



# Bildverarbeitung - Abschlusspräsentation **CoinCat**

Tarik Glasmacher

Tim Bering

Bildverarbeitung WS22/23

Freiwilliges Abschlussprojekt

# Agenda

- Projektüberblick
- Motivation
- Problem
- Vorbereitung
- Methodik
- Evaluation
- Ergebnis
- Fazit
- Weiterführende Ideen



# Projektüberblick



# Überblick: CoinCat

- Nutzen: Zusammenzählen von Kleingeld
- Einfärben von Münzen mit gleichem Wert
- Markieren eines bestimmten Betrages
  - Verschiedene Filter
- Weitere genutzte Toolboxes
  - *Computer Vision Toolbox*
  - *Image Processing Toolbox*
  - *Image Manipulation Toolbox*
  - *Deep Learning Toolbox*
  - *Parallel Computing Toolbox*



Vorher

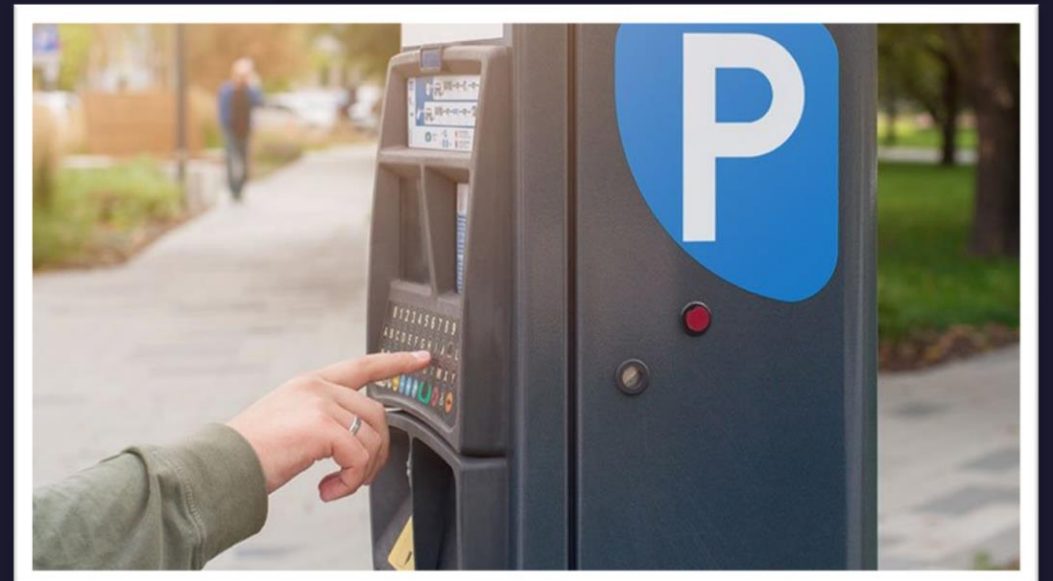
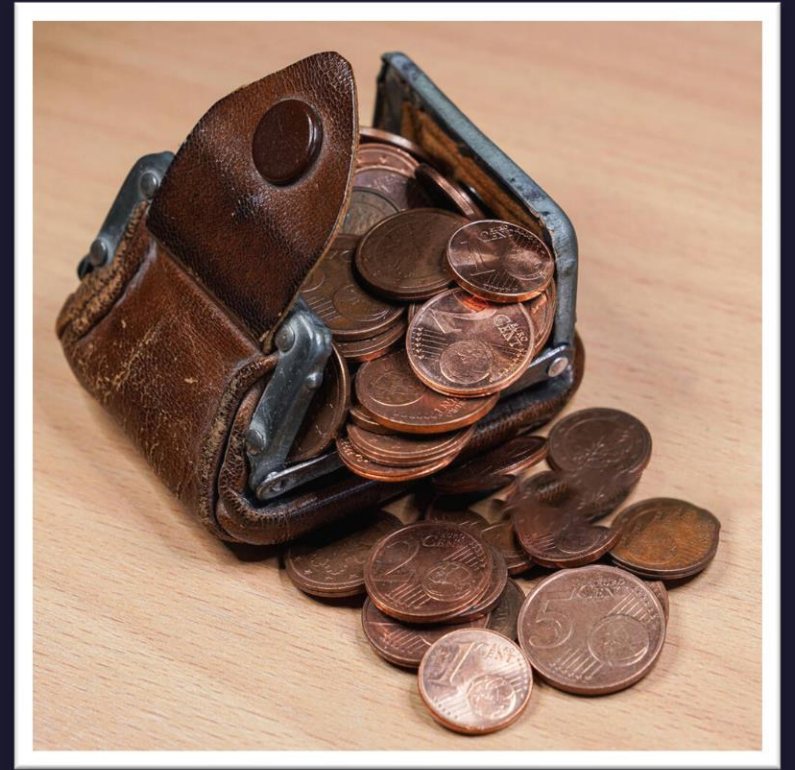


Nachher / Wert mindestens 0,50€

# Motivation

# Motivation

- Häufiger Gebrauch von Kleingeld
- Mühseliges Zusammenzählen
  - Überprüfen, ob genug Kleingeld für geplante Aktivität vorhanden ist
- Scheine zu Münzen wechseln für andere Personen
  - Für Parkautomaten
  - Für Raststätten-Toiletten





# Problem

# Problem

*Es dauert zu lange, das Geld zusammenzuzählen.*





# Vorbereitung

# Vorbereitung I – Aufbau der Münzen

- 8 verschiedene Münztypen
- Große Ähnlichkeiten zwischen einigen Münzen
  - 0,01€ ; 0,02€ ; 0,05€ (Rotgeld)
  - 0,10€ ; 0,20€ ; 0,50€ (Kleingeld)
- Rotgeld: Schriftzug oben rechts, Kreis unten rechts, Diagonale Linien
- Kleingeld: Schriftzug unten rechts, Zeichen links mittig, Vertikale Linien links
- Metalle der 1,00€- und 2,00€-Münze sind jeweils invertiert
- Teilreflektierende Oberflächen



# Vorbereitung II – Aufbau des Eingabebildes

- Potenzielle Überlagerung von Münzen
- Nicht durchgängig gleiche Vorderseite
- Unterschiedliche Belichtung
- Unterschiedliche Auflösung
- Unterschiedliche Brennweiten
- Abgeschnittene Münzen
- Perspektivische Verzerrung\*

*\*siehe Abschnitt „Methodik“*

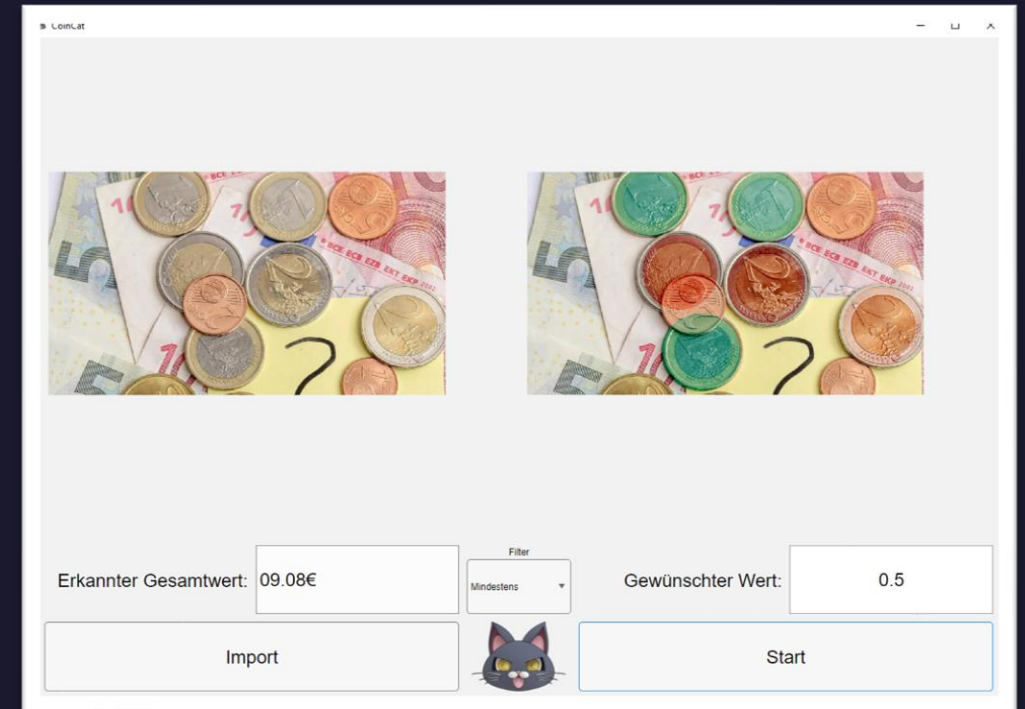




# Methodik

# Methodik - Übersicht

- Image Classification mit eigens trainiertem **neuralem Netzwerk**
- Trainiertes Netzwerk wird mehrfach genutzt
  - Netzwerk wird bei jedem Nutzen **nicht** neu trainiert
- 3-Step-Verfahren bei Bildeingabe:
  1. Vorverarbeitung
  2. Klassifikation
  3. Nachverarbeitung



# Methodik - Vorverarbeitung

## Benutzte Methoden:

`imfindcircles()` [Computer Vision Toolbox]

`viscircles()` [Computer Vision Toolbox]

`imcrop()` [Image Processing Toolbox]

`imresize()` [Image Processing Toolbox]

`ndgrid()`

- Hough-Transformation zur Erkennung von Kreisen
- Ausschneiden der einzelnen Kreise
  - Teilbildgröße auf 256x256px ändern
  - Eingabgröße für neurales Netzwerk
- Erstellt Cells: Index → Bild-Pfad | Position auf Eingabebild (X | Y) | Radius | Münzwert (Platzhalter)





# Methodik - Klassifizierung

Benutzte Methoden:

`classify()` [Deep Learning Toolbox]

`imageDatastore()` [Computer Vision Toolbox]

- Erstellen eines `imageDatastores` der Eingabedaten
- Einspeisen in neurales Netzwerk
- Auslesen der Wertigkeiten
- Evaluieren: *Handelt es sich bei dem Teilbild um eine Münze?*
  - Fester Wertigkeits-Schwellwert
  - Höchste Wertigkeit wird beachtet
- Ändert Dictionary: Index → Münzwert als Dezimalzahl
  - Bei Nicht-Münzen-Einträgen: Münzwert = 0.00
- Geänderte Cells werden weitergegeben

```
scores = 10x8 single matrix
    0.2088    0.0641    0.0693    0.2212 ...
    0.0010    0.0000    0.0566    0.6537
    0.1227    0.0750    0.0201    0.1506
    0.0490    0.0004    0.0620    0.5094
    0.0000    0.0000    0.0144    0.8797
    0.2180    0.0593    0.0704    0.2226
    0.0974    0.0120    0.0880    0.3566
    0.0000    0.0000    0.0240    0.8303
    0.0307    0.0014    0.0869    0.4623
    0.0000    0.0000    0.0009    0.9827
```



# Methodik – End-Zellen (Beispiel)

Index	Bild-Pfad [String]	Position X (Eingabebild) [px]	Position Y (Eingabebild) [px]	Durchmesser [px]	Wert [€]
1	"[...]/readCoins/img_01.jpg"	203	154	74	1.0
2	"[...]/readCoins/img_02.jpg"	451	85	35	0.1
3	"[...]/readCoins/img_03.jpg"	122	384	44	0.2
4	"[...]/readCoins/img_04.jpg"	130	706	62	0.5



# Methodik – Nachbearbeitung

- Zusammenzählen der Werte\*
- Einfärben der Münzen
  - Ggf. mit Filter
- Maskierung Anhand der in den Cells gespeicherten Informationen
  - Position, Radius

*\*In der Logik wird dies bei der Klassifizierung direkt mitberechnet, gehört aber trotzdem zur Nachbearbeitung.*

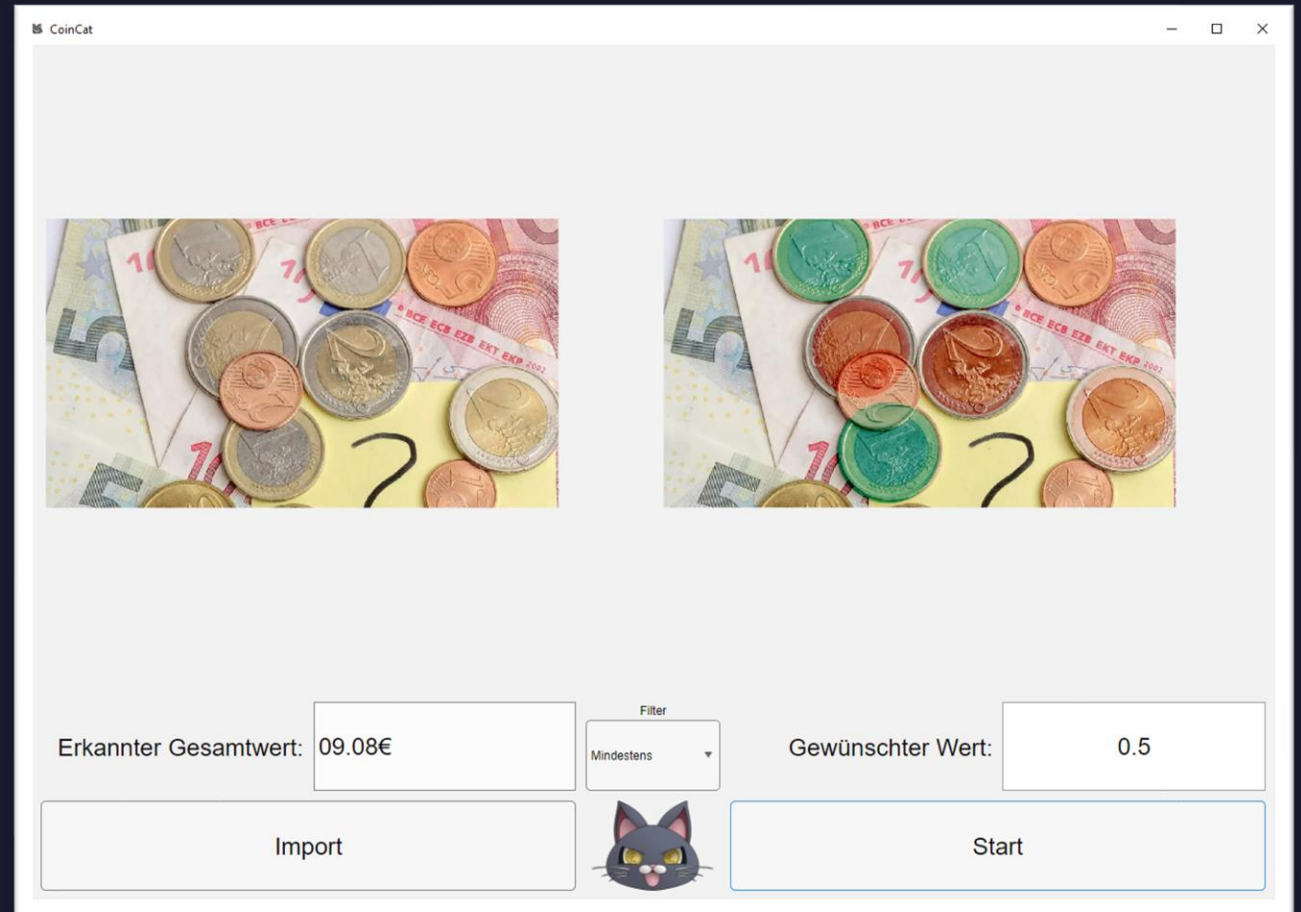
## Benutzte Methoden:

`imblend()` [Image Processing Toolbox]

`gcolorize()` [Image Manipulation Toolbox]

`replacepixels()` [Image Manipulation Toolbox]

`ndgrid()`

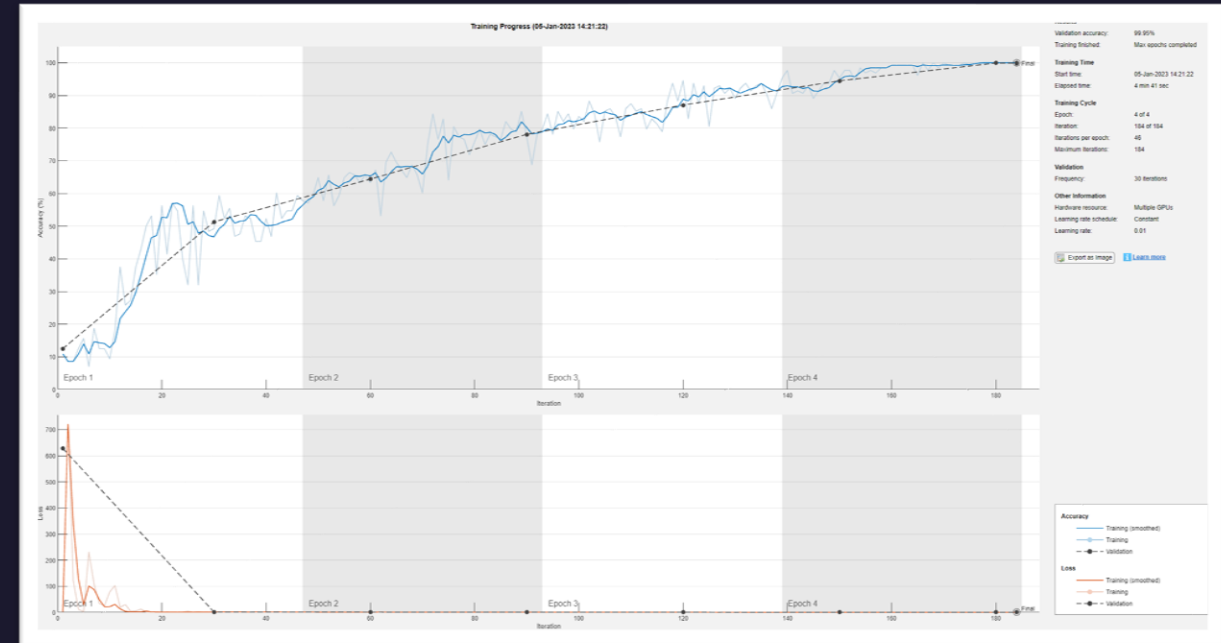




# Methodik – Training des neuronalen Netzwerks

Benutzte Methoden:  
*splitEachLabel()*  
*trainingOptions()* [Deep Learning Toolbox]  
*trainNetwork()* [Deep Learning Toolbox]

- SGDM-NN
- Klassifizierung auf 8 Labels
- Eingabegröße: 256x256px (Farbbilder)
- Verhältnis Trainings- zu Validierungsdaten:  
75% - 25%



*imageInputLayer* → *convolution2dLayer* → *batchNormalizationLayer* → *reluLayer* → *fullyConnectedLayer* → *softmaxLayer* → *classificationLayer*

# Evaluation

# Evaluation – Herausforderungen I

## I. Anfangs: Hough-Algorithmus funktioniert nicht richtig

- Potenzieller Lösungsansatz: Wechsel auf Python / opencv (verworfen)
- Tatsächlicher Lösungsansatz: Parameter angepasst

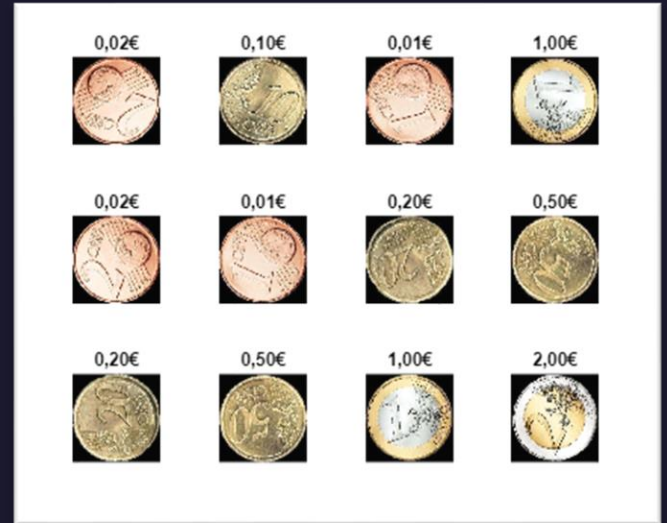




# Evaluation – Herausforderungen II

## 2. Einseitige Trainingsdaten

- Trainingsdaten werden von uns erhoben und durch Transformationen angepasst, um mehrere Trainingsbilder aus einem Referenzbild zu erzeugen
- Anfangs: Nur ein Referenzbild pro Münze => Hohe Genauigkeit bei den Trainings- und Validierungsdaten, aber geringe Genauigkeit beim tatsächlichen Einsatz
- Lösungsansatz: Mehr, unterschiedlichere Trainingsdaten



Validierungsdaten



Testdaten




# Evaluation – Herausforderungen III

## 3. Performance

- Trainieren eines Netzwerkes dauert lange
  - Grund: `trainNetwork(imds, layers, options)` wird auf einzelнем CPU-Kern berechnet
- Lösungsansatz: Nutzen der Parallel Computing Toolbox
  - Parallel Pooling (Parpooling) erlaubt Trainings auf bis zu 8 GPUs
    - Der Trainingsalgorithmus wäre für den Einsatz in einem kleineren Cluster geeignet: Potenziell auf Cluster der FH (Prof. Elsen) ausführbar
  - Zeitersparnis: ca. 57.8% pro Training (Ryzen 7 5800X vs. NVIDIA GeForce GTX 1060)

Training Time	
Start time:	03-Jan-2023 14:10:20
Elapsed time:	11 min 6 sec



Training Time	
Start time:	05-Jan-2023 14:21:22
Elapsed time:	4 min 41 sec



# Evaluation – Herausforderungen IV

## 4. Perspektiv-Fehler

- Lösungsansatz: Bilder sollen möglichst mit 0,2m Abstand direkt auf die Münzen zeigen

## 5. Transfer auf Mobilgeräte

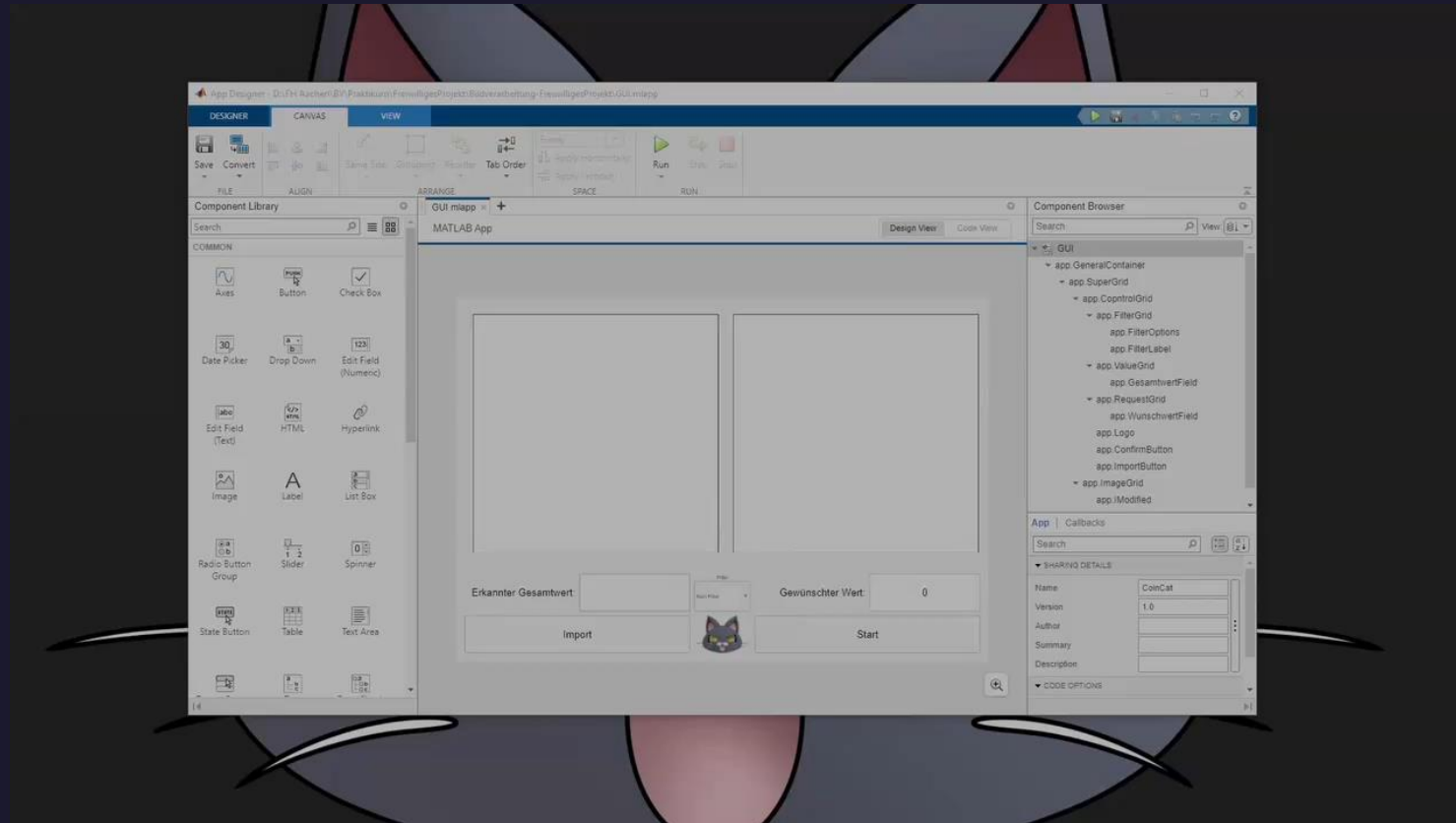
- Ganzes, trainiertes Modell auf Mobilgeräte bringen ist extrem inperformant
- Großer Speicheraufwand wegen des trainierten Modelles





# Ergebnis

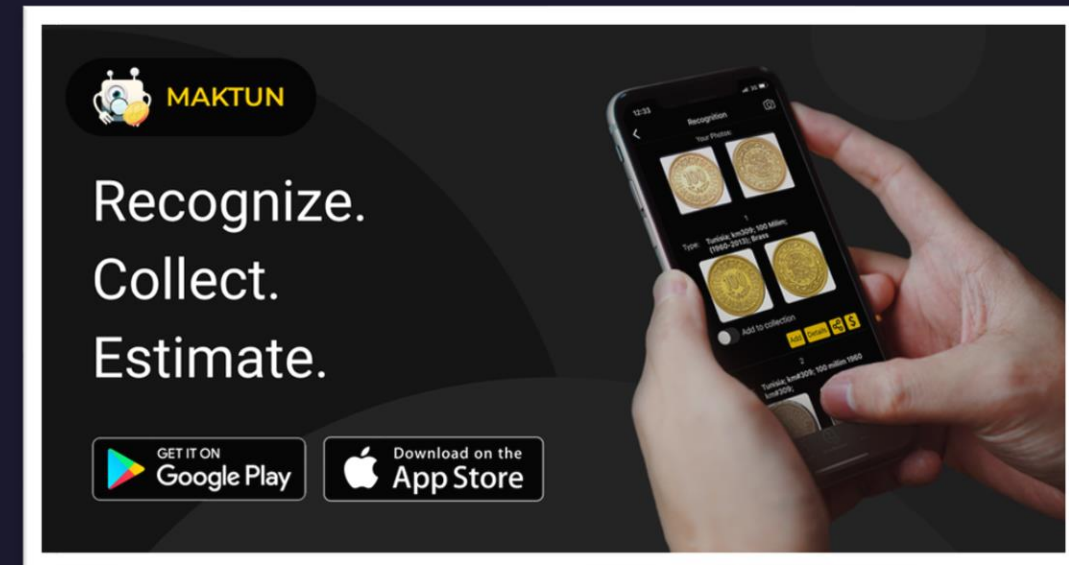
# Ergebnis



# Fazit

# Lessons learned

- Um ein anständiges neurales Netzwerk zu trainieren, sind **viele** Trainingsdaten nötig
- Matlab bietet viele Möglichkeiten, neurale Netzwerke zu gestalten
- Marktalternative: *Maktun*
  - Zugriff auf etwa 300.000 Münztypen und 120.000 Banknotentypen
  - Interne Katalogisierung
  - Andere Zielgruppe: Sammler
- Viele Kleinigkeiten können das Ergebnis beeinträchtigen:
  - Münzseite
    - Viele verschiedene Rückseiten
  - Zustand der Münze
    - Ist die Münze dreckig?
  - Licht und Reflektionen





# Weiterführende Ideen

# Weiterführende Ideen

- Einbindung in eine Mobil-App
  - Optimierung für mobile Endgeräte
- Perspektivische Verzerrung erlauben / Eingabetoleranz erhöhen
- Mehr Filter hinzufügen
  - z.B.: Alle Münzen mit einem Wert von mehr als 0,50€
- Mehr Währungen hinzufügen
- **Mehr Trainingsdaten hinzufügen**
- Scheinerkennung hinzufügen





# Zusammenfassung

Mit **CoinCat** können kleine Geldbeträge schnell zusammengezählt und gefiltert werden. Auch wenn die Software aktuell noch durch die Menge an verschiedenen Eingaben an einigen Stellen ins Schwanken kommen kann, bietet das von uns erstellte Codegerüst eine solide Basis, auf welcher aufgebaut werden kann.

# Vielen Dank für Ihre Aufmerksamkeit!

Für Fragen und Anmerkungen stehen wir Ihnen nun zur Verfügung.

*Tarik Glasmacher, Tim Bering*

