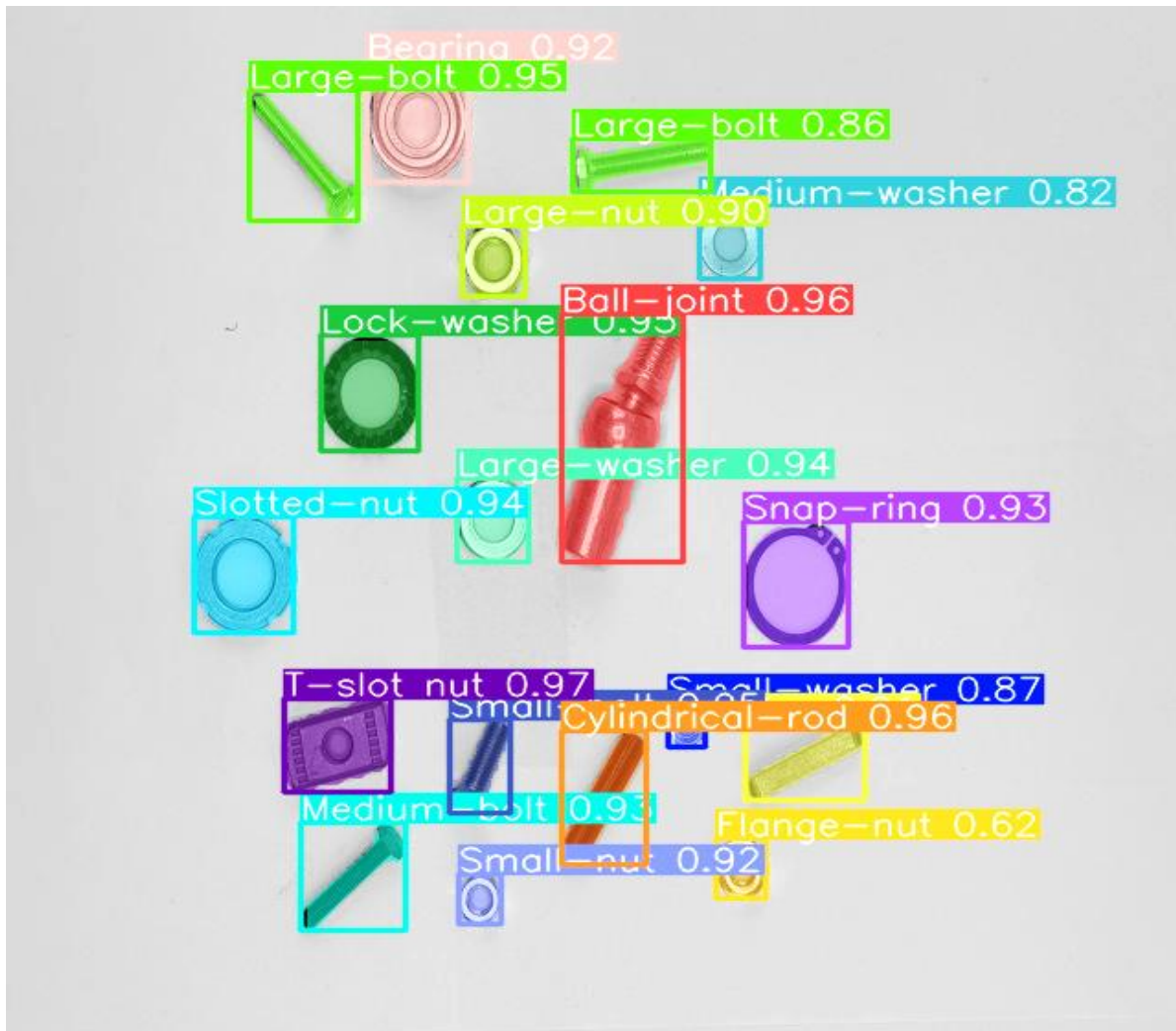


Onderdeel detectie en analyse

Door 2 methodes: Deep neuraal netwerk en conventionele computer vision



Computer Vision

Datum 23-Oct-24

Auteurs: Sven Dekker, Mats van der Vliet

V 1.0

Inhoudsopgave

Inhoudsopgave	2
Introductie	4
Doelstelling	4
Hoofdvraag.....	4
Eisen.....	4
Dataset maken	5
Stap 1: Foto's.....	5
Stap 2: Annotatie	5
Stap 3: Dataset generen	6
Stap 4: Keuze download format.....	6
Voorbeeld foto's:.....	7
Beschrijving annoteren.....	7
Beschrijving dataset.....	8
Flowchart	9
Flowchart Deep Neural Network	9
Flowchart Conventioneel.....	10
Deep neural network	11
Keuze deep neural network.....	11
Interessante onderdelen code	11
Train code	11
Validatie code	12
Evaluatie	14
Conventioneel	15
6.1 inleiding.....	15
6.2 Templates	15
6.3 Overzicht van de Stappen en Functies	16
6.3.1 Preprocessing.....	16
6.4 Detectie van Contouren	17
6.5 Resultaat Weergeven en Exporteren naar Excel.....	18

6.6 Evaluatie.....	19
6.7 Conclusie	20
Conclusie	21
Bibliografie	22

Introductie

In veel productieomgevingen blijven kleine onderdelen na het assemblageproces over. Deze zijn vaak nog bruikbaar, maar het handmatig sorteren ervan is tijdrovend en inefficiënt. Om dit proces te automatiseren, kan computer vision worden ingezet voor het detecteren en classificeren van de onderdelen, wat een eerste stap is richting robots die deze onderdelen kunnen sorteren. In deze opdracht ligt de focus op het bepalen van de positie en oriëntatie van de onderdelen met beeldverwerking, zonder de uiteindelijke sorteerstap met robots. Het doel is om zo de basis te leggen voor efficiënter hergebruik van restonderdelen.

Doelstelling

Hoofdvraag

“Programmeer een applicatie die, op basis van een zelf gemaakte dataset, verschillende soorten onderdelen kan detecteren (positie + oriëntatie bepalen) en classificeren (welk onderdeel is het).”

Eisen

1. **Vergelijking van technieken:**

Zowel (deep) neurale netwerken als conventionele computer vision technieken (zoals edge- en contour-detection, HAAR-cascades) worden onderzocht voor de detectie en classificatie van onderdelen. De resultaten van beide methoden worden vergeleken.

2. **Onderdeelclassificatie:**

Er wordt gewerkt met een verzameling onderdelen die geclassificeerd moeten worden.

3. **Ondergrond keuze:**

De ondergrond waarop de onderdelen liggen wordt zelf gekozen, aangezien de sorteermachine flexibel is in de ondergrondkleur.

4. **Dataset opbouw:**

Een eigen dataset van consistente foto's wordt gecreëerd, waarbij perspectief, belichting en andere beeldinstellingen gelijk worden gehouden voor robuuste detectie.

5. **Oriëntatieherkenning:**

De onderdelen moeten herkend worden in verschillende oriëntaties, ook als ze op hun zijkant of bovenkant liggen.

6. **Losse en overlappende objecten:**

Onderdelen worden herkend, zowel wanneer ze los liggen als wanneer ze elkaar gedeeltelijk overlappen.

7. **Data-extractie en opslag:**

De gedetecteerde onderdelen worden vastgelegd in een eenvoudige database met informatie over het type, de aantallen en de posities van de objecten.

8. **Programmeertaal en tools:**

Het project wordt uitgevoerd in Python, met gebruik van OpenCV en andere relevante bibliotheken en neurale netwerken.

Dataset maken

Tijdens de lessen hebben we kennigemaakt met Roboflow. Roboflow biedt alles wat nodig is om een dataset te creëren, er zijn ook alternatieven zoals datatorch. Voordat we aan de slag kunnen met het maken van een dataset, moeten we eerst foto's verzamelen.

Stap 1: Foto's

Ons werd aangeraden om tussen de 200 en 300 foto's te verzamelen om een model effectief te kunnen trainen. We maakten gebruik van een lichtbox en zorgden ervoor dat het perspectief, de belichting en andere beeldinstellingen consistent waren voor een robuuste training. In totaal hebben we ongeveer 300 foto's genomen, waarbij we varieerden de oriëntatie en positie van de onderdelen om een zo divers mogelijke dataset te creëren; wanneer onderdelen op hun zijkant of bovenkant gelegd konden worden, hebben we dat gedaan en ook verschillende posities binnen het fotokader gebruikt voor een robuuste dataset. Daarnaast hebben we ongeveer 30 foto's gemaakt van objecten die tegen of in elkaar liggen. Het idee hierachter was dat het model deze situaties later ook kan herkennen. Bovendien hebben we van elk onderdeel vier afzonderlijke foto's genomen, wat in totaal ongeveer 60 foto's oplevert. Dit leek ons nuttig voor de conventionele methode.

Stap 2: Annotatie

Op Roboflow hebben we eerst een project aangemaakt voor de eindopdracht waaraan we beiden deelnemen. In de instellingen van ons project hebben we 17 classes aangemaakt en de geüploade foto's in tweeën gesplitst, waarna we begonnen met labelen. We hebben afgesproken om allebei eerst 100 foto's te labelen om te kijken welk resultaat dit opleverde. Mocht dit niet goed werken, dan hadden we altijd nog 100 foto's achter de hand om aan onze dataset toe te voegen.

In deze analyse is er een overzicht te zien van hoe de labelen heeft plaatsgevonden. Er zijn waarschijnlijk enkele onjuiste labels toegewezen, maar de foutmarge bedraagt ongeveer 10, wat erg klein is in vergelijking met de totaal 3628 annotaties.

Number of Images

201

- 0 missing annotations
- 0 null examples

Number of Annotations

3628

- 18.0 per image (average)
- Across 17 classes



Stap 3: Dataset generen

Nu er geannoteerde foto's zijn kan er een dataset gegenereerd worden. Hierbij geeft Roboflow opties zoals hoe de foto's willen splitsen kunnen worden tussen trainen, valideren en testen. Er is gekozen voor de standaard 70% 20% 10%. Volgende keuze is preprocessing. Hierbij zijn er voor de opties Auto-Orient, Resize 640x640 en grayscale aangezet. Bij de optie Augmentation is er gekozen voor de opties 90° rotate en rotation between -15° en +15° gekozen om de dataset te vergroten naar 481 images.

The screenshot shows the Roboflow dataset creation interface with the following configuration:

- Source Images:** Images: 201, Classes: 17, Unannotated: 0
- Train/Test Split:** Training Set: 140 images, Validation Set: 40 images, Testing Set: 21 images
- Preprocessing:** Auto-Orient: Applied, Resize: Stretch to 640x640, Grayscale: Applied
- Augmentation:** 90° Rotate: Clockwise, Counter-Clockwise, Upside Down; Rotation: Between -15° and +15°
- 5 Create:** Review your selections and select a version size to create a moment-in-time snapshot of your dataset with the applied transformations. Larger versions take longer to train but often result in better model performance. See how this is calculated. Maximum Version Size: 481 images (3x). Create

Stap 4: Keuze download format

In de les kregen we een voorbeeld te zien met YOLOv3. Vanwege dit voorbeeld hebben wij ook gekozen voor YOLO maar tijdens ons onderzoek bleken er ook nieuwere versies te zijn. Er was in eerste instantie gekozen voor YOLO11 maar met te weinig kennis liepen we tegen een probleem aan. Het probleem was dat we in de veronderstelling waren dat we ook een yolo11n-seg.yaml file ergens moesten downloaden. Dit bestand konden we nergens vinden.

```
Train
Train YOLO11n-seg on the COCO8-seg dataset for 100 epochs at image size 640. For a full list of available arguments see the Configuration page.

Example
Python CLI
from ultralytics import YOLO

# Load a model
model = YOLO('yolo11n-seg.yaml') # build a new model from YAML
model = YOLO('yolo11n-seg.pt') # load a pretrained model (recommended for training)
model = YOLO('yolo11n-seg.yaml').load('yolo11n.pt') # build from YAML and transfer weights

# Train the model
results = model.train(data='coco8-seg.yaml', epochs=100, imgsz=640)
```

```
FAQ
How do I train a YOLO11 segmentation model on a custom dataset?

To train a YOLO11 segmentation model on a custom dataset, you first need to prepare your dataset in the YOLO segmentation format. You can use tools like JSON2YOLO to convert datasets from other formats. Once your dataset is ready, you can train the model using Python or CLI commands:

Example
Python CLI
from ultralytics import YOLO

# Load a pretrained YOLO11n-seg model
model = YOLO('yolo11n-seg.pt')

# Train the model
results = model.train(data='path/to/your_dataset.yaml', epochs=100, imgsz=640)
```

Toen is er gekozen voor YOLOv8 omdat daar ook veel informatie over te vinden was. Achteraf bleek de oplossing verder op de webpagina te staan onder FAQ. In deze python code heb je geen .yaml file nodig en kunnen we ons eigen segmentatie model trainen op onze eigen dataset. Eenmaal we hier achter kwamen was er al gekozen voor YOLOv8 en deze heeft al de mogelijkheid om voor instantiesegmentatie. “Instantiesegmentatie is een geavanceerde computervisetaak waarbij elk object in een afbeelding op pixelniveau wordt gedetecteerd en afgebakend. In tegenstelling tot objectdetectie, waarbij objecten worden geïdentificeerd en geclassificeerd met begrenzingsvakken, biedt instantiesegmentatie een gedetailleerder inzicht door elke pixel die bij een bepaalde objectinstantie hoort, te labelen.” (Ultralytics, n.d.). Dus uiteindelijk was er gekozen om te downloaden in een YOLOv8 format.

Voorbeeld foto's:

Voorbeeld van de foto's die we gemaakt hebben. De afstand en belichting is gelijk gebleven.



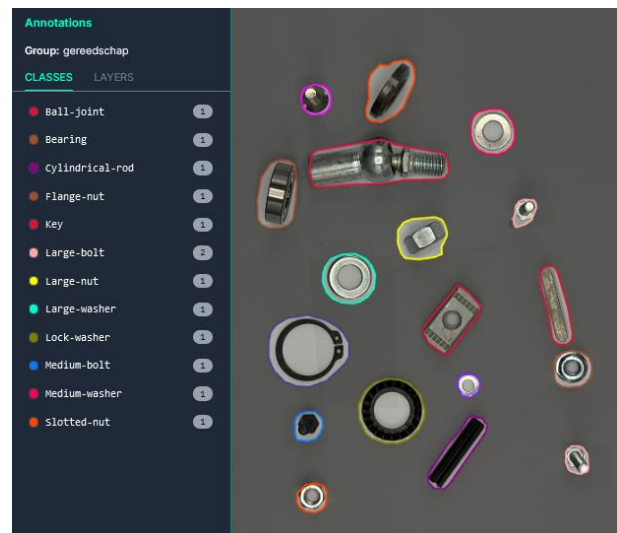
Beschrijving annoteren

Annoteren is het proces van het toekennen van extra informatie of labels aan gegevens, zoals afbeeldingen, tekst, of audio, om deze begrijpelijk te maken voor een machine learning-model. In Computer Vision verwijst annoteren meestal naar het markeren van objecten in afbeeldingen, bijvoorbeeld door het trekken van een bounding box om een object, of door gebieden van een afbeelding te markeren met een specifieke klasse (zoals "auto" of "boom"). Enkele veelvoorkomende vormen van annotatie zijn:

1. **Bounding boxes:** Rechthoeken die objecten in een afbeelding omcirkelen om hun positie en grootte te markeren.
2. **Polygons of segmentation:** Nauwkeuriger dan bounding boxes; hierbij wordt het exacte gebied van het object afgebakend.
3. **Classificatie:** Toekennen van een label aan de gehele afbeelding, zoals "kat" of "hond".

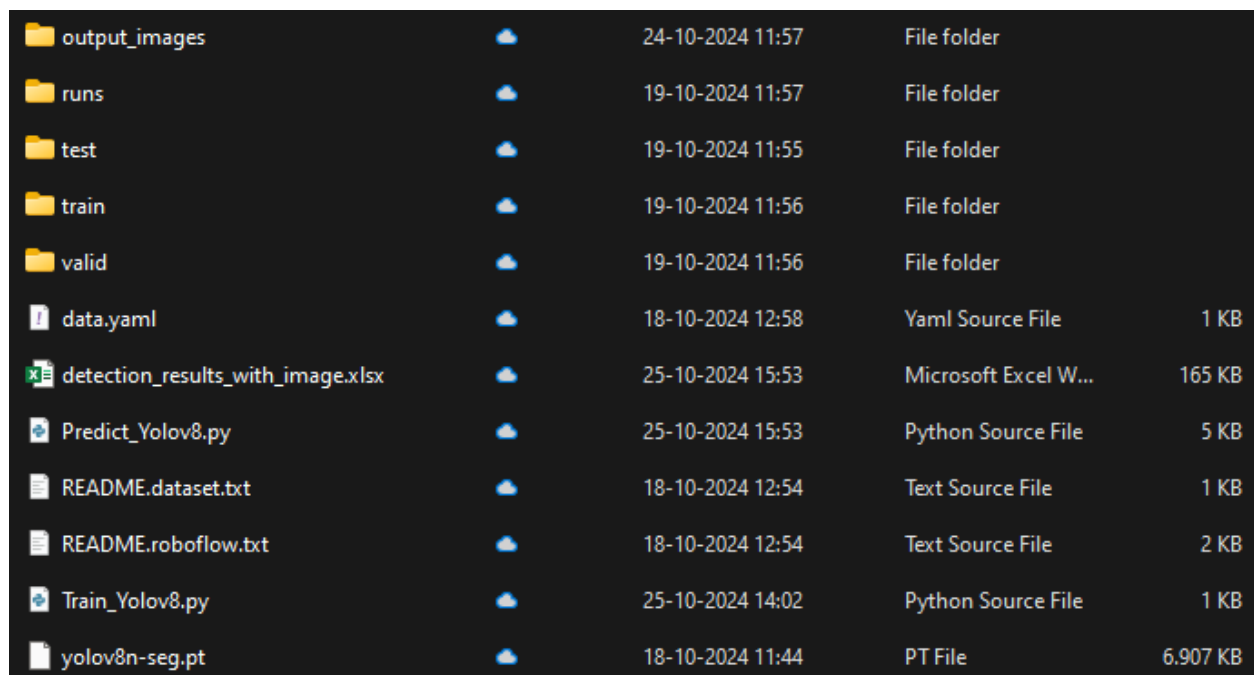
Annotaties zijn essentieel voor het trainen van modellen, omdat ze het algoritme helpen begrijpen wat het moet leren herkennen of classificeren.

Dit is een van onze geannoteerde foto's doormiddel van polygon segmentation.



Beschrijving dataset

Een dataset is een gestructureerde verzameling gegevens die gebruikt wordt voor analyse of het trainen van modellen, bijvoorbeeld in machine learning en computer vision. In computer vision bevat een dataset vaak afbeeldingen met bijbehorende annotaties, zoals objectlocaties en -klassen. Een goede dataset is divers en representatief, met variaties in belichting, oriëntatie en achtergronden om het model robuust te maken. Voor machine learning wordt de dataset meestal opgedeeld in een trainingsset, validatieset en testset om het model te trainen, optimaliseren en evalueren.



Item	Icon	Date	Time	Type	Size
output_images	Folder	24-10-2024	11:57	File folder	
runs	Folder	19-10-2024	11:57	File folder	
test	Folder	19-10-2024	11:55	File folder	
train	Folder	19-10-2024	11:56	File folder	
valid	Folder	19-10-2024	11:56	File folder	
data.yaml	File	18-10-2024	12:58	Yaml Source File	1 KB
detection_results_with_image.xlsx	File	25-10-2024	15:53	Microsoft Excel W...	165 KB
Predict_Yolov8.py	File	25-10-2024	15:53	Python Source File	5 KB
README.dataset.txt	File	18-10-2024	12:54	Text Source File	1 KB
README.roboflow.txt	File	18-10-2024	12:54	Text Source File	2 KB
Train_Yolov8.py	File	25-10-2024	14:02	Python Source File	1 KB
yolov8n-seg.pt	File	18-10-2024	11:44	PT File	6.907 KB

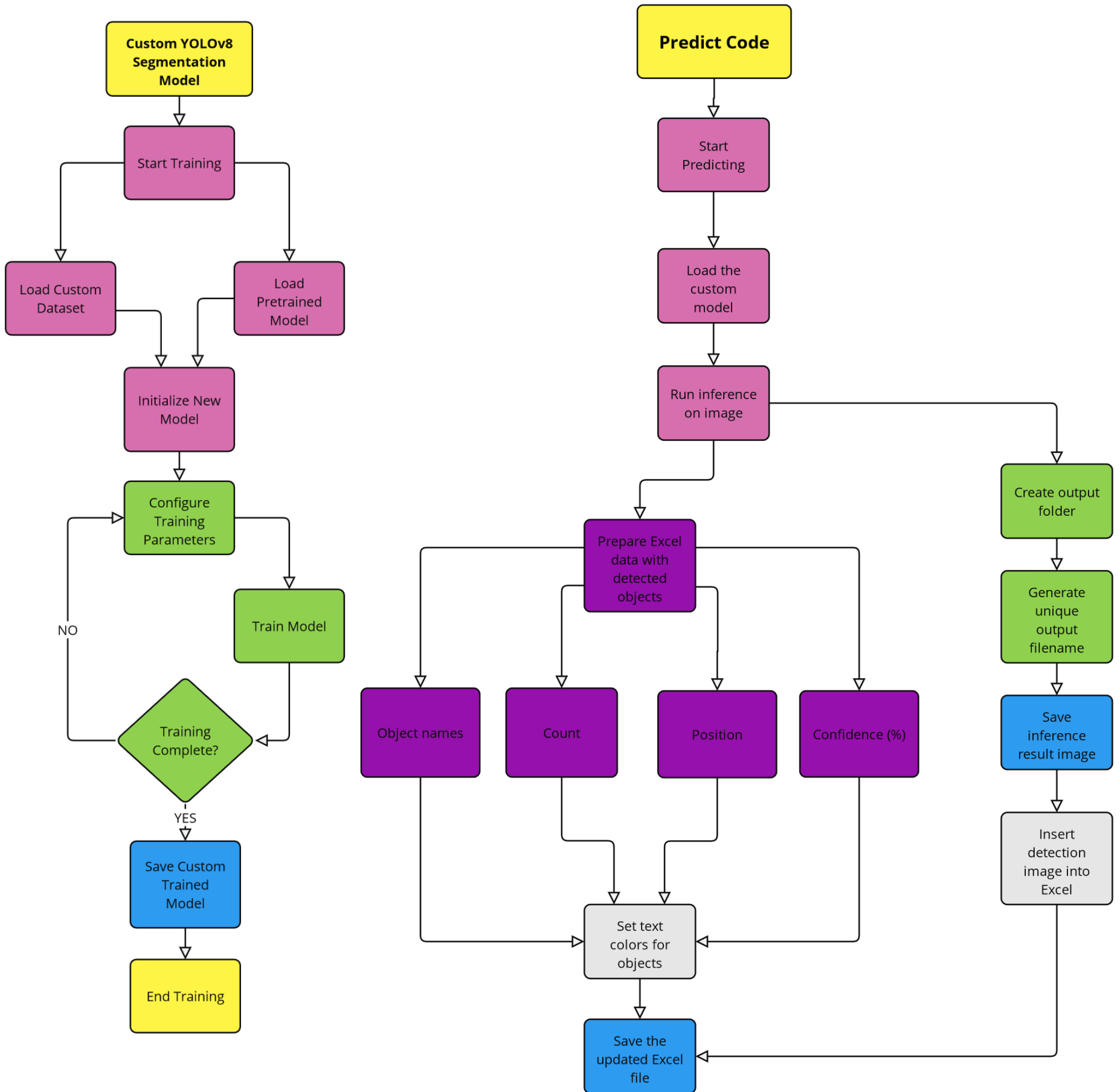
Dit is een voorbeeld van hoe onze dataset eruit ziet. We hebben gewerkt uit deze folder dus er zijn een paar bestanden en een 'output_images' folder toegevoegd. De bestanden die toegevoegd zijn is 'yolov8n-seg.pt' de twee python codes 'Predict_Yolov8.py' en 'Train_Yolov8.py'. Uit deze python codes komt het excel bestand 'detection_result_with_image.xlsx' en de folder met images.

Flowchart

De twee methodes worden hieronder weergegeven doormiddel van een flowchart

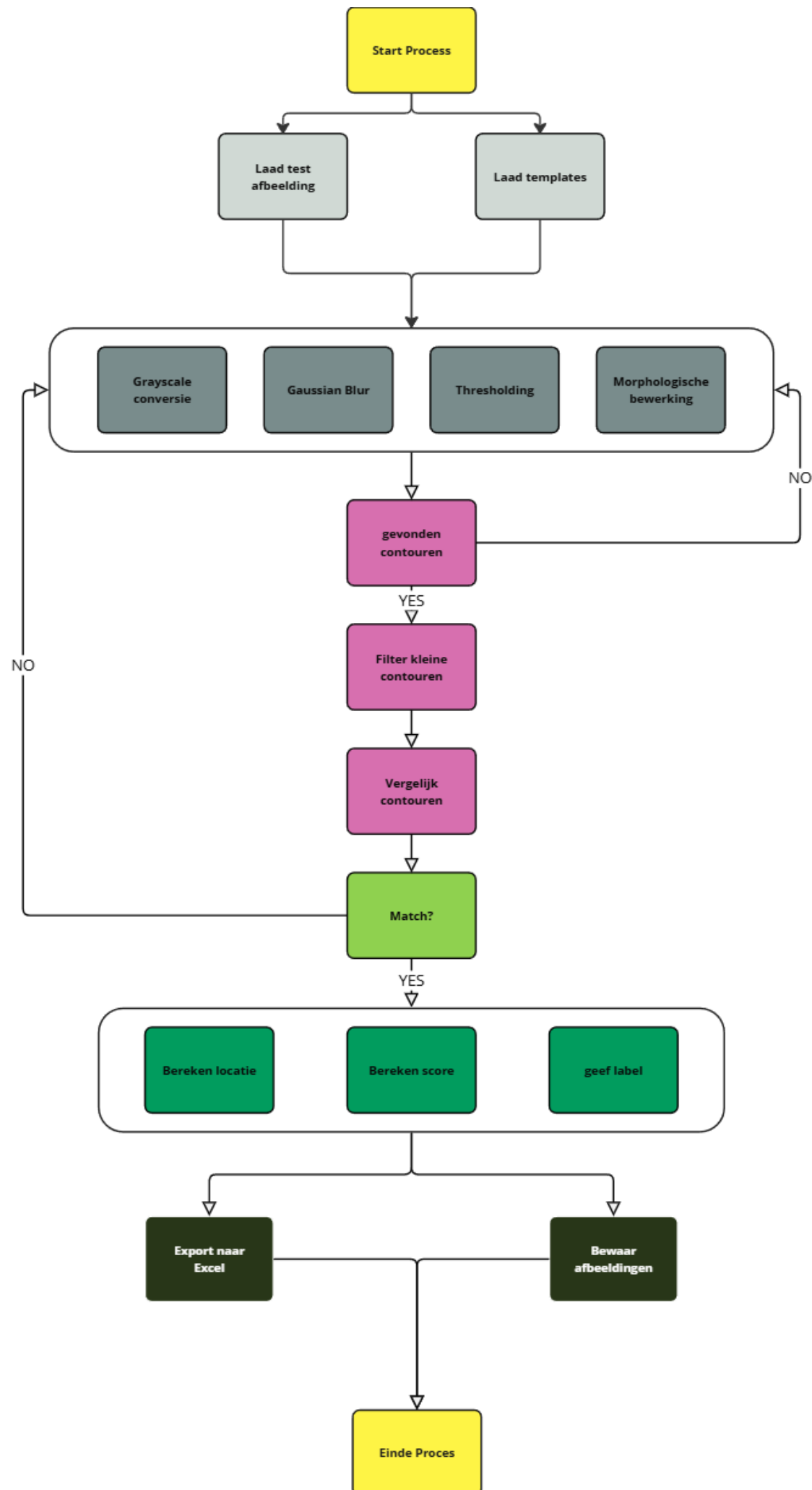
Flowchart Deep Neural Network

Voor deze methode zijn twee codes gebruikt: de train code en de predict code. Aan de linkerkant zie je een flowchart die oppervlakkig de stroomlijn toont voor het trainen van een custom segmentatiemodel met behulp van je eigen dataset. Dit geeft inzicht in wat er ongeveer achter de schermen gebeurt tijdens het trainen. De rechter flowchart visualiseert de predict code, die laat zien hoe voorspellingen worden gemaakt op basis van een afbeelding. Hierbij worden zowel de afbeelding als de gegenereerde gegevens opgeslagen in een Excel-bestand, wat de werking van de code duidelijker maakt.



Flowchart Conventioneel

De flowchart voor de conventionele methode laat vereenvoudigd zien hoe de code in zijn werking gaat. Ook zijn er enkele feedback momenten waar de programmeur zou moeten ingrijpen om bepaalde instellingen aan te passen.



Deep neural network

In dit hoofdstuk wordt het gebruik van deep neural networks (DNNs) binnen het project toegelicht. Er wordt ingegaan op de keuze voor DNNs, met een focus op waarom deze methode is gekozen boven conventionele algoritmen. Daarnaast worden de meest interessante onderdelen van de code besproken om inzicht te geven in de implementatie van het netwerk. Tot slot volgt een evaluatie van de code, waarbij de beperkingen van de huidige aanpak worden belicht en suggesties voor verdere optimalisatie worden aangedragen.

Keuze deep neural network

In het onderdeel 'Dataset' onder 'Stap 4' is al beschreven waarom we aanvankelijk voor YOLO hebben gekozen: tijdens de lessen maakten we hiermee kennis. We hebben de verschillende versies vergeleken en zijn, nadat YOLO11 niet werkte, overgestapt naar YOLOv8, die vervolgens succesvol werd ingezet. Een alternatief had SAM2 kunnen zijn, waarvoor op de GitHub-pagina (facebookresearch, n.d.) installatie- en gebruiksinstructies beschikbaar zijn. De werkwijze van SAM2 vertoont enige gelijkenis met die van YOLO

Interessante onderdelen code

Dit stukje wordt opgesplitst in twee delen voor de twee codes

Train code

In de afbeelding hiernaast is de volledige code voor de training te zien.

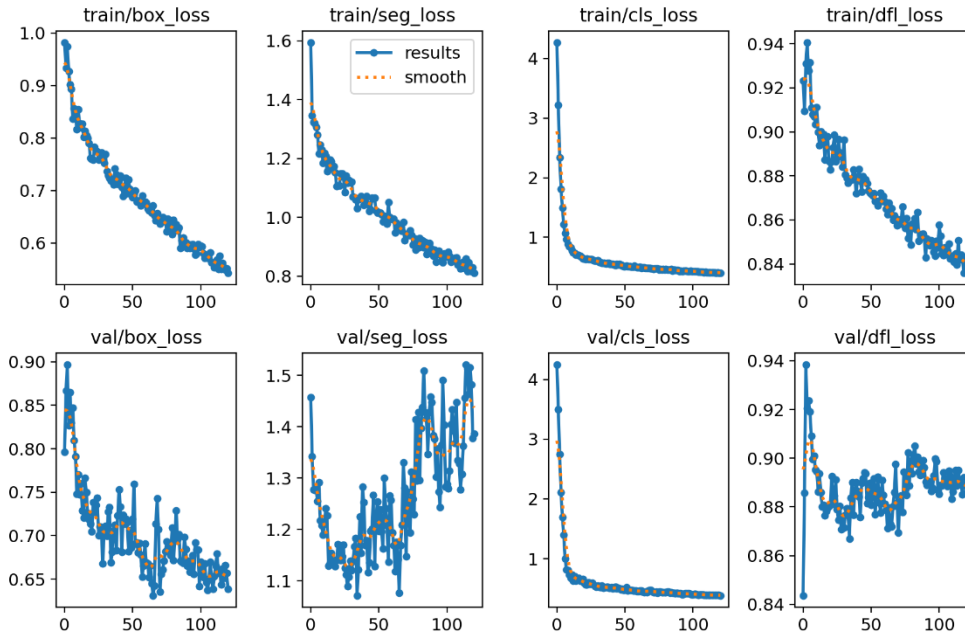
```
Train_Yolov8.py ×
1 from ultralytics import YOLO
2
3 # Load a COCO-pretrained YOLOv8n model
4 model = YOLO("yolov8n-seg.pt")
5
6 # Train the model on the COCO8 example dataset for 150 epochs
7 results = model.train(data="data.yaml", epochs=150, imgsz=640)
8
```

YOLO maakt hierbij gebruik van een *pretrained* model, namelijk *yolov8n-seg.pt*. Dit model is specifiek getraind voor segmentatietaken. Op de website van Ultralytics zijn verschillende segmentatiemodellen beschikbaar voor download (*Segment models*, n.d.). Wij hebben gekozen voor *yolov8n-seg.pt* omdat dit het lichtste model is qua belasting op de CPU, wat belangrijk was omdat de training plaatsvond op een laptop. Voor het aantal *epochs* is gekozen voor 150, zoals aanbevolen tijdens de lessen. De parameter *imgsz* is ingesteld op 640, omdat onze dataset met deze resolutie is gegenereerd.

Model	size (pixels)	mAP ^{val} ₅₀₋₉₅	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Resultaten na de training

Uit de grafiek voor validatie/segmentatieverlies blijkt dat het validatieverlies vanaf epoch 70 toeneemt. De standaardinstelling stopt de training als er na 50 epochs geen prestatieverbeteringen worden waargenomen. Uiteindelijk leverde epoch 71 het beste resultaat op.



Validatie code

Deze code voert objectdetectie uit op een afbeelding met behulp van een YOLOv8-model. De resultaten worden verwerkt en opgeslagen in een Excel-bestand, waarbij de afbeelding met detecties ook wordt toegevoegd. De code bevat kleurcodering en een hyperlink naar de outputmap. Hieronder volgt een uitleg van de belangrijkste onderdelen van de code.

1. Model Laden en Voorspellingen Doen

Het model wordt geladen vanaf een opgegeven pad. Vervolgens wordt het model toegepast op een afbeelding, waarbij het objecten detecteert en voor elk object relevante informatie opslaat, zoals de coördinaten van de bounding box en de confidence score.

```
# Load the custom model
model = YOLO(r"C:\Users\sven-\OneDrive - HVA\HVA\

# Run inference on the provided image
image_path = r"C:\Users\sven-\OneDrive - HVA\
results = model(image_path)
```

2. Opslaan van de Afbeelding met Bounding Boxes

De code toont de detectieresultaten op de originele afbeelding door bounding boxes weer te geven. Deze afbeelding wordt opgeslagen in een output-map. Als er al een afbeelding met dezelfde naam bestaat, wordt een nieuwe naam gegenereerd door een teller toe te voegen.

```
# Check if the output file already exists and modify the filename accordingly
counter = 1
while os.path.exists(output_image_path):
    output_image_path = os.path.join(output_folder, f"output_{counter}.png")
    counter += 1
```

3. Resultaten Opslaan in Excel

Een lege datastructuur wordt voorbereid om de informatie over elk gedetecteerd object op te slaan, zoals de klasse, positie en betrouwbaarheid. Deze data wordt vervolgens omgezet naar een tabel en opgeslagen als een Excel-bestand.

```
# Data storage for Excel output
data = {
  "Object": [],
  "Count": [],
  "Position (x, y, width, height)": [],
  "Confidence (%)": []
}
```

4. Kleuren en Opmaak in de Excel-cellen

De code selecteert een kleurenpalet en past dit toe op de rijen in de Excel, wat de resultaten beter leesbaar maakt. De kleuren worden afgewisseld per rij. Elke objectnaam krijgt een zwarte tekstopmaak, en de rij krijgt een andere kleur, waardoor het gemakkelijk is om onderscheid te maken tussen verschillende objecten.

```
# Apply row colors for alternating background fill
for row in ws.iter_rows(min_row=2, max_row=ws.max_row, max_col=ws.max_column):
    color = colors[(row[0].row - 2) % len(colors)] # Rotate through colors
    fill = PatternFill(start_color=color, end_color=color, fill_type="solid")

    # Apply row fill color
    for cell in row:
        cell.fill = fill
```

5. Afbeelding Toevoegen aan de Excel

De eerder opgeslagen detectieresultaat-afbeelding wordt ingevoegd in de Excel op een specifieke cel (bijvoorbeeld cel E3). Dit geeft een visuele weergave van de detecties naast de gegevens.

```
# Insert the saved output image into Excel and resize it
img = ExcelImage(output_image_path)
img.width = 400 # Set desired width in points
img.height = 300 # Set desired height in points
```

6. Hyperlink Toevoegen naar de Output-map

Er wordt een hyperlink naar de output-map toegevoegd in de Excel, waardoor de gebruiker direct naar de map kan navigeren met de opgeslagen afbeeldingen. De hyperlinktekst is blauw en onderstreept, zodat het eruitziet als een gewone hyperlink.

```
# Add a hyperlink to the output image folder
hyperlink_cell = ws.cell(row=1, column=5) # Example: E1
hyperlink_cell.value = "Image Output Folder"
hyperlink_cell.hyperlink = output_folder # Link to the output image folder
hyperlink_cell.font = Font(color="0000FF", underline="single") # Blue color
```

7. Resultaat excel bestand

Object	Count	Position (x, y, width, height)	Confidence (%)	Image Output Folder
T-slot nut	1	(180.4, 459.5, 57.3, 57.7)	97.4	
Ball-joint	1	(335.5, 268.8, 66.2, 153.7)	96.2	
Cylindrical-	1	(325.1, 492.1, 46.0, 82.5)	95.7	
Lock-washer	1	(197.9, 240.8, 53.8, 72.3)	95.4	
Small-bolt	1	(258.0, 472.4, 32.8, 58.6)	95.1	
Large-bolt	2	(161.9, 93.0, 58.7, 80.4), (346.1, 99.2, 75.4, 32.8)	95.0, 86.1	
Slotted-nu	1	(129.7, 353.8, 53.9, 72.2)	94.4	
Large-wash	1	(265.2, 319.4, 38.4, 51.6)	93.5	
Snap-ring	1	(429.8, 359.6, 56.8, 77.6)	93.4	
Medium-b	1	(188.7, 541.2, 57.2, 67.6)	93.2	
Small-nut	1	(257.7, 555.1, 23.1, 30.8)	91.8	
Bearing	1	(223.9, 72.0, 55.3, 75.1)	91.6	
Key	1	(435.3, 469.8, 64.7, 48.1)	90.7	
Large-nut	1	(265.4, 158.5, 33.6, 43.7)	89.7	
Small-was	1	(370.6, 447.0, 20.0, 26.5)	86.9	
Medium-w	1	(393.8, 147.2, 33.1, 44.6)	82.2	
Flange-nut	1	(400.0, 536.2, 27.8, 37.6)	61.6	

Evaluatie

In dit hoofdstuk worden de beperkingen van het model besproken en mogelijke optimalisaties in de code aangeduid, waarmee de prestaties verder verbeterd kunnen worden.

Verbeterde Foutafhandeling

Door het toevoegen van eenvoudige controles kan de code beter omgaan met problemen, zoals wanneer het model of de afbeelding niet goed geladen kan worden. Dit zorgt ervoor dat de gebruiker duidelijke meldingen krijgt als er iets misgaat.

Batchverwerking:

In plaats van slechts één afbeelding tegelijk te verwerken, zou ik een batchverwerkingsmethode kunnen implementeren die meerdere afbeeldingen in één keer verwerkt. Dit zou de prestaties aanzienlijk kunnen verbeteren.

Toepassing van Segmentatie in Plaats van Objectdetectie

Overweeg om de output van het model te verbeteren door gebruik te maken van image segmentation in plaats van alleen objectdetectie met bounding boxes. Door polygonen te genereren voor elke gedetecteerde instantie kunnen de vormen en grenzen van objecten nauwkeuriger worden vastgelegd, wat nuttig kan zijn voor verdere analyse en verwerking van de afbeeldingen. Dit kan bijvoorbeeld de kwaliteit van de resultaten verhogen in toepassingen zoals robotica, medische beeldvorming en objectherkenning.

Conventioneel

6.1 inleiding

Om de opdracht op de conventionele methode op te lossen hebben we een Computer Vision-applicatie ontwikkeld die net zoals in de DNN methode onderdelen herkent uit een afbeelding. Alleen wordt er nu gebruik gemaakt van template onderdelen in plaats van een dataset.

De applicatie vergelijkt een afbeelding met alle onderdelen met de template onderdelen en gaat dan op zoek naar contouren die overeenkomen. De contouren worden getekend en er wordt een label geplaatst. Ook wordt er een score berekend die aangeeft hoe erg de contour overeenkomt, waarbij 0 het beste en hoe hoger het getal hoe minder de contour overeenkomt. Uiteindelijk worden de resultaten naar een Excel-bestand geëxporteerd.

De applicatie combineert technieken zoals beeldverwerking, contourdetectie en vormvergelijking met OpenCV, en creëert een overzichtelijke database met gevonden objecten, hun posities, en overeenkomstsscores.

Daarnaast zijn tussenstappen van de verwerkingsprocessen opgeslagen, om een goed inzicht te krijgen over wat er gebeurt.

6.2 Templates

Om de templates te gebruiken moet er een overzicht gemaakt worden van de objecten, dit hebben wij gedaan doormiddel van een mappen structuur waar elk onderdeel zijn eigen map heeft.

De map heeft de naam van het onderdeel. Daarna worden in de code de mappen aangeroepen en worden de namen van de mappen ook mee genomen, zo krijgt uiteindelijk in de test afbeelding een afbeelding van een bearing een label met bearing.

Bij het oproepen van de templates wordt de grootste contour gepakt, die wordt uiteindelijk gebruikt om in de test afbeelding een overeenkomende contour te zoeken.



ball joint contour

- ball joint
- bearing
- flange nut
- key
- large bolt
- large nut
- large washer
- lock washer
- medium bolt
- medium washer
- rod
- slotted nut
- small bolt
- small nut
- small washer
- snap ring
- T-slot nut

6.3 Overzicht van de Stappen en Functies

6.3.1 Preprocessing

De eerste stap is het voorbereiden van de invoerafbeelding voor verdere verwerking. We hebben een aantal technieken gebruikt om de invoerafbeelding om te zetten in een vorm die geschikt is voor het detecteren van contouren:

- **Grayscale conversie:** We hebben de afbeelding omgezet naar een zwart-wit (grayscale) beeld, omdat kleuren vaak afleidend zijn en niet noodzakelijk zijn voor contourdetectie.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

- **Gaussian Blur:** We hebben een Gaussiaanse vervaging toegepast om ruis te verminderen, wat kleine foutieve contouren kan veroorzaken. Dit helpt bij het verscherpen van randen.

```
blurred = cv2.GaussianBlur(gray, (7, 7), 0)
```

- **Thresholding:** Hier hebben we gebruik gemaakt van zowel vaste drempelwaarden (thresholding) als adaptieve drempelwaarden om de randen in het beeld duidelijk te isoleren. Dit maakt het gemakkelijker om objecten te onderscheiden.

```
thresh = cv2.threshold(blurred, threshold_value, 255, cv2.THRESH_BINARY_INV)
```

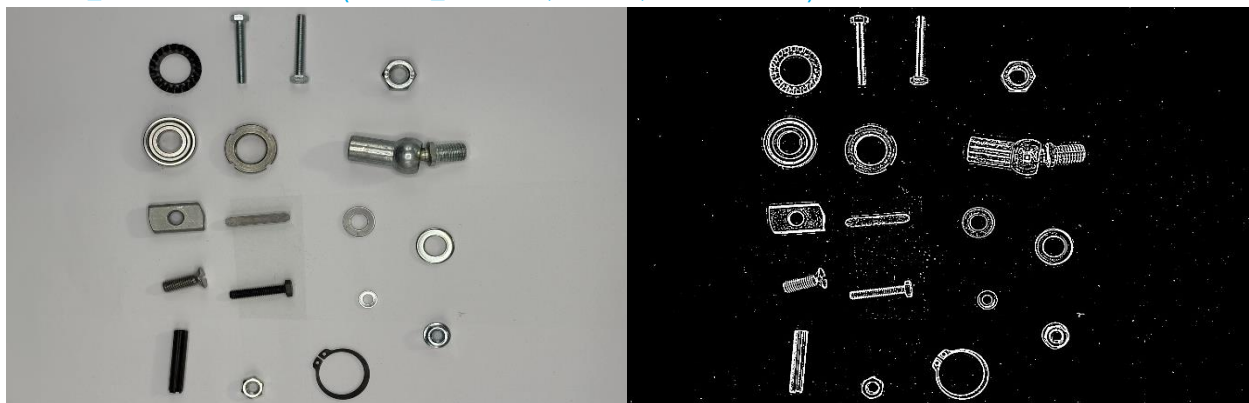
- **Morfologische Operaties:** Dit zijn bewerkingen zoals opening (om kleine ruis te verwijderen) en closing (om kleine gaten in objecten te dichten). Door deze technieken konden we duidelijke, consistente contouren verkrijgen.

```
kernel = np.ones((3, 3), np.uint8)
```

```
thresh_cleaned = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=2)
```

- **Dilatie:** Ten slotte hebben we de contouren verdikt door middel van dilatie, zodat deze gemakkelijker gedetecteerd en gematcht kunnen worden.

```
thresh_dilated = cv2.dilate(thresh_cleaned, kernel, iterations=1)
```

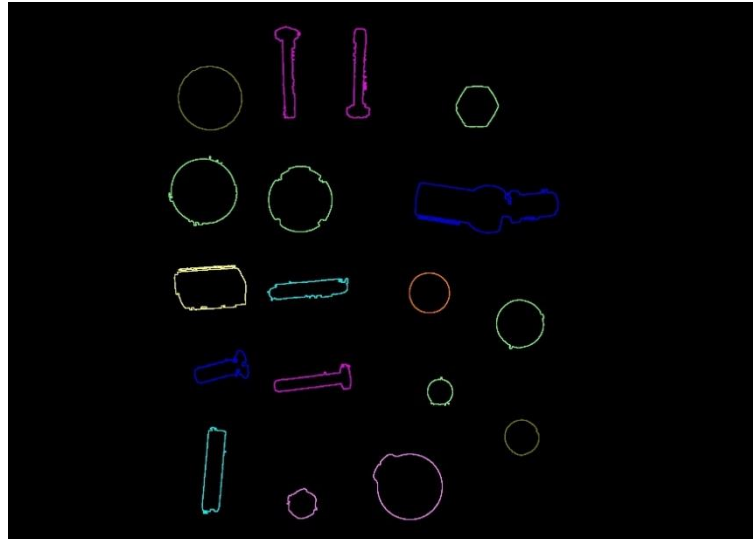


origineel & na het preprossen

6.4 Detectie van Contouren

Na de preprocessing van de afbeelding zijn we overgegaan tot het detecteren van de contouren van objecten. OpenCV biedt de `cv2.findContours()`-functie, die alle contouren in een afbeelding detecteert.

```
contours, _ =  
cv2.findContours(cleaned_image,  
cv2.RETR_EXTERNAL,  
cv2.CHAIN_APPROX_SIMPLE)
```



Contour detectie

De functie `findContours()` detecteert randen en contouren door de overgang van donkere naar lichte pixels in de afbeelding te analyseren. We hebben alleen de buitenste contouren behouden, omdat deze overeenkomen met de omtrek van de objecten en de contouren van de templates.

Vergelijking van Contouren met Template-Objecten

Voor elk gevonden contour in de afbeelding hebben we de contour vergeleken met de templates van bekende onderdelen (zoals moeren, bouten, lagers, etc.) die we van tevoren hadden ingeladen. Hiervoor gebruikten we de functie `cv2.matchShapes()`, die een score geeft die aangeeft hoe goed de vorm van de contour overeenkomt met een sjablooncontour.

```
score =  
cv2.matchShapes(main_contour,  
template_contour,  
cv2.CONTOURS_MATCH_I1, 0)
```

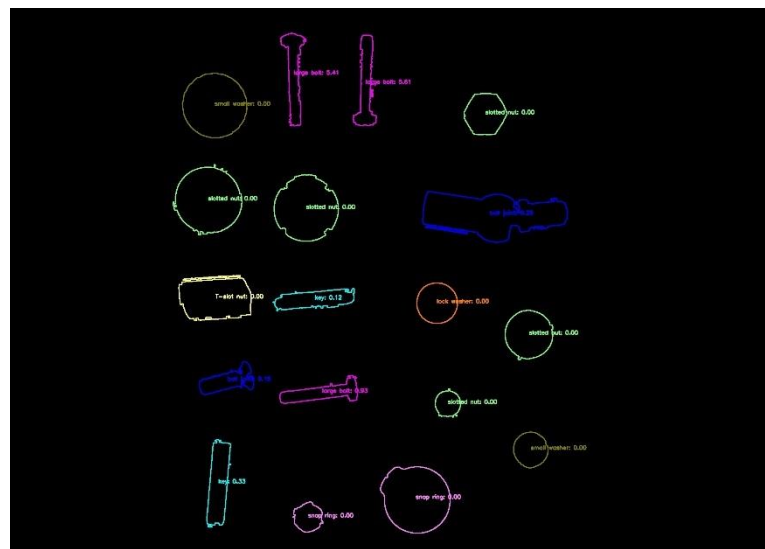


Figure 2: Label & scores

Label & scores

Een lage score betekent een goede overeenkomst tussen de contour en het sjabloon. De functie `match_contours()` vergelijkt elk contour met de sjablooncontouren en geeft een label van het template met de laagste overeenkomstsscore.

6.5 Resultaat Weergeven en Exporteren naar Excel

Nadat de beste match voor elk gevonden object was bepaald, hebben we de resultaten weergegeven door de contouren in specifieke kleuren te tekenen en labels toe te voegen met de naam van het object en de overeenkomstsscore. Hiervoor gebruikten we de functie `cv2.putText()` om tekst op de afbeeldingen te tekenen.

```
cv2.putText(final_image, f'{label}: {score:.2f}', (cx, cy), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

Daarnaast werd alle relevante informatie (onderdeel, positie, score) opgeslagen in een Excel-bestand met behulp van pandas. We gebruikten `openpyxl` om de tekst in het Excel-bestand in dezelfde kleur als het onderdeel in de afbeelding weer te geven.

```
Sdf = pd.DataFrame(export_data)
df.to_excel(excel_output_path, index=False)
```

Kleur instellen in Excel:

```
wb = load_workbook(excel_output_path)
ws = wb.active
for idx, row in df.iterrows():
    label = row['Onderdeel']
    color = color_map.get(label, (255, 255, 255))
    hex_color = rgb_to_hex(color)
    ws[f'A{idx + 2}'].font = Font(color=hex_color)
wb.save(excel_output_path)
```

Daarnaast werden screenshots van elk gevonden object opgeslagen en naar het Excel-bestand verwezen.

Onderdeel	Positie	Overeenkomstscore	Screenshot
snap ring	(806, 1376)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\snap ring_177.jpg
snap ring	(1097, 1326)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\snap ring_272.jpg
key	(563, 1284)	0.33	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\key_331.jpg
small washer	(1408, 1193)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\small washer_350.jpg
slotted nut	(1184, 1069)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\slotted nut_521.jpg
large bolt	(844, 1039)	0.93	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\large bolt_577.jpg
ball joint	(588, 1007)	0.15	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\ball joint_640.jpg
slotted nut	(1403, 882)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\slotted nut_928.jpg
key	(824, 787)	0.12	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\key_994.jpg
lock washer	(1154, 797)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\lock washer_1023.jpg
T-slot nut	(555, 784)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\T-slot nut_1064.jpg
ball joint	(1298, 558)	0.20	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\ball joint_1337.jpg
slotted nut	(800, 542)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\slotted nut_1384.jpg
slotted nut	(535, 519)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\slotted nut_1397.jpg
slotted nut	(1285, 286)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\slotted nut_1481.jpg
small washer	(553, 263)	0.00	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\small washer_1501.jpg
large bolt	(962, 204)	5.61	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\large bolt_1529.jpg
large bolt	(767, 179)	5.41	C:\Users\matsv\Desktop\output resultaat\test x\screenshots\large bolt_1534.jpg

Figure 3: Excel output

Excel output

6.6 Evaluatie

Na het uitvoeren van alle stappen en het analyseren van de uitkomst van de code en de gegenereerde waarden, kunnen we concluderen dat de code degelijk werkt. Voor elk object is een waarde en een label toegewezen, maar het label komt niet altijd overeen met het juiste object. Dit kan te wijten zijn aan verschillende factoren.

- **Contour van objecten**

Aangezien er een aantal objecten zijn waarvan de omtrek erg op elkaar lijkt of zelfs identiek is, kan het lastig zijn om onderscheid te maken.

In de toekomst zou een functie die het oppervlakte van de contour berekent nuttig kunnen zijn. Dit zou helpen om onderscheid te maken tussen bijvoorbeeld een grote en een kleine ring. Het is daarbij wel belangrijk dat de schaal van de testfoto's precies overeenkomt met die van de templatefoto's.

- **Geometrische verschillen**

Kleine variaties in de geometrische vorm van een object, zoals een lichte draaiing, schaalverandering of vervorming, kunnen een hoge matchscore geven, zelfs wanneer het object sterk overeenkomt.

- **On correcte contourdetectie**

De standaard contourdetectie met `cv2.findContours` kan soms ruis of kleine storingen als objecten detecteren, wat leidt tot verkeerde identificatie of dubbele detectie van hetzelfde object.

We hebben dit al geminimaliseerd door een `min_contour_area=350` te gebruiken, wat voorkomt dat kleine foutieve contouren worden gedetecteerd.

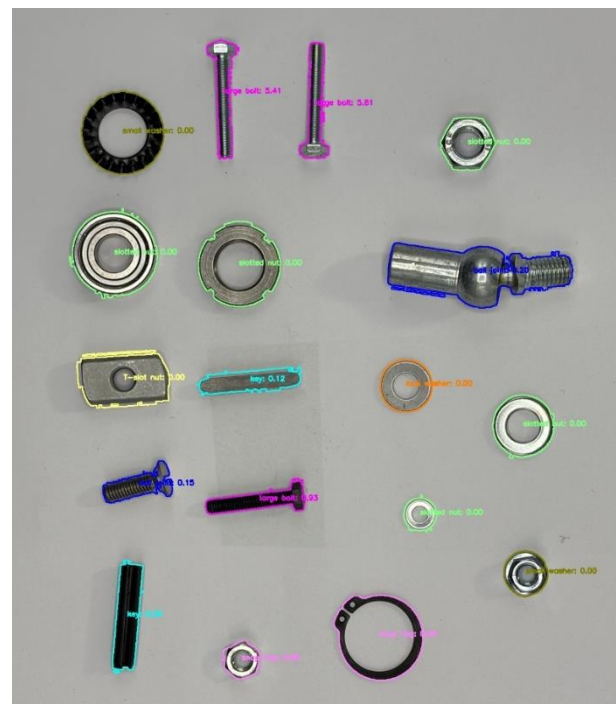
- **Overlappende objecten**

Hoewel dit niet ter sprake kwam in de conventionele methode, zou het in de toekomst nuttig zijn om overlappende objecten te kunnen herkennen en scheiden. We merkten dat wanneer objecten te dicht bij elkaar lagen of elkaar raakten, ze als één object werden gezien. In de testafbeeldingen konden we dit meestal handmatig filteren.

6.7 Conclusie

In dit project hebben we een code geschreven die met behulp van klassieke computer vision-technieken objecten kan herkennen en labelen. De applicatie gebruikt contourdetectie om de objecten te vinden en vergelijkt deze vervolgens met voorbeeldtemplates. De resultaten worden zowel visueel weergegeven als opgeslagen in een Excel-bestand, zodat er een duidelijk overzicht van onderdelen en locaties beschikbaar is.

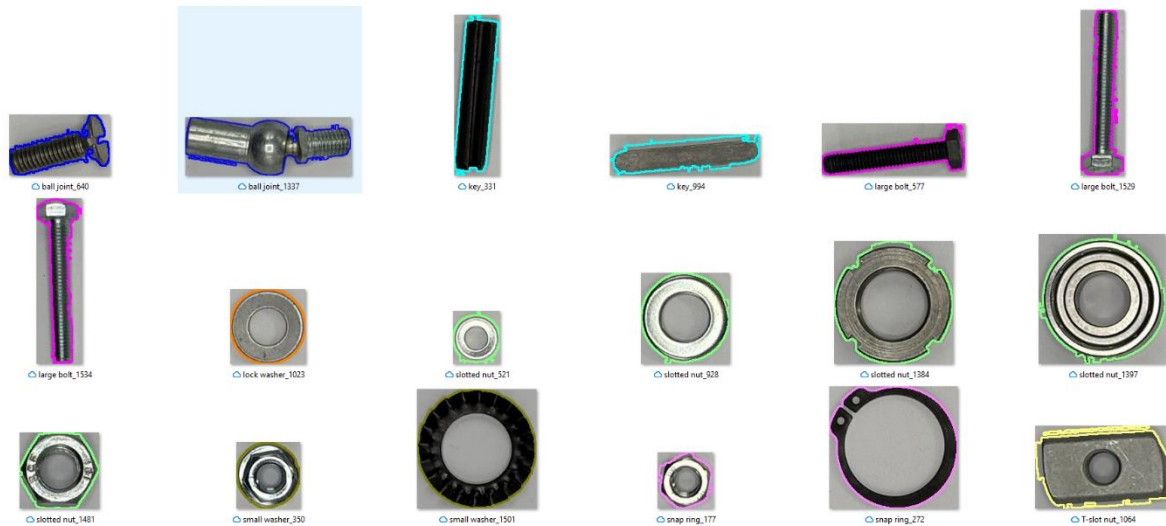
We hebben gebruikgemaakt van methoden zoals `cv2.findContours` voor het detecteren van de contouren en `cv2.matchShapes` om deze te vergelijken met de voorbeelden. Hoewel deze technieken eenvoudig en krachtig zijn, zijn er enkele beperkingen. Zo kan de nauwkeurigheid afnemen bij afbeeldingen met veel ruis of slecht belichte objecten. Ook kunnen overlappende



Eind resultaat conventioneel

objecten een probleem vormen voor de herkenning.

Een verbetering zou zijn om gebruik te maken van oppervlakteberekeningen om onderscheid te kunnen maken wanneer er twee dezelfde onderdelen zijn met een verschillende grootte. Momenteel lijken veel objecten qua oppervlakte sterk op elkaar, wat voor verwarring kan zorgen. Al met al werkt de applicatie goed voor duidelijke, goed gestructureerde afbeeldingen, maar er is zeker ruimte voor verbetering om het programma betrouwbaarder te maken.



Output code, screenshot van de gevonden onderdelen

Conclusie

In dit project zijn twee methoden onderzocht voor de detectie en classificatie van onderdelen: een conventionele computer vision-aanpak en een aanpak met behulp van deep learning. Door gebruik te maken van zowel klassieke technieken, zoals contourdetectie en vormvergelijking, als moderne deep neural networks (YOLOv8).

De conventionele methode, waarbij contouren van objecten werden vergeleken met templates, werkte goed voor eenvoudige objecten met duidelijke vormen. Deze methode bood een goede basis voor het detecteren van onderdelen in eenvoudige situaties, maar toonde beperkingen in objecten die qua vorm sterk op elkaar lijken. Hier zou een oppervlakteberekening een nuttige aanvulling kunnen zijn om vergelijkbare objecten beter te onderscheiden. Daarnaast kunnen toekomstige verbeteringen zich richten op het herkennen en scheiden van dezelfde objecten maar met een verschillende grote.

De deep neural network-methode, in dit geval YOLOv8, bleek een krachtiger en nauwkeuriger alternatief. Deze aanpak presteerde vooral goed in het herkennen van onderdelen in verschillende oriëntaties en in situaties met complexere objectopstellingen. Het trainen van een eigen dataset zorgde voor nauwkeurige resultaten, hoewel er meer experimenten nodig zijn om de nauwkeurigheid verder te verhogen, vooral in situaties met overlappende objecten.

Al met al hebben beide methoden hun eigen sterke en zwakke punten, en de keuze voor een methode hangt af van de specifieke toepassing. In toekomstig werk kan het combineren van beide technieken een mogelijke verbetering zijn om zo de prestaties van de applicatie verder te optimaliseren. Dit project laat zien dat computer vision een krachtig hulpmiddel is voor het automatisch detecteren en classificeren van onderdelen, wat erg hulpzaam kan zijn in productie fabrieken.

Bibliografie

facebookresearch. (n.d.). *SAM 2: Segment Anything in Images and Videos*. Retrieved from Github: <https://github.com/facebookresearch/sam2>

Fernández Villán, A. (n.d.). *Mastering OpenCV 4 with Python*.

Segment models. (n.d.). Retrieved from Ultralytics: <https://docs.ultralytics.com/tasks/segment/#models>

Ultralytics. (n.d.). *Instance Segmentation*. Retrieved from Ultralytics: <https://docs.ultralytics.com/tasks/segment/?h=instance+segmentat>

ChatGPT

Naast het gebruik van de voorbeeldcodes en het boek is er ook gebruikgemaakt van ChatGPT. ChatGPT werd ingezet om foutmeldingen op te lossen en bepaalde processen te versnellen. Daarnaast werd het gebruikt om specifieke functies op te zoeken, zoals hoe de export naar een Excel-bestand werkt. Zo heeft AI zeker een rol gespeeld in de code, waardoor we veel effectiever konden werken en een mooi resultaat hebben behaald. Ook heeft het geholpen bij het opsporen en verbeteren van taalfouten in het verslag.