Automatische Klassifikation von Thorax-Röntgenbildern zur Erkennung von Pneumonie und Tuberkulose

Projektbericht

Gruppe:
Felix Zauner, Timofey Luzin

Klasse & Schuljahr:
5AHETS 2025/26

Abgabe:
15.12.2025

Inhaltsverzeichnis

1 Einleitung

In diesem Projektbericht wird die Entwicklung eines Modells zur automatischen Klassifikation von Thorax-Röntgenbildern zur Erkennung von Pneumonie und Tuberkulose beschrieben.
Die KI-basierte Bildanalyse hat in den letzten Jahren erhebliche Fortschritte gemacht und bietet vielversprechende Möglichkeiten zur Unterstützung medizinischer Diagnosen. Manche Experten sehen in der automatischen Bildanalyse sogar das Potenzial, die Genauigkeit und Effizienz von Diagnosen zu verbessern, insbesondere in ressourcenarmen Umgebungen.
Ziel dieses Projekts ist es, ein Modell zu entwickeln, das in der Lage ist, Thorax-Röntgenbilder zu analysieren und zwischen gesunden Patienten, Patienten mit Pneumonie und Patienten mit Tuberkulose zu unterscheiden.
Der Bericht gliedert sich in mehrere Abschnitte, die den gesamten Entwicklungsprozess abdecken, von der Datensammlung und -aufbereitung über die Modellauswahl und das Training bis hin zur Evaluation des Modells. Abschließend werden die Herausforderungen und Probleme, die während des Projekts aufgetreten sind, sowie eine Reflexion über die Ergebnisse und mögliche zukünftige Verbesserungen diskutiert.

2 Datensatzbeschreibung

Der Datensatz stammt beinhaltet insgesamt 25.600 Thorax-Röntgenbilder, die in drei Klassen unterteilt sind: Gesunde Patienten, Patienten mit Pneumonie und Patienten mit Tuberkulose. Die Bilder sind schon in Trainings-, Validierungs- und Testsets aufgeteilt. In unserer Anwendung verwenden wir *ausschließlich* das *Trainingsset* und das *Testset*.
Die Bilder liegen im JPEG-Format vor und haben keine standartisierte Auflösung. Hier ist eine Übersicht über die Struktur des Datensatzes dargestellt:

```
chest-xray-dataset
└── versions
    └── 1
        ├── test
        │   ├── normal
        │   │   └── [925 Abbildungen]
        │   ├── pneumonia
        │   │   └── [580 Abbildungen]
        │   └── tuberculosis
        │       └── [1064 Abbildungen]
        ├── train
        │   ├── normal
        │   │   └── [7263 Abbildungen]
        │   ├── pneumonia
        │   │   └── [4674 Abbildungen]
        │   └── tuberculosis
        │       └── [8513 Abbildungen]
        ├── val
        │   ├── normal
        │   │   └── [900 Abbildungen]
        │   ├── pneumonia
        │   │   └── [570 Abbildungen]
        │   └── tuberculosis
        │       └── [1064 Abbildungen]
        └── data.yaml
```

Abbildung 1: Struktur des Datensatzes

Das Datenset wurde von Kaggle bereitgestellt und ist öffentlich zugänglich unter folgendem Link:
https://www.kaggle.com/datasets/muhammadrehan00/chest-xray-dataset

3 Datenaufbereitung

Aufgrund der unterschiedlichen Auflösungen der Bilder im Datensatz war eine sorgfältige Datenaufbereitung erforderlich, um eine konsistente Eingabe für das Modell zu gewährleisten.
Die Bilder wurden sowohl auf eine einheitliche Größe von 128x128 Pixel skaliert, als auch in schwarz-weiß umgewandelt, um die Verarbeitung zu erleichtern und die Leistung des Modells zu optimieren.

```python
import cv2
# Transform and load images
def load_images(folder, label, img_size=(128, 128)):
    X, y = [], []

    for file in os.listdir(folder):
        path = os.path.join(folder, file)
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, img_size)
        X.append(img)
        y.append(label)

    return X, y
```

Listing 1: Datenaufbereitungscode

4 Modellauswahl und -training

4.1 Modellauswahl

Es wurden mehrere Modelle ausprobiert, darunter:

- MLP Classifier:
  Einfach zu implementieren und schnell zu trainieren, geeignet für erste Tests.

- MLP Classifier mit mehr Schichten:
  Der Code definiert und trainiert einen optimierten MLP-Klassifikator mit zwei versteckten Schichten und erhöhter Modellkapazität. Durch angepasste Trainingsparameter wird eine stabilere und leistungsfähigere Klassifikation auf den skalierten Trainingsdaten erreicht.

- SVC ohne Feature Extraction
  Der Code initialisiert einen Support Vector Classifier mit RBF-Kernel und balancierten Klassengewichten. Das Modell wird anschließend mit den skalierten Trainingsdaten trainiert, um eine nichtlineare Klassifikationsgrenze zu erlernen.

- SVC mit PCA Feature Extraction:
  Der Code extrahiert HOG-Features aus den Bildern, um relevante Merkmale für die Klassifikation zu gewinnen. Diese Features werden anschließend für das Training eines Support Vector Classifiers verwendet, um die Leistung des Modells zu verbessern.

4.2 Modelltraining

Das Training der Modelle erfolgte mit dem vorbereiteten Datensatz. Verschiedene Hyperparameter wurden getestet, um die Leistung zu optimieren.

- MLP Classifier:
  Der Code initialisiert und trainiert einen mehrschichtigen Perzeptron-Klassifikator (MLP) mit einem versteckten Layer. Dazu werden die Bilddaten zunächst abgeflacht, standardisiert und anschließend zum Trainieren des Modells verwendet.

```
# Train MLP Classifier
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=50,
    alpha=1e-4, solver='adam', verbose=True, random_state=912)

# Flatten images to (n_samples, n_features) because
    MLPClassifier expects 2D input
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_flat)
X_test_scaled = scaler.transform(X_test_flat)

# Fit model
mlp.fit(X_train_scaled, y_train)
```

Listing 2: MLP Classifier

- MLP Classifier mit mehr Schichten:
  Der Code definiert und trainiert einen optimierten MLP-Klassifikator mit zwei versteckten Schichten und erhöhter Modellkapazität. Durch angepasste Trainingsparameter wird eine stabilere und leistungsfähigere Klassifikation auf den skalierten Trainingsdaten erreicht.

```
# Optimized MLP with early stopping, higher max_iter and tuned
    learning rate/batch size
mlp2 = MLPClassifier(hidden_layer_sizes=(100,100),
    max_iter=100, alpha=1e-4, solver='adam', verbose=True,
    random_state=912)

# Fit model
mlp2.fit(X_train_scaled, y_train)
```

Listing 3: MLP Classifier mit mehr Schichten

- SVC ohne Feature Extraction
  Der Code initialisiert einen Support Vector Classifier mit RBF-Kernel und balancierten Klassengewichten. Das Modell wird anschließend mit den skalierten Trainingsdaten trainiert, um eine nichtlineare Klassifikationsgrenze zu erlernen.

```
from sklearn.svm import SVC

svc = SVC(
    kernel="rbf",
    C=10,
    gamma="scale",
    class_weight="balanced"
)

svc.fit(X_train_scaled, y_train)
```

Listing 4: SVC ohne Feature Extraction

- SVC mit PCA Feature Extraction:
  Der Code extrahiert HOG-Features aus den Bildern, um relevante Merkmale für die Klassifikation zu gewinnen. Diese Features werden anschließend für das Training eines Support Vector Classifiers verwendet, um die Leistung des Modells zu verbessern.

```
from skimage.feature import hog

def extract_hog(images):
    features = []
    for img in images:
        hog_feat = hog(
            img,
            orientations=9,
            pixels_per_cell=(8, 8),
            cells_per_block=(2, 2),
            block_norm='L2-Hys'
        )
        features.append(hog_feat)
    return np.array(features)
```

Listing 5: SVC mit PCA Feature Extraction

4.3 Code-Dokumentation

Hier sind die während der Vorvearbeitung und dem Training verwendeten Code-Snippets dokumentiert:

```
# Libraries importieren
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,
                            confusion_matrix
from sklearn.preprocessing import StandardScaler
import kagglehub
```

Listing 6: Hauptlibraries

```
# Download latest version of dataset from Kaggle
path = kagglehub.dataset_download("muhammadrehan00
                            /chest-xray-dataset")

print("Path to dataset files:", path)

# Print the paths to the train and test directories
train_dir = os.path.join(path, "chest_xray", "train")
test_dir = os.path.join(path, "chest_xray", "test")
print("Train directory:", train_dir)
print("Test directory:", test_dir)
```

Listing 7: Herunterladen des Datensatzes und Pfadzuweisung

```python
# Prepare training data
X_train, y_train = [], []

# Train
normal_X, normal_y = load_images(os.path.join(path, "train/NORMAL"), "Normal")
pneu_X, pneu_y     = load_images(os.path.join(path, "train/PNEUMONIA"), "Pneumonia")
tub_X, tub_y       = load_images(os.path.join(path, "train/TUBERCULOSIS"), "Tuberculosis")

X_train.extend(normal_X + pneu_X + tub_X)
y_train.extend(normal_y + pneu_y + tub_y)

X_train = np.array(X_train)
y_train = np.array(y_train)

# Prepare testing data
X_test, y_test = [], []

# Test
normal_X, normal_y = load_images(os.path.join(path, "test/NORMAL"), "Normal")
pneu_X, pneu_y     = load_images(os.path.join(path, "test/PNEUMONIA"), "Pneumonia")
tub_X, tub_y       = load_images(os.path.join(path, "test/TUBERCULOSIS"), "Tuberculosis")

X_test.extend(normal_X + pneu_X + tub_X)
y_test.extend(normal_y + pneu_y + tub_y)

X_test = np.array(X_test)
y_test = np.array(y_test)
```

Listing 8: Einteilung der Bilder in Arrays und Labels zur weiteren Verarbeitung durch das entwickelte Modell

4.4 Probleme und Herausforderungen

Die Ergebnisse waren in unserem Fall unzureichend, mit einer Genauigkeit von nur etwa 0,7. Dies könnte auf verschiedene Faktoren zurückzuführen sein, darunter die Komplexität der Bilder, die begrenzte Größe des Datensatzes oder die Wahl der Modelle und Hyperparameter.
Es wäre sinnvoll, in unserem Fall ein CNN (Convolutional Neural Network) zu verwenden, da diese Modelle speziell für die Verarbeitung von Bilddaten entwickelt wurden und in der Regel bessere Ergebnisse bei der Bildklassifikation erzielen.

5 Evaluation des Modells

Die Evaluation der Modelle erfolgte anhand verschiedener Metriken, darunter Genauigkeit, Präzision, Recall und F1-Score. Zusätzlich wurden Konfusionsmatrizen erstellt, um die Leistung der Modelle visuell darzustellen.

```python
# Vorhersagen auf dem Testset machen
y_pred = mlp.predict(X_test_scaled)

# Ergebnisse auswerten
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Listing 9: Evaluationscode MLP Classifier

Konfusionsmatrix MLP Classifier: $\begin{bmatrix} 680 & 139 & 106 \\ 66 & 507 & 7 \\ 366 & 3 & 695 \end{bmatrix}$

	Precision	Recall	F1-score	Support
Normal	0.61	0.74	0.67	925
Pneumonia	0.78	0.87	0.83	580
Tuberculosis	0.86	0.65	0.74	1064
Accuracy			0.72	2569
Macro avg	0.75	0.75	0.74	2569
Weighted avg	0.75	0.72	0.72	2569

6 Probleme und Herausforderungen im gesamten Projekt

6.1 Technische Herausforderungen

6.2 Modellierungsprobleme

7 Schlussfolgerung und Reflexion

8 Quellen und Literaturverzeichnis

9 Anhang