KAMIEŃ MILOWY NR 4

**evaluate.py** – kod do oceny i wizualizacji modelu klasyfikacji obrazów. W jego skład wchodzą funkcje odpowiadające za:

* Ocenę modelu
* Zapis metryk
* Wizualizację wyników (lossy, predykcje, błędne przykłady)
* Obsługę klas na podstawie struktury folderów

Etykietowanie i metryki:

def get\_class\_names(path="data/train"):

* dataset = ImageFolder(path)  # Tworzy zbior danych obrazów z folderów, gdzie każdy folder to jedna klasa
* return dataset.classes  # Zwraca listę nazw klas (czyli nazw folderów)

Rysowanie przebiegu funkcji kosztu dla treningu i walidacji w kolejnych epokach:

def plot\_training\_curves(train\_losses, val\_losses):

    plt.figure(figsize=(8, 5))  # Tworzy wykres o zadanym rozmiarze

    plt.plot(train\_losses, label='Training Loss')  # Rysuje stratę z treningu

    plt.plot(val\_losses, label='Validation Loss')  # Rysuje stratę z walidacji

    plt.xlabel('Epoch')  # Oś X to epoki

    plt.ylabel('Loss')  # Oś Y to wartość funkcji kosztu

    plt.legend()  # Dodaje legendę

    plt.title('Funkcja kosztu (Loss)')  # Tytuł wykresu

    plt.show()  # Wyświetlenie wykresu

A graph of a graph

AI-generated content may be incorrect.

Opis metryki:  
Funkcja kosztu (loss) mówi o tym, jak bardzo prognozy modelu odbiegają od prawdziwych wartości – im niższy loss, tym lepsze dopasowanie sieci. Na wykresie widzimy dwie krzywe: jedną dla danych treningowych, drugą dla walidacyjnych.

Podsumowanie wyników:  
Już w pierwszej epoce loss treningowy spada bardzo gwałtownie z wysokich wartości do ok. 80, a następnie konsekwentnie maleje do około 15–20 pod koniec treningu. Również loss walidacyjny szybko schodzi z ok. 10 do wartości bliskich zera i utrzymuje się na bardzo niskim poziomie przez kolejne epoki. Taki przebieg oznacza, że model szybko uczy się rozpoznawać wzorce w danych i dobrze uogólnia na zestaw walidacyjny – widoczna jest brakująca albo minimalna różnica między treningiem a walidacją, co sugeruje niewielkie ryzyko nadmiernego dopasowania. Na dole wykresu widać, że dokładność na zbiorze testowym osiągnęła 97,22 %.

Wykres słupkowy precision / recall / f1-score per klasa:

def plot\_classification\_metrics(report\_dict, class\_names):

    metrics = ['precision', 'recall', 'f1-score']  # Interesujące metryki

    values = {metric: [] for metric in metrics}

    for cls in class\_names:  # Dla każdej klasy

        for metric in metrics:

            values[metric].append(report\_dict[cls][metric])

    x = np.arange(len(class\_names))  # Pozycje na osi X

    width = 0.25  # Szerokość słupków

    plt.figure(figsize=(14, 6))

    plt.bar(x - width, values['precision'], width, label='Precision')

    plt.bar(x, values['recall'], width, label='Recall')

    plt.bar(x + width, values['f1-score'], width, label='F1-Score')

    plt.xlabel('Klasa')

    plt.ylabel('Wartość')

    plt.title('Precision / Recall / F1-score per klasa')

    plt.xticks(ticks=x, labels=class\_names, rotation=90)

    plt.legend()

    plt.tight\_layout()

    plt.show()

A graph of different colored lines

AI-generated content may be incorrect.

Opis metryk:  
Precyzja pokazuje, na ile trafne są przewidywania modelu dla danej klasy – czyli spośród wszystkich sytuacji, w których model powiedział “to jest znak X”, ile z nich rzeczywiście było oznaczone poprawnie. Czułość informuje, jak często model potrafi wykryć wszystkie wystąpienia danej klasy – czyli spośród wszystkich rzeczywistych przykładów X, ile z nich model zauważył. F1-score to takie uśrednienie tych dwóch wskaźników, które odzwierciedla ich kompromis.

Podsumowanie wyników:  
Model radzi sobie bardzo dobrze – dla zdecydowanej większości znaków zarówno precyzja, jak i czułość są blisko 1, co oznacza niewiele fałszywych trafień i niewiele pominiętych przypadków. Wyjątkiem jest cyfra „5”, dla której czułość jest nieco niższa, co sugeruje, że czasami bywa mylona z innymi gestami. Z kolei takie litery jak „c”, „d”, „h”, „k”, „m” czy „t” osiągają niemal idealne wyniki we wszystkich trzech metrykach, co wskazuje na ich wyraźne i jednoznaczne wzorce w danych testowych.

Zapis metryk do pliku .csv:

def save\_metrics\_to\_csv(report\_dict, class\_names, path="results/classification\_metrics.csv"):

    metrics = ['precision', 'recall', 'f1-score']

    os.makedirs(os.path.dirname(path), exist\_ok=True)  # Tworzy folder jeśli nie istnieje

    with open(path, mode='w', newline='') as file:  # Otwiera plik CSV do zapisu

        writer = csv.writer(file)

        writer.writerow(['class'] + metrics)  # Nagłówki kolumn

        for cls in class\_names:

            row = [cls] + [report\_dict[cls][m] for m in metrics]  # Dane dla jednej klasy

            writer.writerow(row)

    print(f"Zapisano metryki do pliku: {path}")

Ewaluacja modelu:

def evaluate\_model(model, test\_loader, device="cpu"):

    model.eval()  # Przełączenie modelu w tryb ewaluacji

    correct = 0

    total = 0

    all\_labels = []

    all\_predicted = []

    with torch.no\_grad():  # Bez śledzenia gradientów

        for inputs, labels in test\_loader:

            inputs, labels = inputs.to(device), labels.to(device)  # Przeniesienie na urządzenie

            outputs = model(inputs)  # Przewidywania

            \_, predicted = torch.max(outputs, 1)  # Klasa z najwyższym prawdopodobieństwem

            total += labels.size(0)  # Zliczanie całkowitej liczby przykładów

            correct += (predicted == labels).sum().item()  # Zliczanie poprawnych

            all\_labels.extend(labels.cpu().numpy())

            all\_predicted.extend(predicted.cpu().numpy())

    accuracy = correct / total

    print(f"\nDokładność (Accuracy): {accuracy \* 100:.2f}%\n")

    class\_names = get\_class\_names()

    report = classification\_report(all\_labels, all\_predicted, target\_names=class\_names, output\_dict=True)

    print("Raport klasyfikacji:")

    print(classification\_report(all\_labels, all\_predicted, target\_names=class\_names))

    plot\_classification\_metrics(report, class\_names)  # Wykresy precision/recall/f1

    save\_metrics\_to\_csv(report, class\_names)  # Zapis CSV

    cm = confusion\_matrix(all\_labels, all\_predicted)  # Macierz pomyłek

    plt.figure(figsize=(10, 8))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class\_names, yticklabels=class\_names)

    plt.xlabel('Predykcja')

    plt.ylabel('Rzeczywista')

    plt.title('Macierz Pomyłek')

    plt.tight\_layout()

    plt.show()

A screenshot of a graph

AI-generated content may be incorrect.

Ta macierz pokazuje, jak często poszczególne gesty zostały sklasyfikowane poprawnie (liczby na głównej przekątnej) i gdzie zdarzyły się pomyłki (poza przekątną). Większość klas trafiona jest idealnie – dla niemal wszystkich znaków wszystkie 14 próbek trafiło we właściwe miejsce. Pojawiło się tylko kilka drobnych odchyleń:

* Gest „1” raz został uznany za „8”.
* Gest „7” dwa razy wpadł w klasę „q”.
* Litera „n” przytrafiły się trzy pomyłki: dwa razy jako „o” i raz jako „r”.
* Ponadto jedna próbka „x” trafiła na „y”.

Takie pojedyncze przypadki pokazują, które znaki mają między sobą pewne podobieństwa, ale ogólnie model radzi sobie niemal bezbłędnie.

Przykładowe obrazki z predykcjami modelu:

def visualize\_predictions(model, test\_loader, device="cpu", show\_only\_errors=False, max\_samples=10):

    model.eval()

    class\_names = get\_class\_names()

    data\_iterator = iter(test\_loader)

    images, labels = next(data\_iterator)

    images = images.to(device)

    labels = labels.to(device)

    with torch.no\_grad():

        outputs = model(images)

        \_, predictions = torch.max(outputs, 1)

    images = images.cpu()

    labels = labels.cpu()

    predictions = predictions.cpu()

    all\_samples = list(zip(images, labels, predictions))

    if show\_only\_errors:

        all\_samples = [sample for sample in all\_samples if sample[1] != sample[2]]

    random.shuffle(all\_samples)

    all\_samples = all\_samples[:max\_samples]

    num\_samples = len(all\_samples)

    cols = 5

    rows = (num\_samples + cols - 1) // cols

    fig, axs = plt.subplots(rows, cols, figsize=(cols \* 3.5, rows \* 3.5))

    if rows == 1:

        axs = [axs]

    for i in range(rows \* cols):

        ax = axs[i // cols][i % cols] if rows > 1 else axs[i % cols]

        if i < num\_samples:

            image, true\_label, predicted\_label = all\_samples[i]

            image = image.numpy().transpose(1, 2, 0)

            image = (image \* 0.5) + 0.5

            is\_correct = true\_label == predicted\_label

            color = 'green' if is\_correct else 'red'

            ax.imshow(image)

            ax.set\_title(f'True: {class\_names[true\_label]}\nPred: {class\_names[predicted\_label]}', color=color, fontsize=10)

        else:

            ax.axis('off')

        ax.axis('off')

    plt.tight\_layout(h\_pad=3.0)

    plt.show()

A collage of hands making gestures

AI-generated content may be incorrect.

Obrazki z błędnie sklasyfikowanymi przykładami ze zbioru testowego:

A collage of hands with different gestures

AI-generated content may be incorrect.

Jak można poprawić model?

**Więcej danych** – zwłaszcza dla klas z niskim recall.

**Augmentacja danych** – np. transforms.RandomRotation, RandomCrop, ColorJitter, itp.

**Lepsza architektura** – np. ResNet, EfficientNet, zamiast prostych CNN.

**Wyrównanie klas** **(class balancing)** – np. WeightedRandomSampler, class weights w funkcji kosztu.

**Fine-tuning pretrenowanego modelu** – np. torchvision.models.resnet18(pretrained=True).

**Regularizacja** – np. Dropout, L2, BatchNorm.

**Hyperparameter tuning** – learning rate, batch size, liczba warstw itd