Efterår
2022

Machine Learning

GRUPPE A
REBECKA THYSSEN & PJORT KAT

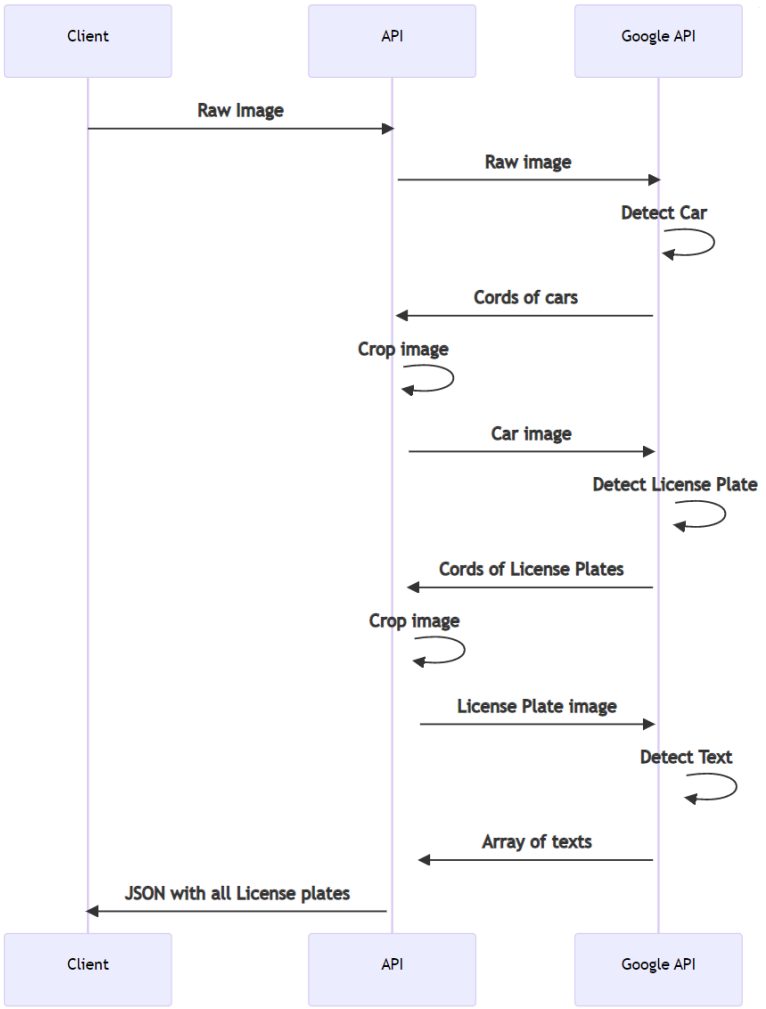
Indhold

Indledning

Dette projekt er fremstillet med udgangspunkt i faget Machine Learning, hvor der ønskes udarbejdet en applikation som anvender minimum 2 Machine Learning services.

Projektbeskrivelse

Vi ønsker at udvikle et REST API, hvor der med POST kan tilsendes et billede. APIet videresender billedet til Google Cloud AI (service 1), som skal kigge efter en bil på billedet. Findes der en bil, skal servicen returnere nogle koordinater, så vi kan beskære billedet og dermed gøre bilen mere tydelig.



Vi træner vores egen model med image object detection (service 2) for at kigge efter nummerplader på billedet af den bil, som APIet har modtaget fra service 1. Findes en nummerplade på billedet, returneres ligeledes her koordinater som vi kan bruge til at beskære nummerpladen.

Det tilpassede billede af nummerpladen sendes til en OCR (Optical Character Recognition) (service 3) og returnerer nummerpladens bogstaver og tal, som APIet returnerer som en streng.

En visualisering af kommunikationen mellem de forskellige services ses i billedet ovenfor.

Object detection

Object detection eller computer vision er hvor man mader et billede til en trænet machine learning model, og så returnere den med koordinaterne og klassifikationen, ud fra hvilke klasser model er trænet ud fra (Superviced learning).

OCR (Optical Character Recognition)

Optical Character Recognition er en trænet machine learning model som er specielt god til at finde tekst i billeder, så som scannet dokumenter. Forskellige OCR modeller kan være trænet til at kunne aflæse håndskrift eller kinesiske tegn, men de har typisk lettest ved at aflæse maskine skrevet ensformig tekst.

Valgte teknologier

Service 1 og 3 kan anvendes i Google Cloud AI. Modellen til image object detection, service 2, trænes i Google Cloud Vertex AI.

Logikken i vores eget API skrives i Python.

Anvendte biblioteker

For at få projektet til at køre optimalt, har vi anvendt en række biblioteker som beskrives i det følgende.

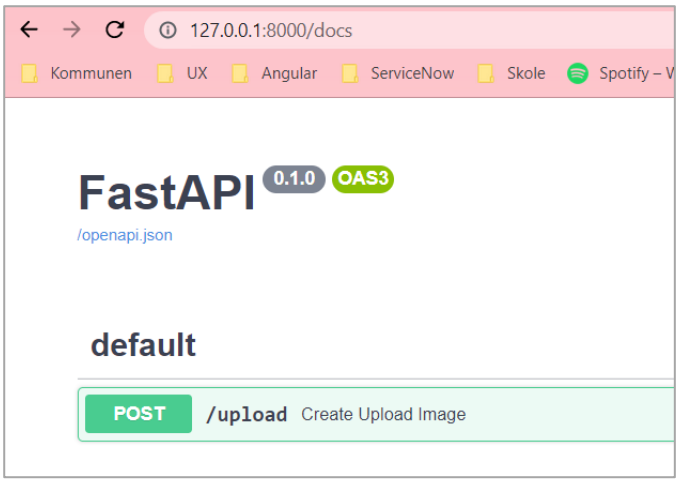
FastAPI

FastAPI er et web framework til opsætning af APIer med Python. Vi anvender dette for nemt og hurtigt at kunne få udstillet et API.

FastAPI installeres i projektet med kommandoen: `pip install fastapi`.

Uvicorn

Uvicorn anvendes sideløbende med FastAPI. Uvicorn fungerer som en asynkron server gateway interface-kompatibelt webserver (ASGI), hvis primære formål er at agere bindeled mellem browser og FastAPI. Uvicorn sørger således for at requests fra browseren sendes til FastAPI. En stor fordel ved dette er, at når først denne webserver startes, vil alle ændringer vises med det samme i browseren uden at den skal loades.



Uvicorn installeres i projektet med kommandoen: `pip install "uvicorn[standard]"`.

Projektet opstartes således med kommandoen `uvicorn main:app --reload`. Vi kan herefter tilgå Swagger i vores FastAPI ved at tilføje "/docs" til vores URL (eksempel til højre).

Figur 1: Adgang til FastAPI med Swagger

OpenCV-python

OpenCV-python er et bibliotek til Python som håndterer billedbearbejdelse. Vi anvender biblioteket til både at beskære vores billeder, men også til at fremstille firkanter på fokusområder som der ønskes genkendelse på.

Biblioteket installeres i projektet med kommandoen: `pip install opencv-python`.

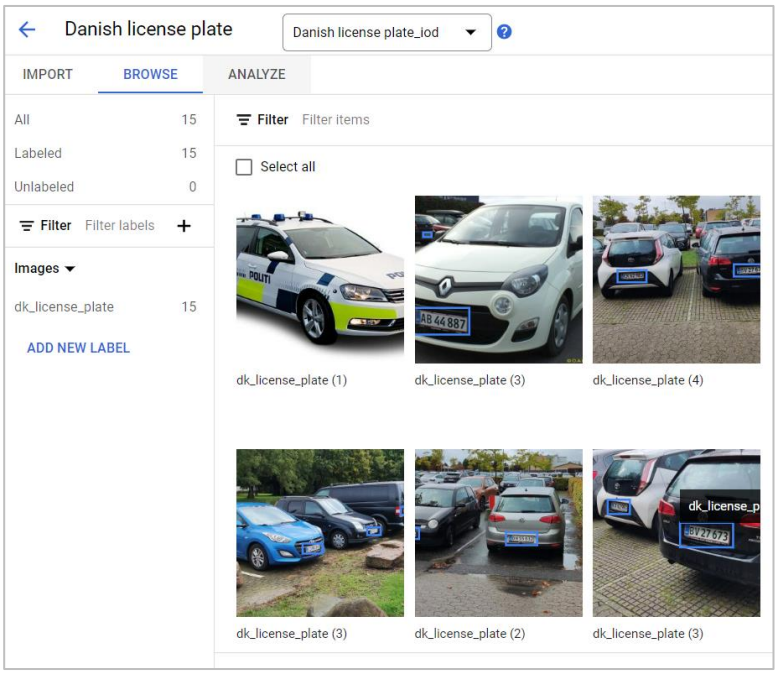
Google Cloud Vision

For at integrere vores kode til Google Cloud Vision APIet installeres dette bibliotek. Vi anvender det både til at genkende teksten i de fundne nummerplader, men også til at genkende de biler som vi sender billeder af.

Biblioteket installeres med kommandoen: `pip install google-cloud-vision`.

Træning af service 2

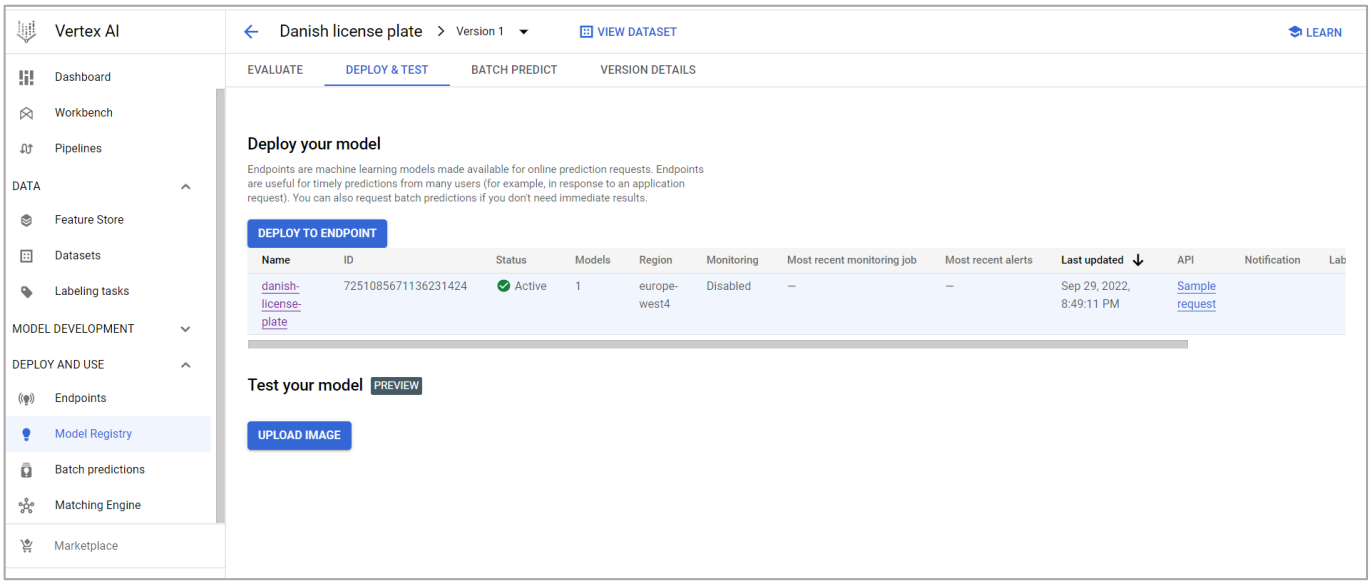
Som første del af projektet er vi startet med at træne vores model til image object detection. Vi har valgt at træne vores egen model, da det ikke er lykkedes os at finde en allerede trænet model til aflæsning af danske nummerplader.



Figur 2: Træning af model i Vertex AI.

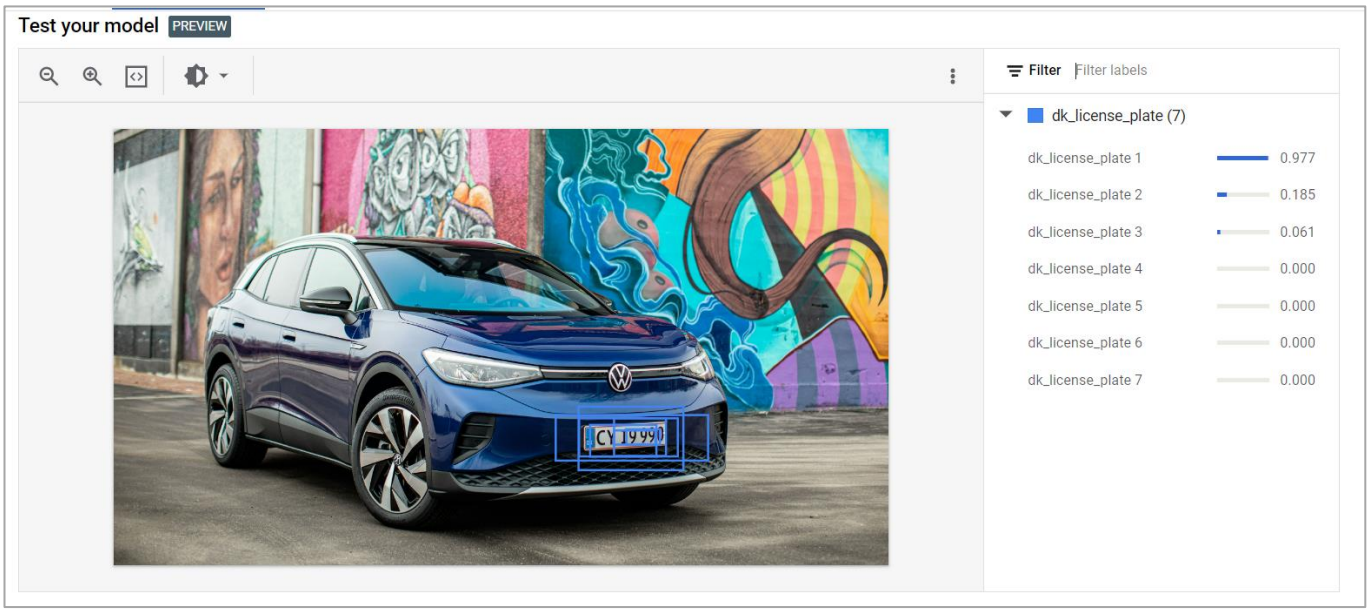
Træningen er sket i Google Cloud Vertex AI, hvor vi uploader billeder af biler med danske nummerplader og labeler disse med indtegning af bounding boxes (billedet ovenfor). Herefter kan vi træne vores egen model til at aflæse danske nummerplader med supervised learning. Vi har anvendt en blanding af billeder af biler med danske nummerplader fundet på Google, samt taget vores egne. For tidsbesparelse har vi kun anvendt 15 billeder til træning, men en mere præcis model ville kunne udvikles med mere træningsdata.

Når træningen er gennemført, kan denne deployes, som giver et endpoint som kan tilgåes:



Figur 3: Deployment af endpoint i Vertex AI

Efter succesfuldt deployment vil endpointet være aktivt, og der kan uploades et billede som servicen vil analysere på. Denne vil herefter vise sine bud på bounding boxes efter nummerpladens placering.



Figur 4: Testresultat efter træning af service 2.

Opstilling af API

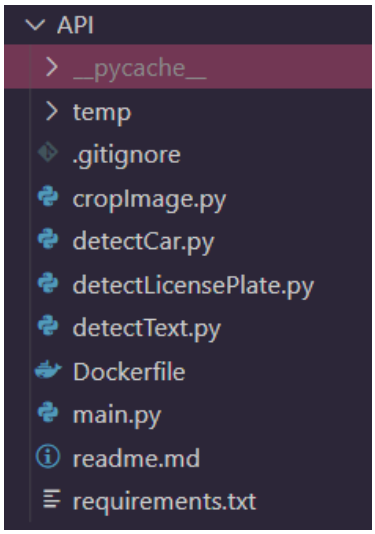
Vores API er opstillet som et projekt skrevet i Python, hvor vi i vores main.py-fil udstiller et enkelt POST-endpoint. Dette endpoint samler vores logik på tværs af projektet og sørger for at kommunikationen mellem de forskellige services finder sted:

```python
@app.post("/upload")
async def create_upload_image(image: UploadFile = File(...)):
    destination_file_path = "temp/" + image.filename
    async with aiofiles.open(destination_file_path, 'wb') as out_file:
        while content := await image.read(1024):
            await out_file.write(content)

    crop_cars_paths = crop_cars(destination_file_path)

    license_plates = []
    count = 1
    for path in crop_cars_paths:
        license_plate = detect_license_plate(path, count)
        count += 1
        if license_plate is not None:
            license_plates.append(license_plate)
    print(license_plates)

    texts = []
    for license_plate in license_plates:
        text = detect_text(license_plate)
        if text is not None:
            texts.append(text)

    return {"license-plates": texts}
```

Figur 5: main.py

Logikken med kommunikation til de forskellige services er således delt op i 4 filer for at holde metoderne adskilt:

- detectCar som har til formål at finde biler på det billede som sendes afsted til APIet – gives i koordinater
- cropImage som beskærer billedet baseret på koordinater og genererer et nyt med navnet ”Cropped_car” + nummeret
- detectLicensePlate som sender det beskårne billede til vores trænede API for at tjekke efter nummerplader
- detectText som sender den beskårne nummerplade til tekstgenkendelse i service 3

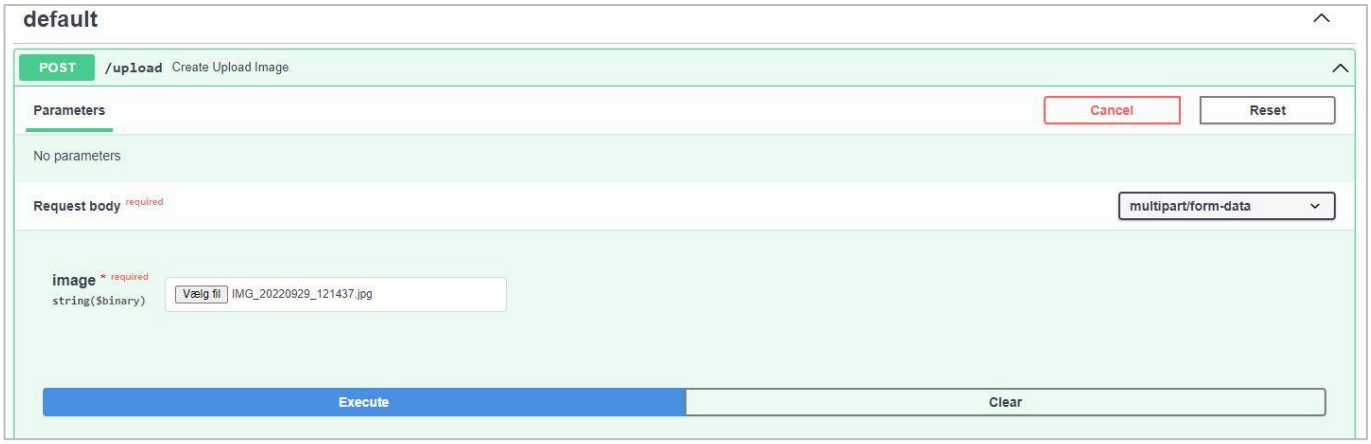


Figur 6: Vores APIs mappestruktur

Disse metoder beskrives mere dybdegående i det følgende.

Trin 1: Find og beskær biler på modtaget billede (service 1)

Når applikationen kører, kan vi via Swagger eller Postman uploade et testbillede af en bil:



Figur 7: POST-endpoint i Swagger

Ved eksekvering rammer vi vores endpoint ”/upload”, hvor det er muligt at tage det indkomne billede ind.

Billedet læses og gemmes i den ønskede sti:

```python
@app.post("/upload")
async def create_upload_image(image: UploadFile = File(...)):
    # Set the destination for the incoming image
    destination_file_path = "temp/" + image.filename
    # Read the incoming image
    async with aiofiles.open(destination_file_path, 'wb') as out_file:
        while content := await image.read(1024):
            await out_file.write(content)

    # Send the image above to the crop_cars method
    # Returns the cropped cars as a 'path'
    crop_cars_paths = crop_cars(destination_file_path)
```

Figur 8: Main.py

Herefter kaldes crop_cars-metoden (i vores detectCar-fil), med det indkomne billede:

```python
from google.cloud import vision_v1p3beta1 as vision
client = vision.ImageAnnotatorClient()

# Read the incoming image
with open(path, 'rb') as image_file:
    content = image_file.read()  # content in bytes
# sdk to create the picture google wants
image = vision.Image(content=content)

objects = client.object_localization(
    image=image).localized_object_annotations  # sends the final picture to the google client

count = 1
paths = []

for object_ in objects:
    # skip object if it is not a Car in object.name
    if "Car" not in object_.name:
        continue

    cords = []
    for vertex in object_.bounding_poly.normalized_vertices:
        # save the coords where a car has been found
        cords.append((vertex.x, vertex.y))

    paths.append(crop_image(path=path, x1=cords[0][0], x2=cords[1][0],
                 y1=cords[0][1], y2=cords[2][1], name="Cropped_car", suffix=count))

    count += 1

return paths  # the paths to the new cropped car images
```

Figur 9: detectCar.py

Her ses det at billedet læses, genereres i et ønsket format som passer til Googles API og herefter sendes afsted til Google-servicen. Servicen vil returnere en række fundne objekter med forskellige annotations (f.eks. ”Person”, ”Car”, ”Tire”, ”Dog” osv.), og derfor sorterer vi på objekterne, da vi kun er interesserede i biler. I et midlertidigt tomt array, cords, gemmes koordinaterne fra bilerne. Disse koordinater bruges til at beskære billedet, som gøres i metoden crop_image:

```python
def crop_image(path, x1, x2, y1, y2, name, suffix):
    # Create a cropped car image from above coords, name and number
    import cv2

    img = cv2.imread(path)

    height, width, channels = img.shape

    crop = img[int(y1*height):int(y2*height),
               int(x1*width):int(x2*width)]

    pathstring = "temp/" + name + str(suffix) + ".jpg"

    cv2.imwrite(pathstring, crop)  # save the new cropped image

    return pathstring  # the new cropped image
```

Figur 10: cropImage.py

Her anvendes biblioteket cv2 til at beskære billedet efter koordinaternes højde og bredde. En ny sti til det nye, beskårne billede genereres med navnet ”Cropped_car” og nummeret på den beskårne bil (parset til metoden), og det nye, beskårne billede gemmes på den nye sti, som returneres.

Alle stierne til de nye, beskårede biler gemmes i et array ”paths”, og vi har nu en mappe med billeder af alle de biler som service 1 fandt på det første, originale billede.

Trin 2: Find og beskær nummerplader på allerede beskåret billede af bil (service 2)

I vores endpoint ses det efterfølgende at vi kalder næste metode, detect_license_plate:

```
license_plates = []
    count = 1
    for path in crop_cars_paths:
        license_plate = detect_license_plate(path, count)
        count += 1
        if license_plate is not None:
            license_plates.append(license_plate)
```

Figur 11: main.py

Det er her, at vi skaber adgang til vores trænede endpoint:

```
def detect_license_plate(path, suffix):
    from cropImage import crop_image

    predicts = predict_license_plate(
        project="172246408209",
        endpoint_id="7251085671136231424",
        location="europe-west4",
        filename=path,
        api_endpoint="europe-west4-aiplatform.googleapis.com",
    )

    # look in the first prediction - is there a bounding box?
    if predicts[0]["bboxes"].__len__() > 0:
        return crop_image(path=path, x1=predicts[0]["bboxes"][0][0],
        y1=predicts[0]["bboxes"][0][2], x2=predicts[0]["bboxes"][0][1],
        y2=predicts[0]["bboxes"][0][3], name="Cropped_license_plate", suffix=suffix)
    # bboxes holds the new coordinates to the license plate, which will be sent to crop_image again
    return None
```

Figur 12: detectLicenseplate.py

Hvor vi via kald til en metode, ”predict_license_plate”, opstillet af Google, kan sende informationer om vores endpoint og det billede vi ønsker analyseret til. Findes der en nummerplade på det indsendte billede, sender vi dette til vores crop_image-metode igen, som på baggrund af nummerpladens koordinater bliver beskåret præcist.

Dette genererer et nyt billede som bliver placeret i vores temp-mappe.

Trin 3: Aflæs tekst på nummerplade (service 3)

Det nye billede af den beskårne nummerplade sendes til service 3, som har til formål at tekstgenkende:

```python
def detect_text(path):
    """Detects text in the file."""
    from google.cloud import vision
    import io
    client = vision.ImageAnnotatorClient()

    with io.open(path, 'rb') as image_file:
        content = image_file.read()

    image = vision.Image(content=content)

    response = client.text_detection(image=image)
    texts = response.text_annotations

    if response.error.message:
        raise Exception(
            '{}\nFor more info on error messages, check: '
            'https://cloud.google.com/apis/design/errors'.format(
                response.error.message))

    if len(texts) > 0:
        license_plates = []
        license_plate_texts = texts[0].description.split('\n')

        for text in license_plate_texts:
            if len(text) > 3:
                if len(text) <= 9:
                    license_plates.append(text)

        return license_plates

    return None
```

Figur 13: detectText.py

Her ses det at billedet sendes til tekstgenkendelsesværktøjet. Forefindes en tekst på billedet, ilægges dette et array, hvori vi kan manipulere formatet af teksten og returnere nummerpladen i det strengformat som vi ønsker. Dette ses ved den respons vi får retur i Swagger, når vi har sendt postet vores billede i vores endpoint:

Figur 14: Response fra Swagger efter POST

Konklusion

Vi kan konkludere at vi nåede i mål med vores projekt ud fra projektbeskrivelsen, og vi kan således via tre forskellige ML-services processere billeder af biler, beskære dem, aflæse deres nummerplader og herefter få returneret informationen om disse i streng-format.

Det er dog vigtigt at påpege at anvendelsen af Vertex AI på ingen måde er gratis. At træne sin egen model hos Google koster et større engangsbeløb afhængig af længden og kvaliteten af træningen, og hertil kommer en timepris for at have sit endpoint til modellen udstillet.

De andre services og biblioteker har været gratis, dog med en begrænsning på 1000 requests. Man kan derfor med fordel undersøge om der kan anvendes et alternativ til træning af egen model.

Perspektivering

Ønsker man at anvende en gratis løsning til at få samme resultat, har vi undersøgt YOLO og PaddleOCR. Begge er open source-teknologier og kan derfor anvendes uden betaling.

YOLO står for ”You Only Look Once” og anvendes til object detection. Dermed kan denne erstatte service 1 og 2. Til YOLO følger ligeledes YOLO-label, som er et værktøj til at indtegne bounding boxes, når man skal træne sin egen model.

PaddleOCR anvendes til text detection, og kan derfor erstatte service 3. Dog er vores erfaring at billedet der laves text detection på skal være simpelt og ikke have for mange farver/objekter, da det forstyrrer aflæsningen og giver et mindre korrekt resultat.