Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Elektro- und Informationstechnik

Deep Vision Projekt

von

Fabian Schmidt

Who is allowed to enter the classroom: Contrastive Learning for face and signature recognition

Inhaltsverzeichnis

Kapitel 1

Ziel des Projekts

Ziel dieses Projekts soll es seien eine KI-Anwendung zu entwickeln und zu trainieren, welche in der Lage seien soll zu bestimmen, ob eine Person Zugang zu einem Klassenzimmer bekommt, oder nicht. Die Entscheidung trifft das Modell anhand von Gesichts- und Unterschriftbildern. Dem Modell werden je zwei Gesichts- und Unterschriftbilder gezeigt, anhand derer das Modell eine Ähnlichkeit dieser Bilder zueinander berechnet. Ist eine dieser Ähnlichkeiten zu klein, gewährt das Modell keinen Zugang.

Die Implementierung dieser KI-Anwendung umfasst mehrere Schritte, darunter die Datensammlung, die Vorverarbeitung der Bilder, die Entwicklung und das Training des Modells sowie die Evaluierung der Leistung. In den Folgenden Kapiteln werden die verwendeten Datensätze und Dataloader, die Modell Architektur, Trainings- und Inferenceprozess sowie Evaluierung der Modellperformance. Zuletzt wird Ausblick gegeben und ein Fazit gezogen.

Kapitel 2

Datensätze

Im Projekt werden zwei Datensätze verwendet. Ein Datensatz für Gesichter, ein Datensatz für Unterschriften. Im Folgenden wird die Struktur und Inhalte beider Datensätze dargelegt.

2.0.1 Gesichtsdatensatz

Der Gesichterdatensatz [1] enthält 10 verschiedene Bilder von 40 Personen. Für manche Personen enthält der Datensatz Bilder, die zu unterschiedlichen Zeiten, Beleuchtung, Gesichtszüge (lächeln/nicht lächeln, Augen geschlossen/offen) und Accessoires (Brille/keine Brille). Der Datensatz enthält 40 Unterordner, die jeweils 10 Bilder beinhalten. Die Bilder sind alle im .pgm-Dateiformat gespeichert und haben eine Breite von 92 Pixeln und eine Höhe von 112 Pixeln. Der Datensatz beinhaltet keinen vordefinierten Trainings- bzw. Testdatensatz.

2.0.2 Unterschriftendatensatz

Der Unterschriftendatensatz [6] besteht aus 119 Unterschrift verschiedener Personen. Jede Person hat hierbei mehrere Unterschriften getätigt. Des Weiteren beinhaltet der Datensatz nochmal 119 Unterschrift, die jedoch gefälscht sind. Zusammengefasst besteht der Datensatz 238 Unterordnern mit jeweils 8 bis 24 Bildern von Unterschriften. Die Bilder sind alle im .png-Dateiformat gespeichert und haben keine einheitliche Größe. Der Datensatz beinhaltet ebenfalls einen vordefinierten Trainings- und Testdatensatz im .csv-Dateiformat. Der Trainingsdatensatz beinhaltet 76721 Samples, der Testdatensatz 8525 Samples.

Kapitel 3

Dataloader

Der folgende Abschnitt beschreibt, wie aus den in Kapitel 2 beschriebenen Datensätzen, PyTorch Dataset-Klassen und Dataloader [2] erstellt wurden.

3.1 SiameseDataset

Die Klasse `SiameseDataset` ist eine abgeleitete Klasse von `torch.utlis.data.Dataset` und wird verwendet, um Paare von Bildern aus dem Gesichtsdatensatz [1] und Unterschriftendatensatz [6] zu erzeugen. Listing 1 zeigt den hierzu erstellten Python Quellcode.

```
class SiameseDataset(Dataset):
    def __init__(self, imageFolderDataset, transform=None):
        self.imageFolderDataset = imageFolderDataset
        self.transform = transform

    def __getitem__(self, index):
        img0_tuple = random.choice(self.imageFolderDataset.imgs)

        # We need approximately 50% of images to be in the same class
        should_get_same_class = random.randint(0, 1)
        if should_get_same_class:
            while True:
                # Look until the same class image is found
                img1_tuple = random.choice(self.imageFolderDataset.imgs)
                if img0_tuple[1] == img1_tuple[1]:
                    break
        else:
            while True:
                # Look until a different class image is found
                img1_tuple = random.choice(self.imageFolderDataset.imgs)
                if img0_tuple[1] != img1_tuple[1]:
                    break

        img0 = Image.open(img0_tuple[0])
        img1 = Image.open(img1_tuple[0])

```

```
27          img0 = img0.convert("L")
28          img1 = img1.convert("L")
29
30          if self.transform is not None:
31              img0 = self.transform(img0)
32              img1 = self.transform(img1)
33
34          return img0, img1, torch.from_numpy(np.array([int(img1_tuple[1]
    != img0_tuple[1])], dtype=np.float32))
35
36      def __len__(self):
37          return len(self.imageFolderDataset.imgs)
```

Listing 3.1: SiameseDataset Class Implementation

3.1.1 Konstruktor

- `__init__(self, imageFolderDataset, transform=None)`: Initialisiert das Dataset mit gegebenen Bilderordner und Transformationen.

3.1.2 Methoden

- `__getitem__(self, index)`: Gibt ein Bilderpaar zurück. Etwa 50% der Bilderpaare gehören der gleichen Klasse an.
- `__len__(self)`: Gibt die Anzahl der Samples des Datensatzes zurück.

3.1.3 Dataset Augmentation

Der Datensatz wird mithilfe der Pytorch Modulen `datasets` [2] und `transforms` [8] initialisiert. Die Transformationen bestehen aus:

- Größenänderung auf 100x100
- Zufälliger horizontaler Seitenwechsel
- zufällige Rotation um bis zu 10 Grad
- zufälliger Bildausschnitt auf 100x100 skaliert
- zufällige Farbänderung
- Transformation zu Tensor und Normalisierung

```
1 # Define the transformations for train, validation, and test
2 train_transform = transforms.Compose([
3     transforms.Resize((100, 100)),
4     transforms.RandomHorizontalFlip(),
5     transforms.RandomRotation(10),
6     transforms.RandomResizedCrop(100, scale=(0.8, 1.0)),
7     transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
      hue=0.1),
8     transforms.ToTensor(),
9     transforms.Normalize(mean=[0.5], std=[0.5])
```

```
])

valid_test_transform = transforms.Compose([
    transforms.Resize((100, 100)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

Listing 3.2: Dataset Initialization and Augmentation

Die Initialisierung des Unterschriften Datasets erfolgt analog, allerdings mit weniger Transformationen, da im Verlauf des Projekts festegestellt wurde, dass das Netzwerk mit mehr Transformationen nicht lernt.

3.1.4 Dataset Splitting

Aufteilung des Datensatzes in Traings-, Test- und Validierungsdatensatz.

```
# Define Dataset Folder
face_folder_dataset = datasets.ImageFolder(root="./data/faces/")

# Split the dataset into train, validation, and test sets
train_size = int(0.8 * len(face_folder_dataset))
valid_size = int(0.1 * len(face_folder_dataset))
test_size = len(face_folder_dataset) - train_size - valid_size
face_train_dataset, face_valid_dataset, face_test_dataset = torch.utils.
    data.random_split(face_folder_dataset, [train_size, valid_size,
    test_size])

# Apply the transformations
face_train_dataset = SiameseDataset(subset=face_train_dataset, transform
    =train_transform)
face_valid_dataset = SiameseDataset(subset=face_valid_dataset, transform
    =valid_test_transform)
face_test_dataset = SiameseDataset(subset=face_test_dataset, transform=
    valid_test_transform)
```

Listing 3.3: Dataset Splitting

Wie aus den Listings 3.2 und 3.3 hervorgeht, werden nur die jeweiligen Trainingsdatensätze Augmentiert, aber nicht die Test- bzw. Validierungsdatensätze. So wird für den Trainingsdatensatz mehr Varianz generiert und zugleich sichergestellt, dass die Test- und Validierungsdatensätze konstant bleiben.

Länge der Gesichtsdatensätze:

- Trainingsdatensatz: 320 Samples
- Validierungsdatensatz: 40 Samples
- Testdatensatz: 40 Samples

Abbildung 3.1: Batch aus Gesichts Trainingsdatensatz

Abbildung 3.2: Batch aus Gesichts Testdatensatz

Länge der Unterschriftendatensätze:

- Trainingsdatensatz: 3431 Samples
- Validierungsdatensatz: 428 Samples
- Testdatensatz: 430 Samples

Abbildung 3.3: Batch aus Unterschriften Trainingsdatensatz

Abbildung 3.4: Batch aus Unterschriften Testdatensatz

Kapitel 4

Modelle

Der Folgende Abschnitt beschreibt die Architektur und die Verwendung des verwendeten Modells. Hierbei wurde ein Modell entwickelt und trainiert. Das `SiameseModel` dient zur Bestimmung von Ähnlichkeiten zwischen zweier Bilder. Das `SiameseModel` wurde zweimal auf dem `SiameseDataset` trainiert. Einmal auf Unterschriften, einmal auf Gesichter.

4.1 SiameseModel

4.1.1 Architektur

Als Backbone-Modell wird ResNet18 [4] verwendet. Da `SiameseDataset` Graustufen liefert, wurde das erste Convolutional Layer des ResNets modifiziert. Des Weiteren wurde das letzte Linear Layer durch mehrere Lineare Layer mit ReLU-Aktivierungsfunktion, Batchnormalisierung und Dropout ersetzt.

Block	Number of Layers	Kernel Size	in channels	out channels	activation
Conv1	1	7x7	3	64	ReLU
MaxPool	1	3x3	64	64	N/A
Res1	6	3x3	64	64	ReLU
Res2	1	3x3	64	128	ReLU
Res2	7	3x3	128	128	ReLU
Res3	1	3x3	128	256	ReLU
Res3	11	3x3	256	256	ReLU
Res4	1	3x3	256	512	ReLU
Res4	5	3x3	512	512	ReLU
AVGPool	1	N/A	512	512	N/A
fc1	1	N/A	512	1024	ReLU
fc1	1	N/A	1024	256	ReLU
fc1	1	N/A	256	2	N/A

Tabelle 4.1: `SiameseNetwork Architektur`

Das Netzwerk hat insgesamt ca. 24 Millionen Parameter und ist fertig trainiert 143MB Groß

4.1.2 Training

Die Modelle wurden auf ihren jeweiligen Trainings- und Validierungsdatensätzen mit einer Batchsize von 256. Als Loss-Funktion wir ein Contrastive Loss verwendet. So wird der Loss für ähnliche Bilder minimiert, während er für unähnliche Bilder maximiert wird.

```
class ContrastiveLoss(torch.nn.Module):
    def __init__(self, margin=5.0):
        super(ContrastiveLoss, self).__init__()
        self.margin = margin

    def forward(self, output1, output2, label):
      euclidean_distance = F.pairwise_distance(output1, output2, keepdim
    = True)
      loss_contrastive = torch.mean((1-label) * torch.pow(
   euclidean_distance, 2) + (label) * torch.pow(torch.clamp(self.margin
   - euclidean_distance, min=0.0), 2))

      return loss_contrastive
```

Listing 4.1: Loss Function

Als Optimizer wird Adam [5] mit einer Lernrate von 0.00001 verwendet. Der Loss auf dem Validierungsdatensatz dient als Early Stopping Kriterium, wird dieser über 15 Iterationen nicht besser, wird das Training beendet. Das Modell trainiert maximal für 100 Epochen. Dieser Ansatz wird sowohl für den Gesichter als auch für den Unterschriftendatensatz angewendet.
Die trainierten Modelle werden als Checkpoints gespeichert.

4.1.3 Inference

In der Inferenz werden beide Modelle zunächst auf den `.eval()` Modus geschaltet. Dann werden an beide Modelle ein Bilderpaar aus dem jeweiligen Testdatensatz gegeben. Das Modell gibt Embeddings, mit denen der euklidische Abstand zwischen den Embeddings berechnet wird. Ist dieser Abstand für Gesichter kleiner als `1.5` werden die Gesichter zur selben Klasse gezählt, für Unterschrift muss der Abstand kleiner sein als `0.4`. Beide Werte wurden durch praktische Erprobung ermittelt. Zählt das Gesichtspaar und das Unterschriftenpaar jeweils zur selben Klasse, wird Eintritt gewährt. Zur Überprüfung, ob diese Zuordnung korrekt ist, wird Gesichts und Unterschriften Label zu einem Boolschen-Wert verknüpft. Zuletzt werden alle evaluierten Samples visualisiert.

```
face_net.eval()
signature_net.eval()

face_test_dataloader = DataLoader(face_test_dataset, num_workers=2,
   batch_size=1, shuffle=True)
```

```
signature_test_loader = DataLoader(signature_test_dataset, shuffle=True,
    num_workers=2, batch_size=1)


face_dataiter = iter(face_test_dataloader)
signature_dataiter = iter(signature_test_loader)

should_have_access_list = []
would_gain_access_list = []

for i in range(len(face_test_dataloader)):

    face0, face1, face_label = next(face_dataiter)
    signature1, signature2, forge_label = next(signature_dataiter)

    face_output1, face_output2 = face_net(face0.to(device), face1.to(
   device))
    euclidean_distance = F.pairwise_distance(face_output1, face_output2)

    face_distance = (euclidean_distance < 1.5).bool()
    signature_output1, signature_output2 = signature_net(signature1.to(
   device), signature2.to(device))

    euclidean_distance = F.pairwise_distance(signature_output1,
   signature_output2)
    signature_distance = (euclidean_distance < 0.4).bool()

    # Concatenate the two images together
    would_gain_access = face_distance.cpu()[0] and signature_distance.
   cpu()[0]

    should_have_access = not face_label.cpu()[0][0].bool() and not
   forge_label.cpu()[0][0].bool()

    should_have_access_list.append(should_have_access)
    would_gain_access_list.append(would_gain_access)

    concatenated = torch.cat((face0, face1, signature1, signature2), 0)
    imshow(torchvision.utils.make_grid(concatenated), f'Should have
   Access: {should_have_access}; Would gain Access: {would_gain_access}'
   )
```

Listing 4.2: Inference Code

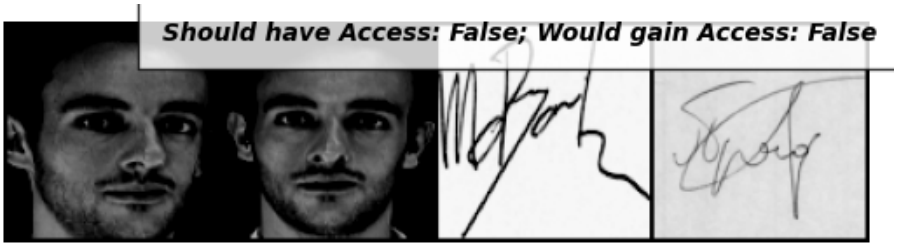

Abbildung 4.1: True Negative Sample

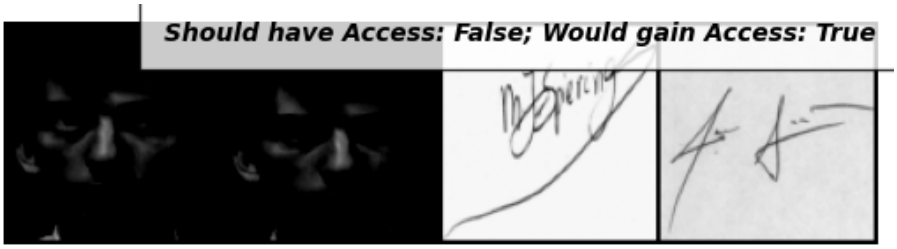

Abbildung 4.2: False Positve Sample

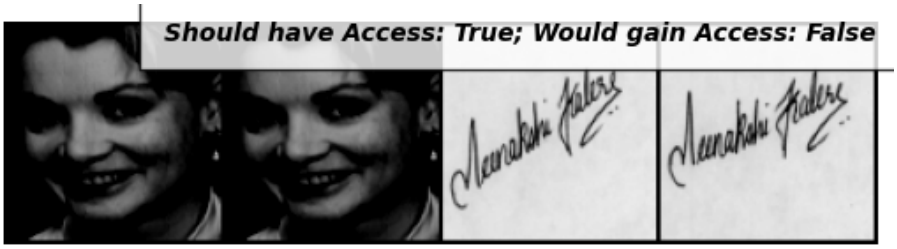

Abbildung 4.3: False Negative Sample

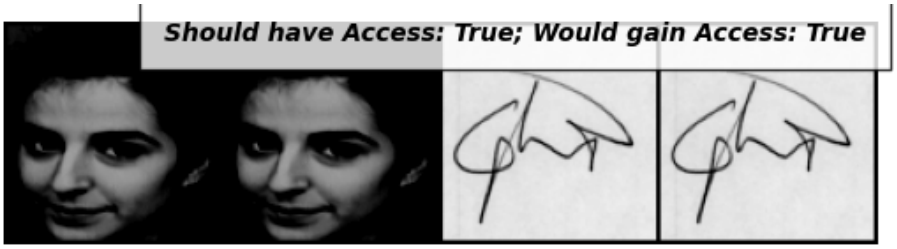

Abbildung 4.4: True Positive Sample

Aus den Variablen `should_have_access` und `would_gain_access` wird eine Konfusionsmatrix erzeugt.

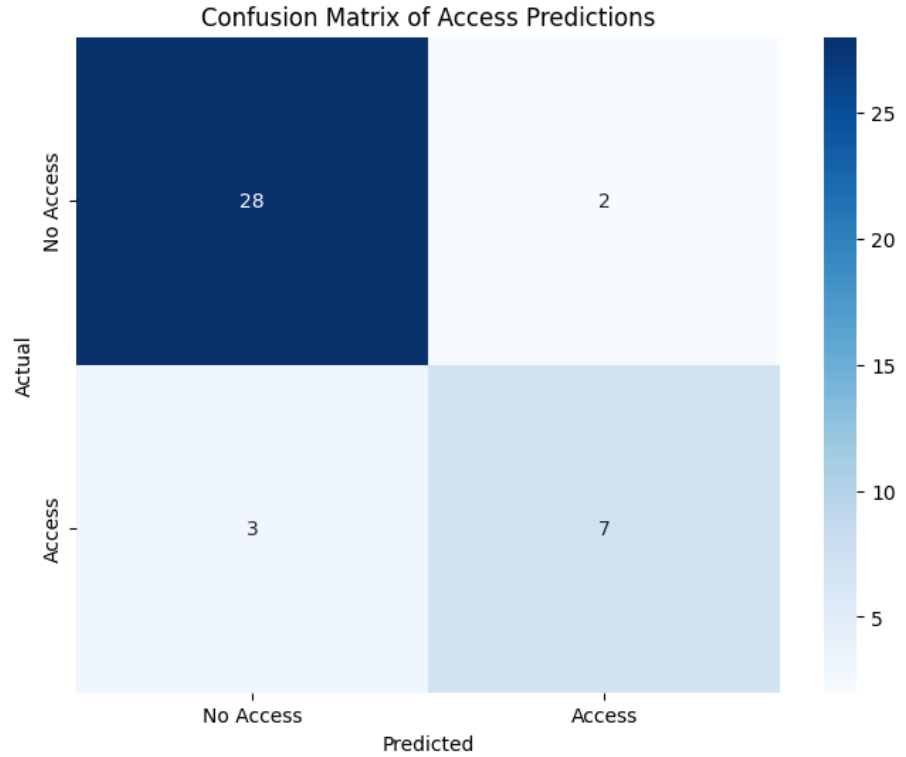

Abbildung 4.5: Konfusionsmatrix

4.1.4 Performance

Mit den Werten aus der Konfusionsmatrix werden nun die Accuracy, Recall und Precision berechnet.

Accuracy

Accuracy beschreibt die Proportion von korrekten Vorhersagen (true positives und true negatives) gegenüber allen Vorhersagen.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{7 + 28}{7 + 28 + 2 + 3} = 0.875$$

Eine Accuracy von 0.875 beschreibt, dass 87.5% aller klassifizierten Fälle korrekt klassifiziert wurden.

Precision

Precision beschreibt die Proportion der korrekt als positiv klassifizierten Fälle gegenüber allen als positiv klassifizierten Fällen (true positives und false positives).

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{7}{7 + 2} = \frac{7}{9} \approx 0.778$$

Eine Precision von 0.778 zeigt, dass 77.8% aller positiv klassifizierten Fälle tatsächlich korrekt sind.

Recall (Sensitivity)

Recall beschreibt die Proportion der korrekt als positiv klassifizierten Fälle gegenüber allen tatsächlichen positiven Fällen (true positives und false negatives).

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{7}{7+3} = \frac{7}{10} = 0.7$$

Ein Recall von 0.7 zeigt, dass 70% aller tatsächlich positiven Fälle korrekt klassifiziert werden.

F1 Score

Der F1 Score ist das harmonische Mittel von Precision und Recall und bietet ein ausgewogenes Maß zwischen beiden.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \cdot \frac{0.778 \cdot 0.7}{0.778 + 0.7} = 2 \cdot \frac{0.5446}{1.478} \approx 0.736$$

Specificity

Specificity beschreibt die Proportion der korrekt als negativ klassifizierten Fälle gegenüber allen tatsächlichen negativen Fällen (true negatives und false positives).

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{28}{28+2} = \frac{28}{30} \approx 0.933$$

Gesamteinschätzung

Das Modell zeigt insgesamt gute Leistung:

- Accuracy ist hoch, was auf eine insgesamt hohe Vorhersagegenauigkeit hinweist.
- Precision ist solide, was bedeutet, dass ein großer Teil der positiven Vorhersagen korrekt ist.
- Recall ist akzeptabel, aber könnte in Situationen, in denen es entscheidend ist, alle positiven Fälle zu identifizieren, verbessert werden.
- F1 Score zeigt eine gute Balance zwischen Precision und Recall.
- Specificity ist sehr hoch, was darauf hinweist, dass das Modell sehr zuverlässig negative Fälle identifiziert.

Die Metriken deuten darauf hin, dass das Modell zuverlässig, die True Positives und True Negatives Samples richtig klassifiziert ist, aber je nach Anwendungsfall könnte es sinnvoll sein, den Recall weiter zu verbessern, um sicherzustellen, dass möglichst viele positive Fälle erkannt werden.

Kapitel 5

Fazit

In diesem Projekt ging es um die Entwicklung eines Systems zur Gesichtserkennung und Unterschriftenerkennung, das den Zugang zu einem Klassenzimmer gewährt. Hierbei wurde ein auf ResNet basiertes Modell entwickelt, das beachtliche Ergebnisse in Bezug auf verschiedene Performance-Metriken erzielt hat. Die wichtigsten Kennzahlen für dieses Modell sind:

- Accuracy: 87.5%
- Precision: 77.8%
- Recall: 70%
- F1 Score: 73.6%
- Specificity: 93.3%

Diese Metriken zeigen, dass das Modell insgesamt gut funktioniert. Besonders hervorzuheben ist die hohe Specificity, die anzeigt, dass das Modell sehr zuverlässig in der Erkennung negativer Fälle ist. Dies ist von besonderer Bedeutung, um unbefugten Zugang effektiv zu verhindern. Die gute Accuracy und Precision zeigen, dass die Mehrheit der Vorhersagen korrekt ist und die meisten positiven Vorhersagen tatsächlich richtig sind. Der F1 Score zeigt eine gute Balance zwischen Precision und Recall, was auf ein ausgewogenes Modell hinweist. Der Recall-Wert könnte jedoch verbessert werden, um sicherzustellen, dass noch mehr positive Fälle erkannt werden, was in sicherheitskritischen Anwendungen wie dieser besonders wichtig ist.

Kapitel 6

Ausblick

Für zukünftige Arbeiten und Weiterentwicklungen dieses Projekts gibt es mehrere potenzielle Ansätze:

1. **Erhöhung des Recalls**: Um sicherzustellen, dass möglichst alle berechtigten Personen erkannt werden, könnten Methoden wie weitere Datenaugmentation oder das Hinzufügen zusätzlicher Trainingsdaten aus verschiedenen Blickwinkeln und Beleuchtungssituationen eingesetzt werden.
2. **Verbesserung der Modellarchitektur**: Es könnten neuere und leistungsfähigere Architekturen, wie z.B. Vision Transformers [3] oder VGG [7], in Betracht gezogen werden.
3. **Verwendung verschiedener Datensätze**: Um zu überprüfen, ob das Modell tatsächlich gut generalisiert, sollte das Modell ebenfalls mit anderen vergleichbaren Datensätzen trainiert werden. Wird mit anderen Datensätzen ein ähnliches Ergebnis erreicht wie in 4.1.4 kann davon ausgegangen werden, dass Modell gut generalisiert.
4. **Erweiterung des Anwendungsbereichs**: Das System könnte über die Klassenzimmerzugangskontrolle hinaus auf andere Bereiche ausgeweitet werden, wie z.B. der Zugang zu Gebäuden, Büros oder sicheren Bereichen.
5. **Robustheit und Sicherheit**: Die Robustheit gegen Spoofing-Angriffe, wie das Verwenden von Fotos oder Videos zur Täuschung des Systems, sollte weiter getestet und verbessert werden.

Insgesamt zeigt dieses Projekt das Potenzial von ResNet-basierten Modellen für Gesichtserkennungs- und Unterschriftenerkennungssysteme. Mit weiteren Optimierungen und Erweiterungen könnte das System zu einer robusten und zuverlässigen Lösung für Zugangskontrollen in verschiedenen Bereichen entwickelt werden.

Literatur

[1] Kasikrit Damkliang. *ATT Database of Faces*. 2002. URL: https://www.kaggle.com/datasets/kasikrit/att-database-of-faces/data (besucht am 26. 06. 2024).

[2] *Datasets & DataLoaders — PyTorch Tutorials 2.3.0+cu121 documentation*. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html. (Accessed on 07/01/2024). Juli 2024.

[3] Alexey Dosovitskiy u. a. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[4] Kaiming He u. a. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[5] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[6] ROBINRENI. $Signature_Verification_Dataset$. 2011. URL: https://www.kaggle.com/datasets/robinreni/signature-verification-dataset/data.

[7] Karen Simonyan und Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[8] *Transforming and augmenting images — Torchvision 0.18 documentation*. https://pytorch.org/vision/stable/transforms.html. (Accessed on 07/01/2024). Juli 2024.

Abbildungsverzeichnis

Tabellenverzeichnis