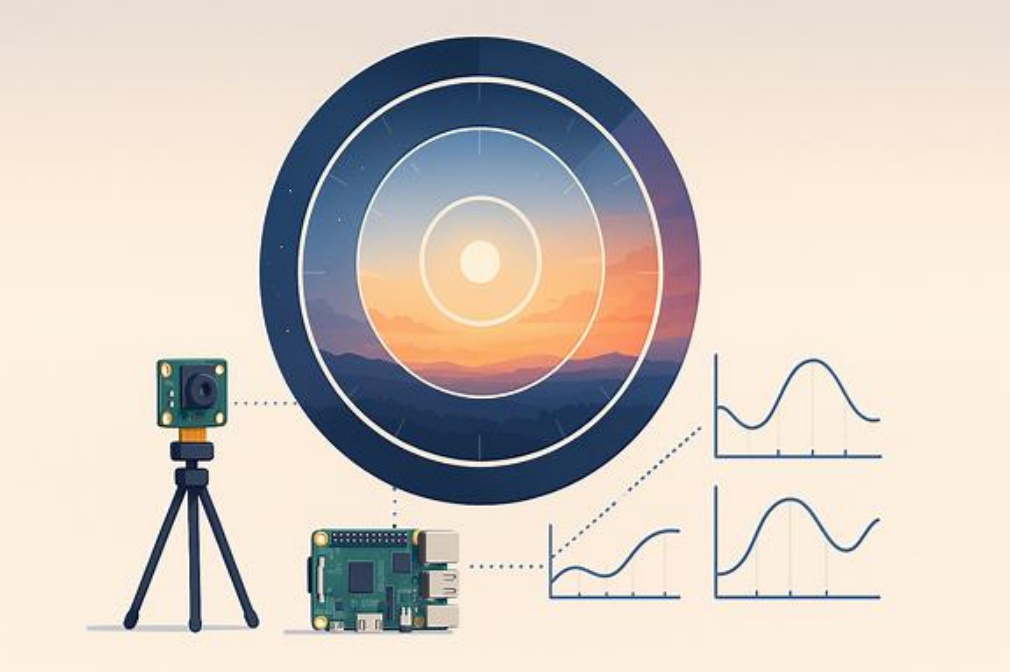
PROJEKT

Zegar Biologiczny

Uczenie Maszynowe



Na wykonanie tego projektu wraz z jego opracowaniem poświęciłem wiele czasu i pracy, jednak został on w około 90% zrealizowany z wykorzystaniem współczesnych dużych modeli językowych, takich jak ChatGPT czy Copilot dostępny w przeglądarce Edge. Całość kodu oraz towarzyszącej dokumentacji (komentarze w kodzie / pliki Markdown) powstała natomiast we współpracy z narzędziem GitHub Copilot i modelami dostępnymi w płatnej wersji tego narzędzia."

Tarnów 2025/2026

Autor: Artur Kwiek

1. Wprowadzenie

Projekt „Zegar biologiczny” miał na celu stworzenie rozwiązania software’owego, które na podstawie pojedynczego obrazu sceny zewnętrznej potrafi oszacować porę doby (godzinę) w sposób zgodny z cykliczną naturą czasu, z oczekiwaną dokładnością na poziomie +-1godz. Rozwiązanie łączy klasyczne przetwarzanie obrazu z metodami uczenia maszynowego, obejmując pełen proces: od akwizycji danych, przez przygotowanie cech i trening modeli, aż po ich uruchomienie w środowisku rzeczywistym (kamera + overlay szacowanej godziny).

2. Repozytorium i dane

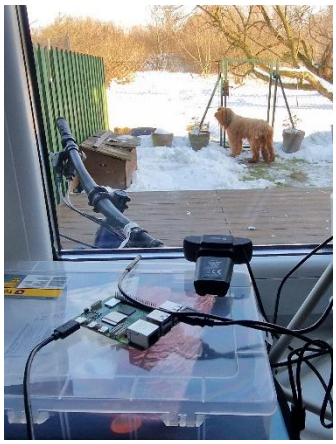
Repozytorium zawierające pełny kod – skrypty obliczeniowe/ uruchomieniowe i akwizycyjne dostępne są pod adresem

https://github.com/arturkwiek/ZegarBiologiczny

3. Struktura i Zawartość

Projekt został stworzony na bazie kilku spójnych modułów, tworzących kompletny łańcuch przetwarzania (swoisty pipeline obliczeniowo-treningowy), które opisano w kolejnych punktach.

3.1. Akwizycja



Jedno ze zdjęć z utworzonego datasetu

Część akwizycyjna polegała na pozyskiwaniu obrazów z kamery i ich trwałym zapisie na dysku. W ten sposób na przestrzeni krótkiego okresu czasu trwającego kilka dni stworzony został bazowy dataset złożony z około 500 tysięcy zdjęć.

W ramach teego etapu opracowano skrypt do akwizycji danych:

- **MLDailyHourClock.py** - Skrypt do ciągłego zbierania obrazów z kamery podłączonej do portu USB na minikomputerze RaspBerry Pi 5 i zapisywania ich w strukturze katalogów.
- Przyjęta struktura katlogów miała następującą postać:
  dataset/YYYY/MM/DD/HH
- Nazwy kolejnych plików obrazu tworzone były na podstawie bieżącego czasu, zgodnie ze wzorcem
  %**Y%m%d_%H%M%S**].jpg
  Kolejne zdjęcia zapisywane były w wyżej opisanej strukturze katalogów, a ich nazwy i ścieżki dostępu zapisywane były, wraz z etykietą odpowiadającą bieżącej godzinie, do pliku l**abels.csv**.
  Przyjęta częstotliwość akwizycji wynosiła 1 Hz (zapic zdjęcia co sekundę.

3.2. Kontrola i weryfikacja

Celem tej części była weryfikacja czy zestaw danych jest spójny, kompletny i w sposób sensowny rozłożony w czasie. Umożliwiała ona wczesne wykrycie brakujących plików, luk w rozkładzie godzin lub nierównomiernego próbkowania w czasie. Opierała się na sprawdzeniu spójności pliku `labels.csv` ze strukturą plików dataset'u oraz rozkładu próbek na godzinowym zakresie czasowym 0-23.

Główne skrypty:

- **src/load_data.py** - Narzędziowy moduł do pracy z plikiem labels.csv i katalogiem z danymi.
- **src/explore_data.py** - Skrypt eksploracyjny do podglądu rozkładu danych wskazanych w labels.csv

3.3. Korekcja

Część korekcyjna miał na celu uporządkowanie i rekonstrukcję etykiet tam, gdzie surowe dane są niekompletne lub niespójne.

Skrypt implementacji funkcjonalności:

- **src/rebuild_labels.py** – generuje nowy `labels.csv` na podstawie struktury katalogów i nazw plików, co zapewnia spójność etykiet z rzeczywistym zbiorem obrazów. Powstał w celu uspójnienia danych wejściowych i umożliwienia wykorzystania danych zebranych w różnych cyklach akwizycji.

3.4. Przygotowanie i normalizacja danych

Część przygotowująca dane miała na celu wyznaczanie cech numerycznych z obrazów oraz ich normalizację. Celem tej części było przekształcenie surowych obrazów do postaci tabelarycznej (cechy numeryczne), odpowiedniej dla modeli klasycznych oraz sieci neuronowych.

Główne skrypty:

- **src/precompute_mean_rgb.py** - dla każdego obrazu liczy średnie wartości kanałów R/G/B.
  Dane zapisywane do trenowania:
  - filepath
  - datetime
  - **hour**
  - r_mean
  - g_mean
  - b_mean

  Plik wynikowy: **features_mean_rgb.csv**

- **src/precompute_features_advanced.py** - wylicza rozszerzony zestaw cech: średnie i odchylenia RGB oraz statystyki HSV (funkcja `extract_rgb_hsv_stats()`)
  Dane zapisywane do trenowania:
  - filepath
  - datetime
  - **hour**
  - r_mean
  - g_mean
  - b_mean
  - r_std
  - g_std
  - b_std
  - h_mean
  - h_std
  - s_mean
  - v_mean

  Plik wynikowy: **features_advanced.csv**

- **src/precompute_features_robust.py** - generuje cechy „robust” (bardziej odporne na warunki oświetleniowe, chmury itd.).
  Dane zapisywane do trenowania:
  - filepath
  - datetime
  - hour
  - r_mean
  - g_mean
  - b_mean
  - r_std
  - g_std

- b_std
- h_mean
- s_mean
- v_mean
- h_std
- s_std
- v_std
- v_p10
- v_p50
- v_p90
- v_iqr_90_10
- s_p10
- s_p50
- s_p90
- s_iqr_90_10
- v_hist_00
- v_hist_01
- v_hist_02
- v_hist_03
- v_hist_04
- v_hist_05
- v_hist_06
- v_hist_07
- v_hist_08
- v_hist_09
- v_hist_10
- v_hist_11
- v_hist_12
- v_hist_13
- v_hist_14
- v_hist_15
- s_hist_00
- s_hist_01
- s_hist_02
- s_hist_03
- s_hist_04
- s_hist_05
- s_hist_06
- s_hist_07
- s_hist_08
- s_hist_09
- s_hist_10
- s_hist_11
- s_hist_12
- s_hist_13
- s_hist_14
- s_hist_15
- v_entropy

    - s_entropy
    - v_top_mean
    - v_bottom_mean
    - v_top_minus_bottom
    - edge_density
      Plik wynikowy: **features_robust.csv**

- **src/normalize_data.py** - normalizuje wybrane pliki cech (`features_*.csv`) z użyciem odpowiedniego skalera (Standard/Robust itp.), i zapisuje zarówno znormalizowane dane (`*_normalized.csv`), jak i parametry skalera (`*.npz`).

We wszystkich przypadkach statystyki normalizacji wyznaczane są **wyłącznie na zbiorze treningowym**, a następnie stosowane do walidacji i testu, co ogranicza ryzyko przecieku informacji.

3.5. Wykorzystane modele ML

Na wstępie realizacji projektu przyjęto, że do implementacji i tworzenia algorytmów wykorzystane zostaną modele językowe.

W taki właśnie sposób wybrano odpowiednie modele na podstawie uzyskanych propozycji dla naszego zestawu danych. Z uwagi na to, iż finalne rozwiązanie miało być stosunkowo proste, skupiono się na rozwiązaniach bazujących głównie na statystyce, tj. regresji liniowej w jej odpowiednich wariantach, chociaż testowane były również inne rozwiązania. Jako końcowe rozwiązanie chciałbym użyć sieci neuronowych, aby móc porównać wyniki tych narzędzi. Ostatecznie więc testowanymi rozwiązaniami były:

- Logistic Regression (klasyfikacja godzin na cechach RGB i advanced)
- k-Nearest Neighbors (k-NN)
- Random Forest (klasyfikacja na cechach advanced, wersja odchudzona pod Raspberry Pi)
- Gradient Boosting (GradientBoostingClassifier)
- Ridge Regression (w wariancie wielowyjściowym na wektorze [sin, cos])
- HistGradientBoostingRegressor (wielowyjściowy regresor)
- RandomForestRegressor (wielowyjściowy regresor [sin, cos])
- MLP – wielowarstwowa sieć neuronowa dla regresji cyklicznej (PyTorch)
- Konwolucyjna sieć neuronowa (CNN) klasyfikująca godzinę bezpośrednio z obrazu.

3.6. Trenowanie

Część realizująca uczenie i porównywanie modeli klasyfikacji i regresji czasu, w tym modeli cyklicznych. Celem tej części jest wytrenowanie zestawu modeli przewidujących godzinę na podstawie cech wyznaczonych z obrazów. W ramach prac dokonano szereg analiz pozwalających na ocenę możliwości wykorzystania konkretnych rozwiązań ML. Opracowane implementacje wyszczególnione zostały poniżej.

1. Modele bazowe oparte na prostych cechach RGB:

- **src/baseline_rgb.py** – klasyfikacja godzin na podstawie `**features_mean_rgb.csv**`.
  - Skrypt do trenowania bazowego modelu klasyfikacji godzin na podstawie średnich wartości RGB obrazów.
  - Trenuje model LogisticRegression.
  - Dziennik wyników trenowania: **Logs/baseline_rgb_log.txt**

- **src/baseline_advanced.py** – modele na ręcznie wybranych cechach z `**features_advanced.csv**`.
  - Skrypt do trenowania bazowego modelu klasyfikacji godzin na podstawie rozszerzonych cech obrazu 'features_advanced.csv': mean/std RGB + statystyki HSV z kanałów H/S/V).
  - Trenuje kilka modeli klasyfikacyjnych (LogisticRegression, KNN, RandomForest, GradientBoosting)
  - Dziennik wyników trenowania: **Logs/baseline_rgb_log.txt**
-
- **src/baseline_advanced_logreg.py** – LogisticRegression na pełnym zestawie cech advanced.
  - Skrypt do trenowania bazowego modelu klasyfikacji godzin na podstawie rozszerzonych cech '**features_advanced.csv**'.
  - Trenuje model LogisticRegression w pipeline ze skalowaniem (StandardScaler)
  - Dziennik wyników trenowania: **Logs/baseline_advanced_logreg.txt**

2. Modele operujące na cechach robust:

- **src/train_robust_time.py**
  - klasyfikacja godzin z binningiem (przedziały czasowe),
  - LogisticRegression / RandomForest / GradientBoosting, metryki top-k i circular MAE.
  - Dziennik wyników trenowania: **baseline_advanced_logreg**

3. Modele regresji cyklicznej:

- **src/train_hour_regression_cyclic.py**
  - regresja (sin, cos) z użyciem modeli sklearn (Ridge, HGB, RF) na `features_robust.csv`,
  - metryki: Cyclic MAE, P90/P95, czas trenowania.
  - Dziennik wyników trenowania: **train_hour_regression_cyclic_log.txt**
- **src/train_hour_nn_cyclic.py**
  - sieć MLP w PyTorch (wejście: cechy robust, wyjście: `[sin, cos]`),
  - **czasowy split danych**: `time_based_split()` dzieli dane chronologicznie na train/val/test po całych dniach (`datetime → date`),
  - normalizacja cech liczona tylko na train, identyczna transformacja stosowana do val/test,

- zapisywany jest najlepszy checkpoint `models/best_mlp_cyclic.pt` według walidacyjnego Cyclic MAE.
- Dziennik wyników trenowania: **train_hour_nn_cyclic_log.txt**

4. Modele CNN na pełnym obrazie:

- **src/train_hour_cnn.py**
  - konwolucyjna sieć neuronowa w PyTorch uczona bezpośrednio na obrazach (bez ręcznego feature engineeringu),
  - wejście: obraz RGB przeskalowany do zadanej rozdzielczości kwadratowej (np. 224×224),
  - wyjście: klasy godzin 0..23 lub ich rozkład prawdopodobieństwa,
  - zapis checkpointu (stan modelu oraz metadane, m.in. `num_classes`, `img_size`) do pliku `models/rpi/best_cnn_hour.pt`.
  - Dziennik wyników trenowania: **train_cnn_log.txt**

Wytrenowane modele klasyczne zapisywane są jako pliki `.pkl` w katalogu `models/`, natomiast model MLP jako plik `.pt` (stan sieci oraz parametry normalizacji).

3.7. Implementacja rozwiązań

Część implementacyjna – wykorzystanie wytrenowanych modeli w trybie on-line (overlay godziny na obrazie, praca na żywym strumieniu z kamery). Celem tej części jest wykorzystanie wytrenowanych modeli do przewidywania godziny na nowych obrazach
oraz prezentowanie tej informacji użytkownikowi w sposób czytelny i ciągły.

Główne skrypty:

- **src/predict_hour.py**
  - wczytuje zapisane modele oraz skalery,
  - przyjmuje ścieżkę do obrazu, wylicza cechy (robust / advanced) i zwraca przewidywaną godzinę (tryb batch/off-line).
- **src/camera_hour_overlay.py**
  - prosty overlay czasu na obrazie (tryb podglądu / debug),
  - może korzystać z prostszych modeli baseline.
- **src/camera_hour_overlay_advanced.py**, **src/camera_hour_overlay_rpi.py**
  - integracja z OpenCV i/lub Raspberry Pi,
  - pobranie klatki z kamery, predykcja godziny **aktualnym modelem produkcyjnym** (np. `best_mlp_cyclic.pt`),
  - naniesienie opisu (overlay) i zapis / wyświetlenie w pętli.

- **src/camera_hour_overlay_mlp.py**, **src/camera_hour_overlay_mlp_rpi.py**
  - warianty overlay oparte o regresję cykliczną MLP na cechach robust oraz tryb fallback (RandomForest na cechach advanced),
  - wersja RPi obsługuje dodatkowo model CNN trenowany na pełnym obrazie (`best_cnn_hour.pt`, przełącznik `--use_cnn`),

- skrypty zapisują klatki z nałożoną godziną do pliku JPG oraz opcjonalny log CSV z predykcjami.

3.8. Problemy

Zmiana pogody.

To jest prawdopodobnie jeden z ważniejszych czynników zakłócających pomiar, akwizycję danych i ostateczne wyniki. Duża zmienność warunków środowiskowych, pociągająca za sobą znaczący wpływ na oświetlenie sceny, jest jednym z głównych problemów takiego zagadnienia.

Pora roku

Ciągłe zmiany wynikające z pór roku i rocznego obiegu Ziemi wokół Słońca.
Wydaje się, że taki zegar miałby sens, ale musiałby być sukcesywnie, cały czas, dotrenowywany w miarę upływu roku kalendarzowego. Jest to do zaimplementowania w prosty sposób i mogłoby okazać się całkiem skuteczne.

Zmiana sceny

Problemem okazuje się również przypadkowa lub chwilowa zmiana na scenie kamery. Powodem może być np. pojawienie się jakiegoś obiektu niebędącego w danych treningowych.

Przypadkowe zakłócenia

Podobnie problematyczne są także inne zakłócenia wynikające np. z trącanie kamery i tym samym zmiana zawartości kadru.

Zakłócenia wewnętrzne

Kwizycji danych i integracja ostatecznego rozwiązania realizowane były z perspektywy mieszkania.

A więc zarówno na dane treningowe, jak i na finalne rozwiązanie wpływało oświetlenie pochodzące z wnętrza mieszkania, co mocno zakłócało wyniki działania algorytmów.

Rozmiar obrazu

Nie dokonano szczegółowej analizy wpływu rozmiaru i jakości obrazu na wyniki.
Problem ten powinien być rozważany w dalszych krokach, aby uzyskać odpowiednie odpowiedzi na takie pytania. Prawdopodobnie szersze widmo detektora pozwoliłoby na bardziej efektywne wyniki trenowania.

4. Podsumowanie

***User**: Jesteś recenzentem naukowym oceniającym eksperyment z zakresu uczenia maszynowego. Masz dostęp do kodu, danych wejściowych, metryk ewaluacyjnych oraz wyników eksperymentów.*
Twoim zadaniem jest:

1. *Odtworzyć, co faktycznie zostało zmierzone i jakiego typu było zadanie (klasyfikacja, regresja, liczba klas, baseline losowy).*
2. *Zidentyfikować wszystkie użyte metryki (np. accuracy, MAE, top-k, train vs test).*
3. *Sprawdzić spójność liczb – czy nie odnoszą się do różnych zadań, zbiorów lub definicji sukcesu.*
4. *Ocenić, które wyniki są:*
    - *istotnie lepsze od losowych,*
    - *umiarkowanie obiecujące,*
    - *potencjalnie mylące lub trywialne.*
5. *Oddzielić twarde wnioski wynikające z danych od interpre. Do dyspozycji masz oczywiście całe repozytorium w którym zawarte są: dokumentacje, kody skryptów Pythonowych oraz pliki logów badań i pomiarów oraz przetwarzania.*

GitHub Copilot: Created 5 todos
Zacznę od zidentyfikowania skryptów treningowych i zadań, potem wejdę w logi i na końcu podsumuję wyniki i wnioski.
Searched for files matching **/src/train_*, 5 matches
Searched for files matching **/Logs/**, 65 matches
Searched for files matching **/docs/*.md, 20 matches
=> [**src/baseline_advanced**.py]
Mam już mapę głównych skryptów; teraz sprawdzę definicje zadań i metryk w train_* oraz wyciągnę konkretne liczby z logów, żeby potem to zsyntetyzować.
=> [**src/baseline_advanced_logreg**.py]
=> [**src/train_robust_time**.py]
=> [**src/train_hour_regression_cyclic**.py]
=> [**src/train_hour_nn_cyclic**.py]
=> [**src/train_hour_cnn**.py]
Starting: *Przejrzeć kluczowe logi eksperymentów i użyte metryki* (2/5] Teraz wyciągnę konkretne liczby z logów (baseline, robust, MLP, regresja cykliczna), żeby później je porównać i ocenić względem losowego baseline.
=> [Logs/**baseline_rgb_log**.txt]
=> [Logs/**baseline_advanced_log**.txt]
=> [Logs/**baseline_advanced_logreg_log**.txt]
=> [Logs**/train_robust_time_log**.txt]
=> [Logs/**train_hour_regression_cyclic_log**.txt]
Mam wyniki dla baseline_RGB, advanced (różne modele) i robust (binning + regresja cykliczna). Teraz dociągnę logi MLP (train_hour_nn_cyclic) i ew. podsumowanie w Summary.md, żeby domknąć obraz i przejdę do oceny względem baseline losowego.
=> [**Logs/train_hour_nn_cyclic**.txt]
=> [**Logs/2025.12.26/train_hour_nn_cyclic_2025.12.26_log**.txt]
=> [**Logs/2025.12.26/Summary**.md]

=> [**Logs/2025.12.29/trenowanie_pod_rpi**.md]
=> [**Logs/2025.12.29/train_cnn_log**.txt]
Starting: *Sprawdzić spójność zadań, zbiorów danych i definicji metryk* (3/5] Mam już komplet kluczowych logów (RGB, advanced, robust, MLP, CNN); teraz sprawdzę spójność definicji zadań i podsumowania w dokumentacji, a potem przejdę do oceny jakości wyników i sformułowania wniosków.
=> [**docs/baseline_overview**.md]
=> [**docs/Podsumowanie%20etapu**.md]
=> [**docs/Podsumowanie.md**#1-1], lines 1 to 260
Poniżej traktuję repo jak pełny „eksperyment naukowy": opisuję, **co faktycznie zmierzono**, jakiego typu było zadanie, jakie metryki, czy liczby są spójne oraz jak je interpretować względem baseline'u losowego. Na końcu rozdzielam „twarde fakty" od interpretacji.

---

1. Co było mierzone i jaki był typ zadania
Dane i etykiety

- Dane: pojedyncze obrazy z kamery, zebrane w strukturze katalogów dataset/YYYY/MM/DD/HH/*.jpg.
- Etykieta: godzina doby hour ∈ {0,…,23} + datetime (dokładny timestamp) – patrz settings.py, load_data.py.
- Rozmiar zbioru:
    - labels.csv: ok. 471 k rekordów (logi: Wczytano cechy: (471031, …) w wielu plikach logów).
    - Rozkład godzin jest praktycznie wyrównany (potwierdzone w dokumentacji i w histogramach z explore_data.py).

Typy zadań

1. **Klasyfikacja 24-klasowa (dokładna godzina)**
    - Wejście:
        - cechy tablicowe (mean RGB, advanced RGB+HSV, robust),
        - lub cały obraz (CNN).
    - Wyjście: klasa godzinowa 0–23.
    - Skrypty: baseline_rgb.py, baseline_advanced.py, baseline_advanced_logreg.py, train_hour_cnn.py.
2. **Klasyfikacja binowa w przedziałach czasowych („binning")**
    - Godzina mapowana do binu co 4 godziny (0–3,4–7,…,20–23 ⇒ 6 klas).
    - Skrypt: train_robust_time.py.
    - Mierzone: top-k accuracy (k=1,2,3), circular MAE (błąd w godzinach liczony od środka binu), „confident accuracy" po odrzuceniu niepewnych próbek.
3. **Regresja cykliczna (ciągła godzina 0–24, sin/cos)**
    - Cel: przewidywać pozycję na okręgu 24h jako $(\sin\theta, \cos\theta)$, $\theta = 2\pi h/24$.
    - Skrypty:
        - klasyczne modele: train_hour_regression_cyclic.py,
        - MLP (PyTorch) na cechach robust: train_hour_nn_cyclic.py.
    - Wyjście: wektor 2D → konwersja na godzinę → błąd cykliczny.

Baseline losowy

- Dla klasyfikacji 24-klasowej:
    - losowy wybór klasy: oczekiwane accuracy $\approx \frac{1}{24} \approx 4.2\%$.
    - „zawsze najczęstsza klasa" – przy równym rozkładzie też ≈ 4–5%.

- Dla binów (6 klas, bin_size=4): losowy wybór $\approx \frac{1}{6} \approx 16.7\%$.
- Dla regresji cyklicznej:
    - jeśli zawsze przewidujemy tę samą godzinę przy równym rozkładzie, oczekiwany **circular MAE** = ok. 6 h (średnia odległość po okręgu).

---

2. Jakie metryki były używane (z podziałem na eksperymenty)

Poniżej tylko metryki „końcowe" z logów.

(A) Baseline RGB – Logistic Regression na mean RGB

- Skrypt: baseline_rgb.py.
- Dane: features_mean_rgb_normalized.csv, 3 cechy (r_mean,g_mean,b_mean).
- Podział: train_test_split, test_size=0.3, stratify=y.
- Log: baseline_rgb_log.txt.
- Wyniki (test, 24 klasy, n≈141 k):
    - Accuracy: **0.2428**.
    - Pełne classification_report + confusion_matrix w logu.
    - Macro-F1 ≈ 0.22 (z raportu).
- Metryki użyte: accuracy, precision/recall/F1 per godzina, macierz pomyłek.

(B) Baseline advanced – kilka modeli na cechach RGB+HSV

- Skrypt: baseline_advanced.py; opis w baseline_overview.md.
- Cecha wejściowe: 8–10 cech (mean/std RGB + mean S,V, opcjonalnie H/H_std).
- Dane: features_advanced.csv, 471 031 próbek.
- Podział: train_test_split, test_size=0.3, stratify=y.
- Logi:
    - Wersja 1: baseline_advanced_log.txt – **RF najlepszy**.
    - Wersja 2 (pipeline 2025-12-29): baseline_advanced_2025.12.29_log.txt – **KNN najlepszy**.
- Wyniki (przykładowo z pierwszego logu):
    - Logistic Regression: accuracy ≈ **0.337**.
    - KNN: accuracy ≈ **0.766**.
    - Random Forest: accuracy ≈ **0.795**.
    - Gradient Boosting: accuracy ≈ **0.702**.
- W drugiej sesji RF ma 0.744, KNN 0.766, więc widełki: **0.74–0.80** dla najlepszego modelu.
- Metryki: accuracy, pełny classification_report, confusion_matrix dla najlepszego modelu.

(C) Baseline advanced – czysty LogisticRegression

- Skrypt: baseline_advanced_logreg.py.
- Dane: features_advanced.csv, wszystkie kolumny numeryczne oprócz hour.
- Podział: jak wyżej (70/30, stratified).
- Log: baseline_advanced_logreg_log.txt.
- Wyniki:
    - Accuracy: **0.3369**.
    - Pełne classification_report i confusion_matrix.
- Metryki: jak w (A), 24-klasowy klasyfikator.

(D) Robust binning – klasyfikacja godzin do 6 binów

- Skrypt: train_robust_time.py.
- Dane: features_robust.csv (56 cech + metadane), 471 031 próbek.

- Podział: train_test_split, test_size=0.2, stratify po binie.
- Log: train_robust_time_log.txt.
- Dla najlepszego modelu (RandomForestClassifier):
    - top-1 accuracy: **0.9266**.
    - top-2: **0.9904**.
    - top-3: **0.9992**.
    - Circular MAE (od środka binu, w godzinach): **1.157 h**.
    - „Confident accuracy" (threshold max-prob=0.45): **0.9323** przy coverage **0.989**.
    - Pełny classification_report i confusion_matrix (6 klas).
- Metryki:
    - accuracy top-1/top-2/top-3,
    - circular MAE w godzinach,
    - confident accuracy + coverage.

(E) Regresja cykliczna – klasyczne modele

- Skrypt: train_hour_regression_cyclic.py.
- Dane: features_robust.csv, 471 031 próbek, 56 cech.
- Podział: train_test_split, 80/20, stratify po hour.
- Log: train_hour_regression_cyclic_log.txt.
- Modele:
    - Ridge (liniowy),
    - HistGradientBoostingRegressor (HGB),
    - RandomForestRegressor (RF), wszystkie w MultiOutputRegressor na $(\sin, \cos)$.
- Wyniki (test, 94 207 próbek):
    - Ridge:
        - Cyclic MAE: **2.022 h**,
        - Naive MAE: **3.487 h**.
    - HGB:
        - Cyclic MAE: **0.656 h**,
        - Naive MAE: **1.469 h**.
    - RF:
        - Cyclic MAE: **0.360 h** (≈ 21.6 min),
        - Naive MAE: **0.877 h**,
        - P90 cyclic error: **1.173 h**,
        - P95 cyclic error: **1.782 h**.
- Metryki: Cyclic MAE (główna), „zwykłe" MAE, P90, P95 błędu cyklicznego.

(F) Regresja cykliczna – MLP (PyTorch)

- Skrypt: train_hour_nn_cyclic.py; opis w train_overview.md.
- Dane: features_robust.csv.
- Podział: 80/10/10 (train/val/test), stratyfikacja po hour.
- Log (wersja z 56 cech): train_hour_nn_cyclic.txt.
    - BEST epoch: val_cMAE = **0.581 h**.
    - Test:
        - Cyclic MAE: **0.589 h** (≈ 35.3 min),
        - P90: **1.561 h**,
        - P95: **2.289 h**.

- Log wersji pipeline (rozszerzone cechy, 58 kolumn): train_hour_nn_cyclic_2025.12.26_log.txt.
    - BEST epoch: val_cMAE = **0.539 h**.
    - Test:
        - Cyclic MAE: **0.543 h** (≈ 32.6 min),
        - P90: **1.461 h**,
        - P95: **2.153 h**.
- Metryki:
    - Cyclic MAE na zbiorach val/test,
    - P90, P95 błędu,
    - ewolucja train_loss/val_cmae po epokach.

(G) CNN na obrazach

- Skrypt: train_hour_cnn.py.
- Dane: pełne obrazy + labels.csv (471 095 rekordów).
- Podział: random_split: val_frac=0.1 (brak osobnego testu).
- Log: train_cnn_log.txt.
- Wyniki (walidacja, 10 epok, 24 klasy):
    - EPOCH 1: train_acc=0.355, val_acc=0.516.
    - …
    - EPOCH 9: train_acc=0.706, **val_acc=0.718**, val_loss=0.7001 (najlepszy model).
    - EPOCH 10: val_acc=0.707 (nie poprawia val_loss).
- Metryki: train_loss, train_acc, val_loss, val_acc per epoka.

---

3. Spójność liczb, zadań i metryk

- **Spójność definicji zadań**
    - Wszystkie eksperymenty klasyfikacyjne (RGB, advanced, CNN) traktują problem jednoznacznie jako **24-klasową klasyfikację godzin**.
    - train_robust_time.py jasno przerzuca to na 6-klasową klasyfikację z binami co 4h – metryki i raport odnoszą się konsekwentnie do tych 6 klas.
    - Regresja cykliczna (sklearn + MLP) ma spójne kodowanie celu (sin/cos) i jednolitą definicję błędu (circular MAE).
- **Spójność zbiorów danych**
    - Tablicowe eksperymenty używają spójnych CSV:
        - features_mean_rgb(_normalized).csv – baseline_rgb,
        - features_advanced.csv – advanced,
        - features_robust.csv – robust/binned/regresja/MLP.
    - Liczba próbek jest bardzo zbliżona we wszystkich logach (ok. 471 k).
    - Podziały:
        - baseline'y: 70/30 (train/test),
        - robust_time / regresja: 80/20,
        - MLP: 80/10/10,
        - CNN: 90/10 (train/val).
    - To znaczy, że **eksperymenty nie używają dokładnie tych samych przykładów testowych**, ale są spójne wewnątrz każdej rodziny (tzn. każda metryka dotyczy „prawdziwego" hold-outu, nie traina).
- **Spójność metryk**

    - Dla klasyfikacji 24-klasowej wszędzie główna jest **accuracy**, wspierana przez classification_report i confusion_matrix.
    - Dla binned-classification: top-k accuracy + circular MAE + confident accuracy – wszystkie liczone spójnie z predict_proba.
    - Dla regresji cyklicznej: te same definicje MAE i błędu cyklicznego w sklearn i MLP.
- **Czego NIE robią eksperymenty**
    - Brak osobnego time-based splitu w większości (z wyjątkiem koncepcji w train_hour_nn_cyclic_2.py opisanej w train_overview.md): podziały są losowe, stratyfikowane po godzinie, co **może zawyżać wyniki** przy silnej redundancji między kolejnymi sekundami/dniami.
    - CNN ma tylko train/val (brak osobnego testu), więc *val_acc* ≈0.71 jest estymatą, ale nie „ostatecznym" wynikiem generalizacji.

Ogólnie: **liczby są spójne względem definicji zadań i metryk, ale trzeba pamiętać, że wszystkie są oparte na losowym splicie po pojedynczych obrazach, nie po dniach**.

---

4. Ocena jakości wyników względem baseline losowego

Zestawiam w trzech kategoriach zgodnie z Twoją prośbą.

4.1. Wyniki istotnie lepsze od losowych (silny sygnał w danych)

- Baseline RGB:
    - Accuracy ≈ **0.24** vs losowe ≈ **0.04**.
    - Już proste mean RGB daje ok. **6×** lepiej niż losowo – bardzo mocny dowód, że „gołym kolorem" da się wnioskować o porze dnia (dzień/noc, wschód/zachód).
- Baseline advanced (KNN / RF):
    - Accuracy w przedziale **0.74–0.80** (24 klasy).
    - To ≈ **18–19×** lepiej niż losowe i zdecydowanie poza marginesem błędu statystycznego przy 141 k przykładów testowych.
- Robust binning (RF, 6 klas):
    - top-1 accuracy **0.93** vs losowe **0.17**,
    - circular MAE ≈ **1.16 h** (przy binach 4h),
    - top-2/top-3 accuracy ≈ **0.99–1.00**.
    - Oznacza to, że w ogromnej większości przypadków model trafia w poprawne okno 4-godzinne, a prawdziwa godzina jest prawie zawsze w 2–3 najlepszych binach.
- Regresja cykliczna – RF:
    - Cyclic MAE ≈ **0.36 h** (≈ 22 min) vs:
        - Ridge ≈ **2.02 h**,
        - naiwny stały predyktor ≈ **6 h**.
    - To oznacza ok. **6–17×** mniejszy błąd niż sensowny baseline, przy bardzo małych P90/P95 (1.17–1.78 h).
- Regresja cykliczna – MLP:
    - Test Cyclic MAE ≈ **0.54–0.59 h** (32–35 min),
    - P90 ≈ **1.46–1.56 h**, P95 ≈ **2.15–2.29 h**.
    - Gorsze niż RF, ale wciąż **rząd wielkości lepsze** niż baseline losowy / liniowy.
- CNN:
    - Val accuracy ≈ **0.71–0.72** dla 24 klas, co jest wyraźnie > losowe 0.04 i > baseline RGB 0.24.

    - To dowód, że informacja o godzinie jest dostępna także dla modelu end-to-end bez ręcznych cech.

4.2. Wyniki umiarkowanie obiecujące (sensowne, ale nie przełomowe)

- Logistic Regression na advanced features:
    - Accuracy ≈ **0.33–0.34** – wyraźna poprawa względem baseline RGB (0.24), ale daleko od RF/MLP/CNN.
    - Wskazuje, że **dodatkowe cechy** pomagają, ale „siła" metody leży raczej w nieliniowych modelach.
- CNN vs tablicowe RF/MLP:
    - CNN z accuracy ≈0.71 jest nieco gorszy od najlepszych tablicowych (0.74–0.80).
    - Jest jednak trenowany tylko 10 epok prostym modelem i na surowych obrazach – wyniki są obiecujące, ale nie dowodzą jeszcze przewagi CNN nad klasycznym ML.

4.3. Wyniki potencjalnie mylące lub trywialne (bez kontekstu)

- Bardzo wysoki top-3 accuracy ≈ 0.9992 w train_robust_time:
    - top-3 przy 6 klasach to „czy prawdziwy bin jest w połowie wszystkich klas".
    - Dla dobrego modelu to oczekiwane, ale jako samodzielna metryka jest odrobinę trywialna – istotniejsze są top-1 i circular MAE.
- Prawie perfekcyjne klasyfikacje w macierzach pomyłek RF / KNN (dla godzin 8–15):
    - Przy losowym splicie po pojedynczych obrazach jest bardzo prawdopodobne, że:
        - zdjęcia z tych samych godzin i dni trafiają zarówno do train, jak i do testu,
        - warunki są niemal identyczne, stąd pozornie „idealna" generalizacja.
    - Jeśli interpretować te wyniki jako **„działa to na przyszłych dniach/pogodach"**, mogłyby być mylące.
    - Jako *dowód, że modele potrafią wykorzystać sygnał z danych przy mocno losowym splicie* – są poprawne.
- Dokumentacyjne stwierdzenia w Podsumowanie.md o „30–40% accuracy" dla advanced:
    - Są po prostu **nieaktualne** względem późniejszych logów (RF/KNN dochodzą do 0.74–0.80).
    - Kierunek jest spójny (advanced >> RGB), ale liczby w tekście nie odzwierciedlają nowszych wyników.

---

5. Twarde wnioski vs interpretacje

5.1. Twarde wnioski (prosto z danych/logów)

- Dane:
    - Zbiór ma ok. **471 k obrazów** z równomiernym rozkładem godzin.
    - Struktura labels.csv i istnienie plików zostało sprawdzone przez load_data.py, logi w Logs/load_data_*.txt wskazują na pojedyncze lub brakujące wiersze, ale ogólną spójność.
- Zadanie:
    - Główne zadanie to **24-klasowa klasyfikacja godziny** oraz odpowiadające mu warianty binning/regresja cykliczna.
- Baseline:

    - LogisticRegression na mean RGB osiąga **ok. 24% accuracy**, co jest zdecydowanie powyżej losowego 4.2%.
- Cechy advanced:
    - Dodanie std i HSV oraz nieliniowych modeli tablicowych podnosi accuracy do **0.74–0.80**.
    - LogisticRegression na tych cechach dochodzi do **0.33–0.34** accuracy.
- Cechy robust:
    - W zadaniu 6-klasowym (biny 4h) RF osiąga:
        - top-1 accuracy ≈ **0.93**,
        - circular MAE ≈ **1.16 h**.
    - W regresji cyklicznej RF osiąga:
        - Cyclic MAE ≈ **0.36 h**,
        - P90 ≈ **1.17 h**, P95 ≈ **1.78 h**.
    - MLP na tych samych cechach osiąga:
        - Test Cyclic MAE ≈ **0.54 h**,
        - P90 ≈ **1.46 h**, P95 ≈ **2.15–2.29 h**.
- CNN:
    - Na obrazach osiąga **≈0.71–0.72 accuracy** na zbiorze walidacyjnym (24 klasy).
- Pipeline:
    - Log Summary.md podsumowuje czas całego biegu pipeline'u na ~42.6 h, przy czym czyste trenowanie modeli (bez precompute) to łącznie kilka tysięcy sekund – to jest zbieżne z czasami raportowanymi przez poszczególne skrypty.

5.2. Interpretacje (sensowne, ale wymagają zastrzeżeń)

- **Silny sygnał światła dziennego**:
    - Różnica między baseline losowym (4%) a baseline RGB (24%) i advanced RF (≈80%) jest tak duża, że można stwierdzić: *obraz nieba/sceny zawiera bardzo mocny sygnał informujący o porze dnia*.
    - To jednak nie mówi jeszcze nic o generalizacji w innych lokalizacjach, porach roku czy przy innej ekspozycji kamery.
- **„Zegar biologiczny" na poziomie ~20–35 minut błędu średniego**:
    - Wyniki RF (0.36 h) i MLP (0.54 h) w regresji cyklicznej sugerują, że **na tym konkretnym zbiorze, przy losowym splicie, model przeciętnie myli się o 20–35 minut**.
    - To jest bardzo dobry wynik, ale:
        - może być częściowo efektem korelacji między próbkami (kolejne sekundy/minuty tego samego dnia),
        - nie mamy tu „prawdziwego" time-based splitu na poziomie dni.
- **CNN jako potencjał na przyszłość**:
    - Przy dość prostym architekturze i 10 epokach CNN osiąga val_acc ≈0.71, co jest już lepsze niż liniowe baseline'y.
    - Nie ma jeszcze dowodu, że CNN pobije tablicowe RF/MLP przy porządnej optymalizacji architektury i augmentacji, ale wyniki są zdecydowanie zachęcające.
- **Możliwa nadmierna optymistyczność wyników tablicowych**:
    - Ponieważ split jest losowy po pojedynczych obrazach, a nie po dniach, istnieje ryzyko, że:

        - do testu trafiają bardzo podobne przykłady jak w train (kolejne sekundy tego samego poranka),
        - przez co accuracy oraz MAE mogą być **zawyżonymi estymatami generalizacji na „przyszłe dni/pory roku”**.
    - To nie unieważnia wyników, ale ogranicza zakres, w jakim można je interpretować jako „zegarek działający w dowolnych warunkach”.

---

6. Co bym zaproponował jako następny krok naukowy

To już trochę wykracza poza Twoje pytania, ale logicznie wynika z powyższej recenzji:

- Dla pełnej rzetelności:
    - dodać wariant eksperymentu z **time-based split** (np. trenowanie na pierwszych 70–80% dni, test na ostatnich 20%),
    - uruchomić go przynajmniej dla:
        - robust RF,
        - MLP (train_hour_nn_cyclic_2),
        - CNN.
- W sprawozdaniu:
    - wyraźnie rozdzielić wyniki „random split” (pokazujące siłę sygnału w danych) od wyników „time-split” (bardziej odpowiadających rzeczywistej generalizacji),
    - raportować obie grupy metryk obok baseline’u losowego.