidentyfikował tablicę rejestracyjną

Rysunek 4: Detekcja tablic rejestracyjnych przy zbliżeniu

Rysunek 5: Detekcja tablic rejestracyjnych w ruchu ulicznym

Rysunek 6: Detekcja tablic rejestracyjnych z kamery samochodowej

W przypadku klatek pochodzących ze zbioru danych wykorzystanego przez nasz zespół pojawił się problem z zamazanymi tablicami rejestracyjnymi. Jak pokazano na Rys. 7, dwa widoczne pojazdy mają zamazane tablice rejestracyjne, co uniemożliwia ich detekcję przez model. Dodatkowo, w tym przypadku model popełnił błąd, błędnie rozpoznając światło ciężarówki jako tablicę rejestracyjną.

Rysunek 7: Zdjęcie z datassetu - zamazane tablice rejestracyjne

Aby sprawdzić, czy detekcja tablic rejestracyjnych ma w ogóle sens, przeprowadzono test na próbie 1000 losowo wybranych obrazów. W jego ramach dokonano detekcji tablic rejestracyjnych. W wyniku tej operacji uzyskano następujące dane: tablica rejestracyjna została

wykryta jedynie na 69 zdjęciach, co stanowi mniej niż 7% wszystkich przetestowanych obrazów.

W związku z powyższymi problemami, które niestety nie zostały zidentyfikowane na początkowych etapach projektowania rozwiązania, zespół podjął decyzję o zmianie podejścia. Zamiast ekstrakcji charakterystycznych cech obrazu i tworzenia na ich podstawie wektora cech, zdecydowano się na zastosowanie klasycznej konwolucyjnej sieci neuronowej.

5 Utworzenie i wytrenowanie modelu

W celu klasyfikowania obrazów do konkretnych krajów stworzono model oparty na warstwach konwolucyjnych.

5.1 Model

Architektura modelu :

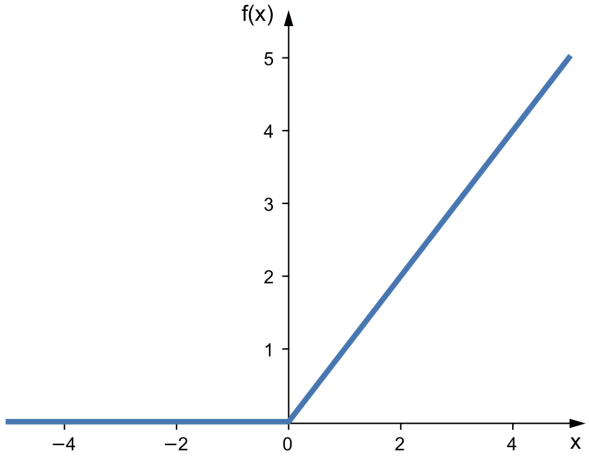
```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.3)
        self.fc1 = nn.Linear(32 * 16 * 16, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = self.dropout(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Listing 7: Klasa SimpleCNN

Dropout czyli parametr wyrzucający losowo część danych zdefiniowano na poziomie 0.3. Wspomaga on proces uogólniania treningu - zapobiega zbytniemu dopasowaniu się do danych treningowych(overfitting).

Kolejną zastosowaną techniką jest MaxPooling, który redukuje wymiarowość danych. Dzieli obraz na okienka a następnie na podstawie podanego wymiaru wyróżnia z nich największą wartość.

Wykorzystano funkcję aktywacji ReLU, która prezentuje się następująco:



Rysunek 8: Funkcja aktywacji Rectified Linear Unit

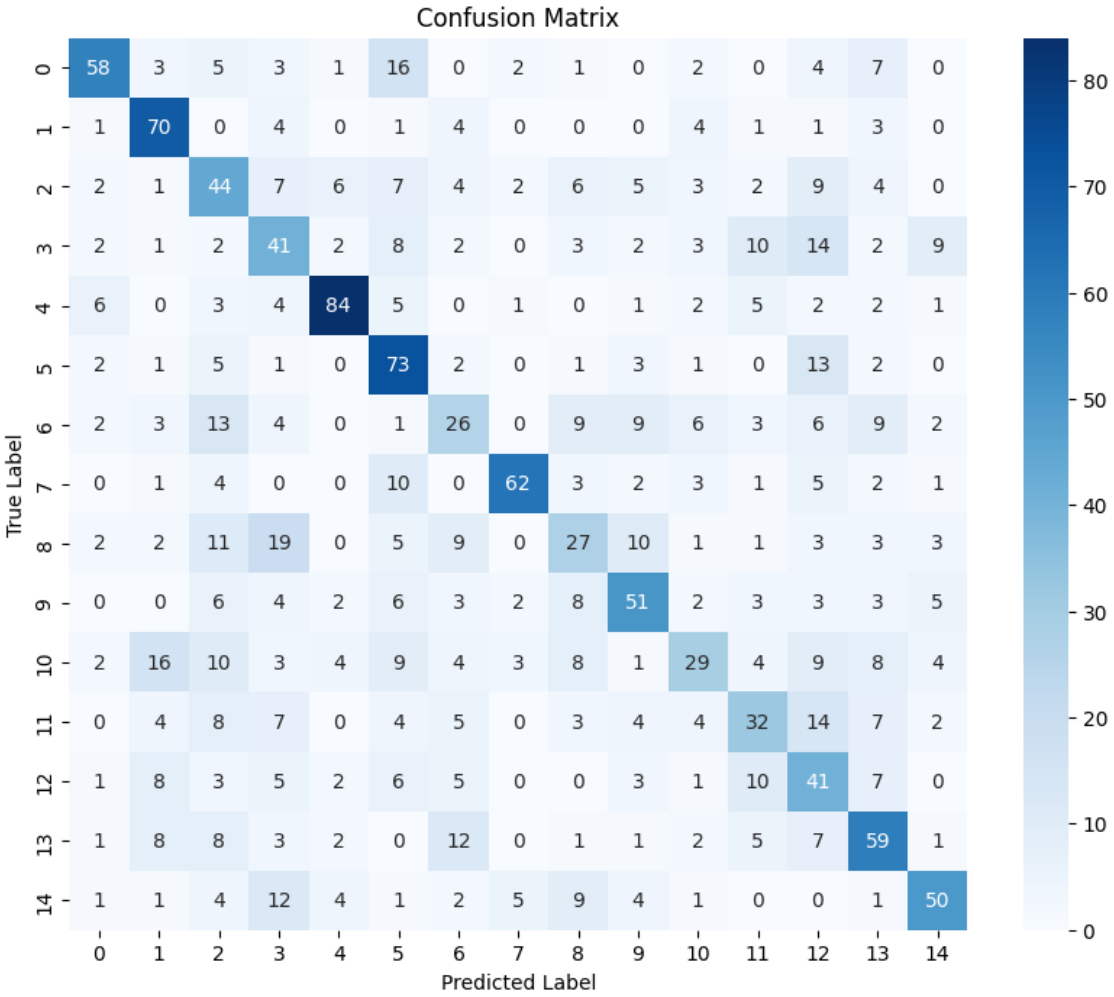
5.2 Trening

W celu wytrenowania modelu dane podzielono na zbiór treningowy i testowy w proporcjach 8:2. Liczbę epok ostatecznego treningu ustawiono na 9 - były podejmowane również próby przeprowadzenia treningu na 2 oraz 5 epokach, jednak wyniki nie były zadowalające. Learning rate ustawiono na poziomie 0.0001.

Trening trwał 5 godzin ze względu na sporą ilość danych - same dane treningowe to 6000 zdjęć(15 klas). Również brak dostępu do GPU zdecydowanie wydłużył proces trenowania - ograniczone zasoby GPU zostały już wcześniej wykorzystane. Po wykonanym treningu wagi zostały zapisane do pliku.

6 Ewaluacja modelu

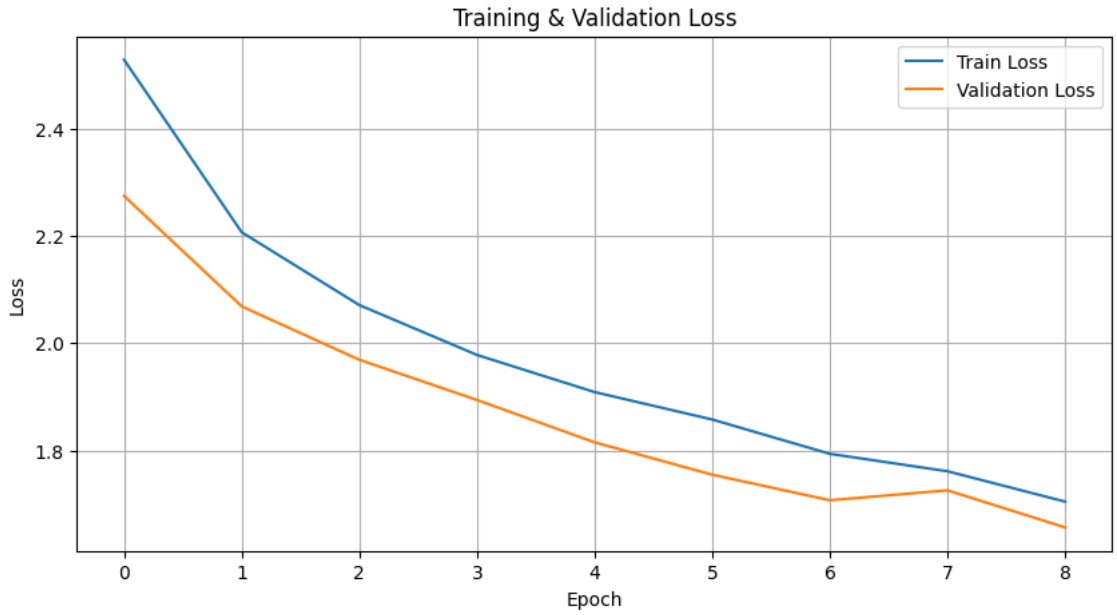
Macierz pomyłek prezentuje się następująco:



Rysunek 9: Macierz pomyłek

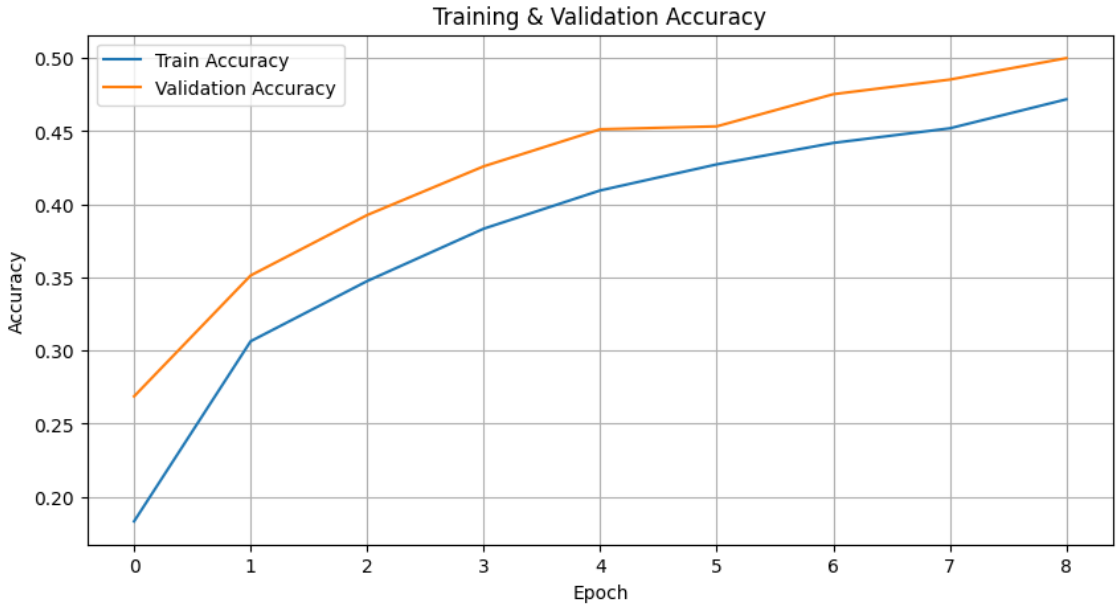
Rezultaty są zadowalające - celem jest uzyskanie jak najwyższych wartości na przekątnej i takie też zostały zaprezentowane. Oczywiście liczne pomyłki również mają miejsce - widać np że model wielokrotnie niesłusznie przewidywał klasę numer 12, czy numer 2 lub 3. Pomimo wielu nieprawidłowych predykcji wyniki są dość obiecujące.

Funkcja strat dla danych treningowych i testowych:



Rysunek 10: Funkcje strat

Funkcje znajdują sie stosunkowo blisko siebie, co jest dobrym sygnałem - wskazuje na brak przetrenowania. Brak wypłaszczenia na koniec sugeruje, że być może trening powinien zostać delikatnie wydłużony, jednak ze względu na jego czasochłonność, nie zostało to sprawdzone.



Rysunek 11: Metryka accuracy

Metryka accuracy prezentuje się również obiecująco, widać tendencję rosnącą.

6.1 Raport klasyfikacji

Klasa	Precision	Recall	F1-score	Support
0	0.72	0.57	0.64	102
1	0.59	0.79	0.67	89
2	0.35	0.43	0.39	102
3	0.35	0.41	0.38	101
4	0.79	0.72	0.75	116
5	0.48	0.70	0.57	104
6	0.33	0.28	0.30	93
7	0.81	0.66	0.73	94
8	0.34	0.28	0.31	96
9	0.53	0.52	0.53	98
10	0.45	0.25	0.33	114
11	0.42	0.34	0.37	94
12	0.31	0.45	0.37	92
13	0.50	0.54	0.52	110
14	0.64	0.53	0.58	95
Accuracy: 0.50 (na 1500 próbkach)				
Średnia makro	0.51	0.50	0.49	1500
Średnia ważona	0.51	0.50	0.50	1500

Tabela 1: Miary jakości klasyfikacji dla poszczególnych klas

- **Precision (precyzja)** – określa, jaki procent przykładów zaklasyfikowanych do danej klasy rzeczywiście do niej należy. Wysoka precyzja oznacza niski odsetek fałszywych alarmów (False Positives).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall (czułość)** – mierzy, jaki procent przykładów należących do danej klasy został poprawnie rozpoznany przez model. Wysoka czułość oznacza, że klasyfikator wykrywa większość przypadków tej klasy.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1-score** – średnia harmoniczna precyzji i czułości, stanowi kompromis między tymi dwoma miarami.

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Support** – liczba przykładów danej klasy w zbiorze testowym. Pokazuje, na ilu przypadkach oparta jest ocena klasyfikatora dla każdej klasy.

- **Accuracy (dokładność)** – ogólny odsetek poprawnych predykcji względem wszystkich przykładów. Może być niewiarygodna w przypadku niezrównoważonych klas.

$$\text{Accuracy} = \frac{\text{Liczba\_poprawnych\_predykcji}}{\text{Liczba\_wszystkich\_przykładow}}$$

- **Macro avg (średnia makro)** – średnia arytmetyczna metryk (precision, recall, F1-score) obliczonych dla każdej klasy, bez względu na liczność klas.
- **Weighted avg (średnia ważona)** – średnia metryk ważona według liczności klas. Klasy z większą liczbą przykładów mają większy wpływ na wynik końcowy.

7 Podsumowanie i wnioski

Nie do końca udało się zrealizować pierwotne założenia. Jednym z większych niepowodzeń jest niesfinalizowanie połączenia funkcji czytania tablic rejestracyjnych z ostatecznym modelem (wynikało to między innymi z przekroju danych - jedynie mała część zdjęć zawierała tablice rejestracyjne). Nie zrealizowano również planu wektoryzacji - zostało to zamienione na standardowe podejście z użyciem warstw konwolucyjnych. Ze względu na ograniczone zasoby nie udało się również wytrenować modelu na przygotowanym zestawie danych - został wykorzystany jedynie jego mały fragment.

Główne problemy:

- Trudność w znalezieniu zbioru nagrań - większość znalezionych przez nas datasetów opierało się na zdjęciach z dashacamów, a nie na materiałach wideo. Model został wytrenowany w ten sposób aby przyjmował zdjęcia, pojedyńcze klatki. Utworzony skrypt pozwala na konkwerję materiału wideo, na zdjęcie odpowiadające modelowi.
- Problemy ze znalezieniem zbioru danych z widocznymi tablicami rejestracyjnymi - pierwotnym celem projektu była ekstrakcja cech takich jak napisy na tablicach rejestracyjnych, czy znaki drogowe. Nastąpiła próba wytrenowania modeli wyszukujących te elementy na zdjęciach. Jednak w używanych przez nas danych, było tych cech tak mało, że używanie ich było bezcelowe.
- Mała dywersyfikacja krajów (zazwyczaj zbiory opierały się na jednym kraju).
- Trudności w połączeniu przygotowanych danych i modelu.
- Problem stanowi również testowanie - pozytywne rezultaty są osiągane głównie dla danych pochodzących z użytego zestawu danych. Zauważono mniejszą skuteczność dla danych spoza zbioru Mappilary.

Udało się natomiast stworzyć model, który da się wykorzystać do predykcji kraju na podstawie nagrań. Stworzony został skrypt, który dzieli nagrania na klatki, na których odbywa się predykcja. Główny cel projektu został więc zrealizowany.

Literatura

[1] *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html*, scikit-learn

[2] *https://www.geeksforgeeks.org/confusion-matrix-machine-learning/*, confusion matrix

[3] *https://docs.pytorch.org/docs/stable/index.html*, pytorch

[4] *https://medium.com/@mahijain9211/license-plate-detection-from-video-files-using-yolo-and-easyocr-6b647f0c94d5*, License Plate Detection from Video Files Using YOLO and EasyOCR

[5] *https://www.kaggle.com/datasets/noepinefrin/yolov8-plate-detection-model-and-fine-tuned?resource=download*, YOLOv8 | Plate Detection Model & Finetuned weights

[6] *https://www.mapillary.com/dataset/places*, Mappilary Dataset