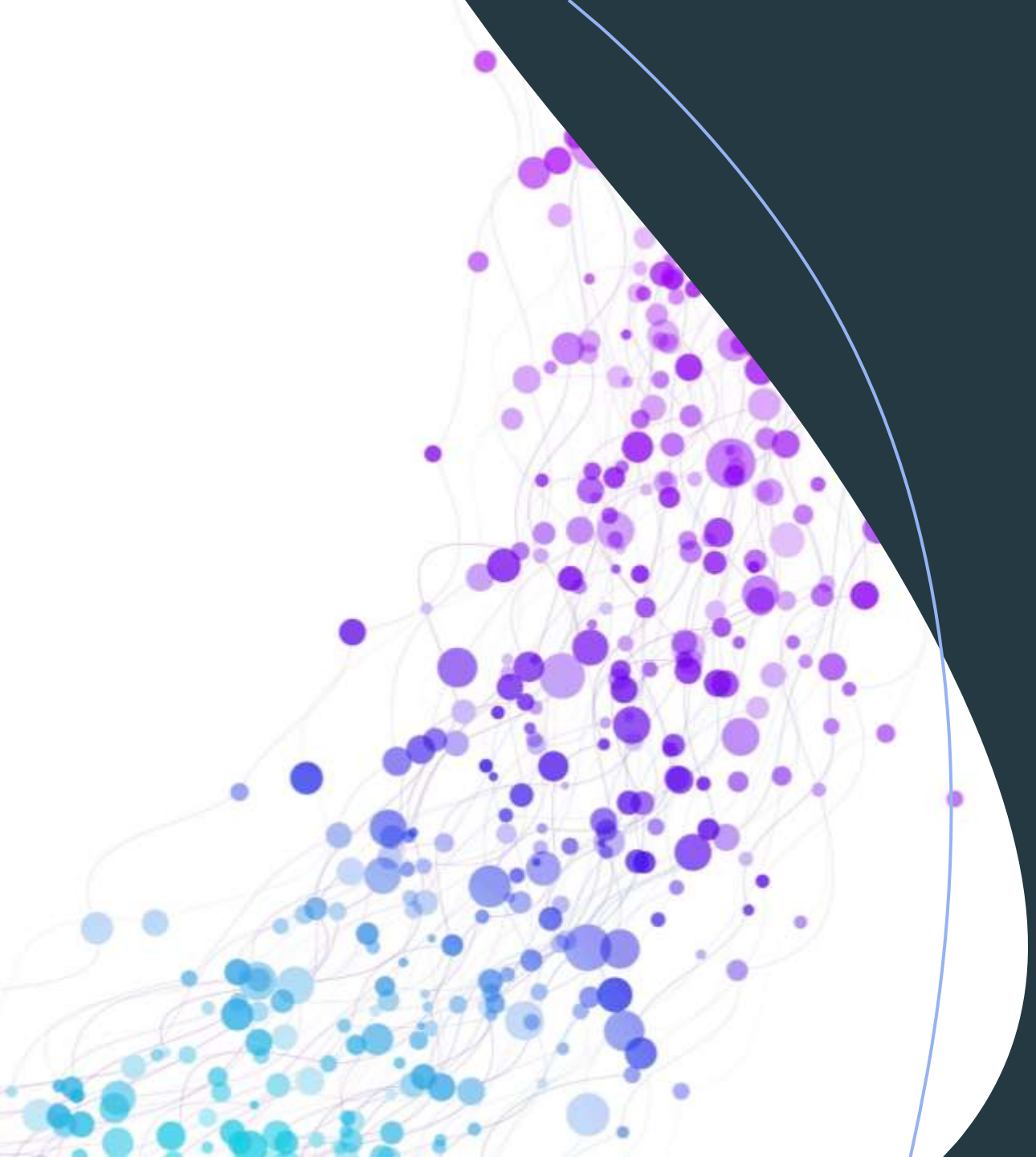


Echtzeit-Erkennung von Gebärden/Handzeichen (A–Y) mit Python & Mediapipe

Oguzhan Yildirim & Dervis Pivolic

Giederung

- Motivation & Problemstellung
- Zielsetzung & Projektüberblick
- Datenakquise und –aufbereitung
- Trainingsprozess & Modellierung
- Architektur & Skriptstruktur
- Demo
- Ergebnisse & Ausblick
- Fragen

Motivation & Problemstellung

- Motivation
  - Gebärdensprache (oder vereinfachte Handzeichen) automatisiert erkennen.
  - Erleichterung der Kommunikation zwischen hörenden und gehörlosen Menschen.
  - Einsatzmöglichkeiten in Bildung, Telemedizin, Barrierefreiheit etc.

Motivation & Problemstellung

- Problemstellung
  - Wie können wir mithilfe einfacher Computer-Vision-Methoden (z.B. Mediapipe) und Machine Learning Handzeichen robust erkennen?
- Herausforderung
  - Unterschiedliche Handpositionen, Beleuchtungen, Kamerawinkel, usw.
  - Datenmenge & Datenqualität müssen ausreichend sein.

Zielsetzung & Projektüberblick

- Ziel:
  - Ein System entwickeln, das anhand von Live-Kameradaten (Webcam) einzelne Buchstaben (A–Y) erkennen und klassifizieren kann.
  - Speicherung der erkannten Buchstaben in einer „Textzeile“ und Möglichkeit zum Löschen/Eingabe bestätigen.

Zielsetzung & Projektüberblick

- Vorgehensweise:
  - Datenerfassung (mit collect_imgs.py)
  - Vorverarbeitung/Feature-Extraktion (mit create_dataset.py)
  - Training & Modellierung (mit train_classifier.py)
  - Echtzeit-Inferenz (mit main.py)
- Technologien
  - Python (3.9)
  - OpenCV
  - Mediapipe (Hand Landmark Detection)
  - Scikit-learn (z.B. RandomForestClassifier)

Zielsetzung & Projektüberblick

Datenakquise und -aufbereitung

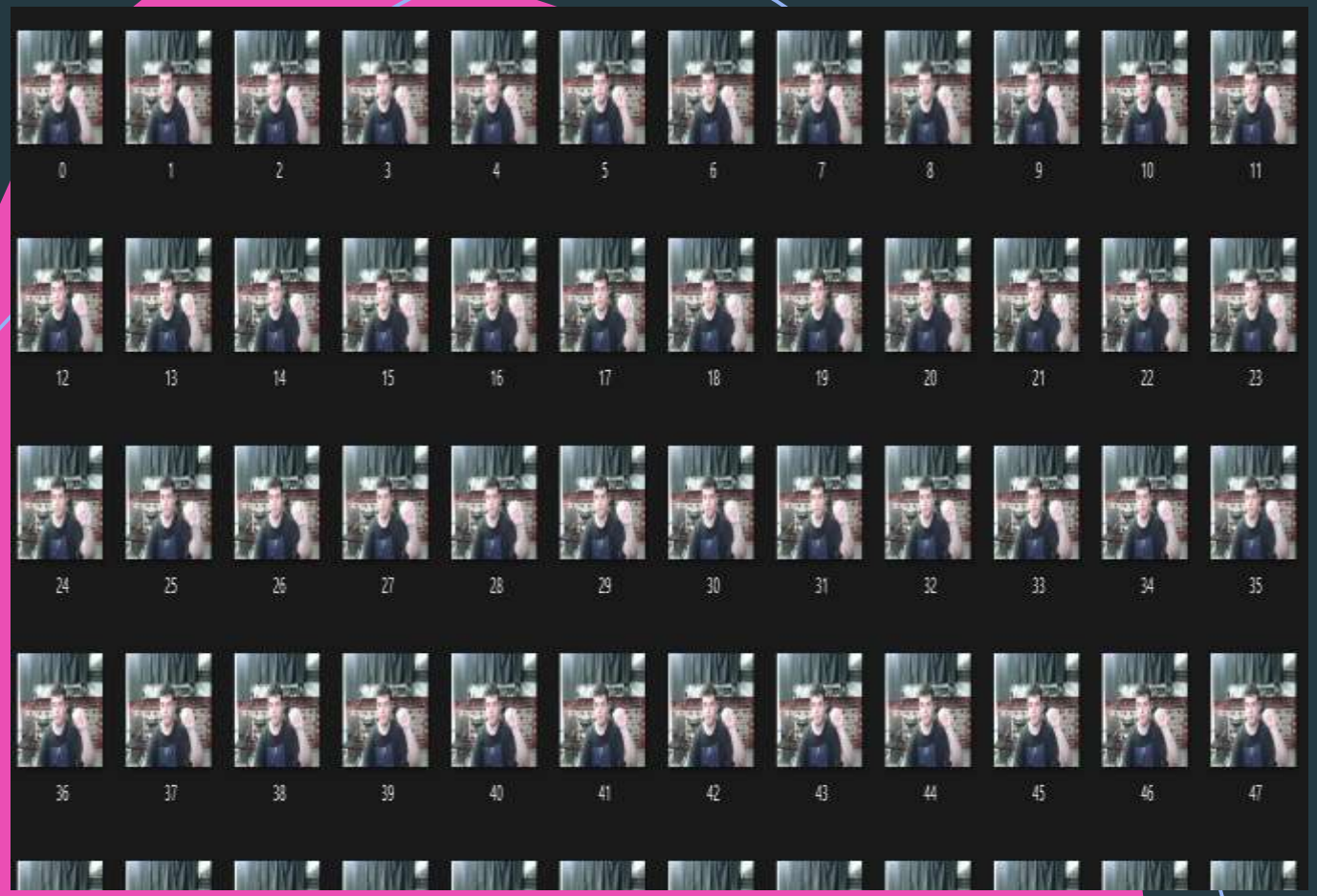
- Datenstruktur
  - Ordner ./data mit Unterordnern 0, 1, 2, …, 23
  - Jeder Ordner = ein Buchstabe (z.B. 0 = A, 1 = B, …, 23 = Y)
  - Jeweils ~99 Bilder (JPG) pro Klasse
- Skript: collect_imgs.py
  - Öffnet die Webcam (cv2.VideoCapture)
  - Für jeden Klassen-Index (z.B. 0 bis 23) werden per Tastendruck Bilder aufgenommen.
  - Speicherung unter ./data/<Klasse>/<index>.jpg
- Vorteile:
- Schnelles Sammeln neuer Daten (z.B. verschiedene Hände, Hintergründe, etc.).
- Erweiterbar, wenn neue Zeichen dazukommen sollen.

Feature-Extraktion mit Mediapipe

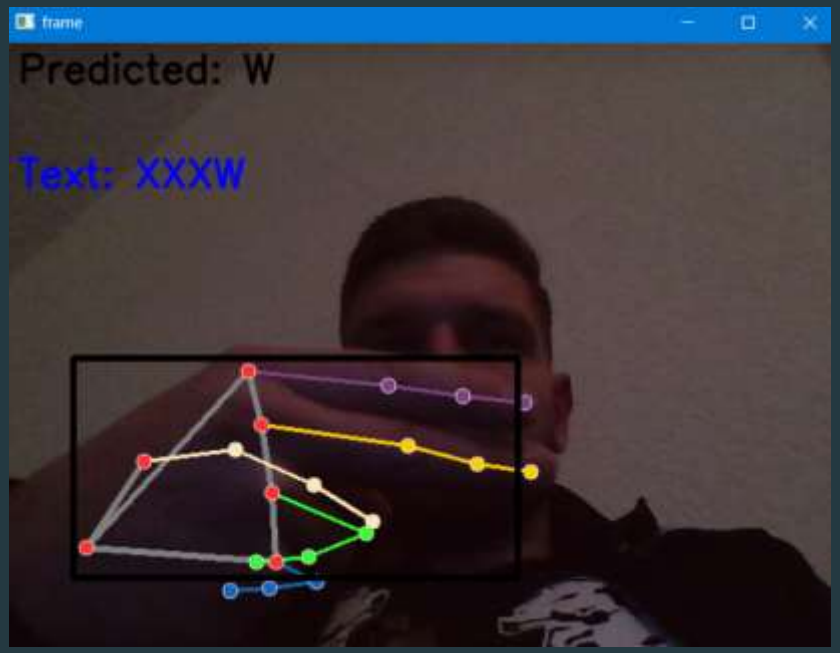
- Skript: create_dataset.py
  - Nutzt Mediapipe’s Hands-Modul, um Handlandmarken zu erkennen (21 Punkte pro Hand).
  - Berechnet relative Positionen der Landmarken (Subtraktion von min(x_) und min(y_)).
  - Erstellt daraus einen Vektor (z.B. 42 Features, da 21 Punkte * 2 Koordinaten), um die Handgeste zu beschreiben.
  - Speichert Features + Label in data.pickle für das spätere Training.
- Wichtige Punkte:
  - Normalisierung der Koordinaten → robust gegenüber Positionsverschiebungen im Bild.
  - Optional: Erweiterungen möglich (z.B. Rotation, Skalierung).

Trainingsprozess & Modell

- Skript: train_classifier.py
  - Lädt data.pickle: Enthält die Feature-Vektoren und zugehörige Labels (0–23).
  - Trennt Daten in Training (80%) und Test (20%).
  - Modell: RandomForestClassifier (scikit-learn)
    - Hyperparameter (z.B. Anzahl Bäume) ggf. anpassbar.
  - Trainiert Modell → Ausgabe: Genauigkeit in %.
  - Speicherung des finalen Modells als model.p.



Trainingsprozess & Modell



Skriptarchitektur

- collect_imgs.py
  - Daten aufnehmen mit Webcam
  - Abspeichern in ./data/...
- create_dataset.py
  - Mediapipe-Landmark-Erkennung (Offline)Features + Labels pickeln → data.pickletrain_
- classifier.py
  - Lesen von data.pickle, Training & Speichern → model.
- pmain.py
  - Lädt model.p, Mediapipe-Landmark-Erkennung in Echtzeit (Webcam)
  - Zeichnet Bounding Box, erkennt Buchstabe, zeigt ihn an, baut „Wort“ auf

main.py: Echtzeit-Erkennung

- Hauptidee
  - Öffnet Webcam (cv2.VideoCapture(0))
  - Mediapipe verarbeitet jeden Frame → Landmarken → Feature-Vektor (42 Längen).
  - model.predict(...) → Buchstabe (Index 0–23 → A–Y).
  - Implementierte Logik
    - Gleicher Buchstabe muss ein Zeitfenster (z.B. 2 Sekunden) stabil erkannt werden, um als „akzeptiert“ zu gelten.
    - Gespeicherte Buchstaben werden als „Wort“ angezeigt (entweder OSD im Frame oder Konsole).
    - Backspace/Enter, um Zeichen zu löschen/Zeile zurückzusetzen.
- Anzeigefunktion
  - cv2.putText(frame, 'Predicted: ...', ...)
  - Markieren der Hand mit mp_drawing.draw_landmarks(...)
  - Rechteck um Hand → visuelle Rückmeldung

Demo

Ergebnisse

- Genauigkeit (Train/Test)
  - Beispiel: 92–98 % in Testdaten
- Live-Erkennungsquote
  - Abhängig von Beleuchtung, Abstand zur Kamera, Hintergrund.
- Stabilität
  - Kurze Latenz dank Mediapipe, aber noch Verbesserungen möglich (z.B. GPU oder Tuning).

Probleme & Limitierungen

- Begrenzte Anzahl Buchstaben (keine Umlaute, kein J, etc.)
  - Im Code wird z.B. das Label „9“ übersprungen, was einem anderen Buchstaben entspricht („J“ nicht enthalten).
- Empfindlich auf Bildrauschen
  - Schlechte Beleuchtung → Landmarken ggf. ungenau.
- Keine Gestenerkennung mit Bewegung
  - Nur statisches „Frame für Frame“-Prinzip.

Ausblick

- Erweiterung um mehr Zeichen
  - Inklusive Umlaute, Ziffern oder andere Gebärden (z.B. ganze Wörter).
- Einsatz eines tieferen Neuronalen Netzes
  - Z.B. TensorFlow/Keras-Modell statt RandomForest → Bessere Generalisierung.
- Optimierung
  - Hyperparameter-Tuning
  - Datenaugmentierung (verschiedene Hintergründe, verschiedene Personen etc.).
- Einsatz auf Embedded-Systemen
  - Z.B. Jetson Nano oder Raspberry Pi mit Intel NCS → mobile oder Edge-Lösung.

Zusammenfassung

- Projektziele erreicht:
  - Erkennung der Buchstaben A–Y in Echtzeit.
  - Demonstration über Webcam mit Mediapipe und Machine Learning.
- Technische Eckpunkte:
  - ~2–3k Bilder (99 pro Klasse).
  - RandomForest, ~ 90–95 % Genauigkeit.
  - Gute Ausgangsbasis für Erweiterungen.
- Lernpotenzial:
  - Umgang mit Computer Vision, Python, Mediapipe, scikit-learn.
  - Projektstruktur, Skriptlogik für reproducible ML-Pipeline.

Vielen Dank!

Noch Fragen?