|  |  |  |
| --- | --- | --- |
| Podstawy sztucznej inteligencji | Łukasz Burliga  Adam Borek | Temat projektu: Klasyfikacja symboli w języku migowym |

**Przygotowanie danych wejściowych**

W celu przetrenowania modelu rozpoznającego gesty w amerykańskim języku migowym (ASL), przygotowano dedykowany pipeline. Proces ten obejmuje:

1. **Pobieranie danych**

Funkcja **download\_and\_prepare\_data()** pobiera dane z repozytorium Kaggle (korzystając z kagglehub) i rozpakowuje je lokalnie do folderu **data/**

def download\_and\_prepare\_data(data\_dir="data"):  
 *"""Pobiera dane z repozytorium american-sign-language-dataset"""* dataset\_path = kagglehub.dataset\_download("esfiam/american-sign-language-dataset")  
 print(f"Pobrano dane do cache: {dataset\_path}")  
  
 # Skopiuj wszystko z cache do ./data  
 if os.path.exists(data\_dir):  
 shutil.rmtree(data\_dir)  
 os.makedirs(data\_dir, exist\_ok=True)  
  
 # Znajdź folder ASL\_Gestures\_36\_Classes w pobranych danych  
 source\_root = Path(dataset\_path)  
 asl\_folder = None  
 for item in source\_root.iterdir():  
 if item.is\_dir() and "ASL\_Gestures" in item.name:  
 asl\_folder = item  
 break  
  
 if not asl\_folder:  
 raise RuntimeError("Nie znaleziono folderu ASL\_Gestures\_36\_Classes w danych Kaggle!")  
  
 # Przenieś zawartość tego folderu do ./data  
 for item in asl\_folder.iterdir():  
 dest = Path(data\_dir) / item.name  
 if item.is\_dir():  
 shutil.copytree(item, dest)  
 else:  
 shutil.copy2(item, dest)  
  
 print(f"Skopiowano dane do: {data\_dir}")

1. **Scalanie danych treningowych i testowych**

Funkcja **merge\_to\_temp()** scala oryginalny podział na **train/** i **test/** w jeden zbiór tymczasowy **temp/**, pozwalając na nowy, bardziej kontrolowany podział później.

def merge\_to\_temp(data\_dir="data", temp\_dir="temp"):  
 *"""Scala dane z folderów train i test do folderu temp/"""* train\_dir = Path(data\_dir) / "train"  
 test\_dir = Path(data\_dir) / "test"  
  
 os.makedirs(temp\_dir, exist\_ok=True)  
  
 for split\_dir in [train\_dir, test\_dir]:  
 for class\_dir in split\_dir.iterdir():  
 if class\_dir.is\_dir():  
 class\_name = class\_dir.name  
 dest\_class\_dir = Path(temp\_dir) / class\_name  
 dest\_class\_dir.mkdir(parents=True, exist\_ok=True)  
  
 for img\_file in class\_dir.iterdir():  
 if img\_file.is\_file():  
 dest\_file = dest\_class\_dir / img\_file.name  
 if dest\_file.exists():  
 dest\_file = dest\_class\_dir / f"{img\_file.stem}\_from\_{split\_dir.name}{img\_file.suffix}"  
 shutil.copy(img\_file, dest\_file)  
  
 print(f"Dane scalone do folderu: {temp\_dir}/")  
  
 # Usuń stare foldery train/test  
 for old\_split in ["train", "test"]:  
 old\_path = Path(data\_dir) / old\_split  
 if old\_path.exists():  
 shutil.rmtree(old\_path)

1. **Augmentacja danych**

Funkcja **augment\_dataset()** zwiększa ilość danych poprzez losowe przekształcenia obrazów, takie jak:

* 1. obrót (do ±15°),
  2. zmiana jasności i kontrastu,
  3. odbicie lustrzane.

Wszystko po to, by uodpornić model na różnorodność danych wejściowych.

def augment\_dataset(temp\_dir="temp", target\_count\_per\_class=140):  
 *"""Augmentuje dane zwiększając ich ilość"""* temp\_dir = Path(temp\_dir)  
  
 # Przykładowe augmentacje (można rozszerzyć)  
 augment\_ops = transforms.Compose([  
 transforms.RandomHorizontalFlip(p=0.5),  
 transforms.RandomRotation(degrees=15),  
 transforms.ColorJitter(brightness=0.3, contrast=0.3),  
 ])  
  
 for class\_dir in temp\_dir.iterdir():  
 if not class\_dir.is\_dir():  
 continue  
  
 image\_files = list(class\_dir.glob("\*.jpeg"))  
 current\_count = len(image\_files)  
 needed = target\_count\_per\_class - current\_count  
  
 if needed <= 0:  
 continue # ta klasa już ma wystarczająco  
  
 print(f"Augmentuję klasę '{class\_dir.name}': {current\_count} -> {target\_count\_per\_class}")  
  
 i = 0  
 while i < needed:  
 # losowy obraz z oryginałów  
 src\_img\_path = random.choice(image\_files)  
 with Image.open(src\_img\_path) as img:  
 img = img.convert("RGB") # na wszelki wypadek  
  
 # wykonaj augmentację  
 augmented = augment\_ops(img)  
  
 # nowa nazwa pliku  
 new\_filename = f"{src\_img\_path.stem}\_aug{i}{src\_img\_path.suffix}"  
 new\_path = class\_dir / new\_filename  
  
 augmented.save(new\_path)  
 i += 1  
  
 print("Augmentacja zakończona.")

1. **Nowy podział zbioru danych**

Funkcja **split\_dataset\_from\_temp()** dzieli dane z folderu temp/ na:

* 1. zbiór treningowy (80%),
  2. zbiór walidacyjny (10%),
  3. zbiór testowy (10%).

1. def split\_dataset\_from\_temp(temp\_dir="temp", output\_dir="data", split=(0.8, 0.1, 0.1)):  
    *"""Dzieli dane na zbiór treningowy, walidacyjny i testowy"""* temp\_dir = Path(temp\_dir)  
    output\_dir = Path(output\_dir)  
     
    # Nowe foldery: train, val, test  
    for split\_name in ["train", "val", "test"]:  
    (output\_dir / split\_name).mkdir(parents=True, exist\_ok=True)  
     
    for class\_dir in temp\_dir.iterdir():  
    if not class\_dir.is\_dir():  
    continue  
     
    class\_name = class\_dir.name  
    all\_images = list(class\_dir.glob("\*.jpeg"))  
    random.shuffle(all\_images)  
     
    total = len(all\_images)  
    train\_end = int(split[0] \* total)  
    val\_end = train\_end + int(split[1] \* total)  
     
    splits = {  
    "train": all\_images[:train\_end],  
    "val": all\_images[train\_end:val\_end],  
    "test": all\_images[val\_end:]  
    }  
     
    for split\_name, files in splits.items():  
    dest\_dir = output\_dir / split\_name / class\_name  
    dest\_dir.mkdir(parents=True, exist\_ok=True)  
     
    for file\_path in files:  
    shutil.copy(file\_path, dest\_dir)  
     
    print(  
    f"{class\_name}: {total} obrazków -> train:{len(splits['train'])}, val:{len(splits['val'])}, test:{len(splits['test'])}")  
     
    print("Podział danych zakończony.")  
     
    # Usuń folder temp po zakończeniu  
    if temp\_dir.exists():  
    shutil.rmtree(temp\_dir)  
    print(f"Usunięto folder tymczasowy: {temp\_dir}")

**Analiza rozkładu klas**

Po przetworzeniu i podzieleniu danych wykonano analizę liczby przykładów w każdej klasie, osobno dla zbiorów:

* treningowego,
* walidacyjnego,
* testowego.

Do tego celu użyto funkcji **analyze\_balance()**, która:

* zlicza liczbę obrazów w każdej klasie (**count\_images\_per\_class()**),
* generuje wykresy kołowe ilustrujące proporcje (**plot\_distribution()**),
* wypisuje wyniki w terminalu.
* def download\_and\_prepare\_data(data\_dir="data"):  
   *"""Pobiera dane z repozytorium american-sign-language-dataset"""* dataset\_path = kagglehub.dataset\_download("esfiam/american-sign-language-dataset")  
   print(f"Pobrano dane do cache: {dataset\_path}")  
    
   # Skopiuj wszystko z cache do ./data  
   if os.path.exists(data\_dir):  
   shutil.rmtree(data\_dir)  
   os.makedirs(data\_dir, exist\_ok=True)  
    
   # Znajdź folder ASL\_Gestures\_36\_Classes w pobranych danych  
   source\_root = Path(dataset\_path)  
   asl\_folder = None  
   for item in source\_root.iterdir():  
   if item.is\_dir() and "ASL\_Gestures" in item.name:  
   asl\_folder = item  
   break  
    
   if not asl\_folder:  
   raise RuntimeError("Nie znaleziono folderu ASL\_Gestures\_36\_Classes w danych Kaggle!")  
    
   # Przenieś zawartość tego folderu do ./data  
   for item in asl\_folder.iterdir():  
   dest = Path(data\_dir) / item.name  
   if item.is\_dir():  
   shutil.copytree(item, dest)  
   else:  
   shutil.copy2(item, dest)  
    
   print(f"Skopiowano dane do: {data\_dir}")  
    
    
  def merge\_to\_temp(data\_dir="data", temp\_dir="temp"):  
   *"""Scala dane z folderów train i test do folderu temp/"""* train\_dir = Path(data\_dir) / "train"  
   test\_dir = Path(data\_dir) / "test"  
    
   os.makedirs(temp\_dir, exist\_ok=True)  
    
   for split\_dir in [train\_dir, test\_dir]:  
   for class\_dir in split\_dir.iterdir():  
   if class\_dir.is\_dir():  
   class\_name = class\_dir.name  
   dest\_class\_dir = Path(temp\_dir) / class\_name  
   dest\_class\_dir.mkdir(parents=True, exist\_ok=True)  
    
   for img\_file in class\_dir.iterdir():  
   if img\_file.is\_file():  
   dest\_file = dest\_class\_dir / img\_file.name  
   if dest\_file.exists():  
   dest\_file = dest\_class\_dir / f"{img\_file.stem}\_from\_{split\_dir.name}{img\_file.suffix}"  
   shutil.copy(img\_file, dest\_file)  
    
   print(f"Dane scalone do folderu: {temp\_dir}/")  
    
   # Usuń stare foldery train/test  
   for old\_split in ["train", "test"]:  
   old\_path = Path(data\_dir) / old\_split  
   if old\_path.exists():  
   shutil.rmtree(old\_path)  
    
    
  def augment\_dataset(temp\_dir="temp", target\_count\_per\_class=140):  
   *"""Augmentuje dane zwiększając ich ilość"""* temp\_dir = Path(temp\_dir)  
    
   # Przykładowe augmentacje (można rozszerzyć)  
   augment\_ops = transforms.Compose([  
   transforms.RandomHorizontalFlip(p=0.5),  
   transforms.RandomRotation(degrees=15),  
   transforms.ColorJitter(brightness=0.3, contrast=0.3),  
   ])  
    
   for class\_dir in temp\_dir.iterdir():  
   if not class\_dir.is\_dir():  
   continue  
    
   image\_files = list(class\_dir.glob("\*.jpeg"))  
   current\_count = len(image\_files)  
   needed = target\_count\_per\_class - current\_count  
    
   if needed <= 0:  
   continue # ta klasa już ma wystarczająco  
    
   print(f"Augmentuję klasę '{class\_dir.name}': {current\_count} -> {target\_count\_per\_class}")  
    
   i = 0  
   while i < needed:  
   # losowy obraz z oryginałów  
   src\_img\_path = random.choice(image\_files)  
   with Image.open(src\_img\_path) as img:  
   img = img.convert("RGB") # na wszelki wypadek  
    
   # wykonaj augmentację  
   augmented = augment\_ops(img)  
    
   # nowa nazwa pliku  
   new\_filename = f"{src\_img\_path.stem}\_aug{i}{src\_img\_path.suffix}"  
   new\_path = class\_dir / new\_filename  
    
   augmented.save(new\_path)  
   i += 1  
    
   print("Augmentacja zakończona.")  
    
    
  def split\_dataset\_from\_temp(temp\_dir="temp", output\_dir="data", split=(0.8, 0.1, 0.1)):  
   *"""Dzieli dane na zbiór treningowy, walidacyjny i testowy"""* temp\_dir = Path(temp\_dir)  
   output\_dir = Path(output\_dir)  
    
   # Nowe foldery: train, val, test  
   for split\_name in ["train", "val", "test"]:  
   (output\_dir / split\_name).mkdir(parents=True, exist\_ok=True)  
    
   for class\_dir in temp\_dir.iterdir():  
   if not class\_dir.is\_dir():  
   continue  
    
   class\_name = class\_dir.name  
   all\_images = list(class\_dir.glob("\*.jpeg"))  
   random.shuffle(all\_images)  
    
   total = len(all\_images)  
   train\_end = int(split[0] \* total)  
   val\_end = train\_end + int(split[1] \* total)  
    
   splits = {  
   "train": all\_images[:train\_end],  
   "val": all\_images[train\_end:val\_end],  
   "test": all\_images[val\_end:]  
   }  
    
   for split\_name, files in splits.items():  
   dest\_dir = output\_dir / split\_name / class\_name  
   dest\_dir.mkdir(parents=True, exist\_ok=True)  
    
   for file\_path in files:  
   shutil.copy(file\_path, dest\_dir)  
    
   print(  
   f"{class\_name}: {total} obrazków -> train:{len(splits['train'])}, val:{len(splits['val'])}, test:{len(splits['test'])}")  
    
   print("Podział danych zakończony.")  
    
   # Usuń folder temp po zakończeniu  
   if temp\_dir.exists():  
   shutil.rmtree(temp\_dir)  
   print(f"Usunięto folder tymczasowy: {temp\_dir}")

**Przykładowy wynik w terminalu:**

**Obraz zawierający tekst, Czcionka, zrzut ekranu, design

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.Obraz zawierający tekst, Czcionka, zrzut ekranu, design

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.Obraz zawierający tekst, Czcionka, zrzut ekranu, design

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

**Przykładowy wykres:**

**Obraz zawierający tekst, Wielobarwność, krąg, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

**Serializacja i ładowanie danych do modelu**

W celu przyspieszenia wczytywania oraz umożliwienia elastycznego przetwarzania danych obrazowych, zastosowano proces serializacji z użyciem formatu pickle.

1. **Transformacje danych**

Przed trenowaniem dane są przekształcane przy pomocy **torchvision.transforms**. Dla zbioru treningowego stosowane są dodatkowe augmentacje (np. obrót, jasność), natomiast walidacyjny i testowy są tylko standaryzowane.

# === Transformacje ===  
train\_transform = transforms.Compose([  
 transforms.RandomHorizontalFlip(),  
 transforms.RandomRotation(10),  
 transforms.ColorJitter(brightness=0.2, contrast=0.2),  
 transforms.ToTensor(),  
 transforms.Normalize(mean=[0.5], std=[0.5])  
])  
  
test\_transform = transforms.Compose([  
 transforms.ToTensor(),  
 transforms.Normalize(mean=[0.5], std=[0.5])  
])

1. **Serializacja zbiorów**

Funkcja **serialize\_dataset()** wczytuje wszystkie obrazy z ImageFolder i zapisuje je jako osobne pliki **.pkl**, zawierające:

* 1. obraz (**PIL.Image**),
  2. etykietę numeryczną.

Pliki są zapisywane osobno, dzięki czemu można je potem wczytywać *on-the-fly* bez konieczności ładowania całego datasetu do RAM-u.

# === Serializacja ===  
def serialize\_dataset(imagefolder\_root, output\_path):  
 dataset = ImageFolder(imagefolder\_root)  
 os.makedirs(output\_path, exist\_ok=True)  
  
 for idx, (img\_path, label) in enumerate(dataset.samples):  
 img = Image.open(img\_path).convert("RGB")  
 data = {  
 "image": img,  
 "label": label  
 }  
 file\_name = f"{Path(img\_path).stem}\_{idx}.pkl"  
 with open(os.path.join(output\_path, file\_name), "wb") as f:  
 pickle.dump(data, f)  
  
 print(f"Zserializowano: {len(dataset.samples)} plików → {output\_path}")

**Przykład wyjścia:**

**Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

1. **Niestandardowy dataset**

Klasa **SerializedASLDataset** implementuje interfejs **torch.utils.data.Dataset** i pozwala na wygodne wczytywanie serializowanych danych. Dzięki temu można stosować DataLoader, który pobiera dane w batchach.

# === Dataset ===  
class SerializedASLDataset(Dataset):  
 def \_\_init\_\_(self, serialized\_path, transform=None):  
 self.serialized\_path = serialized\_path  
 self.data\_files = sorted(os.listdir(serialized\_path))  
 self.transform = transform  
  
 def \_\_len\_\_(self):  
 return len(self.data\_files)  
  
 def \_\_getitem\_\_(self, idx):  
 with open(os.path.join(self.serialized\_path, self.data\_files[idx]), "rb") as f:  
 data = pickle.load(f)  
 img = data["image"]  
 label = data["label"]  
  
 if self.transform:  
 img = self.transform(img)  
  
 return img, label

1. **Dataloadery**

Funkcja **get\_serialized\_dataloaders()** tworzy trzy loadery:

* 1. **train\_loader** – z szuflowaniem,
  2. **val\_loader** i **test\_loader** – bez.

def get\_serialized\_dataloaders(root="serialized", batch\_size=32):  
 train\_dataset = SerializedASLDataset(f"{root}/train", transform=train\_transform)  
 val\_dataset = SerializedASLDataset(f"{root}/val", transform=test\_transform)  
 test\_dataset = SerializedASLDataset(f"{root}/test", transform=test\_transform)  
  
 train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  
 val\_loader = DataLoader(val\_dataset, batch\_size=batch\_size, shuffle=False)  
 test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)  
  
 return train\_loader, val\_loader, test\_loader

1. **Wizualizacja danych**

Funkcja **show\_batch()** pozwala na szybki podgląd przykładowych obrazów z każdego zbioru. To przydatne w celu ręcznej walidacji, czy dane są poprawnie wczytywane i przetwarzane.

# === Podgląd batcha ===  
def show\_batch(data\_loader, class\_names=None, title=""):  
 images, labels = next(iter(data\_loader))  
 grid = torchvision.utils.make\_grid(images[:8], nrow=4, normalize=True)  
  
 plt.figure(figsize=(10, 5))  
 plt.imshow(grid.permute(1, 2, 0))  
 plt.title(f"{title} – Przykładowe obrazy")  
 plt.axis("off")  
 print("Etykiety:", labels[:8].tolist())  
 plt.show()

Obraz zawierający palec, Ciało/mięso, osoba, dłoń

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Obraz zawierający Ciało/mięso, palec, paznokieć, Język migowy

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Obraz zawierający Ciało/mięso, paznokieć, osoba

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Etykiety odpowiadające obrazom powyżej:

Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

**Podsumowanie etapu przygotowania danych:**

W projekcie wykorzystano dobrze przygotowany zbiór *ASL\_Gestures\_36\_Classes* z serwisu Kaggle, który zawiera równo rozłożone dane między klasy oraz spójny format obrazów. W ramach przygotowania danych przeprowadzono podział na trzy zbiory: treningowy (80%), walidacyjny (10%) i testowy (10%). Dodatkowo zastosowano augmentację zwiększającą różnorodność treningowego zbioru, standaryzację obrazów oraz serializację danych dla efektywnego ładowania. Dane zostały poddane analizie rozkładu klas, a ich poprawność i zbalansowanie zweryfikowano za pomocą wykresów i podglądu batchy. Nie było konieczności usuwania przypadków odstających, gdyż zbiór źródłowy był kompletny i dobrze oczyszczony.