Tekoälypohjainen hahmontunnistus – YOLO-algoritmin käyttöönotto, kouluttaminen ja käyttöliittymän kehittäminen

Jussi Kosonen

Haaga-Helia ammattikorkeakoulu
Tradenomi
Amk-opinnäytetyö
2023

Tiivistelmä

Tekijä
Jussi Kosonen

Tutkinto
Tradenomi

Opinnäytetyön nimi
Tekoälypohjainen hahmontunnistus – YOLO-algoritmin käyttöönotto, kouluttaminen ja käyttöliittymän kehittäminen

Sivu- ja liitesivumäärä
44 + 20

Tämän toiminnallisen opinnäytetyön tavoitteena oli toteuttaa hahmontunnistusta esittelevä ohjelma Haaga-Helian AI-driver hankkeen käyttöön. Ohjelmassa tuli ajaa videokuvaa tietokoneen kamerasta tai muista lähteistä hahmontunnistuksen algoritmiin, ja esittää algoritmin antamat tulokset videon kanssa. Ohjelma rakennettiin käyttäen hahmontunnistukseen YOLOv8-algoritmia ja rakentamalla Python ohjelmointikielellä ja sille saatavilla olevilla kirjastoilla YOLO-algoritmin hahmontunnistusmallia käyttävä käyttöliittymä. Tässä raportissa kerrotaan tämän ohjelman rakentamiseen käytetyistä teknologioista, sekä käydään läpi ohjelman rakentamisen eri vaiheet. Lopuksi esitellään valmista ohjelmaa.

Raportin tietoperustassa keskitytään tässä projektissa hahmontunnistuksen ohjelmaa toteutettaessa käytettyihin teknologioihin. Ohjelmaa varten otettiin ensin käyttöön ja koulutukseen YOLOv8- algoritmi, jonka käyttöä myös kuvataan hieman raportissa. Projektissa käytössä olleiden laitteiden algoritmin koulutukselle asettamista haasteista johtuen, siirryttiin projektissa tämän jälkeen rakentamaan käyttöliittymää.

Käyttöliittymä rakennettiin käyttämällä C++ kielellä kirjoitettua QT –kirjastoa, jota käytettiin Python kielellä Pyside-sidoksella. Python on nykyisin yleisesti käytetty kieli koneälyn ja hahmontunnistuksen teknologioissa, ja YOLOv8-algoritmikin käyttää sitä. Tämän vuoksi käyttöliittymäkin rakennettiin Python kieltä käyttäen. Ohjelman rakentamisesta kerrotaan QT –kirjaston tarjoamien työkalujen käyttämisestä, sekä työkaluilla tehtyjen ohjelman osien yhdistämisestä ja niille toiminnallisuuden ohjelmoinnista. Käyttöliittymän rakentamisen jälkeen kerrotaan raportissa vielä YOLOv8-algoritmin kouluttamisesta.

Projektissa käytetty YOLOv8 algoritmi koulutettiin ensin testimielessä testiaineistolla, jota käytettiin ohjelman rakentamisessa ja lopulta, oman mallin kouluttamisen haasteista johtuen, ohjelman käyttämisessäkin hyödyksi. Algoritmille koulutettiin myös oma malli, jossa algoritmi opetettiin tunnistamaan sieniä, mutta mallin kouluttaminen sienien tunnistukseen osoittautui todella haastavaksi.

Raportin lopuksi esitellään valmis hahmontunnistuksen ohjelma, joka on jaettu Open Source –periaatteiden mukaisesti myös julkisesti GitHub –palvelussa. Lisäksi pohditaan vielä, mitä hyvää ja kehitettävää tuotoksessa ja projektissa olisi ollut. Osiossa mietitään myös ohjelman tekijän oppimista projektin aikana, sekä miten ohjelmaa voisi jatkokehittää.

Asiasanat
Hahmontunnistus, tekoäly, käyttöliittymä, YOLO, Python, Qt -ohjelmisto

Sisällys

1 Johdanto

Koneäly ja sitä soveltavat teknologiat ovat nousseet todella suosituksi uutisaiheiksi, ja siten kaikkien kahvipöytien keskusteluihin, uusien chattibottien avustuksella. Ehkä myös siksi etsin opinnäytetyöni aiheeksi jotain näihin teknologioihin sopivaa aihetta, jossa voisin yhdistää aloittamaani ohjelmoinnin opettelua koneälyyn. Opinnäytetyöpaja –kurssilla sain kuulla koulun etsivän tekijöitä eri projekteihin, joten tuon ilmoituksen myötä otin yhteyttä Ari Alamäkeen. Hän ehdotti aiheeksi hahmontunnistuksen demon tekemistä Haaga-Helian AI-driver projektille.

Viime vuosina konenäköä ja hahmontunnistusta on laajalti sovellettu teknologian eri aloilla. Esimerkiksi teollisuudessa voidaan tutkia tuotteiden laatua kuvantunnistuksen avulla, tai parkkipaikan portti voi tunnistaa auton rekisterikilven hahmontunnistuksen ja konenäön avulla. Hahmontunnistus ja kuvien tulkitseminen tietokoneen avulla onkin hyvin tärkeä teknologia todella monissa eri sovelluksissa.

Tämän toiminnallisen opinnäytetyön produktina on siis hahmontunnistusohjelma, demo, jolla voidaan esitellä hahmontunnistusta. Käyttäjälle tämä näkyy kameralla varustetulla tietokoneena, joka kamerallaan ottaa videota, joita tekoälymalli tutkii ja josta se tunnistaa hahmoja. Demossa on käyttöliittymä, jolle ajetaan videota, jonka algoritmi sitten tutkii ja kertoo käyttöliittymässä tuloksen. Jotta työstä ei tulisi liian laajaa, rajataan työtä siten, että siitä tehdään sellainen versio, joka ei ole kaikenkattava, mutta jota voidaan jatkokehittää. Ohjelmassa käytettävää hahmontunnistusmallia ei ole tarpeen saada kaikenkattavaksi, vaan malliksi riittää yleiseen tunnistukseen sopiva malli, jolla kuvista voidaan tunnistaa erilaisia hahmoja tunnistustarkkuuden hieman kärsiessä. Näin saadaan esiteltyä hahmontunnistusta yleisölle. Ohjelma tehdään kuitenkin siten, että siihen voidaan tuoda myöhemmin paremmin koulutettuja malleja.

Tässä työtä tehdessä selvitettiin, miten konenäön tekniikoita ja koneälyä voidaan valjastaa käyttöön, minkälaisia resursseja ja taitoja se vaatii sekä miten hahmontunnistusta voitaisiin hyödyntää erilaisiin tarpeisiin.

Työstä jää koulun käytettäväksi valmiiksi tietokoneelle asennettu demo, jolla hahmontunnistusta voidaan esitellä ja joka voitaneen tarvittaessa viedä eri paikkoihin esittelyä varten. Työstä jätetään koulun, sekä muiden halukkaiden käyttäjien käyttöön open source periaatteiden mukaisesti, käyttöön myös ohjelmakoodin repositoriot, joiden pohjalta projekti voidaan kopioida käyttöön muualla tai useammilla laitteilla, ja joiden pohjalta projektia voidaan jatkokehittää.

Työ voisi siis toivottavasti esitellä konenäköä ja koneälyä kaikille siitä kiinnostuneille opiskelijoille, sekä toimia alustana uusille innovaatioille ja jatkokehitykselle. Työstä siis toivottavasti hyötyvät niin opiskelijat kuin opettajat.

Käsitteitä

Konenäkö Digitaaliseen kameraan ja automaattiseen kuvankäsittelyyn perustuva järjestelmä, jolla tietokone voi havainnoida ympäristöään (Kielitoimiston sanakirja 2022).

Hahmontunnistus Konenäön osa-alue, jossa kuvasta tunnistetaan, etsitään ja luokitellaan hahmoja (Mathworks 2023).

YOLO You Only Look Once. Nimi algoritmille, joka on kehitetty hahmontunnistusta varten. (Zvornicanin 2022.)

Koneoppiminen Tekoälyn osa-alue, joka keskittyy datan ja algoritmien käyttöön tavoilla, jotka kopioivat ihmisten tapaa oppia (IBM 2023).

Syväoppiminen Yksinkertaisten prosessointiyksiköiden muodostamien kerrosten yhdistämistä neuroverkoiksi, siten että tieto kulkee niiden läpi vuoron perään. Eli prosessointiyksiköiden rakenne on koetettu tehdä ihmisen aivojen kaltaiseksi ja pienemmällä datamäärällä oppivammaksi. (Minnalearn 2023.)

Ohjattu oppiminen Konetta opetetaan syöttämällä sille dataa, johon on ennalta merkitty mitä opittavia asioita datassa on. Esimerkiksi kuvia opetettaessa merkitään kuviin, missä mikäkin hahmo kuvassa on. Data yleensä jaetaan kolmeen osaan, opettamista, opetuksen onnistumisen seuraamista ja testaamista varten. Ohjattua oppimista käytetään usein opettamaan luokittelua vaativia tehtäviä. (Choi, Coyner, Kalpathy-Cramer, Chiang& Campell 2020.)

Ohjaamaton oppiminen Ohjaamattomassa oppimisessa koneelle syötetään paljon dataa, jota ei ole ennalta merkitty millään tavalla. Kone etsii siis itse datasta kaavamaisuuksia tai erilaisuuksia, joiden perusteella se oppii. Ohjaamattoman oppimisen algoritmeja voidaan käyttää esimerkiksi datan ryhmittelyyn. (Choi, Coyner, Kalpathy-Cramer, Chiang& Campell 2020.)

Puoliohjattu oppiminen Puoliohjatussa oppimisessa konetta opetetaan osin merkatulla osin merkkaamattomalla datalla, yhdistäen näin molempien opetustyylien hyötyjä. (Choi, Coyner, Kalpathy-Cramer, Chiang& Campell 2020.)

Vahvistusoppiminen Vahvistusoppimisessa koneelle annetaan jokin tehtävä ja kerrotaan, mikä on haluttu lopputulos. Tämän jälkeen kone kehittää keinot halutun lopputuloksen saavuttamiseksi. (Choi, Coyner, Kalpathy-Cramer, Chiang& Campell 2020.)

Neuroverkko Ihmisen aivojen rakennetta mukailevaksi tehty rakenne. Rakenne koostuu kerroksiin jaetuista neuroneista ja niiden välillä olevista synapseista. (Alander & Honkela & Jacobsson 1996.)

Qt Qt on alustariippumaton ohjelmistojen kehityksen viitekehys, jolla voidaan kehittää työpöytä-, mobiilisovelluksien lisäksi sulautettujen järjestelmien sovelluksia. Qt on kirjoitettu C++ kielellä ja suunnattu käyttöliittymien rakentamiseen. (Qt Wiki 2022.)

Pytorch Pytorch on koneoppimisen kirjasto, jolla voidaan luoda ja kouluttaa neuroverkkoja. Pytorchin avulla saadaan ohjattua tietokoneen resursseja neuroverkon käyttöön. Pytorchia käytetään joko Python tai C++ kielillä. (Solegaonkar 2019.)

OpenCV Open Source Computer Vision. Intel –yhtiön kehittämä kirjasto, jota käytetään kuvien ja videoiden käsittelyyn sekä analysointiin. Kirjaston avulla kuvia voi esimerkiksi jakaa pikseleihin, jotta tietokone voi tulkita niitä. (Culjak, Abram, Pribanic, Dzapo & Cifrek 2012, 1725–1729.)

NVIDIA Teknologiayhtiö, joka on keskittynyt kehittämään ja valmistamaan näytönohjaimien siruja sekä ohjelmointirajapintoja tekoälyn ja muun nopean laskennan tarpeisiin. Johtava tekijä tekoälyn laitteistoissa. (Goel 2023.)

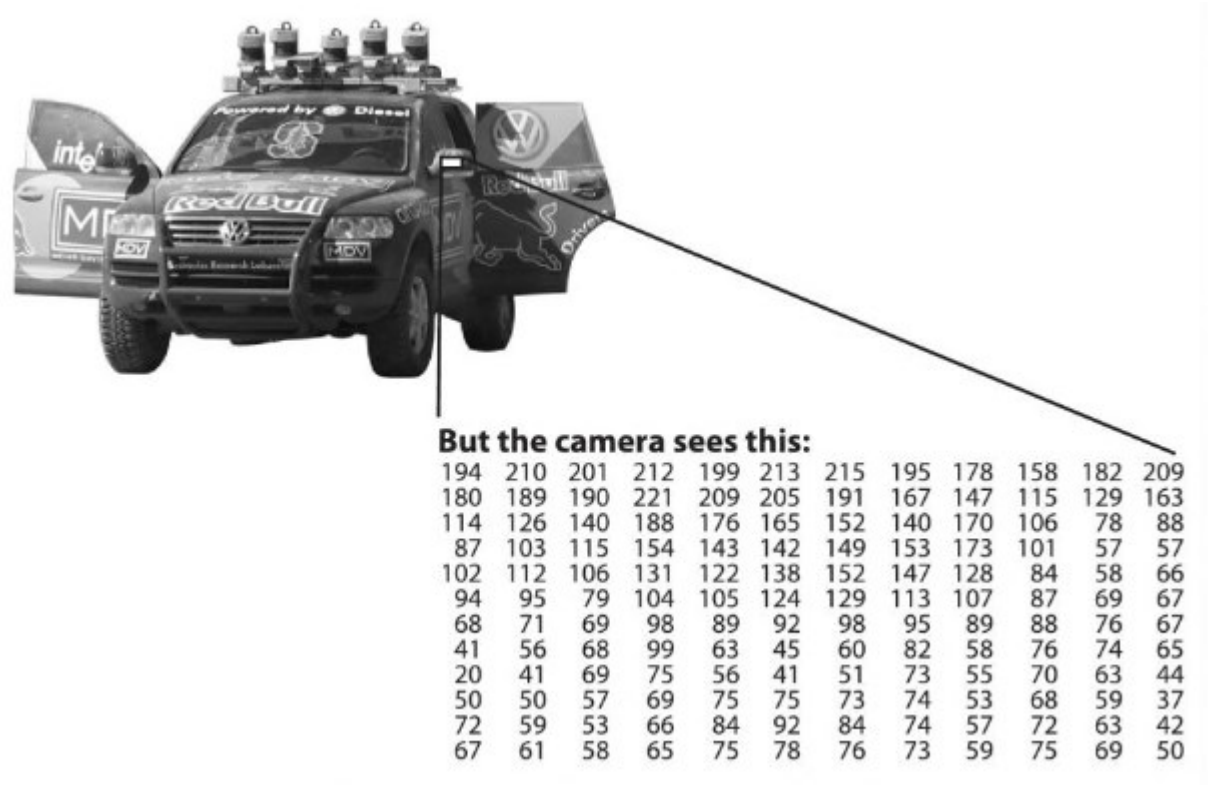
AMD Advanced Micro Devices. Teknologiayhtiö, joka kehittää siruja prosessoreihin sekä näytönohjaimiin, sekä niihin liittyviä teknologioita. (Burns 2022.)

GUI Graphical user interface, eli graafinen käyttöliittymä tietokoneen ja ohjelmien käyttämistä varten.

Widget Käyttöliittymän osanen, joka voi esitellä tietoa tai vastaanottaa syötteitä käyttäjältä. Widget voi myös sisältää muita widgettejä. Qt:ssa widgettiä, joka ei ole toisen widgetin sisällä kutsutaan ikkunaksi. (Qt Documentation 2023a.)

2 Mitä on hahmontunnistus

Hahmontunnistus on konenäön tekniikka, jolla tietokone tunnistaa ja löytää kuvasta asioita eli hahmoja. Hahmontunnistus on yksi tärkeistä syväoppimisen ja koneoppimisen tuottamista tuloksista. Ihmisen on helppo katsoa kuvaa tai videota ja tunnistaa siitä ihmiset sekä muut objektit, mutta tietokoneelle tehtävä on vaikeampi. Hahmontunnistuksen tavoitteena onkin opettaa tietokone katsomaan kuvaa ihmisen kaltaisesti ja ymmärtämään mitä kuva sisältää. (Mathworks 2023.)



Kuva 1. Tietokone näkee kuvasta auton peilin vain numeroina. (Bradski & Kaehler 2008, 3.)

Ihmisenä voi olla vaikea tajuta, miten vaikeaa hahmontunnistus tietokoneelle voi olla. Hahmontunnistus toimii ihmisen aivoissa niin automaattisesti, että voisi kuvitella tietokoneenkin selviävän siitä tehtävästä helposti. Mutta aivot käsittelevät käsittämättömän määrän dataa hetkessä ja tietokoneen tai neuroverkon on vaikea pystyä samaan. Sen vuoksi tietokoneelle pitää osata kertoa, miten kuvaa tulkitaan ja miten siitä etsitään hahmoja. Tieto pitää myös syöttää tietokoneelle sen ymmärtämässä muodossa. Tietokoneelle syötetäänkin kuvasta tiedot koordinaatteina, kuten kuvassa 1. Tietokone näkee siis vain numeroita, ja hahmontunnistuksen mallin tekijälle, eli tietokoneen opettajalle, jää vastuu siitä, miten tietokone nuo numerot tulkitsee. Tehtävää ei helpota sekään, että tietokone ei, kuten ei ihminenkään, osaa muodostaa kaksiulotteisesta kuvasta kolmiulotteista mallia, ellei sitä ole opetettu ennalta siihen. Eikä tietokoneella ole yleensä vuosien kokemusta hahmojen

tunnistamisesta, kuten ihmisellä on. Ja, mikäli kuvassa on minkäänlaisia opetetusta mallista poikkeavia ominaisuuksia, kuten epätarkkuutta, erilaista valaistusta, erilaista säätä, ei tietokone osaa ilman opetusta näitä tulkita. (Bradski & Kaehler 2008, 2–5.)

Kolmiulotteinen näkeminen on myös koneelle vaikeaa, eikä sitä varsinkaan yhdellä kameralla pysty helposti tekemään. Ihminen voi nähdä kaksiulotteisen kuvan ja muodostaa siitä kolmiulotteisen hahmon, mutta koneelle asia on vaikeampi. Koneelle voidaan kuitenkin rakentaa kyky hahmottaa esineitä kolmiulotteisesti, mutta tähän vaaditaan kaksi kameraa. Samalla tavalla kuin ihmisellä on kaksi silmää. Kameroilla voidaan yhtäaikaisesti kuvata hahmoa hieman eri kulmista ja laskea kuvissa olevista eroista kolmiulotteinen malli. Tämä vaatii kuitenkin melko paljon laskentatehoa ja kameroiden kuvaaman suunnan tietämistä sekä muuta dataa, joten tällaisen sovelluksen rakentamien huomattavasti hankalampaa sekä sen käyttäminen liikkuvien hahmojen tai liikkuvan kuvan tulkitsemiseen vielä haastavampaa. (Park & Baek 2013, 2-3.)

Nykyaikana syväoppimista ja neuroverkkoja voidaan kuitenkin hyödyntää yhä paremmin hahmontunnistuksessakin. Syväoppimisessa tavoitteena on kouluttaa tietokoneesta älykäs opettamalla sitä valtavalla määrällä dataa. Neuroverkot kehitettiin simuloimaan ihmisen aivojen toimintaa ja niiden avulla saadaan järjesteltyä ja käsiteltyä yhä isompia määriä dataa syväoppimisen mahdollistamiseksi. Neuroverkot suorittavat erilaisia algoritmeja, joiden avulla dataa käsitellään. Ensimmäisiä neuroverkkoja syväoppimiseen kehiteltiin 80-luvulla ja näistä tunnetuimmaksi tai tunnustetuimmaksi nousi CNN, konvoluutioverkko. Konvoluutioverkot todettiin hyviksi varsinkin kuvien tiedon käsittelyyn ja niitä käytetään nykyistenkin, kuten YOLOv8, hahmontunnistusalgoritmien käsittelyyn. (Graupe 2016, 1–3.)

Hahmontunnistuksen malleja voidaan kouluttaa erilaisilla tavoilla. Ohjatussa oppimisessa mallille syötetään tietoja, joissa selvästi kerrotaan mitä mikäkin on, eli se merkitään. Esimerkiksi kuvien tapauksessa kuviin merkataan missä mikäkin erilainen tunnistettava hahmo on. Kun dataa esimerkeistä on tarpeeksi, pystyy kone niiden perusteella löytämään samankaltaisia hahmoja muistakin kuvista. Ohjatussa oppimisessa data yleensä jaetaan kolmeen eri ryhmään, training, validation ja testing. Training ryhmän datalla malli koulutetaan, validation ryhmän datalla tutkitaan, miten hyvin malli on oppinut, ja testi ryhmän datalla testataan lopullinen malli. Monet kuvien tunnistuksen mallit opetetaan juuri ohjatulla oppimisella tai puoliohjatulla oppimisella, jossa algoritmille syötetään merkattua ja merkkaamatonta dataa, eli koneelle osaltaan kerrotaan mitä datasta pitäisi oppia ja osaltaan annetaan koneen itse löytää opittavat asiat. Koneälyä voidaan opettaa myös ohjaamattomasti, syöttämällä algoritmille merkkaamatonta dataa. Tällöin jää täysin algoritmin päätettäväksi, miten se dataa tulkitsee ja minkälaisia hahmoja tai malleja se datasta löytää. (Culjak, Abram, Pribanic, Dzapo & Cifrek 2012, 1–4.)

3 Teknologioita hahmontunnistuksen sovellukseen

Konenäön sovellusta varten tarvitaan erilaisia teknologioita. Kuva tai video pitää ensin saada muotoon, jota tietokone ymmärtää. Tuon jälkeen kuvaa tai videota voidaan analysoida jonkin algoritmin avulla ja tulokset esittää ihmiselle sopivassa muodossa. Konenäköä esittelevään projektiin tarvitaan myös jokin käyttöliittymäkerros, jonka avulla käyttäjä voi syöttää tutkittavaa kuvaa järjestelmälle ja jossa käyttäjä näkee analyysin tulokset.

Jokaiseen vaiheeseen löytyy erilaisia sovelluksia ja teknologioita. Seuraavaksi esitellään tähän projektiin valitut teknologiat.

3.1 OpenCV

Jotta kuvia ja videoita voidaan ohjelmallisesti tutkia, tulee niitä käsitellä muotoon, jonka tietokone ymmärtää. OpenCV, pidemmältä nimeltään Open Source Computer Vision Library, kehitettiin yhtenäistämään ja helpottamaan kuvien ja videoiden käsittelyä konenäön tarkoituksiin. Ennen OpenCV:tä ei kuvien ja videoiden tulkitsemiseen ollut tehtynä muiden kirjastojen kanssa yhtä hyvin yhteensopivia kirjastoja eikä koodi ollut yhtä vakaata ja nopeaa. OpenCV on alun perin Intelin kehittämä kirjasto kuvien ja videoiden analysointiin. Sittemmin sitä ovat kehittäneet useat eri kehittäjät ja suurin kehitysharppaus tehtiin vuonna 2009. Nykyään OpenCV kirjasto sisältää yli 2500 optimoitua algoritmia ja sitä käytetään ympäri maailmaa. Kirjasto on jaettu eri moduuleihin, joista jokainen yrittää tarjota ratkaisuja tietyn konenäön ongelmien osalle. Kirjasto tarjoaa työkaluja esimerkiksi kuvan jakamiseen pikseleihin ja tuon tiedon tallettamiseen sopivanlaiseen datastruktuuriin, jotta kuvaa voidaan analysoida ohjelmallisesti. Kirjaston avulla tietokone saadaan siis näkemään kuva. (Culjak, Abram, Pribanic, Dzapo & Cifrek 2012, 1725–1729.)

OpenCV suunniteltiin tehokkaasti laskentatehoa käyttäväksi ja oikean elämän sovellukset mielessä. OpenCV kirjoitettiin optimoidulla C –kielellä ja se pystyy hyödyntämään moniydinprosessorien useita ytimiä. OpenCV:stä tehtiin siis open source –kilpailija muille kaupallisille ratkaisuille. Yksi OpenCV:n tavoitteista oli myös tarjota helppokäyttöinen konenäön infrastruktuuri, jonka pohjalle on nopea ja helppo kehittää pitkälle kehittyneitä konenäön sovelluksia. OpenCV sisältää toimintoja eri konenäön aloille, kuten tehtaiden laaduntarkastukseen, turvallisuus sovelluksiin, lääketieteelliseen kuvantamiseen ja robotiikkaan. OpenCV:n uusimman version lisenssi on Apache 2, jonka turvin, joten sitä on helppo käyttää ilmaiseksi erilaisissa sovelluksissa (Petrovicheva 2020). Lisenssin turvin OpenCV:tä voidaan käyttää vapaasti, ilman maksuja tai rajoituksia, minkälaisiin sovelluksiin vain niin harrastuksissa, koulutuksissa kuin kaupallisissakin sovelluksissa. OpenCV:tä käyttävätkin tai ovat käyttäneet monet yritykset, esimerkiksi IBM, Microsoft, Intel, Siemens ja Google. (Bradski & Kaehler 2008, 1–2)

3.2 YOLOv8 algoritmi

YOLO on lyhenne sanoista You Only Look Once ja on nimi syväoppimisalgoritmille, joka on suunniteltu reaaliaikaiseen hahmojen tai objektien tunnistamiseen kuvista ja videosta. YOLO algoritmi käyttää kuvan prosessointiin neuroverkkoa, jossa kuvaa tutkitaan useassa kerroksessa. YOLO on tehokas ja nopea algoritmi, koska se suorittaa objektien tunnistamisen kuvasta yhdellä läpikäynnillä. Yhden läpikäynnin mallit sopivat yleisesti ottaen paremmin nopeuden ja tehokkuuden takia ”oikean elämän” sovelluksiin, kuten autonomisten ajoneuvojen, turvakameroiden ja robotiikan ongelmiin. Kun taasen useamman läpikäynnin mallit voivat olla tarkempia. (Kundu 2023.)

YOLO ei ole ainut olemassa oleva yhden läpikäynnin malli, toinen suosittu malli on nimeltään MobileNet SSD, jossa SSD on lyhenne sanoista ”Singe Shot Detector”. MobileNet SSD on hieman tarkempi ja sopii joihinkin sovelluksiin paremmin. YOLO on näistä hieman nopeampi ja kevyempi, joten se sopii sovelluksiin, joissa tarkkuudesta voidaan hieman tinkiä, mutta tarvitaan laitteistovaatimuksiltaan kevyempää ja nopeampaa tunnistusta. (Technostacks 2023.)

YOLO sopinee siis hyvin juuri tähän kyseiseen opinnäytetyöhön, jossa ollaan tekemässä hahmontunnistusta yleisesti esittelevää demoa ja jota voidaan soveltaa monissa erilaisissa sovelluksissa.

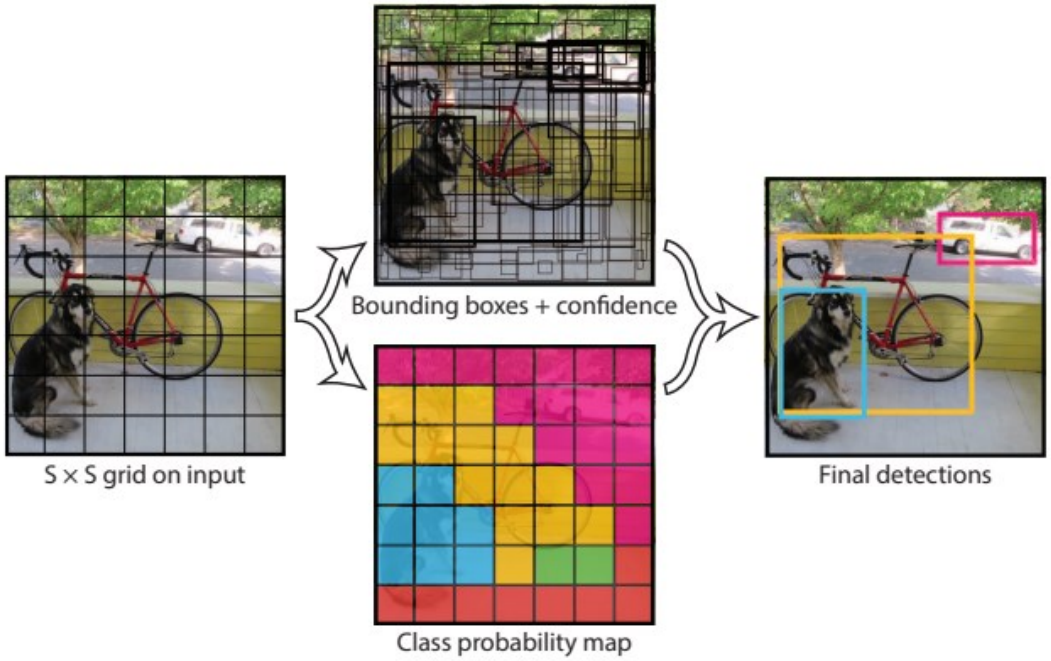
YOLO algoritmin kehittivät Joseph Redmond ja Ali Farhadi Washingtonin yliopistossa. Algoritmin kehitys on ollut nopeaa. Ensimmäinen versio algoritmista julkaistiin vuonna 2015. Seuraava ja paranneltu versio, YOLOv2, julkaistiin vuonna 2016, v3 vuonna 2018 ja v4 vuonna 2021. Jokaisessa versiossa lisättiin ja paranneltiin ominaisuuksia. Viidennessä versiossa, YOLOv5 lisättiin muiden ominaisuuksien lisäksi käyttöä helpottavia ominaisuuksia, kuten mahdollisuus käyttää algoritmia Python –kielellä, aiemman C tai C++ lisäksi. (Ultralytics 2023a.)

Python lienee nykyään suositumpi ja käytetympi kieli, ainakin erilaisten koneälyä ja dataa käsittelevien sovellusten parissa. Python lienee myös helpompi opetella ja se on suoraviivaisempi kieli pieniin ja suurempiinkin projekteihin.

YOLO algoritmissa koko kuva ajetaan yhden kerran neuroverkon (CNN) läpi. Kuva jaetaan ruutuihin, joista jokaisesta arvioidaan sen kuuluminen ennalta tunnettuun luokkaan (class probability) ja määritetään kuuluvatko sitä ympyröivät ruudut samaan objektiin vai ei. Kuvassa 2 on havainnollistettu tätä toimintaa.

Kuten kuvassa 2 on esitetty, jaetaan kuva pieniin ruutuihin, esimerkiksi kooltaan 16x16 pikseliä oleviin nelikulmioihin ja käsitellään jokaista ruutua samanaikaisesti yhdellä ajokerralla käyttäen useita CNN-kerroksia. Jos tällainen ruutu on lopulta keskellä hahmoa, tunnistetaan tämän ruudun avulla koko hahmo. YOLO-verkko tuottaa jokaiselle ruudulle ennusteen, joka sisältää tunnistettujen

objektien luokat ja niiden keskipisteen koordinaatit (x, y), objektien leveys- ja korkeusmitat (w, h) sekä objektin todennäköisyyden kuulua tiettyyn luokkaan. Jokaisen ruudukon ruudun viereiset ruudut tutkitaan ja niille annetaan pisteitä sen mukaan, miten varma malli on siitä, että ne kuuluvat tutkittavassa ruudussa olevaan hahmoon ja miten tarkka malli uskoo tuon arvionsa olevan. Näin saadaan siis määritettyä kuvasta löytyvien, ja mallin jo ennestään tuntevien, hahmojen ympärille laatikot. Tämän prosessin avulla algoritmi löytää kaikki kuvassa esiintyvät hahmot ja antaa niistä luokka- ja sijaintitiedot. (Redmon, Divvala, Girshick & Farhadi 2016, 780.)



Kuva 2. YOLO algoritmin malli. (Redmon, Divvala, Girshick & Farhadi 2016, 780.)

YOLOv8 on algoritmin uusin versio, jonka on kehittänyt ja paketoinut Ultralytics nimellä kulkeva yritys. YOLOv8 julkaistiin tammikuussa 2023. YOLOv8 on oikeastaan framework, joka yhdistää aiemmat versiot algoritmista ja jonka avulla voidaan käyttää myös aiempia versiota YOLO algoritmista. Uusi versio, toi uusiakin ominaisuuksia, mutta mikäli käytössä on jo jokin vanhempi malli, voidaan sitä hyödyntää myös uuden YOLOv8 version kanssa. Suurimpia uudistuksia uudessa versiossa on sen nopeus uutta mallia opetettaessa, jonka YOLOv8 tekee huomattavasti nopeammin kuin aikaisemmat versiot. (Bhalerao 2023.)

Kaiken kaikkiaan YOLOv8 käyttöönotto on tehty helpoksi ja se toimii helppokäyttöisenä alustana konenäön sovelluksille. YOLOv8 mallin lataaminen omaan käyttöön vaatii vain muutaman komen-

non kirjoittamisen (Ultralytics 2023b) YOLOv8 on helposti saatavissa valmiiksi koulutettuja hahmontunnistuksen malleja, joita voi ottaa suoraan käyttöön tai joita voi kouluttaa tarkemmiksi tai sopimaan erilaisiin sovelluksiin. Toki mallin voi myös kouluttaa puhtaalta pöydältä ja aivan itse, haluamallaan aineistolla. Vaikka konenäön tekniikoista ei paljoa tietäisikään, on YOLOv8 algoritmin käyttöönotto sovelluskehittäjälle helppoa.

YOLOv8 on ladattavissa valmiiksi tehtynä pakettina. Paketti sisältää kaiken tarvittavan YOLOv8 mallin käyttöönottoa varten ja käytön voi aloittaa, vaikka komentorivikäyttöliittymästä. YOLO –algoritmilla on myös laaja käyttäjäkunta, joka voi tarvittaessa auttaa hahmontunnistuksen ja algoritmin ongelmien kanssa. Paketti sisältää myös Python kirjaston, joten algoritmia on nopea alkaa käyttämään Python koodia käyttävissä projekteissa. (Francesco, Solawetz, 2023.)

3.3 Pyside

Ultralyticsin YOLOv8 algoritmin käyttäminen Python –ohjelmointikielellä on tehty helpoksi, ja algoritmi käyttää Pythonia jo muutenkin, joten on luontevaa rakentaa algoritmia ajava käyttöliittymä myös Pythonin avulla. Pythoniin on tehty monia erilaisia kirjastoja ja viitekehyksiä myös käyttöliittymän tekoa varten (Python Wiki 2023), esimerkiksi Tkinter tai Kivy, joista Tkinter kuuluu Pythonin standardikirjastoon. Moni usealla alustalla toimivista viitekehyksistä näyttää kuitenkin nojaavan suomalaislähtöisen ja kovassa nosteessa olleen (Heikkilä 2020) Qt -yhtiön GUI, eli käyttöliittymän rakentamisen, työkaluihin. Qt viitekehyksen avulla on rakennettu ohjelmia, kuten Adoben Phoshop Elements, Telegram, TeamViewer, sekä sen avulla on rakennettu käyttöliittymiä esimerkiksi LG:n ja HP:n laitteisiin (Solovev 2022). Qt toimii natiivin näköisenä Windowsin lisäksi Linuxilla ja MacOSilla, joten sitä käyttämällä ohjelmasta saadaan helpommin useammalle käyttöjärjestelmälle yhteensopiva.

Qt kirjastoa voi Pythonilla käyttää muutaman eri sidoksen avulla, mutta Pyside, tai Qt for Python, on näistä Qt –yhtiön virallinen sidos (Qt Group 2023a). Qt kirjasto on kirjoitettu C++ kielellä, ja Pysiden avulla voidaan siis käyttää Qt kirjaston työkaluja Python kieltä käyttämällä. Koska Pyside on Qt-yhtiön tekemä virallinen sidos, valikoitui se käyttöön tässä projektissa.

Qt sisältää ison kasan erilaisia käyttöliittymän elementtejä. Kirjastossa on nappuloita, liukusäätimiä, erilaisia ikkunoita ja keinoja asetella näitä haluamallaan tavalla ja skaalautuvasti ruudulle. Kehittäjän ei siis tarvitse kuin valita sopiva palanen, laittaa se sopivaan paikkaan ja muokata tarvittaessa haluamakseen. Suurin osa elementeistä toimii suoraan eri käyttöjärjestelmillä ja näyttävät samalta kuin käyttöjärjestelmän normaalit samanlaiset elementit, mutta tarvittaessa voidaan eri käyt-

töjärjestelmille käyttää myös jotain omia elementtejään. Qt tarjoaa myös paljon erilaisia lisäosia peruspakettinsa päälle. Lisäosilla voidaan esimerkiksi ohjata jotain spesifimpiä asioita, kuten sensoreita tai sarjaportin kautta ohjattavia laitteita. Qt voidaan myös tuoda kolmansien osapuolten tekemiä elementtejä, mikäli kirjastosta ei löydy sopivaa. (Qt Group 2023b.)

Qt tarjoaa myös käyttöliittymän suunnittelua ja toteuttamista helpottavia työkaluja, kuten Qt Designer –ohjelman. Qt designerin avulla voidaan käyttöliittymä suunnitella graafisessa ympäristössä asettelemalla kirjaston valmiita työkaluja sekä muita widgettejä, tai käyttäjän itse tekemiä widgettejä, haluamilleen paikoilleen. Käyttöliittymästä voidaan luoda ohjelmassa lopullisen näköinen versio painikkeineen ja teksteineen. Tämän jälkeen nuo mallit voidaan kääntää Python koodiksi ja ohjelmoijalle jää koodattavaksi ohjelman logiikka. Eli mitä tapahtuu mistäkin napista ja miten ohjelma vaihtaa näkyville eri ikkunoita ja widgettejä. (Qt Documentation 2023b.)

Pyside sopinee siis tällaiseen projektiin vallan hyvin helppokäyttöisyytensä ja monipuolisten ominaisuuksiensa puolesta. Pyside ja Qt voidaan myös käyttää ilmaisella lisenssillä opiskelu ja opetuskäyttöön GPL lisenssillä (Qt Group 2023c).

3.4 Pytorch

Pytorch on Facebookin kehittämä koneoppimiseen suunnattu Python kirjasto, jossa on työkaluja syväoppimisen mallien rakentamiseen. Pytorchin avulla voidaan rakentaa, kehittää ja ajaa erilaisia koneoppimisen algoritmeja. Pytorchin avulla saadaan valjastettua tietokoneen resurssit käyttämään algoritmia. Pytorch tukee esimerkiksi resurssien jakamista samanaikaisesti eri laitteille, vaikka useammalle näytönohjaimelle ja suorittimelle tai jopa useammalle koneelle. Pytorchin käyttämä datastruktuuri on Tensor, joka on lähellä Pythonin normaaleja matriisia tai taulukkoa. Tensoreita voidaan ajaa näytönohjaimella tai muulla Tensoreiden ajamiseen tehdyllä laitteella. (Nvidia 2023.)

Muita mahdollisia samaan asiaan kehitettyjä kirjastoja ovat TensorFlow ja Keras. Pytorch on kirjoitettu kokonaan Pythonilla, joten Pythoniin tottuneelle henkilölle kirjaston käyttö voi olla helpompaa. Pytorchia pidetään myös helpompana, joustavampana ja muistia paremmin hyödyntävänä muihin verrattuna. (Terra 2023.)

3.5 Google Colab

Koneoppimisen ja hahmontunnistuksen teknologiat, varsinkin mallien kouluttamisen osalta, vaativat melko paljon tehoa tietokoneelta. Muutaman tuhannen kuvan datasetin kouluttaminen hahmontunnistuksen mallille voi tehottomalla koneella kestää päiviä. Mikäli käytössä ei ole tarpeeksi tehokasta konetta, on onneksi tarjolla verkossa toimivia apuja. Näistä yksi opettelu ja koulutuskäyttöön ilmainen työkalu on Google Colab.

Google Colab on suunniteltu juuri koneoppimisen ja vastaavien kehittämiseen. Palvelu tarjoaa käyttäjänsä käyttöön laskentatehoa, suorittimien ja näytönohjainten muodossa. Palvelussa voit kirjoittaa selaimeen Python koodia, joka sitten ajetaan virtuaalikoneella Googlen tarjoamilla servereillä. Palvelu on ilmainen, mutta mikäli palvelussa on ruuhkaa voi resursseja joutua odottamaan. Tämä ei kuitenkaan ole ongelma, sillä palveluun voi kirjoittaa haluamansa koodit muistikirjaan muistiin, jonka voi ajaa myöhemmin. Palveluun voi myös tuoda itsensä tai muiden tekemiä muistikirjoja, joten koodin jakamien ja asioiden testailu on helppoa. (Google 2023)

Google Colab palvelussa on Python asennettuna valmiiksi, kuten on Pythonin mukana normaalisti tuleva pakettienhallintaohjelma PIP. Tämä tarkoittaa sitä, että palvelussa voit myös asentaa erilaisia kolmansien osapuolten Python kirjastoja ja elementtejä ja käyttää niitä projekteissa. Palveluun voi ladata tiedostoja omalta koneelta, tai liittää oman Google Driven projektiin, jolloin se toimii normaalin kovalevyn tavoin projektissa. Kaiken koodin voi tallentaa muistikirjaan, jonka voi sitten myöhemmin tarvittaessa kaivaa esille tai jakaa samaa asiaa kokeilevalle kaverille. (Bonner 2019.)

Mikäli koodarin oma kone ei siis ole tarpeeksi tehokas, tai siinä ei ole koneoppimista hallitsevan NVIDIAN näytönohjainta, voi koneoppimisen sovelluksia tehdä ja kokeilla Googlen Colab palvelussa samaan tapaan kuin omalla koneella. On siis mahdollista esimerkiksi asentaa YOLOv8 algoritmi, ladata datasetti Google Drivestä ja kouluttaa YOLO –malli huomattavasti nopeammin käyttäen Googlen vapaata laskentakapasiteettia.

4 Hahmontunnistuksen demon rakentaminen

4.1 Lähtötilanteen kuvaus ja suunnittelun esittely

Tämän työn tilasi Haaga-Helian AI-Driver hanke. Tarpeena oli saada tehtyä demo tai esitys konenäöstä ja hahmontunnistuksesta. Demo tulisi olla sellainen, jonka voi asentaa, vaikka koulun käytävällä tai luokassa olevaan koneeseen, jossa sitä voi seurata ja kokeilla. Demossa käytettäisiin webbikameraa tai muuta lähdettä videokuvan välittämiseen tietokoneelle, joka tutkii kuvan ja esittää siitä löytämänsä hahmot tietokoneen ruudulla. Demo piti myös tehdä tarpeeksi helppokäyttöiseksi, jotta sitä voi käyttää lähes kuka vain, ilman aikaisempaa tietoa hahmontunnistuksesta tai syvempää ymmärrystä tietokoneista tai koodauksesta.

Käytännössä tehtäväksi tuli siis kouluttaa ja ottaa käyttöön jokin hahmontunnistuksen algoritmi ja tehdä käyttöliittymäkerros, jossa tuolle algoritmille voidaan syöttää tutkittavaa dataa ja esittää algoritmin antamia tuloksia. Lisäksi ohjelma pitäisi saada paketoitua sellaiseksi, että se on helppo asentaa ja siirtää uusille tietokoneille. Algoritmiksi ehdotettiin YOLO-algoritmia. Algoritmista oli juuri vuoden 2023 alussa julkaistu uusi versio (Jain 2023), YOLOv8, joka valikoitui algoritmiksi.

Toimeksiantaja antoi tuotoksen ulkonäölle melkoisen vapaat raamit. Aikataulu ei myöskään ollut tarkka, mutta valmista piti tulla kevään 2023 aikana. Projektin eniten aikaa vievät osuudet olivat hahmontunnistuksen mallin kouluttaminen ja käyttöliittymän rakentaminen. Mallin kouluttamisessa aikaa vei varsinkin kuvien merkkaaminen ja testinä tehdyn kouluttamisen ajaminen. Esimerkiksi pelkkään yhden datasetin kuvien ajamiseen algoritmiin, meni käytetyltä tietokoneelta 6 täyttä vuorokautta. Käyttöliittymän rakennusta varten tuli kehittäjän opetella käyttämään käyttöliittymän rakentamisen työkaluja ja koodin kirjoittamista, joiden opetteluun ja käyttöliittymän rakentamiseen meni aikaa noin yhden koulussa suoritetun ohjelmoinnin kurssin verran. Työn tuottamiseen menetelmät olivat täysin kehittäjän valittavissa, joten kehittäjä pääsi valitsemaan tekniikoita sen perusteella mistä ajatteli olevan hyötyä omassa tulevaisuudessaan ja työelämässä.

4.2 Tuotoksen tuottaminen

Ohjelman teko muodostui kolmivaiheiseksi. Ensin tutustuttiin YOLOv8 algoritmiin, asennettiin se, testailtiin sen toimintaa sekä sen käyttämistä Pythonilla ynnä koulutettiin algoritmiin saatavilla olevasta testiaineistosta hahmontunnistuksen malli. Toisena vaiheena oli tarkoitus kouluttaa YOLOv8 algoritmin malli omalla aineistolla, mutta kehittäjän laitteiston asettamista haasteista johtuen, pelkässä YOLOv8 –paketin mukana toimitetun testiaineiston ajamisessa algoritmiin kesti noin viikon. Oli siis aikataulullisista syistä ja projektin tyhjäkäynnin estämiseksi, hypättävä kolmanteen vaiheeseen, eli käyttöliittymän tekoon, jo ennen oman mallin kouluttamista, samalla miettien ratkaisua,

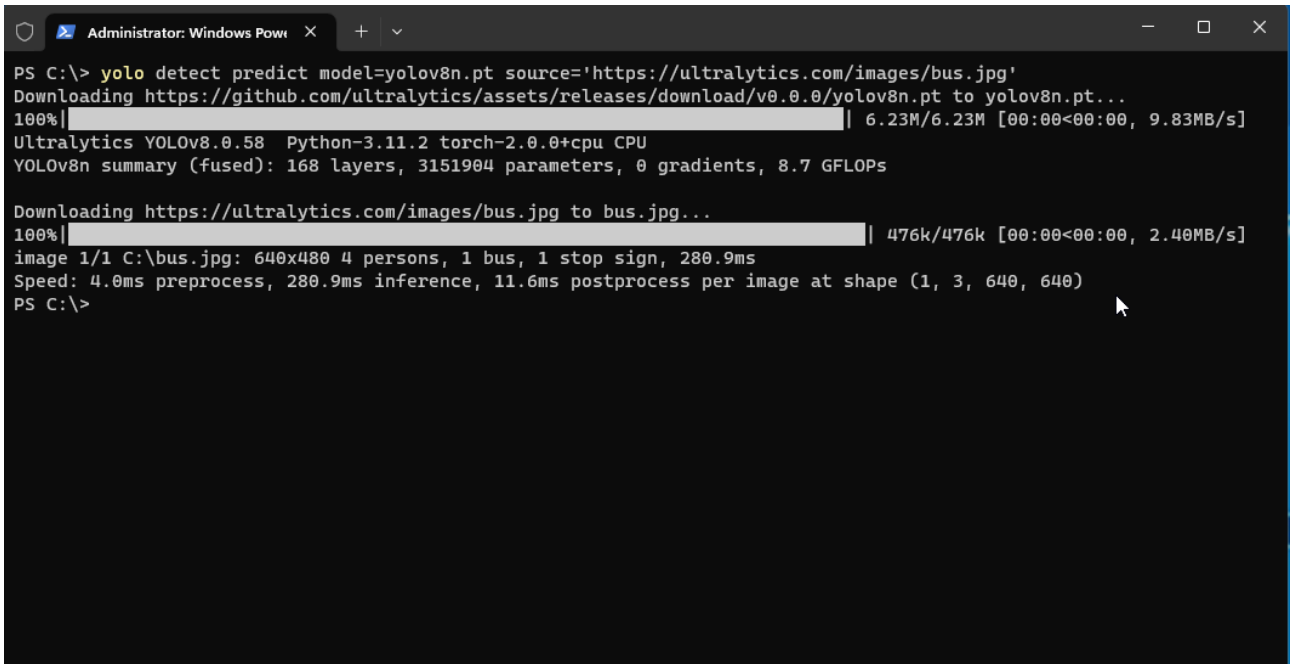
jolla oman hahmontunnistuksen mallin kouluttaminen saataisiin tehtyä paremmilla resursseilla. käyttöliittymää pystyi testaamaan ja käyttämään tuolla testinä koulutetulla mallilla, joten käyttöliittymän rakentaminen voitiin aloittaa heti. Käyttöliittymän teon jälkeen oli vielä edessä oman mallin kouluttaminen, eli aineiston merkkaaminen ja algoritmin kouluttaminen tuolla aineistolla. Kouluttamisen nopeuttamista varten etsittiin keinoja, ja lopulta löydettiin Google Colab palvelu, jossa malli saatiin koulutettua paremmilla resursseilla ja siten nopeammin.

4.3 YOLOv8 käyttöönotto

Projektissa käytettäväksi algoritmiksi valikoitui YOLOv8. Algoritmia ehdotettiin tilaajan puolelta ja se sopii muutenkin hyvin tähän projektiin. YOLO-malli on melko nopea ja kevyt laitteistovaatimuksiltaan, joten sen ajaminen onnistuu hyvin monella erilaisella tietokoneella. YOLO on myös helppo ottaa käyttöön ja sitä on yksinkertainen käyttää, joten vaikka kehittäjä ei tietäisi tekniikasta paljoa, on YOLO algoritmin käyttöönotto helppoa. (Rath 2023.)

YOLOv8 oli projektin aloitushetkellä sen verran uusi, että dokumentaatio oli vielä hieman vajavainen. Onneksi kuitenkin algoritmin käyttö oli hyvin samanlaista kuin aikaisemmalla YOLOv5 mallilla, johon löytyi dokumentaatiota paremmin. Dokumentaatio parani myös v8 osalta projektin edetessä ja esimerkiksi algoritmin käyttöön Python koodin avulla on sittemmin tullut lisää ohjeita myös virralliseen dokumentaatioon.

YOLOv8 paketin käyttö vaatii Python ympäristöä, joten se tulee olla asennettuna koneelle, jossa kehittämistä aletaan tekemään. Ultralyticsin tekemän YOLOv8 paketin saa asennettua Pythonin mukana tulevalla PIP-pakettihallinta ohjelmalla, joka asentaa samalla kaikki paketin vaatimat kirjastot, kuten Pytorch ja Opencv. Asentamisen jälkeen YOLOv8 algoritmia voi käyttää heti suoraan tietokoneen komentoriviltä tai tuomalla se Python koodiin. Komentorivillä käyttämällä algoritmia pääsee testaamaan heti ja ohjelma lataa tarvittaessa esikoulutettuja malleja, mikäli koulutettua mallia ei koneelta valmiiksi löydy. (Ultralytics 2023b.)



Kuva 3. YOLOv8 algoritmin käyttö komentoriviltä.

Kuvassa 3 on esimerkki, jossa algoritmia käytetään komentoriviltä. Kuvassa olevalla komennolla algoritmille syötetään linkistä löytyvä kuva, jonka ohjelma lataa koneelle. Ohjelma myös lataa esikoulutetun hahmontunnistuksen mallin, koska sitä ei koneelta löytynyt. Lataamisen jälkeen algoritmi käy läpi ja kertoo mitä hahmoja kuvasta löysi. Esimerkissä malli löysi kuvasta 4 henkilöä, yhden bussin ja yhden stop-merkin.

4.4 YOLO-mallin kouluttaminen

YOLOv8 ja sen mukana tuleva PyTorch pystyi projektin aloittamisen aikaan Windows käyttöjärjestelmässä suorittamaan prosessejaan vain Nvidia näytönohjaimilla tai koneen prosessorilla. Prosessorin voimin kouluttaminen on todella hidasta, joten näytönohjaimen lisääminen kokoonpanoon helpottaisi odotteluun käytettyä aikaa huomattavasti. Toki datasetin voi laittaa ajoon, vaikka yöksi, mutta muutaman tuhatta kuvaa sisältäneen datasetin ajaminen algoritmiin kesti noin 6 vuorokautta. Linuxiin saisi AMD:n kehittämän ROCM-tekniikan, jolla laskentaan voisi hyödyntää AMD:n joitain näytönohjaimia. Näytönohjaimella voi nopeuttaa mallin opettamista, mutta projektin tekijällä oli käytössään vain vanhempi NVIDIA:n Quadro näytönohjain, joka ei enää suoriutunut laskuista, sekä uudempi AMD:n 6000-sarjan kuluttajakäyttöön suunnattu näytönohjain. Tuota AMD:n näytönohjainta koetettiin saada toimimaan Linux –käyttöjärjestelmässä. Virallinen tuki ei kuitenkaan ulottunut Navi23 teknologian näytönohjaimiin, joka projektin tekijän 6000-sarjan näytönohjain on, eikä näytönohjainta saatu toimimaan PyTorchin kanssa, vaikka sitä yritettiin erilaisilla ajureilla ja apuoh-

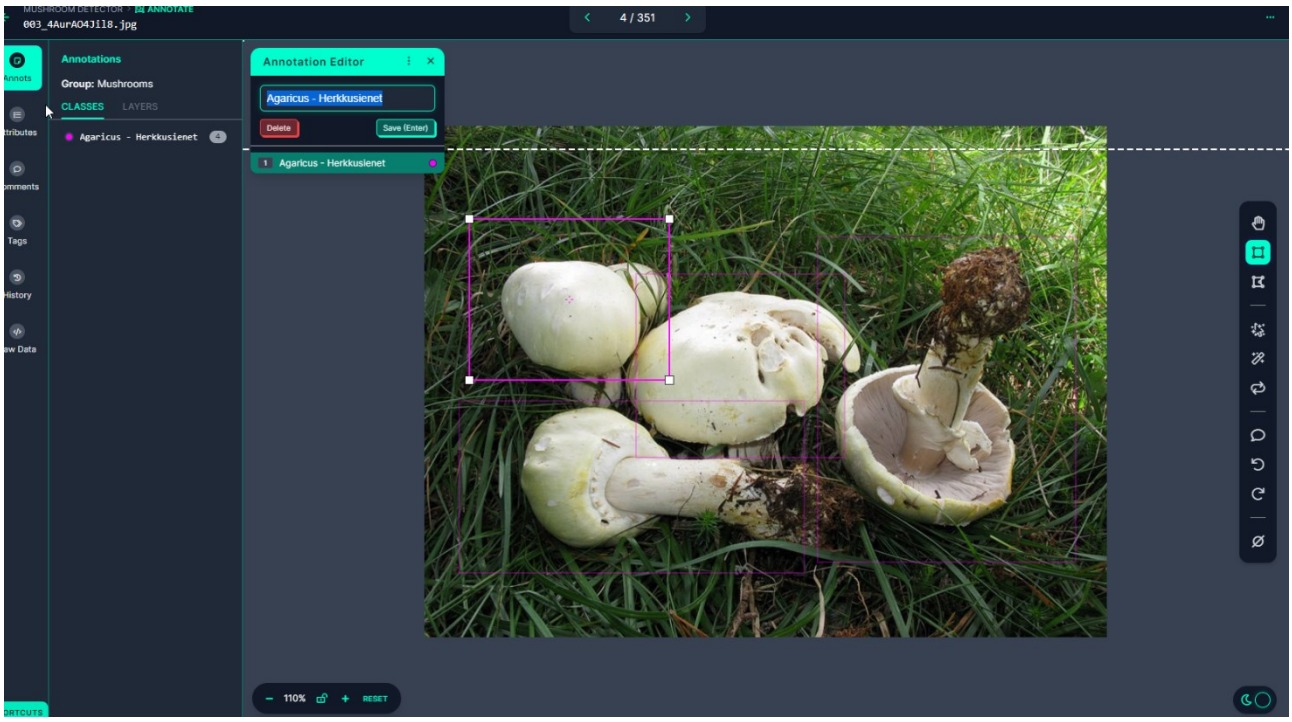
jelmilla. Nykyään tilanne voisi olla toinen, sillä huhtikuussa 2023 AMD ilmoitti tulevasta ROCM tuesta Windows-ympäristössä sekä AMD:n kuluttajakäyttöön suunnatuilla näytönohjaimilla (Liu 2023). Tätä teknologiaa ei kuitenkaan ehditty tämän projektin puitteissa testata ja kouluttaminen tehtiin Google Colab palvelussa.

Google Colab palvelu valikoitui tähän projektiin lähinnä sen vuoksi, että kehittäjän käytössä oli jo valmiiksi paljon Googlen ekosysteemin tuotteita. Google Colab palveluun ei tarvitse Google tunnusten lisäksi tehdä muita tunnuksia ja parhaimpana ominaisuutena Google Drive –tallennustilan saa liitettyä Google Colab kehitysympäristöön kiintolevyksi. Tällöin ei tarvinnut Drivessä jo tallennettuna ollutta suurta datasettiä siirtää erikseen taas uuteen paikkaan, vaan se oli kehitysympäristössä käytössä heti. Muitakin palveluja olisi tarjolla muiltakin isoilta toimijoilta, kuten Amazon sekä Microsoft, mutta Googlen erilaisia palveluita erilaisissa ohjelmistokehitystehtävissä käyttäneelle Googlen Colab oli helppo tutustuttava.

Ideana oli siis kouluttaa YOLO-algoritmi tunnistamaan erilaisia sienilajeja. Kouluttamista varten tarvitaan mahdollisimman paljon kuvia eri sienistä ja niitä löytyi pienen etsimisen jälkeen Kaggle-palvelusta. Kaggle on palvelu, joka on suunnattu juuri tähän tarkoitukseen ja johon käyttäjät voivat lisätä datasettejä erilaisten koneoppimisprojektien avuksi (Mahmoud 2022). Palvelusta löydettiin valikoima sienten kuvia, jossa eri sienet oli jaettu kansioihin sukunsa mukaan. Tätä projektia varten näistä kuvista muodostettiin noin 2700 kuvaa sisältävä datasetti, noin kolmesataa kuvaa per sieniryhmä. YOLO-mallin kouluttamista varten nämä kuvat pitää kuitenkin vielä käsitellä.

YOLO-mallia koulutetaan kuvilla ja niihin liitetyillä tekstitiedostoilla, joissa on kerrottuna koordinaatit kuvassa oleville hahmoille ja hahmoluokkien nimet. Tätä annotointia varten on olemassa joitain tekoälyavusteisia työkaluja, mutta käytännössä annotointi on ihmisen tehtävä, sillä mikäli konetta ei ole opetettu tunnistamaan kuvissa olevia hahmoja, ei se niitä luonnollisesti osaa tunnistaa. Annotointia varten on olemassa erilaisia työkaluja, mutta tässä projektissa annotointityökaluksi valikoitui Roboflow –palvelu, joka on myös YOLO-algoritmin tekijöiden ehdottama palvelu (Ultralytics 2023e). Palvelussa kuvat pitää ladata Roboflowta pyörittävän yrityksen palvelimille, joten palvelu ei välttämättä kaikille tämän vuoksi sovi. Koulutus ja opettelutarkoituksiin palvelu on kuitenkin sopiva ja ilmainen.

Kuvien palveluun lataamisen jälkeen päästään kuvia käsittelemään. Jokaiseen kuvaan piirretään laatikko kuvasta löydetyn hahmon ympärille ja kirjataan hahmon luokka, kuten kuvassa 4.

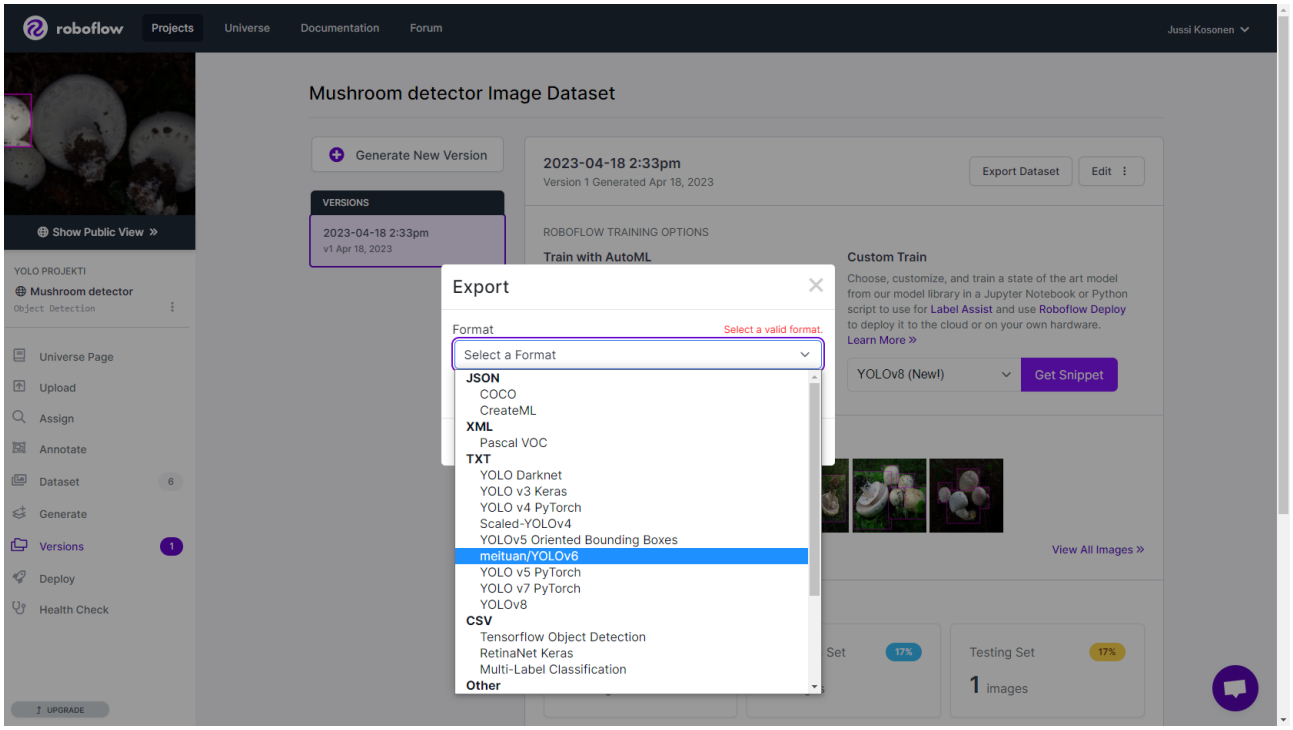


Kuva 4. Kuvien annotointia Roboflow:ssa

Mikäli kuvassa on useampia tunnistettavaksi haluttavia hahmoja samasta tai eri luokista, piirretään kaikkien ympärille laatikot ja merkataan niiden luokkien nimet (Ultralytics 2023e). Tämä lienee projektin työläin ja tylsin vaihe, sillä laatikoiden piirtäminen käy melko yksitoikkoiseksi muutaman sadan kuvan jälkeen. Mutta datasetin on hyvä olla mahdollisimman laaja, jotta algoritmi saadaan opetettua tunnistamaan varmemmin oikein. Suosituissa ja valmiiksi tehdyissä dataseteissä voi olla satoja tuhansia kuvia, ja hyvään datasettiin kuuluu vähintään pari sataa kuvaa tunnistettavaa luokka kohti (Lee 2021). Tähän datasettiin kuului yhteensä 9 eri hahmoluokkaa, joista jokainen koulutettiin noin 220 kuvalla, validoitiin noin 60 kuvalla ja testattiin noin 30 kuvalla per hahmoluokka. Yhteensä datasettiin tuli siten noin 2800 kuvaa.

Kun kaikki kuvat on käyty läpi, voidaan datasetti muuntaa YOLO –algoritmin vaatimaan muotoon. YOLO vaatii, että kuvat ovat omassa kansiossaan ja tekstitiedostot koordinaatteineen toisessa, kuvalla ja tekstitiedostolla tulee myös olla sama nimi. Kuvat tulee myös jakaa kolmeen kategoriaan, kuviin, jolla mallia opetetaan, kuviin, jolla malli validoidaan ja kuviin, joilla malli testataan. Tämän lisäksi tarvitaan yaml tiedosto, jossa kerrotaan missä mitkäkin kuvat ovat. Roboflown suuri etu on

se, että nämä toimenpiteet voi Roboflow tehdä automaattisesti. Kun kaikki kuvat on annotoitu, valitaan vain missä muodossa datasetti muodostetaan ja datasetti on valmis käytettäväksi, niin kuin kuvassa 5 ollaan tekemässä. (Ultralytics 2023e.)



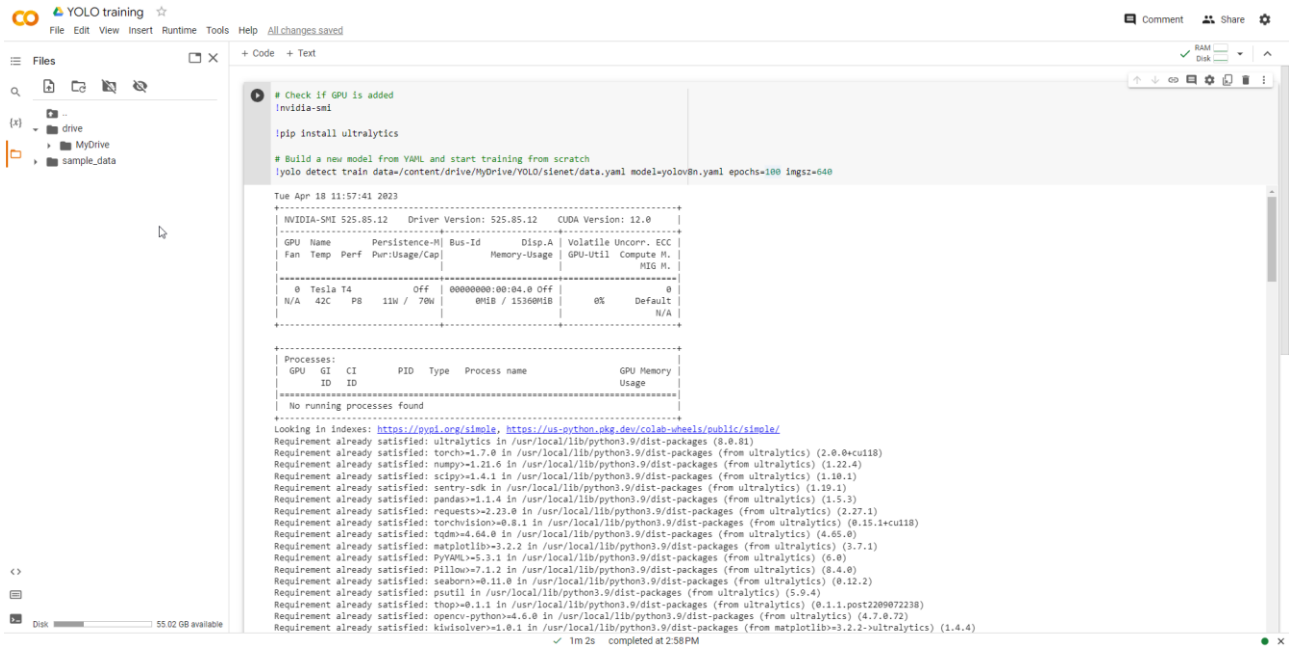
Kuva 5. Datasetin muuntaminen oikeanlaiseksi

Roboflow tarjoaa myös API:n, jonka avulla datasettiä voitaisiin käyttää Google Colab palvelussa ilman, että sitä tarvitsee ladata omalle koneelle, mutta tätä ei tässä projektissa testattu, koska datasetti oli tarkoitus käyttää omalla laitteistolla YOLO-mallia treenaten. Datasetti kuitenkin ladattiin tämän jälkeen Google Drive -palveluun, jotta se oli käytettävissä projektin tekijällä usealla laitteella. Google Drivesta datasetti saatiin myös ladattua suoraan Google Colab -palveluun.

Datasetin valmistuttua, oli vuorossa YOLO-mallin kouluttaminen. Tämä kouluttaminen tehtiin Google Colab -palvelussa, jossa kouluttaminen saatiin tehtyä paremmilla resursseilla ja siten nopeammin kuin projektin tekijän omalla laitteistolla. Google Colab palvelussa käytössä on Python ympäristö, ja käyttöliittymään voidaan kirjoittaa suoraan niin Python koodia kuin komentorivikomentoja. Palveluun voi myös tuoda omia tiedostoja joko käyttäjän koneelta tai suodaan Google Drivestä, liittämällä Google Drive kehitysympäristössä olevaksi kiintolevyksi.

Käyttöliittymään voidaan kirjoittaa koodia ja komentoja, ja sen jälkeen ajaa se ympäristössä. Kouluttamista varten tuli ensimmäisenä ladata ympäristöön YOLO-paketti, samaan tapaan kuin se olisi

ladattu omalle koneelle. Tämän jälkeen tehtiin vielä tarkastus, että resursseina oli varmasti käytettävissä myös näytönohjain, jotta kouluttaminen menee nopeammin. Lopuksi voitiin käynnistää YOLO-mallin kouluttaminen halutulla datasetillä.



Kuva 6. Koulutuksen etenemistä Google Colab -palvelussa.

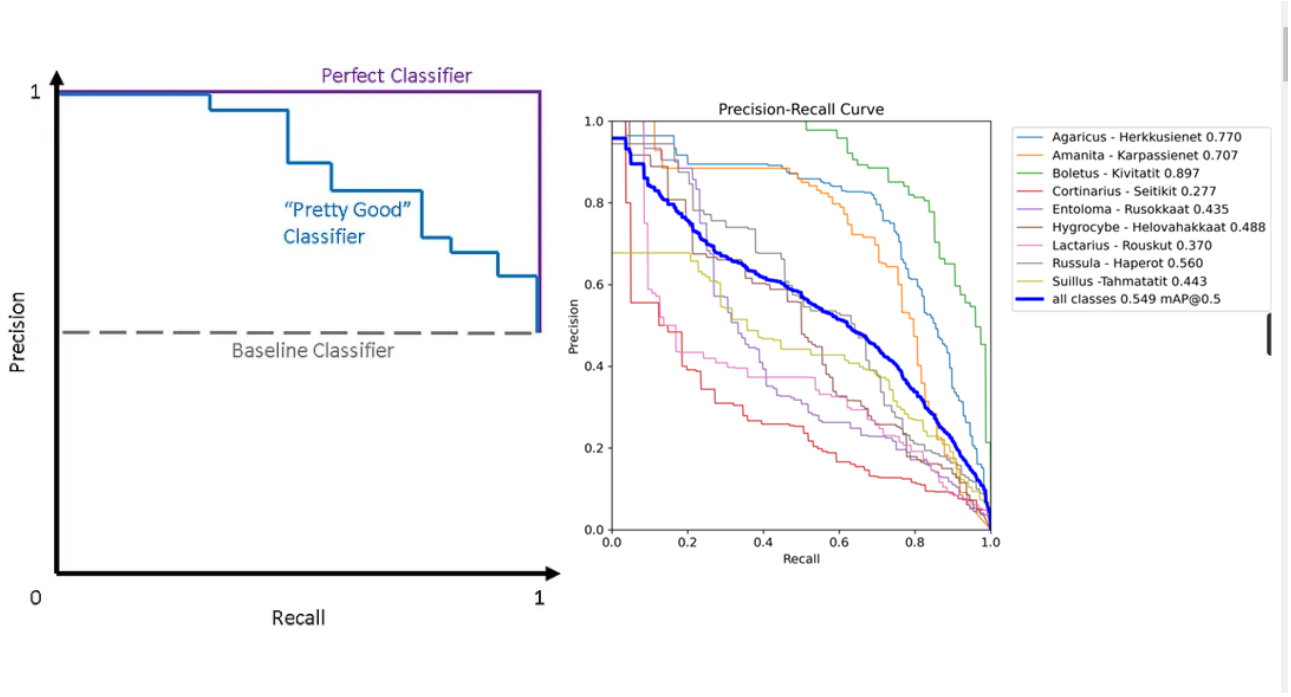
Ajon aikana on näkymä samantapainen kuin se olisi tietokoneen komentoriviltä ajettaessa, kuten kuvasta 6 on nähtävissä. Omassa ikkunassaan on näkyvillä samat asiat mitä komentorivi-ikkunassa näkyisi.

Koulutuksen valmistuttua tiedostoja koulutuksen onnistumisesta voi tarkastella palvelussa tai ne voi ladata, koulutun mallin kanssa, omalle koneelle ja mallin liittää ohjelmaan. Koulutuksen suoriutumisesta on saatavilla monenlaista eri tietoa ja graafeja. Näistä mallin tiedoista ja mallin parantamisesta näiden tietojen avulla olisi varmasti tarpeeksi tutkimista ja yhden opinnäytetyön tarpeiksi, mutta käydään tuloksi karkeasti läpi seuraavissa kappaleissa.

YOLO-mallia koulutettaessa tutkii algoritmi myös, miten hyvin mallin koulutus on onnistunut ja tallentaa testien tulokset, sekä piirtää niistä kuvaajat. Näistä kuvaajista voidaan nopeasti nähdä, miten hyvin mallin koulutus on onnistunut. Tarkastellaan ensin PR-käyrää.

PR-käyrä näyttää minkälainen on mallin tarkkuus, eli Precision, suhteessa mallin kykyyn tunnistaa, eli Recall. Täydellinen malli tunnistaisi kaikki hahmot tarkasti, mutta todellisuudessa hyvät mallit eivät tähän pysty, mutta pysyttelevät kuitenkin kuvan 7 vasemmanpuoleisen kuvaajan mukaisesti jossain yli 70% tarkkuudessa (Steen 2020). Sienien koulutettu malli ei pääse lähellekään tällaisiin

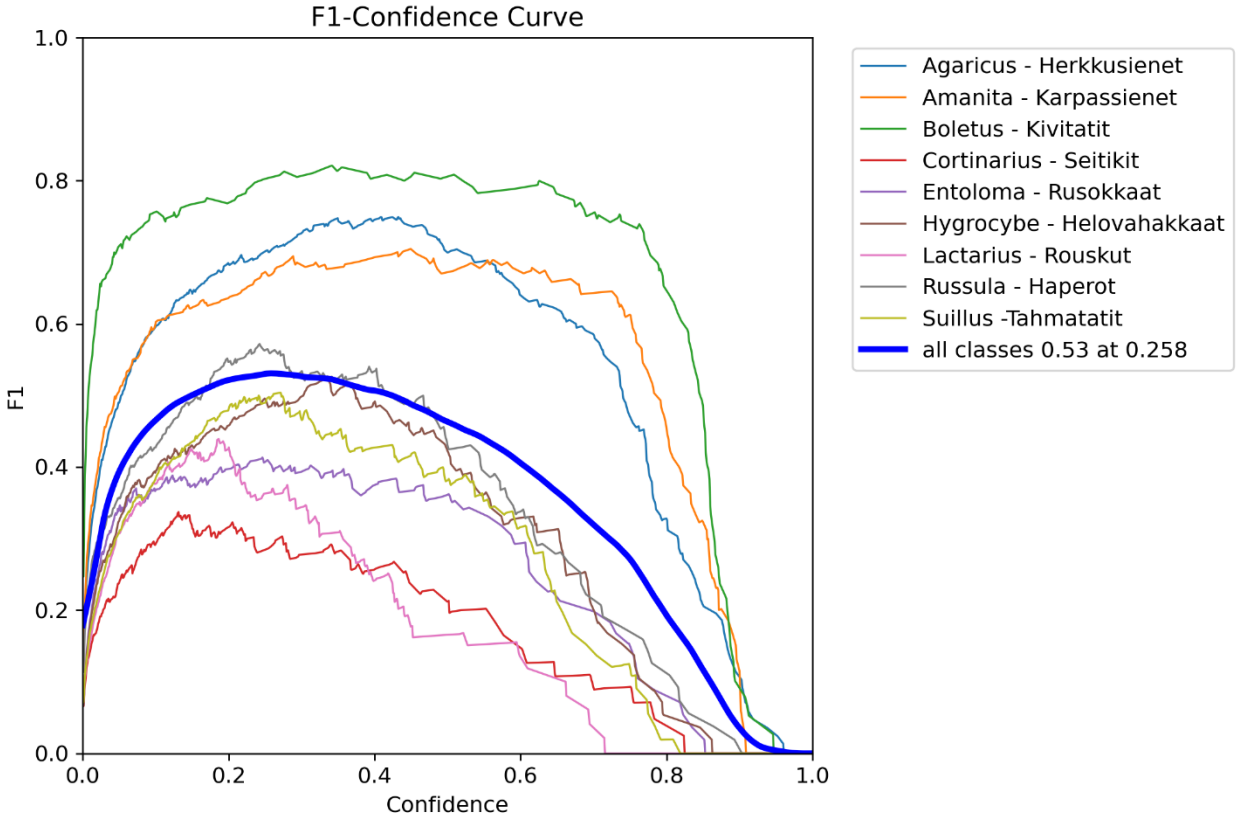
lukemiin, vaan jää keskimäärin 50% tarkkuuteen, kuten kuvan 7 oikeanpuoleisesta kuvaajasta on nähtävissä. Jotkin hahmot tunnistetaan paremmin, mutta keskimäärin tulos on aika heikko.



Kuva 7. Vasemmalla malli siitä, minkälainen PR-käyrä olisi hyvä olla (Steen 2020).

Yksi toisista kuvaajista, joka YOLO-mallin tuloksista saadaan, on F1-käyrä. F1-käyrä piirtää kuvaa siitä, minkälaisissa arvoissa tarkkuuden ja tunnistuskyvyn pitäisi olla, jotta mallista saadaan parhaimmat tulokset, mikäli tavoitteena on tunnistaa hahmoja mahdollisimman tarkasti ja varmasti. Käyrässä piirretään F1-tulokset y-akselille ja tunnistuksen tarkkuuden tulos, josta käytetään termiä confidence score, x-akselille. Tästä piirtyvästä käyrästä saadaan arvioitua millä tunnistuksen tarkkuuden arvolla saadaan mallista parhaimmat tulokset ulos. (Czakon 2023.)

Kouluttamassamme mallissa tulokset ovat melko heikkoja, kuten kuvasta 8 on nähtävissä. Parhaimmat tulokset saataisiin silloin kun confidence score on vain 0.258, maksimin ollessa 1.0. YOLO-mallia käytettäessä oletuksena confidence score on asetettu arvoon 0.5 (Shivaprasad 2019). Jos siis asetettaisiin confidence score arvoon 0.5 olisi F1-tulos vain 0.5 tai hieman alle, joka ei ole aivan parasta laatua. Hyvä F1-tulos olisi 0.8 tai sitä isompi (Allwright 2022).



Kuva 8. Koulutetun YOLO-mallin F1-käyrä

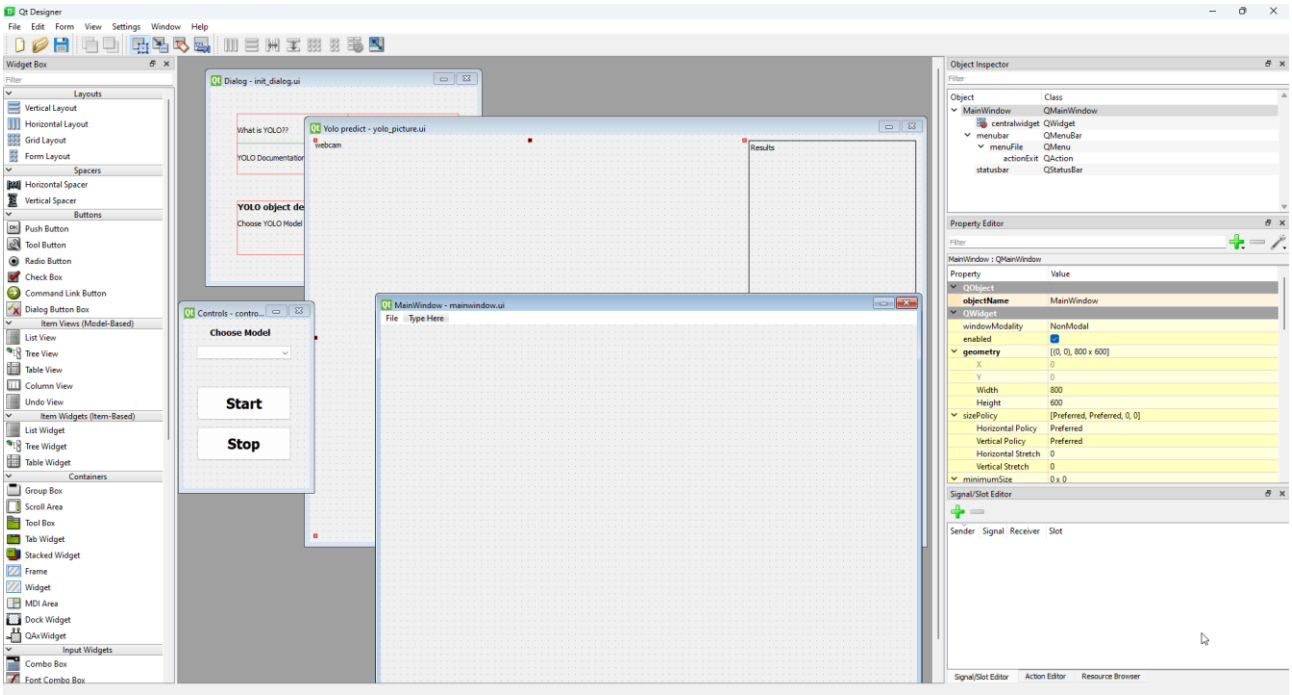
Kuvaajista voidaan kuitenkin huomata, että joillain yksittäisillä sienilajikkeilla tunnistus onnistuu paremmin ja joillain heikommin. Hajonta on myös melko suuri. Osaltaan nämä voivat johtua siitä, että vaikka kuvia oli jokaiselle hahmoluokalle sama määrä, oli kuvissa olleiden hahmojen lukumäärä kuvissa eri. Eli joillain hahmoluokilla oli enemmän opetukseen käytettyjä kuvan osia, kuin toisilla, eikä näiden määrää tullut tarkastettua ennen koulutuksen tekemistä. Koulutuksen tuloksista, jotka ovat kaikki liitteessä 7, saadaan selville, että esimerkiksi hahmoluokalle Agaricus – Herkkusienet oli hahmoja datasetissä lähes 600 kappaletta, kun luokalla Boletus – Kivitatit alle 300. Mutta, jos tarkastellaan tarkkuutta ja hahmojen löytämistä, ei hahmojen määrä kuitenkaan täysin korreloi löytämisen onnistumisen tai tarkkuuden kanssa.

4.5 Käyttöliittymän rakentaminen

Käyttöliittymän rakentaminen tehtiin käyttämällä Qt –työkaluja ja kirjastoa. Koska Qt –kirjasto on kirjoitettu käyttäen C++ kieltä ja tämä ohjelma kirjoitettiin Pythonilla, käytettiin rakentamiseen Pyside –sidosta, jolla Qt –kirjaston elementtejä voidaan käyttää Python –kieltä kirjoittamalla. Qt tarjoaa apuvälineitä käyttöliittymän suunnitteluun, sekä jopa koodin muodostamiseen suoraan suunnitelmista. Näitä Qt: työkalujen avulla muodostettuja koodinpätkiä on myös helppo muokata jälkikäteen, ja Qt-designerillä luotuja luokkia hyväksikäyttämällä voidaan käyttöliittymään lisätä helposti uusia osia jälkikäteen, koodarin omaan koodiin koskematta ja koko koodia uudelleenmuokkaamatta.

4.5.1 Qt-Designer

Kun YOLOV8 algoritmin hahmontunnistus toimimaan, oli aika aloittaa käyttöliittymän rakentaminen. Ensimmäisenä piti perehtyä Pyside6 sidoksen avulla käytettävään QT-kirjastoon ja sen työkaluihin. Opiskelun jälkeen alkoi käyttöliittymän suunnittelu ja rakentaminen QT-Designer työkalulla.



Kuva 9. GUI suunnittelua QT-Designerissä

QT-Designer on graafinen työkalu, jolla QT kirjaston valmiit, ja tarvittaessa kolmannen osapuolen tekemät, ikkunat ja muut käyttöliittymän palaset voidaan asetella haluamille paikoilleen. QT-Designerillä suunnitellut ja rakennetut ikkunat saadaan muutettua Python koodiksi. Designerissä eri käyt-

töliittymän palaset voidaan asetella oikeille paikoilleen, kuten kuvassa 9, ja valmiisiin tuotoksiin tarvitsee koodarin koodata vai toiminnallisuus. Suunnittelun jälkeen. Kun ikkunat on suunniteltu Designerilla, tallentuvat ne designerin omalle .ui tiedostomuodolle. Nämä tiedostot voidaan taasen muuttaa python koodiksi, tai sille kielelle, jolla ohjelmaa koodataan. Nämä tiedostot muodostavat designerillä tehdyistä ikkunoista luokkia, joista koodiin voidaan luoda olioita ja käyttää niitä toiminnallisuuden koodaamiseen. Kaikki Designerillä luotujen ikkunoiden koodi on löydettävissä liitteestä 5. (Fitzpatrick 2022a)

QT-Designerillä luotu luokka sisältää kyseisen ikkunan sommittelut, eli missä mikäkin nappi ja teksti on ja esimerkiksi, miten ne skaalautuvat ikkunan kokoa muutettaessa. QT-Designerillä luotujen luokkien käyttäminen koodissa on myös kätevää koodin ja käyttöliittymän ulkoasun muokkaamisen kannalta. Kun Designerin avulla luotu koodi jätetään koodarin toimesta koskematta, ja niistä tehdään olioita ohjelmakoodiin, voidaan näitä Designerilla tehtyjä luokkia helposti muokata jälkikäteen Designerissa, tuoda muokattu koodi ohjelmaan (Fitzpatrick 2022a). Näin saadaan tuo uusittu luokka heti käyttöön, eikä koodarin itse kirjoittamaan koodiin tarvitse välttämättä koskea ollenkaan.

Kun käyttöliittymän eri osaset on luotu QT-Designerissä, voidaan niitä alkaa käyttämään koodissa ja yhdistelemään osia käyttöliittymän rakentamiseksi.

```
class Controls(QWidget, Ui_controls):

    def __init__(self, parent = None):
        super(Controls, self).__init__()
        self.setupUi(self)

        self.stopButton.setEnabled(False)


class Picture(QWidget, Ui_pictureFrame):

    def __init__(self, parent = None):
        super(Picture, self).__init__()
        self.setupUi(self)

class Setup(QDialog, Ui_Dialog):

    def __init__(self, parent= None):
        super(Setup, self).__init__()
        self.setupUi(self)
        self.label_youtube.setEnabled(False)
        self.youtubeURL.setEnabled(False)

        #Fetch available models
        models = find_files("pt")
        for x in models:
            self.modelListBox.addItem(x)
```

Kuva 10. QT-Designerillä luotuja luokkia muokataan ja otetaan käyttöön

Eri osasista luodaan omat olionsa kuvan 10 tapaan, joita voidaan sittemmin käyttää toiminnallisuuden rakentamiseen. Toiminnallisuuden rakentamista varten näitä luotuja olioita voidaan tämän jälkeen käyttää käyttämällä niissä olevia metodeja, jotka muodostuivat QT-designerissa tehdyistä pohjista, tai muodostamalla omia metodeja.

QT-käyttöliittymän tekeminen aloitetaan tekemällä applikaatio, kuten kuvan 11 koodin alussa, tarvittaessa asettamalla sille jokin ikoni. Tämän jälkeen applikaatioon luodaan pääikkuna, jossa pääosat käyttöliittymästä suoritetaan, sekä asettamalla tämä ikkuna näkyväksi. Tämän jälkeen applikaatio suoritetaan. (Fitzpatrick 2022b)

```
223
224   app = QApplication(sys.argv)
225   app.setWindowIcon(QIcon('logo.ico'))
226
227   window = MainWindow()
228   window.show()
229   app.exec()
```

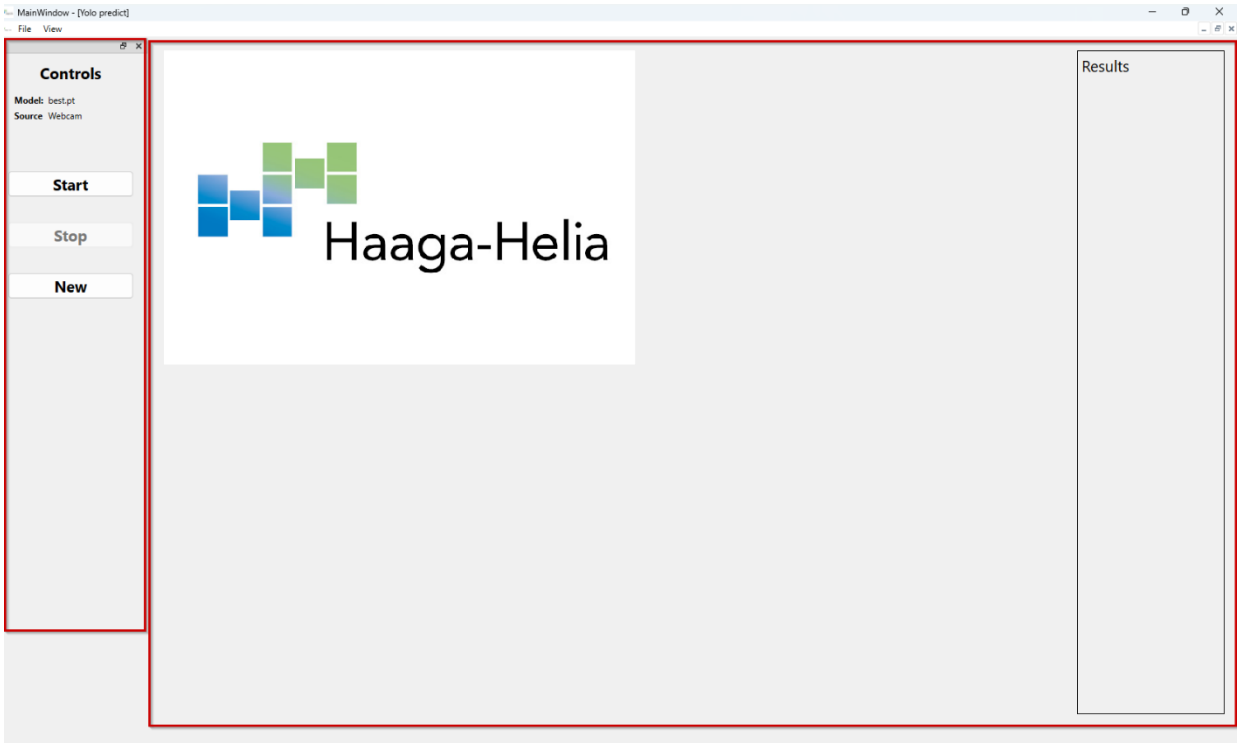
Kuva 11. QT-ohjelman luonti

Kuvassa 12 oleva MainWindow() objekti on siis pääikkuna, jonka sisälle voidaan tuoda lisää ikkunoita luomalla ennalta QT-Designerillä tehdyistä luokista olioita. Näin voidaan esimerkiksi tuoda pääikkunaan ikkuna, jossa ovat nappulat käyttöliittymän käyttämistä varten ja näyttää se. Tämän jälkeen voidaan pääikkunaan luoda ikkuna, jossa YOLO-algoritmin läpi ajettu kuva ja tulokset tekstinä näytetään.

```
class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setupUi(self)
        self.showMaximized()

        #Adding controls widget
        self.controls = Controls(self)
        self.dockControls.setWidget(self.controls)
        self.controls.startButton.setEnabled(False)

        #Adding picture area
        self.picture = Picture(self)
        self.mdiArea.addSubWindow(self.picture)
        self.picture.showMaximized()
```

Kuva 12. Muodostetaan pääikkuna ja sen sisällä olevat ikkunat.

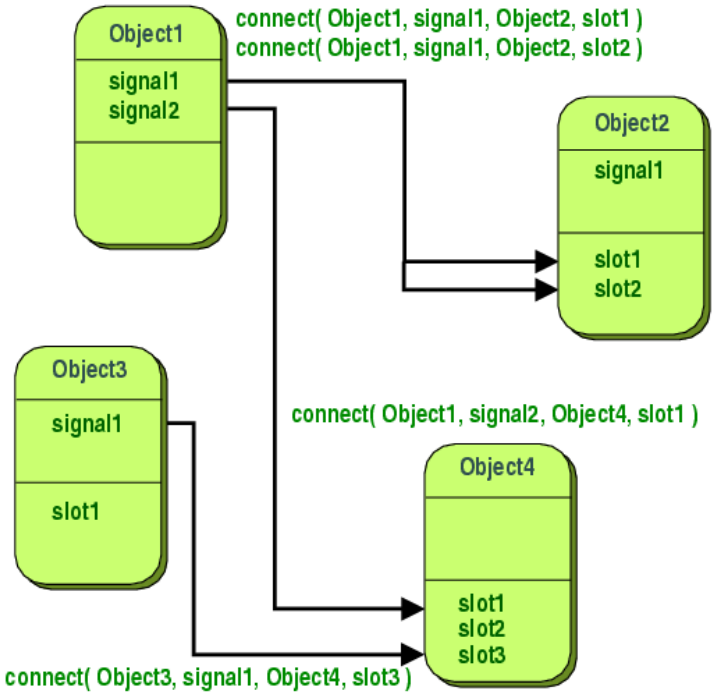
Näin saadaan pienellä määrällä itse kirjoitettua koodia luotua kuvan 13 mukainen pääikkuna, jossa sisällä nappulat käyttöliittymän käyttämistä varten sisältävä ikkuna, sekä ikkuna kuvan ja tulosten näyttämistä varten.



Kuva 13. Pääikkuna, jonka sisällä olevat ikkunat ympyröity kuvaan punaisella.

Pääikkunan ja muiden ikkunoiden luomisen jälkeen, voidaan niiden ominaisuuksia vielä muokata ja alkaa luomaan toiminnallisuutta nappuloihin ja muihin elementteihin.

Toiminnallisuuden tekemiseen ja kommunikointiin eri olioiden välillä QT tarjoaa Signal – Slot -mallin. Signaalit lähettävät tietoa oliosta toiseen silloin kun jokin haluttu tapahtuma tapahtuu, slotit taas ovat normaaleja funktioita, joita signaali kutsuu, eli jotka suoritetaan silloin kun signaali saavuttaa slotin. QT widgeteillä on valmiita signaaleja, mutta niille voidaan myös tehdä omia. Esimerkkinä QT widgetissä oleva nappi voidaan asettaa lähettämään signaali, kun nappia on painettu. Kuten kuvassa 14 on esitetty, signaalit voivat välittää tietoa useammille sloteille ja useampi signaali voi kutsua samaa slottia. Signaalit ja slotit eivät toisaalta ole varsinaisesti sidoksissa toisiinsa. Signaali ei siis tiedä saavuttaako mikään slotti sen lähettämää signaalia, eikä slotti tiedä mitä signaaleja siihen on yhdistetty. Ne toimivat erillään toisistaan eivätkä ole riippuvaisia toisistaan, joka helpottaa esimerkiksi itsenäisten ja helposti uudelleenkäytettävien käyttöliittymän palasten rakentamisessa. (QT Documentation, 2023c.)



Kuva 14. Signal – Slot rakenne (QT Documentation, 2023c).

Käyttöliittymän toiminnallisuuden rakentamista varten tarvitsee siis valita haluamansa widgetin osanen, asettaa se lähettämään signaalia ja kertoa mitä signaalin tapahtuessa pitää suorittaa. Liitteessä 2 on koko main.py tiedoston koodi, johon on kirjoitettu suurin osa käyttöliittymän toiminnallisuudesta.

YOLOv8 -algoritmin läpi ajettua videota varten tehtiin käyttöliittymään oma Ikkunansa. Ikkunassa olevassa ruudussa esitetään alkuperäinen video, lisättynä havaittujen hahmojen ympärille piirretyillä laatikoilla, sekä laatikko, jossa luetellaan havaitut hahmot listana. Seuraavaksi vuorossa oli siten YOLOv8 -algoritmin yhdistäminen käyttöliittymään.

YOLOv8 algoritmin käyttäminen Python ympäristössä on melko yksinkertaista. Tuodaan YOLO kirjasto koodiin, syötetään videota tai kuvia halutusta lähteestä, kerrotaan mitä hahmontunnistuksen mallia halutaan käyttää ja otetaan vastaan tulokset. Videota voidaan syöttää koneella olevista tiedostoista. YOLO ymmärtää suurimman osan eri videoformaateista, kuten .mp4, .avi, .mpeg ja niin edelleen. (Ultralytics 2023c.)

Videota voidaan myös syöttää algoritmille koneessa olevasta kamerasta, OpenCV kirjaston avulla. Myös YouTubesta videota pystytään syöttämään Pythoniin saatavilla olevan PaFy kirjaston avulla (Pafy 2023). Video saadaan OpenCV:tä käyttäen jaettua yksittäisiin kuviin, joita voidaan syöttää YOLO –algoritmille ja joka palauttaa tiedot kuvasta löytämistään hahmoista, niiden koordinaateista kuvassa ja muita tietoja. Tätä videon pilkkomista ja lähettämistä algoritmiin voidaan while –loopin avulla toistaa niin kauan kuin videossa on pituutta tai kunnes video keskeytetään käyttöliittymässä olevaa Stop –nappia painamalla, kuten kuvan 15 koodissa (Ultralytics 2023d).

```
@Slot()
def run(self):

    model = YOLO(self.model)
    source = self.source

    if source == "Webcam":
        source = 0
    if source == "Youtube":
        video = pafy.new(self.url)
        best = video.getbest(preftype="mp4")
        source = best.url


    cap = cv2.VideoCapture(source)
    if not cap.isOpened():
        print("Cannot open camera")
        sum_output = ["ERROR", "Cannot open camera"]
        self.sendResults.emit(sum_output)
        cap.release()
        cv2.destroyAllWindows()

    while True:
        # Capture frame-by-frame
        ret, frame = cap.read()

        # if frame is not read correctly
        if not ret:
            print("Can't receive frame (stream end?). Exiting ...")
            sum_output = ["ERROR", "Can't receive frame (stream end?). Exiting ..."]
            self.sendResults.emit(sum_output)
            cap.release()
            cv2.destroyAllWindows()
            break

        # if frame is read correctly
        if ret :
            results = model.predict(frame, conf=self.confidenceScore)
```

Kuva 15. Videon ajaminen YOLOv8 hahmontunnistuksen malliin

Kun kuva on syötetty algoritmiin, saadaan vastaukseksi siis olio, jossa on algoritmin antamat tiedot kuvasta. Oliosta saadaan muun muassa kuvasta löydettyjä hahmoja ympäröivien laatikoiden koordinaatit pikseleinä, tunnistuksen luotettavuuden suuruus, sekä luokan järjestysnumero, joita kaikkia tarvitaan, jos halutaan piirtää kuvassa olevan objektin ympärille laatikko ja kirjoittaa sen luokan

nimi kuvan yhteyteen kuvan 16 tapaan. Lisäksi objektista on kaivettavissa kaikkien luokkien järjestysnumerot, joiden avulla saadaan tietoon luokan nimi.

yolo.py misc.py

misc.py > draw_boxes

```python
# https://docs.ultralytics.com/usage/python/

import cv2

def draw_boxes(frame, results):
    #Get box measures from results
    xyxy = results.boxes.xyxy.numpy()
    #Get confidence score
    confidence = results.boxes.conf.numpy()
    #Get class ID
    class_id = results.boxes.cls.numpy().astype(int)
    #Get class name
    class_name = [results.names[x] for x in class_id]

    print(class_name)
    return
```

Kuva 16. Results oliosta saatavia tietoja

Näiden tietojen avulla voidaan kuvaan piirtää laatikot hahmojen ympärille ja lisätä laatikoiden yhteyteen tieto tunnistetun hahmon luokasta (Krittapholchai 2023). Lisäksi voidaan esimerkiksi listata käyttöliittymään esille kaikki kuvasta löydetyt hahmot ja niiden tunnistuksen tarkkuus. Koko laatikoiden piirtämisen koodi on löydettävissä misc.py -tiedostosta, joka löytyy liitteestä 3.

4.5.2 Video QT-käyttöliittymässä

Kun videon erilliset kuvat on käytetty algoritmin läpi ja niihin on lisätty laatikot hahmojen ympärille, voidaan kuvat esittää käyttöliittymässä. QT tarjoaa kuvien esittämiseen QPixmap –luokan, joka voi tuoda kuvia eri muodoissa käyttöliittymään näkyville.

Videon lisääminen käyttöliittymään muodosti kuitenkin ongelman, koska koko käyttöliittymä meni jumiin videon pyöriessä. Ohjelma oli tähän mennessä pyörinyt yhdessä prosessissa, jossa kaikki toiminnot suoritetaan toisensa jälkeen. Tämän takia käyttöliittymää ei voinut enää käyttää kuvan pyöriessä ruudulla. Nykyaikainen tietokoneen prosessori voisi käsitellä eri tehtäviä prosessorin eri säikeissä, mutta mikäli tietokoneelle vain annetaan eri tehtäviä suoritettavaksi, suorittaa se ne vain peräkkäisinä tehtävinä yhdessä säikeessä. Qt:ssa on onneksi Multithreading -ominaisuus, jossa tehtäviä voidaan teettää prosessorin toisissa säikeissä. (Qt Documentation 2023d).

Ohjelmaan tehtiin siis erillinen Worker-luokka, jossa videon syöttämistä YOLO-algoritmille suoritetaan (Fitzpatrick 2022c). Worker luokkaan siirrettiin siten kaikki koodi, joka kirjoitettiin algoritmin käyttämistä varten. Worker-luokka asetettiin toimimaan toisessa prosessorin säikeessä ja keskustelu säikeen ja pääohjelman kanssa käytiin Qt:n Signal – Slot menetelmällä (Qt Wiki 2019). Pääohjelmaan main.py tiedostossa luotiin signaali, joka käynnisti Worker-threadissa olevan YOLO-algoritmia käyttävän prosessin. Pääohjelmaan tehtiin myös toinen signaali, jossa algoritmin läpikäyminen voidaan lopettaa, jotta webbikameraakin käytettäessä voidaan kuvan lukeminen lopettaa. Kun algoritmin läpikäyminen lopetetaan, tuhotaan myös koko Worker -olio, jotta säie vapautuu käytettäväksi eikä taustalle jää päällä olevia prosesseja. Koko worker luokan koodi on liitteessä 4.

4.6 Ohjelman paketointi ja installerin teko

Kun käyttöliittymä oli valmis ja se toimi YOLO algoritmiin tehdyllä mallilla, oli aika paketoida sovellus, jotta se on helppo siirtää ja asentaa toisille koneille, siten että se ei vaadi tietokoneelta muiden ohjelmien tai kirjastojen asentelua. Tämän Python ohjelman paketoimiseen käytettiin PyInstaller –nimistä työkalua.

PyInstaller paketoi ohjelman ja kaikki sen vaatimat osaset ja kirjastot yhteen pakettiin. Tämän jälkeen käyttäjä voi siis ajaa ohjelman ilman, että käyttäjän tarvitsee asentaa erikseen koneelleen Python –tulkkia tai muita paketteja ja kirjastoja. PyInstaller paketoi myös siis samaan pakettiin kaikki ohjelmassa käytetyt kirjastot, kuten esimerkiksi tässä projektissa käytetyt YOLO –kirjasto ja sen vaatimat kirjastot. PyInstaller toimii useilla käyttöjärjestelmillä, joihin on Python asennettuna, mutta sillä tehdyt paketit toimivat vain sillä käyttöjärjestelmällä, jolla paketti on tehty. Jos siis ohjelma paketoidaan käyttäen Windows käyttöjärjestelmää, toimii paketti vain Windowsissa. Jos paketointi taasen tehtäisiin MacOS –järjestelmässä, toimisi paketti vain MacOS:lla. (PyInstaller 2023.)

PyInstallerilla ohjelman paketointi onnistuu yhdellä komentorivin komennolla, mutta mikäli pakettiin halutaan mukaan tiedostoja, tai halutaan määrittää pakettiin muodostuvan .exe –tiedoston nimi tai muita yksityiskohtia, pitää paketoinnin asetukset määrittää kuvan 12 mukaiseen main.spec –tiedostoon, joka on kokonaisuudessaan nähtävillä liitteessä 6. Koska projektiin tehty ohjelma tarvitsee erinäisiä tiedostoja, kuten videon, kuvia, YOLO –mallin, pitää ne määrittää main.spec tiedostoon (Fitzpatrick 2022d).

```
# -*- mode: python ; coding: utf-8 -*-


block_cipher = None
import sys ; sys.setrecursionlimit(sys.getrecursionlimit() * 5)

a = Analysis(
    ['main.py'],
    pathex=[],
    binaries=[],
    datas=[("test50.mp4", "."), ("logo.ico", "."), ("logo.jpg", "."), ("best.pt", "."), ("yolov8s.pt", "."), ("yolov8n.pt", "."), ("default.yaml", "./ultralytics/yolo/cfg")],
    hiddenimports=[],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=block_cipher,
    noarchive=False,
)
pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)
```

Kuva 17. Spec –tiedosto, johon paketoinnin asetukset määritettiin

Spec –tiedostoon voi märitellä paljonkin erilaisia asioita, mutta tässä projektissa tiedostoon määriteltiin mitä tiedostoja halutaan pakettiin mukaan paketoitavan, mikä on suoritettavan .exe tiedoston nimi, mitä kuvaa käytetään tuon .exe –tiedoston kuvakkeena. Nämä asiat olisi voinut antaa myös komentoriviltä, liittäen ne paketoimisen aloittavaan komentoon, mutta kun ne ovat liitettynä main.spec tiedostoon voidaan ohjelma paketoida samoilla asetuksilla helposti myöhemminkin (Fitzpatrick 2022d). Eli mikäli ohjelmaa tarvitsee muuttaa tai päivittää, voidaan paketointi tehdä helposti samoilla asetuksilla.

PyInstallerilla tehdyn paketin voisi siis jakaa toiseen tietokoneeseen, mutta paketti koostuu monista eri tiedostoista ja kansioista, joten sen jakaminen on vielä haastavaa. Jakamisen voisi tehdä, vaikka paketoimalla kaikki paketoidun ohjelman osat zip-tiedostoon, mutta ohjelman asentamisen helpottamiseksi tehtiin ohjelmasta vielä installer –tiedosto. Eli samanlainen tiedosto, jolla muutkin Windowsin ohjelmat yleensä koneelle asennetaan.

Installerin avulla ohjelma voidaan siis lähettää ja jakaa toisiin Windows –laitteisiin yhtenä tiedostona. Installer –tiedosto on myös loppukäyttäjän helppo ajaa koneellaan, varsinkin mikäli ohjelma tarvitsee muuta asentamista, kuin vain tiedostojen siirtämisen kansioon. Installer voidaan laittaa asentamaan ohjelmalle pikakuvakkeet työpöydälle ja Windowsin käynnistysvalikkoon, tai tekemään rekisterimerkintöjä, mikäli vaikka ohjelma halutaan käynnistyvän aina Windowsin käynnistyessä.

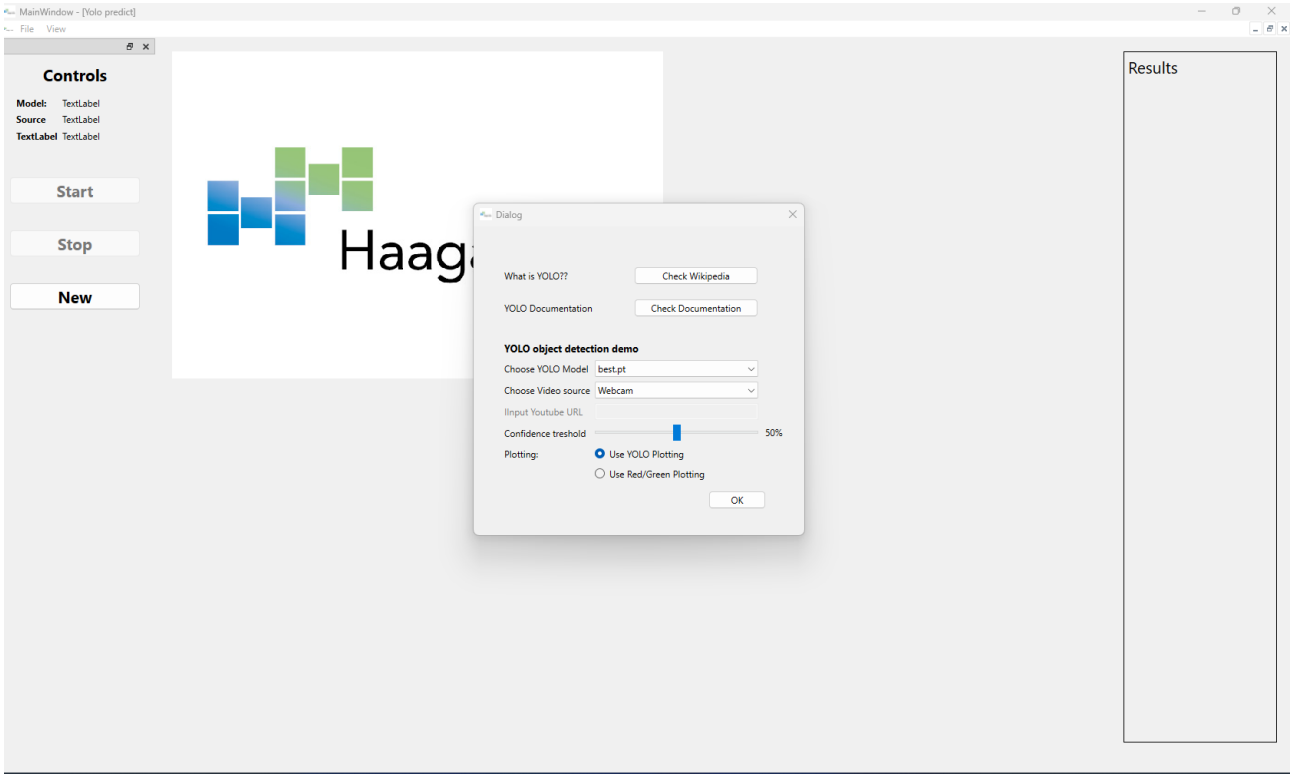
Tässä projektissa installerin tekemiseen käytettiin InstallForge –nimistä ohjelmaa. Muitakin ohjelmia olisi tarjolla, mutta InstallForge on ilmainen ja helppo käyttää graafisesta käyttöliittymästä. (InstallForge 2023.)

Kun installer oli tehty, testattiin vielä sen toimivuus muutamalla eri koneella, jonka jälkeen ohjelma lähetettiin tilaajalle. Ohjelma on tehty siten, että siihen voidaan lisätä käytettäviä YOLO-malleja myöhemmin, lisäämällä ne ohjelman juurikansioon. Siksi ohjelma voitiin lähettää jo nyt tilaajalle testiin sekä asennettavaksi, ja lähettää toinen eri datasetillä koulutettu YOLO-malli myöhemmin.

5 Tuotoksen esittely

Sienien mallin kouluttamisen jälkeen olivat kaikki ohjelmaa varten tarvittavat osaset valmiina. Valmis tuotos on installer –tiedosto, jonka avulla hahmontunnistuksen demo-ohjelma voidaan asentaa Windows 10 käyttöjärjestelmän tietokoneille. Lisäksi ohjelman koodi on julkaistu GitHub –palvelussa, josta sitä voi vapaasti kopioida ja kehittää. Ohjelma ottaa vastaan kuvaa joko tietokoneeseen liitetystä kamerasta, tietokoneelle tallennetusta videosta tai YouTubesta. Ohjelma ajaa tämän jälkeen videon kuva kuvalta valitun YOLO –algoritmin koulutetun mallin lävitse. YOLO –algoritmin antamat tulokset piirretään kuvaan ja esitetään käyttöliittymässä.

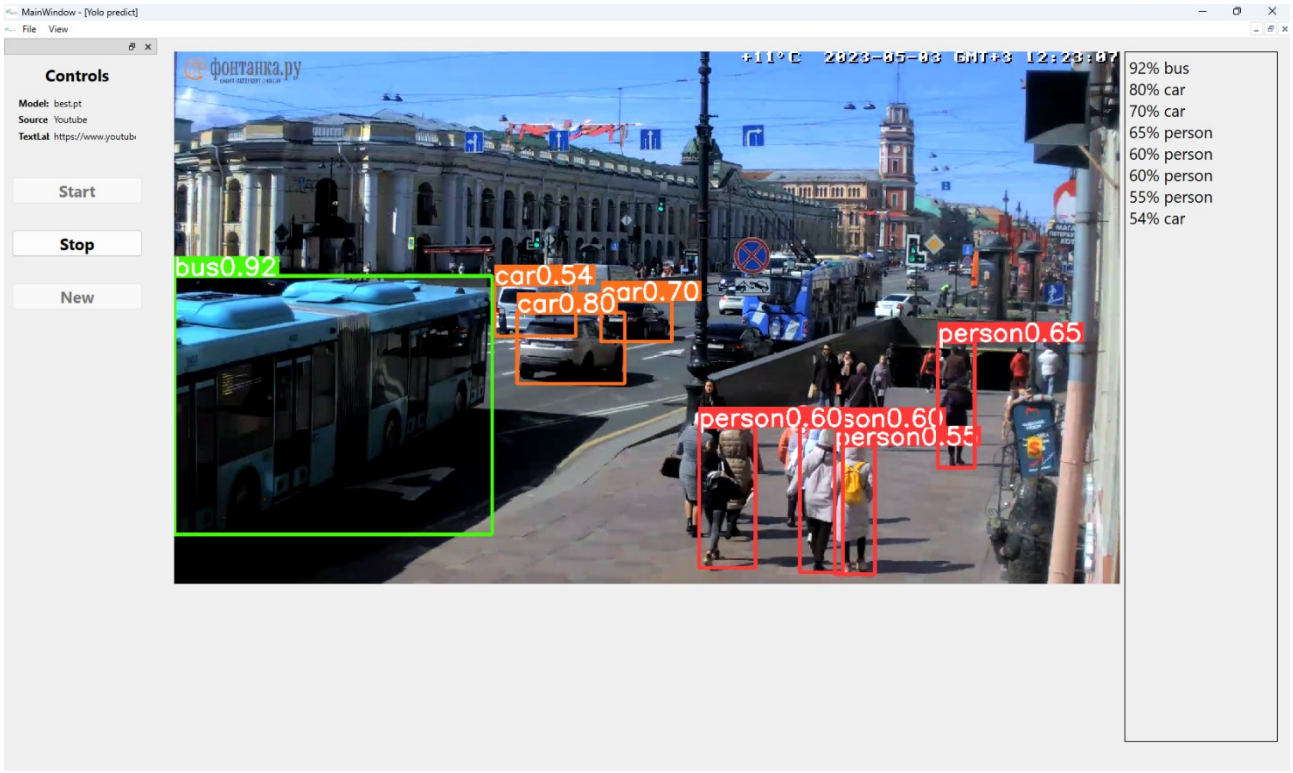
Ohjelman asennuksen jälkeen ja ohjelmaa avatessa ensimmäiseksi vastaan tulee kuvan 17 näköinen ikkuna, jossa halutut asetukset valitaan.



Kuva 18. Ohjelman aloitusikkuna.

Ikkunasta valitaan ensiksi mitä koneella tallennettuna olevaa YOLO-mallia käytetään hahmontunnistukseen. Tämän jälkeen voidaan valita kuvan lähde, ja mikäli kyseessä on YouTube video, tulee videon verkko-osoite syöttää myös. Sitten valitaan vielä millä tavoilla hahmojen ympärille laatikoita piirretään, jonka jälkeen voidaan siirtyä seuraavaan vaiheeseen.

Hahmojen tunnistaminen videosta voidaan aloittaa painamalla Start-näppäintä. Tämän jälkeen ruutu on kuvan 18 tapainen, eli ohjelma näyttää kuvan tunnistettuine hahmoineen ruudulla, sekä hahmoluokat ja varmuus luokan oikeellisuudesta prosentteina listalla.



Kuva 19. Hahmontunnistus käynnissä.

Suurimman tilan vie luonnollisesti kuva ja tulosten esittely, mutta ohjelmaan on luotu myös muutaman näppäimen ohjauspaneeli. Ohjauspaneelissa on näkyvillä valittu malli ja kuvan lähde, sekä siitä löytyvät näppäimet toiston pysäyttämiseen ja uusien asetusten valintaan. Ohjauspaneeli voidaan myös halutessa siirtää tai poistaa ruudulta, jolloin ruudulle saadaan enemmän tilaa esitettävälle kuvalle. Ohjauspaneelin saa takaisin näkyviin vetovalikosta ikkunan ylälaidassa. Muuten ohjelma toimii kuten muutkin Windows ympäristön ikkunat, näyttäen melko perinteiseltä, eikä ehkä moderneimmalta, Windows ohjelmalta.

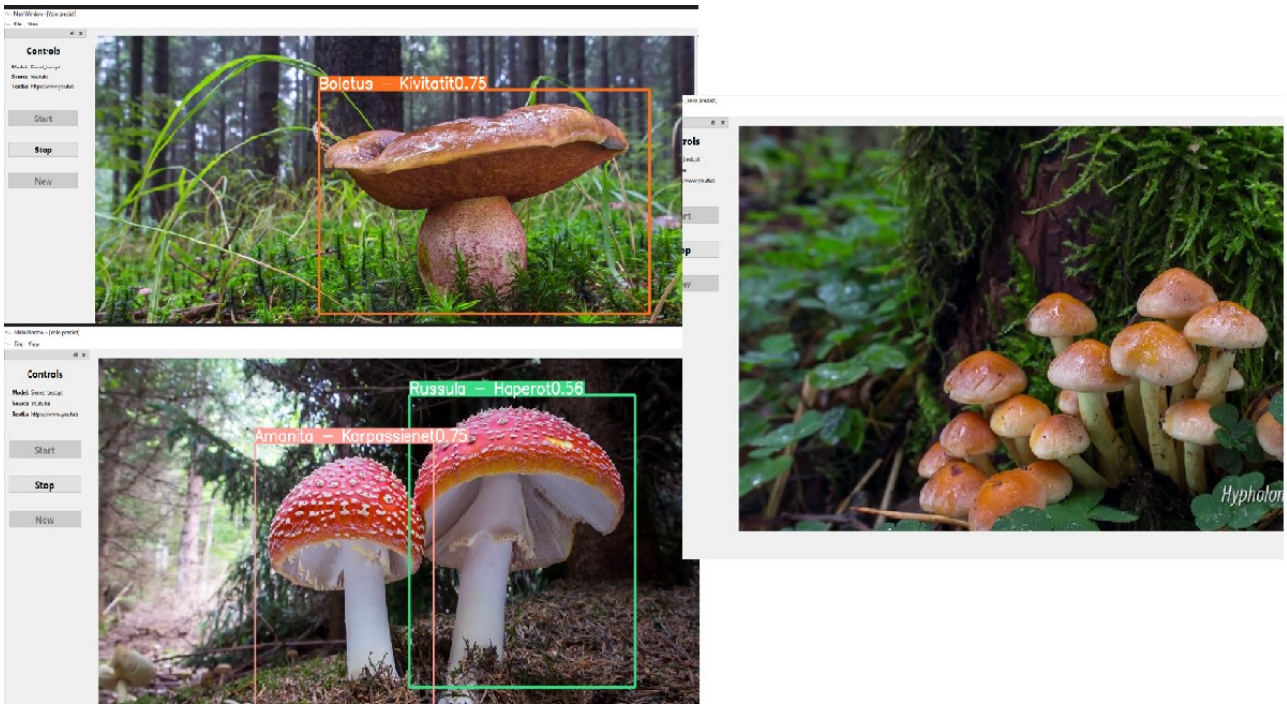
6 Pohdinta

Tämä opinnäytetyö tarjosi tekijälleen paljon tietoja ja kurkistuksia eri teknologioihin. Opinnäytetyön teon aikana tuli opiskella niin konenäön ja hahmontunnistuksen tekniikoista ja tekoälyn kouluttamisesta, kuin myös ohjelmoinnista, Python kirjastojen käytöstä sekä käyttöliittymän suunnittelusta ja valmistuksesta. Aiheiden käsittely jäi kuitenkin hieman pintapuoliseksi esittelyksi tai perehdytykseksi aiheeseen. Projektin jokaiseen vaiheeseen tarkemmin syventymällä, olisi jokaisesta vaiheesta varmasti riittänyt tutkittavaa omiksi opinnäytetöikseen. Onneksi kuitenkin aihe oli rajattu ja haluttu lopputuote varsin hyvin tiedossa, joten aiheiden käsittelyn syvyys riitti tämän projektin tavoitteen saavuttamiseen. Ja kuten mainittua, tekijä pääsi näkemään ja opiskelemaan monen eri teknologian aiheita, sekä kenties kirkastamaan suuntaansa tulevaisuuden osaamisen kehittämisensä osalta. Työ lisäsi tekijän innostuneisuutta aiheisiin, joita työssä käsiteltiin.

Työ lienee myös hyvin jatkojalostettavissa, mikäli työn osa-alueita halutaan tutkia lisää. Käytettyä algoritmia voidaan kenties vaihtaa sovelluksen käyttökohteen mukaan. Hahmontunnistuksen mallia voidaan kouluttaa tarkemmaksi ja esimerkiksi tutkia minkälaisilla dataseteillä tuloksesta saataisiin paras mahdollinen. Hahmontunnistuksen malleja voidaan myös kouluttaa eri tarkoituksiin ja tunnistamaan eri hahmoluokkia. Käyttöliittymää voidaan varmasti parantaa ja kaunistaa, sekä se voidaan tehdä erilaisiin sovelluksiin sopivaksi. Yksi todella hyvä kehitysidea voisi olla tehdä hahmontunnistuksen algoritmista selkeämmin oma backendinsä, jonka kanssa voitaisiin keskustella eri käyttöliittymistä. Tällöin yhtä hahmontunnistuksen backendiä voitaisiin hyödyntää eri sovelluksissa, ja tehdä vaikka mobiiliapplikaatio sienien tunnistukseen.

Yleisen demon kanssa testinä koulutettu malli, joka tunnistaa erilaisia hahmoja ihmisistä matkalaukkuihin ja autoista puhelimiin on paljon parempi ja ajaa asiansa varsin hyvin. Tällöin demon voidaan helpommin ajaa erilaisia videoita ja kuvaa kamerasta, ja täten esiteltyä hahmontunnistuksen kykyjä. Sienien tunnistaminen sopisi paremmin esimerkiksi puhelinapplikaatioksi, jossa puhelimella otettu kuva voitaisiin ajaa suoraan algoritmiin, eikä sitä tarvitse näyttää puhelimen ruudulta tietokoneen kameralle. Toki demoon voidaan ajaa videota tai kuvia eri sienistä, mutta ehkä ideana se on näin jälkikäteen ajateltuna hieman hassu.

Hahmontunnistuksen mallin kouluttamien sienien tunnistukseen ei siis aiheena ollut tähän projektiin ollut paras mahdollinen. Kouluttamisessa ei myöskään onnistuttu kovin hyvin. Tämän voi huomata myös ajettaessa sienien kuvia malliin ja tarkastellessa ohjelman antamia tuloksia niistä. Kuvassa 20 on esiteltynä ohjelman antamia tuloksia erilaisten sienien kuvista. Ison osan sienistä ohjelma tunnistaa vallan hyvin, kuten kuvassa olevan kivitatin, mutta ohjelma antaa vääriä vastauksia joissain sienissä, kuten kuvan 20 kärpässienen tapauksessa. Joissain tapauksissa, ja varsinkin sienten ollessa ryppäässä, ei ohjelma tunnista sieniä välttämättä ollenkaan.



Kuva 20. Sienien tunnistusta ohjelmalla.

Osa sienistä melko saman näköisiä, ja vaativat tunnistajaltaan kolmiulotteisen näkemisen lisäksi tuntoaistia sekä jopa sienen halkaisemista, jotta tuloksesta saataisiin tarpeeksi hyvä. Mikäli datasetti olisi ollut huomattavasti isompi, olisi toki tilanne voinut olla hieman erilainen. Mallia kouluttaessa huomattiin myös, että sieniä tunnistavaa ohjelmaa on koetettu tehdä aiemminkin, mutta laihoin tai jopa hengenvaarallisin tuloksin (Vincent 2017). Sieniä on myös todella monta eri lajia ja mikäli hahmontunnistusmallille ei ole jotakin yksittäistä lajia opetettu, ei se sitä tunne. Mikäli malli ei taas tunne lajia, mutta hahmo näyttää samankaltaiselta kuin jokin toinen sieni, tekee se joka tapauksessa ehdotuksen lajista, ehdottaen siis väärää lajia. Lisäksi sienet esiintyvät välillä ryppäissä yhteen kasvaneina ja välillä hieman toisistaan erillään, joten hahmot saattavat olla hyvin erilaisia. Ryppäissä saattaa myös olla eri kasvuvaiheessa olevia sieniä, toisilla on jo isoksi kasvanut heltta, kun taas toiset ovat vielä pallomaisempia. Hahmontunnistusmallin kouluttamista varten tarvittaisiin

siis todella laaja datasetti. Hahmontunnistuksen mallin kouluttaminen on muutenkin niin laaja aihe, että siitä saisi varmasti tehtyä oman opinnäytetyön.

Tällaiseen tässä opinnäytetyöprosessissa tehtyyn ohjelmaan sopisi toisaalta hyvin esimerkiksi jokin liikkuvaa kuvaa tulkitseva malli. Ohjelmaa voisi esimerkiksi kehittää tunnistamaan liikennemerkkejä, jolloin ohjelmaa voisi soveltaa, vaikka auton apuvälineenä. Toki tämä vaatisi autoon asennettuna muokkauksia paljonkin, niin käyttöliittymään kuin hahmontunnistuksen malliinkin. Mutta hahmontunnistusta esittelevänä demona se voisi toimia hyvin.

Erilaisiin kuvantunnistuksen sovelluksiin tulisi myös muistaa valita sopivat teknologiat sekä kehittää sopivat mallit ja käyttöliittymät. Kuten aikaisemmin raportissa on mainittu, sopii YOLO –algoritmi hyvin liikkuvaa kuvaa käsitteleviin malleihin ja sillä voisi esimerkiksi rakentaa turvakameroihin systeemin, joka tunnistaisi milloin paketti on toimitettu kotiovelle. Mutta vielä tarkempaa tarkkuutta ja enemmän laskentatehoa vaativiin sovelluksiin voisi harkita jotain toista teknologiaa.

Lopputuloksen käytettävyyden kannalta hahmontunnistuksen mallin kohteen valinnalla onneksi ole suurta roolia. Tuotoksesta tuli helposti jaettava ja asennettava, sitä voi käyttää monella eri koneella Windows-ympäristössä ja se voidaan pienellä vaivalla siirtää toimimaan myös muissa ympäristöissä. Tuotoksella voidaan esitellä hahmontunnistusta ja sitä, miten tietokone näkee kuvasta hahmoja. Näillä mittareilla tuotos on siten onnistunut. Toivottavasti tuotoksesta on myös apua työelämässä ja uusien työhaasteiden etsimisessä. Tuotoksessa käsiteltyjen asioiden osaajille luulisi olevan markkinoita ja niiden parissa työskentely olisi varmasti mukavaa.

Lähteet

Alander, J & Honkela, T & Jakobsson, M. 1996. Neuroverkot: johdatus moderniin tekoälyyn. Luettavissa: http://users.ics.aalto.fi/tho/stes/step96/honkela2/ Luettu: 19.4.2023.

Allwright, S. 2022. What is a good F1 score and how do I interpret it? Luettavissa: https://stephenallwright.com/good-f1-score/ Luettu: 12.05.2023.

Bhalerao, C. 2023. YOLO v8! The real state-of-the-art? Luettavissa: https://medium.com/mlearning-ai/yolo-v8-the-real-state-of-the-art-eda6c86a1b90 Luettu: 19.04.2023.

Bonner, A. 2019. Getting Started with Google Colab. Luettavissa: https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c Luettu: 19.04.2023.

Bradski, G., Kaehler, A. 2008. Learning OpenCV. O'Reilly Media Inc. Sebastopol.

Burns, E. 2022. What is AMD? Luettavissa: https://techmonitor.ai/what-is/what-is-amd-4954145 Luettu: 03.05.2023.

Choi, R., Coyner, A., Kalpathy-Cramer, J., Chiang, M & Campell, P. 2020. Introduction to machine learning, neural networks, and deep learning. Trans Vis Sci Tech. Luettavissa: https://tvst.arvojournals.org/article.aspx?articleid=2762344 Luettu: 03.05.2023.

Culjak, I., Abram, D., Pribanic, T., Dzapo, H & Cifrek, M. 2012. "A brief introduction to OpenCV," 2012 Proceedings of the 35th International Convention MIPRO. Opatija, Croatia.

Czakon, J. 2023. F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose? Luettavissa: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc Luettu: 12.05.2023.

Francesco & Solawetz, J. 2023. What is YOLOv8? The Ultimate Guide. Luettavissa: https://blog.roboflow.com/whats-new-in-yolov8/#:~:text=YOLOv8%20was%20launched%20on%20January%2010th%2C%202023 Luettu: 19.04.2023.

Fitzpatrick, M. 2022a First steps with Qt Designer. Luettavissa: https://www.pythonguis.com/tutorials/pyside6-first-steps-qt-designer/ Luettu: 19.04.2023.

Fitzpatrick, M. 2022b. Creating your first app with PySide6 Luettavissa: https://www.pythonguis.com/tutorials/pyside6-creating-your-first-window/ Luettu: 19.04.2023.

Fitzpatrick, M. 2022c. Multithreading PySide2 applications with QThreadPool. Luettavissa: https://www.pythonguis.com/tutorials/multithreading-pyside-applications-qthreadpool/ Luettu: 19.04.2023.

Fitzpatrick, M. 2022d. Packaging PySide6 applications for Windows with PyInstaller & InstallForge. Luettavissa: https://www.pythonguis.com/tutorials/packaging-pyside6-applications-windows-pyinstaller-installforge/ Luettu: 19.04.2023.

Goel, S. 2023. What does Nvidia do: Business model analysis. Luettavissa: https://thestrategystory.com/2023/01/05/what-does-nvidia-do-business-model-analysis/ Luettu: 03.05.2023.

Google. 2023. Google Colaboratory FAQ. Luettavissa: https://research.google.com/colaboratory/faq.html Luettu: 19.04.2023.

Graupe, D. 2016. Deep learning neural networks. World Scientific Publishing Co. Pte. Lte. Singapore.

Heikkilä, T. 2020. Qt ja Revenio ovat pörssin teknologiatähdet. Sijoittaja. Luettavissa: https://www.sijoittaja.fi/237536/qt-ja-revenio-ovat-porssin-teknologiatahdet/ Luettu: 19.04.2023.

IBM. 2023. What is machine learning? Luettavissa: https://www.ibm.com/topics/machine-learning Luettu: 19.04.2023.

InstallForge. 2023. Features. Luettavissa: https://installforge.net/features/ Luettu 19.04.2023.

Jain, S. 2023. YOLOv8 The new State of The Art Detector? Luettavissa: https://medium.com/the-modern-scientist/yolov8-the-new-state-of-the-art-detector-89f627ad17aa Luettu 19.04.2023.

Kielitoimiston sanakirja. 2022. Konenäkö. Luettavissa: https://www.kielitoimistonsanakirja.fi/#/konen%C3%A4k%C3%B6 Luettu: 19.04.2023.

Kundu, R. 2023. YOLO: Algorithm for Object Detection Explained. Luettavissa: https://www.v7labs.com/blog/yolo-object-detection Luettu: 19.04.2023.

Krittapholchai, C. Build Object Detection GUI with YOLOv8 and PySimpleGUI. Luettavissa: https://medium.com/@chanon.krittapholchai/build-object-detection-gui-with-yolov8-and-pysimplegui-76d5f5464d6c Luettu: 19.04.2023.

Lee, C. 2021. How many images do you need for object detection? Medium.com. Luettavissa: https://changsin.medium.com/how-many-images-do-you-need-for-object-detection-d33185629843 Luettu: 19.04.2023.

Liu, Z. 2023. AMD ROCm Comes to Windows on Consumer GPUs. Tom's Hardware. Luettavissa: https://www.tomshardware.com/news/amd-rocm-comes-to-windows-on-consumer-gpus Luettu: 19.04.2023.

Mahmoud, H. 2022. What is Kaggle? Kaggle.com. Luettavissa: https://www.kaggle.com/general/328265 Luettu: 19.04.2023.

Mathworks. 2023. Object Recognition, 3 things you need to know. Luettavissa: https://www.mathworks.com/solutions/image-video-processing/object-recognition.html Luettu: 19.04.2023.

Minnalearn. 2023. Elements of AI. Luettavissa: https://course.elementsofai.com/fi/5/1 Luettu: 19.04.2023.

Nvidia. 2023. Pytorch. Luettavissa: https://www.nvidia.com/en-us/glossary/data-science/pytorch/ Luettu: 19.04.2023.

Pafy. 2023. Pafy Documentation. Luettavissa: https://pythonhosted.org/pafy/ Luettu: 19.04.2023.

Park, S-Y. & Baek. S-H. 2013. Handbook of 3D Machine Vision: Optical Metrology and Imaging. Taylor & Francis Group. Boca Raton.

Petrovicheva, A. 2020. OpenCV is to change the license to Apache 2. Luettavissa: https://opencv.org/opencv-is-to-change-the-license-to-apache-2/ Luettu: 03.05.2023.

PyInstaller. 2023. PyInstaller manual. Luettavissa: https://pyinstaller.org/en/stable/ Luettu: 19.04.2023.

Python Wiki. 2023. GUI Programming for Python Luettavissa: https://wiki.python.org/moin/GuiProgramming Luettu: 19.04.2023.

Qt Documentation. 2023a. Qt Widgets. Luettavissa: https://doc.qt.io/qt-6/qtwidgets-index.html Luettu: 03.05.2023.

Qt Documentation. 2023b. Qt Designer manual. Luettavissa: https://doc.qt.io/qt-6/qtdesigner-manual.html Luettu: 19.04.2023.

Qt Documentation. 2023c. Signals & Slots. Luettavissa: https://doc.qt.io/qt-6/signalsandslots.html Luettu: 19.04.2023.

Qt Documentation. 2023d. Multithreading Technologies in Qt. Luettavissa: https://doc.qt.io/qt-6/threads-technologies.html Luettu 19.04.2023.

Qt Group. 2023a. Qt for Python. Luettavissa: https://www.qt.io/qt-for-python Luettu: 19.04.2023.

Qt Group .2023b. Qt Features, Framework Essentials, Modules, Tools & Add-Ons. Luettavissa: https://www.qt.io/product/features Luettu: 19.04.2023.

Qt Group. 2023c. Qt Licensing. Luettavissa: https://www.qt.io/licensing/ Luettu: 19.04.2023.

Qt Wiki. 2019. Qt for Python Signals and Slots. Luettavissa: https://wiki.qt.io/Qt_for_Python_Signals_and_Slots Luettu: 19.04.2023.

Qt Wiki. 2022. About Qt. Luettavissa: https://wiki.qt.io/About_Qt Luettu: 03.05.2023.

Rath, S. 2023. YOLOv8 Ultralytics: State-of-the-Art YOLO Models Luettavissa: https://learnopencv.com/ultralytics-yolov8/ Luettu: 19.04.2023.

Redmond, J., Divvala, S., Girshick, R & Farhadi, A. 2016. You Only Look Once: Unified, Real-Time Object Detection. Luettavissa: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf Luettu: 19.04.2023.

Shivaprasad, P. A Comprehensive Guide To Object Detection Using YOLO Framework — Part I. Luettavissa: https://towardsdatascience.com/object-detection-part1-4dbe5147ad0a Luettu: 12.05.2023.

Steen, D. 2020. Precision-Recall Curves. Luettavissa: https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248 Luettu: 12.05.2023.

Solegaonkar, V. 2019. Introduction to PyTorch. Luettavissa: https://towardsdatascience.com/introduction-to-py-torch-13189fb30cb3 Luettu: 03.05.2023.

Solovev, A. 2022. Using Qt: 10 Famous and Successful Cases. Luettavissa: https://hackernoon.com/using-qt-10-famous-and-successful-cases Luettu 19.04.2023.

Technostacks. 2023. YOLO Vs. SSD: Choice of a Precise Object Detection Method. Luettavissa: https://technostacks.com/blog/yolo-vs-ssd/ Luettu: 19.04.2023.

Terra, J. 2023. Keras vs Tensorflow vs Pytorch: Key Differences Among Deep Learning. Luettavissa: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article Luettu: 19.04.2023.

Ultralytics. 2023a. A brief history of YOLO Luettavissa: https://docs.ultralytics.com/ Luettu: 19.04.2023.

Ultraytics. 2023b. Quickstart. Luettavissa: https://docs.ultralytics.com/quickstart/ Luettu: 19.04.2023.

Ultralytics. 2023c. Predict. Luettavissa: https://docs.ultralytics.com/modes/predict/ Luettu: 19.04.2023.

Ultralytics. 2023d. Python Usage. Luettavissa: https://docs.ultralytics.com/usage/python/ Luettu: 19.04.2023.

Ultralytics. 2023e. Train Custom Data. Luettavissa: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data Luettu: 19.04.2023.

Vincent, J. 2017. A ‘potentially deadly’ mushroom-identifying app highlights the dangers of bad AI. The Verge. Luettavissa: https://www.theverge.com/2017/7/28/16054834/mushroom-identifying-app-machine-vision-ai-dangerous Luettu: 19.04.2023.

Zvornicanin, E. 2022. What is YOLO algorithm? Luettavissa: https://www.baeldung.com/cs/yolo-algorithm Luettu: 19.04.2023.

Liitteet

Liite 1. Projektin koodi

Kaikki projektin koodi on saatavilla osoitteessa: https://github.com/koulujussi/object_detection_demo

Liite 2. main.py

```
#https://www.pythonguis.com/tutorials/pyside6-first-steps-qt-designer/
#https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QMainWindow.html
#https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/
#https://doc.qt.io/qtforpython/PySide6/QtGui/QDesktopServices.html
#https://doc.qt.io/qt-6/widget-classes.html#basic-widget-classes

import sys
from PySide6.QtWidgets import QWidget, QMainWindow, QApplication, QDialog
from PySide6.QtCore import Slot, Qt
from PySide6.QtGui import QPixmap, QResizeEvent, QDesktopServices, QIcon

from mainWindow import Ui_MainWindow
from controls import Ui_controls
from picture_window import Ui_pictureFrame
from init_dialog import Ui_Dialog
from yolo_worker import Worker
from misc import *

class Controls(QWidget, Ui_controls):

    def __init__(self, parent = None):
        super(Controls, self).__init__()
        self.setupUi(self)

        self.stopButton.setEnabled(False)


class Picture(QWidget, Ui_pictureFrame):

    def __init__(self, parent = None):
        super(Picture, self).__init__()
        self.setupUi(self)

class Setup(QDialog, Ui_Dialog):

    def __init__(self, parent= None):
        super(Setup, self).__init__()
        self.setupUi(self)
        self.label_youtube.setEnabled(False)
        self.youtubeURL.setEnabled(False)

        #Fetch available models
        models = find_files("pt")
        for x in models:
            self.modelListBox.addItem(x)

```

```
47          #Add sources and fetch available videos
48          self.sourceListBox.addItem("Webcam")
49          self.sourceListBox.addItem("Youtube")
50          videoFormats = [
51              "asf",
52              "avi",
53              "gif",
54              "m4v",
55              "mkv",
56              "mov",
57              "mp4",
58              "mpeg",
59              "mpg",
60              "ts",
61              "wmv",
62              "webm"
63          ]
64          for format in videoFormats:
65              sources = find_files(format)
66              for video in sources:
67                  self.sourceListBox.addItem(video)
68
69          self.confidenceLabel.setText(str(self.confidenceSlider.value()) + "%")
70          self.confidenceSlider.valueChanged.connect(self.slider_moved)
71
72          self.sourceListBox.currentTextChanged.connect(self.checkYoutube)
73
74          self.whatIsYoloButton.clicked.connect(self.yolo_button_clicked)
75          self.pushButton.clicked.connect(self.documentation_button_clicked)
76
77      def slider_moved(self):
78          print(self.confidenceSlider.value())
79          self.confidenceLabel.setText(str(self.confidenceSlider.value()) + "%")
80
81      def yolo_button_clicked(self):
82          QDesktopServices.openUrl("https://en.wikipedia.org/wiki/Object_detection")
83      def documentation_button_clicked(self):
84          QDesktopServices.openUrl("https://docs.ultralytics.com/")
85
86      def checkYoutube(self):
87          if self.sourceListBox.currentText() == "Youtube":
88              self.youtubeURL.setEnabled(True)
89              self.label_youtube.setEnabled(True)
90          else:
91              self.youtubeURL.setEnabled(False)
92              self.label_youtube.setEnabled(False)
93
94  class MainWindow(QMainWindow, Ui_MainWindow):
95      def __init__(self):
96          super(MainWindow, self).__init__()
97          self.setupUi(self)
98          self.showMaximized()
99
```

```
100         #Adding controls widget
101         self.controls = Controls(self)
102         self.dockControls.setWidget(self.controls)
103         self.controls.startButton.setEnabled(False)
104 
105         #Adding picture area
106         self.picture = Picture(self)
107         self.mdiArea.addSubWindow(self.picture)
108         self.picture.showMaximized()
109 
110         #Adding button to toggle control widget
111         self.controls_button = self.dockControls.toggleViewAction()
112         self.controls_button.setText("Show controls")
113         self.menuView.addAction(self.controls_button)
114 
115         #To Close program from File -menu
116         self.actionExit.triggered.connect(lambda: self.close())
117 
118         #Preparing for extra thread for video
119         self.yolo_worker = None
120 
121         #Setting picture to picturearea
122         self.videoHeight = 640
123         self.videoWidth = 480
124         self.img = QPixmap("./logo.jpg")
125         self.img = self.img.scaled(self.videoHeight,self.videoWidth, Qt.KeepAspectRatio)
126         self.picture.videoFrame.setPixmap(self.img)
127 
128         #Showing initial setup dialog
129         self.setup = Setup(self)
130         self.setup.buttonBox.accepted.connect(self.setOptions)
131         self.setup.exec()
132 
133         self.controls.startButton.clicked.connect(self.start_button_clicked)
134         self.controls.stopButton.clicked.connect(self.stop_button_clicked)
135         self.controls.newButton.clicked.connect(lambda: self.setup.exec())
136 
137     def setOptions(self):
138         self.model = self.setup.modelListBox.currentText()
139         self.controls.chosenModel.setText(self.model)
140         self.source = self.setup.sourceListBox.currentText()
141         self.controls.chosenSource.setText(self.source)
142         if self.setup.youtubeURL.isEnabled():
143             self.youtubeUrl = self.setup.youtubeURL.text()
144             self.controls.youtubeUrl.setText(self.youtubeUrl)
145             self.controls.youtubeLabel.setVisible(True)
146         else:
147             self.youtubeUrl = ""
148             self.controls.youtubeUrl.setText(self.youtubeUrl)
149             self.controls.youtubeLabel.setVisible(False)
150 
```

```
151             try:
152                 self.picture.isVisible() == True
153             except:
154                 self.picture = Picture(self)
155                 self.mdiArea.addSubWindow(self.picture)
156                 self.picture.showMaximized()
157                 self.videoHeight = 640
158                 self.videoWidth = 480
159                 self.img = QPixmap("./logo.jpg")
160                 self.img = self.img.scaled(self.videoHeight,self.videoWidth, Qt.KeepAspectRatio)
161                 self.picture.videoFrame.setPixmap(self.img)
162 
163             self.YOLO_plotting = self.setup.radioButtonYOLOPlotting.isChecked()
164 
165             self.controls.startButton.setEnabled(True)
166 
167             self.picture.videoFrame.resizeEvent = self.video_resize
168 
169             self.confidenceTreshold = self.setup.confidenceSlider.value()
170 
171             self.videoHeight = self.picture.videoFrame.height()
172             self.videoWidth = self.picture.videoFrame.width()
173 
174 
175         def video_resize(self, resizeEvent:QResizeEvent):
176             self.videoWidth = self.picture.videoFrame.width()
177             self.videoHeight = self.picture.videoFrame.height()
178             if self.yolo_worker != None:
179                 self.yolo_worker.change_video_size(self.videoWidth, self.videoHeight)
180 
181 
182         @Slot(list)
183         def updateResults(self, result_list):
184 
185             text = ""
186             if result_list[0] == "ERROR":
187                 text = f"{result_list[0]}: {result_list[1]}"
188             else:
189                 for x in result_list:
190                     class_name, confidence = x
191                     confidence = "{0:.0%}".format(confidence)
192                     text = text + f"{confidence} {class_name}" + "\n"
193 
194             self.picture.resultsFrame.setText(text)
195 
```

```
196
197
198     def start_button_clicked(self):
199         self.controls.startButton.setEnabled(False)
200         self.controls.stopButton.setEnabled(True)
201         self.controls.newButton.setEnabled(False)
202         try:
203             self.picture.isVisible() == True
204         except:
205             self.picture = Picture(self)
206             self.mdiArea.addSubWindow(self.picture)
207             self.picture.showMaximized()
208         self.yolo_worker = Worker(self.model, self.confidenceTreshold, self.source, self.youtubeUrl, self.videoWidth, self.videoHeight, self.YOLO_plotting)
209         self.yolo_worker.changePixmap.connect(self.picture.videoFrame.setPixmap)
210         self.yolo_worker.sendResults.connect(self.updateResults)
211         self.yolo_worker.start()
212
213     def stop_button_clicked(self):
214         self.controls.startButton.setEnabled(True)
215         self.controls.stopButton.setEnabled(False)
216         self.controls.newButton.setEnabled(True)
217         self.picture.resultsFrame.setText("Results")
218         self.yolo_worker.stop()
219         self.yolo_worker.quit()
220         self.yolo_worker.terminate()
221
222
223
224 app = QApplication(sys.argv)
225 app.setWindowIcon(QIcon('logo.ico'))
226
227 window = MainWindow()
228 window.show()
229 app.exec()
```

Liite 3. misc.py tiedoston koodi

```python
from glob import glob
import cv2

#https://docs.python.org/3/library/glob.html
def find_files(type):
    results = glob(f"./*.{type}")

    models = [ x.replace(".\\", "") for x in results]
    return models

#This script for drawing the boxes to frames
def draw_boxes(frame, results):
    # https://docs.ultralytics.com/usage/python/
    # https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html
    # https://medium.com/@chanon.krittapholchai/build-object-detection-gui-with-yolov8-and-pysimplegui-76d5f5464d6c

    #Get box measures from results
    xyxy = results.boxes.xyxy.numpy()
    #Get confidence score
    confidence = results.boxes.conf.numpy()
    #Get class ID
    class_id = results.boxes.cls.numpy().astype(int)
    #Get class name
    class_name = [results.names[x] for x in class_id]
    # Pack together for easy use
    sum_output = list(zip(class_name, confidence,xyxy))

    for run_output in sum_output :
        #Unpack
        class_name, confidence, xyxy = run_output

        #Color for the box, depending on the confidence score
        if confidence > 0.8:
            box_color = (0, 100, 0)
            text_color = (0,100,0)
        else:
            box_color = (0, 0, 255)
            text_color = (0,0,255)
        #Draw box around the object
        first_half_box = (int(xyxy[0]),int(xyxy[1]))
        second_half_box = (int(xyxy[2]),int(xyxy[3]))
        cv2.rectangle(frame, first_half_box, second_half_box, box_color, 2)

        #Create text
        text_print = '{label} {con:.2f}'.format(label = class_name, con = confidence)
        # Specify where the box is drawn
        text_location = (int(xyxy[0]), int(xyxy[1] - 10 ))

        #Write text to image
        cv2.putText(frame, text_print ,text_location
                    , cv2.FONT_HERSHEY_SIMPLEX , 1
                    , text_color, 2 ,cv2.LINE_AA)
    return frame
```

Liite 4. yolo_worker.py tiedoston koodi

```
#https://www.pythonguis.com/tutorials/pyside6-first-steps-qt-designer/
#https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QMainWindow.html
#https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/
#https://doc.qt.io/qtforpython/PySide6/QtGui/QDesktopServices.html
#https://doc.qt.io/qt-6/widget-classes.html#basic-widget-classes

import sys
from PySide6.QtWidgets import QWidget, QMainWindow, QApplication, QDialog
from PySide6.QtCore import Slot, Qt
from PySide6.QtGui import QPixmap, QResizeEvent, QDesktopServices, QIcon

from mainWindow import Ui_MainWindow
from controls import Ui_controls
from picture_window import Ui_pictureFrame
from init_dialog import Ui_Dialog
from yolo_worker import Worker
from misc import *

class Controls(QWidget, Ui_controls):

    def __init__(self, parent = None):
        super(Controls, self).__init__()
        self.setupUi(self)

        self.stopButton.setEnabled(False)


class Picture(QWidget, Ui_pictureFrame):

    def __init__(self, parent = None):
        super(Picture, self).__init__()
        self.setupUi(self)

class Setup(QDialog, Ui_Dialog):

    def __init__(self, parent= None):
        super(Setup, self).__init__()
        self.setupUi(self)
        self.label_youtube.setEnabled(False)
        self.youtubeURL.setEnabled(False)

        #Fetch available models
        models = find_files("pt")
        for x in models:
            self.modelListBox.addItem(x)
```

```
46
47          #Add sources and fetch available videos
48          self.sourceListBox.addItem("Webcam")
49          self.sourceListBox.addItem("Youtube")
50          videoFormats = [
51              "asf",
52              "avi",
53              "gif",
54              "m4v",
55              "mkv",
56              "mov",
57              "mp4",
58              "mpeg",
59              "mpg",
60              "ts",
61              "wmv",
62              "webm"
63          ]
64          for format in videoFormats:
65              sources = find_files(format)
66              for video in sources:
67                  self.sourceListBox.addItem(video)
68
69          self.confidenceLabel.setText(str(self.confidenceSlider.value()) + "%")
70          self.confidenceSlider.valueChanged.connect(self.slider_moved)
71
72          self.sourceListBox.currentTextChanged.connect(self.checkYoutube)
73
74          self.whatIsYoloButton.clicked.connect(self.yolo_button_clicked)
75          self.pushButton.clicked.connect(self.documentation_button_clicked)
76
77      def slider_moved(self):
78          print(self.confidenceSlider.value())
79          self.confidenceLabel.setText(str(self.confidenceSlider.value()) + "%")
80
81      def yolo_button_clicked(self):
82          QDesktopServices.openUrl("https://en.wikipedia.org/wiki/Object_detection")
83      def documentation_button_clicked(self):
84          QDesktopServices.openUrl("https://docs.ultralytics.com/")
85
86      def checkYoutube(self):
87          if self.sourceListBox.currentText() == "Youtube":
88              self.youtubeURL.setEnabled(True)
89              self.label_youtube.setEnabled(True)
90          else:
91              self.youtubeURL.setEnabled(False)
92              self.label_youtube.setEnabled(False)
93
```

```
class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setupUi(self)
        self.showMaximized()

        #Adding controls widget
        self.controls = Controls(self)
        self.dockControls.setWidget(self.controls)
        self.controls.startButton.setEnabled(False)

        #Adding picture area
        self.picture = Picture(self)
        self.mdiArea.addSubWindow(self.picture)
        self.picture.showMaximized()

        #Adding button to toggle control widget
        self.controls_button = self.dockControls.toggleViewAction()
        self.controls_button.setText("Show controls")
        self.menuView.addAction(self.controls_button)

        #To Close program from File -menu
        self.actionExit.triggered.connect(lambda: self.close())

        #Preparing for extra thread for video
        self.yolo_worker = None

        #Setting picture to picturearea
        self.videoHeight = 640
        self.videoWidth = 480
        self.img = QPixmap("./logo.jpg")
        self.img = self.img.scaled(self.videoHeight,self.videoWidth, Qt.KeepAspectRatio)
        self.picture.videoFrame.setPixmap(self.img)

        #Showing initial setup dialog
        self.setup = Setup(self)
        self.setup.buttonBox.accepted.connect(self.setOptions)
        self.setup.exec()

        self.controls.startButton.clicked.connect(self.start_button_clicked)
        self.controls.stopButton.clicked.connect(self.stop_button_clicked)
        self.controls.newButton.clicked.connect(lambda: self.setup.exec())

```

```
137     def setOptions(self):
138         self.model = self.setup.modelListBox.currentText()
139         self.controls.chosenModel.setText(self.model)
140         self.source = self.setup.sourceListBox.currentText()
141         self.controls.chosenSource.setText(self.source)
142         if self.setup.youtubeURL.isEnabled():
143             self.youtubeUrl = self.setup.youtubeURL.text()
144             self.controls.youtubeUrl.setText(self.youtubeUrl)
145             self.controls.youtubeLabel.setVisible(True)
146         else:
147             self.youtubeUrl = ""
148             self.controls.youtubeUrl.setText(self.youtubeUrl)
149             self.controls.youtubeLabel.setVisible(False)
150 
151         try:
152             self.picture.isVisible() == True
153         except:
154             self.picture = Picture(self)
155             self.mdiArea.addSubWindow(self.picture)
156             self.picture.showMaximized()
157             self.videoHeight = 640
158             self.videoWidth = 480
159             self.img = QPixmap("./logo.jpg")
160             self.img = self.img.scaled(self.videoHeight,self.videoWidth, Qt.KeepAspectRatio)
161             self.picture.videoFrame.setPixmap(self.img)
162 
163         self.YOLO_plotting = self.setup.radioButtonYOLOPlotting.isChecked()
164 
165         self.controls.startButton.setEnabled(True)
166 
167         self.picture.videoFrame.resizeEvent = self.video_resize
168 
169         self.confidenceTreshold = self.setup.confidenceSlider.value()
170 
171         self.videoHeight = self.picture.videoFrame.height()
172         self.videoWidth = self.picture.videoFrame.width()
173 
174 
175     def video_resize(self, resizeEvent:QResizeEvent):
176         self.videoWidth = self.picture.videoFrame.width()
177         self.videoHeight = self.picture.videoFrame.height()
178         if self.yolo_worker != None:
179             self.yolo_worker.change_video_size(self.videoWidth, self.videoHeight)
180 
```

```
181
182     @Slot(list)
183     def updateResults(self, result_list):
184
185         text = ""
186         if result_list[0] == "ERROR":
187             text = f"{result_list[0]}: {result_list[1]}"
188         else:
189             for x in result_list:
190                 class_name, confidence = x
191                 confidence = "{0:.0%}".format(confidence)
192                 text = text + f"{confidence} {class_name}" + "\n"
193
194         self.picture.resultsFrame.setText(text)
195
196
197
198     def start_button_clicked(self):
199         self.controls.startButton.setEnabled(False)
200         self.controls.stopButton.setEnabled(True)
201         self.controls.newButton.setEnabled(False)
202         try:
203             self.picture.isVisible() == True
204         except:
205             self.picture = Picture(self)
206             self.mdiArea.addSubWindow(self.picture)
207             self.picture.showMaximized()
208         self.yolo_worker = Worker(self.model, self.confidenceTreshold, self.source, self.youtubeUrl, self.videoWidth, self.videoHeight, self.YOLO_plotting)
209         self.yolo_worker.changePixmap.connect(self.picture.videoFrame.setPixmap)
210         self.yolo_worker.sendResults.connect(self.updateResults)
211         self.yolo_worker.start()
212
213     def stop_button_clicked(self):
214         self.controls.startButton.setEnabled(True)
215         self.controls.stopButton.setEnabled(False)
216         self.controls.newButton.setEnabled(True)
217         self.picture.resultsFrame.setText("Results")
218         self.yolo_worker.stop()
219         self.yolo_worker.quit()
220         self.yolo_worker.terminate()
221
222
223
224 app = QApplication(sys.argv)
225 app.setWindowIcon(QIcon('logo.ico'))
226
227 window = MainWindow()
228 window.show()
229 app.exec()
```

Liite 5. GUI ikkunoiden koodi

mainWindow.py > ...

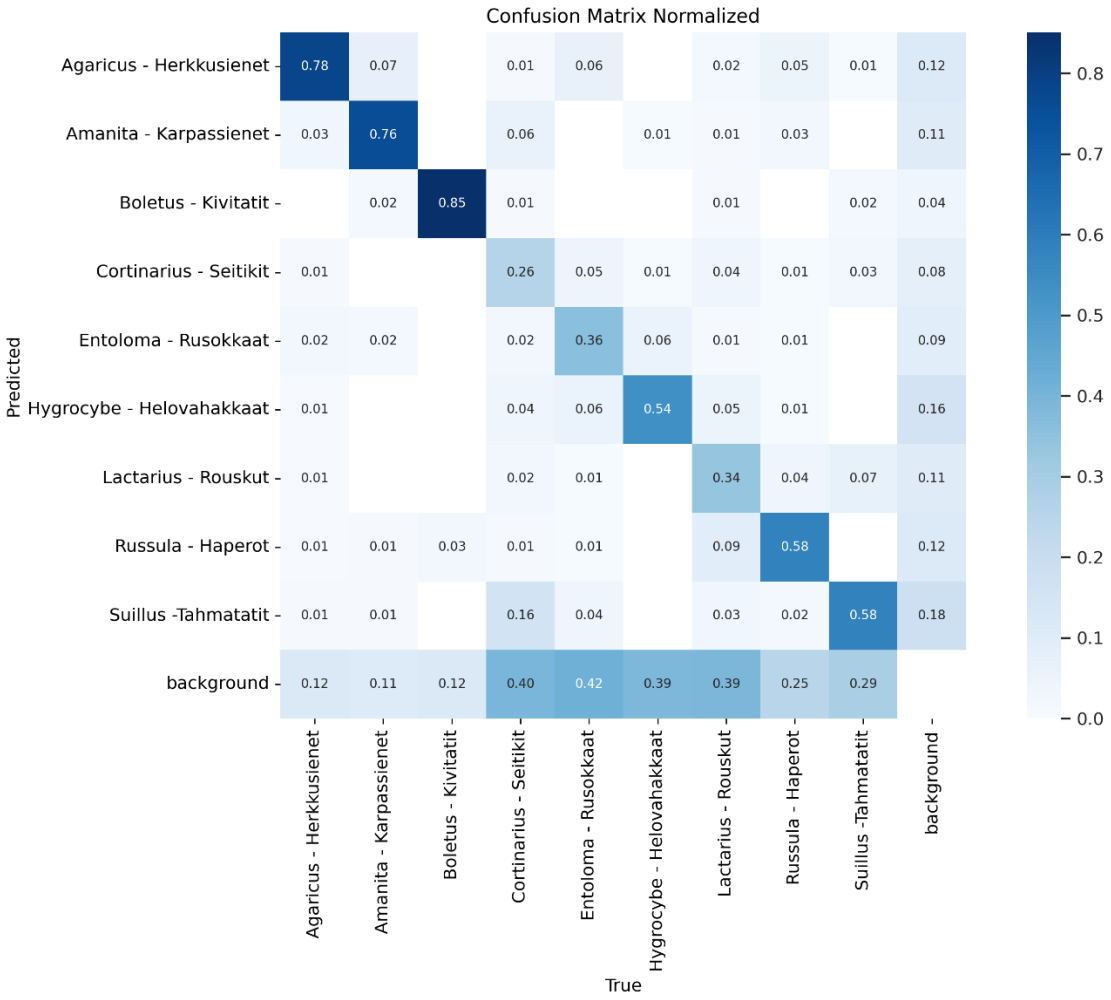
```python
# -*- coding: utf-8 -*-

################################################################################
## Form generated from reading UI file 'mainwindow.ui'
##
## Created by: Qt User Interface Compiler version 6.4.3
##
## WARNING! All changes made in this file will be lost when recompiling UI file!
################################################################################

from PySide6.QtCore import (QCoreApplication, QDate, QDateTime, QLocale,
    QMetaObject, QObject, QPoint, QRect,
    QSize, QTime, QUrl, Qt)
from PySide6.QtGui import (QAction, QBrush, QColor, QConicalGradient,
    QCursor, QFont, QFontDatabase, QGradient,
    QIcon, QImage, QKeySequence, QLinearGradient,
    QPainter, QPalette, QPixmap, QRadialGradient,
    QTransform)
from PySide6.QtWidgets import (QApplication, QDockWidget, QGridLayout, QMainWindow,
    QMdiArea, QMenu, QMenuBar, QSizePolicy,
    QStatusBar, QWidget)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        if not MainWindow.objectName():
            MainWindow.setObjectName(u"MainWindow")
        MainWindow.resize(963, 604)
        self.actionExit = QAction(MainWindow)
        self.actionExit.setObjectName(u"actionExit")
        self.actionShow_Controls = QAction(MainWindow)
        self.actionShow_Controls.setObjectName(u"actionShow_Controls")
        self.centralwidget = QWidget(MainWindow)
        self.centralwidget.setObjectName(u"centralwidget")
        self.gridLayout = QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName(u"gridLayout")
        self.mdiArea = QMdiArea(self.centralwidget)
        self.mdiArea.setObjectName(u"mdiArea")
        sizePolicy = QSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.mdiArea.sizePolicy().hasHeightForWidth())
        self.mdiArea.setSizePolicy(sizePolicy)
        brush = QBrush(QColor(89, 110, 160, 255))
        brush.setStyle(Qt.SolidPattern)
        self.mdiArea.setBackground(brush)

        self.gridLayout.addWidget(self.mdiArea, 0, 0, 1, 1)

```

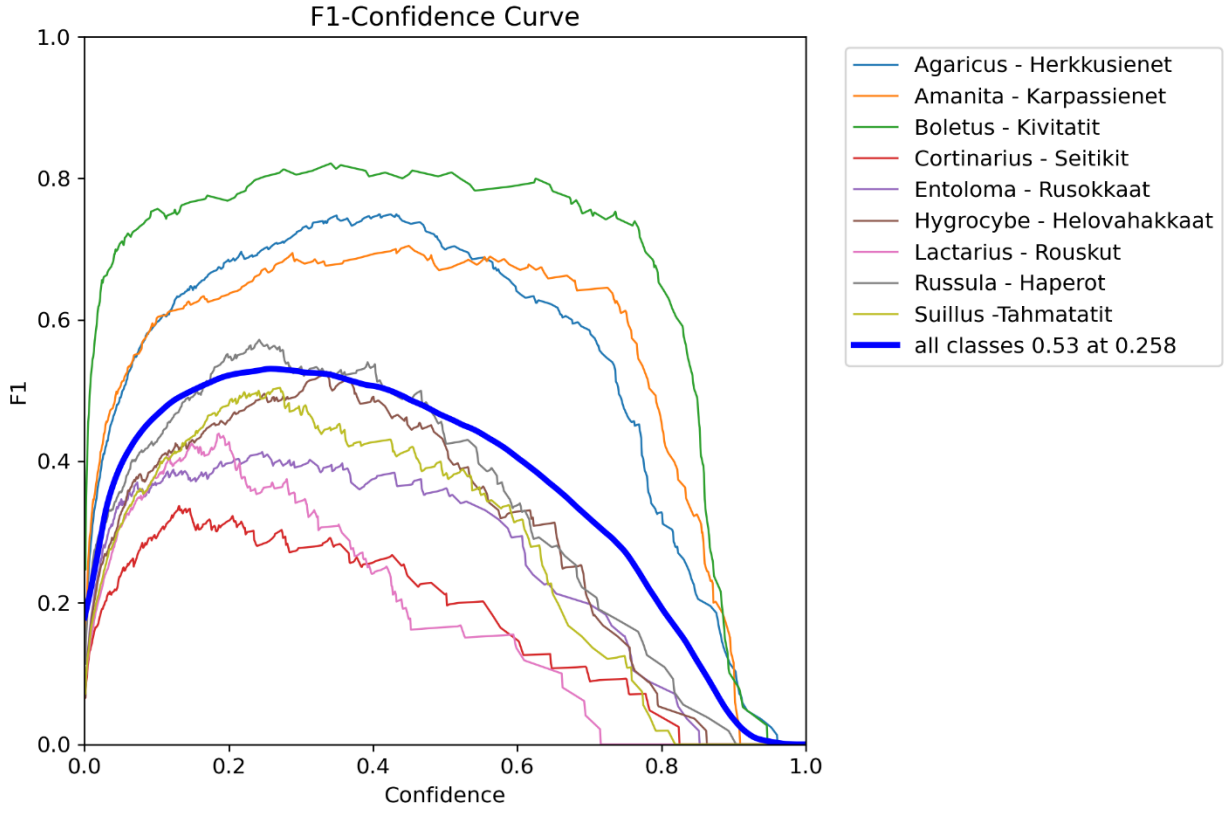
```
49          MainWindow.setCentralWidget(self.centralwidget)
50          self.menubar = QMenuBar(MainWindow)
51          self.menubar.setObjectName(u"menubar")
52          self.menubar.setGeometry(QRect(0, 0, 963, 22))
53          self.menuFile = QMenu(self.menubar)
54          self.menuFile.setObjectName(u"menuFile")
55          self.menuView = QMenu(self.menubar)
56          self.menuView.setObjectName(u"menuView")
57          MainWindow.setMenuBar(self.menubar)
58          self.statusbar = QStatusBar(MainWindow)
59          self.statusbar.setObjectName(u"statusbar")
60          MainWindow.setStatusBar(self.statusbar)
61          self.dockControls = QDockWidget(MainWindow)
62          self.dockControls.setObjectName(u"dockControls")
63          sizePolicy1 = QSizePolicy(QSizePolicy.MinimumExpanding, QSizePolicy.Preferred)
64          sizePolicy1.setHorizontalStretch(0)
65          sizePolicy1.setVerticalStretch(0)
66          sizePolicy1.setHeightForWidth(self.dockControls.sizePolicy().hasHeightForWidth())
67          self.dockControls.setSizePolicy(sizePolicy1)
68          self.dockControls.setAutoFillBackground(True)
69          self.dockWidgetContents_3 = QWidget()
70          self.dockWidgetContents_3.setObjectName(u"dockWidgetContents_3")
71          self.dockControls.setWidget(self.dockWidgetContents_3)
72          MainWindow.addDockWidget(Qt.LeftDockWidgetArea, self.dockControls)
73
74          self.menubar.addAction(self.menuFile.menuAction())
75          self.menubar.addAction(self.menuView.menuAction())
76          self.menuFile.addAction(self.actionExit)
77
78          self.retranslateUi(MainWindow)
79
80          QMetaObject.connectSlotsByName(MainWindow)
81      # setupUi
82
83      def retranslateUi(self, MainWindow):
84          MainWindow.setWindowTitle(QCoreApplication.translate("MainWindow", u"MainWindow", None))
85          self.actionExit.setText(QCoreApplication.translate("MainWindow", u"Exit", None))
86          self.actionShow_Controls.setText(QCoreApplication.translate("MainWindow", u"Show Controls", None))
87          self.menuFile.setTitle(QCoreApplication.translate("MainWindow", u"File", None))
88          self.menuView.setTitle(QCoreApplication.translate("MainWindow", u"View", None))
89      # retranslateUi
90
91
```

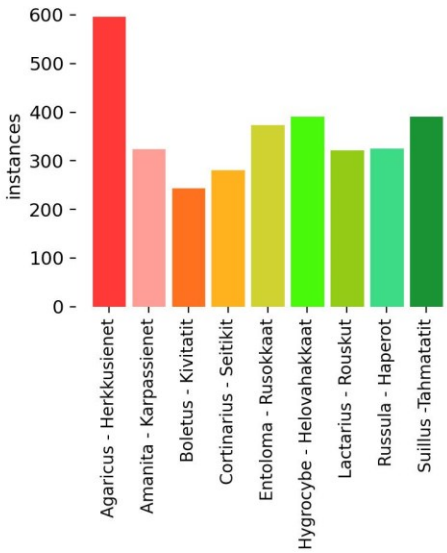
Liite 6. main.spec tiedosto

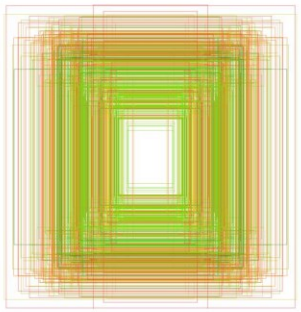
```
# -*- mode: python ; coding: utf-8 -*-


block_cipher = None
import sys ; sys.setrecursionlimit(sys.getrecursionlimit() * 5)

a = Analysis(
    ['main.py'],
    pathex=[],
    binaries=[],
    datas=[("test50.mp4", "."), ("logo.ico", "."), ("logo.jpg", "."), ("best.pt", "."), ("yolov8s.pt", "."), ("yolov8n.pt", "."), ("default.yaml", "./ultralytics/yolo/cfg")],
    hiddenimports=[],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=block_cipher,
    noarchive=False,
)
pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)

exe = EXE(
    pyz,
    a.scripts,
    [],
    exclude_binaries=True,
    name='YOLO Demo',
    debug=False,
    bootloader_ignore_signals=False,
    strip=False,
    upx=True,
    console=False,
    disable_windowed_traceback=False,
    argv_emulation=False,
    target_arch=None,
    codesign_identity=None,
    entitlements_file=None,
    icon='logo.jpg',
)
coll = COLLECT(
    exe,
    a.binaries,
    a.zipfiles,
    a.datas,
    strip=False,
    upx=True,
    upx_exclude=[],
    name='main',
)

```

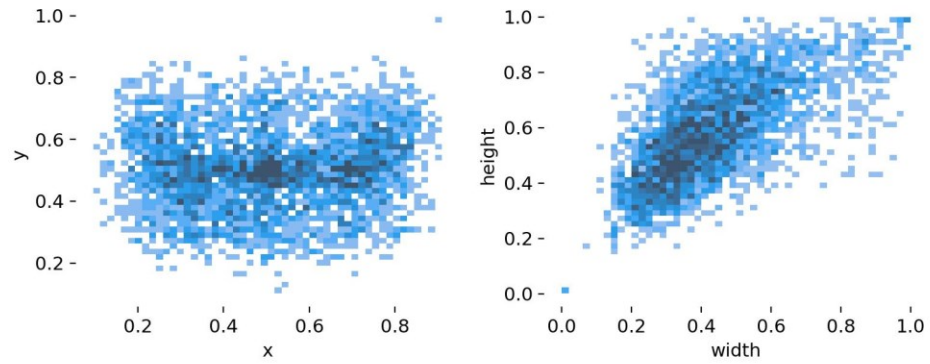
Liite 7. YOLO-mallin koulutuksen tulokset

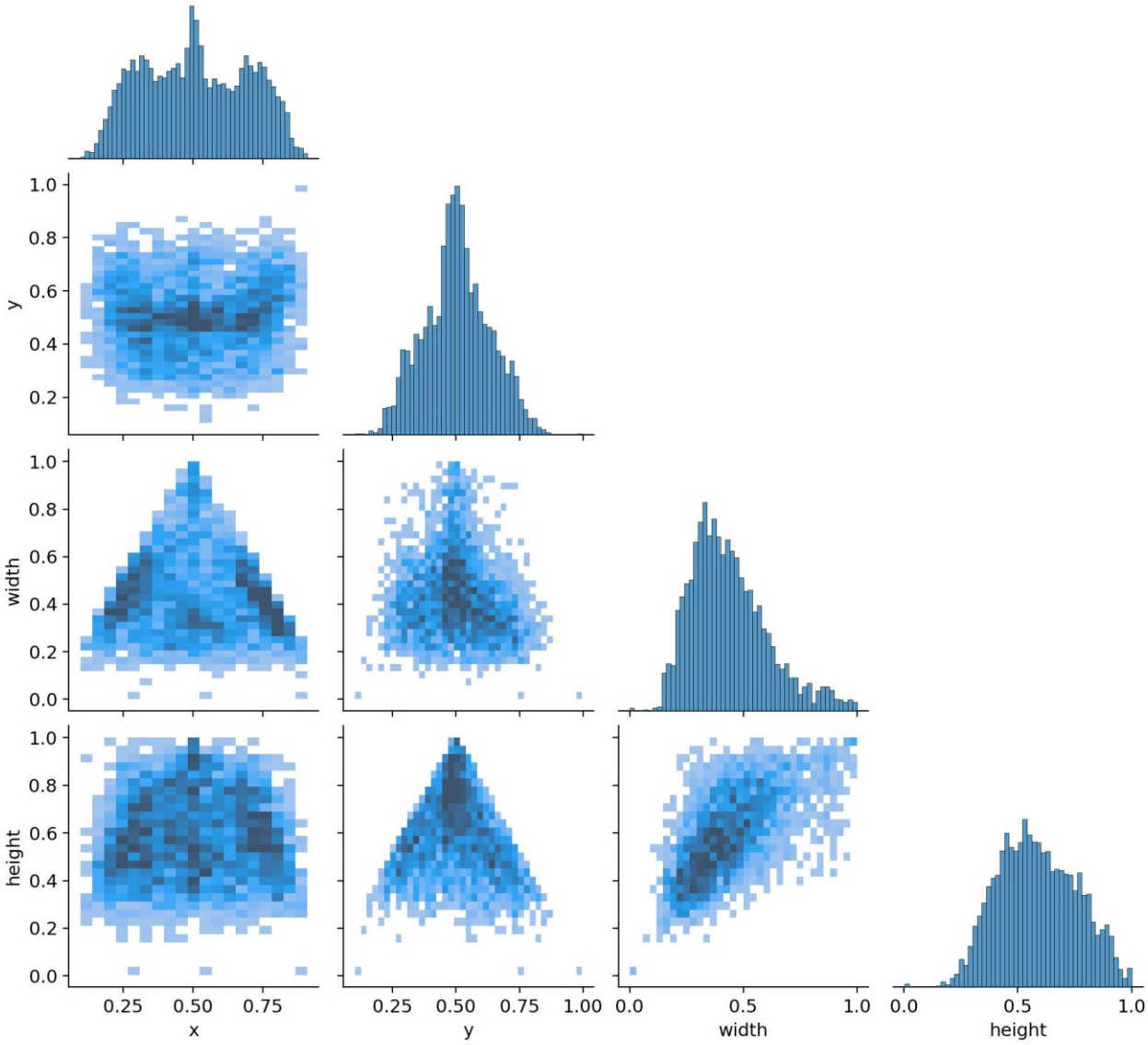


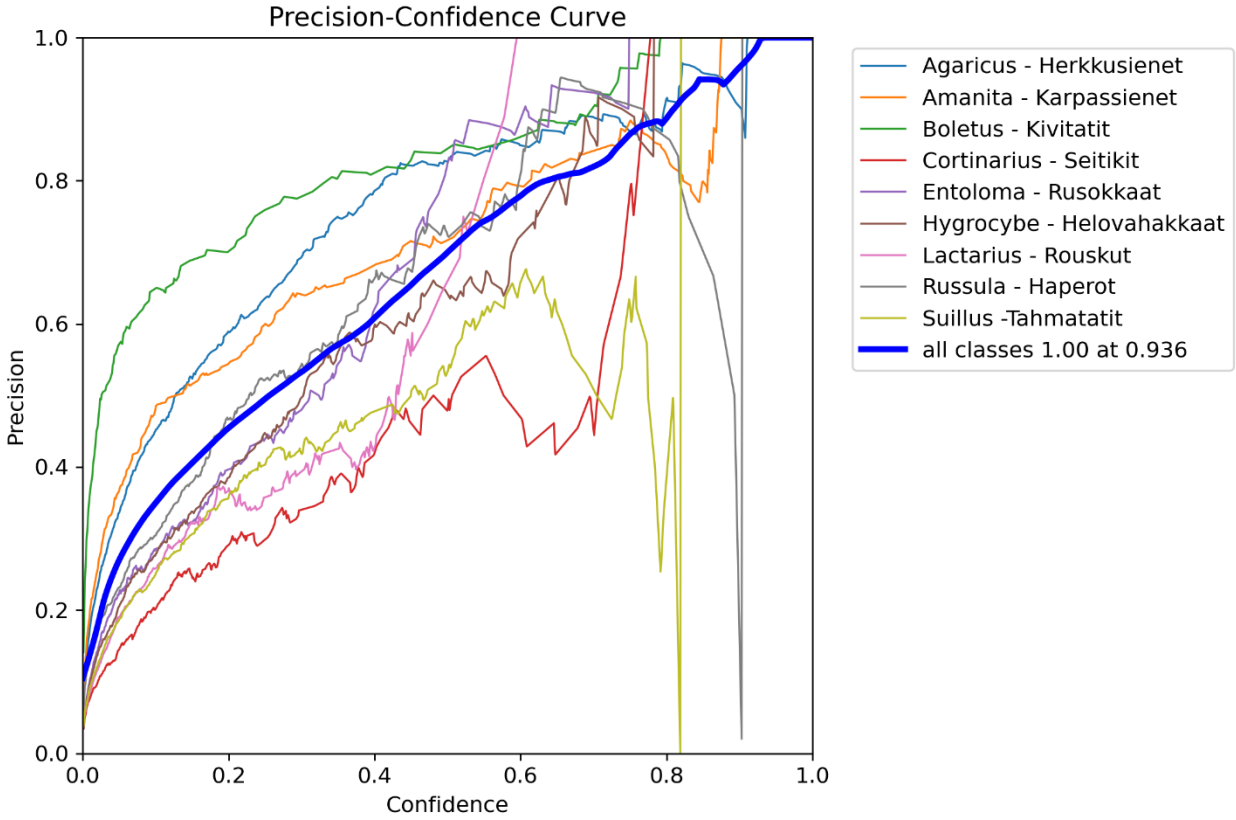

F1-Confidence Curve
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Agaricus - Herkkusienet
Amanita - Karpassienet
Boletus - Kivitatit
Cortinarius - Seitikit
Entoloma - Rusokkaat
Hygrocybe - Helovahakkaat
Lactarius - Rouskut
Russula - Haperot
Suillus -Tahmatatit
all classes 0.53 at 0.258

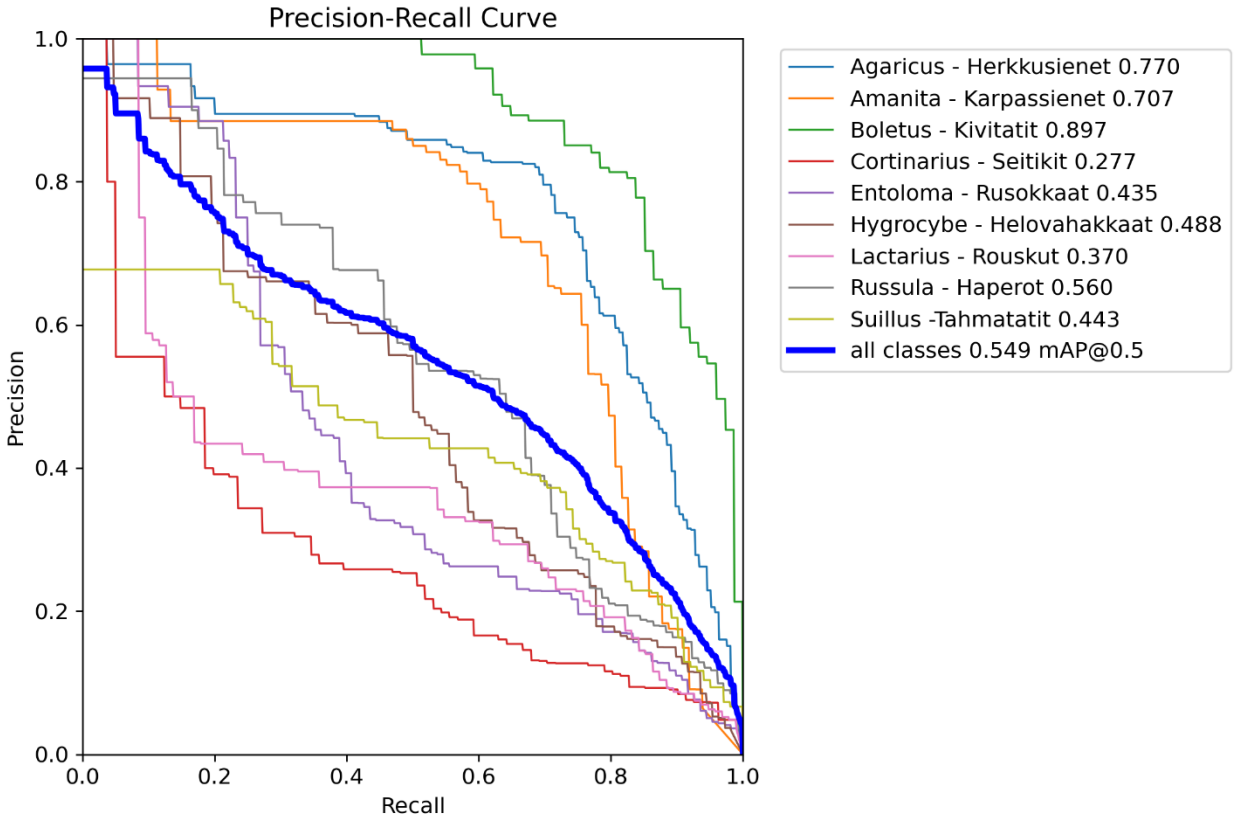

instances
0
100
200
300
400
500
600
Agaricus - Herkkusienet
Amanita - Karpassienet
Boletus - Kivitatit
Cortinarius - Seitikit
Entoloma - Rusokkaat
Hygrocybe - Helovahakkaat
Lactarius - Rouskut
Russula - Haperot
Suillus -Tahmatatit

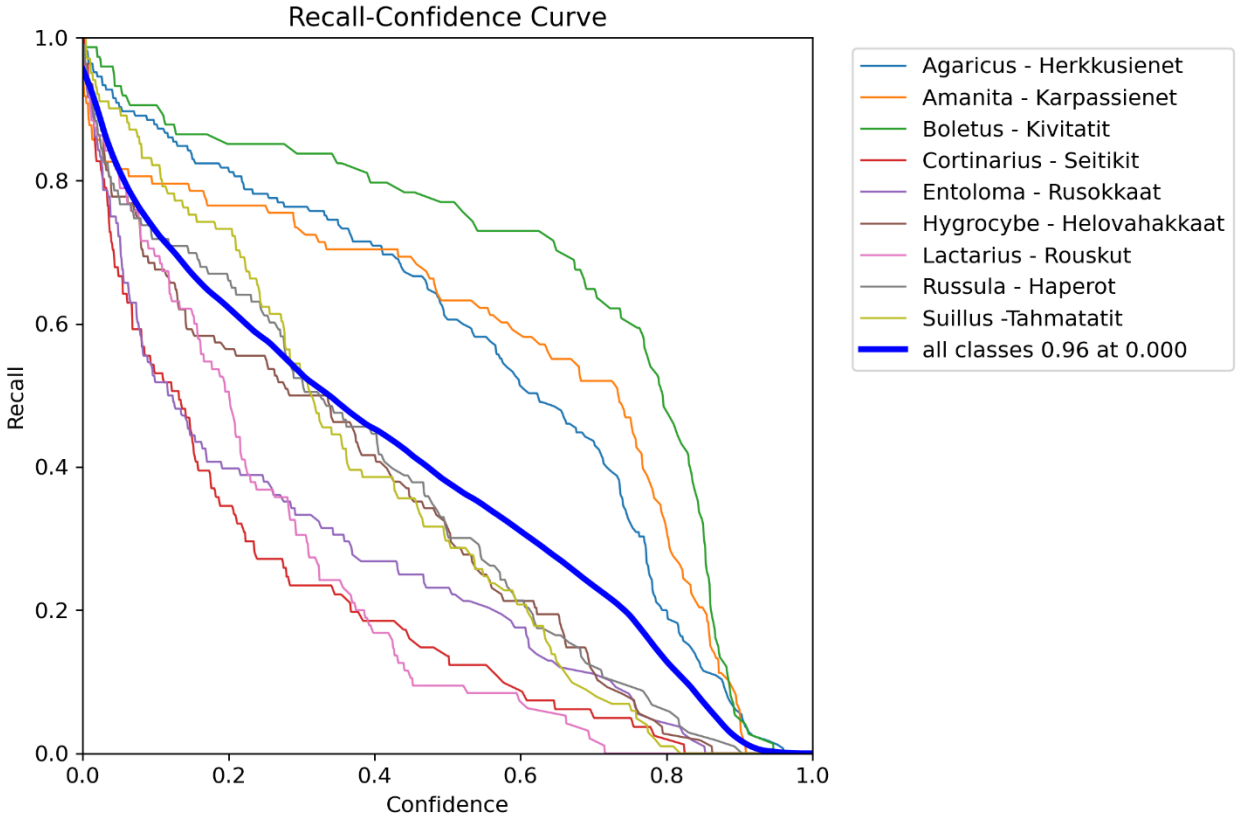


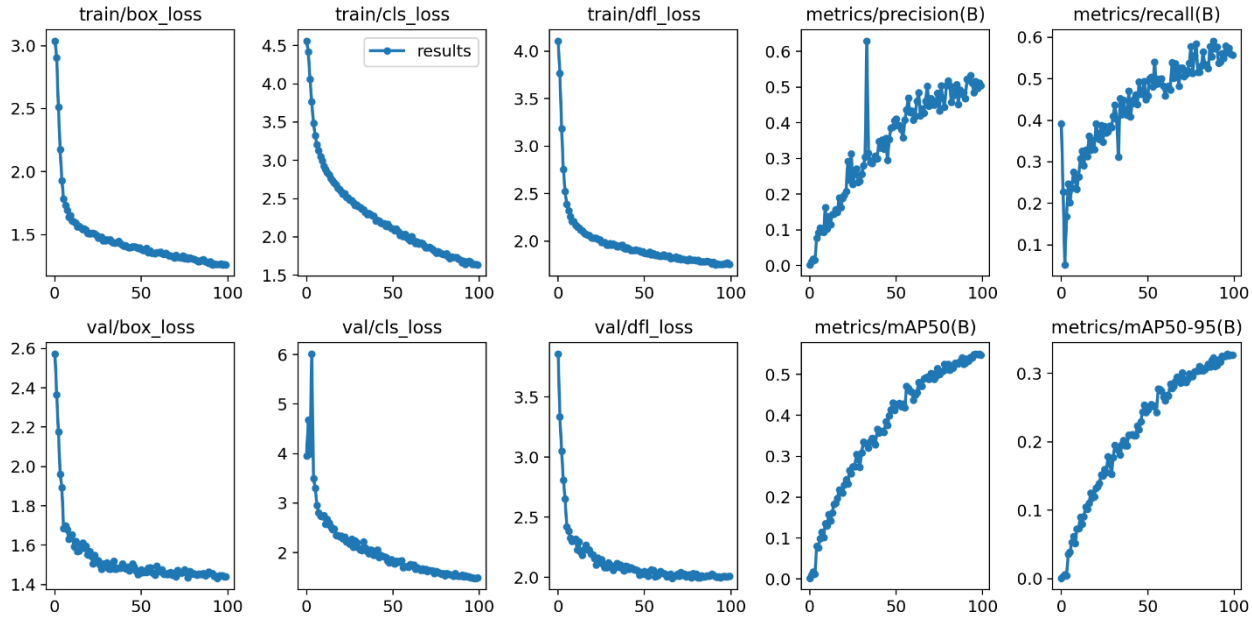

y
x
height
width
0.0
0.2
0.4
0.6
0.8
1.0


y
width
height
x
y
width
height


Precision-Confidence Curve
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Agaricus - Herkkusienet
Amanita - Karpassienet
Boletus - Kivitatit
Cortinarius - Seitikit
Entoloma - Rusokkaat
Hygrocybe - Helovahakkaat
Lactarius - Rouskut
Russula - Haperot
Suillus -Tahmatatit
all classes 1.00 at 0.936


Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Agaricus - Herkkusienet 0.770
Amanita - Karpassienet 0.707
Boletus - Kivitatit 0.897
Cortinarius - Seitikit 0.277
Entoloma - Rusokkaat 0.435
Hygrocybe - Helovahakkaat 0.488
Lactarius - Rouskut 0.370
Russula - Haperot 0.560
Suillus -Tahmatatit 0.443
all classes 0.549 mAP@0.5


Recall-Confidence Curve
Recall
Confidence
Agaricus - Herkkusienet
Amanita - Karpassienet
Boletus - Kivitatit
Cortinarius - Seitikit
Entoloma - Rusokkaat
Hygrocybe - Helovahakkaat
Lactarius - Rouskut
Russula - Haperot
Suillus -Tahmatatit
all classes 0.96 at 0.000


train/box_loss
train/cls_loss
results
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)