

## **Wochenbericht: Entwicklung eines Systems zur automatischen Prädatoren-Erkennung**

Berichtszeitraum: 12.11.2025 – 22.11.2025

Thema: Datenvorverarbeitung und Anforderungsanalyse

### **1. Zusammenfassung**

Folgende Aufgaben wurden durchgeführt.

#### **1.1. Einrichtung der Entwicklungsumgebung:**

**Aufsetzen einer virtuellen Umgebung in Jupyter** und Konfiguration der ML-Bibliotheken  
(TensorFlow usw.)

#### **1.2. Datenbereinigung:**

**Entwicklung eines Python-Skripts mit OpenCV**, das die Metadatenleiste am unteren Bildrand automatisch entfernt, um fehlerhafte Merkmalszuweisungen durch das Modell „ResNet50“ zu verhindern.

#### **1.3. Vermeidung von Data Leakage:**

**Implementierung eines Algorithmus zur Keyframe-Extraction (Best-Shot-Selection)**, der aus Serienaufnahmen automatisch das schärfste Einzelbild extrahiert und redundante Daten eliminiert.

#### **1.4. Anforderungsanalyse:**

Finalisierung der Klassifizierungsstrategie (Ausschluss des Austernfischers als Prädator) und der Systemziel durch direkte Rücksprache mit dem Projektpartner NABU (Herr Cimiottio)

## 2. Detaillierter Projektfortschritt

2.1. Jupyter Setup (18. November) : Einrichtung einer virtuellen Umgebung in Jupyter

2.2. Analyse und Vorverarbeitung der Bilddaten (19. November– 20. November)

### a) Problem A:

- Analyse: Die Bilder enthalten am unteren Rand einen schwarzen Balken mit Metadaten (Datum, Temperatur, Kamera-ID). Es besteht die Gefahr, dass das Modell „ResNet50“ diesen schwarzen Balken fälschlicherweise als Merkmal für Prädatoren lernt.
- Lösung: Implementierung eines Cropping-Algorithmus mittels OpenCV
- Begründung: OpenCV wurde gewählt, da es nativ auf numpy-Arrays basiert. Da TensorFlow/Keras ebenfalls Matrix-Formate benötigen, entfällt der Konvertierungsschritt, was die Pipeline effizienter macht.
- **Ergebnis: Der untere Bildbereich (ca. 12%) wird automatisch entfernt. Der Python-Code für dieses Cropping-Skript befindet sich im Anhang.**



Schaubild 1: Originalaufname der Wildkamera mit Metadatenleiste am unteren Rand



Schaubild 2: Bereinigte Aufnahme nach dem automatischen Zuschnitt der unteren 12%

**b) Problem B:**

- Analyse: Die Daten liegen als Serienaufnahmen vor. Die Verwendung fast identischer Bilder in Trainings- und Testdaten würde zu Data Leakage führen und die Validierungsergebnisse verfälschen (Overfitting)
- Lösung: Entwicklung eines Algorithmen zur Keyframe-Extraktion (Best-Shot-Selection)
- Algorithmus-Logik:
  - I. Gruppierung: Bilder mit einem Zeitabstand von < 10 Sekunden werden als ein Ereignis (Event) zusammengefasst.
  - II. Bewertung: Innerhalb eines Ereignis wird mittels Laplace-Varianz (Kantenerkennung) das Bild mit der höchsten Schärfe und signifikanten Änderungen ausgewählt.
  - III. Selektion: Nur das Beste Bild wird gespeichert, Redundanzen werden verworfen.
- **Test: Ein Testlauf mit dem Datensatz „Dachs“ (25 Objekte) ergab 23 relevante, nicht-redundaten Bilder Die vollständige Implementierung (Python-Code) dieses Algorithmus ist im Anhang beigefügt.**

### **2.3. Klärung fachlicher Anforderungen (Kommunikation mit Cimiottio, 19.11– 21.11)**

Missverständnisse bei der Labeling-Strategie und den Systemzile zu vermeiden, wurde direkter Kontakt mit dem Projektpartner NABU (Herr Cimiottio) Aufgenommen. Nachfolgend sind die zentralen Punkte der Korrespondenz im Wortlaut dokumentiert:

#### **Frage 1 (Ich):**

Ich habe die 23 Ordner mit den Prädator-Fotos durchgesehen. Dabei ist mir aufgefallen, dass darin auch "Austernfischer" enthalten sind. Zählt diese Vogel zu den "Prädatoren" (Feinden), die das System melden soll? Falls ja, welche Tierarten greift sie an? Da ich über kein ökologisches Fachwissen verfüge, wäre diese Information für mein Verständnis sehr hilfreich.

#### **Antwort 1 (Cimiottio):**

Der Austernfischer tritt selten als Prädator (Feind) bei kleineren Vogelarten wie Sandregenpfeifer, Seeregenpfeifer und Zwergseeschwalbe auf (das heißt, Austernfischer fressen deren Eier). Aber in den meisten Fällen ist er das Opfer (das heißt, andere Tiere wie Füchse fressen die Eier der Austernfischer). Ich weiß nicht, ob man das so einstellen kann, dass der Austernfischer nur bei Nestern von anderen Vogelarten (= nicht Austernfischer) als möglicher Prädator erkannt wird. Falls das zu kompliziert ist, können Sie den Austernfischer aber auch als Feind weglassen. Denn er frisst nur selten Eier, sondern normalerweise Würmer und Muscheln.

#### **Frage 2 (Ich):**

Die Daten liegen als Serienaufnahmen (mehrere Bilder pro Sekunde) vor. Soll die fertige Software ganze Ordner mit solchen Serienaufnahmen automatisch einlesen und auswerten?

#### **Antwort 3 (Cimiottio):**

Ja, das wäre sehr gut. Das würde uns an unserem Institut helfen, wenn wir die Serienaufnahmen mit der von Ihnen entwickelten KI durchschauen lassen könnten. Bilder mit Prädatoren-Tierarten müssten dann markiert oder Ähnliches werden.

**Frage 3 (Ich):**

Ist es Ihre Absicht, die Software einzusetzen, um Bilder von Kamerafallen (Wildkameras) einzulesen und darin automatisch Prädatoren zu identifizieren? Falls ja, gibt es spezielle Anforderungen an die Benutzung, die berücksichtigt werden sollen?

**Antwort 3 (Cimiottio):**

Ja, genau. Das wäre sehr gut und unser Ziel. Außerdem möchten wir zusammen mit den Professoren Frochte und Schillberg später einen Apparat entwickeln, der in Echtzeit von der KI entdeckte Prädatoren vertreibt. Erst einmal geht es aber um die Auswertung vorhandener Serienaufnahmen aus Wildkameras.

### 3. Ausblick und nächste Schritte

Für die kommende Woche sind folgende Aufgaben geplant:

#### 3.1. Massenverarbeitung:

Anwendung der entwickelten Preprocessing-Skripte (Cropping + Keyframe Extraction) auf den gesamten NABU-Datensatz.

#### 3.2. Datenanalyse:

Überprüfung der Klassenbalance nach der Bereinigung, um Bias zu vermeiden.

#### 3.3. Modellierung:

Beginn der Implementierung des ersten ML-Modells basierend auf den bereinigten Daten.

**Anhang 1: Python-Code zur automatischen Entfernung der Metadatenleiste**

**Anhang 2: Python-Code zur Keyframe-Extraktion aus Serienaufnahmen (Best-Shot-Selection)**

```
1 # Code zum Zuschneiden der Metadaten (Bilder)
2
3 import os
4 import cv2
5
6 # 1. Pfad zum Ordner mit den Originalbildern (Raw Data)
7 input_folder = '/home/surkgoun/Bilder/input_images'
8
9 # 2. Pfad zum Speicherordner für die bearbeiteten
10 Bilder
11 output_folder = '/home/surkgoun/Bilder/output_images'
12
13 # 3. Anteil des Beschnitts am unteren Rand (crop
14 Percentage)
15 # 0.12 bedeutet, dass die unteren 12% des Bildes
16 entfernt werden.
17 crop_percentage = 0.12
18
19 # 4. Liste aller Bilddateien im Verzeichnis abrufen
20 image_files = []
21 for file in os.listdir(input_folder):
22     if file.lower().endswith('.png', '.jpg', '.jpeg'):
23         image_files.append(file)
24
25 # 5.
26 print(f"Starte die Verarbeitung von {len(image_files)}
27 Bildern")
28 print(f"Die unteren {crop_percentage*100}% werden
29 entfernt.")
30
31 count = 0
32
33 for filename in image_files:
34     # 1) Pfade generieren
35     input_path = os.path.join(input_folder, filename)
36     output_path = os.path.join(output_folder, filename)
37
38     # 2) Bild laden und schneiden
39     image = cv2.imread(input_path)
40     height, width, channels = image.shape
41     crop_width = int(width * crop_percentage)
42     crop_height = int(height * crop_percentage)
43     crop_x = width // 2 - crop_width // 2
44     crop_y = height // 2 - crop_height // 2
45     cropped_image = image[crop_y:crop_y+crop_height, crop_x:crop_x+crop_width]
46
47     # 3) Bild speichern
48     cv2.imwrite(output_path, cropped_image)
49
50     count += 1
51
52 # 4) Fazit
53 print(f"Verarbeitung abgeschlossen. {count} Bilder wurden
54 bearbeitet.")
```

```
34      # 2) Bild einlesen (OpenCV liest Bilder als NumPy-  
35      # Arrays)  
36      img = cv2.imread(input_path) # Zurückgeben: Höhe,  
37      Breite, 3 (Blue, Green, Red) oder None  
38  
39      if img is None:  
40          print(f" Fehler: Datei {filename} konnte nicht  
41          gelesen werden.")  
42          continue  
43  
44      # 3) Bilddimensionen abrufen (Höhe, Breite)  
45      height, width, _ = img.shape  
46  
47      # 4) Neue Höhe berechnen (Originalhöhe -  
48      # abzuschneidend Bereich)  
49      new_height = int(height * (1 - crop_percentage))  
50  
51      # 5) Das Zuschneiden  
52      # Zur neuen Höhe, Breite  
53      cropped_img = img[0:new_height, 0:width]  
54  
55      # 6) Bearbeitetes Bild speichern  
56      cv2.imwrite(output_path, cropped_img)  
57      count += 1  
58  
59      # 7) Fortschrittsanzeige  
60      if count % 10 == 0: # Wenn Bilder 10 sind  
61          print(f" Verarbeitete Bilder: {count}")  
62  
63      print("_" * 30)  
64      print(f" Fertig. Insgesamt {count} Bilder wurden im  
65          Ordner '{output_folder}' gespeichert.")  
66
```

```
1 # Auswahl des besten Bildes aus Serienaufnahmen (Keyframe Extraction)
2
3 import os
4 import shutil
5
6 import cv2
7 from PIL import Image
8 from datetime import datetime
9
10 # 1. Pfad zum Ordner mit den Originalbildern
11 input_folder = '/home/surkgoun/Bilder/input_images'
12
13 # 2. Pfad zum Speicherordner für die ausgewählten besten Bilder
14 output_folder = '/home/surkgoun/Bilder/output_images'
15
16 # 3. Zeitintervall für Serienaufnahmen (in Sekunden)
17 # Wenn der Zeitabstand zwischen zwei Bildern kleiner als dieser Wert ist,
18 # werden sie als ein zusammengehörige Serie betrachtet
19 .  

20 burst_gap_seconds = 10 # 10 Sekunden
21
22 # =====
23 # Liest das Aufnahmedatum aus den Exif-Metadaten des Bildes aus.
24 def get_date(path):
25     try:
26         return Image.open(path).getexif()[36867]
27     except Exception:
28         return None
29
30 # Berechnet die Schärfe des Bildes mithilfe der Laplace-Varianz
31 # Ein höherer Wert bedeutet ein schärferes Bild
32 def get_sharpness(img_path):
33     try:
34         img = cv2.imread(img_path)
35         if img is None:
```

```

36         return 0
37
38     # Umwandlung in Graustufen für die
39     # Kantenanalyse
40     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
41
42     # Laplace-Varianz berechnen
43     return cv2.Laplacian(gray, cv2.CV_64F).var()
44 except Exception:
45     return 0
46
47 # =====
48 # 4. Alle Bilddateien auflisten
49 image_files = [file for file in os.listdir(
    input_folder) if file.lower().endswith('.png', '.jpg',
    '.jpeg')]
50
51 print(f"Starte die Analyse von {len(image_files)}"
    "Bildern zur Auswahl der besten Aufnahmen.")
52
53 # 5. Liste für Dateiinformationen erstellen
54 file_data = []
55
56 for file in image_files:
57     input_path = os.path.join(input_folder, file)
58     time_str = get_date(input_path)
59
60     if time_str:
61         dt = datetime.strptime(time_str, '%Y:%m:%d %H
            :%M:%S')
62         file_data.append({'name': file, 'path':
            input_path, 'time': dt})
63     else:
64         timestamp = os.path.getmtime(input_path) #
            Wenn keine Exif-Daten vorhanden sind
65         dt = datetime.fromtimestamp(timestamp)
66         file_data.append({'name': file, 'path':
            input_path, 'time': dt})
67
68 # 6. Chronologisch sortieren (Sehr wichtig für die

```

```

68 Gruppierung)
69 file_data.sort(key=lambda x: x['time'])
70
71 # 7. Gruppierung und Auswahl
72 if not file_data:
73     print("Keine Bilder im Ordner gefunden.")
74     exit()
75
76 current_series = [file_data[0]]
77 selected_count = 0 # Anzahl der ausgewählten besten
    Bilder
78
79 print("Verarbeitung läuft...")
80
81 # 8. Ab dem zweiten Bild vergleichen
82 for i in range(1, len(file_data)):
83     prev_img = file_data[i-1]
84     cur_img = file_data[i]
85
86     # Zeitdifferenz berechnen (in Sekunden)
87     time_diff = (cur_img['time'] - prev_img['time']).total_seconds()
88
89     if time_diff <= burst_gap_seconds:
90         current_series.append(cur_img) # Zur
            aktuellen Gruppen hinzufügen (gehört zur gleichen
            Serie)
91     else: # Die Serie ist beendet. Jetzt, das beste
            Bild aus der vorherigen Gruppe wählen
92         best_img = None
93         max_score = -1
94         # Der Schärfewert (Laplace-Varianz) ist immer
            positiv (>=0)
95         # Da -1 kleiner ist als jeder mögliche
            Schärfewert,
96         # gewinnt das erste Bild sofort und als
            aktuelles Maximum gespeichert.
97
98         # Schärfe vergleichen
99         for img_info in current_series:
100             score = get_sharpness(img_info['path'])

```

```

101         if score > max_score:
102             max_score = score
103             best_img = img_info
104
105             # Das Gewinner-Bild kopieren
106             if best_img:
107                 shutil.copy2(best_img['path'], os.path.
108                             join(output_folder, best_img['name']))
108                 # shutil.copy2: Kopiert das beste Bild in
109                 # den "output"-Ordner und behält die Metadaten (z.B.
110                 # Zeitstempel)
111                 selected_count += 1
110
111             # Neue Gruppe starten
112             current_series = [cur_img]
113
114 # 9. The Last Piece
115 # Verarbeitung der letzten Gruppe nach dem Ende der
116 # Schleife
116 # Da es nach dem letzten Bild kein nächstes Bild gibt
117 #
117 # muss die letzte Gruppe manuell nach der Schleife
117 # gespeichert werden.
118 if current_series:
119     best_img = None
120     max_score = -1
121     for img_info in current_series:
122         score = get_sharpness(img_info['path'])
123         if score > max_score:
124             max_score = score
125             best_img = img_info
126
127     if best_img:
128         shutil.copy2(best_img['path'], os.path.join(
128             output_folder, best_img['name']))
129         selected_count += 1
130
131
132 print("-" * 30)
133 print("Fertig.")
134 print(f"Ursprüngliche Anzahl: {len(image_files)}")

```

```
134 Bilder.")  
135 print(f"Beste Bilder: {selected_count} Bilder.")  
136 print(f"Die Bilder wurden im Ordner '{output_folder}'  
' gespeichert.")
```