DATA WYDANIA: 06.2025

ŚCIEŻKA KARIERY PROGRAMISTA AI

COMPUTER VISION

VS CODE, ANACONDA, GIT, DOCKER, JUPYTER | PYTHON Z NUMPY, PYTORCH, ONNX | MODELOWANIE: KLASYFIKACJA, DETEKCJA (YOLOV8), SEGMENTACJA (MASK R-CNN), GŁĘBIA 3-D | PREPROCESSING Z OPENCV, ALBUMENTATIONS, LABELME, CVAT | FASTAPI, DOCKER, GITHUB ACTIONS, DVC KOMPLETNE PORTFOLIO

Spis Treści

Wstęp

Wyobraź sobie, że za dwanaście miesięcy ktoś zapyta: „Kim jesteś z zawodu?" — a Ty pewnie odpowiesz: „Jestem programistą AI, specjalizuję się w detekcji obrazów". Na potwierdzenie pokażesz własną stronę z gif-ami YOLO, dashboard Grafany z latencją 60 ms, PDF z analizą biasu według EU AI Act i zmergowany PR do OpenCV. Nie prezentacja, nie obietnice — żywe dowody, że przez dwanaście miesięcy zamieniłeś ciekawość w produkcyjne systemy Computer Vision.

Plan? Dwanaście modułów — Miesiąc I do Miesiąca XII — sklejonych jak klocki LEGO. Najpierw warsztat (Python 3.11, Git, Docker, VS Code), potem biegłość w NumPy i matematyce gradientu, własny dataset i pierwszy CNN. W połowie roku wystawiasz FastAPI w Dockerze, latem dobudowujesz YOLO, Mask R-CNN i głębię 3-D, jesienią wdrażasz monitoring, retrain, a potem fairness, robustness, prywatność i ocenę ryzyka AI Act. Miesiąc XII wieńczy całość: release v1.0, portfolio online, CV z twardymi KPI.

Rok. 12 miesięcy. 56 tygodni. 365 dni. A możesz wtedy zawodowo być zupełnie innym człowiekiem. Doświadczonym devem. Specjalistą od AI. Twórcą własnych projektów... Tylko od Ciebie zależy, jak będzie wyglądać ta przyszłość!

Jak pracować z ebookiem?

Zakładamy, że w każdym miesiącu jesteś w stanie poświęcić 8-10h/tygodniowo (choć większość zadań powinna zająć mniej czasu). Jeśli zdarzy Ci się dodatkowy czas wolny, który chcesz poświęcić na naukę lub szybciej skończysz zadania, możesz wykonać zadania dodatkowe – w każdym miesiącu są zadania na 1 wolny weekend i 2 wolne weekendy (wymagają odpowiednio ok. 8h i 16h).

Swoje notatki (tzw. notki techniczne) przechowuj w repo, które przygotujemy w pierwszy miesiącu. Pisanie swoich spostrzeżeń to dobry sposób na zapamiętywanie informacji (podobno powoduje przenoszenie informacji z pamięci krótkotrwałej do pamięci długotrwałej). A wrzucanie commitów na GitHub pomoże Ci poćwiczyć sprawne korzystanie z Gita.

Na koniec każdego miesiąca powinieneś/powinnaś mieć:

- journal/miesiąc-X.md - plik z notatkami wysłany na repo
- notebooks/ - do tego folderu wrzucamy wszystkie pliki .ipynb (jeśli projekt jest bardziej złożony, załóż dla niego oddzielne repo, a notatki i spostrzeżenia wyciągnięte w trakcie pracy wrzuć do folderu journal.
- README.md - to główny plik z dokumentacją projektu. Aktualizuj go co miesiąc (Warto, bo to będzie podstawa Twojego portfolio).
- Opcjonalnie - zeszyt (papierowy) z notatkami i wydrukami kodu.

Miesiąc I. — Setup & Flow

Miesiąc I to czyszczenie warsztatu. Zanim dotkniesz CNN-ów, potrzebujesz środowiska, które można odtworzyć jedną komendą. Dlatego zaczynamy od lokalnego Pythona 3.11, managera env, Gita i Dockera – z tym zestawem każdy kolejny projekt w ciągu roku odpalisz bez „u mnie działa".

VS Code wygrywa na start: lekki, w 100 % darmowy, identyczny na Win/Mac/Linux, ma Jupyter-notebooki i wbudowany Git. PyCharm Pro jest płatny i cięższy; sam Colab nie nauczy Cię pracy z repozytorium ani debugowania poza przeglądarką.

Pytest + CI pojawiają się od razu, bo to minimalna dawka kultury DevOps. Nawet test dla skryptu hello NumPy i workflow GitHub Actions budują nawyk, który później przeniesiesz na milion-parametrowe modele.

1. Instalacja i konfiguracja środowiska – 4 h

- **Co robisz:** instalujesz Python 3.11, menedżera środowisk (Conda lub Poetry) i VS Code z wtyczkami Python oraz Jupyter. Jeśli masz kartę NVIDIA, doinstaluj sterowniki + CUDA Toolkit.
- **Materiały:**
  - „Python 3.11 Installation Guide" (oficjalna dokumentacja)
  - „Setting Up VS Code for Python" (film freeCodeCamp, 15 min)
- **Cel:** w terminalu polecenie python – version zwraca 3.11.x, a w VS Code możesz uruchomić prosty skrypt w nowym environment.

2. Git od zera – 6 h

- **Co robisz:** uczysz się inicjować repozytorium (git init), tworzyć gałęzie, commitować i wysyłać na GitHub; konfigurujesz plik .gitignore i piszesz mały README w Markdown.
- **Materiały:**
  - interaktywny kurs learngitbranching.js.org (2-3 h)
  - książka „Pro Git" rozdziały 2-3
- **Cel:** masz publiczne repo *sciezka-kariery-computer-vision* z pierwszym commitem i poprawnym README.

3. Mini-projekt „hello NumPy” – 6 h
    - **Co robisz:** piszesz skrypt *hello_numpy.py*, który tworzy losową macierz 3×3, liczy determinant i wypisuje wynik; dodajesz test w pytest.
    - **Materiały:**
        - „NumPy Quickstart” (oficjalny tutorial, 30 min)
        - „Getting Started with pytest” (blog Real Python)
    - **Cel:** pytest przechodzi lokalnie; wynik skryptu pojawia się w terminalu bez błędów.

4. Pull Request workflow – 3 h
    - **Co robisz:** tworzysz nową gałąź feature/readme-update, zmieniasz README, otwierasz Pull Request na GitHubie i sam go merdżujesz.
    - **Materiały:** „GitHub Flow in 15 Minutes” (YouTube, GitHub Training)
    - **Cel:** na stronie repo widać pierwszy zakończony PR.

5. Ćwiczenia z CLI i Basha – 4 h
    - **Co robisz:** uczul się poruszania po terminalu: cd, ls, grep, cat, przekierowania |, >.
    - **Materiały:**
        - książka „The Linux Command Line” – rozdziały 1-5
    - **Cel:** potrafisz w jednym poleceniu wyszukać słowo numpy we wszystkich plikach .py repo.

6. Podstawy Markdown – 2 h
    - **Co robisz:** rozszerzasz README o nagłówki, listy, blok kodu i link do dokumentacji NumPy.
    - **Materiał:** „Mastering Markdown” (GitHub Guides, 15 min)
    - **Cel:** README renderuje się poprawnie na GitHubie i zawiera sekcję „How to Run”.

7. Podsumowanie & retrospektywa – 2 h
    - **Co robisz:** spisujesz w notatniku, co zadziałało, a co wymaga dopracowania (czas, narzędzia).
    - **Cel:** masz 10-linijkowy log wniosków na kolejny miesiąc.

Jeśli masz ekstra wolny weekend (8h)

Docker "Hello World" – 4 h

- **Co robisz:** Instalujesz Dockera, piszesz Dockerfile, który uruchamia python *hello_numpy.py*, budujesz obraz i pushujesz go na Docker Hub (konto free).
- **Materiał:** oficjalny tutorial „Docker – Get Started".
- **Cel:** polecenie docker run <twój-obraz> wypisuje determinant macierzy.

GitHub Actions CI – 4 h

- **Co robisz:** Tworzysz workflow .github/workflows/ci.yml, który po każdym push instaluje zależności i odpala pytest.
- **Materiał:** wideo „GitHub Actions for Python Projects" (freeCodeCamp, 45 min).
- **Cel:** zielony znacznik ✔️ „build passing" przy ostatnim commicie.

Jeśli masz 2 ekstra wolne weekendy (16h)

Automatyczne formatowanie kodu – 6 h

- **Co robisz:** Dodajesz pre-commit, konfigurujesz black i ruff; każdy commit jest auto-formatowany.
- **Materiał:** artykuł „Clean Python with Black & Ruff" (Real Python).
- **Cel:** pliki przechodzą formatowanie i linting bez ostrzeżeń.

Głębszy kurs Bash – 6 h

- **Co robisz:** Przerabiasz rozdziały 6-10 „The Linux Command Line"; piszesz skrypt install.sh, który tworzy środowisko i instaluje zależności.
- **Cel:** nowy współpracownik potrzebuje tylko ./install.sh, aby uruchomić projekt.

Notka techniczna – 4 h

- **Co robisz:** Tworzysz plik *journal/miesiac-1.md* w repo *sciezka-kariery-computer-vision*. Opisujesz krok po kroku, jak skonfigurowałaś VS Code, Dockera, środowisko wirtualne i odpaliłaś pierwszy pytest. Dodajesz README.md, .gitignore (jeśli jeszcze ich nie masz) i robisz pierwszy commit.

- **Cel:** publiczne repo z folderem *journal/, wpisem miesiac-1.md* i commitem.

Kamień milowy I. miesiąca

Publiczne repo *cv-playground* z działającym środowiskiem, testem i pierwszym Pull Requestem, a (jeśli masz czas) — obrazem Docker i automatycznym CI.

Miesiąc II. — Python Core + NumPy Foundations

Miesiąc II stawia na "język matki" wszystkich bibliotek – czysty Python i NumPy. Zanim przejdziesz do sieci neuronowych, musisz biegle składać list-comprehensions, klasy i operacje macierzowe. Dlatego w planie ląduje 50 zadań Exercism/LeetCode, szybki kurs Python Crash Course i „100 NumPy Exercises": to najszybszy sposób, by palce same wiedziały, kiedy użyć reshape, a kiedy broadcastingu.

Dlaczego NumPy, nie od razu PyTorch? Bo PyTorch jest nadbudówką nad tablicami NumPy; zrozumienie widoków, kopiowania i osi sprawi, że późniejsze gradienty i tensor tricks będą oczywiste. Lepiej spocić się teraz na macierzy 3 × 3 niż w maju na GPU.

Mały pakiet matrix_utils + testy uczą modularności. Wydzielanie funkcji, paczkowanie przez Poetry i automatyczne formatowanie black/ruff to pierwszy kontakt z jakością kodu produkcyjnego – fundament, na którym bez bólu postawisz kolejne biblioteki DL.

1. Szybki kurs Pythona — 9 h

- **Co robisz:** przerabiasz rozdziały 1-12 książki "*Python Crash Course (2 ed.)*" E. Matthesa.
- **Materiały:** PDF/print książki + repo autora na GitHub.
- **Cel:** w Pythonie napiszesz pętlę, funkcję, klasę i list-comprehension bez zerkania do ściągi.

2. 50 krótkich zadań kodowych — 6 h

- **Co robisz:** rozwiązujesz min. 50 zadań na Exercism – Python track albo „Easy" na LeetCode.
- **Cel:** zielony znacznik „completed 50" na platformie i commit z rozwiązaniami w repo cv-playground.

3. NumPy od podstaw — 6 h

- **Co robisz:** przechodzisz oficjalny tutorial NumPy Quickstart + notatnik „100 NumPy Exercises".

- **Materiały:** *numpy.org/doc/stable/user/quickstart.html, repo 100-NumPy-Exercises*.
- **Cel:** potrafisz użyć broadcastingu, np.dot, axis, reshape i wyjaśnić różnicę view vs copy.

4. Mini-biblioteka matrix_utils — 6 h

- **Co robisz:** tworzysz katalog matrix_utils/ z modułami determinant, inverse, normalize; dodajesz *__init__.py*.
- **Materiały:** artykuł *Real Python „Python Modules and Packages"* (30 min).
- **Cel:** python -m pytest przechodzi dla wszystkich funkcji, a w REPL-u działa from matrix_utils import inverse.

5. Wprowadzenie do pandas — 4 h

- **Co robisz:** serię ćwiczeń *„10 Minutes to pandas"*, ładujesz CSV z Exercism wynikami i robisz prosty wykres .plot().
- **Materiały:** oficjalny notebook pandas.
- **Cel:** ekran Jupyter pokazuje tabelę z Twoimi wynikami i histogram czasów rozwiązania.

6. Retrospektywa i log nauki — 2 h

- **Co robisz:** piszesz w notion/md: co zaskoczyło, ile zadań było trudnych, co poprawić w marcu.
- **Cel:** masz odhaczony checklist za luty i listę „pain points" do przećwiczenia.

Jeśli masz ekstra wolny weekend (8h)

Poetry + automatyczna publikacja pakietu — 4 h

- **Co robisz:** Instalujesz Poetry, zamieniasz *matrix_utils* w paczuszkę z *pyproject.toml*, budujesz wheel.
- **Materiały:** oficjalny tutorial poetry-python.dev.
- **Cel:** lokalnie działa *pip install dist/matrix_utils-*.whl*.

Pierwszy skrypt *CLI z argparse* — 4 h

- **Co robisz:** Piszesz *cli.py*, który przyjmuje ścieżkę do CSV i zwraca średnią z kolumny.

- **Materiały:** dokumentacja argparse, krótkie wideo Corey Schafer "Python CLI Arguments".
- **Cel:** python cli.py scores.csv --col time drukuje liczbę.

Jeśli masz 2 ekstra wolne weekendy (16h)

Git pre-commit, black, ruff — 6 h

- **Co robisz:** Konfigurujesz plik *.pre-commit-config.yaml*, dodajesz hooki black i ruff.
- **Materiały:** *Real Python „Managing Code Quality with pre-commit"*.
- **Cel:** po git commit kod jest auto-sformatowany, a ruff nie zgłasza błędów (exit 0).

Mały projekt *„Image Stats" z OpenCV* — 6 h

- **Co robisz:** Skrypt ładuje folder JPG-ów i wypisuje średnią jasność + zapisuje histogram PNG.
- **Materiały:** PyImageSearch blog „Loading and Displaying Images with OpenCV".
- **Cel:** w repo folder samples/ + wygenerowane hist.png; README tłumaczy użycie.

Notka techniczna / Artykuł „Dlaczego broadcasting rządzi" — 4 h

- **Co robisz:** Tworzysz artykuł na temat: „Dlaczego broadcasting rządzi". Wrzucasz post jako plik *journal/miesiac-2.md* na repo: *sciezka-kariery-computer-vision*.
- **Cel:** wpis journal/miesiac-2.md.

Kamień milowy II. miesiąca

Python & NumPy Ninja: repo cv-playground zawiera bibliotekę matrix_utils z testami, zaliczone 50 zadań Exercism/LeetCode i notatnik Jupyter z ćwiczeniami NumPy. Jeśli zrobiłeś rozszerzenia — projekt „Image Stats" oraz paczka Poetry pojawiają się w README.

Miesiąc III. — Matematyczne Fundamenty pod Deep Learning

Miesiąc III to zastrzyk matematyki "pod maską" uczenia głębokiego. Bez algebry liniowej nie zrozumiesz, dlaczego SVD lub attention działają; bez rachunku różniczkowego – jak w ogóle liczyć gradient. Stąd pakiet: odcinki 3Blue1Brown, rozdziały Goodfellowa o optymalizacji i własnoręczny notebook z gradient-descent na regresji liniowej. To minimum, by później modyfikować sieć świadomie, a nie metodą prób i błędów.

Rachunek prawdopodobieństwa i statystyka pojawiają się tu, bo metryki i straty to po prostu operacje na rozkładach. Zamiast czekać, aż "wyskoczy" cross-entropy, już teraz generujesz próbki z N(0, 1) i rysujesz histogram – dzięki temu, gdy w maju sieć zacznie się chwiać, będziesz wiedzieć, czy to błąd losowy, czy systematyczny.

Mini-projekt "Logistic zera" domyka miesiąc praktyką. Pisząc klasyfikator 100 % NumPy, łączysz teorię z kodem i pokazujesz sobie, że matematyczne wzory naprawdę działają – to najlepsza szczepionka przeciwko traktowaniu frameworków jak czarnej skrzynki.

1. Algebra liniowa – 10 h

- **Co robisz:** Oglądasz odcinki 7-12 serii 3Blue1Brown – Essence of Linear Algebra (wektory własne, SVD, zmiana baz).
- **Ćwiczenia:** Robisz ćwiczenia z zeszytu „Linear Algebra for Machine Learning" (Deisenroth & Faust) – sekcje 3.1-3.3.
- **Cel:** potrafisz policzyć rozkład SVD w NumPy i wyjaśnić, co oznacza wektor własny obrazu 64×64.

2. Rachunek różniczkowy i gradient descent – 8 h

- **Co robisz:** Przerabiasz rozdział 4 książki Deep Learning (Goodfellow, Bengio, Courville) + film StatQuest „Gradient Descent".
- **Ćwiczenia:** Piszesz notatnik gradient_linreg.ipynb, w którym od zera implementujesz regresję liniową i wizualizujesz spadek MSE.
- **Cel:** wykres z malejącą krzywą straty oraz funkcja grad_step() działająca bez autograd.

3. Podstawy prawdopodobieństwa i statystyki – 7 h
    - **Co robisz:** Kurs Khan Academy – Probability & Statistics (działy: zmienne losowe, rozkład normalny, CLT).
    - **Ćwiczenia:** Notebook probability_playground.ipynb generujący 10 000 próbek z N(0, 1) i porównujący histogram z PDF.
    - **Cel:** umiesz policzyć średnią, wariancję i narysować histogram z krzywą gęstości.

4. Mini-projekt „Logistic zera” – 6 h
    - **Co robisz:** Budujesz klasę *LogisticRegressionScratch* (NumPy only) do klasyfikacji dwóch rodzajów kwiatów Iris. Walidujesz za pomocą accuracy i rysujesz wykres granicy decyzji.
    - **Cel:** ≥ 90 % accuracy na Iris-binary oraz wykres decision boundary w matplotlib.

5. Retrospektywa i plan na kolejny miesiąc – 4 h
    - **Co robisz:** Spisujesz, co było trudne (np. interpretacja wartości własnych) i rezerwujesz czas na powtórki.
    - **Cel:** plik march_notes.md z checklistą i samodzielną oceną opanowania tematów (0-5).

Jeśli masz ekstra wolny weekend (8h)

Implementacja XOR z jedną warstwą NN – 5 h

- **Co robisz:** Kodujesz sieć 2-2-1 z aktywacją sigmoid, back-prop liczone ręcznie w NumPy.
- **Cel:** strata < 0.02 po 5 000 epok; wykres trajektorii punktów w przestrzeni cech.

Wizualizacja spadku gradientu – 3 h

- **Co robisz:** Tworzysz notebook `gradient_descent.ipynb`, w którym:
    - implementujesz własny gradient descent dla funkcji

      $y = x^2$ i $y = \log(x + 2)^2$,

        - wizualizujesz spadek kulki po krzywej (matplotlib, animacja lub wykres kroków),
        - zapisujesz finalny wykres jako *gradient_plot.png*.
    - **Cel:** Plik `.ipynb` i `.png` w repo (folder `notebooks/`) + commit: `Miesiąc III: gradient działa.`

Jeśli masz 2 ekstra wolne weekendy (16h)

Micrograd (Karpathy) – 8 h

- **Co robisz:** Przepisujesz lub rozszerzasz projekt micrograd – autograd w < 100 linijkach.
- **Cel:** notebook demonstrujący trening małej MLP na problemie spirali.

StatQuest "Machine Learning Fundamentals" – 4 h

- **Co robisz:** Maraton wideo (playlist 1h40) + notatki w OneNote.
- **Cel:** karta notatek A4 z najważniejszymi wzorami: sigmoid, softmax, cross-entropy.

Flashcards Anki – 4 h

- **Co robisz:** Tworzysz talię 50 kart (pytanie-odpowiedź) z algebry, statystyki i rachunku różniczkowego.
- **Cel:** codzienny trening spaced-repetition, wskaźnik „young cards" = 0 przed końcem miesiąca.

Kamień milowy III. miesiąca

"Math-Ready for DL" – w repo cv-playground znajdują się:
・notatnik gradient_linreg.ipynb z własną implementacją gradient descent,
・klasa LogisticRegressionScratch działająca na Iris,
・notatki march_notes.md podsumowujące algebrę, statystykę i rachunek różniczkowy.

Miesiąc IV. — pierwsze kroki w Computer Vision i przygotowanie własnego datasetu

Miesiąc IV przenosi Cię z macierzy do realnych pikseli. OpenCV służy tu jako „szwajcarski nóż" – pokaże, jak obraz to tablica liczb i jak operatory Canny, Laplacian czy morfologia czyszczą dane, zanim nakarmisz je siecią. Nauka klasycznego CV na początku chroni przed sytuacją, w której model uczy się brudu zamiast cech.

Własny mini-dataset (≥ 500 zdjęć) to kluczowy krok, bo prawdziwe projekty rzadko opierają się na COCO. Samodzielne nagrywanie, etykietowanie i augmentacja uczą, ile kosztuje dobra próba i gdzie czają się błędy anotacji, które później rozwalają precision. Dzięki skryptowi Albumentations od razu widzisz, jak rotacja, clahe czy blur wpływają na dystrybucję danych.

Prosty detektor konturów zamyka miesiąc dowodem, że klasyczne filtry potrafią rozwiązać realny problem bez GPU. Od tej pory każdy głęboki model będziesz porównywać z „filtr-baseline" i świadomie decydować, czy warto płacić rachunek za trening.

1. Instalacja OpenCV i pierwszy skrypt – 3 h

- **Materiały:** oficjalny „Getting Started with OpenCV-Python" + film sentdex „OpenCV in 5 minutes".
- **Cel:** python show_image.py lena.png wyświetla obraz w oknie i zapisuje wersję w odcieniach szarości.

2. Operacje podstawowe – 6 h

- **Co robisz:** w notatniku opencv_basics.ipynb ćwiczysz resize, crop, zamianę przestrzeni kolorów, rysowanie prostokątów/tekstów.
- **Materiały:** rozdziały 1-3 książki „Practical Python and OpenCV" (Silva).
- **Cel:** na końcu notatnika masz kolaż czterech przekształconych wariantów jednego zdjęcia.

3. Przetwarzanie niskopoziomowe – 7 h

- **Co robisz:** implementujesz Canny, Laplacian, progowanie adaptacyjne i morfologię (erozja, dylacja).

- **Materiały:** blog PyImageSearch „Canny Edge Detection", „Morphological Operations".
- **Cel:** folder edges/ z 10 obrazami pokazującymi skutki kolejnych operacji.

4. Nagranie i opis własnego mini-datasetu – 8 h

- **Co robisz:** telefon lub kamera: zbierasz min. 500 zdjęć jednego tematu (np. śrub, ziaren kawy, pudełek). Porządkujesz je w strukturze data/train / data/test.
- **Materiały:** poradnik Roboflow „How to Collect a Dataset
- **Cel:** gotowy katalog z CSV labels.csv (kolumny: filename, class).

5. Adnotacja i pierwszy pipeline augmentacji – 6 h

- **Co robisz:** etykietujesz 100 zdjęć w LabelImg lub CVAT; budujesz skrypt z Albumentations (flip, blur, CLAHE).
- **Materiały:** „Albumentations — Getting Started" (blog) + wideo CVAT „Basic Annotation Workflow".
- **Cel:** skrypt augment.py input/ output/ generuje 2× więcej próbek; plik README opisuje etapy.

6. Mini-projekt „Detektor konturu" – 3 h

- **Co robisz:** klasyczny algorytm: Canny → kontury → filtr rozmiaru → rysowanie bounding-boxów.
- **Cel:** demo GIF „before / after" wrzucone do repo.

7. Retrospektywa i plan na kolejny miesiąc – 2 h

- **Co robisz:** podsumowujesz, które filtry okazały się przydatne i co chcesz ulepszyć przy trenowaniu CNN-ów w maju.
- **Cel:** plik april_notes.md z listą „gotowe" i „do poprawki".

Jeśli masz ekstra wolny weekend (8h)

Kolor i histogramy – 4 h

- **Co robisz:** ćwiczysz rozkłady kanałów HSV, rysujesz histogramy, stosujesz CLAHE;

- **Materiały:** kurs Coursera „Image Processing" (moduł 2, Color).
- **Cel:** notebook color_analysis.ipynb z porównaniem histogramów zdjęcia dziennego i nocnego.

Haar-cascade Face Detector – 4 h

- **Co robisz:** implementujesz szybki wykrywacz twarzy w OpenCV, zapisujesz obszar twarzy do pliku;
- **Materiały:** oficjalny tutorial „Face Detection with Haar Cascades".
- **Cel:** python detect_face.py img.jpg zwraca licz-bę twarzy i zapisuje podgląd z ramką.

Jeśli masz 2 ekstra wolne weekendy (16h)

Powiew C++ i GPU – 6 h

- **Co robisz:** instalujesz OpenCV z flagą WITH_CUDA=ON; przepisujesz algorytm Canny w C++;
- **Materiały:** kurs JetsonHacks „OpenCV CUDA basics".
- **Cel:** porównanie czasu obliczeń CPU vs GPU (≥ 2× przyspieszenie) w pliku benchmark.md.

CLI „Batch Augmentor" – 6 h

- **Co robisz:** piszesz narzędzie augmentor-cli (argparse) z opcjami: liczbę kopii, rodzaj transformacji;
- **Cel:** augmentor-cli --n 3 --blur 2 input/ output/ tworzy 3 augmentacje każdych plików.

Wpis „5 trików OpenCV, które od razu podnoszą jakość danych" – 4 h

- **Co robisz:** Tworzysz plik journal/miesiac-4.md. Opisujesz 5 sposobów, w jakie OpenCV podnosi jakość danych. Wrzucasz na repo: sciezka-kariery-computer-vision.
- **Cel:** Wpis na repo journal/miesiac-4.md.

Kamień milowy IV. miesiąca

"First Dataset & Classic CV" – repo zawiera folder data/ z ≥ 500 zdjęciami, etykiety dla 100 z nich, skrypt augment.py, notatnik opencv_basics.ipynb oraz

demo „Detektor konturu”. Jeśli zrobiłeś rozszerzenia: benchmark GPU/C++ i artykuł z gifami są podlinkowane w README.

Miesiąc V. — Twój pierwszy głęboki model: klasyfikacja obrazów i transfer learning

Miesiąc V to pierwszy skok na głęboką wodę — uczysz się, jak gotowy model zmienić w działający produkt. Fastai + PyTorch pozwalają w godzinę zbudować klasyfikator na CIFAR-10 i od razu zobaczyć, jak lr-finder, augmentacje i schedulery kształtują krzywą straty. Dzięki temu skupiasz się na koncepcjach, a nie na ręcznym implementowaniu back-propa.

Transfer learning na własnym zbiorze z kwietnia pokazuje magię „trzech epok i działa". Przez podmianę jednej warstwy ResNet-34 dostajesz accuracy ≥ 85 % bez tygodniowego trenowania, a przy okazji uczysz się, jak fine-tune wpływa na rozmiar i czas inferencji. Grad-CAM i confusion matrix dodajemy od razu, żebyś widział dlaczego model się myli, a nie tylko gdzie.

Eksport ONNX i prosty skrypt infer.py to pierwszy kawałek produkcji. Gdy model działa offline w < 0,2 s i ma wagi w uniwersalnym formacie, możesz go jutro wrzucić do Dockera albo na telefon — i dokładnie o to chodzi w tym miesiącu.

1. Środowisko „DL-ready" – 4 h

- **Co robisz:** Instalujesz PyTorch 2 + torchvision (lub fastai, jeśli wolisz), weryfikujesz, że torch.cuda.is_available() zwraca True.
- **Materiały:** „Start Locally — PyTorch" (strona oficjalna), film „Install PyTorch w 15 min" – freeCodeCamp.
- **Cel:** skrypt check_cuda.py wypisuje nazwę GPU i wersję CUDA.

2. Kurs fastai – lekcje 1-3 – 9 h

- **Co robisz:** przerabiasz „Practical Deep Learning for Coders" (2023) – klasyfikacja kot-pies i CIFAR-10.
- **Cel:** notebook cifar10_fastai.ipynb osiąga ≥ 75 % accuracy na zbiorze testowym.

3. Transfer learning na Twoim zbiorze z kwietnia – 8 h

- **Co robisz:** Używasz gotowej sieci (ResNet-34 / EfficientNet-B0), zamieniasz ostatnią warstwę na liczbę Twoich klas. Robisz split train/valid (80/20), stosujesz augmentacje z Albumentations

- **Materiały:** fastai Lesson 2, rozdz. 11 „Transfer Learning" w „Dive into Deep Learning".
- **Cel:** train_custom.ipynb osiąga ≥ 85 % accuracy (lub +20 pp wzgl. baseline losowego).

4. Ewaluacja i wyjaśnialność – 5 h

- **Co robisz:** Generujesz confusion matrix, precision/recall/F1; tworzysz Grad-CAM lub heat-mapy dla 5 błędnie sklasyfikowanych zdjęć.
- **Materiały:** „Grad-CAM with PyTorch" (A. Rosebrock), scikit-learn metrics
- **Cel:** folder explain/ z PNG-ami heat-map i plik metrics.json.

5. Eksport i prosty inference CLI – 5 h

- **Co robisz:** Zapisujesz wagi .pth, eksportujesz model do ONNX (torch.onnx.export). Piszesz infer.py image.jpg → etykieta + prawdopodobieństwo
- **Materiały:** oficjalny tutorial „PyTorch to ONNX", film Corey Schafer „argparse
- **Cel:** python infer.py sample.jpg zwraca poprawną klasę w < 0,2 s.

6. Retrospektywa i plan na kolejny miesiąc – 4 h

- **Co robisz:** Spisujesz, jakie augmentacje pomogły, jakie metryki są najniższe, co chcesz poprawić (np. danych jest za mało).
- **Cel:** plik may_notes.md z checklistą „działa / do poprawki".

Jeśli masz ekstra wolny weekend (8h)

Hyper-tuning & przyspieszenie

W&B Sweep lub MLflow – 4 h

- **Co robisz:** konfigurujesz dwa-trzy hiperparametry (learning-rate, batch-size), uruchamiasz automatyczny przegląd, generujesz wykres najlepszej krzywej straty.

Quantization-aware training – 4 h

- **Co robisz:** wykorzystujesz torch.quantization do INT8; mierzysz zmianę rozmiaru modelu i latency (time python infer.py).
- **Cel:** markdown speed_vs_size.md pokazuje tabelę: FP32 vs INT8.

Jeśli masz 2 ekstra wolne weekendy (16h)

Wejście w detekcję i deploy

YOLOv8 fine-tune – 8 h

- **Co robisz:** instalujesz ultralytics, anotujesz 50 klatek pod bounding-boxy, trenujesz yolo detect train. cel: runs/detect/train/weights/best.pt + mAP ≥ 0,25.

FastAPI + Docker deploy – 4 h

- **Co robisz:** tworzysz endpoint POST /predict zwracający JSON {class, prob}. Budujesz Dockerfile, obraz cv-infer:latest.

Notka techniczna – 4 h

- **Co robisz:** Tworzysz plik *journal/miesiac-4.md* w repo *sciezka-kariery-computer-vision*. Opisujesz w nim używane w tym miesiącu narzędzia, dodajesz GIF z testu API
- **Cel:** publiczne repo z folderem *journal/, wpisem miesiac-4.md* i commitem.

Kamień milowy V. miesiąca

“First Deep Model in Production Clothes” – w repo znajdują się:
• notebook cifar10_fastai.ipynb,
• train_custom.ipynb + wagi .pth i .onnx,
• skrypt infer.py działający offline,
• folder explain/ z Grad-CAM,
a dla ambitnych – plik speed_vs_size.md, detektor YOLO oraz obraz Docker z API, wszystko opisane w README.

Miesiąc VI. — Pierwsze kroki w MLOps: kontenery, wersjonowanie danych i automatyczny deploy

Miesiąc VI uczy, że model bez infrastruktury to tylko plik z wagami. Zaczynamy od Dockerfile'a produkcyjnego i FastAPI async, żebyś potrafił wystawić /predict na każdej maszynie w identycznym środowisku. GitHub Actions buduje obraz i odpala pytesty przy każdym pushu — to Twoje pierwsze, w pełni automatyczne CI/CD.

DVC wprowadza wersjonowanie danych i modeli z prywatnym zdalnym storage. Dzięki temu klonujesz repo na czystym laptopie, robisz dvc pull i za minutę masz dokładnie tę samą paczkę danych i wagi co na serwerze. Wieczorny retrain albo rollback do poprzedniej wersji stają się jedną komendą.

Monitoring zamyka miesiąc pokazem „co się dzieje po pierwszym deployu". Middleware Prometheusa mierzy latency, a test integracyjny w Actions odpala kontener, wysyła obraz i sprawdza kod 200. Od teraz wiesz, ile trwa inference, kiedy przycina się CPU i że każda zmiana kodu przechodzi ten sam tunel jakości, zanim trafi do użytkownika.

1. Docker w trybie „produkcyjnym" — 6 h

- **Instalacja i praktyka:** poznajesz wieloetapowe buildy, komendy COPY --from, ENTRYPOINT, zmienne środowiskowe.
- **Materiały:** oficjalny ebook Docker – Best Practices (rozdz. 1-4) + film „Dockerfile 101" (TechWorld-with-Nana).
- **Cel:** obraz cv-infer:prod uruchamia model z maja w < 1 GB i < 6 s cold-startu (docker run …).

2. FastAPI ➜ wersja async — 5 h

- **Co robisz:** Tworzysz plik main.py z routerem POST /predict (async def), walidacją Pydantic i middleware CORS.
- Materiały: FastAPI – The Official Tutorial sekcje Path Ops i Dependencies.
- **Cel:** uvicorn main:app --reload zwraca JSON {class, prob} dla przykładowego obrazu.

3. GitHub Actions → test + build + push — 6 h
    - **Co robisz:** Piszesz workflow, który:
        - (a) uruchamia pytest,
        - (b) buduje obraz Dockera,
        - (c) pushuje go do Docker Hub lub GitHub Container Registry.
    - **Materiały:** kurs wideo „CI/CD with GitHub Actions & Docker" – freeCodeCamp (1 h 20 min).
    - **Cel:** zielony "✔ workflow succeeded" przy każdym push main.
4. DVC – wersjonowanie danych i modeli — 6 h
    - **Co robisz:** Inicjalizujesz dvc init, dodajesz swój dataset oraz plik wag .pth; konfigurujesz zdalny storage (np. AWS S3 free tier).
    - **Materiały:** oficjalny DVC Get Started (4 lekcje) + blog Iterative.ai "Storing Models".
    - **Cel:** dvc push wysyła dane, a dvc pull na czystej maszynie odtwarza data/ i models/.
5. Testy integracyjne API — 5 h
    - **Co robisz:** Przy pomocy pytest + httpx piszesz test, który: uruchamia kontener, wysyła obraz, sprawdza kod 200 i poprawność pola class.
    - **Materiały:** artykuł Test-driven FastAPI (Miguel Grinberg).
    - **Cel:** pytest -k integration przechodzi lokalnie i w GitHub Actions.
6. Prosty monitoring + logi — 4 h
    - **Co robisz:** Dodajesz logger (standard logging) z formatem JSON oraz middleware, który mierzy czas inferencji.
    - **Materiały:** rozdz. "Logging" w dokumentacji FastAPI.
    - **Cel:** każde wywołanie /predict zapisuje w logu: timestamp, filename, latency ms.
7. Retrospektywa i plan na kolejny miesiąc — 3 h
    - **Co robisz:** Spisujesz wrażenia z DVC i CI/CD, notujesz najdłuższy krok w workflow oraz pomysły na skalowanie.
    - **Cel:** plik june_notes.md z checklistą DONE/TO IMPROVE.

Jeśli masz ekstra wolny weekend (8h)

docker-compose & mini-stack — 4 h

- **Co robisz:** Definiujesz docker-compose.yml z dwoma usługami: api (FastAPI) i proxy (Traefik lub Nginx).
- **Cel:** docker-compose up wystawia API pod http://localhost/api/predict.

MLflow Tracking — 4 h

- **Co robisz:** Instalujesz MLflow, logujesz parametry z majowego treningu i porównujesz runy.
- **Materiały:** "MLflow Quickstart" + film Databricks 25 min.
- **Cel:** dashboard MLflow pokazuje min. 2 run'y z różnymi learning-rate.

Jeśli masz 2 ekstra wolne weekendy (16h)

Deploy w chmurze (Render / Fly.io / AWS EC2) — 8 h

- **Co robisz:** Konfigurujesz pipeline GitHub Actions, który po merge do main wykonuje render deploy lub fly deploy.
- **Cel:** publiczny URL https://cv-infer.myapp.dev/predict zwraca wynik w < 1 s dla obrazu 256×256.

Grafana + Prometheus — 4 h

- **Co robisz:** W kontenerze Prometheus scrapujesz endpoint /metrics (FastAPI middleware prometheus_fastapi_instrumentator).
- **Cel:** dashboard Grafany z Widgetem „Average Inference Latency".

Artykuł „Od notebooka do CI/CD w 30 dni" — 4 h

- **Co robisz:** Publikujesz krok-po-kroku z kodem i zrzutami ekranu.
- **Cel:** publiczny URL + ≥ 30 odsłon w pierwszym tygodniu.

Kamień milowy VI. miesiąca

"Hello MLOps" — repo zawiera:
- Dockerfile produkcyjny + workflow GitHub Actions budujący i pushujący obraz,
- aplikację FastAPI async z testami integracyjnymi,
- konfig DVC z danymi i wagami w zdalnym storage,
- logi latency i plik june_notes.md.

Opcjonalnie (czas extra): docker-compose, MLflow tracking oraz działające publiczne demo w chmurze z dashboardem Grafany.

Miesiąc VII. — Object Detection & Optymalizacja pod Edge

Miesiąc VII przerzuca Cię z klasyfikacji na “tu jest, wytnij i nazwij” — czyli detekcję obiektów. YOLOv8 to najszybsza droga: gotowy kod, pre-trening na COCO i jedno polecenie, by zacząć fine-tune. Konwertując swój kwietniowy zbiór do formatu YOLO i trenując model nano, uczysz się przygotowywać dane pod bounding-boxy oraz czytać metrykę mAP — walutę wszystkich projektów detekcyjnych.

Eksport ONNX i dynamiczna quantization pokazują, co znaczy „model na edge”. Liczysz latency CPU / GPU, kompresujesz INT8 i od razu widzisz, ile kosztuje każdy milisekundowy zysk. To kluczowa umiejętność, gdy aplikacja ma działać na Jetsonie czy w kamerze przemysłowej, a nie tylko w notebooku.

Włączenie YOLO do istniejącego FastAPI zamyka pętlę: dane → trening → produkcyjny endpoint. Od teraz Twoja usługa obsługuje zarówno /predict, jak i /detect, a Ty umiesz dostarczyć klientowi gotowy kontener, który w sekundę zwróci listę obiektów z koordynatami — dokładnie to, czego oczekuje rynek “computer-vision-as-a-service”.

1. Instalacja środowiska YOLOv8 – 4 h

- **Co robisz:** pip install ultralytics + onnxruntime
  krótkie demo yolo detect predict model=yolov8n.pt source=bus.jpg
- **materiały:** oficjalny README Ultralytics + film „YOLOv8 in 20 minutes” (Roboflow)
- **cel:** skrypt check_yolo.py drukuje nazwy klas i FPS na Twoim GPU / CPU

2. Konwersja własnego mini-zestawu do formatu YOLO – 6 h

- **Co robisz:** zrobione w kwietniu etykiety (bbox) eksportujesz z LabelImg do YOLO TXT
  piszesz split_yolo.py (80 / 20 train/val)
- **materiały:** blog Ultralytics „Convert datasets to YOLO format”
- **cel:** katalog dataset_yolo/ z plikiem data.yaml gotowym do treningu

3. Fine-tune YOLOv8n – 8 h
    - **Co robisz:** yolo detect train model=yolov8n.pt data=data.yaml epochs=50 imgsz=640
      śledzisz loss, mAP; zapisujesz wagi best.pt
    - **materiały:** Ultralytics docs sekcja Training tips
    - **cel:** mAP ≥ 0,25 (lub 3× lepiej niż random) na walidacji
4. Eksport, ONNX i pomiar opóźnienia – 6 h
    - **Co robisz:** yolo export format=onnx opset=12
      prosty skrypt benchmark.py mierzy średni czas inferencji na ✔ CPU ✔ GPU
    - **Materiały:** artykuł „PyTorch → ONNX → ONNX Runtime for YOLOv8" (A. Rosebrock)
    - **Cel:** plik benchmarks.md z tabelą FPS / latency ms
5. Quantization INT8 – 5 h
    - **Co robisz:** używasz onnxruntime.quantization.quantize_dynamic porównujesz dokładność versus szybkość (powtórny benchmark.py)
    - **Materiały:** tutorial ONNX „Dynamic Quantization" + wideo „INT8 in 10 min" (Microsoft)
    - **Cel:** zmniejszenie wagi modelu ≥ 4× przy spadku mAP < 3 pp
6. Integracja z FastAPI – 4 h
    - **Co robisz:** dodajesz nową ścieżkę POST /detect → zwraca listę bbox + klas + score
      logujesz czas inferencji i liczbę detekcji
    - **Cel:** curl -F image=@test.jpg localhost:8000/detect zwraca JSON z bbox
7. Retrospektywa & plan na kolejny miesiąc – 2 h
    - **Co robisz:** oceniasz jakość danych, notujesz najwolniejszy fragment pipeline
    - **Cel:** july_notes.md z TODO: np. więcej danych lub dalsza optymalizacja

Jeśli masz ekstra wolny weekend (8h)

Hyper-sweep W&B – 4 h

- **Co robisz:** konfigurujesz search po learning-rate i imgsz, wybierasz najlepszy run; cel: dashboard W&B z ≥ 6 runami.

TensorRT INT8 – 4 h

- **Co robisz:** yolo export format=engine i testujesz na GPU; cel: kolejne ≥ 30 % przyspieszenia w benchmarks.md.

Jeśli masz 2 ekstra wolne weekendy (16h)

Deploy na Jetson / Raspberry Pi + Coral – 8 h

- **Co robisz:** cross-build Dockera, optymalizacja CUDA_ARCH_BIN lub EdgeTPU Compiler
- **Cel:** kamera USB stream → średni FPS ≥ 15 na urządzeniu edge

Streamer RTSP + web-dashboard – 4 h

- **Co robisz:** używasz opencv VideoCapture + WebSocket, wyświetlasz bbox online
- **Cel:** lokalny adres http://localhost:8501 pokazuje live detections

Post „Jak spakowałem YOLO do 100 MB i działa w kieszeni" – 4 h

- **Co robisz:** publikujesz na Medium, dodajesz wideo z Jetsona; cel: ≥ 30 wyświetleń

Kamień milowy VII. miesiąca

"YOLO in the Wild" – repo zawiera:
・dataset_yolo/ z data.yaml,
・train_yolo.sh lub notebook z fine-tuningiem,
・modele best.pt, best.onnx, best_int8.onnx,
・tabela FPS/latency w benchmarks.md,
・endpoint /detect w FastAPI.
Dodatkowe punkty: sweep W&B, silnik TensorRT, edge-deploy i blog-post podlinkowane w README.

Miesiąc VIII. — Segmentacja obrazów i “pixel-level AI”

Miesiąc VIII przenosi Cię z “co jest w obrazku” do “który piksel do czego należy” — czyli segmentacji. Zaczynasz od ręcznego maskowania 100 zdjęć w Labelme, bo własne anotacje pokazują prawdziwy koszt danych i uczą, jak łatwo błąd rysika psuje metrykę. Potem dopuszczasz do pracy Segment Anything, żeby przyspieszyć etykietowanie, i szybko odkrywasz, że autolabel to świetny pomocnik, ale wymaga ludzkiej korekty.

Fine-tune Mask R-CNN w Detectron2 daje pierwszy “pixel-level” model z konkretnym mIoU, a notebook z overlayami Heat-map uczy patrzeć, gdzie sieć gubi krawędź. Dzięki eksportowi do ONNX i endpointowi /segment Twój serwer potrafi teraz zwracać gotową maskę w jednym request-response, co kończy się pełnym cyklem: zbiór → trening → API. W bonusowym czasie testujesz lżejsze DeepLab-Mobile i publikujesz demo w Streamlit, żeby każdy mógł wrzucić zdjęcie i zobaczyć magię segmentacji w przeglądarce.

1. Nowe repozytorium danych – 3 h

- **Co robisz:** Kopiujesz swój lipcowy zbiór i wybierasz 150 zdjęć, na których granica obiektu ma znaczenie (np. rysy, detale produktu, kształt liścia).
  Zakładasz folder dataset_seg/ z podfolderami images i masks.

2. Ręczna segmentacja ✍️ – 6 h

- **Co robisz:** Używasz Labelme lub CVAT w trybie polygon/brush; maska = PNG w odcieniach szarości (255 = obiekt, 0 = tło).
- **Cel:** 100 poprawnie zmaskowanych obrazów (wystarczy jedna klasa).

3. SAM (Segment Anything) inference – 5 h

- **Co robisz:** Instalujesz segment-anything + PyTorch ≥ 2.1, pobierasz punktowe wagi sam_vit_b.pth.
  Skrypt sam_autolabel.py generuje masky dla kolejnych 50 zdjęć.
- **Cel:** folder sam_masks/ do późniejszej korekty.

4. Łączenie i czyszczenie masek – 3 h
    - **Co robisz:** Ręcznie poprawiasz błędy SAM-a w Labelme (najgrubsze dziury, przekroczenia granicy).
    - **Cel:** kompletne dataset_seg/ z ≥ 150 parą image + mask.
5. Fine-tune Mask R-CNN (Detectron2) – 9 h
    - **Co robisz:** Instalujesz Detectron2; konfigurujesz config.yaml (backbone = ResNet-50-FPN, lr = 0.0025, iters = 1500).
      Trenowanie na swoim GPU; zapisujesz model_final.pth.
    - **Materiał:** oficjalny tutorial "Detectron2 for custom datasets".
    - **Cel:** AP 50 ≥ 0,55 (lub 3× losowy) na walidacji.
6. Ewaluacja i wizualizacja – 4 h
    - **Co robisz:** Tworzysz eval_seg.py → confusion matrix pikselowa, Dice Coefficient, mIoU.
      Zapisujesz overlay PNG (mask + półprzezroczysty oryginał) dla 20 zdjęć.
    - **Cel:** folder vis/ i plik metrics_seg.json.
7. Integracja z API – 3 h
    - **Co robisz:** W FastAPI dodajesz POST /segment; zwracasz PNG maski + czas inferencji w nagłówku.
    - **Cel:** curl -F image=@test.jpg localhost:8000/segment --output mask.png.
8. Retrospektywa & plan na kolejny miesiąc – 2 h
    - **Co robisz:** Sprawdzasz, które błędy → dane? model? post-proc? Zapisujesz w august_notes.md listę pomysłów (np. CRF-refine, więcej klas).

Jeśli masz ekstra wolny weekend (8h)

Augmentacje przestrzenne + fotometryczne – 4 h

- **Co robisz:** Dodajesz do pipeline Albumentations: RandomRotate90, ElasticTransform, RandomBrightnessContrast.

- **Cel:** +2 pp mIoU przy tym samym modelu.

Streamlit demo – 4 h

- **Co robisz:** Budujesz appkę app.py: upload obrazu → pokazuje maskę w przeglądarce.
- **Cel:** local URL localhost:8501 działa, GIF w README.

Jeśli masz 2 ekstra wolne weekendy (16h)

Distil-Seg / DeepLabV3 INT8 – 6 h

- **Co robisz:** Próbujesz lżejszej sieci (DeepLabV3-MobileNetV3); eksport do ONNX + dynamic quantization.
- **Cel:** plik deeplab_mobile_int8.onnx, latency CPU-only ≤ 120 ms.

CRF-post-processing + TTA – 4 h

- **Co robisz:** Implementujesz DenseCRF (pydensecrf) i test-time-augmentation (flip, scale) na 5 próbkach.
- **Cel:** +1 pp mIoU bez zwiększania czasu > 30 %.

Artykuł "Od Segment Anything do własnego Mask R-CNN" – 4 h

- **Co robisz:** Publiczny post krok po kroku; cel: ≥ 30 odsłon i min. 1 komentarz.

Test E2E w CI – 2 h

- **Co robisz:** Dodajesz do GitHub Actions test, który wysyła obraz i sprawdza, że zwrócony PNG ma ≥ 1 % pikseli „1".
- **Cel:** job test_segment passed.

Kamień milowy VIII. miesiąca

"Pixel-Perfect" – w repo znajduje się:
• dataset_seg/ z ≥ 150 maskami,
• notebook/skrypt sam_autolabel.py,
• trening Detectron2 z modelem model_final.pth,
• metrics_seg.json + overlay w vis/,
• endpoint /segment w FastAPI.
Bonusy: Streamlit demo, Distil-Seg INT8, artykuł i pełna CI z testem pikseli.

Miesiąc IX. — „Wyjście z płaskiego świata": pierwsze kroki w 3-D Vision & SLAM

Miesiąc IX otwiera trzeci wymiar – uczy, jak zamienić dwa zdjęcia w głębię i chmurę punktów. Kalibracja pojedynczej kamery i pary stereo to pierwszy krok: drukujesz szachownicę, liczysz macierz K, oglądasz, jak drobiazgowe 0,4 px błędu zdecyduje o jakości całej geometrii. Potem Stereo-SGBM + Open3D pokazują, że zwykłe kamery mogą dać prawdziwy „depth map" i plik PLY, który obracasz w 3-D viewerze – świetne przygotowanie do LiDAR-ów i robotyki.

COLMAP i ORB-SLAM3 dorzucają rekonstrukcję wielozdjęciową i ruch kamery w czasie. Kilkadziesiąt zdjęć obiektu lub krótki film z telefonu przekształcasz w gęstą chmurę, poznając po drodze pojęcia bundle-adjustment i drift trajektorii. To fundamentalne, jeśli myślisz o AR, dronach czy autonomicznych wózkach AGV.

Na koniec dobudowujesz endpoint /depth, który z pary obrazów zwraca mapę głębi, i benchmarkujesz go co do milisekundy. Od teraz Twój serwer CV odpowiada nie tylko „co" i „gdzie", ale też „jak daleko" – czyli kompletny pakiet percepcji, gotowy do zasilenia kolejnych projektów robotycznych czy analizy scen 3-D.

1. Model kamery i kalibracja – 5 h

- **Co robisz:** czytasz o modelu otworkowym, zniekształceniach radialnych; drukujesz szachownicę 9×6, nagrywasz 20 zdjęć z różnych kątów.
- **Materiały:** blog PyImageSearch "Camera Calibration with OpenCV" + rozdz. 1 książki Multiple View Geometry (Hartley & Zisserman, skondensowana wersja).
- **Cel:** skrypt calibrate.py zapisuje do pliku intrinsics.yaml macierz K i współczynniki dystorsji; RMS error < 0,4 px.

2. Stereo → mapa dysparycji – 7 h

- **Co robisz:** montujesz dwa telefony / kamerki 7-10 cm od siebie, przeprowadzasz kalibrację stereo, używasz cv2.StereoSGBM_create.

- **Materiały:** oficjalny tutorial OpenCV Stereo Correspondence + film "Stereo Depth in 15 Minutes" (Computerphile).
- **Cel:** notatnik stereo_depth.ipynb wyświetla obraz disparity i zapisuje 16-bitowy TIFF depth.tiff.

3. Chmura punktów z Open3D – 5 h

- **Co robisz:** Ładujesz disparity + parametry kamery, wyliczasz chmurę punktów Open3D.geometry.PointCloud, filtrujesz outliery.
- **Materiały:** Open3D Official Tutorials → RGB-D Integration.
- **Cel:** plik cloud.ply (≥ 50 k punktów) widoczny w Open3D viewer.

4. COLMAP multi-view reconstruction – 6 h

- **Co robisz:** Robisz 40 zdjęć obiektu (orbita 360°), uruchamiasz pipeline COLMAP (automatic reconstruction).
- **Materiały:** wiki COLMAP + wideo "Photogrammetry in COLMAP" (3 D Scan Academy).
- **Cel:** pliki sparse/0/ + dense/fused.ply; zapisujesz liczbę punktów i błąd reprojekcji.

5. ORB-SLAM3 demo – 6 h

- **Co robisz:** Kompilujesz ORB-SLAM3, uruchamiasz na własnym krótkim filmie z telefonu (monocular) lub na zbiorze Euroc.
- **Materiały:** README ORB-SLAM3 + blog Gaoxiang "ORB-SLAM explained".
- **Cel:** wykres trajektorii trajectory.txt i rzut na oś X-Y (matplotlib); ocena czy dryf ≤ 1 % przebytej trasy.

6. API „/depth" – 4 h

- **Co robisz:** Dodajesz do FastAPI endpoint POST /depth – przyjmuje obraz stereo L+R, zwraca mapę głębi (PNG) oraz średnią głębię w nagłówku.
- **Cel:** curl -F left=@L.png -F right=@R.png localhost:8000/depth --output depth.png działa w < 1 s dla 640×480.

7. Retrospektywa i plan na kolejny miesiąc – 2 h
    - **Co robisz:** Piszesz w september_notes.md: jakie błędy (szum, tekstury) zabiły stereo, co ulepszyć (Baseline? lepszy algorytm?).

Jeśli masz ekstra wolny weekend (8h)

Monokularna głębia MiDaS – 5 h

- **Co robisz:** Instalujesz midas (timm). Skrypt midas_depth.py oblicza pseudo-depth dla pojedynczego obrazu, normalizuje 0-1.
- **Cel:** porównanie histogramu głębi MiDaS vs stereo w depth_compare.md.

Meshing i render on-line – 3 h

- **Co robisz:** Używasz open3d.geometry.TriangleMesh.create_from_point_cloud_poisson; zapisujesz mesh.ply, oglądasz w Meshlab.
- **Cel:** mesh z widocznym kształtem obiektu (brak dziur > 5 % powierzchni).

Jeśli masz 2 ekstra wolne weekendy (16h)

PointNet ++ na Twojej chmurze – 6 h

Deploy w chmurze (Render / Fly.io / AWS EC2) — 8 h

- **Co robisz:** Z cloud.ply tworzysz npz (N×3), trenujesz prosty klasyfikator / segmentator w repo PointNet++.
- **Cel:** ≥ 80 % accuracy segmentacji „obiekt / tło”.

WebGL viewer (Potree) – 4 h

- **Co robisz:** Konwertujesz cloud.ply → PotreeConverter; hostujesz statycznie na GitHub Pages.
- **Cel:** publiczny URL z interaktywną chmurą (orbit, zoom).

Artykuł „Od szachownicy do chmury punktów” – 4 h

- **Co robisz:** Publikujesz na Medium / dev.to, dołączasz GIF z trajektorią ORB-SLAM.

- **Cel:** ≥ 40 odsłon / 2 komentarze.

CI-benchmark 3-D – 2 h

- **Co robisz:** Dodajesz test GitHub Actions, który pobiera przykładowe stereo, liczy depth i sprawdza, że średnia głębia > 0.
- **Cel:** workflow 3d_test zielony.

Kamień milowy IX. miesiąca

“Hello 3-D!” – repo zawiera:
• intrinsics.yaml + stereo_depth.ipynb,
• cloud.ply (Open3D) oraz dense/fused.ply z COLMAP,
• trajektorię ORB-SLAM trajectory.txt + wykres,
• endpoint /depth działający z FastAPI,
a w wersji rozszerzonej: MiDaS, mesh, PointNet++, viewer Potree i artykuł podlinkowane w README.

Miesiąc X. — Monitoring, drift i automatyczny retrain: „życie modelu po deplojmencie”

Miesiąc X włącza modelowi “zmysł samokontroli”. Po pierwszym zachwycie produkcyjnym przychodzi rzeczywistość: obraz z nowej kamery ma inne kolory, klient zaczyna wysyłać GIF-y zamiast JPG-ów, a latency skacze, gdy ktoś w firmie odpali kompilację. Prometheus + Grafana instalujesz więc jako cyfrę pulsu: dashboard pokazuje średni i p99 dla /detect, /segment, /depth, a middleware loguje, co wpada do sieci i ile trwa odpowiedź.

DVC-pipeline “nightly retrain” uczy, że dane żyją. Middleware wrzuca każde zapytanie do kolejki, batch-evaluator mierzy nową jakość, a detektor driftu z Evidently odpala alarm, jeśli rozkład pikseli odjedzie od treningowego. Gdy metryka spadnie, GitHub Actions uruchamia retrain i commituje świeże wagi na gałęź autoupdate. W ten sposób serwer sam leczy się z powolnego starzenia danych, zanim zgłosi to klient.

Alertmanager, test integracyjny i Model Card domykają całość zgodnie z praktyką DevOps/MLOps. Kiedy zasymulujesz awarię (negatyw obrazu lub kontener K.O.), mail i Slack ping informują o dryfie lub błędzie 5xx; jednocześnie w dokumentacji pojawia się nowy numer wersji i hash modelu. Od tej chwili Twój system CV nie tylko działa — on się obserwuje, diagnozuje i aktualizuje niczym dojrzały mikro-serwis w dużej organizacji.

1. Prometheus + Grafana w 20 minut — 5 h
    - **Co robisz:** Instalujesz Prometheus i Grafanę w docker-compose.yml; Dodajesz do FastAPI middleware prometheus_fastapi_instrumentator.
    - **Materiały:** blog PromLabs „Prometheus in Docker” + film Grafana „Getting Started” (30 min).
    - **Cel:** dashboard „Model latency & throughput” pokazuje średnią i p99 dla /detect, /segment, /depth.
2. Kolejka surowych zapytań — 5 h
    - **Co robisz:** Uruchamiasz Redis Streams lub Kafka (Docker); W middleware zapisujesz każde żądanie + odpowiedź do kolejki (klucz = hash obrazu).

- **Cel:** redis-cli XLEN inference-stream zwraca rosnący licznik, a script consumer.py potrafi pobrać i zrzucić plik JPG + JSON metryk.

3. Batch-evaluator i metryki jakości — 6 h

- **Co robisz:** Piszesz skrypt nightly_eval.py: pobiera 200 najnowszych próbek, liczy precision/recall (porównuje z maskami GT jeśli są) lub proxy-metrykę (ocena manualna 0/1 w CSV);
  Wyniki pushuje do Prometheus jako custom_quality.
- **Materiały:** biblioteka evidently docs „Batch Evaluation".
- **Cel:** na dashboardzie pojawia się wykres „Model precision last N" z punktacją >= tej, którą miałeś w maju/sierpniu.

4. Detektor driftu danych — 6 h

- **Co robisz: U**żywasz evidently lub river do statystycznego porównania rozkładu pikseli, rozmiaru bbox, głębi;
  Próg alarmu zapisujesz w Prometheus Alertmanager.
- **Cel:** symulujesz „dziwną" zmianę danych (negatyw obrazu), AlertManager wysyła ci e-mail „Possible input drift".

5. Automatyczny retrain pipeline — 7 h

- **Co robisz:** Tworzysz plik dvc.yaml z etapami: fetch-data (pull z Redis), train, eval, push-model;
  W GitHub Actions dodajesz job „nightly-retrain" na gałęzi autoupdate.
- **Materiały:** Iterative – „DVC Pipelines" + film „CI for ML with DVC & GitHub Actions" (45 min).
- **Cel:** zielony job w Actions, który po przejściu testu jakości commit-uje model-YYYYMMDD.onnx do branch autoupdate.

6. Model Card + Data Sheet — 4 h

- **Co robisz:** W docs/ tworzysz model_card.md (cel, dane, metryki, ryzyka) i dataset_sheet.md;
  Dodajesz wersję i hash commit wagi modelu.
- **Materiał:** Google Model Cards Toolkit „Template".
- **Cel:** oba pliki zawierają wymagane sekcje; link do HTML-rendera w README.

7. Retrospektywa i plan na kolejny miesiąc — 2 h
    - **Co robisz:** W october_notes.md zapisujesz, które alerty były fałszywe, a co wymaga lepszych progów; notujesz, czy koszt GPU-retrain zmieścił się w budżecie.

Jeśli masz ekstra wolny weekend (8h)

Sentry + Slack alerty – 4 h

- **Co robisz:** Integrujesz Sentry SDK z FastAPI, konfigurujesz web-hook Slack;
- **Cel:** gdy infer.py rzuci wyjątek, wiadomość trafia na kanał #cv-alerts.

Canary deploy na Render/Fly.io – 4 h

- **Co robisz:** Tworzysz drugą usługę cv-infer:v-next, ruch 10 % → nowa wersja;
- **Cel:** metryka błędów 5xx nie wzrasta > 1 pp, po godzinie canary promuje się do stable.

Jeśli masz 2 ekstra wolne weekendy (16h)

Kubeflow Pipelines – 6 h

- **Co robisz:** Na minikube instalujesz Kubeflow, zamieniasz DVC → komponenty Kubeflow (fetch → train → eval → push);
- **Cel:** UI Kubeflow pokazuje DAG, status zielony.

A/B testing (50-50) – 4 h

- **Co robisz:** W NGINX In-gress konfigurujesz header-based routing model-version, logujesz wyniki;
- **Cel:** plik ab_results.md – która wersja ma lepszy F1 i latency.

Artykuł „Od logów do autopilota: monitoring CV w praktyce” – 4 h

- **Co robisz:** Publikujesz zrzuty ekranowe Grafana, Alertmanager, Kubeflow;
- **Cel:** ≥ 40 odsłon i co najmniej 1 udostępnienie na LinkedIn.

Chaos test – 2 h

- **Co robisz:** Piszesz skrypt, który w losowym momencie zabija kontener api;
- **Cel:** Kubernetes lub docker-compose automatycznie restartuje usługę, monitor pozwala to zauważyć.

Kamień milowy X. miesiąca

"Model żyje i sam o sobie dba" — w repo pojawiają się:
• docker-compose.yml z Prometheus + Grafana + (ew. Redis);
• middleware i dashboard latency/quality;
• pipeline DVC (lub Kubeflow) nightly-retrain, commitujący nowe wagi;
• Alertmanager/sentry wysyłający powiadomienia;
• model_card.md, dataset_sheet.md, october_notes.md.
Bonusy: canary deploy, A/B testing, artykuł i chaos-test opisane w README.

11. Listopad — Etyka, interpretowalność i zgodność z prawem: „czy mój system CV jest bezpieczny i fair?"

Miesiąc XI sprawdza, czy Twój system jest nie tylko skuteczny, ale i sprawiedliwy. Raporty z evidently-ai/Aequitas ujawniają, która klasa lub scenariusz cierpi na najwyższy false-negative; Captum i Grad-CAM++ dorzucają kolorowe heat-mapy, dzięki którym zobaczysz, czy model patrzy na obiekt, czy na tło. Testy adversarialne (FGSM, PGD) pokazują drugą stronę medalu: jak bardzo można go oszukać jednym pikselowym szumem.

Prywatność i zgodność z prawem wchodzą od razu w kod. Skrypt privacy.py rozmywa twarze — prosta, ale wymierna demonstracja, że potrafisz wdrożyć anonimizację przed wysłaniem danych do chmury. Następnie wypełniasz arkusz oceny ryzyka według draftu EU AI Act, który klasyfikuje systemy CV i wymusza listę kontroli (high-risk lub limited). W praktyce to pierwszy krok do rozmowy z działem prawnym dowolnej korporacji.

Wszystko ląduje w zaktualizowanej Model Card, a jeśli masz dodatkowy czas — przeprowadzasz mitigację biasu lub trenujesz wariant DP-SGD dla ochrony

danych. Tak powstaje „compliance binder": jeden PDF z metrykami fairness, raportem robust-attack i oceną AI Act — gotowy argument, że Twój produkt nie tylko działa, ale da się go bezpiecznie wprowadzić na rynek UE.

1. Diagnoza biasu i metryki fairness — ok. 6 h
    - **Co robisz:** Instalujesz bibliotekę evidently-ai lub Aequitas i przepuszczasz przez nią swoje dane z maja/sierpnia (klasyfikacja i segmentacja).
      Liczysz: procent fałszywych alarmów w każdej klasie, equal opportunity difference, statistical parity.
    - **Cel:** plik fairness_report.html + krótki komentarz, która grupa/wzorzec jest najbardziej poszkodowany.

2. Interpretowalność modelu — ok. 8 h
    - **Co robisz:** Dodajesz do projektu Captum (klasyfikacja) i Grad-CAM++ (detekcja i segmentacja).
      Tworzysz notebook explainability.ipynb generujący mapy uwagi dla 20 poprawnych i 20 błędnych predykcji; zapisujesz PNG.
    - **Cel:** folder explain/ zawiera heat-mapy, a w notatniku potrafisz wskazać, dlaczego model myli dwie najtrudniejsze klasy.

3. Robustness: test adversarialnych ataków — ok. 6 h
    - **Co robisz:** Instalujesz Foolbox lub CleverHans; generujesz FGSM (ε=4/255) i PGD (ε=8/255) przeciwko sieci z maja.
      Mierzysz spadek accuracy i mAP, zapisujesz w robustness.md.
    - **Cel:** znasz dokładne liczby („Model traci 27 pp accuracy przy PGD ε=8/255") i masz przykładowe obrazy ataku w adv_examples/.

4. Prywatność & GDPR: anonimizacja obrazu — ok. 4 h
    - **Co robisz:** Piszesz moduł privacy.py: wykrywa twarze (Haar-cascade albo YOLO) i rozmywa je gaussowsko; obsługuje batch folderu.
    - **Cel:** skrypt python privacy.py input/ output/ tworzy kopię zbioru z rozmytymi twarzami w < 1 s na obraz 1280×720.

5. Ocena ryzyka wg EU AI Act — ok. 6 h
    - **Co robisz:** Pobierasz draft EU AI Act; wypełniasz szablon „AI Risk Assessment" (dostępny w repo odpr. Fathon Consulting). Kategoryzujesz swój system (prawdopodobnie „Limited" lub „High risk": monitoring przemysłowy).
    - **Cel:** plik ai_act_assessment.pdf podpisany datą, z tabelą wymagań i wskazaniem, które spełniasz / plan działania na brakujące punkty.

6. Aktualizacja Model Card — ok. 3 h
    - **Co robisz:** Uzupełniasz sekcję „Fairness & Privacy" o wyniki metr yk, samouczek anonimizacji i opis wpływu ataków.
    - **Cel:** docs/model_card.md zawiera nowe rozdziały + link do fairness_report.html.

7. Retrospektywa i plan na kolejny miesiąc — 2 h
    - **Co robisz:** W november_notes.md zapisujesz, które ryzyka są krytyczne, ile spadła dokładność po anonimizacji i co chcesz poprawić (np. balans klas).

Jeśli masz ekstra wolny weekend (8h)

Mitigacja biasu i re-audyt – ok. 6 h

- **Co robisz:** Stosujesz jedną metodę: re-weighting klas lub oversampling najmniej reprezentowanej.
  Trenujesz szybką wersję modelu, ponownie uruchamiasz fairness_report.
- **Cel:** poprawa najgorszej metryki o ≥ 10 %.

Minipost „SHAP w Computer Vision" – ok. 2 h

- **Co robisz:** Piszesz notatkę na LinkedIn z jednym GIF-em: heat-mapa przed/po.
- **Cel:** ≥ 15 wyświetleń w 48 h.

Jeśli masz 2 ekstra wolne weekendy (16h)

Differential Privacy-SGD – 6 h

- **Co robisz:** Używasz Opacus (PyTorch) i trenujesz model na zaszumionych gradientach (ε = 8).
- **Cel:** accuracy spada < 5 pp względem zwykłego modelu, privacy budget obliczony w raporcie.

Red-teaming & pentest CV – 4 h

- **Co robisz:** Symulujesz „nocny" feed z kamerą IR, zmieniasz balans bieli, oceniasz recall detektora; zapisujesz wnioski.

Binder zgodności (single-PDF) – 4 h

- **Co robisz:** Łączysz Model Card, Data Sheet, AI Act assessment i raport bias w jeden PDF.
- **Cel:** compliance_binder.pdf gotowy do pokazania potencjalnemu klientowi korporacyjnemu.

Webinar 10-min „Etyka CV w praktyce" – 2 h

- **Co robisz:** Nagrywasz krótkie wideo (OBS), wrzucasz na YouTube jako Niepubliczne; link w README.

Kamień milowy XI. miesiąca

„Fair & Safe" — projekt posiada:

- raport fairness_report.html,
- notebook explainability.ipynb z heat-mapami,
- robustness.md + folder adv_examples/,
- moduł privacy.py (rozmywanie twarzy),
- formalny ai_act_assessment.pdf,
- zaktualizowaną Model Card z sekcją fairness/robustness,

oraz w wersji rozszerzonej: wyniki mitigacji biasu, DP-SGD model, compliance binder PDF i link do webinaru.

Miesiąc XII. — Zamknięcie Roku & Start na Rynek: portfolio, rekrutacje, open-source

Miesiąc XII zamyka pętlę: zamieniasz rok nauki w ofertę dla rynku. Porządkujesz kod w release v1.0.0 – jedno repo, każdy folder z README i docker run – żeby rekruter lub klient mógł uruchomić demo bez pytań. Na GitHub Pages / Netlify stawiasz lekką stronę z GIF-ami projektów, linkami do dashboardów Grafany i PDF-ami Model Card – to Twoja wizytówka, którą można otworzyć na telefonie podczas rozmowy.

Drugi filar to społeczny dowód kompetencji. Merdżujesz choćby drobną poprawkę w OpenCV czy Ultralytics, pokazując, że umiesz pracować z code-review. Jednocześnie robisz dwa mock-interview, podbijasz LinkedIn i wrzucasz CV z konkretnymi KPI (mAP, latency, p99). Gdy przycisk “Apply” pójdzie w ruch, masz: porfolio online, zmergowany PR i tag v1.0.0 – komplet, by wejść na rynek jako Junior Computer Vision Engineer.

1. Porządek w repozytorium – 6 h
    - **Co robisz:** scalasz kod zebrany przez rok w monorepo lub dobrze opisane sub-repo (classification, detection, segmentation, 3-D, MLOps);
      Każde pod-repo ma plik README.md z: opisem problemu, instrukcją docker run …, linkiem do demo;
      Commitujesz tag v1.0.0 i generujesz Release na GitHub z changelogiem.
2. Portfolio online – 8 h
    - **Co robisz:** zakładasz stronę GitHub Pages (lub Netlify) z szablonu Hugo / Next.js;
      sekcje: O mnie, Projekty (linki do repo, gif/youtube demo), Blog/Artykuły, Kontakt;
      dodajesz faviconę, własną domenę (jeśli chcesz) i Google Analytics.
    - **Cel:** publiczny adres np. https://jan-ai.dev dostępny z telefonu.
3. Open-source contribution – 6 h
    - **Co robisz:** Wybierasz bibliotekę, z której korzystałeś (OpenCV, Ultralytics, Albumentations);

- otwierasz Issue → PR: poprawka literówki w docs lub drobny przykład notebooka;
  przechodzisz code-review maintainerów.
- **Cel:** zaakceptowany i zmergowany PR + link w sekcji "Contributions" na stronie.

4. Ćwiczenia rekrutacyjne – 7 h

- **Co robisz:** 40 zadań LeetCode „array / string / hashmap" (utrwalenie algorytmów);
  2 mock-interview techniczne (Pramp, Interviewing.io) + 1 system-design: „Realtime object detection pipeline";
  zapisujesz pytania, na które nie odpowiedziałeś pewnie.
- **Cel:** wynik mock ≥ „Strong Hire" lub feedback 4 / 5 w ocenie partnera.

5. Uzupełnienie dokumentów aplikacyjnych – 6 h

- **Co robisz:** Aktualizujesz CV (dwie strony, PDF) – dodajesz KPI projektów (mAP, mIoU, latency);
  piszesz uniwersalny list motywacyjny + 3 wersje pod konkretne firmy (startup, korpo, software-house);
  profil LinkedIn: banner, summary w pierwszej osobie, sekcja Projects z linkami.
- **Cel:** wszystkie dokumenty umieszczone w Dropbox / Google Drive z krótkim linkiem bit.ly.

6. Retrospektywa roczna + plan na Rok 2 – 2 h

- **Co robisz:** W december_notes.md spisujesz: TOP 3 sukcesy, TOP 3 bóle, metryki (commity, PR-y, wyświetlenia bloga, certyfikaty), listę celów na kolejne 12 mies.;
  commitujesz roadmap_year2.md (np. "3-D LiDAR → Edge TPU, Leadership, Public Talk").

Jeśli masz ekstra wolny weekend (8h)

Certyfikat NVIDIA DLI "Fundamentals of Deep Learning" – 6 h

- **Co robisz:** kończysz laby + quiz on-line, pobierasz PDF certyfikatu; link do portfolio.

Setup newslettera – 2 h

- **Co robisz:** MailerLite/Substack: tytuł, opis i formularz zapisu na stronie; zapowiadasz serię „CV w praktyce".

Jeśli masz 2 ekstra wolne weekendy (16h)

Mini-projekt świąteczny "Style-Transfer Card" – 6 h

- **Co robisz:** Wżywasz diffusion lub fast-style-transfer, generujesz kartkę AI, publikujesz repo "xmas-style-transfer".

Webinar live "Rok w Computer Vision od zera" – 4 h

- **Co robisz:** 20-min prezentacja na YouTube/LinkedIn Live; Q&A; później umieszczasz nagranie na stronie.

Batch aplikacji – 4 h

- **Co robisz:** identyfikujesz 30 ofert „Junior / Graduate Computer Vision"; personalizujesz CV/LM; wysyłasz; logujesz follow-up w arkuszu.

Open-source sprint weekend – 2 h

- **Co robisz:** uczestniczysz w Hacktoberfest-like evencie (jeśli jeszcze trwa) lub lokalnym hackathonie AI; dokumentujesz w repo.

Kamień milowy XII. miesiąca:

"Gotowy do gry" — masz:
- release v1.0.0 uporządkowanego kodu,
- publiczne portfolio WWW z działającymi linkami do demo,
- co najmniej jeden zmergowany PR w projekcie open-source,
- aktualne CV + profil LinkedIn,
- log z mock-interview i listę wysłanych aplikacji,
- plik december_notes.md z retrospektywą i roadmap_year2.md z celami.

Checklista - Rok z Computer Vision

Miesiąc I – Setup & Flow

- ☐ Python 3.11 + VS Code + Docker zainstalowane
- ☐ Repozytorium sciezka-kariery-computer-vision utworzone
- ☐ Plik README.md, .gitignore, folder journal/
- ☐ Pierwszy test pytest działa lokalnie
- ☐ Commit: Miesiąc I: środowisko gotowe

Miesiąc II – Python & NumPy

- ☐ 50 zadań z Exercism/LeetCode ukończonych
- ☐ numpy_playground.ipynb z własnymi operacjami
- ☐ Własny pakiet matrix_utils + testy
- ☐ Commit: Miesiąc II: NumPy w palcach

Miesiąc III – Matematyka modeli

- ☐ Gradient descent zaimplementowany (notebook gradient_descent.ipynb)
- ☐ Wizualizacja spadku kulki lub straty
- ☐ Notatka journal/miesiac-3.md gotowa
- ☐ Commit: Miesiąc III: gradient działa

Miesiąc IV – OpenCV & dane

- ☐ 500+ zdjęć, oznaczenia ręczne lub Labelme
- ☐ Augmentacja w Albumentations
- ☐ Skrypt filter_baseline.py + wyniki
- ☐ Commit: Miesiąc IV: dane gotowe

Miesiąc V – Klasyfikacja & transfer learning

- ☐ Model FastAI wytrenowany na własnym zbiorze
- ☐ Grad-CAM + confusion matrix
- ☐ Eksport ONNX + skrypt infer.py
- ☐ Commit: Miesiąc V: model rozpoznaje

Miesiąc VI – API & produkcja

- ☐ FastAPI endpoint /predict
- ☐ Dockerfile + docker run działa
- ☐ CI: testy + budowanie kontenera
- ☐ DVC: wersjonowanie danych/modelu
- ☐ Commit: Miesiąc VI: model jako usługa

Miesiąc VII – Detekcja (YOLO)

- ☐ Konwersja danych do YOLO format
- ☐ Fine-tune YOLOv8 + mAP ≥ 0.25
- ☐ Eksport ONNX + endpoint /detect
- ☐ Commit: Miesiąc VII: YOLO działa

Miesiąc VIII – Segmentacja

- ☐ Ręczne maski + Segment Anything
- ☐ Fine-tune Mask R-CNN lub DeepLab
- ☐ API /segment + test działania
- ☐ Commit: Miesiąc VIII: każdy piksel przypisany

Miesiąc IX – 3-D Vision & głębia

- ☐ Kalibracja kamery + intrinsics.yaml
- ☐ Mapa głębi + depth.tiff
- ☐ Chmura punktów cloud.ply
- ☐ Commit: Miesiąc IX: głębia zadziałała

Miesiąc X – Monitoring & retrain

- ☐ Dashboard Grafana: latency + quality
- ☐ Drift detection działa (Evidently)
- ☐ DVC pipeline nightly-retrain
- ☐ Commit: Miesiąc X: model sam się pilnuje

Miesiąc XI – Fairness & interpretowalność

- ☐ Raport biasu (fairness_report.html)
- ☐ Heat-mapy: Captum / Grad-CAM
- ☐ Adversarial attack: FGSM / PGD
- ☐ Ocena AI Act + model_card.md
- ☐ Commit: Miesiąc XI: system bezpieczny

Miesiąc XII – Portfolio & deploy

- ☐ Portfolio WWW z projektami
- ☐ Release v1.0.0 na GitHubie
- ☐ Mock-interview przećwiczone
- ☐ CV + profil LinkedIn gotowe

Commit: Miesiąc XII: gotowa/y do gry

Podsumowanie

Za nami długa droga — dwanaście miesięcy, w czasie których pot nieraz spływał z czoła i nie raz, nie dwa byliśmy o krok od rzucenia klawiaturą w ścianę… Ale przetrwaliśmy. Ramię w ramię. Wspólnie szukaliśmy bugów, próbowaliśmy nie usnąć w czasie nightly-retrainów i zrobiliśmy wszystkie zadania i projekty… No dobrze, prawie wszystkie, ale tamten jeden czy dwa nawet się nie liczą, bo temat kompletnie nam nie podszedł i w ogóle to był dziwny miesiąc (i to też jest okej). Dziś masz release v1.0, portfolio online, działające API i świadomość, że potrafisz przejść od pustego repo do produkcyjnego systemu CV.

Ale to wcale nie jest meta. AI pędzi do przodu szybciej niż patch-notesy TensorFlow, a kariera deva to zdecydowanie bieg długodystansowy. Już szykujemy Kontynuację Ścieżki — rok drugi, z LiDAR-ami, Edge TPU, leadershipem i green-AI — bo ten zawód wymaga nieustannego upgrade'u. Zanim jednak wskoczysz do kolejnego sprintu, złap oddech, zjedz lody pistacjowe, poklep się po plecach i powiedz głośno: YOU DID IT!

Jesteśmy z Ciebie dumni! To najlepszy commit, jaki mogłeś/mogłaś wrzucić do własnej gałęzi „życie-zawodowe".