Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Elektro- und Informationstechnik
Faculty of Electrical Engineering and Information Technology

Bildbasierte Künstliche Intelligenz

Vergleich von Methoden zur Klassifikation von maskierten und unmaskierten Gesichtern mit CNNs

Angefertigt an der Hochschule Düsseldorf

Studiengang: Elektro- und Informationstechnik
Fachrichtung: Informationstechnik

Vorgelegt von: Asena Esra Güler 949870

Betreuung: M.Sc. Simon Geerkens
M.Sc. Christian Sieberichs

Eingereicht am: 28. Juni 2025

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Bildbasierte Künstliche Intelligenz (KI) hat in den letzten Jahren enorme Fortschritte gemacht und findet heute breite Anwendung in Bereichen wie autonomem Fahren (z. B. das Projekt safe.trAIn), medizinischer Bildanalyse oder Sicherheitsüberwachung. Besonders im Bereich der Gesichtserkennung ermöglichen Convolutional Neural Networks (CNNs) eine präzise Analyse und Klassifikation komplexer Bildinformationen. Diese Netzwerke sind darauf ausgelegt, visuelle Merkmale in mehreren Hierarchiestufen zu extrahieren und auch in anspruchsvollen Szenarien robuste Vorhersagen zu treffen.

Die COVID-19-Pandemie hat eindrücklich gezeigt, wie wichtig das Tragen von Gesichtsmasken im öffentlichen Raum sein kann, um die Verbreitung von Infektionskrankheiten einzudämmen. Dabei geht es nicht nur ausschließlich um COVID-19, sondern grundsätzlich um den Schutz vor verschiedenen übertragbaren Atemwegserkrankungen. Systeme zur automatisierten Maskenerkennung können helfen, Präventionsmaßnahmen effizient umzusetzen, Zugangskontrollen zu unterstützen oder statistische Erhebungen zur Maskenquote durchzuführen. Sie stellen somit einen wichtigen Baustein in umfassenden Gesundheits- und Sicherheitskonzepten dar [[1]].

Für dieses Projekt wird ein öffentlich verfügbares Maskenerkennungs-Datenset verwendet. Ziel ist es, eine zuverlässige Klassifikation durchzuführen, ob eine auf einem Foto abgebildete Person eine Gesichtsmaske trägt oder nicht. Dabei wird die Aufgabe als explizites binäres Klassifikationsproblem formuliert, bei dem zwischen *with_mask* (Label 1) und *without_mask* (Label 0) unterschieden wird. Eine robuste Lösung muss in der Lage sein, verschiedene Herausforderungen wie Lichtverhältnisse, Gesichtsperspektiven und Bildqualitäten zu berücksichtigen und konsistente Ergebnisse zu liefern.

Im Rahmen dieses Projekts werden zwei unterschiedliche Ansätze vergleichen: Ein selbst implementiertes einfaches CNN als Baseline und ein einige vortrainiertes ResNet50-Modell mit Transfer Learning, das für die Maskenerkennungsaufgabe angepasst wird. Beide Modelle werden auf einer gezielt ausgewählten Teilmenge des Datensatzes trainiert und validiert, um eine faire Vergleichbarkeit zu gewährleisten. Wichtige Kennzahlen wie Accuracy, Precision, Recall, F1-Score und Loss werden analysiert und gegenübergestellt, um das Lernverhalten und die Generalisierungsfähigkeit der Modelle besser zu verstehen.

1.1 Aufgabestellung

Die zentrale Aufgabe dieses Projekts besteht darin, ein System zur automatisierten bildbasierten Klassifikation zu entwickeln, das zuverlässig erkennt, ob eine auf einem Foto dargestellte Person eine Gesichtsmaske trägt oder nicht. Es handelt sich dabei explizit um ein binäres Klassifikationsproblem, bei dem zwischen den beiden Klassen *with_mask* (Label 1) und *without_mask* (Label 0) unterschieden wird. Ziel ist es, ein Modell zu entwerfen, das trotz Varianz in Lichtverhältnissen, Gesichtsperspektiven und Bildqualität robuste Vorhersagen liefern kann und für den Einsatz in realen Szenarien geeignet ist.

Für die Lösung dieser Aufgabe werden Convolutional Neural Networks (CNNs) eingesetzt, die speziell für Bildverarbeitung konzipiert sind und Merkmale über mehrere Faltungsschichten hierarchisch lernen [[2]]. Sie kombinieren Faltungs-, Pooling- und Aktivierungsschichten, um komplexe visuelle Strukturen effektiv zu erkennen und für die Klassifikation nutzbar zu machen. Das Projekt vergleicht dabei ein einfaches selbstprogrammiertes CNN als Baseline mit einem vortrainierten ResNet50-Modell, das mittels Transfer Learning für die Maskenerkennung angepasst wird.

1.2 Motivation

Die Motivation dieses Projekts liegt sowohl im gesellschaftlichen Nutzen solcher Systeme als auch in meinem Wunsch, bildbasierte KI-Methoden praxisnah zu erlernen. Gerade während der Pandemie wurde mir bewusst, wie wichtig digitale Lösungen zur Prävention sein können. Die Idee, mit Hilfe von KI-Technologien automatisiert zu überprüfen, ob Menschen Masken tragen, finde ich nicht nur spannend, sondern auch gesellschaftlich wertvoll und langfristig relevant über die COVID-19-Pandemie hinaus.

Da ich zu Beginn des Projekts keine tiefere Erfahrung mit bildbasierter KI hatte, war mein Ziel, ein funktionierendes System mit einem balancierten Subset des Datensatzes umzusetzen. Ich wollte verstehen, wie man die Daten aufbereitet, Transformationen anwendet, Modelle trainiert und die relevanten Metriken wie Accuracy, Precision und Loss berechnet. Dabei habe ich mich darauf konzentriert, eine saubere, nachvollziehbare Implementierung abzuliefern, die als Grundlage für weiterführende Verbesserungen dienen kann. Ich sehe das Projekt daher als Einstieg in ein komplexes Themenfeld, das in Zukunft vielfältig erweitert und optimiert werden kann.

2 Datensatz

Die Auswahl eines passenden Datensatzes bildet die Grundlage für eine erfolgreiche Modellierung. In diesem Projekt wird das *Real-World Masked Face Dataset (RMFD)* verwendet, das zur Unterstützung von Forschung und Entwicklung im Bereich der automatisierten Maskenerkennung erstellt wurde. Dieses Kapitel beschreibt zunächst die Struktur und Eigenschaften des Datensatzes. Anschließend werden die spezifischen Herausforderungen und Anpassungen erläutert, die im Rahmen der Datenaufbereitung und Integration ins Projekt notwendig waren.

2.1 Real-World Masked Face Dataset

Das Real-World Masked Face Dataset (RMFD) enthält eine große Anzahl an realen Bildern von Personen mit und ohne Gesichtsmaske und wurde mit dem Ziel zusammengestellt, robuste Trainings- und Testdaten für Klassifikationsaufgaben bereitzustellen.

Laut offizieller Dokumentation umfasst das Dataset nach dem Labeling ca. 5.000 maskierte Gesichter von 525 verschiedenen Personen und etwa 90.000 unmaskierte Gesichter. Damit bietet es eine hohe Varianz an Gesichtern, Posen, Beleuchtungen und Maskentypen. Diese Diversität ermöglicht eine realitätsnahe Abbildung, stellt jedoch auch eine Herausforderung dar, da Inkonsistenzen in Qualität und Klassenzuordnung auftreten können.

Im Rahmen dieses Projekts wurde das RMFD lokal in einem Ordner `data/RWMFD` integriert, der ursprünglich über 92.671 Bilddateien in zwei Hauptordnern enthält.[[1]].Die klare Zweiklassenstruktur ist wie folgt definiert: **with_mask**: Bilder von Gesichtern mit Maske und **without_mask**: Bilder von Gesichtern ohne Maske.

Abbildung 1 zeigt die verwendete Ordnerstruktur im Projekt.

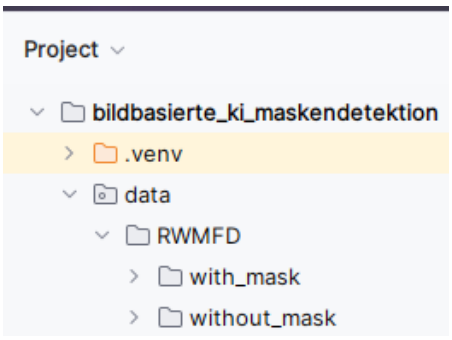


Figure 1: Projektinterne Ordnerstruktur

2.2 Datenaufbereitung und Integration ins Projekt

Das RMFD enthält eine sehr große Menge an Bildern pro Klasse, wobei es innerhalb der Klassen zu Imbalancen und hoher Varianz in den Bildqualitäten kommt. Um eine faire Klassifikation und ein ausgeglichenes Training zu ermöglichen, wurde das Laden und die Selektion der Daten über ein angepasstes Skript `data_loader.py` gesteuert.

2.2.1 Laden und Vorverarbeitung der Bilddaten

Um die Bilddaten in das Maschine Lernen Modell zu integrieren, wurde zunächst eine Funktion **get_data_loader_RWMFD** entwickelt, die Bilder aus dem lokalen Verzeichnis lädt und dabei grundlegende Transformationen vornimmt. Die Bilder werden einheitlich auf 224x224 Pixel skaliert und in Tensoren konvertiert, um sie in PyTorch verarbeiten zu können, einer flexiblen Deep-Learning-Bibliothek, die speziell für den einfachen Aufbau und das Training neuronaler Netze entwickelt wurde.[[3]]

Listing 1: Datenladefunktion

```
def get_data_loader_RWMFD(data_dir, batch_size=64, val_ratio
   =0.2):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ])
    dataset = datasets.ImageFolder(root=data_dir, transform=
       transform)
    class_dict = dataset.class_to_idx
    total_size = len(dataset)
    indices = list(range(total_size))
    train_indices, val_indices = train_test_split(
        indices,
        test_size=val_ratio,
        stratify=[dataset.targets[i] for i in indices]
    )
    train_dataset = Subset(dataset, train_indices)
    val_dataset = Subset(dataset, val_indices)
    train_loader = DataLoader(train_dataset, batch_size=
       batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=
       batch_size, shuffle=False)
    return train_loader, val_loader, class_dict
```

Diese Funktion liest die Bilder aus dem angegebenen Verzeichnis mithilfe von `ImageFolder`, skaliert sie auf ein einheitliches Format und konvertiert sie in Tensoren, die für die Verarbeitung in PyTorch geeignet sind. Anschließend wird der gesamte Datensatz in 80% Trainings- und 20% Validierungsdaten aufgeteilt, wobei ein stratified Split sicherstellt, dass das Klassenverhältnis erhalten bleibt. Schließlich werden die entsprechenden `DataLoader`-Objekte für Training und Validierung sowie ein Wörterbuch `class_dict` zurückgegeben, das die Klassenzuordnungen enthält.

Beim Laden des Datensatzes über `ImageFolder` werden die Labels alphabetisch zugewiesen. Dies führte dazu, dass der Ordner `with_mask` fälschlicherweise `Label 0` erhielt und `without_mask` ebenfalls `Label 0`. Für das Training war jedoch eine konsistente Zuordnung erforderlich, bei der Maskenbilder das Label 1 haben sollten. Dieses Problem wurde gelöst, indem im Testblock eine manuelle Umkehrung des Label-Mappings implementiert wurde. Dadurch wird sichergestellt, dass `with_mask` stets Label 1 und `without_mask` Label 0 erhält.[[4, 5]]

Listing 2: Label-Mapping-Korrektur für RWMFD

```
if __name__ == "__main__":
    train_loader, val_loader, class_dict =
       get_balanced_subset_loader_RWMFD(r"...",
        batch_size=64
    )
    # Label-Zuweisung: with_mask  1, without_mask  0
       umwandeln
    true_class_dict = {k: 1 - v for k, v in class_dict.items
       ()}
    print("Korrigierter class_dict:", true_class_dict)
    for images, labels in val_loader:
        labels = 1 - labels
        print("Label Batch (after flip):", labels)
        break
```

Durch diese Korrektur konnte das Modell konsistent mit den richtigen Labels trainiert werden, was eine entscheidende Voraussetzung für eine sinnvolle Klassifikation darstellt.[[6]]

2.2.2 Flatten-Struktur und Subset-Auswahl

Das RMFD liegt ursprünglich in einer verschachtelten Ordnerstruktur vor, in der für jede Person eigene Unterordner existieren. Diese Struktur erschwert eine direkte Nutzung mit `ImageFolder`, das eine klare Klassenzuweisung auf oberster Ebene erwartet. Um dies zu lösen, wurde ein Skript entwickelt, das alle Bilder in flache Verzeichnisse `with_mask_flat` und `without_mask_flat` überführt: [[7, 8]]

Listing 3: Flatten-Dataset-Skript

```
import shutil
import os
def flatten_image_directory(root_dir):
    count = 0
    for class_name in ["with_mask", "without_mask"]:
        class_path = os.path.join(root_dir, class_name)
        new_dir = os.path.join(root_dir, f"{class_name}_flat"
           )
        os.makedirs(new_dir, exist_ok=True)
        for person_folder in os.listdir(class_path):
            person_path = os.path.join(class_path,
               person_folder)
            if os.path.isdir(person_path):
                for filename in os.listdir(person_path):
                    if filename.lower().endswith((".jpg", ".
                       jpeg", ".png")):
                        src = os.path.join(person_path,
                           filename)
                        new_name = f"{person_folder}_{
                           filename}"
                        dst = os.path.join(new_dir, new_name)
                        shutil.copy2(src, dst)
                        count += 1
    print(f"Insgesamt wurden {count} Bilder verschoben und
       umbenannt.")
flatten_image_directory(r"...")
```

Diese Vereinfachung der Verzeichnisstruktur erleichtert Label-Zuweisung und nachdem innere Folder entfernt `flat` Ergänzen gelöscht werden.

Da das RMFD über 90.000 Bilder enthält, wäre ein Training auf dem gesamten Datensatz ressourcenintensiv gewesen. Um Speicher und Rechenzeit zu sparen und eine faire Klassifikation zu ermöglichen, wurde eine Funktion `get_balanced_subset_loader_RWMFD` entwickelt. Sie wählt gezielt eine kleinere, ausgewogene Subset für 1280 pro Klasse aus:

Listing 4: get_balanced_subset_loader_RWMFD Funktion

```
def get_balanced_subset_loader_RWMFD(data_dir, batch_size=64,
    subset_size_per_class=1280, val_ratio=0.2):
    ...
    targets = [sample[1] for sample in dataset.samples]
    class_indices = {cls: [] for cls in set(targets)}
    for idx, label in enumerate(targets):
        if len(class_indices[label]) < subset_size_per_class:
            class_indices[label].append(idx)
    selected_indices = class_indices[0] + class_indices[1]
    subset = Subset(dataset, selected_indices)
    train_indices, val_indices = train_test_split(
        list(range(len(subset))),
        test_size=val_ratio,
        stratify=[targets[i] for i in selected_indices]
    )
    ...
    return train_loader, val_loader, class_to_idx
```

Diese Funktion stellt sicher, dass Trainings- und Validierungsdaten ein ausgeglichenes Verhältnis zwischen `with_mask` und `without_mask` enthalten. Dadurch wird verhindert eine Überlastung der CPU auf Standardcomputern ermöglicht.

Abbildung 2 zeigt ein typisches Ausgabe des DataLoaders nach der Subset-Selektion und Label-Korrektur:[[5]]

```
Train batch size: 64
Validation batch size: 64
Train dataset size: 2048
Validation dataset size: 512
Korrigierter class_dict: {'with_mask': 1, 'without_mask': 0}
Label Batch (after flip): tensor([1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
        0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
        1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1])
```

Figure 2: Konsolenausgabe des DataLoaders

3 Netzwerkarchitektur

Neuronale Netzwerke sind Modelle, die aus Schichten von Neuronen bestehen und Eingaben über gewichtete Verbindungen in Ausgaben transformieren. Ziel ist es, komplexe Abbildungen zu lernen und Muster in den Daten zu erkennen. In diesem Projekt werden solche neuronalen Netzwerke für die Bildverarbeitung in zwei unterschiedlichen Modellen eingesetzt: dem selbst erstellten SimpleCNN und dem ResNet50-Modell, das mithilfe von Transfer Learning in seiner letzten Schicht zu einem binären Klassifikator angepasst wird.

Zunächst werden die grundlegenden Prinzipien von Convolutional Neural Networks (CNN) und das im Projekt implementierte SimpleCNN-Modell erläutert. Anschließend wird ein Überblick über die Nutzung des ResNet-50-Modells mit Transfer Learning und dessen Integration ins Projekt gegeben. Darüber hinaus ist der Code-Aufbau in den Dateien für Training, Evaluation, Sample und Plot für beide Modelle identisch; lediglich das importierte Modell und die spezifisch erzeugten Ausgaben unterscheiden sich.

3.1 Convolutional Neural Networks

CNNs sind eine spezielle Form neuronaler Netze und gelten als besonders leistungsfähig für Bildklassifikation. Im Gegensatz zu Fully Connected Layers, die alle Eingaben global verknüpfen, verwenden Convolutional Layers lokale Filter (Kernels), um die räumliche Struktur von Bildern auszunutzen. Sie erkennen Merkmale wie Kanten oder Texturen durch das Verschieben eines kleinen Filters (Sliding Window) über das Bild. Dies verringert die Parameterzahl deutlich und verbessert die Generalisierung bei Datensätzen wie dem RWMFD.

3.1.1 Grundprinzipien und Formeln

Das Kernprinzip eines CNN besteht aus mehreren hintereinander geschalteten Schichten, die unterschiedliche Aufgaben übernehmen: Diese Kernkomponenten sind Teil der Modellarchitektur und werden beim Aufbau des **CNN-Modells** genutzt.

- **Convolutional Layer:** wendet Filter an, um lokale Merkmale zu erkennen. Formal kann die Faltung als

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) \cdot K(m, n)$$

  beschrieben werden, wobei I das Eingangsbild und K der Filter ist.

- **Aktivierungsfunktion:** [9] führt Nichtlinearität ein.Hier wird häufig ReLU verwendet:
$$\text{ReLU}(x) = \max(0, x)$$
- **Pooling Layer:** reduziert die räumliche Dimension und sorgt für Invarianz gegenüber kleinen Verschiebungen.
$$O(i, j) = \max_{(m,n)\in W} I(i + m, j + n)$$
beschrieben werden, wobei W das Pooling-Fenster ist.
- **Padding, Stride und Kernelgröße:** steuern die Größe und Überlappung der Ausgabe-Feature-Maps. Sie beeinflussen, wie stark Details extrahiert werden.[10]
$$H_{out} = \frac{H_{in} + 2 \cdot \text{padding} - \text{dilation} \cdot (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1$$
$$W_{out} = \frac{W_{in} + 2 \cdot \text{padding} - \text{dilation} \cdot (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1$$
- **Fully Connected Layer:** verbindet alle Neuronen der vorherigen Schicht mit den Ausgabeneuronen und ermöglicht die Klassifikation basierend auf den extrahierten Merkmalen.

Die folgenden Formeln werden während des Trainingsprozesses eingesetzt.

- **Lossfunktion:** misst die Abweichung zwischen Vorhersage und Zielwert :
$$\text{Loss} = -\sum_i y_i \log(\hat{y}_i)$$
- **Adam-Optimizer:** erweitert klassischen SGD um adaptives Lernen und Momentum. Aktualisierung basiert auf:
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_w \text{Loss}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_w \text{Loss})^2$$
$$\hat{w} \leftarrow w - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$
Diese Schritte werden in Code typischerweise mit `optimizer.zero_grad()`, `loss.backward()` und `optimizer.step()` durchgeführt.

Diese Metriken werden zur Evaluierung der Modellleistung im Training und bei der Validierung berechnet.

- **Accuracy:** misst den Anteil korrekt klassifizierter Beispiele an allen Beispielen.
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
wobei TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives sind.

- **Precision:** Anteil korrekt vorhergesagter Positiver unter allen vorhergesagten Positiven.
$$Precision = \frac{TP}{TP + FP}$$

- **Recall (Sensitivität):** Anteil korrekt vorhergesagter Positiver unter allen tatsächlich Positiven.
$$Recall = \frac{TP}{TP + FN}$$

- **F1-Score:** harmonisches Mittel von Precision und Recall.
$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

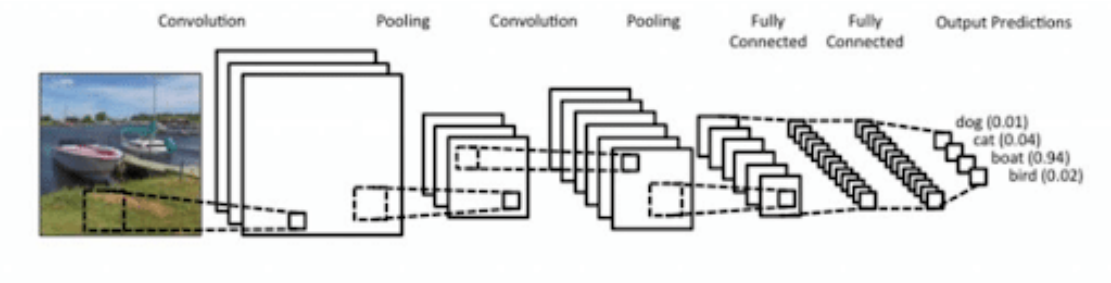


Figure 3: Architektur eines CNN

Diese Kombination ermöglicht es dem Netzwerk, komplexe Muster in Bilddaten zu erlernen und robuste Klassifikationen durchzuführen.[[11]]

3.1.2 SimpleCNN-Modell und Training

Das im Projekt entwickelte SimpleCNN-Modell wurde speziell für die binäre Klassifikation (with mask / without mask) auf Bildern ausgelegt. Es kombiniert Convolutional-Layer zur Merkmalserkennung mit Fully-Connected-Layern für die Klassifikation. Nachfolgende ist der Aufbau des Modells dargestellt:

Listing 5: SimpleCNN-Modellstruktur

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels
           =128, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=128, out_channels
           =64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 56 * 56, 128)
        self.fc2 = nn.Linear(128, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 56 * 56)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Die Wahl der Kanalzahlen folgt dabei einem klaren Prinzip: Das erste Convolutional-Layer nimmt färbliche-Bilder (3 Kanäle) als Eingang und extrahiert 128 Feature-Maps, was eine hohe Detailtiefe ermöglicht. Das zweite Layer reduziert die Kanäle auf 64, um eine kompaktere, Repräsentation zu erhalten.Padding, Stride und Kernelgröße mit die Hilfe Formel berechnen wurde und für SimpleCNN modell angewendet. Nach dem Pooling werden die Merkmalskarten in einen Vektor umgeformt (Flatten) und über zwei Fully-Connected-Schichten klassifiziert. Diese Modell ermöglicht es, Bildmerkmale zu erfassen und eine binäre Entscheidung zu treffen. [[10]]

Das Training folgte einer Standard-PyTorch-Schleife mit Forward, Loss-Berechnung, Backpropagation und Parameterupdate. Dabei wurde CrossEntropy-Loss als Verlustfunktion und Adam als Optimierer genutzt. Der Trainingsprozess lief über 10 Epochen, wobei in jeder Epoche die Metriken Accuracy, Precision, Recall und F1-Score berechnet wurden. Nachfolgend sind die durchschnittlichen Ergebnisse über alle Epochen zusammengefasst:

Table 1: Trainings-und Validierungsmetriken für SimpleCNN

	Accuracy	Precision	Recall	F1-Score
Training	0.88	0.96	0.93	0.95
Validation	0.89	0.92	0.93	0.91

Die Metriken zeigen, eine deutliche Leistungssteigerung erzielt hat und letztlich sowohl im Training als auch in der Validierung stabile, hohe Werte für Accuracy und F1-Score erreichen konnte.

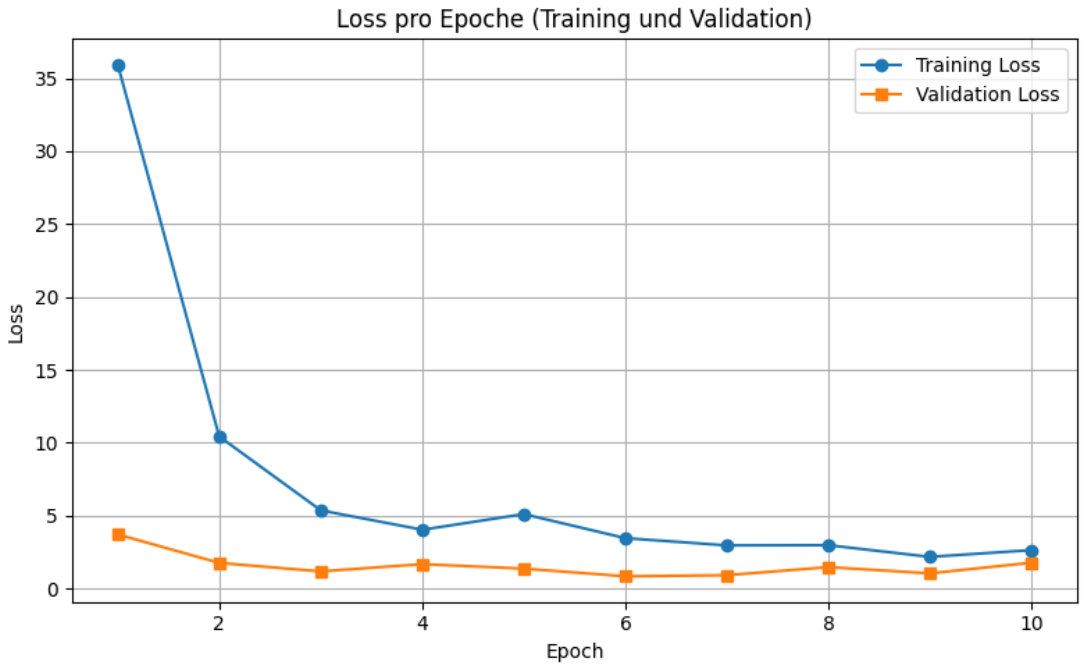


Figure 4: Loss-Werts für Training und Validierung

Abbildung 4 zeigt den Verlauf des Loss-Werts über die 10 Trainings-Epochen hinweg. Man erkennt einen deutlichen Abfall des Training-Loss in den ersten Epochen, gefolgt von einer Stabilisierung auf niedrigem Niveau. Auch der Validation-Loss sinkt deutlich und bleibt über die Epochen hinweg gleichmäßig niedrig, was auf eine erfolgreiche Generalisierung des Modells hinweist.

Die relative Nähe zwischen Training- und Validation-Loss deutet darauf hin, dass kein starkes Overfitting aufgetreten ist. Kleine Schwankungen im Validation-Loss sind üblich und reflektieren die Varianz in den Maskenbildern.

3.1.3 Validation und Ergebnisse

Die Validierung ist ein entscheidender Schritt, um die Generalisierungsfähigkeit eines Modells auf nicht gesehenen Daten zu prüfen. Dabei wird das Modell in den `eval()`-Modus versetzt, um Effekte wie Dropout oder Batch-Normalisierung zu deaktivieren und stabile Vorhersagen zu ermöglichen.

In diesem Projekt wurde ein Validierungsset mit einer Größe von 512 Bildern genutzt, um die Leistung von SimpleCNN objektiv zu bewerten. und erzielte dabei folgende durchschnittliche Ergebnisse:

Table 2: Durchschnittliche Validierungsmetriken für SimpleCNN

	Accuracy	Precision	Recall	F1-Score
Validation	91.02%	92.00%	89.84%	90.91%

Diese Werte deuten darauf hin, dass das Modell in der Lage ist, die Maskenerkennung mit hoher Zuverlässigkeit durchzuführen. Besonders der F1-Score von über 90% zeigt ein ausgewogenes Verhältnis zwischen Precision und Recall.

Zur qualitativen Beurteilung wurden zudem Beispielvorhersagen, Abbildung 5 illustriert von 15 Stichprobe der Vorhersagen. In dem roten Kasten ist ein Beispiel markiert, bei dem das Modell fälschlicherweise `without_mask` vorhergesagt hat. Solche Fehler können durch unzureichende Beleuchtung, ungünstige Blickwinkel oder ungewöhnliche Maskenformen entstehen.

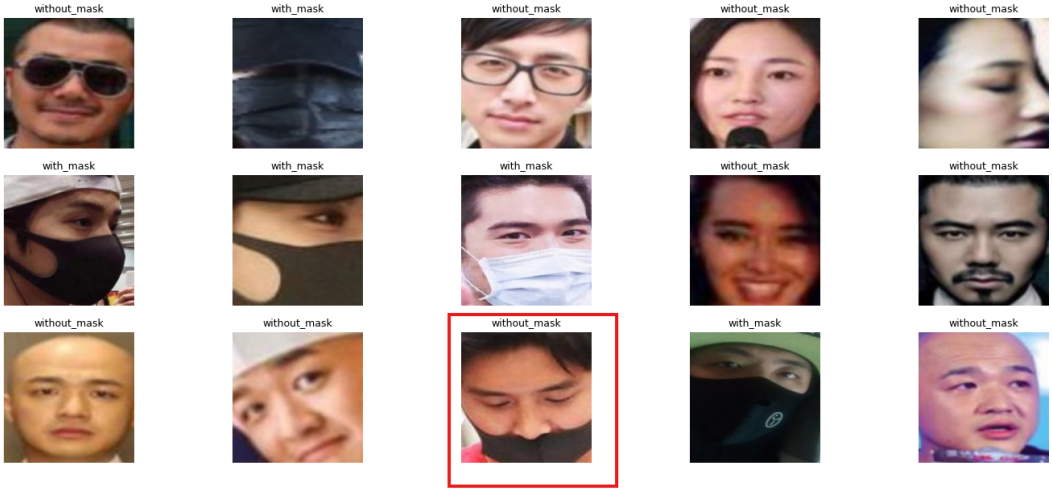


Figure 5: Beispielhafte Vorhersagen aus dem Validierungsset

Nachdem 5 oder 10 weitere verschiedene Beispiele aus 512 Bildern ausgeführt haben, bestätigen die Gesamtvalidierungsergebnisse und die Analyse der Fehlklassifizierungen, dass das Modell gut verallgemeinert werden kann und speziell für sicherheitskritische Erkennungen (Fehlalarme beim Tragen einer Maske) konzipiert ist.

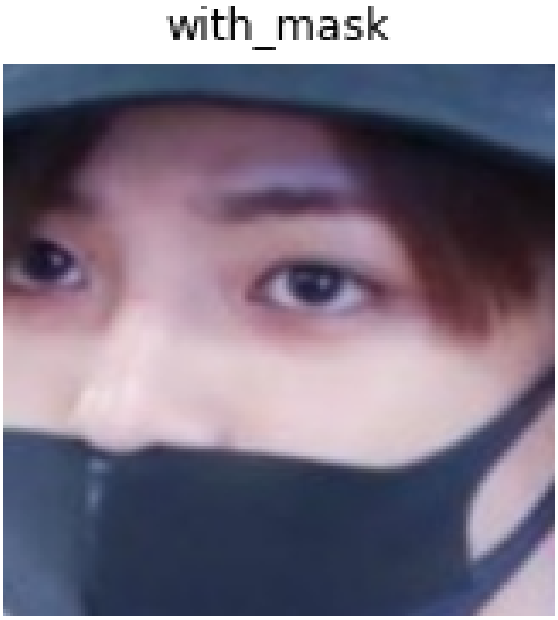


Figure 6: Eine Beispielhafte aus dem Validierungsset

3.2 ResNet-50 und Transfer Learning

ResNet-50 ist ein tiefes Convolutional Neural Network mit 50 Schichten. Dank sogenannter Residual-Blöcke kann es sehr tiefe Architekturen effizient trainieren, da die Skip-Connections das vanishing-gradient Problem vermeiden und den Informationsfluss stabil halten [12]. Durch sein Design kann es komplexe Merkmale aus Bildern extrahieren und wird deshalb häufig als Feature Extractor in Transfer-Learning-Szenarien eingesetzt.

Beim Transfer Learning wird die Idee verfolgt, die auf einem großen Datensatz wie ImageNet vortrainierten Gewichte für eine neue, aber verwandte Aufgabe zu nutzen. Das Ziel ist es, von den bereits gelernten generischen Bildfeatures zu profitieren und so die Trainingszeit und den Bedarf an großen beschrifteten Datensätzen deutlich zu reduzieren [13, 14].

In diesem Projekt wurde das **Feature Extraction**-Verfahren genutzt, wobei alle Convolutional-Layer des vortrainierten ResNet-50 eingefroren und nur die finale Klassifikationsschicht für die binäre Aufgabe neu trainiert wurde.

3.2.1 Implementierung im Projekt

Die Anpassung des ResNet-50-Modells für unsere binäre Klassifikationsaufgabe wurde in einem dedizierten Modul `resnet50_transfer.py` implementiert. Dabei wird das vortrainierte ImageNet-Modell geladen mit `from torchvision import models` , alle Layer bis auf die letzte Schicht werden eingefroren, und die finale fully-connected-Schicht wird auf zwei Klassen ("with_mask" und "without_mask") angepasst.Indem Function get-resnet50-model mit num-classes = 2 bezeichnet werden.

Listing 6: Modell-Initialisierung mit Feature Extraction

```
def get_resnet50_model(num_classes=2, pretrained=True,
   feature_extract=True):
    model = models.resnet50(pretrained=True)
    if feature_extract:
        for param in model.parameters():
            param.requires_grad = False
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, num_classes)
    return model
```

Das Transfermodell wird daher grundsätzlich mit vortrainierten ImageNet-Gewichten geladen. Mit `feature extract` werden alle vorhandenen Layer eingefroren und fungieren als fester Feature-Extraktor [14]. Nur der letzte vollständig verbundene Layer wird modifiziert und für zwei Ausgabeklassen trainiert. Diese Architektur nutzt die bereits gelernten, generischen Features des ResNet-50-Modells und beschränkt das Training auf die Entscheidungsschicht. Das reduziert Overfitting und ist rechnerisch sehr effizient.

3.2.2 Evaluation und Ergebnisse

Nach dem Training wurde das Modell mit einem separaten Validierungsset evaluiert. Die berechneten Metriken zeigen eine sehr hohe Klassifikationsqualität:

Table 3: Validierungsmetriken für das ResNet-50 Modell

	Accuracy	Precision	Recall	F1-Score
Validation	99.02%	100.00%	98.05%	99.01%

Diese Ergebnisse belegen, dass das gewählte Transfer-Learning-Verfahren mit Feature Extraction in diesem Projekt sehr effektiv war. Trotz der Beschränkung auf einen kleinen Datensatz konnte eine nahezu perfekte Trennung erreicht werden. Für zukünftige Arbeiten könnte zusätzliches Fine Tuning auf einem größeren und diverseren Datensatz helfen, verbleibende False Negatives weiter zu reduzieren.

Die Abbildung zeigt eine Auswahl von Modellvorhersagen aus dem Validierungsdatensatz. Alle abgebildeten Beispiele wurden korrekt klassifiziert, was die hohe Qualität der Feature-Extraktion und die Generalisierung des vortrainierten ResNet-50 Netzwerks unterstreicht.

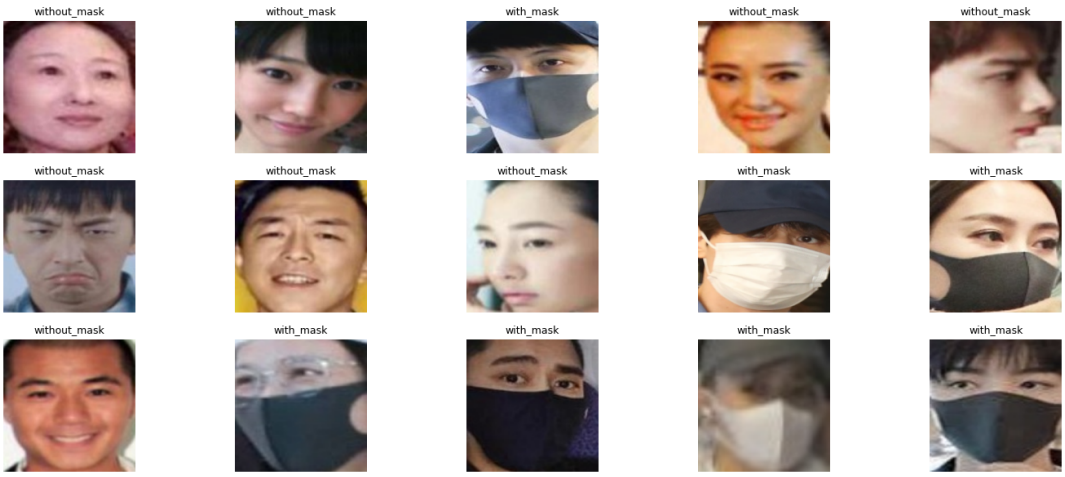


Figure 7: ResNet-50 Modells auf Sample.

4 Evaluieren und Visualisieren

In diesem Abschnitt werden die Ergebnisse von ResNet-50 und SimpleCNN verglichen. Neben dem Loss-Verlauf werden Metriken und Confusion-Matrizen betrachtet, um die jeweiligen Stärken und Schwächen zu erkennen. Die beider als Validation Datensatz von 512 und 1280 Training Datensatz per Klassen durchgeführt.

Abbildung 8 zeigt, dass ResNet-50 dank vortrainierter Gewichte von Beginn an niedrigere Loss-Werte und eine stabilere Konvergenz erreicht. SimpleCNN weist insgesamt höhere Werte und stärkere Schwankungen auf.

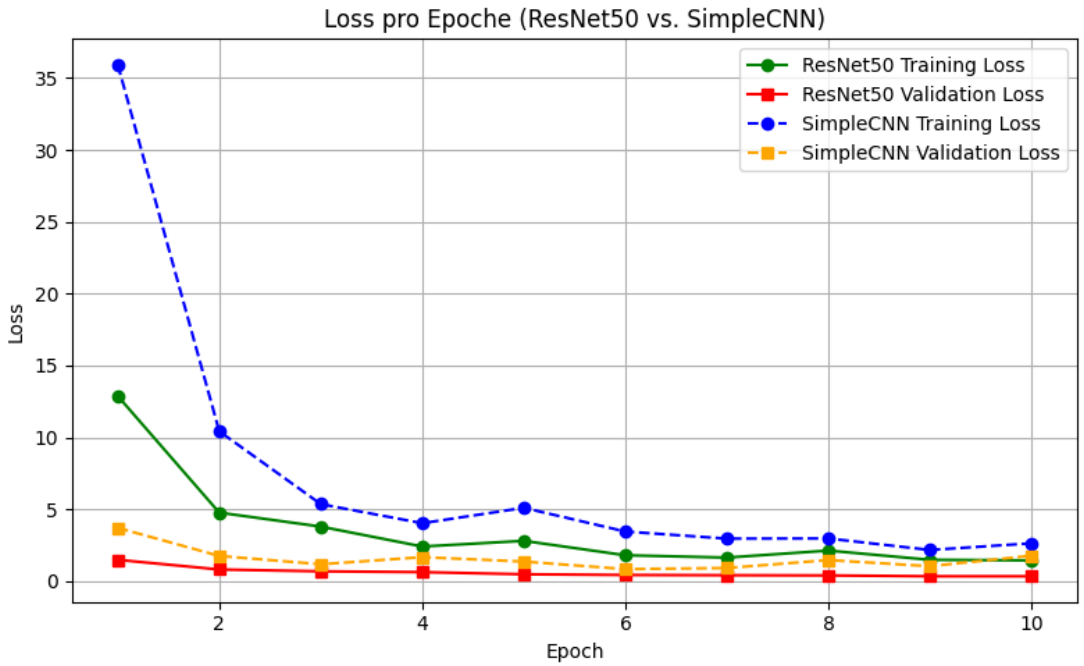


Figure 8: Loss Werte für ResNet-50 und SimpleCNN.

Die folgenden Abbildungen illustrieren die Entwicklung von Accuracy, Precision, Recall und F1-Score über alle Epochen. ResNet-50 erzielt insgesamt höhere und stabilere Werte, insbesondere Precision bleibt nahe 100%, während Recall nur leichte Schwankungen zeigt. SimpleCNN zeigt geringere Werte und größere Varianz.

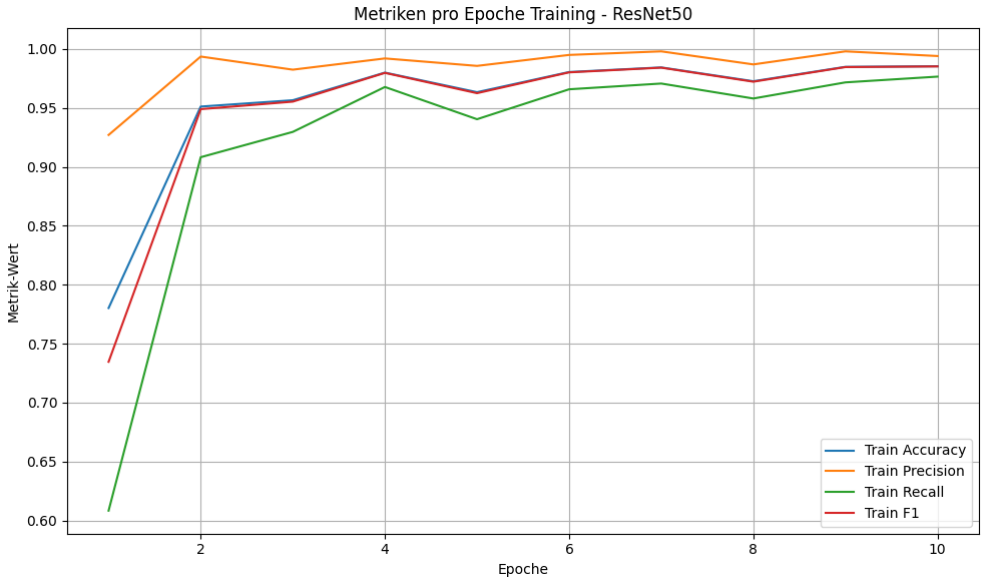


Figure 9: Trainingsmetrikenfür ResNet-50.

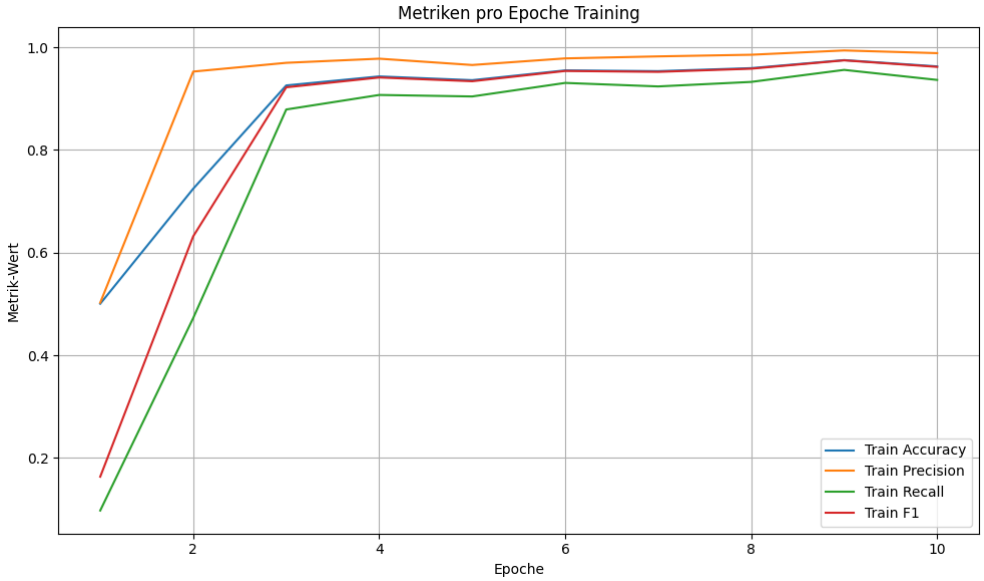


Figure 10: Trainingsmetriken für SimpleCNN.

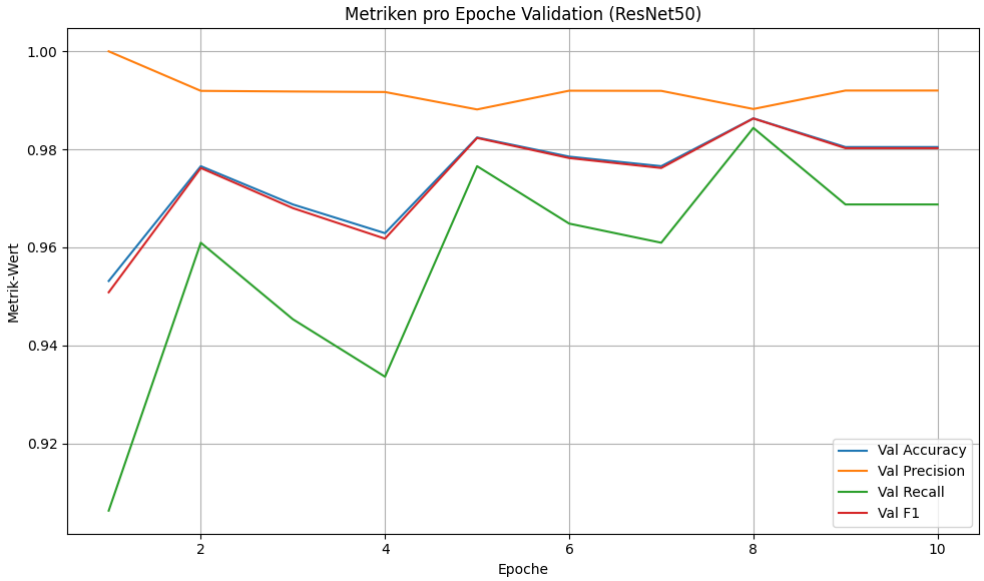


Figure 11: Validierungsmetriken für ResNet-50.

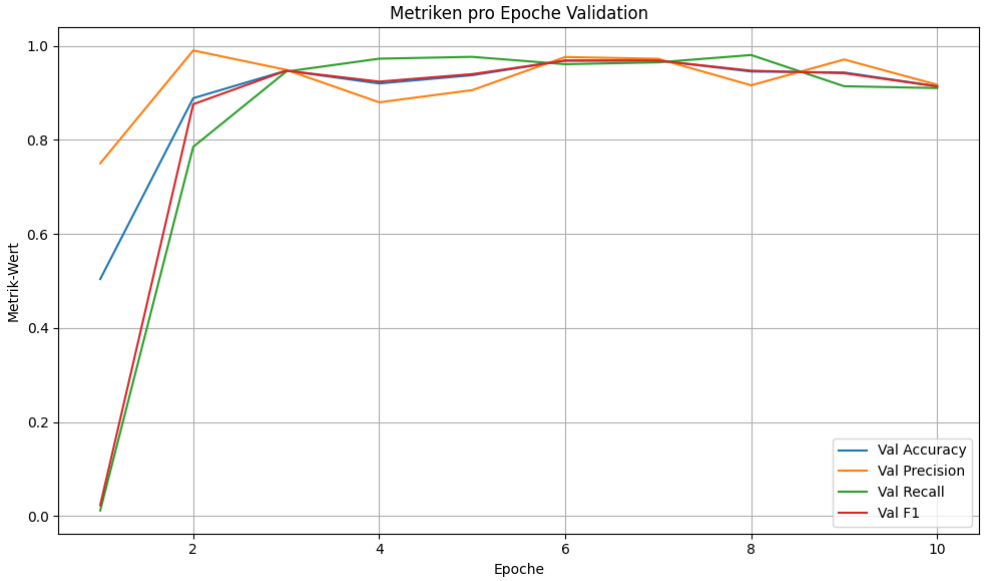
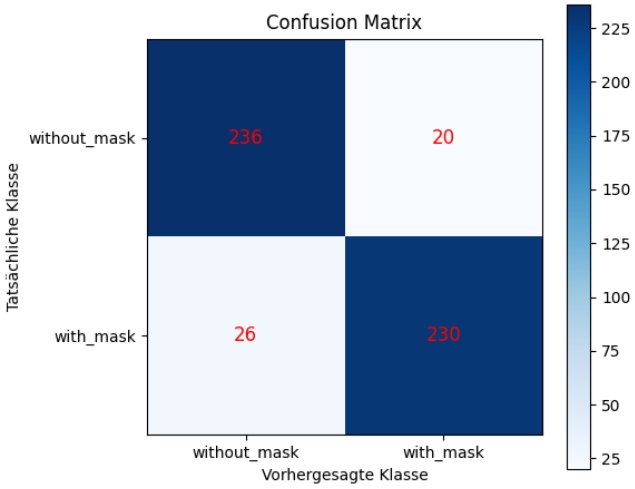
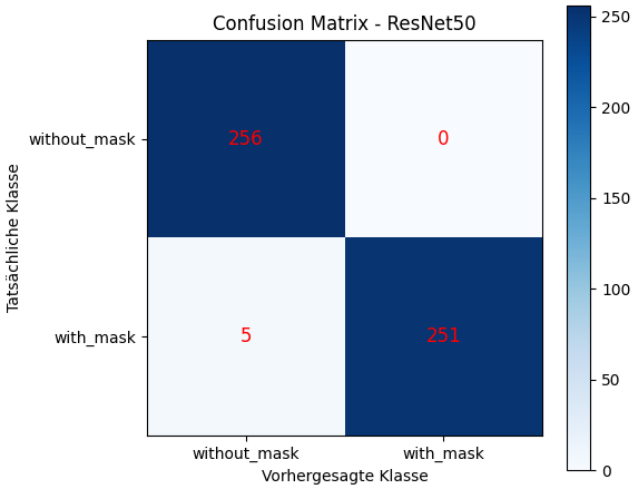


Figure 12: Validierungsmetriken für SimpleCNN.

Die Confusion-Matrizen in Abbildung 13 zeigen ebenfalls klare Unterschiede: ResNet-50 produziert kaum False Positives und wenige False Negatives, was auf eine sehr zuverlässige Erkennung hinweist. SimpleCNN weist deutlich mehr Fehlklassifikationen in beiden Klassen auf und zeigt damit eine geringere Generalisierungsfähigkeit auf dem Validation-Set.



(a) Confusion Matrix SimpleCNN



(b) Confusion Matrix ResNet-50

Figure 13: Confusion-Matrizen für SimpleCNN und ResNet-50.

Insgesamt bestätigt der Vergleich den Vorteil von Transfer Learning mit ResNet-50, das auch bei begrenzten Daten eine robustere und präzisere Klassifikation ermöglicht.

5 Fazit

Beide Modelle konnten die Bilder im Datensatz mit hoher Genauigkeit den Labels „with_mask“ und „without_mask“ zuordnen und die Klassifikationsaufgabe grundsätzlich erfolgreich lösen. ResNet-50 erzielte dabei insgesamt stabilere und bessere Ergebnisse als SimpleCNN, insbesondere durch höhere Precision und geringere Verlustwerte über die Epochen.

Die Ergebnisse zeigen, dass das gewählte Transfer-Learning-Verfahren mit Feature Extraction auf ResNet-50 eine sehr hohe Precision von 100 % erreichen konnte, jedoch mit einem Recall von ca. 98 % einige Maskenträger nicht erkannt wurden. Diese Verteilung hängt auch mit der Größe und Balance des gewählten Validierungs-Subsets zusammen, das bewusst kleiner gewählt wurde, um das Training effizienter und speicherschonender zu gestalten. Zwar vereinfacht das Einfrieren aller vortrainierten Layer das Training, schränkt jedoch die Möglichkeit ein, feinere Unterschiede zwischen Maskentypen zu lernen.

Für zukünftige Arbeiten wird angestrebt, größere und vielfältigere Datensätze einzusetzen und auch Fine Tuning tieferer Schichten zu berücksichtigen, um die Generalisierungsfähigkeit des Modells weiter zu verbessern.

Literaturverzeichnis

[1] Zhongyuan Wang et al. “Masked face recognition dataset and application”. In: *IEEE Transactions on Biometrics, Behavior, and Identity Science* 5.2 (2023), pp. 298–304.

[2] Wikipedia. *Convolutional Neural Network*. 2023. URL: `https://de.wikipedia.org/wiki/Convolutional_Neural_Network`.

[3] Python Docs. *idx with class*. Zugriff am 2024-06-XX. 2019. URL: `https://discuss.pytorch.org/t/how-to-represent-class-to-idx-map-for-custom-dataset-in-pytorch/37510`.

[4] PyTorch. *torchvision.datasets.ImageFolder*. Zugriff am 2024-06-XX. 2024. URL: `https://docs.pytorch.org/vision/main/generated/torchvision.datasets.ImageFolder.html`.

[5] StackOverflow. *Taking subsets of a PyTorch Dataset*. Zugriff am 2024-06-XX. 2017. URL: https://stackoverflow.com/questions/47432168/taking-subsets-of-a-pytorch-dataset.

[6] ChatGPT. *Diskussion und Erarbeitung einer Label-Mapping-Korrektur für ImageFolder-Datensätze*. Beratungsgespräch am 2024-06-26. 2024. URL: https://openai.com/chatgpt.

[7] StackOverflow. *Create destination path for shutil.copy files*. Zugriff am 2024-06-XX. 2010. URL: https://stackoverflow.com/questions/2793789/create-destination-path-for-shutil-copy-files.

[8] Python Docs. *shutil - High-level File Operations*. Zugriff am 2024-06-XX. 2024. URL: https://docs.python.org/3/library/shutil.html.

[9] Juan C. Olamendy. *Understanding ReLU, LeakyReLU and PReLU: A Comprehensive Guide*. Zugriff am 2024-06-26. 2023. URL: https://medium.com/@juanc.olamendy/understanding-relu-leakyrelu-and-prelu-a-comprehensive-guide-20f2775d3d64.

[10] PyTorch. *torch.nn.ConvTranspose2d*. Zugriff am 2024-06-26. 2024. URL: https://docs.pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html.

[11] DataScientest. *Convolutional Neural Networks (CNN)*. Zugriff am 2024-06-26. 2023. URL: https://datascientest.com/de/convolutional-neural-network-2.

[12] Roboflow. *What is ResNet-50?* 2023. URL: https://blog.roboflow.com/what-is-resnet-50/.

[13] The Data Frog. *Image Recognition and Transfer Learning*. URL: https://thedatafrog.com/en/articles/image-recognition-transfer-learning/. 2023.

[14] Medium Author. *Deep Learning using Transfer Learning – Python Code for ResNet50*. URL: https://medium.com/data-science/deep-learning-using-transfer-learning-python-code-for-resnet50-8acdfb3a2d38. 2020.