künstliche intelligenz
kognitive systeme

HOCHSCHULE
ANSBACH

3D-Anomaly

Projektarbeit von Tobias Saur, Niclas Hart und Jakob Kramer im Rahmen der Zweitjahresprojekte im Sommersemester 2024

Standard-Schraube

Zylinder oben

Zylinder unten

Zylinder ganz

Zylinder

Inhaltsverzeichnis

1. Einleitung

Unser Projekt "3D-Anomaly" hat das Ziel die Möglichkeiten von Künstlicher Intelligenz in der industriellen Qualitätssicherung aufzuzeigen. Hierbei werden insbesondere Bildverarbeitungstechniken und maschinelles Lernen eingesetzt, um fehlerhafte Produkte anhand von Kamerabildern in Echtzeit zu identifizieren. Dies kann zu einer Effizienzsteigerung und Kostenreduktion in der Fertigung führen.

In der modernen industriellen Produktion spielt die Qualitätssicherung eine entscheidende Rolle, um den hohen Anforderungen an die Produktqualität gerecht zu werden. Traditionelle Methoden der Fehlererkennung durch visuelle Inspektionen der Mitarbeiter stoßen jedoch oft an ihre Grenzen. Hier setzen fortschrittliche Techniken wie das Template Matching und Convolutional Neural Networks (CNNs) an, die durch ihre Fähigkeit zur Mustererkennung und -klassifizierung neue Möglichkeiten für die automatisierte Qualitätskontrolle bieten.

Im Rahmen des Projekts wurde ein Bilddatensatz von 3D-gedruckten Bolzen aus verschiedenen Perspektiven (seitlich und von oben) erstellt und mittels Data Augmentation erweitert. Es wurden drei Ansätze zur Anomalieerkennung getestet: YOLO (You Only Look Once), ein vortrainiertes CNN, das sich durch hohe Geschwindigkeit und Genauigkeit auszeichnet und besonders in der Industrie weit verbreitet ist. Template Matching ist ideal für die Erkennung unbekannter Fehler, da es kein vortrainiertes Modell benötigt. Außerdem wurde auch ein CNN selbst aufgebaut, trainiert und hinsichtlich seiner Performance evaluiert.

Die Ergebnisse zeigen, dass alle drei Ansätze wertvolle Beiträge zur kamerabasierten Anomalieerkennung und somit zur Qualitätssicherung in der Industrie leisten können.

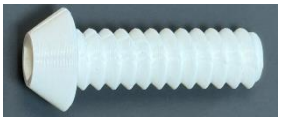
2. Erstellung des Bilddatensatzes

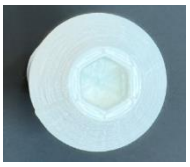
Zu Beginn des Projekts gab es kein vorgegebenes 3D-Druck Modell, das für die Bearbeitung des Projekts verwendet werden sollte. Nach kurzer Recherche fiel die Wahl auf ein fertiges Modell eines 25x8 Bolzens von Thingiverse (https://www.thingiverse.com/thing:193647/files). Aufgrund der potenziellen Nutzbarkeit im industriellen Umfeld wurde dieses Modell gewählt.

Anschließend wurden dieser Bolzen zum ersten Mal 3D-gedruckt. Dabei wurde ersichtlich, dass es bei dieser Bauteilkomplexität nicht automatisch zu Fehlern beim Druck kommen würde. Aus diesem Grund wurden die Modelle mittels Tinkercad künstlich manipuliert, um Fehler zu erzeugen, welche die Nutzbarkeit des Bauteils erheblich einschränken. Dabei wurde zum Beispiel das Gewinde teilweise oder gänzlich entfernt oder der Ansatz für den Sechskant-Schlüssel entfernt.

Um tatsächlich eine vollständige Überprüfung der Bauteile zu gewährleisten, sind Bildaufnahmen aus der seitlichen Ansicht (Prüfung des Gewindes) und von oben (Schraubansatz) notwendig. In diesem Zug wurden folgende Fehlerklassen definiert und Bilder in verschiedenen Positionen, mit verschiedenen Hintergründen und unterschiedlicher Beleuchtung aufgenommen:

1. Standard-Schraube (ohne Fehler): 104 Bilder seitlich / 40 Bilder oben





2. Schraube-Zylinder_ganz: 63 Bilder seitlich / 18 Bilder oben

3. Schraube-Zylinder_oben: 59 Bilder seitlich / 25 Bilder oben


4. Schraube-Zylinder_unten: 70 Bilder seitlich / 24 Bilder oben


5. Schraube-ohne_Schraubansatz: 46 Bilder seitlich / 22 Bilder oben


6. Zylinder: 53 Bilder seitlich / 20 Bilder oben


4. Implementierung und Evaluierung der 3 Modellansätze

Im Folgenden wird die Implementierung der drei ausgewählten Modellansätze (Template Matching, YOLO, CNN) genauer erläutert. Außerdem werden die drei Modelle hinsichtlich ihrer Performance auf dem Datensatz, sowie weiterer Gesichtspunkte evaluiert.

4.1 CNN

Beim ersten Modellansatz handelt es sich um ein Convolutional-Neural-Network, das selbständig aufgebaut wurde (Schichten) und das vollständig auf dem erstellten Datensatz mit den 3D-Druck Bildern trainiert, validiert und getestet wurde.

4.1.1 Implementierung

In diesem Abschnitt wird die Funktionsweise für das Modell der seitlichen Ansicht genauer erklärt. Das Modell, das die obere Ansicht betrachtet funktioniert analog und wird deshalb hier nicht genauer betrachtet.

Zu Beginn des Skriptes werden zunächst die Hyperparameter eingestellt, die für den Trainingsdurchlauf beziehungsweise den Programmablauf verwendet werden sollen.

```
24  # Parameter:
25
26  image_size = (224, 224)
27
28  use_augmentation = True
29
30  num_augmented_images_per_original = 80
31
32  epoch = 30
33
34  batch_size = 32
35
36  num_classes = len(os.listdir(image_directory))
```

Die Bildgröße wurde final auf 224x224 Pixel eingestellt, da dies sowohl eine gute Genauigkeit, aber auch akzeptablen Speicher- und Zeitbedarf für das Training bedeutete. Außerdem kann eingestellt werden, ob der Datensatz mittels Data Augmentation künstlich erweitert werden soll und wie viele augmentierte Bilder pro Originalbild erstellt werden sollen. Außerdem müssen die Epochenzahl und die Batch-Size festgelegt werden.

Als Nächstes erfolgt das Laden der Bilder aus dem angegebenen Ordner. Die Labels werden dabei anhand des Ordnernamens erstellt, in dem die Bilder enthalten sind. Außerdem erfolgt die Unterteilung in Testdaten (58%) und Validierungs- beziehungsweise Testdaten (jeweils 21%). Die augmentierten Bilder werden zum Trainingsdatensatz hinzugefügt. Dabei ist hervorzuheben, dass für diesen Ansatz zwischen den tatsächlichen Fehlerklassen und nicht nur zwischen Anomalie-Gutteil unterschieden wurde. Bei einem Versuch, bei dem nur die binäre Unterscheidung trainiert wurde, konvergierte das Netzwerk nicht. Dieser Ansatz wurde für das selbst trainierte CNN deshalb auch nicht weiter fortgeführt.

Jetzt kann das Convolutional-Neural-Network trainiert werden. Dabei wurden zum Vergleich zwei verschiedene Netzwerke angelegt:

Ein Mini-CNN, das nur aus einer einzigen Convolutional-Schicht, besteht und deshalb nur sehr einfache Muster in den Bildern erkennen kann.

```
38 def train_minicnn(epoch, batch_size, image_size, num_classes, train_images, train_labels, val_images, val_labels):
39     # define mini cnn
40     mini_cnn_model = models.Sequential([
41         layers.Conv2D(32, (3, 3), activation='relu',
42                       padding='same', input_shape=(image_size[0], image_size[1], 3)),
43         layers.MaxPooling2D(2, 2),
44         layers.Flatten(),
45         layers.Dense(num_classes, activation='softmax')])
46 
47     # compile mini cnn
48     mini_cnn_model.compile(optimizer='adam',
49                            loss=tf.keras.losses.CategoricalCrossentropy(),
50                            metrics=['accuracy'])
51 
52     callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=15, restore_best_weights=True)
53     reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', factor=0.2,
54                   patience=10, min_lr=0.001)
```

Außerdem ein weiteres, tieferes CNN mit mehreren sich abwechselnden Convolutional-Schichten und Max-Pooling Schichten. Dieses ist in der Lage auch komplexere Muster in den Bildern zu erkennen.

```python
def train_cnn(epoch, batch_size, image_size, num_classes, train_images, train_labels, val_images, val_labels):
    # define cnn
    cnn_model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu',
                      padding='same', input_shape=(image_size[0], image_size[1], 3)),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D(2, 2),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D(2, 2),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(num_classes, activation='softmax')])

    # compile cnn
    cnn_model.compile(optimizer='adam',
                      loss=tf.keras.losses.CategoricalCrossentropy(),
                      metrics=['accuracy'])

    callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=15, restore_best_weights=True)
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', factor=0.2,
                              patience=10, min_lr=0.001)
```

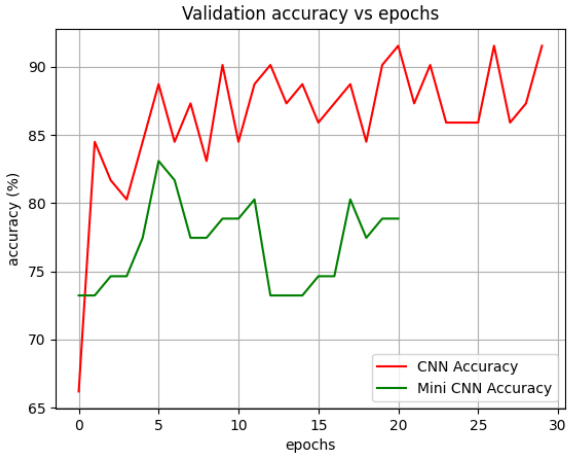
Als Optimierer wurde bei beiden Netzwerken Adam mit Categorical Crossentropy als Loss-Funktion und Accuracy als Metrik verwendet. Außerdem wurden Callbacks für das Training verwendet. EarlyStopping sorgt dafür, dass das Training vorzeitig gestoppt wird, falls sich die Validation-Accuracy über 15 Epochen nicht verbessert. ReduceLROnPlateau verringert die Learning-Rate, falls sich die Validation-Accuracy über 10 Epochen nicht verbessert und kann helfen tiefer in Minima der Fehlerfunktion abzusteigen.

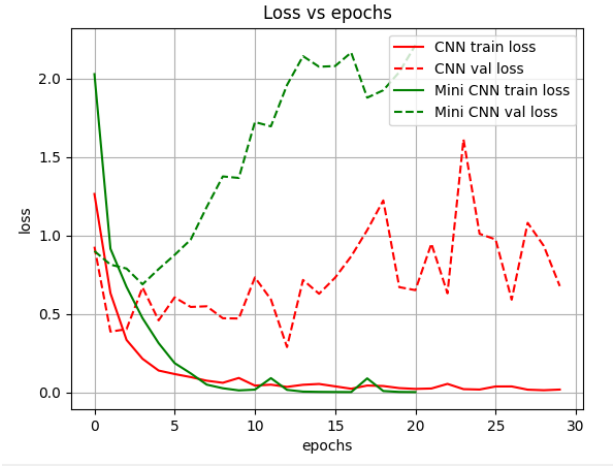
Zuletzt werden die beiden Modelle auf dem Testdatensatz angewendet und die Vorhersage für eine der Klassen getroffen.

```python
# Vorhersage auf Testdaten
cnn_test_probs = cnn_model.predict(X_test)

mini_cnn_test_probs = mini_cnn_model.predict(X_test)
```

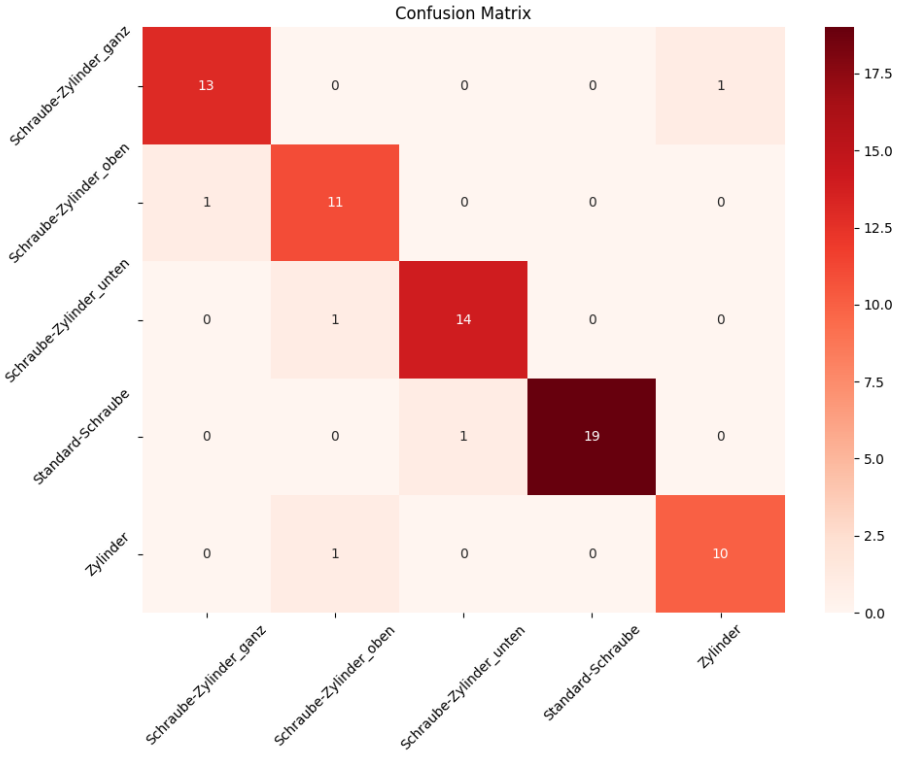
4.1.2 Auswertung

Insgesamt konnten für beide Perspektiven gute Ergebnisse für die Erkennung der Fehlerklassen erreicht werden. Im Rahmen dieses Berichts wird sich hier erneut auf die seitliche Ansicht beschränkt. Die Ergebnisse der oberen Ansicht können direkt im Gitlab-Projekt angesehen werden.





Insgesamt fällt auf, dass das Training in beiden Fällen sich bereits nach einer sehr kurzen Anzahl an Epochen an die Trainingsdaten angepasst hat. Die Genauigkeit schwankt während des Trainings stark und verbessert sich auch nach deutlich mehr als 30 Epochen nicht. Auffallend ist auch die Entwicklung des Loss auf den Validierungsdaten. Bereits nach ca. drei Epochen steigt der Loss sowohl beim Mini-CNN (stark) als auch beim tieferen CNN (weniger stark), was zwar deutlich auf eine Überanpassung des Netzwerkes (Overfitting) hindeutet, die Genauigkeit auf den Test- und Validierungsdaten aber nicht wirklich negativ beeinflusst hat.



Auch die Confusion-Matrix (hier für das tiefere CNN) bestätigt ein gutes Ergebnis. Die auf der y-Achse dargestellten wahren Klassen stimmen bis auf vereinzelte Fehlklassifikationen mit den vorhergesagten Klassen (x-Achse) überein. Positiv ist des Weiteren, dass deutliche Fehlerbilder wie der Zylinder nicht mit der Standard-Schraube verwechselt wurden.

Insgesamt erreicht das Mini-CNN mit nur einer Convolutional-Schicht erreicht eine Genauigkeit von 92% auf die Unterscheidung zwischen den fünf einzelnen Fehlerklassen und eine Genauigkeit von 94% auf die zusammengefasste binäre Entscheidung zwischen Anomalie und Gutteil.

Das tiefere CNN schneidet mit einer Genauigkeit von 93% bei der Unterscheidung der fünf Fehlerklassen und sogar ca. 99% bei der binären Klassifizierung nochmal deutlich besser ab.

4.2 YOLO

Bei diesem Modellansatz handelt es sich um ein YOLO-Network, das auf den COCO-Datensatz vortrainiert wurde und anschließend auf unseren Datensatz nachtrainiert wurde.

4.2.1 Implementierung

In diesem Abschnitt wird ebenfalls „nur“ die Funktionsweise des Modells für die seitliche Ansicht betrachtet, da das Modell für die obere Ansicht analog funktioniert.

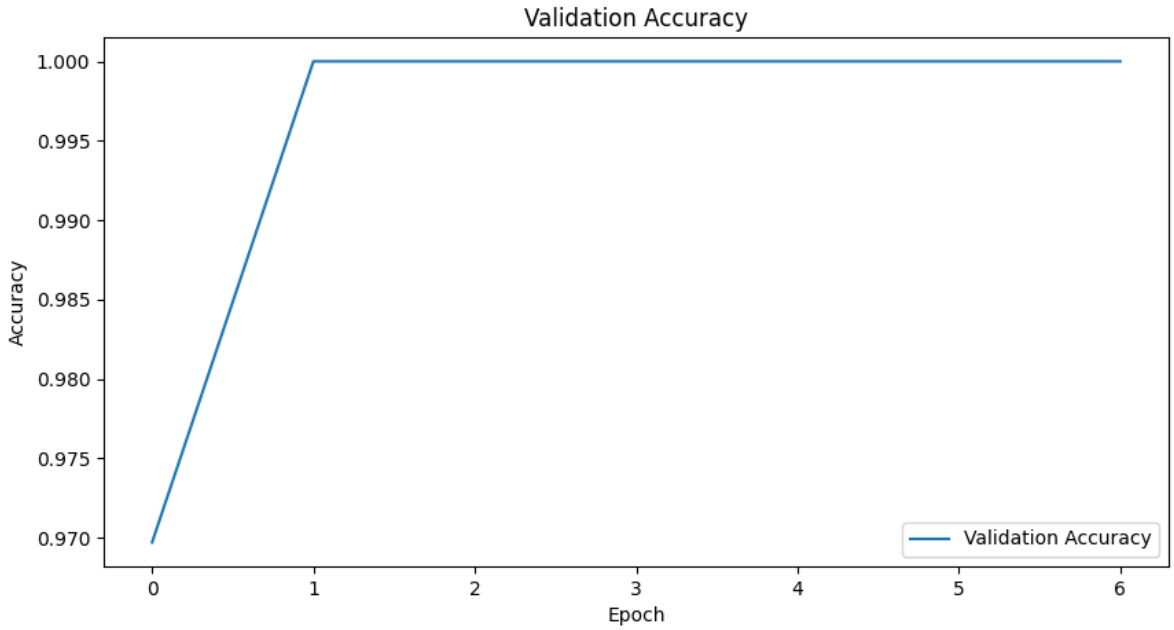
Zu Beginn des Skriptes wird die „train“-Funktion aufgerufen. In dieser wird das vortrainierte Modell geladen. Die Wahl der Hyperparameter erfolgt teils automatisch und teils selbst. Die Performance des Modells war mit den voreingestellten Parametern sehr gut und es wurde nur leicht nachjustiert. Die selbst gewählte Bildgröße wurde auf 128x128 Pixel eingestellt, da sich dadurch die Performance schneller gut entwickelt hat. Vorab werden die Bilder unterteilt, in Trainingsdaten (80% und künstlich erweitert) und Validierungs- beziehungsweise Testdaten (jeweils 10%). Anschließend werden die Bilder aus den entsprechenden Ordnern geladen, das Training wird mit 50 Epochen

gestartet und durch das implementierte EarlyStopping vorzeitig gestoppt, sollte sich die Validation-Accuracy über 5 Epochen nicht verbessern.

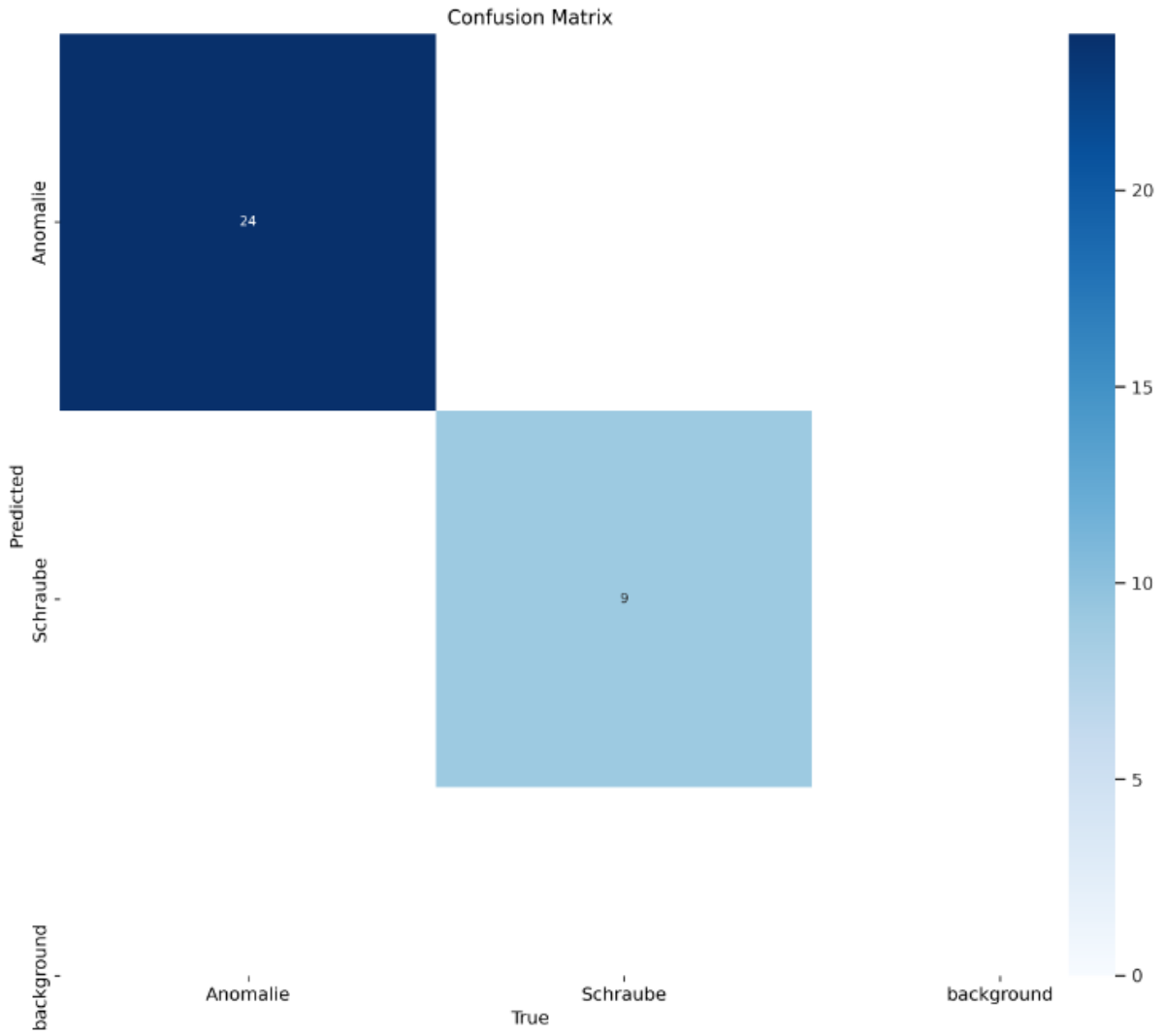
Zur Vorhersage der Testbilder wird das Modell mit den besten Gewichten verwendet. Der Ordner mit den Bildern wird in einem Durchlauf getestet, die wahren und vorhergesagten Klassen, mit welcher Wahrscheinlichkeit diese vorhergesagt wurden und welche Gesamtgenauigkeit beim Testen vorlag, wird ausgegeben und im Ordner „results“ abgespeichert.

4.2.2 Auswertung

Insgesamt konnten für beide Perspektiven gute sehr gute Ergebnisse für die Erkennung Fehler/Gutteil erreicht werden. Auch hier findet die Analyse nur für die Ergebnisse der seitlichen Ansicht statt.



Es fällt auch, dass das Training sich bereits nach einer Epoche an die Trainingsdaten angepasst hat. Die Genauigkeit steigt auf 100% und verändert sich auch nach dem Training über weitere Epochen nicht und so wird das Training automatisch durch das EarlyStopping beendet.



Auch die Confusion-Matrix bestätigt das gute Ergebnis. Die Klassen werden vollständig richtig vorhergesagt. Insgesamt kann man sagen, dass das Modell von Beginn an sehr gut performt und in der Komplexität noch deutlich belastbarer ist.

4.3 Template Matching

Beim dritten Modellansatz handelt es sich um eine Implementierung, die auf der OpenCV-Bibliothek für Python beruht. Template Matching ist ein Bildverarbeitungsverfahren, bei dem eine Template innerhalb eines Testbildes gesucht wird, um festzustellen, ob und wo es übereinstimmende Regionen gibt.

4.3.1 Implementierung

In diesem Abschnitt wird die allgemeine Implementierung des Template-Matching Ansatzes erläutert.

Zu Beginn wird die Template-Matching Funktion für die spätere Anwendung initiiert, welche zuerst das übergebene Testbild und Templatebild in ein Graustufenbild umwandelt, um so eine schnellere Berechnung des Algorithmus zu ermöglichen.

Folglich werden Mindestmaße für das Templatebild und ein Schwellwert für die Übereinstimmung festgelegt.

Die Funktion durchläuft daraufhin verschiedene Skalierungen des Templatebildes, beginnend bei 20% bis hin zu 100% der Originalgröße des Bildes. Wenn das skalierte Bild hierbei nun größer wird als das Testbild, wird die Schleife abgebrochen, da keine sinnvolle Übereinstimmung mehr möglich ist.

Nun wird die Template-Matching Funktion der OpenCV-Bibliothek angewendet, um das tatsächliche Matching durchzuführen. Das Ergebnis ist eine Matrix, die die Übereinstimmungen enthält. Diese wird der Variable „result“ zugewiesen.

Als letzten Schritt werden die Positionen im Ergebnis gesucht, an denen der Übereinstimmungswert den angegebenen Schwellwert überschreitet. Außerdem wird für jede gefundene Übereinstimmung überprüft, ob die Größe des Templates die zu Beginn festgelegten Mindestmaße unterschreitet. Wenn eine gültige Übereinstimmung gefunden wird, gibt die Funktion den Rückgabewert „True“ zurück.

```python
# Function to perform template matching
def template_matching(test_image, template):
    # Convert the images to grayscale
    test_image_gray = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY)
    template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

    min_width = 30
    min_height = 30
    # Set a threshold
    threshold = 0.782

    # Perform template matching in multiple scales
    for scale in np.linspace(0.2, 1.0, 20)[::-1]:
        # Resize the template according to the scale
        resized_template = cv2.resize(template_gray, (int(template_gray.shape[1] * scale), int(template_gray.shape[0] * scale)))

        # If the resized template is larger than the test image, break the loop
        if resized_template.shape[0] > test_image_gray.shape[0] or resized_template.shape[1] > test_image_gray.shape[1]:
            break

        # Perform template matching
        result = cv2.matchTemplate(test_image_gray, resized_template, cv2.TM_CCOEFF_NORMED)

        # Find the locations where the match score is above the threshold
        loc = np.where(result >= threshold)

        # Additional validation for detected matches
        for pt in zip(*loc[::-1]):
            match_width, match_height = resized_template.shape[::-1]
            if match_width < min_width or match_height < min_height:  # min_width und min_height müssen definiert werden
                continue

            return True

    # If no match was found, return False
    return False
```

Als weiteren wichtigen Schritt wurde eine Entfernung des Hintergrundes der übergebenen Bilder durchgeführt, da dieser zu fehlerhaften Klassifizierungen der Bilder geführt hat. Es werden eine Maske sowie Hintergrund- und Vordergrundmodelle initialisiert. Die Maske hat die gleiche Größe wie das Bild, während die Modelle für den GrabCut-Algorithmus erforderlich sind.

Wenn das Rechteck zur Bestimmung des zu betrachtenden Bereichs zu Beginn des Programms noch nicht ausgewählt wurde, wird die selectROI Funktion verwendet, um ein Rechteck manuell im Bild auszuwählen, auf dem die Überprüfung stattfinden soll. Das ausgewählte Rechteck wird in der globalen Variable „selected_rect“ gespeichert und das Auswahlfenster wird geschlossen.

Der GrabCut-Algorithmus wird auf das Bild angewendet, wobei die zuvor definierte Maske und die Modelle verwendet werden. Das ausgewählte Rechteck wird als initiale Maske für den Algorithmus verwendet.

Die Maske wird in mehreren Iterationen verfeinert, indem sie abwechselnd erweitert und wieder verkleinert wird. Dies hilft, die Kanten der Maske zu glätten und kleine Fehler zu korrigieren.

Eine finale Maske wird erstellt, bei der die Hintergrundpixel auf 0 gesetzt werden und die Vordergrundpixel auf den Wert 1 gesetzt. Diese Maske wird dann auf das Bild angewendet, um den Hintergrund zu entfernen.

```
def remove_background(image, refine_iterations=5):
    global selected_rect  # Verwenden der globalen Variable

    mask = np.zeros(image.shape[:2], np.uint8)
    bgdModel = np.zeros((1,65),np.float64)
    fgdModel = np.zeros((1,65),np.float64)

    # Überprüfen, ob das Rechteck bereits ausgewählt wurde
    if selected_rect is None:
        # Rechteck auswählen und in der globalen Variable speichern
        selected_rect = cv2.selectROI("Image", image, False, False)
        cv2.destroyWindow("Image")  # Schließt das Fenster nach der Auswahl

    # Verwenden des gespeicherten Rechtecks für grabCut
    cv2.grabCut(image, mask, selected_rect, bgdModel, fgdModel, 10, cv2.GC_INIT_WITH_RECT)

    kernel = np.ones((3, 3), np.uint8)
    for _ in range(refine_iterations):
        mask = cv2.dilate(mask, kernel, iterations=1)
        mask = cv2.erode(mask, kernel, iterations=1)

    mask2 = np.where((mask==2)|(mask==0), 0, 1).astype('uint8')
    image = image * mask2[:,:,np.newaxis]

    return image
```

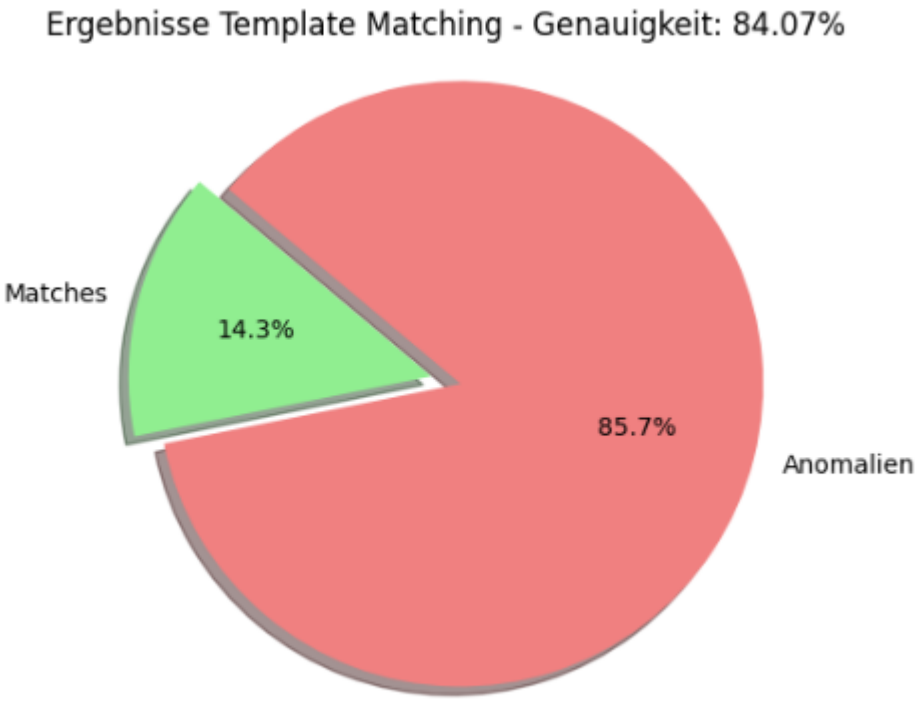
Um nun noch bestmögliche Ergebnisse erzielen zu können, wird eine Funktion implementiert, mit der das übergebene Bild skaliert und rotiert wird. Es werden drei Skalierungsfaktoren definiert: 80%, 100% und 120% der Originalgröße. Folglich wird eine Liste erstellt, um die skalierten und gedrehten Bilder zu speichern. Für jede Skalierung wird das Bild mit der resize-Funktion auf die entsprechende Größe skaliert. Für jeden Winkel von 0 bis 360° werden die Dimensionen des skalierten Bildes und das Zentrum des Bildes berechnet.

Mit der getRotationMatrix2D- Funktion wird eine Rotationsmatrix M erstellt, die das Bild um den angegebenen Winkel dreht. Die warpAffine-Funktion wird dann verwendet, um das Bild basierend auf der Rotationsmatrix zu transformieren und das gedrehte Bild zu erzeugen.

```python
def scale_and_rotate(image):
    # Define the scales to apply
    scales = [0.8, 1.0, 1.2]

    # Initialize a list to store the transformed images
    transformed_images = []

    # Iterate over each scale
    for scale in scales:
        # Resize the image
        resized_image = cv2.resize(image, None, fx=scale, fy=scale)

        # Iterate over each angle from 0 to 360
        for angle in range(360):
            # Get the image's dimensions
            (h, w) = resized_image.shape[:2]

            # Compute the center of the image
            center = (w / 2, h / 2)

            # Perform the rotation
            M = cv2.getRotationMatrix2D(center, angle, 1.0)
            rotated_image = cv2.warpAffine(resized_image, M, (w, h))

            # Add the transformed image to the list
            transformed_images.append(rotated_image)

    return transformed_images
```

4.3.2 Auswertung

Insgesamt konnte für diesen Ansatz ein zufriedenstellendes Ergebnis erzielt werden. Ergänzend ist jedoch wichtig zu nennen, dass der Template-Matching Ansatz unter für den Algorithmus nicht optimalen Bedingungen (variierende Belichtung, Rotation und Objektgröße) durchgeführt wurde, was die Performance deutlich negativ beeinträchtigt hat.



Das Diagramm zeigt die letztliche Klassifizierung des Algorithmus, der 85,7% aller Bilder als Anomalien und die verbleibenden 14,3% als Matches klassifiziert hat.

Letztlich wurde eine Genauigkeit von 84,07% für diesen Ansatz erzielt, die nach folgender Formel berechnet wurde:

$$\text{Genauigkeit} = \left(\frac{\text{eigentliche Anomalien} + \text{detektierte Matches}}{\text{alle Bilder}} \right) \times 100$$

5. Fazit und mögliche Erweiterungen

Alle drei getesteten Ansätze liefern zufriedenstellende Ergebnisse bei der kamerabasierten Anomalie Erkennung und können einen wertvollen Beitrag zu Qualitätskontrolle in Industrieunternehmen liefern.

Der Ansatz mit CNNs (YOLO eingeschlossen) ist für die Klassifikation und Fehlererkennung hervorragend geeignet und kann flexibel eingesetzt werden. Die Performance in unserer Problemstellung ist sehr gut, sodass wir zuversichtlich sind, dass dieser Ansatz auch bei komplexeren Problemstellungen gute Ergebnisse liefern kann.

Das Template Matching ist ein vielversprechender und einfach anzuwendender Ansatz, da er ohne vorheriges Training und ohne umfangreiche Beispieldaten (v.a. problematisch bei Bildern seltener auftretender Fehlerklassen) durchgeführt werden kann. In einem tatsächlichen industriellen Einsatzfall könnte Template Matching noch eine bessere Performance erreichen, da der gesamte Algorithmus sehr sensibel auf Unterschiede bei der Rotation, Belichtung und Skalierung (Größe der Objekte) ist. In realen Anwendungsszenarien sind die Umwelteinflüsse auf die aufgenommenen Bilder meist deutlich geringer, was die Nutzung von Template Matching noch deutlich robuster und vielversprechender machen sollte.