Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap

Institutt for datateknologi, elektroteknologi og realfag

Bildekvalitetsdetektor for dekk på kjøretøy i bevegelse

Image Quality Detector for tires on vehicles in motion

Systemdokumentasjon

Versjon <2.0>

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
<30/04/24>	<1.0>	Fylt ut det meste av kapitlene	Bjørn Ellingsen Øyvind Holter Håkon Lervåg
<11/05/24>	<2.0>	Oppdatert mange kapitler	Håkon Lervåg Bjørn Ellingsen Øyvind Holter

INNHOLDSFORTEGNELSE

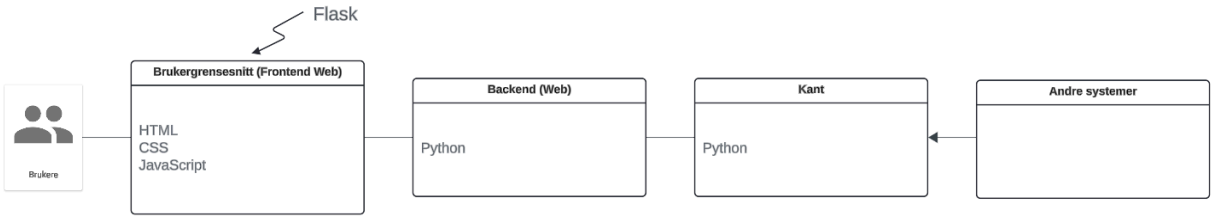
1 INNLEDNING

Dette dokumentet har til hensikt å gi en beskrivelse av systemet, inkludert designet og implementasjonen. Vi vil dekke aspekter som arkitektur, klassediagrammer, databasemodeller, prosjektstruktur, servertjenester, sikkerhet, testing, oppstart og kjøring av nettsiden vår, samt dokumentasjon av kildekoden.

2 ARKITEKTUR

Systemet vårt er bygget opp som en pipeline. Figur 2.1 viser de viktigste komponentene i system, som inkluderer brukere, brukergrensesnitt, backend (web), kant og andre systemer. I punktene nedenfor forklarer vi hvordan arkitekturen til figur 2.1 ser ut:

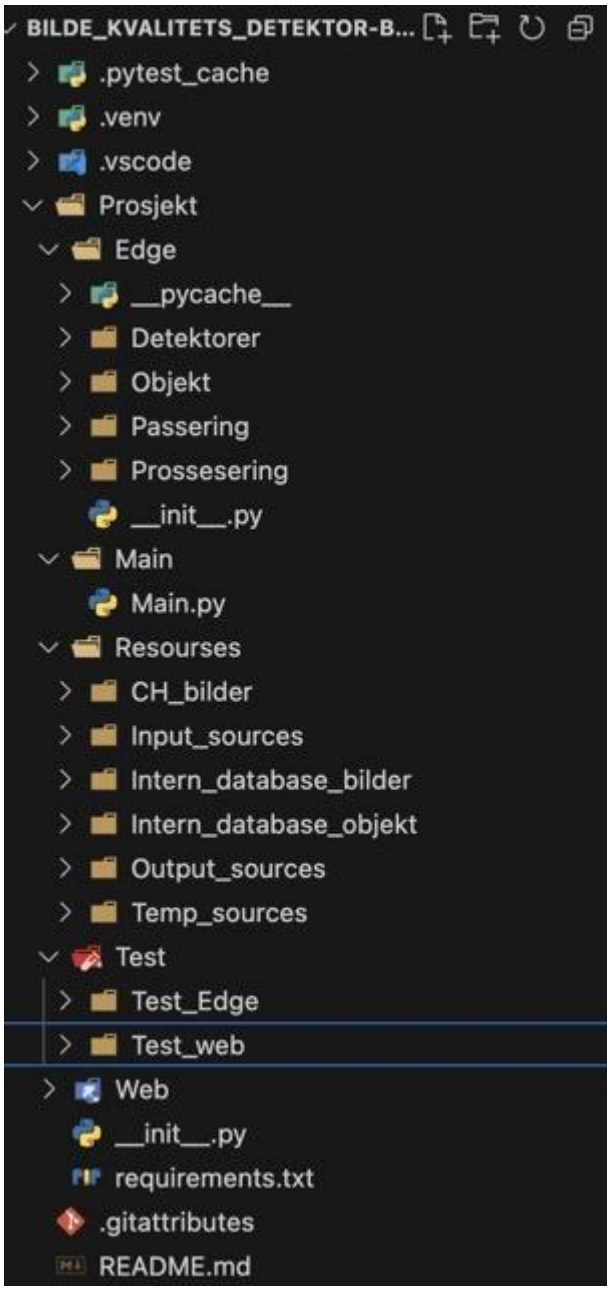
- Andre systemer: De systemene i Counting Hero som identifiserer det beste bildet av dekket på bilen og gir kanten vår et beskjært bilde som er klart for prosessering. Systemet er ikke integrert med Counting Hero sine systemer, så vi simulerer dette ved å beskjære bildene selv. Kapittel 4.2.1 i rapporten forklarer nærmere hvorfor vi beskjærer bildene som vi bruker.
- Kant: Dette er stedet hvor vi utfører prosessering av bildene som vi mottar fra kameraene til Counting Hero.
- Backend (Web): En serviceklasse som henter objekter og bilder som skal vises på nettsiden.
- Brukergrensesnitt (Frontend Web): Nettsiden viser resultatene av deteksjonen på bildene.
- Brukere: Brukere sjekker om det oppdages feil i bildekvaliteten ved å utføre søk på nettsiden.



Figur 2.1 Systemarkitektur

3 PROSJEKTSTRUKTUR

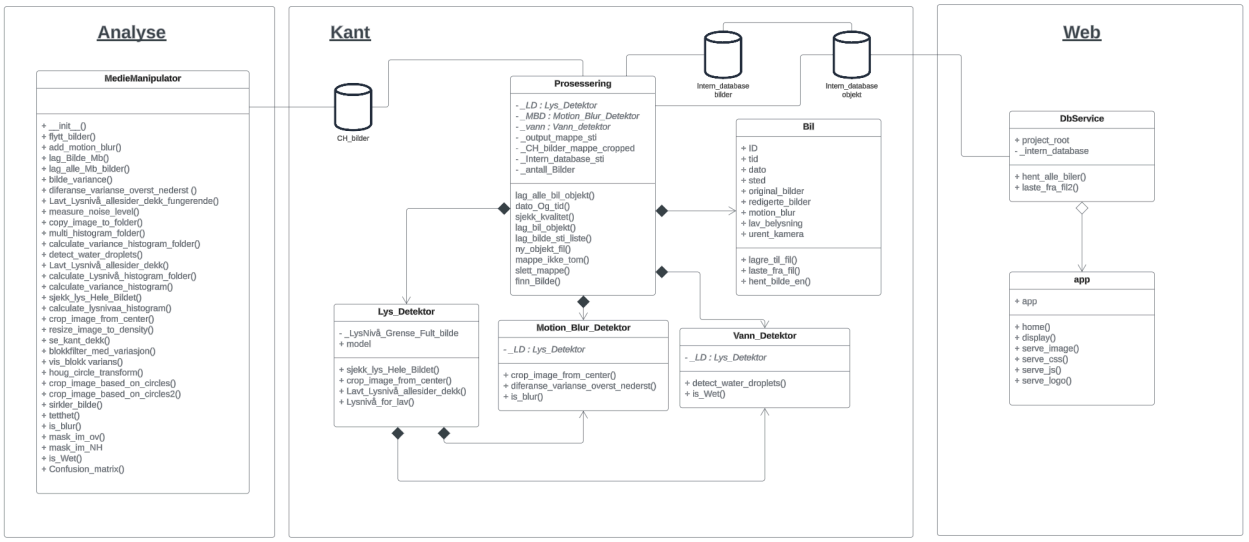
Figur 3.1 viser katalogstrukturen for prosjekt i VSCode [1]. Under "Prosjekt" mappen finner du koden vår, som er organisert i "Edge" (Kant), "Resources" (Ressurser), "Test" og "Web". I tillegg har vi en README-fil for installasjon og kjøring, som beskrives nærmere i kapittel 8.



Figur 3.1 Katalogstruktur

4 KLASSEDIAGRAM

Vi har utarbeidet et klassediagram for systemet som beskriver klassene, attributtene, funksjonene og forholdet mellom dem. Klassediagrammet i figur 4.1 er strukturert på en litt anderledes måte enn domenemodellen og er inndelt i analyse, kant og web. I figur 4.1 presenteres klassediagrammet for systemet. Store deler av klassediagrammet ligner på domenemodellen i kravdokumentet. Klassediagrammet inneholder en del endringer som følge av en del avgrensninger som forklares i kap 2.2 i rapporten. En detalj å merke seg i figur 4.1 er at alle variabler med "_" foran seg indikerer at de er private. Andre variabler er ellers offentlige.



Figur 4.1 Klassediagram

4.1 Analyse i "MediaManipulator"

Figur 4.2 viser klassen "MedieManipulator", som spilte en viktig rolle i testingen av detekorene, samt for å finne de passende terskelverdiene. I denne klassen har vi mange funksjoner for å teste bildene som ligger i "CH_bilder". Denne klassen blir blant annet brukt til å generere histogrammer av dekkene fra bildene vi mottok fra Counting Hero. Histogrammene som genereres av denne klassen vil være relevante i kapittel 4.2 og 4.3 i rapporten når vi analyserer våre resultater og evaluerer om vi har identifisert tilstrekkelige terskelverdier.

"MedieManipulator" klassen som vist i figur 4.2 inneholder en stor del av alle løsningene og metodene vi har undersøkt. Noen av funksjonene i denne klassen har ikke gitt tilfredsstillende resultater og er derfor forkastet. Metodene er derfor ikke brukt i fastsettelsen av terskelverdiene, men er likevel metoder som vi har testet.

Figur 4.2 Klassediagram - Analyse

4.2 Kant

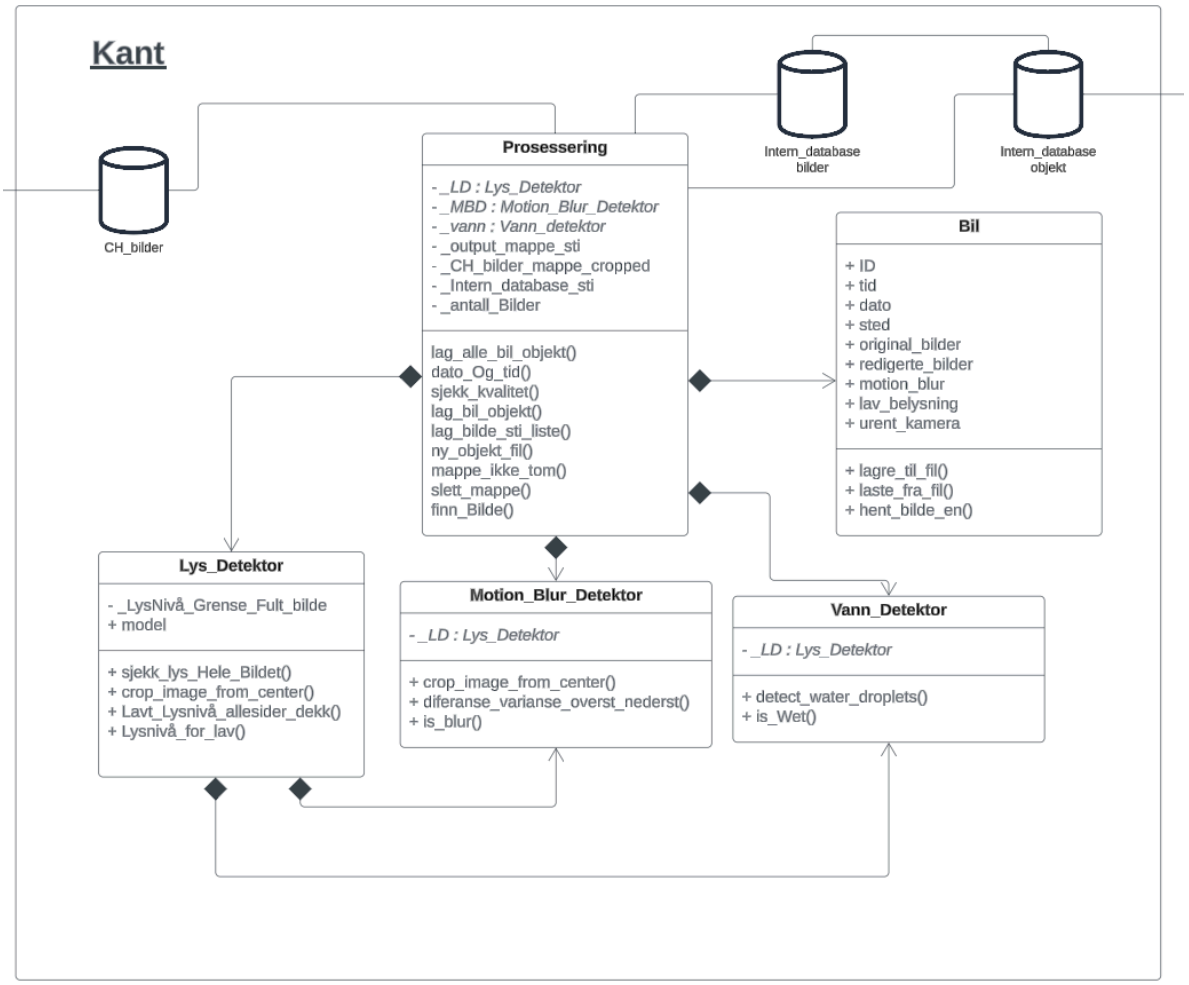
Fra figur 4.3 ser vi at kanten vår hadde muligheten til å prosessere bilder fra en mappe som heter “CH_bilder”. Disse bildene ble gitt til oss av oppdragsgiveren en stund inn i prosjektperioden. Mappen "CH_bilder" inneholdt bildene som vi fikk fra Counting Hero sine kameraer, og er lagt inn i mapper markert med tid og dato. Et eksempel på et mappenavn kunne være "D20230324_T134257", hvor alt etter "D" indikerer dato og alt etter "T" indikerer tiden. Disse mappene inneholder ett bilde per dekk på kjøretøyet.

En utfordring vi støttet på utover i prosjektperioden var at vanndråper utøvde et problem i bildene. Ettersom mange bilder fra oppdragsgiver inneholdt mye vann, ble en vanndråpedetektor aktuell for den endelige løsningen senere i prosjektet. Vi endte opp med å lage en egen klasse "Vann_Detektor", som blir forklart nærmere i kapittel 4.3 i rapporten.

Ettersom vi håndterer en mengde data, er det viktig å opprettholde orden på de ulike typene bilder både før og etter behandling. Dette håndteres av klassen "Prosessering", som også

fungerte som et bindepunkt for systemet vårt. "Prosessering" tar bildene den mottar og kjører detektorene på dem for å sjekke etter feil i bildekvaliteten. Klassen bruker detektorene til å sjekke bildene for lysnivå, vått dekk og "motion blur". Som vist i figur 4.3, er "Motion_Blur_Detektor" og "Vann_Detektor" avhengige av "Lys_Detektor" fordi de trenger lysnivået i bildet for å bestemme terskelverdien de skal bruke. Grunnen til at disse detektorene er avhengige av lysnivået blir forklart i kapittel 4.3 i rapporten.

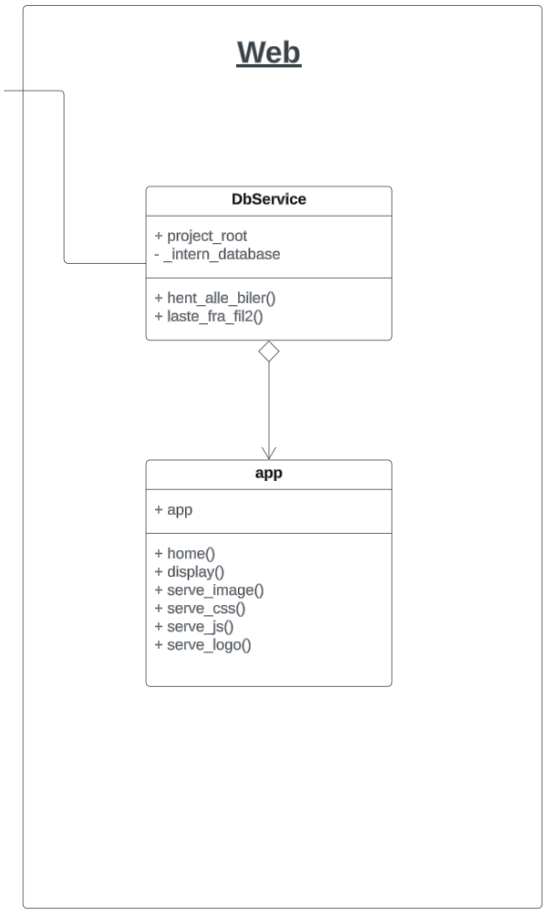
"Prosessering" oppretter bilobjekter ved hjelp av klassen "Bil". Disse bilobjektene blir deretter lagret i "Intern_database_objekt". Disse objektene inneholder viktig informasjon som vises frem på nettsiden. "Prosessering" lagrer så de bildene som har blitt prosessert i en mappe ved navn "Intern_database_bilder". Bilobjektene som er lagret i "Intern_database_objekt" inneholder variabler med en referanse til hvor bildene er lagret i "Intern_database_bilder". Etter at bildene og objektene er lagret, er de klare til å vises frem på nettsiden. En nærmere beskrivelse av hvordan bilobjektene fungerer, samt hvorfor vi har to separate mapper, vil bli gitt i kapittel 4.1.1.1 i rapporten.



Figur 4.3 Klassediagram - Kant

4.3 Web

Figur 4.4 viser webdelen av klassediagrammet. Denne figuren har ingen avvik domenemodellen og står forklart



Figur 4.4 Klassediagram - Web

5 DATABASEMODELL

Vi har ikke konfigurert en database i systemet, men vi har forsøkt å legge til rette for implementasjon av dette. Som beskrevet i kap 2.2 i rapporten, benytter Counting Hero for tiden et databasesystem som vi valgte å ikke integrere i vår utvikling, ettersom dette ikke var relevant for å oppnå målet for oppgaven. Derfor valgte vi å lagre all informasjonen lokalt. Vi har opprettet en serviceklasse i web-delen av prosjektet, som er ment å fungere som et utgangspunkt for implementeringen. I den nåværende koden vår, henter DbService-klassen bare objekter og bilder fra andre mapper lokalt på maskinen.

6 SERVER-TJENESTER

Vi bruker Flask [2], et Python-bibliotek, for å sette opp nettsiden vår. Nettsiden vår er en applikasjon som kjører lokalt på maskinen og krever ingen database eller server som forklart i forrige kapittel. Utenom dette er ikke kapittelet om server-tjenester relevant for prosjektet.

7 SIKKERHET

Slik vi omtalte i visjonsdokumentet har vi håndtert sensitive data slik som blant annet bilder av personbiler. Derfor har vi vært nødt til å være forsiktig med informasjonen vi bruker. Gruppen måtte blant annet passe på at ingen bilder viser registreringsnummer, fotograferinger av personer eller lignende. Utenom dette er ikke kapittelet om sikkerhet relevant for prosjektet.

8 INSTALLASJON OG KJØRING

8.1 Installasjoner og kjøring

For å kjøre prosjektet må man laste ned Python versjon 3.11.7 [3].

Videre må vi sette opp python-miljøet i VSCode:

- Trykk på kommandolinjen på toppen i VSCode
- Skriv: > Python: Create Environment
- Velg Venv → Python 3.11.7 → requirements.txt → Ok

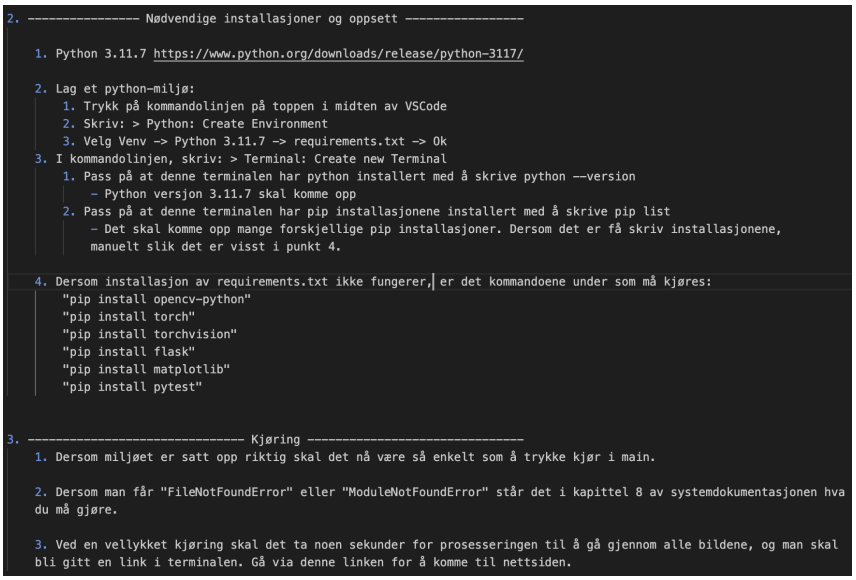
Requirements.txt filen inneholder alle pip installasjonene vi trenger for å kjøre prosjektet. “pip” blir brukt i Python til å installere pakker i terminalen og blir inkludert i Python 3.4 eller nyere versjoner [4].

Videre setter vi opp terminalen for kjøring:

- Trykk på kommandolinjen på toppen i VSCode
- Skriv: > Terminal: Create New Terminal

Pass på at denne terminalen har python installert med å skrive python --version. Dersom det er satt opp riktig skal python versjon 3.11.7 vises. Pass også på at pip installasjonene er installert i terminalen med å skrive pip list. Dersom det blir skrevet ut mange installasjoner er det korrekt. Nå skal man kunne gå inn i main og kjøre programmet. Pass på at programmet kjører i terminalen hvor “venv” er aktivert. Når main kjøres blir prosesseringen utført på bildene og linken til nettsiden printes ut i terminal. Alternativt er det mulig å kjøre klassen "Video_Slicer" i main. Som forklart i kravdokumentet, kapittel 3, vil denne klassen kjøre med en testvideo. Forklaringen på hvordan man kjører "Video_Slicer" er beskrevet i kommentarene i main.

Figur 8.1 er en “README” fil som ligger i prosjektet som også forklarer hvordan man installerer nødvendige installasjoner og kjører prosjektet.



Figur 8.1 README fil

8.2 Avhengigheter

Nedenfor forklarer vi avhengighetene som du må installere med "pip" for at prosjektet skal fungere:

"pip install …"	Forklaring av avhengighetene
opencv-python	OpenCV [5] er et kraftig åpen kilde kodebibliotek for bildebehandling og datavisjon.
torch	Torch [6] er et maskinlæringsbibliotek som er spesielt fokusert på dyp læring. Det gir støtte for tensorbehandling og beregninger på GPU (grafikkprosessor). torch er kjernen i PyTorch-rammeverket [7].
torchvision	Torchvision [7] er en del av PyTorch-økosystemet[6] og gir verktøy for datavisjon, inkludert datasett, transformasjoner og modellarkitekturer. Den inneholder også enkle funksjoner for bildebehandling.
flask	Flask [2] er et mikro-webrammeverk for Python som lar deg enkelt lage webapplikasjoner. Flask gir verktøy og struktur for å bygge webapplikasjoner og RESTful API-er.
matplotlib	Matplotlib [9] er et populært bibliotek for å lage grafer og plott i Python. Det gir et enkelt grensesnitt for å lage forskjellige typer grafer, inkludert linjeplott, histogrammer, søylediagrammer og mer.
pytest	Pytest [10] er et rammeverk for enhetstesting i Python. Det gjør det enkelt å skrive og kjøre tester, og det gir omfattende funksjoner for testoppdagelse, -organisering og -rapportering.

8.3 Feilhåndtering

8.3.1 "FileNotFoundError"

FileNotFoundError: No such file or directory:'Prosjekt\\Resourses\\Intern_database_objekt\\bild_id_1.pkl'

Blir man møtt med denne feilkoden har mest sannsynlig mappen Intern_database_objekt ikke kommet med prosjektet. Dette er en rask fiks:

1. I mappestrukturen gå på Prosjekt -> Resourses
2. Høgreklikk på Resourses og velg New Folder.
3. Sett navnet på mappen til Intern_database_objekt
4. Man skal nå kunne kjøre prosjektet fra main

8.3.2 “ModuleNotFoundError: No module named ‘cv2’ ”

Dersom man blir møtt med error-en over, har mest sannsynlig ikke python-miljøet blitt aktivert. Dette har bare hendt med oss på Windows, så vi kan ikke garantere at det fungerer på Mac / Linux. Gjør følgende:

Dersom man vil gjøre det helt fra begynnelsen:

1. Dersom du har en .venv fil - slett denne.
2. I søkebaren på toppen i VSCode (kommandobaren) skriv “> Python: Create Environment”
    a. Velg “Venv” → Python 3.11.7 64-bit → og trykk på “Prosjekt/requirements.txt”
3. I søkebaren på toppen i VSCode (kommandobaren) skriv “> Terminal: Create New Terminal”.
    a. I den nye terminalen, skriv: python --version. Dersom Python versjon 3.11.7 kommer opp er det riktig. Skriv så “pip list”. Dersom det kommer opp mange installasjoner er det også riktig.
    b. Man skal nå kunne gå inn i main og kjøre den. Pass på at main kjører i samme terminal hvor python og pip installasjonene er installert.
    c. Fungerer det ikke, gå til steg 4.

Ellers skal dette fungere greit:

4. Dersom man ikke får lov til å installere pip installasjonene må vi aktivere miljøskriptet.
5. I terminalen i VSCode skriv cd .\.venv\Scripts
6. Skriv så .\activate
7. Skriv deretter “R” i terminalen. Dette skal fyre i gang miljøet, og det skal nå stå (.venv) foran filstien man er i.
8. Gå deretter tilbake til filstien i det ytterste punktet i prosjektet med cd ..\Users\....\Bilde_Kvalitets_Detektor-BKD- (Her må prikkene bli byttet ut med din sti.)
9. Nå skal man kunne gå inn i main og kjøre programmet. Pass på at programmet kjører i terminalen hvor “venv” er aktivert.

9 DOKUMENTASJON AV KILDEKODE

Vi har dokumentert koden ved hjelp av docstrings, som er innebygde dokumentasjonsstrenger i Python. Vi har fulgt retningslinjene i PEP 257 [11] for å skrive gode doc-strings, slik at koden vår blir lettleselig og oversiktlig.

For å dokumentere koden vår, har vi benyttet autoDocstring [12]. Dette er en innebygd utvidelse i VSCode som automatisk genererer en generisk mal som man kan fylle inn for hver funksjon slik vi ser i figur 9.1.

```
def is_blur(self,image_path,Lysverdi,verdi = False):
    """_summary_

    Args:
        image_path (_type_): _description_
        Lysverdi (_type_): _description_
        verdi (bool, optional): _description_. Defaults to False.

    Returns:
        _type_: _description_
    """
```

Figur 9.1: Den generiske malen som autoDocstring gir oss

10 KONTINUERLIG INTEGRASJON OG TESTING

Brukertesting:

I løpet av prosjektet hadde vi regelmessige møter med oppdragsgiveren, hvor vi mottok tilbakemeldinger angående prosjektets fremgang. Dette bidrog til at vi kunne få luftet idéer og fremgangsmåter, samt få profesjonell assistanse med teoriområder som kunne være ekstra vanskelige. Det var også en fin måte for oppdragsgiver å få holde seg oppdatert i prosjektet.

Oppdragsgiveren hadde ikke direkte tilgang til prosjektet vårt utenom møtene, ettersom alt var lokalt lagret på vår maskin. Imidlertid ga oppdragsgiveren muntlige tilbakemeldinger om hva vi kunne gjøre for å forbedre løsningen vår. Disse tilbakemeldingene var svært nyttige for å gjennomføre deteksjonen på en bra måte og fremheve viktige aspekter på nettsiden vår.

Enhetstesting:

I løpet av prosjektet har vi benyttet enhetstesting i VSCode ved hjelp av Pytest [10]. Et enkelt-å-bruke testrammeverk med et rikt økosystem av plugins, som gjør det til et fleksibelt verktøy for testing [13]. Vi brukte enhetstesting for å sikre at enkelte komponenter i programmet fungerte som forventet. Resten av programmet gikk på testing av metoder for detektorene, noe som “MedieManipulator”-en hadde ansvar for å evaluere. Enhetstesting ble derfor bare brukt til å evaluere noen få gjenværende aspekt av koden. Vi opplevde enhetstesting som både effektiv og rask, og dermed passende for testing av kodebiter. Mer om hvordan man kjører enhetstestene står beskrevet under.

Testing i “MedieManipulator”:

En god del av funksjonene, spesielt i klassen “MedieManipulator”, som står beskrevet i kapittel 4.1.1 i rapporten, er laget for å undersøke og kunne evaluere resultatene fra deteksjonene, samt hvor aksepterte de er. Funksjonene i “MedieManipulator” har blant annet blitt brukt til å lage histogrammer. Disse ble aktivt brukt for å evaluere fremgangen vår i å nærme oss mer pålitelige og effektive terskelverdier over tid. Klassen har vært viktig for å komme frem til de resultatene vi fikk, samt visualisere dem.

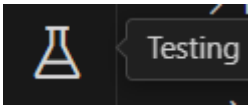
Kjøring av enhetstester på Windows:

I prosjektet kan man åpne og kjøre testfilene individuelt ved å gå inn i testklassene og trykke kjør.

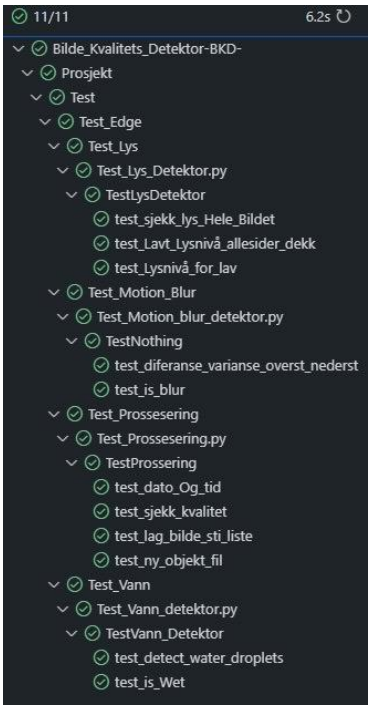
Ellers kan man også kjøre alle testene samtidig, men man kan ikke ha (.venv) aktivert når man gjør dette. Venv er python-miljøet som man kan operere i. Dersom man ikke har (.venv) foran filstien til hvor man er i terminalen er ikke dette noe man trenger å tenke på. Dersom filstien i terminalen starter med (.venv), skriv “deactivate”.

For å kjøre alle testene må du først trykke på testikonet i VSCode som vises i figur 10.1. Trykk på "Configure Python Tests" som dukker opp, velg “pytest”, og til slutt trykk på "Prosjekt". Trykk

deretter på kjørikonet ved siden av testen som dukker opp hvor mappestrukteren var tidligere. Figur 10.2 viser resultatet av kjøring av alle testene.



Figur 10.1 Testing ikon

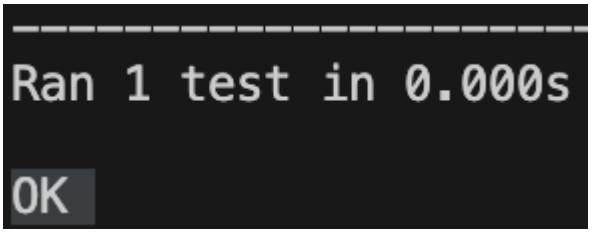


Figur 10.2 Alle tester

Kjøring av enhetstester på Mac:

I prosjektet kan man åpne og kjøre testfilene individuelt ved å gå inn i testklassene og trykke kjør. Ettersom man ikke kan gå ut av (.venv) på Mac slik man kan på Windows, blir ikke pytestene oppdaget av mac-en. Man kan derfor bare kjøre testfilene individuelt og ikke samlet slik en kan på Windows. Figur 10.3 viser utskrift av individuell kjøring av en test.

I tillegg vil testen “test_lag_bilde_sti_liste” feile på Mac ettersom det er noen komplikasjoner med hvordan de to operativsystemene kjører tester. Dersom man vil at den skal kjøre riktig på mac må sti en settes til 0, slik: “sti_en = liste [0]”, og sti to settes til 1, slik “sti_en = liste[1]”.



Figur 10.3 Individuell testkjøring på Mac

11 REFERANSER

[1] Microsoft, (2024). Visual Studio Code [Online]. Tilgjengelig:
https://code.visualstudio.com/

[2] Flask, (2010). Flask Documentation [Online]. Tilgjengelig:
https://flask.palletsprojects.com/en/3.0.x/

[3] Python, (2023, Des. 04). Python 3.11.7 [Online]. Tilgjengelig:
https://www.python.org/downloads/release/python-3117/

[4] W3Schools (2024). Python Pip [Online]. Tilgjengelig:
https://www.w3schools.com/python/python_pip.asp

[5] OpenCV, (2024). OpenCV [Online]. Tilgjengelig:
https://opencv.org/

[6] torch (2023). torch - PyTorch Documentation [Online]. Tilgjengelig:
https://pytorch.org/docs/stable/torch.html

[7] PyTorch, (2023). PyTorch Documentation [Online]. Tilgjengelig:
https://pytorch.org/docs/stable/index.html

[8] torchvision (2023). torchvision - Torchvision Documentation [Online]. Tilgjengelig:
https://pytorch.org/vision/stable/index.html

[9] matplotlib (2024). Matplotlib - Visualization with Python [Online]. Tilgjengelig:
https://matplotlib.org/

[10] Pytest (2024). pytest: helps you write better programs - pytest documentation [Online]. Tilgjengelig:
https://docs.pytest.org/en/8.1.x/

[11] Python (2024). PEP 257 - Docstring Conventions [Online]. Tilgjengelig:
https://peps.python.org/pep-0257/

[12] Visual Studio, (2024). autoDocstrings - Python Docstring Generator [Online]. Tilgjengelig:
https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring

[13] Medium, (2023, Mai 23). Mastering Unit Tests in Python with pytest: A Comprehensive Guide [Online]. Tilgjengelig:
https://medium.com/@adocquin/mastering-unit-tests-in-python-with-pytest-a-comprehensive-guide-896c8c894304