LAB University of Applied Sciences

# Siirtolohkareiden automaattinen tunnistaminen ja luokittelu konvoluutioneuroverkkoja hyödyntäen

LAB-ammattikorkeakoulu
Insinööri (ylempi AMK), IoT:stä tekoälyyn
2024
Marko Rastas

**Tiivistelmä**

| Tekijä(t) | Julkaisun laji | Valmistumisaika |
|---|---|---|
| Marko Rastas | Opinnäytetyö, YAMK | 2024 |
| | Sivumäärä<br>59 | |

Työn nimi

**Siirtolohkareiden automaattinen tunnistaminen ja luokittelu konvoluutioneuroverkkoja hyödyntäen**

Tutkinto ja koulutusala

Insinööri (ylempi AMK), IoT:stä tekoälyyn

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)


Tiivistelmä

Opinnäytetyössä tutkittiin siirtolohkareiden automaattista tunnistamista ja luokittelua konvoluutioneuroverkkoihin perustuvan syväoppimismallin avulla. Kehitettiin prosessi, jonka avulla mahdollistettiin siirtolohkareiden automaattinen tunnistaminen ilmakuvasta. Tällä voidaan lisätä mahdollisuutta liikkumiseen ja urheiluun ulkona mitkä tukevat myös työssä jaksamista.

Työssä käytettiin YOLOv8 syväoppimismallia, jonka avulla tunnistettiin siirtolohkareet ilmakuvista. Syväoppimismallia opetettiin ohjatusti manuaalisesti valikoidun ja annotoidun ilmakuva-aineiston perusteella. Menetelmässä käytettiin Google Earth -palvelua ilmakuva-aineiston hankkimiseen. Syväoppimismalleista YOLOv8 valittiin sen tehokkuuden ja tarkkuuden vuoksi.

Ohjatun oppimisen menetelmillä syväoppimismalli saatiin tunnistamaan ja luokittelemaan siirtolohkareet oikein 66-73% todennäköisyydellä. Tulokset osoittavat, että konvoluutioneuroverkkojen avulla voidaan automaattisesti tunnistaa ja luokitella luonnossa esiintyviä geologisia kohteita. Työssä kehitettyä prosessia ja ratkaisua voidaan laajentaa muihin geologisiin tutkimuksiin.



Asiasanat

koneoppiminen, konvoluutioneuroverkko, YOLO, siirtolohkare

**Abstract**


| Author(s) | Type of Publication | Published |
|---|---|---|
| Marko Rastas | Master's Thesis | 2024 |
| | Number of Pages<br>59 | |


Title of Publication

**Automatic identification and classification of boulders using convolutional neural networks**

Possible subtitle(s)

Degree, Field of Study

Master of Engineering, From IoT to AI

Organisation of the client (if the thesis work is commissioned by another party)


Abstract

The thesis investigated the automatic identification and classification of boulders using a deep learning model using convolutional neural networks. A process was developed that enabled the automatic identification of boulders from an aerial image. This can increase the opportunity for exercise and sports outdoors which also support coping at work.

The work used the YOLOv8 deep learning model, which was used to identify the boulders from aerial images. The deep learning model was taught using supervised learning based on manually selected and annotated aerial image data. The method used the Google Earth service to acquire aerial image data. Among the deep learning models, YOLOv8 was chosen due to its efficiency and accuracy.

With supervised learning methods, the deep learning model was able to identify and classify the boulders correctly with a probability of 66-73%. The results indicate that convolutional neural networks can be utilized to automatically identify and classify naturally occurring geological objects. The process and solution can be extended to other geological studies.



Keywords

machine learning, convolutional neural network, YOLO, boulder

Sisällys

# 1 Johdanto

## 1.1 Työn tausta ja tavoitteet

Tämän tutkimuksen kirjoittaja on aiemmin tutkinut kalliokiipeilyalueiden tunnistamista korkeus- ja kaltevuusluokittelun avulla käyttäen maanmittauslaitoksen avoimen aineiston laserkeilausdataa. Aiempi tutkimus sisälsi laserkeilausdatan analysointia paikkatieto-ohjelmistoilla ja -algoritmeilla sekä analyysien tuloksista johdettujen polygonien käsittelyä ja tiedon yhdistämistä sekä tiedon visualisointia mobiililaitteessa ja verkkoselaimessa. Aiemman tutkimuksen aikana havaittiin, että laserkeilausdatasta ei voi tunnistaa ja luokitella siirtolohkareita koska ne eivät ole osana KM2-korkeusmallia (Oksanen ym. 2016, 9). Tällöin heräsi mielenkiinto, voisiko siirtolohkareita tunnistaa ja luokitella automaattisesti ilmakuvista.

Tekoälyn ja koneoppimisen kehitys etenee tällä hetkellä nopeasti ja nykyteknologia mahdollistaa monimutkaisten algoritmien suorittamisen laajalla tietojoukolla. Tässä opinnäytetyössä keskitytään erityisesti syväoppimiseen, kuvan tunnistamiseen ja luokitteluun. Näiden avulla pyritään löytämään vastauksia edellä mainittuun kysymykseen.

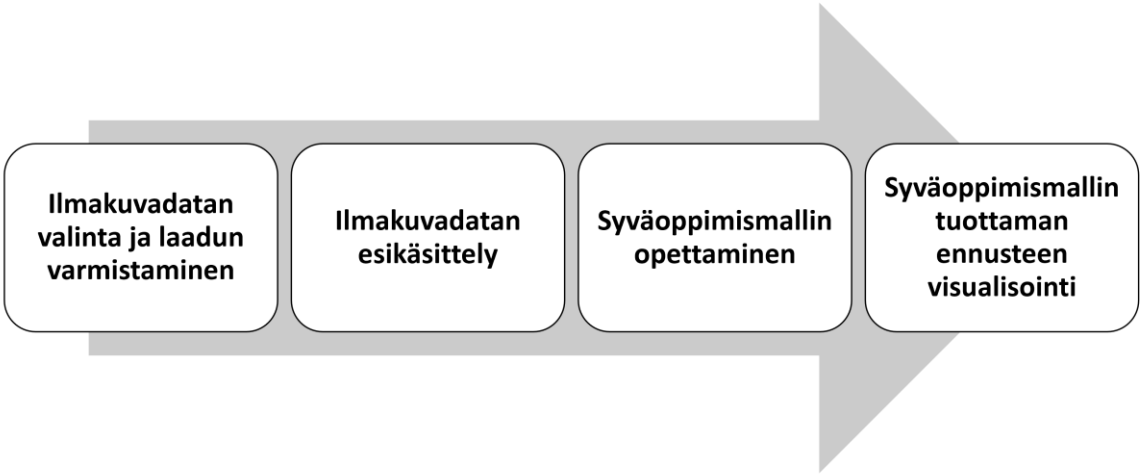
Tämän opinnäytetyön ensisijaisena tavoitteena on tutkia ja kehittää prosessi, jonka avulla mahdollistetaan siirtolohkareiden automaattinen tunnistaminen ilmakuvasta. Tutkimuksen tuloksista voivat hyötyä esimerkiksi Suomen kiipeilijäyhteisöt. Heille voi tässä työssä esitetyn prosessin kautta löytyä uusia siirtolohkareita, joilla voi boulderoida. Tällä tavoin voidaan lisätä mahdollisuuksia ulkona liikkumiseen ja urheiluun, mikä puolestaan parantaa fyysistä ja henkistä hyvinvointia sekä tukee työssä jaksamista.

Boulderointi on kiipeilyä siirtolohkareilla ja matalilla kallionseinämillä ilman köysivarmistusta. Boulderointi on suosittu harrastus Suomessa, ja ulkokiipeilymahdollisuuksia löytyy luonnosta erittäin paljon. Suojavarusteita käytetään, kuten putoamisalustoja, jotta ulkokiipeily olisi turvallisempaa. (Kesäläinen & Kejonen 2019, 93.)

Tämän opinnäytetyön toisena tavoitteena on kehittää tekninen ratkaisu. Se sisältää syväoppimismallin, joka tunnistaa ja luokittelee siirtolohkareet ilmakuvasta ja myös työpöytäsovelluksen, jolla visualisoidaan syväoppimismallin tuottama data. Ratkaisuun kuuluu kriittisenä osana myös ilmakuvien hankkiminen, niiden esikäsittely ja luokittelu, jotta syväoppimismalli saadaan toimimaan.

## 1.2 Tutkimuskysymykset ja rajaus

Tässä opinnäytetyössä kehitettävän prosessin päävaiheet on esitetty kuvassa 1. Prosessi alkaa datan valinnasta ja laadun varmistamisesta. Seuraavissa vaiheissa data esikäsitellään ja syväoppimismalli opetetaan. Lopulta syväoppimismallin perusteella tehdyt ennusteet visualisoidaan.




Kuva 1. Syväoppimista hyödyntävä ilmakuvadatan analyysiprosessi

Tutkimuksessa keskitytään prosessiin liittyviin tutkimuskysymyksiin:

- Kuinka ilmakuvat tulisi valita, jotta syväoppimismalli tunnistaisi siirtolohkareita luotettavasti?
- Miten siirtolohkareen rajaaminen ilmakuvien esikäsittelyvaiheessa vaikuttaa syväoppimismallin tuottamaan ennusteeseen?
- Mitkä tekijät vaikuttavat syväoppimismallin kykyyn erotella eri kokoisia siirtolohkareita ilmakuvissa?
- Miten syväoppimismallia voidaan hienosäätää parempien tulosten saavuttamiseksi?
- Mitä rajoitteita syväoppimismallin visualisoinnissa työpöytäsovelluksessa havaitaan?

Tutkimusta rajattiin niin että vertaillaan eri koneoppimisalustoja, syväoppimismalleja, ilmakuvien karttapalveluja ja käyttöliittymäkirjastoja, mutta ratkaisua varten valitaan vain yksi jokaisesta. Syväoppimismallin opettamiseen hankittava ilmakuvadatan keruu rajoitetaan Suomen alueelle. Aineisto koostuu vain Suomen siirtolohkareista otetuista ilmakuvista.

Työpöytäsovellus toteutetaan vain Windows-alustalle. Syväoppimismallin kehityksessä käytetään vain avoimen lähdekoodin lisenssejä. Google Earth -palvelun ilmakuvia saa käyttää tieteellisessä tutkimuksessa ilman erillistä lupaa, mutta kuvien esittäminen opinnäytetyössä vaativat kuvan alle attribuutiotiedon.

## 1.3 Tutkimusmenetelmät

Työ noudattaa konstruktiivisen tutkimusotteen seitsemää vaihetta, joilla pyritään ratkaisemaan reaalimaailman ongelmaa ja tuottamaan lisäarvoa siihen liittyviin tieteenaloihin (Lukka, 2001). Tämän työn keskeiset tieteenalat ovat geoinformatiikka ja syväoppiminen. Ensimmäinen ja toinen vaihe, käytännön ongelman tunnistaminen, ja mahdollisuudet pitkän aikavälin tutkimusyhteistyöhön, ilmenevät johdannossa. Siinä esitellään työn tausta ja tavoitteet siirtolohkareiden automaattisesta tunnistamisesta ja luokittelusta ilmakuvista, sekä potentiaaliset mahdollisuudet ja sovellukset esimerkiksi kiipeilijäyhteisöille.

Kolmannessa vaiheessa tutkimusaiheen syvällinen ymmärrys saavutetaan tutustumalla syväoppimiseen ensin teoreettisesti ja sen jälkeen käytännöllisesti valitsemalla syväoppimismalli sekä ratkaisun toteuttamiseen tarvittavat työkalut. Neljäs ja viides vaihe, ratkaisun innovointi, rakentaminen ja testaaminen, toteutuu ilmakuvadatan valinnan kautta ja syväoppimismallin opettamisessa sen avulla sekä keskeisten mittarien avulla arvioiden mallin suoriutumista siirtolohkareiden tunnistamisessa. Kuudennessa vaiheessa käytännön toimivuutta arvioidaan työpöytäsovelluksen avulla, joka visualisoi syväoppimismallin tuottamat tulokset. Seitsemäs vaihe, ratkaisun tuoman lisäarvon arviointi, ilmenee yhteenvedossa ja pohdinnassa, joissa tarkastellaan tutkimuskysymyksiä, mallin suorituskykyä ja jatkokehittämismahdollisuuksia.

Koneoppimisalustoja, syväoppimismalleja, ilmakuvien karttapalveluja ja käyttöliittymäkirjastoja vertaillaan kvantitatiivisesti. Tällä pyritään selvittämään, mitkä teknologiat soveltuvat parhaiten opinnäytetyön ratkaisuosuuteen. Vertailuissa keskitytään toiminnallisiin ja määrällisiin eroihin, joihin perustuen valinnat tehdään. Näitä eroja ovat esimerkiksi mahdollisuus hankkia dataa eri ajanjaksoilta, yhteensopivuus eri ympäristöissä, skaalautuvuus ja mahdolliset käyttökustannukset.

Ilmakuvien valinnassa ja luokittelussa keskitytään laadullisiin ominaisuuksiin. Näitä ovat esimerkiksi kuvien selkeys, yksityiskohtaisuus ja ääriviivojen piirteet, joita ei voida mitata numeerisesti mutta jotka vaikuttavat syväoppimismallin toimivuuteen. Siirtolohkareen koon luokittelussa käytetään kirjallisuuden ja karttapalveluiden tarjoamien tietojen lisäksi tämän

työn kirjoittajan kokemusperäistä arviointia siitä, miten siirtolohkareet näkyvät ja erottuvat ilmakuvissa.

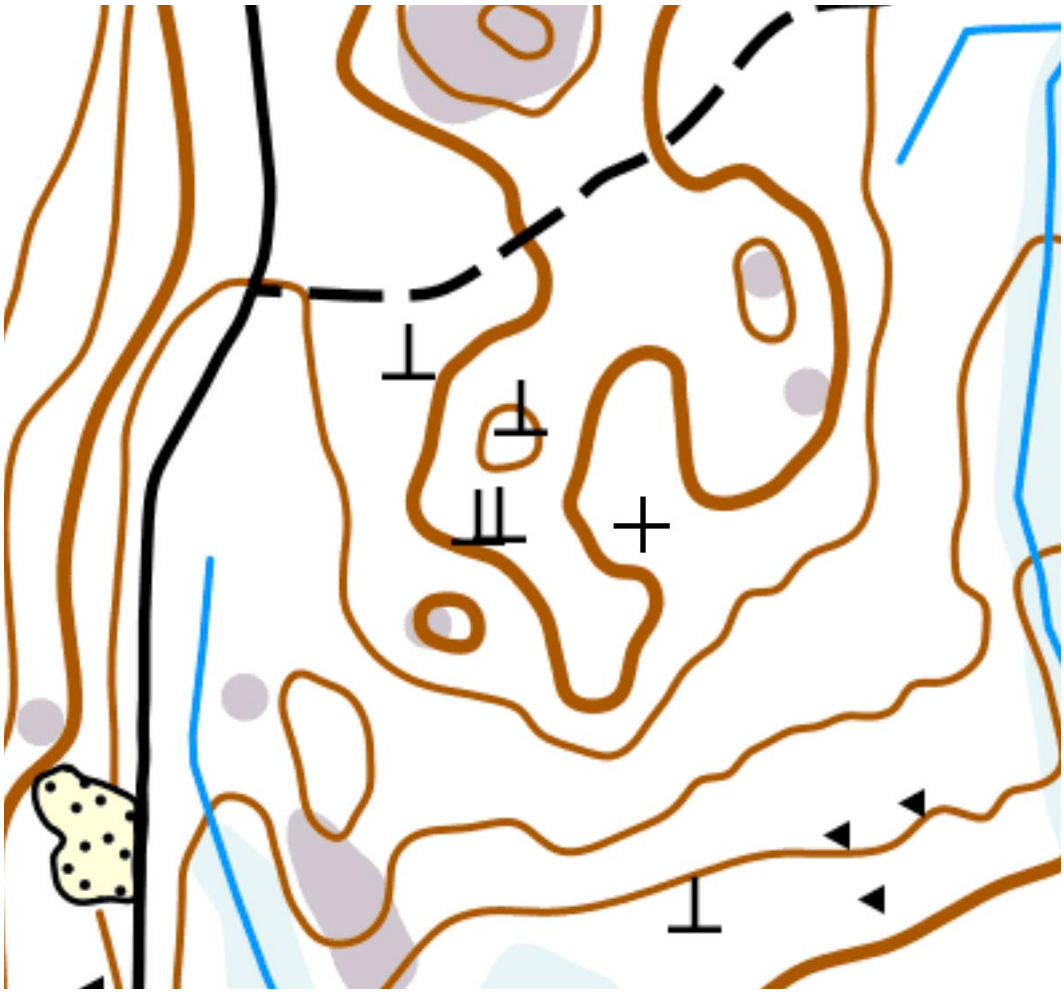
# 2 Siirtolohkare

## 2.1 Siirtolohkareet kartalla ja maastossa

Siirtolohkareet ovat poikkeuksellisen suuria kiven lohkareita, jotka ovat syntyneet eri tavoin jääkauden aikana. Jotkut siirtolohkareista ovat kulkeutuneet nykyisille sijainneilleen jäävuorten tai jäätiköiden mukana. Siirtolohkareet voivat olla kokonaan tai osittain maahan hautautuneita. (Kesäläinen & Kejonen 2019, 53.)

Tässä työssä analysoidaan erityisesti siirtolohkareita, joista osa on merkitty myös maanmittauslaitoksen karttoihin kivimerkillä eli väärinpäin olevan T-merkin avulla. Kaikkia luonnossa esiintyviä siirtolohkareita ei ole merkitty karttoihin. Merkityt siirtolohkareet ovat yli 2,5 metriä korkeita, erottuvat hyvin ympäröivästä maastosta tai ne tunnetaan tietyllä nimellä (Niemelä 2004, 71).

Kuvassa 2 esitetään millä tavoin nämä siirtolohkareet näkyvät maanmittauslaitoksen kartalla. Osa näistä voi sopia hyvin kiipeilyyn, jos ne ovat riittävän suuria. Kartta ei kuitenkaan tässä muodossa kerro vielä riittävästi kiven ominaisuuksista, esimerkiksi todellisesta korkeudesta.



Kuva 2. Karttakuva, jonka keskellä näkyy kivimerkkejä (Maanmittauslaitos 2024)

Kuvassa 3 esitetään erityisesti boulderointiin hyvin sopiva siirtolohkare. Se on riittävän korkea, noin 4-5 metriä, mutta ei liian korkea, jotta kiipeäminen sen päälle on vielä turvallista. Suojavarusteina käytetään esimerkiksi putoamisalustaa, joka vaimentaa putoamisesta syntyvää iskuvoimaa. (Korosuo 2017, 199.)

Kuva 3. Boulderointiin sopiva siirtolohkare (Kuva: Marko Rastas)

Siirtolohkareiden löytäminen ilmakuvista ihmissilmin on haastavaa, koska ne ovat yleensä sammaleen, lehtien ja oksien peitossa, kuten kuvassa 4 esitetään. Sopivia kiviä etsitään paljon myös suoraan luontoon lähtemällä. Suunnistusseurojen tuottamat suunnistuskartat

tuovat lisätietoa alueen kivistä. Kun sopiva kivi löytyy, se puhdistetaan kevyesti sekä alastulo siivotaan ja tasoitetaan, ennen kuin kivellä voidaan kiivetä. (Lindroos 2023, 8-10.)

Kuva 4. Siirtolohkareita keskellä metsää (Kuva: Marko Rastas)

## 2.2 Siirtolohkareen koon luokittelu

Kun syväoppimismallia opetetaan ohjatulla tavalla, sille kerrotaan mihin luokkaan mikäkin lähtödata kuuluu, jotta malli oppii tunnistamaan erilaisia luokkia (Kananen & Puolitaival 2019, 49). Siirtolohkareiden koon luokitteluun ei ole olemassa yhtä oikeaa tapaa. Tässä työssä käytetään seuraavaa 3-portaista siirtolohkareiden koon luokittelua, jotta voidaan testata, oppiiko syväoppimismalli tunnistamaan eri kokoisia siirtolohkareita:

- normaali (2,5 - 5 metriä korkea)
- suuri (5 - 10 metriä korkea)
- todella suuri (yli 10 metriä korkea).

Siirtolohkareiden koon luokittelussa hyödynnetään kirjoista löytyvää tietoa, internet-palveluja kuten 27 Crags, joissa on kiipeilijöiden tuottamaa tietoa, sekä tämän työn kirjoittajan kokemuksesta saatua tietoa. Siirtolohkareen korkeuteen liittyy kuitenkin aina virhemarginaali, koska niitä ei ole pääsääntöisesti mitattu tieteellisesti, vaan arvioitu

silmämääräisesti. Tästä syystä eri luokkien rajalla olevat siirtolohkareet voivat aiheuttaa syväoppimismallissa luokitteluun liittyviä haasteita. Esimerkiksi tasan 5 metriä korkea siirtolohkare voidaan tulkita joko normaali- tai suuri-luokan ilmentymäksi.

# 3 Ilmakuvien karttapalvelut

## 3.1 Google Earth

Google Earth -palvelun kautta voidaan selailla maailmaa ilmasta käsin sekä suoraan ylhäältä että osittain myös 3D-perspektiivistä. Palvelu on ilmainen ja sitä voidaan käyttää verkkoselaimella. Palvelusta on myös ilmainen Google Earth Pro -versio, joka toimii käyttäjän tietokoneella. (Google 2024.)

Google Earthin avulla voidaan selailla historiallisia ilmakuvia ajan suhteen, eli miltä alue on näyttänyt tiettyinä ajankohtina. Ilmakuvia voi tallentaa Google Earth Pron kautta suoraan omalle koneelle useassa eri kuvakoossa. Lisäksi karttaa voidaan kääntää eri ilmansuuntiin. (Google 2024.)

Google kerää ilmakuvat useilta eri toimittajilta esimerkiksi Maxar ja Airbus Defence & Space. Kuvat ovat yhdistelmä satelliittikuvia sekä ilmakuvia. Satelliittikuvien pikselitarkkuus voi olla jopa 0,3 metriä, ja ilmakuvien jopa 0,15 metriä. Kuvien päivitysaikataulua ei ole erikseen määritelty, koska se riippuu kuvien toimittajien omista aikatauluista. Google ilmoittaa kuvien keskimääräiseksi iäksi 1-3 vuotta. (Google 2024.)

## 3.2 Karttapaikka ja Paikkatietoikkuna

Maanmittauslaitoksen tarjoamassa Karttapaikka-palvelussa voi selata maastokarttoja, ilmakuvia ja muita rajapintoja esimerkiksi kiinteistörajoja. Historiallisia ilmakuvia ei voi selata, eikä karttaa voi kääntää eri ilmansuuntiin. Karttanäkymiä voi tulostaa ja tallentaa omalle koneelle. (Maanmittauslaitos 2024.)

Karttapaikassa näkyvät ilmakuvat ovat Maanmittauslaitoksen itse keräämiä, ne on otettu lentokoneilla ja niihin liittyvä kartastotyö aloitettiin 1930-luvulla. Kuvia päivitetään kolmen vuoden välein. Poikkeuksen muodostaa Pohjois-Lappi, joka päivitetään 12 vuoden välein. Pikselitarkkuus vaihtelee 0,25 – 0,5 metriä. (Niemelä 2004, 87; Maanmittauslaitos 2024.)

Historiallisia ilmakuvia voi selata erillisestä Paikkatietoikkuna-palvelusta eri vuosilta alueesta riippuen. Palvelussa ei kuitenkaan ole suoraan toimintoa, jolla voisi ladata kuvia omalle koneelle. Karttanäkymien tulostus kuitenkin onnistuu A3- ja A4-arkkikoossa. Tässä palvelussa karttaa voi myös kääntää eri ilmansuuntiin 3D-näkymässä. (Maanmittauslaitos 2024.)

## 3.3 GPS koordinaattijärjestelmä

WGS84 eli World Geodetic System on maailmanlaajuinen GPS koordinaattijärjestelmä, joka on ollut 80-luvulta asti käytössä. Se kehitettiin yhtenäistämään ja korvaamaan maakohtaisia koordinaattijärjestelmiä. WGS84:ssa käytetty keskimääräinen säde maapallolle on 6368,139 kilometriä ja maapallon litistyneisyysarvo 298,257223563. Nämä ovat merkittäviä suureita, koska ne aiheuttavat 20-180 metrin eroavaisuuksia esimerkiksi Suomessa aiemmin käytettyyn paikalliseen KKJ-koordinaattijärjestelmään verrattuna, johtuen eri koordinaattijärjestelmien käyttämistä maapallon muotoa kuvaavista malleista. (Miettinen 2006, 166-167.)

Maanmittauslaitoksen Karttapaikka-palvelussa erilaisia koordinaattijärjestelmiä on valittavissa useita esimerkiksi ETRS89. ETRS89-järjestelmä on koko Euroopan laajuinen WGS84-toteutus, joka luotiin 90-luvulla. EUREF-FIN pohjautuu ETRS89:een ja vastaa noin metrin tarkkuudella WGS84-koordinaatistoa. (Miettinen 2006, 166-167.) Google Earth käyttää vain WGS84-koordinaatistoa.

## 3.4 Karttapalveluiden vertailu ja valinta

Sekä Google Earth että Karttapaikka ovat ilmaisia palveluja, ja molemmista voidaan ladata tai tulostaa ilmakuva suoraan omalle tietokoneelle Suomen alueelta. Vaikka palveluntarjoajien sivuilla molempien palvelujen ilmakuvat ilmoitettiin lähes yhtä tarkoiksi, Google Earth -palvelussa ilmakuvien tarkkuus vaikutti silmämääräisesti korkeammalta. Karttapaikan ilmakuvat voivat joiltain alueilta olla uudempia, joka voi olla hyödyllistä, jos kuvien ajantasaisuus on oleellista.

Google Earthin sisäänrakennettu toiminnallisuus selata historiallisia ilmakuvia mahdollisti tarkimman mahdollisen kuvan löytämisen. Tämän avulla pystyttiin löytämään sellaiset kuvat, joissa olisi vähiten epäselvyyttä ja kohinaa. Erityisesti kohina vaikuttaa suoraan syväoppimismallin tarkkuuteen (Kananen & Puolitaival 2019, 64).

Tämän työn tavoitteiden kannalta ilmakuvien tarkkuus oli tärkein valintakriteeri, joten Google Earth valittiin. Kuvassa 5 näkyy Google Earth Pro -ohjelman kautta tallennettu ilmakuva, josta nähdään, että se on hyvin tarkka, jonka ansiosta esimerkiksi siirtolohkareen reunat erottuvat selkeästi. Aineiston keräämisen kannalta oli oleellista saada kuvat tallennettua helposti ja nopeasti sekä määrämittaisesti suoraan ohjelmasta tutkimuskoneelle. Tässäkin Google Earth Pro toimi paremmin.


Kuva 5. Ilmakuva (Google, © 2024 Airbus)

# 4 Käyttöliittymäkirjastot

## 4.1 WindowsForms ja WPF

C#:lla voidaan ohjelmoida käyttöliittymä Windows-ympäristöön käyttäen esimerkiksi WindowsForms- tai WPF-käyttöliittymäkirjastoa. WindowsForms on yksinkertaisempi, mutta ei niin tehokas. WPF on modernimpi, monipuolisempi ja tehokkaampi koska se käyttää laitteistokiihdytystä DirectX:n kautta. WindowsForms avulla tehty käyttöliittymä voidaan muuntaa myös macOs- ja Linux-ympäristöihin. (ByteHide 2023.)

Selkein ero WindowsForms ja WPF välillä on se, että WPF käyttää vektoreihin perustuvaa esitystapaa, joka mahdollistaa korkealaatuisen skaalautuvuuden ja animoinnin. WindowsForms puolestaan on rasteri eli pikselipohjainen ratkaisu. Toinen selkeä ero on, että WPF käyttää XAML-pohjaista asettelutapaa, kun taas WindowsForms käyttää lomakkeisiin ja ohjausobjekteihin perustuvaa lähestymistapaa. (ByteHide 2023.)

## 4.2 Python

Pythonilla on useita ilmaisia käyttöliittymäkirjastoja kuten Tkinter, PyQT, Kivy, wxPython ja PySimpleGUI. Valinta kirjastojen välillä riippuu käyttöliittymän vaatimuksista, missä ympäristöissä sovelluksen tulee toimia ja kuinka tehokas sovelluksen tulee olla. Esimerkiksi Tkinter on erittäin yksinkertainen käyttää, mutta sillä on vaikea toteuttaa visuaalisesti näyttäviä ratkaisuja. Kivy puolestaan saattaa olla monimutkaisempi käyttää mutta sillä voidaan toteuttaa laitteistokiihdytetty mukautuva mobiiliratkaisu. (Kummarikuntla 2023.)

Python on myös koneoppimisalustoilla yleisesti käytössä oleva ohjelmointikieli, joten se integroituu ratkaisuun saumattomasti, eikä aiheuta yhteensopivuusongelmia (Kananen ja Puolitaival 2019, 188). Python on kieliopiltaan selkeä, dynaamisesti tyypitetty, sisältää paljon avoimen lähdekoodin lisäosia ja on myös käyttöjärjestelmäriippumaton. Python soveltuu monimutkaisten sovellusten toteuttamiseen. (Järvenpää 2019, 18.)

## 4.3 Käyttöliittymäkirjastojen vertailu ja valinta

Tässä työssä tärkeintä oli yksinkertaisuus, käyttöönoton helppous, ja suora integroituminen muuhun ratkaisuun, joten Python valittiin. Pythonin vakioasennuksessa mukana tulevalla Tkinter-kirjastolla saatiin toteutettua kaikki käyttöliittymän vaatimukset muutamalla rivillä koodia. Lisäksi, vaikka työssä rajattiin tuki vain Windows-ympäristöön, Python mahdollistaa tuen ilman erillistä muuntamista myös macOS- ja Linux-ympäristöissä. Näin ollen ratkaisu voidaan helposti ottaa käyttöön myös näissä ympäristöissä.

Tkinter on kuitenkin ominaisuuksiltaan rajoittunut. Jos ratkaisua halutaan laajentaa esimerkiksi mobiilikäyttöön, voidaan käyttöliittymäkirjastona hyödyntää esimerkiksi Kivyä, joka on monipuolisempi ja juuri tähän tarkoitukseen kehitetty. Kivy tarjoaa mahdollisuuden luoda käyttöliittymiä, jotka ovat yhteensopivia sekä mobiililaitteiden että työpöytäympäristöjen kanssa.

# 5 Koneoppimisalustat

## 5.1 TensorFlow

TensorFlow on Googlen kehittämä avoimen lähdekoodin alusta koneoppimiseen, syväoppimiseen ja tensorilaskentaan (Kananen & Puolitaival 2019, 189). Syväoppimismallin kehityksessä voidaan hyödyntää esimerkiksi Googlen Colaboratory-palvelua, pilvessä toimivaa Jupyter notebook -ympäristöä. Tämä ympäristö ei vaadi käyttäjältä mitään asennuksia, vaan tarvittavat kirjastot ovat valmiina käytettäväksi. (TensorFlow 2024.)

TensorFlow'ta voidaan hyödyntää selaimessa TensorFlow.js avulla ja mobiililaitteissa TensorFlow Liten avulla. TFX-putken avulla voidaan rakentaa ja suorittaa syväoppimismalleja tuotannossa, jotka vaativat korkeaa suorituskykyä ja skaalautuvuutta. Keras API:n avulla voidaan luoda malleja korkean tason käskyillä. (TensorFlow 2024.)

TensorFlow perustuu graafipohjaiseen laskentamalliin, jossa eri operaatiot esitetään solmupisteinä tietovirtakaaviossa. TensorBoardin avulla voidaan visualisoida sekä itse malli että sen suorituskykymetriikat. TensorBoard helpottaa mallin suunnittelua ja analysointia. (Alvi 2024.)

## 5.2 PyTorch

PyTorch on Facebookin (nykyinen Meta) kehittämä avoimen lähdekoodin alusta erityisesti syväoppimismallien kehitykseen (Kananen & Puolitaival 2019, 189). PyTorch on suosittu tutkijoiden keskuudessa, sen dynaamisen laskentagraafin ja helppokäyttöisyyden ansiosta (Alvi 2024). Se sisältää myös suoran tuen viedä syväoppimismallit ONNX-formaattiin (Patel 2017).

Kuten myös TensorFlow, PyTorch voidaan helposti asentaa lokaalisti kehittäjän koneelle, muutamalla Python-komennolla. PyTorch onkin tunnettu sen pythonmaisesta luonteesta ja yksinkertaisuudesta, jonka takia sen oppiminen on helppoa. PyTorch sopii erityisesti kompleksisiin ja iteratiivisiin kehitysprojekteihin, joissa muutoksia tehdään useasti. (Alvi 2024.)

## 5.3 Koneoppimisalustojen vertailu ja valinta

Tässä työssä alustan helppokäyttöisyys, nopea käyttöönotto, sekä sisäänrakennettu tuki ONNX-formaatille oli tärkeää, siksi PyTorch valittiin. Työssä keskityttiin syväoppimismallin opettamisen yksityiskohtiin, jonka takia laskennan suorituskyvyn skaalautuvuus ei ollut

tärkeää, jota TensorFlow puolestaan olisi tarjonnut. Lisäksi, valittu syväoppimismalli YOLOv8 on kehitetty PyTorch-alustalle, jolloin sitä ei jouduttu erikseen muuntamaan.

TensorFlowssa olisi parempi tuki mobiililaitteille, mutta sekään ei ollut tämän työn tulosten kannalta oleellinen ominaisuus. Kuitenkin, siirryttäessä mahdolliseen tuotantokäyttöön, TensorFlown tuomia etuja suorituskyvyn skaalautuvuudessa sekä mobiilikäytössä tulee harkita. Ympäristö päätettiin asentaa paikalliselle tutkimuskoneelle kustannusten minimoimiseksi. Pilviympäristön käyttöä kuitenkin harkitaan, jos laskentatehovaatimus kasvaa datan lisääntyessä niin, ettei mallin kouluttaminen paikallisesti ole enää kustannustehokasta.

# 6 Syväoppimismallit

## 6.1 YOLOv8

YOLOv8 (You Only Look Once) on syväoppimismalli, jolla voidaan tunnistaa objekteja reaaliajassa annetusta lähtödatasta. Reaaliaikaista tunnistusta käytetään usealla eri alueella kuten esimerkiksi videovalvonnassa, robotiikassa ja itsestään ajavissa autoissa. YOLOv8:n erityispiirteenä on sen hyvä tasapaino nopeuden ja tarkkuuden välillä. YOLOv8 arkkitehtuuri on suunniteltu niin että se käsittelee syötteenä annetun kuvan yhdellä kertaa, eli on yksivaiheinen. (Terven ym. 2023, 1, 23.)

Aiempiin YOLO-versioihin nähden YOLOv8:n arkkitehtuuria on kehitetty parantamaan piirteiden tunnistusta ja objektien havaitsemista. Se hyödyntää ankkurivapaata tunnistamista, joka lisää tarkkuutta ja tehokkuutta ankkuripohjaisiin menetelmiin verrattuna. YOLOv8 on optimoitu niin että se tasapainottaa nopeuden ja tarkkuuden välillä. YOLOv8:n mukana tulee useita valmiiksi opetettuja malleja, mikä nopeuttaa oman mallin opettamista. (Ultralytics 2024.)

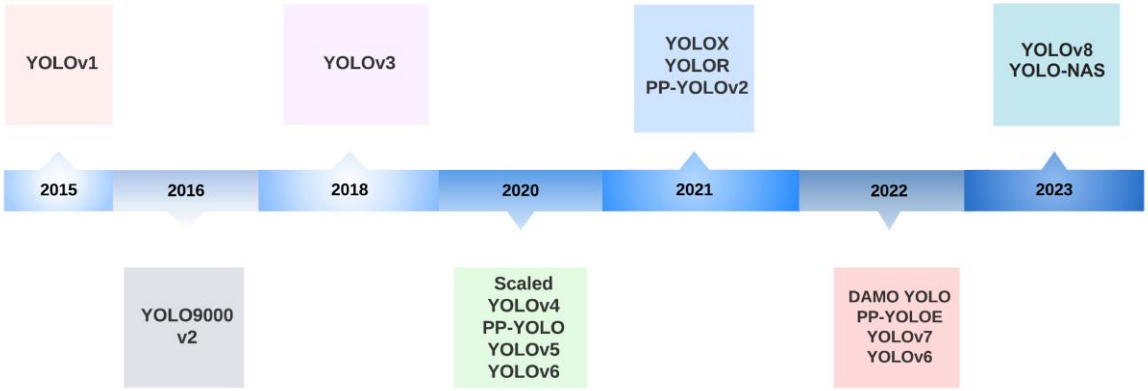
### 6.1.1 YOLOn historiaa

YOLO-mallia on kehitetty iteratiivisesti vuodesta 2015 lähtien. Jokainen uusi versio on suunniteltu poistamaan edellisen version rajoituksia sekä parantamaan mallin suorituskykyä. Mallin version 2 ja version 8 välillä on kehitetty muun muassa neuroverkon rakennetta, kustannusfunktioiden ja ankkurilaatikoiden toimintaa, sekä lähtödatan resoluution skaalausta. (Terven ym. 2023, 1-2.)

Kuvasta 6 nähdään YOLOn versioiden kehitys aikajanalla. Versiosta 3 vuodesta 2018 lähtien arkkitehtuurissa kuvattiin kolme pääosaa: selkäranka (backbone), kaula (neck) ja pää (head). Jokaisella osalla on tärkeä merkitys tunnistusprosessissa. (Terven ym. 2023, 13.)

Version 5 julkaisi vuonna 2020 Glenn Jocher, Ultralytics yrityksen perustaja. YOLOv5:ssa siirryttiin käyttämään Darknet-arkkitehtuurin sijasta PyTorchia. YOLOv5 toi mukanaan monia parannuksia kuten SPPF, joka nopeutti laskentaa. (Terven ym. 2023, 17-18.)

Ultralytics julkaisi version 8 vuonna 2023, ja se pohjautui versioon 5. Parannuksena oli C2f moduuli parantamaan tunnistamisen tarkkuutta sekä ankkurivapaa tunnistaminen. Samalla Ultralytics julkaisi viisi eri kokoista valmiiksi koulutettua YOLOv8-versiota. YOLOv8 tukee useita eri kuvantunnistus- ja tietokonenäkötehtäviä kuten luokittelu, segmentointi ja objektien tunnistaminen, seuranta ja asennon estimointi. (Terven ym. 2023, 22-23.)




Kuva 6. YOLO-versioiden kehitys aikajanalla (Terven ym. 2023, 1)

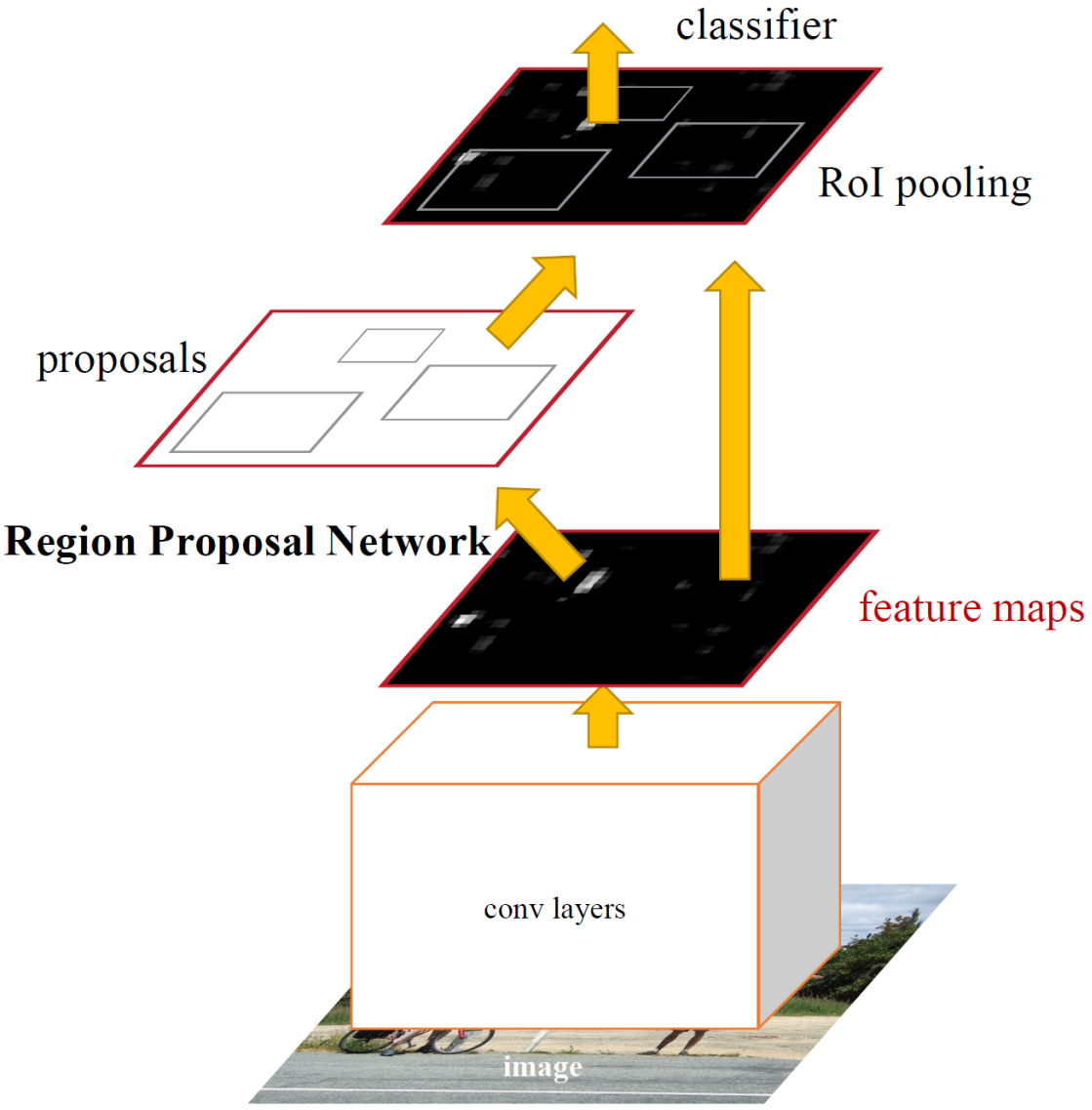
### 6.1.2 Ankkurivapaa tunnistaminen

Aiemmissa YOLO-versioissa käytettiin ennalta määrättyjä ankkurilaatikoita. Niiden avulla pystyttiin ennustamaan kullekin ankkurille koordinaatit ja tunnistettua objektin luokka suhteessa ruudukon solun sijaintiin. Malli opetettiin ennustamaan jokaiselle kohteelle esimäärätyssä ankkurilaatikoiden joukossa leveyden ja korkeuden skaalaus sekä keskiosoitteen siirtymä. (Magnusson 2023, 16.)

YOLOv8 käyttää niin kutsuttua ankkurivapaata mallia, joka tarkoittaa sitä, että mallin ulostulossa kuvatut rajauslaatikot ovat valmiiksi laskettuina niin, että ne kuvastavat rajauslaatikon keskipisteen sijaintia ja kokoa suhteessa alkuperäiseen kuvaan. Lisäksi sijainnin ja koon arviointi eriytettiin arkkitehtuurissa omaksi haarakseen. Nämä muutokset paransivat ennustamisen tarkkuutta ja yksinkertaistivat prosesseja. (Terven ym. 2023, 23.)

## 6.2 Faster R-CNN

Microsoft Research -ryhmän kehittämä syväoppimismalli Faster R-CNN on kaksivaiheinen. Se koostuu kahdesta erillisestä moduulista, alue-ehdotusverkosta (Region Proposal Network eli RPN) ja Fast R-CNN. Ensimmäisen moduulin tehtävä on ehdottaa alueita, joilla objekteja voi olla. Toisen moduulin tehtävä on havaita ja luokitella objektit näiltä alueilta. Faster R-CNN käyttää valmiiksi määriteltyjä referenssiankkurilaatikoita eri skaaloilla ja kuvasuhteilla eri alue-ehdotusten tunnistamiseen. Kuvassa 7 on esitetty tämä Faster R-CNN toimintaperiaate. (Ren ym. 2016, 2-3.)




Kuva 7. Faster R-CNN mallin toimintaperiaate (Ren ym. 2015, 3)

Alue-ehdotusverkon lisääminen osaksi Fast R-CNN:ää nopeuttaa ja tarkentaa tunnistamista huomattavasti. Lisäksi se parantaa alue-ehdotusten laatua ja yleistä objektien tunnistamisen tarkkuutta. Faster R-CNN mahdollistaa lähes reaaliaikaisen kuvantunnistuksen. (Ren ym. 2015, 11-12.)

## 6.3 ONNX standardi

ONNX (Open Neural Network Exchange) on avoin standardi, joka mahdollistaa koneoppimismallien siirtämisen eri koneoppimisalustojen välillä. Kun malli on käännetty ONNX-formaattiin, sitä voidaan hyödyntää missä tahansa ympäristössä. ONNX-formaatissa malli koostuu syötteistä, solmuista sekä ulostuloista. Solmut edustavat laskentaoperaatioita, esimerkiksi painokertoimien päivitys. (ONNX 2024.)

ONNX on Microsoftin ja Facebookin yhteistyössä kehittämä avoimen lähdekoodin projekti. Sen tarkoituksena on helpottaa kehittäjien liikkumista eri alustojen välillä ja poistaa viivettä mallien konvertoimisessa eri alustojen vaatimiin formaatteihin. Tällä tavoin tekoäly saadaan nopeammin tuottamaan arvoa ja enemmän saataville. (Boyd 2017.)

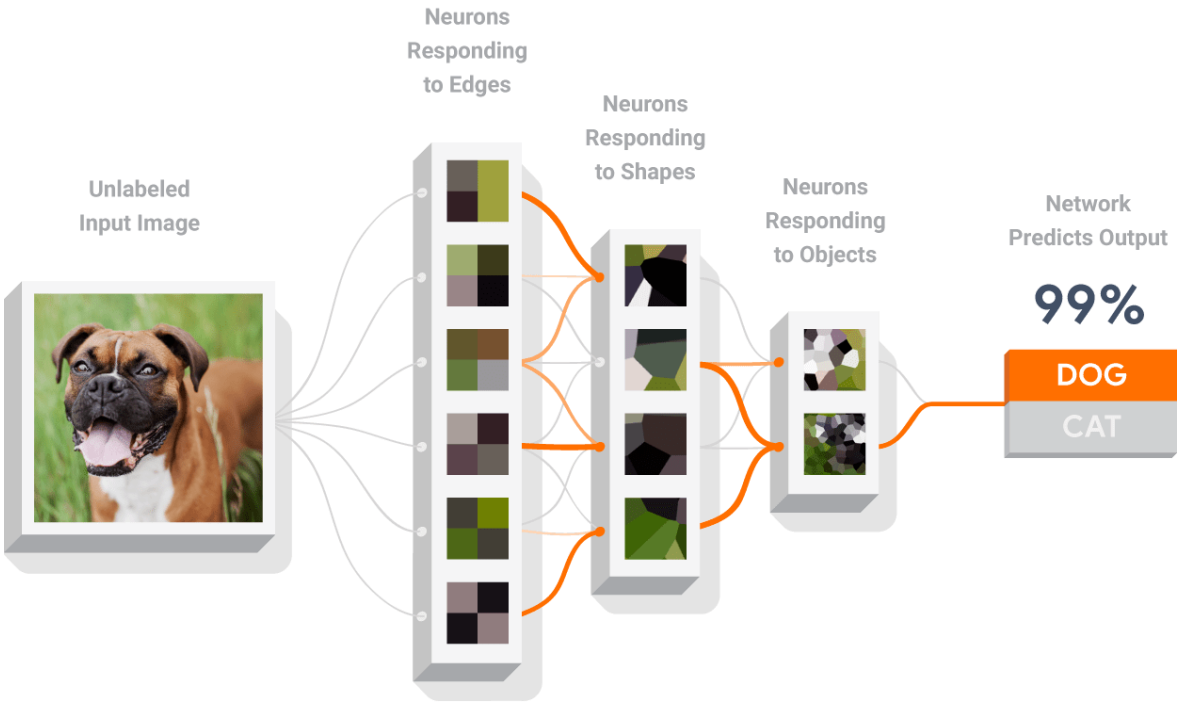
## 6.4 Syväoppimismallien vertailu ja valinta

YOLOv8 valittiin sen helpon käyttöönoton sekä suorituskyvyn ja nopeuden takia. Haluttiin ottaa käyttöön malli, joka soveltuu hyvin reaaliaikaiseen kuvantunnistukseen ja on myös tarkka. Faster R-CNN olisi voinut mahdollistaa YOLOv8:a tarkemman tunnistamisen, mutta tämän työn tavoitteiden kannalta tunnistamisen maksimaalinen tarkkuus ei ollut kriittistä. Tärkeintä oli tunnistamisen ja luokittelun prosessi ja sen onnistuminen.

YOLOv8 voi käyttää suoraan komentoriviltä valmiilla käskyillä joka entisestään helpotti käyttöönottoa. Faster R-CNN käyttöönotto olisi vaatinut enemmän oman Python-koodin kirjoittamista ja konfigurointia (Terven ym. 2023, 24; Ren ym. 2015, 2). Faster R-CNN asennukseen vaadittavat vaiheet on kuvattu GitHub-projektissa py-faster-rcnn.

## 6.5 Valitun syväoppimismallin toimintaperiaate

YOLOv8 oppii tunnistamaan ohjatun oppimisen ja optimointialgoritmien avulla onko annetussa lähdekuvassa yksi tai useampi mallin aiemmin oppiman luokan objekti. Jos objekteja tunnistetaan, saadaan myös ennuste missä kohtaa lähdekuvaa ne sijaitsevat ja millä todennäköisyydellä. Kuvassa 8 havainnollistetaan tunnistamisen toimintaperiaate korkealla tasolla.




Kuva 8. Tunnistamisen toimintaperiaate ylätasolla (Tensorflow 2024)

YOLOv8:ssa selkäranka-konvoluutio havaitsee reunoja ja muita perusominaisuuksia eri skaaloilla. Kaula-konvoluutio yhdistää selkärangan ja pään. Se yhdistää eli aggregoi ja tarkentaa selkärangan havaitsemia piirteitä. Pää-konvoluutio on viimeinen osa, jonka tehtävä on toteuttaa luokittelu ja paikannus eli itse ennustaminen. (Terven ym. 2023, 13-14.)

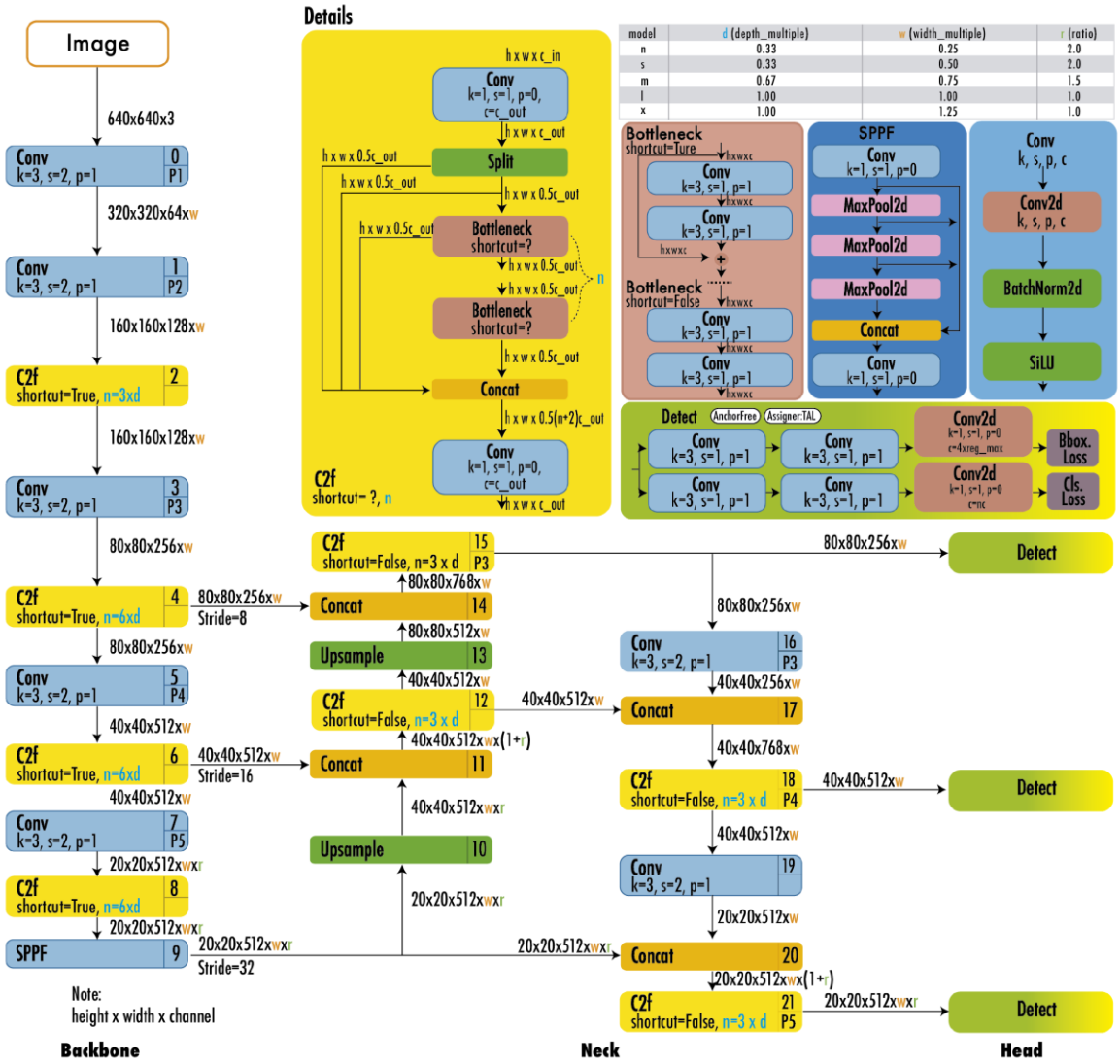
### 6.5.1 Neuronit ja tensorit

YOLOv8:ssa kuten kaikissa syväoppimismalleissa hyödynnetään neuroneja, jotka ovat neuroverkon perusyksikköjä. Ne käsittelevät syötteitä painokertoimien ja aktivaatiofunktioiden avulla (Alpaydin 2021, 108-109). Tensorit ovat moniulotteisia taulukoita, joiden avulla neuroverkkojen dataa käsitellään vektori- ja matriisilaskennan avulla (Kananen & Puolitaival 2019, 190).

Neuroverkot käsittelevät arvoja nollan ja yhden välillä, joten lähtödata täytyy muuntaa siihen muotoon. Esimerkiksi monivärikuvien tapauksessa värikanavakohtaiset pikselimatriisien väriarvot pitää skaalata arvosta 0-255 välille 0-1, jotta syväoppimismalli toimii oikein. Neuroverkko näkee kuvat pelkästään numeromatriiseina. (Kananen & Puolitaival 2019, 81-83.)

### 6.5.2 Konvoluutioneuroverkon rakenne ja toiminta

Kuvassa 9 esitetyssä YOLOv8:n arkkitehtuurissa on yhteensä 22 kerrosta (numeroitu 0-21). Ne koostuvat konvoluutioista, C2f:stä (Cross-stage partial bottleneck with two convolutions), SPPF:stä (Spatial Pyramid Pooling – Fast), Upsampleista, jotka lisäävät ominaisuustensorien tarkkuutta ja Concateista, jotka yhdistävät eri tasojen ominaisuustensorit toisiinsa. Detect, lopullinen kerros, tuottaa ennusteet tunnistettavista kohteista. (Terven ym. 2023, 17, 23.)




Kuva 9. YOLOv8:n arkkitehtuuri (Terven ym. 2023, 23)

**Konvoluutiot**

Konvoluutioiden tehtävä on havaita ja eristää piirteitä ja perusominaisuuksia kuvasta, kuten ääriviivoja, pysty- ja vaakalinjoja, ja tuottaa nämä syötteeksi seuraavalle kerrokselle. Jokainen kerros monimutkaistaa ja laajentaa mallia edellisen kerroksen syötteen

perusteella. Toistamalla tätä tarpeeksi, pystytään tunnistamaan pelkästä pikselidatasta monimutkaisia rakenteita kuten ihmisen kasvot. Tämä on syväoppimisen perusperiaate. On tutkittu, että ihmisen näköaisti toimii samalla periaatteella. (Alpaydin 2021, 123-128.)

**Cross-stage partial bottleneck with two convolutions**

C2f-moduulin tehtävä on yhdistää korkean tason ominaisuuksia konvoluutioiden eristämiin piirteisiin. Tämä yhdistäminen lisää kontekstuaalista tietoa, mikä parantaa tunnistamisen tarkkuutta. CSPLayer nimellä aiemmin tunnettu C2f-moduuli hyödyntää kahta konvoluutiota piirteiden yhdistämisessä. (Terven ym. 2023, 22.)

**Spatial Pyramid Pooling – Fast**

Pooling-vaiheen tarkoitus on tuoda esille ja korostaa kaikkein oleellisimmat konvoluutioiden tunnistamista piirteistä. Kuvainformaation määrää pienennetään, esimerkiksi 5x5 matriisista voi tulla 3x3 kokoinen. Tämä pienennetty kuvainformaatio sisältää kaikkein oleellisimmat piirteet, joita kuvasta on tunnistettu. (Kananen & Puolitaival 2019, 152-153.)

SPPF-menetelmässä kuva jaetaan eri kokoisiin alueisiin, ja piirteet kerätään eri tasoilta. Piirteet yhdistetään kiinteäksi vektoriksi, joka syötetään neuroverkon seuraavaan kerrokseen. SPPF-menetelmän tarkoitus on myös poistaa kiinteän kuvakoon rajoite, joka tulee vaatimuksena konvoluutioneuroverkkoon kytketystä neuroverkosta. (He ym. 2014, 1-2.)

**Detect**

Jos malli on kooltaan 640x640 niin lopullisten ennusteiden määrä voidaan päätellä seuraavasti. Ne määräytyvät YOLOv8:n selkärangan stride-arvoista $8^2$,$16^2$ ja $32^2$ pikseliä per solu, kuten kaavassa 1 on esitetty. Toisin sanoen 640x640-malli ennustaa yhteensä 8400 rajauslaatikkoa, joista jokainen koostuu usean luokan todennäköisyyksistä.

$$\left(\frac{640}{8}\right)^2 + \left(\frac{640}{16}\right)^2 + \left(\frac{640}{32}\right)^2 = 6400 + 1600 + 400 = 8400 \qquad (1)$$

### 6.5.3 Ohjattu oppiminen

YOLOv8 käyttää ohjattua oppimista (supervised learning). Se on yleinen oppimistyyppi kuvan tunnistuksessa. Puhutaan myös syväoppimisesta, joka käytännössä tarkoittaa monimutkaisten neuroverkkojen käyttöä mallin opettamisessa. Ohjatussa oppimisessa tarvitaan paljon dataa, jotta malli saadaan toimimaan. (Kananen & Puolitaival 2019, 43, 127.)

Ohjatussa oppimisessa tiedetään etukäteen jokaisen tietoaineiston esiintymää vastaava ulostuloarvo. Tämä aiheuttaa myös sen, että tietoaineisto pitää etukäteen kerätä ja valmistella, mikä on aikaa vievä prosessin vaihe. Tietoaineiston kerääminen voi myös olla taloudellisesti kallista. (Kelleher 2020, 32.)

**Data - kaikkein tärkein tekijä**

Ohjatussa oppimisessa data on kaikkein tärkein tekijä mallin laadun ja suorituskyvyn kannalta. Mitä enemmän dataa ja mitä laadukkaampaa - yhdenmukaista ja yksiselitteistä - se on, sitä parempi mallista voidaan saada. Data jaetaan koulutusdataan, validointidataan ja testausdataan. Kuvassa 10 on esitetty yksi tapa jakaa datasetti, jos lähtödataa on käytössä vähän. (Kananen & Puolitaival 2019, 47, 62, 72.)


Kuva 10. Datan jakaminen oppimisen eri vaiheisiin

Datan ja aineiston vinouma on huomioitava tekijä. Jos halutaan mallin oppivan yleistämään hyvin eri luokkien välillä, tulee aineistoa kerätä paljon ja tasapuolisesti näistä luokista. Muutoin malli oppii vinoutuneesti tunnistamaan paremmin sen luokan, josta se on saanut eniten dataa. (Alpaydin 2021, 184-185.)

**Takaisinvirtausalgoritmi**

YOLOv8:n sisäinen oppimisprosessi eli takaisinvirtausalgoritmi (backpropagation) on monivaiheinen. Koulutusdatasetistä otetaan satunnainen joukko kuvia se määrä mitä parametreillä on säädetty, niin kutsuttu minierä (batch). Nämä kuvat syötetään neuroverkon läpi ja saadaan ennusteina rajauslaatikot sekä luokat. Näitä ennusteita verrataan todellisiin oikeisiin tietoihin, jotka saadaan koulutusdatasetin annotaatiotiedostoista. (Germanov 2023; Kananen & Puolitaival 2019, 43, 136-137.)

Ennusteen ja todellisen tiedon välinen erotus eli virhe lasketaan kustannusfunktiolla. Kustannusfunktion arvoa pyritään minimoimaan optimointialgoritmin avulla, joka on käytännössä derivointia. Optimoinnin avulla säädetään mallin painokertoimia taaksepäin ylös tai alaspäin. (Germanov 2023; Kananen & Puolitaival 2019, 43, 136-137.)

Painokertoimia säädetään jokaiselle ulostulokerroksen neuronille laskemalla niille virhegradientit ja syöttämällä ne sen jälkeen neuroverkossa taaksepäin. Painokertoimia päivitetään suhteessa ulostulojen virhegradientteihin. Tämä on iteratiivinen prosessi, jonka

tarkoitus on saada neuroverkon ulostulot supistumaan mahdollisimman tarkalle tasolle. (Kelleher 2020, 191.)

**Mallin validointi**

Mallin validointivaiheessa YOLOv8 syöttää validointidatasetin mallin läpi ja laskee ennusteen tarkkuuden suhteessa todelliseen dataan (Germanov 2023). Tätä vaihetta toistetaan koulutusprosessin aikana, jolloin nähdään miten hyvin malli ennustaa datalla, jota se ei ole ennen nähnyt. Näin voidaan seurata koulutuskierrosten edetessä, että malli oppii sekä opetusdatan hyvin, mutta myös tarkkuus validointidatasetin suhteen paranee. (Kananen & Puolitaival 2019, 138.)

Validointi on vaiheena tärkeä koska se näyttää, jos malli ylisovittaa. Ylisovittaminen tarkoittaa, että malli oppii koulutusdatan piirteet niin hyvin, että se menettää kyvyn yleistää. Tällöin se ei suoriudu hyvin, kun se ennustaa uudella datalla. Ongelmaa voidaan pienentää lisäämällä malliin monipuolisempaa dataa. (Kananen & Puolitaival 2019, 101.)

Kun malli on koulutettu, sitä voidaan testata ja tällöin käytetään testausdataa, jota ei käytetty lainkaan koulutuksen aikana. Testitulokset antavat käsityksen siitä, kuinka hyvin malli yleistää uusiin havaintoihin. Kun malli suoriutuu hyvin testidatalla, sen voidaan olettaa toimivan luotettavasti myös käytännössä. (Kananen & Puolitaival 2019, 47.)

### 6.5.4 Siirto-oppiminen

YOLOv8 hyödyntää myös siirto-oppimista. Oppimisprosessin pohjana käytetään jo valmiiksi laajalla aineistolla opetettua konvoluutioneuroverkkoa, ja siinä valmiiksi laskettuja painokertoimia. Nämä valmiiksi lasketut ensimmäiset neuroverkon kerrokset kopioidaan käyttöön, jolloin myöhempien kerrosten opettaminen voidaan tehdä pienemmällä datamäärällä. (Alpaydin 2021, 136).

YOLOv8:ssa on tarjolla useita valmiiksi opetettuja malleja (nano, small, medium, large ja extra large), joiden pohjadatana on käytetty COCO (Common Objects in Context) nimistä datasettiä. COCO sisältää yli 200 000 kuvaa yli 80 objektikategoriasta kuten auto, moottoripyörä, kissa ja koira. COCO on laajasti käytössä tietokonenäön tutkimuksessa, koska se tarjoaa monipuolisen joukon eri objektikategorioita, jotka on opetettu suurella määrällä dataa. (Terven ym. 2023, 22; Ultralytics 2024.)

### 6.5.5 Kustannusfunktiot

YOLOv8 käyttää Distribution Focal Loss ja Complete Intersection over Union kustannusfunktioita, kun se laskee miten hyvin mallin nykyinen ennuste ja todellisuus

täsmäävät rajauslaatikkojen osalta. Distribution Focal Loss parantaa objektintunnistusta korostamalla vaikeasti havaittavia kohteita, kuten pieniä tai päällekkäisiä esineitä, ja jakamalla painotukset eri objektien skaalojen ja luokkien välillä. Complete Intersection over Union mittaa, kuinka suuri osa ennustetun ja todellisen rajauslaatikon alueista menee päällekkäin suhteessa niiden yhteenlaskettuun kokonaisalaan. Tämä antaa mittausarvon siitä, kuinka hyvin ennustetut rajauslaatikot vastaavat todellisia kohteita. (Terven ym. 2023, 4, 23-25.)

YOLOv8 käyttää myös binääristä ristikkäisentropiafunktiota. Sen avulla lasketaan miten hyvin luokan ennustus vastaa oikeaa todellista arvoa eli kuinka merkittävästi todennäköisyysjakaumat eroavat toisistaan. YOLOv8 muuttaa mallin painokertoimia vähentääkseen tätä eroavaisuutta. (Terven ym. 2023, 23; Louridas 2021, 210.)

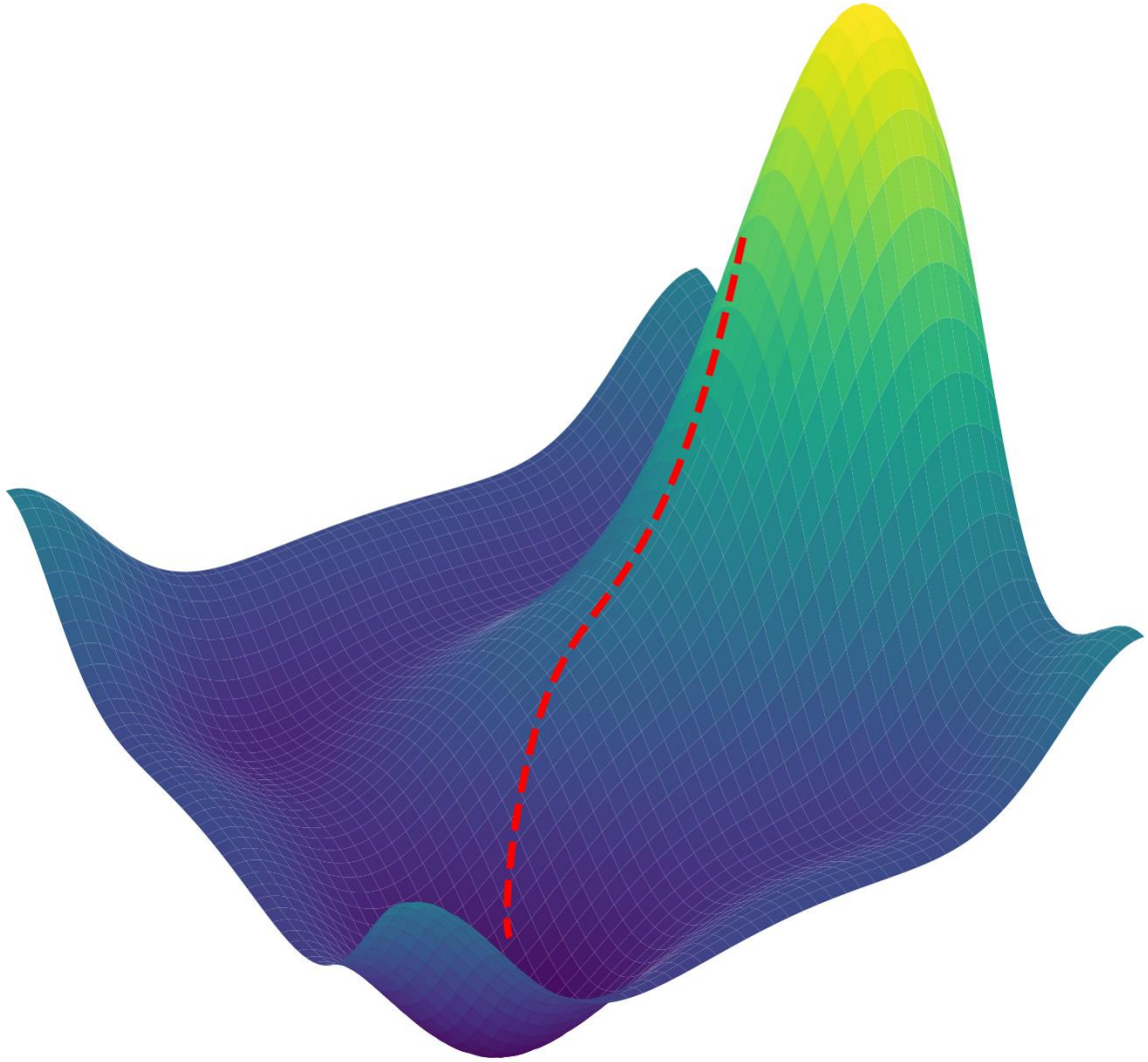
### 6.5.6 Optimointialgoritmit ja painokertoimien päivitys

Optimointialgoritmin avulla muutetaan neuronien painokertoimia vähän kerrallaan. Näin päästään lähemmäs kohti kustannusfunktion minimikohtaa, eli virhe pienenee. Tätä derivointia kutsutaan nimellä gradienttilasku, ja tällä tavalla malli konkreettisesti oppii (Kananen & Puolitaival 2019, 137).

Gradientti on vektori, mikä sisältää kaikkien neuronien virheiden osittaisderivaatat. Se ilmaisee suunnan, johon kuljettaessa virhe suurenee. Gradienttilasku kuvaa sitä, että kuljetaan tämän suunnan vastaisesti. (Louridas 2021, 191.)

YOLOv8 käyttää oletuksena stokastista gradienttilaskua (SGD), mutta käytettävissä on myös muita optimointialgoritmeja kuten Adaptive Moment Estimations (Adam) (Ultralytics 2024; Germanov 2023). Painokertoimia muutetaan oppimisnopeus-parametrin verran – vähän kerrassaan – jotta saavutetaan kustannusfunktion minimi. Jos oppimisnopeus on liian suuri, voidaan hypätä kustannusfunktion minimin yli. Stokastinen tarkoittaa satunnaista, eli gradienttilaskuun liitetään satunnaisuutta, kun gradienttilaskun suuntaa lasketaan. (Kelleher 2020, 174, 182.)

Kuvassa 11 esitetään, miten gradienttilasku -algoritmi toimii käytännössä. Pythonin pyplot-kirjaston avulla on visualisoitu virhepinta, ja toteutettu gradienttilasku kukkulan päältä oppimisnopeudella 0.01 (punainen katkoviiva). Virhegradienttien suunta siirtyy per opetuskierros kohti virheen globaalia minimiä ja lopulta optimaalisin ratkaisu löydetään.



Kuva 11. Gradienttialgoritmin eteneminen virhepinnalla (mukailtu Kolari & Kallio 2023, 133)

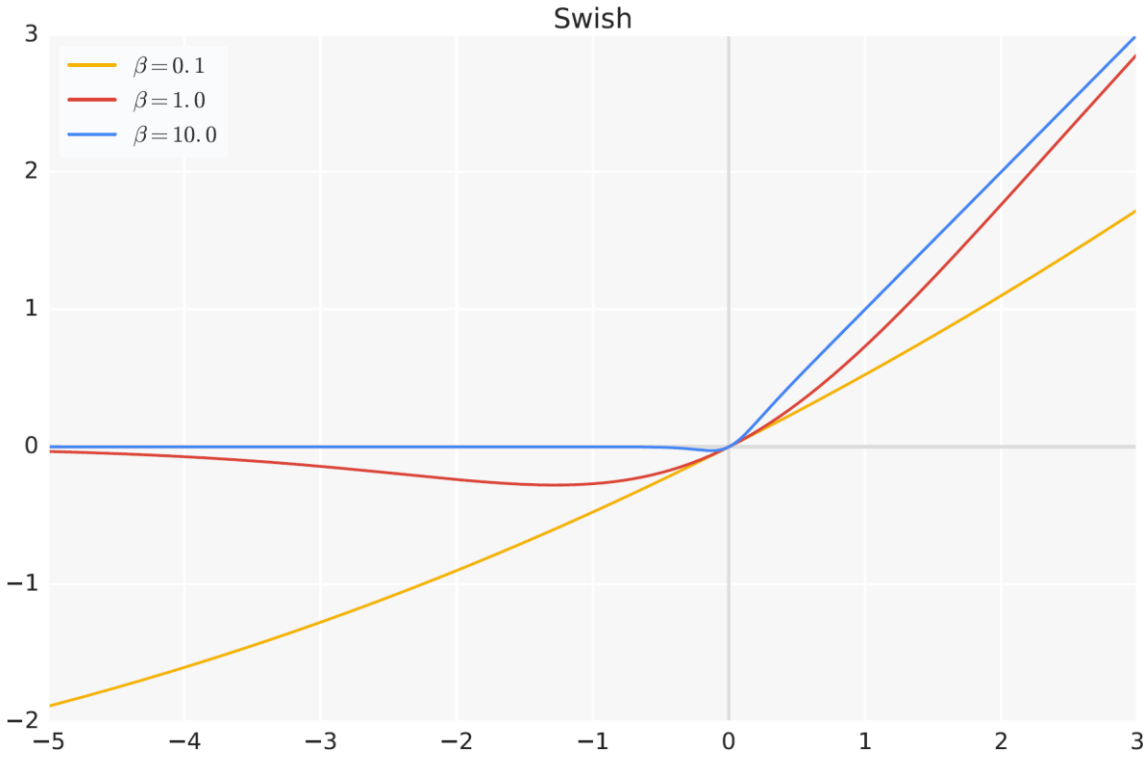
### 6.5.7 Aktivaatiofunktiot

Aktivaatiofunktio määrittää milloin yksittäinen neuroni aktivoituu. Sigmoid, ReLu ja Leaky ReLu ovat yleisesti käytössä olevia aktivaatiofunktioita. Aktivaatiofunktion laskentanopeus vaikuttaa mallin suorituskykyyn. (Kananen & Puolitaival 2019, 132.) Aktivaatiofunktion on oltava derivoituva, eli siinä ei saa olla epäjatkuvuuskohtia, jotta sitä voidaan käyttää takaisinvirtaavissa neuroverkoissa. (Kelleher 2020,194.)

YOLOv8 käyttää Sigmoid-Weighted Linear Unit (Swish) -aktivaatiofunktiota konvoluutiokerrosten välillä. (Terven ym. 2023, 23.). Swish-aktivaatiofunktio etsittiin ja löydettiin tekoälyn avulla. Sen muoto on esitetty kaavassa 2. (Ramachandran ym. 2017, 5.)

Swish-funktio interpoloi ei-lineaarisesti ReLU-funktion ja lineaarisen funktion välillä. Interpoloinnin astetta voidaan hallita parametrin β avulla. Swish-funktion toiminta eri parametrin β:n arvoilla on esitetty kuvassa 12. (Ramachandran ym. 2017, 5.)

$$f(x) = x \times sigmoid\ (\beta x) = \frac{x}{1+ e^{-\beta x}} \quad (2)$$




Kuva 12. Swish-aktivaatiofunktio (Ramachandran ym. 2017, 5)

### 6.5.8 Non-Maximum Suppression

Syväoppimismalli voi tuottaa tuhansia ennusteita luottamusvälillä 0 – 100 %. Jotta ennusteiden joukosta voidaan poimia luotettavimmat, ja verrata niitä todellisiin rajauslaatikkojen arvoihin oppimisprosessin aikana, YOLOv8 hyödyntää NMS eli Non-Maximum Suppression -algoritmia. NMS-algoritmi käy jokaisen ennustetun rajauslaatikon läpi aloittaen siitä, millä on suurin luottamus-%. Valittua rajauslaatikkoa verrataan kaikkiin muihin jäljellä oleviin. Jos pienemmän luottamus-%:n saaneella rajauslaatikolla on yhtenäistä pinta-alaa suhteessa yhteiseen pinta-alaan enemmän kuin tietyn raja-arvon verran, se laatikko poistetaan. Tätä päällekkäisen ja yhteisen pinta-alan suhdelukua kutsutaan termillä Intersection Over Union (IoU). Kuvassa 13 on esitetty NMS-algoritmin lopputulos. (Terven ym. 2023, 4-5.)

IoU lasketaan seuraavalla kaavalla 3, jossa A ja B ovat laatikoiden pinta-alat ja C on laatikoiden päällekkäinen pinta-ala. Huomataan että IoU-arvo kasvaa päällekkäisen pinta-

alan kasvaessa. IoU arvolla 1 tarkoittaa täydellistä päällekkäisyyttä, jossa ennustettu ja todellinen alue ovat identtiset.

$$IoU = \frac{C}{A+B-C} \quad (3)$$


Kuva 13. NMS-algoritmin lopputulos (Terven ym. 2023, 5)

Mallin opetuksen aikana mitataan, kuinka tehokkaasti NMS-algoritmi tunnistaa objektin oikeasta sijainnista tietylle luokalle. Tätä suorituskykymittaria kutsutaan termillä Average Precision. Lopuksi lasketaan keskiarvo luokkakohtaisista mittaustuloksista, jolloin saadaan koko mallin luokittelukykyä kuvaava mittari Mean Average Precision eli mAP. (Terven ym. 2023, 3-4.)

## 6.6 Valitun syväoppimismallin keskeiset parametrit

Hyperparametrien avulla ohjataan itse syväoppimismallin opettamisprosessia. Niiden arvot määräytyvät kokeellisen työn eli yrityksen ja erehdyksen kautta. Esimerkiksi oppimisnopeuden määrittämiseen ei ole suoraa vastausta, vaan sen vaikutus täytyy testata käytännössä. (Kelleher 2020, 92-93.)

YOLOv8:ssa voidaan säätää opettamisprosessia hyvin monipuolisesti erilaisten hyperparametrien, eli ennen opetusta asetettavien parametrien avulla. Näitä ovat esimerkiksi koulutuskierrosten lukumäärä, oppimisnopeus ja optimoija-algoritmi. Osa parametreistä säätää oppimisprosessin tehokkuutta ja kestoa, kun taas osa parametreistä vaikuttaa siihen kuinka tarkka tulos mallin opettamisessa saavutetaan. (Ultralytics 2024.)

### 6.6.1 Epookki, kärsivällisyys ja satunnaissiemen

Epookki-muuttujan (epoch), eli koulutuskierrosten avulla hallitaan sitä, kuinka monta kertaa koko koulutusaineisto käydään läpi, ja painokertoimia muutetaan, kun mallia opetetaan.

Mitä enemmän koulutuskierroksia suoritetaan, sitä todennäköisemmin mallin luotettavuus paranee. Koulutuskierroksia suoritetaan yleensä tuhansia. (Kananen & Puolitaival 2019, 136.; Ultralytics 2024.)

Kärsivällisyys-muuttujan (patience) avulla voidaan lopettaa opetus, jos havaitaan tietyn opetuskierroslukumäärän jälkeen, että merkittävää parannusta suorituskykymittareissa ei ole enää tapahtunut suhteessa validointidatasettiin. Satunnaissiemen-muuttujan (seed) avulla voidaan malli asettaa tiettyyn lähtötilanteeseen satunnaismuuttujien suhteen, ja näin ollen taata, että jokainen koulutuskerta tuottaa saman lopputuloksen. Tämä tekee mallista täysin deterministisen, eli se tuottaa saman lopputuloksen jokaisella koulutuskerralla. (Ultralytics 2024.)

### 6.6.2 Minierä, työprosessit ja välimuisti

Minierä-muuttujan (batch) avulla hallitaan sitä, kuinka monta kuvaa prosessoidaan ennen kuin syväoppimismallin sisäiset parametrit päivitetään. Tämän muuttujan kokoa voidaan kasvattaa, mutta se vaatii enemmän muistia koneelta, jolla mallia lasketaan. Isommalla minierä-muuttujan arvolla oppiminen voi tapahtua nopeammin, kun taas pienemmällä arvolla malli voi oppia yleistämään paremmin. (Ultralytics 2024.)

Käytössä olevaa RAM-muistia voidaan hyödyntää kasvattamalla työprosessit-muuttujan (workers) arvoa sekä asettamalla välimuisti (cache) käyttöön. Välimuistin avulla kuvat ladataan muistiin eikä näin ollen kuormiteta levyjärjestelmää oppimisprosessin aikana. Työprosessien lukumäärä vaikuttaa siihen, kuinka nopeasti dataa ladataan ja esikäsitellään rinnakkain. (Ultralytics 2024.)

### 6.6.3 Kuvakoko ja kuvasuhde

Kuvakoko-muuttujan (ImgSz) avulla määritetään mihin pikselikokoon (korkeus x leveys) lähdekuva venytetään, ennen kuin se syötetään syväoppimismallille opetusta varten. Mitä suurempi pikselikoko, sitä tarkempi mallista tulee. Kuvakoon kasvattaminen oletusarvosta 640x640 kasvattaa vastaavasti laitteiston vaatimuksia, erityisesti muistin määrää. (Ultralytics 2024.)

Kuvasuhde-muuttuja (Rect) on oletuksena False, joka tarkoittaa, että kuvan molemmat dimensiot venytetään kuvakoko-muuttujan arvoon. Tämä johtaa kuvasuhteen vääristymään. Jos lähtöaineisto ei ole neliön muotoista, ja kuvasuhteen ei haluta vääristyvän, voidaan käyttää kuvasuhde-muuttujan arvoa True. Tällöin jokaisessa minierässä lähdekuvat skaalataan pidemmän sivun mukaan kuvakoko-muuttujan arvoon. Esimerkiksi jos kuvakoko on 640, niin 1024x768 kokoinen lähdekuva skaalataan 640x480

kuvasuhde säilyttäen, käyttäen Pythonin OpenCV-kirjaston lineaarista interpolaatiota, kuten kuvassa 14 esitetään. (Ultralytics 2024.)

```
21    class BaseDataset(Dataset):
144       def load_image(self, i, rect_mode=True):
160               h0, w0 = im.shape[:2]  # orig hw
161               if rect_mode:  # resize long side to imgsz while maintaining aspect ratio
162                   r = self.imgsz / max(h0, w0)  # ratio
163                   if r != 1:  # if sizes are not equal
164                       w, h = (min(math.ceil(w0 * r), self.imgsz), min(math.ceil(h0 * r), self.imgsz))
165                       im = cv2.resize(im, (w, h), interpolation=cv2.INTER_LINEAR)
166               elif not (h0 == w0 == self.imgsz):  # resize by stretching image to square imgsz
167                   im = cv2.resize(im, (self.imgsz, self.imgsz), interpolation=cv2.INTER_LINEAR)
```

Kuva 14. Lähdekuvan käsittely YOLOv8:n BaseDataset-luokassa (Ultralytics 2024)

### 6.6.4 Oppimisnopeus ja optimointialgoritmi

Oppimisnopeus-muuttuja (Lr0) on yksi YOLOv8 tärkeimmistä hyperparametreistä, yhdessä optimointialgoritmin (optimizer) kanssa. Näiden avulla voidaan hallita sitä, kuinka nopeasti mallin painokertoimet päivitetään ja kuinka nopeasti malli lähestyy paikallista minimiä ja maksimia. Oletuksena YOLOv8 päättelee tämän arvon automaattisesti välille 0.01 – 0.001 optimointialgoritmin perusteella. (Ultralytics 2024.)

Jos oppimisnopeus asetetaan liian suureksi, voi käydä niin että neuroverkon painoja päivitetään liian paljon, jolloin hypätään optimaalisen kohdan yli. Jos taas oppimisnopeus on liian pieni, opettamisprosessi voi kestää todella pitkän ajan. Oppimisnopeuden määrittäminen on kompromissi opetustuloksen tarkkuuden ja opetusprosessin keston välillä. (Kelleher 2020, 102-104.)

### 6.6.5 Näytönohjain, CUDA ja automaattinen sekatarkkuus

Jotta oppiminen olisi mahdollisimman nopeaa ja tehokasta, voidaan hyödyntää näytönohjaimien CUDA:a (Compute Unified Device Architecture) -prosessoritehoa ja cuDNN-kirjastoa (CUDA Deep Neural Network). Näiden avulla virhegradientteja voidaan laskea tehokkaasti rinnakkain. Näytönohjaimen CUDA-laskennan aktivointi onnistuu YOLOv8:ssa valinnalla device=0. (Ultralytics 2024.)

CUDA on NVIDIAn vuonna 2007 kehittämä C-kieleen perustuva ohjelmointirajapinta. CUDA:sta on muodostunut standardi, ja sitä käytetään yleisesti laskentatehtävissä, joissa vaaditaan suurta laskentatehoa. CUDA mahdollistaa rinnakkaislaskennan hyödyntämisen NVIDIAn näytönohjaimilla, mikä nopeuttaa merkittävästi syväoppimismallien opettamista. (Kelleher 2020, 139.)

Tensorien eli moniuloitteisten taulukoiden avulla laskenta on hyvin keskeistä tekoälyssä, ja siihen tarvitaan paljon laskentatehoa. Tekoälyn kouluttamiseen on saatavilla erityisesti niitä varten kehitettyjä prosessoreja, Tensor Processing Unitteja. Laitteistoa ja laskentatehoa on mahdollista ostaa myös palveluna pilvestä. (Kananen & Puolitaival 2019, 190-191.)

Automaattinen sekatarkkuus (Automatic Mixed Precision eli AMP) mahdollistaa sekä 16- että 32-bittisten liukulukujen käytön laskennassa, jolloin laskenta voi nopeutua ja muistinkäyttö pienentyä. Laskennan tarkkuus voi kuitenkin hieman heikentyä, koska aktivaatioarvot ja virhegradientit konvertoidaan 16-bittiseen muotoon. Asetus voidaan tarvittaessa kytkeä pois päältä asetuksella amp=false. (Ultralytics 2024.)

## 6.7 Valitun syväoppimismallin keskeiset suorituskykymetriikat

Syväoppimismallin kelpoisuutta mitataan erilaisilla mittareilla, jotta tunnistetaan kuinka hyvin malli tunnistaa oikein suhteessa annettuun lähtöaineistoon. Oikeiden kelpoisuusfunktioiden valinta on ratkaisevaa, jotta mallin laatu saadaan halutulle tasolle. Uusia kelpoisuusfunktioita tutkitaan jatkuvasti. (Kelleher 2020, 31-32.)

YOLOv8 tarjoaa kelpoisuuden mittaamiseen automaattisesti useita yleisiä, hyväksi todettuja mittareita. Näistä keskeisimmät ovat tarkkuus, herkkyys ja $F_1$-arvotus. Niistä saadaan myös vastaavat graafit tuotettua automaattisesti kuten $F_1$-graafi.

### 6.7.1 Tarkkuus

Tarkkuus (precision) mittaa kuinka monta todellista positiivista tapausta havaittiin suhteessa kaikkiin havaintoihin, jotka sisältävät sekä oikeat että väärät positiiviset tapaukset. Kun tarkkuus on lähellä arvoa 1, malli tunnistaa hyvin oikeat tapaukset. Kääntäen kun tarkkuus on vähemmän kuin 1, malli tunnistaa oikeiksi myös vääriä tapauksia. (Alpaydin 2021, 75.)

Kuten kaavasta 4 nähdään, tarkkuus (P) kasvaa sitä suuremmaksi mitä vähemmän vääriä positiivisia (VP) havaintoja tehdään. Tämän työn näkökulmasta väärä positiivinen havainto on, että tunnistetaan lähtökuvasta siirtolohkareeksi jotain mikä ei ole oikeasti siirtolohkare. Todelliset positiiviset (TP) ovat oikein tunnistettuja siirtolohkareita.

$$P = \frac{TP}{TP+VP} \qquad (4)$$

### 6.7.2 Herkkyys

Herkkyys (recall) mittaa kuinka hyvin todelliset positiiviset tapaukset tunnistetaan. Se on todellisten positiivisten suhde kaikkiin todellisiin tapauksiin, eli todelliset positiiviset ja väärät

negatiiviset. Korkea herkkyys tarkoittaa, että malli havaitsee suuren osan todellisista positiivisista tapauksista. (Alpaydin 2021, 76.)

Kaava 5 esittää herkkyyden (R) määritelmän. Siitä nähdään, että herkkyys kasvaa sitä suuremmaksi mitä vähemmän vääriä negatiivisia (VN) havaintoja tehdään. Tämän työn näkökulmasta väärä negatiivinen havainto on, että lähtökuvassa ei havaita siirtolohkaretta, vaikka siinä on sellainen.

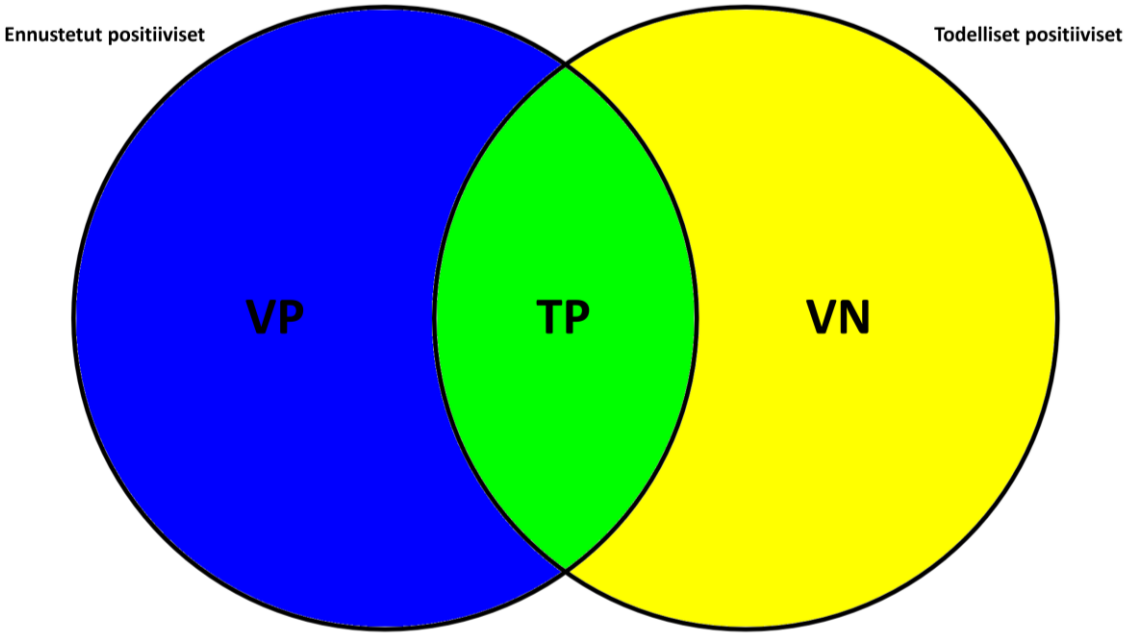
$$R = \frac{TP}{TP+VN} \qquad (5)$$

### 6.7.3 $F_1$-arvotus

Tarkkuuden ja herkkyyden välillä on tasapaino. Kasvattamalla toista voi pienentää toista arvoa (Alpaydin 2021, 75). Tämän takia näiden mittarien välillä lasketaan harmoninen keskiarvo $F_1$, joka tasapainottaa molemmat mittarit ja jonka arvo on välillä 0 ja 1 (Sasaki 2007, 1).

Kuten kaavasta 6 nähdään, $F_1$-arvotus kasvaa sitä suuremmaksi mitä vähemmän vääriä positiivisia (VP) ja vääriä negatiivisia (VN) havaintoja tehdään. Kuvassa 15 nähdään geometrisenä esityksenä eri mittareiden välinen suhde: Mitä suurempi TP eli vihreä alue, sitä suurempi on $F_1$. $F_1$ arvolla 100 % tarkoittaa mallin täydellistä suorituskykyä, jossa kaikki positiiviset tapaukset ennustetaan oikein, eikä vääriä positiivia tai vääriä negatiivisia ollut lainkaan.

$$F_1 = \frac{2PR}{P+R} = \frac{2TP}{2TP+VP+VN} \qquad (6)$$




Kuva 15. Tarkkuuden ja herkkyyden suhde Venn-diagrammina (mukailtu Alpaydin 2021, 76)

### 6.7.4 Sekaannusmatriisi

Sekaannusmatriisi on taulukko, joka havainnollistaa graafisesti, kuinka syväoppismalli oppi tunnistamaan eri luokat suhteessa annettuun validointidataan. Taulukon riveillä kuvataan todelliset luokat, jotka tiedetään validointidatasta. Taulukon sarakkeissa kuvataan ennustetut luokat. (Microsoft 2023.)

Matriisin diagonaalisoluista nähdään, kuinka hyvin todellisuus vastaa ennustusta, eli kuinka monta todellista luokkaa ennustettiin oikein. Muista soluista nähdään mahdolliset sekaannukset, jos tietyn luokan ilmentymistä osa on luokiteltu virheellisesti toiseen luokkaan. Taulukossa 3 nähdään sekaannusmatriisi käytännössä. (Microsoft 2023).

| Todellinen / Ennustettu | Ajoissa | Myöhässä | Todella myöhässä |
|---|---|---|---|
| Ajoissa | 40 | 7 | 3 |
| Myöhässä | 5 | 25 | 5 |
| Todella myöhässä | 3 | 5 | 8 |

Taulukko 3. Sekaannusmatriisi (mukailtu Microsoft 2023)

# 7 Ilmakuvien valinta siirtolohkareista

## 7.1 Siirtolohkareiden valinta

Työhön valittiin dataa siirtolohkareista, jotka pystyttiin silmämääräisesti tunnistamaan ilmakuvista ja jotka täyttivät työssä vaadittavan kokoluokittelun. Siirtolohkareita valittiin 33 kappaletta, ja niistä tallennettiin taulukossa 1 näkyviä tietoja. Työnimi ja sijaintikoordinaatit eivät ole oleellisia tietoja syväoppimismallin oppimisen kannalta. Ne kuitenkin helpottivat datan keräämistä ja sen jäsentämistä huomattavasti.

| Siirtolohkareen työnimi | Siirtolohkareen sijaintikoordinaatit (DD WGS84) | Siirtolohkareen korkeus (m) | Siirtolohkareen kokoluokka |
|---|---|---|---|
| Isokivi | 62.990167, 22.323750 | 3-4 | Normaali |
| Riuttalohkon kivi | 60.852210, 25.371979 | 5-6 | Suuri |
| Kukkarokivi | 60.429094, 22.088980 | 12-17 | Todella suuri |

Taulukko 1. Siirtolohkareiden opetusdataa

## 7.2 Ilmakuvadatan esikäsittely

Ilmakuvat rajattiin ja tallennettiin Google Earth Pro -ohjelmalla. Ohjelmaa ajettiin paikallisella tutkimuskoneella. Ohjelman asetuksista kytkettiin pois kuvan jälkikäsittelytoiminnot kuten pakkaus, reunanpehmennys ja anisotrooppinen suodatus, jotta ne eivät vaikuttaneet opetusdatan laatuun.

Ohjelman avulla navigoitiin siirtolohkareen sijaintikoordinaattiin. Tämän jälkeen varmistettiin, että kallistusta ei ole, ja zoomattiin mittakaavaan 10 metriä. Tällä mittakaavan valinnalla varmistettiin, että kaikki ilmakuvat otetaan suurin piirtein samalla tarkkuudella. Kuvia olisi voitu ottaa myös eri mittakaavoissa ja eri tarkkuuksilla, jotta malli olisi oppinut yleistämään paremmin mittakaavan suhteen, mutta se päätettiin siirtää jatkokehitykseen.

Google Earthin historialliset kuvat toiminnolla sekä kompassisuuntaa vaihtamalla yhdestä siirtolohkareesta saatiin tallennettua monta erinäköistä kuvaa. Tällä saatiin kasvatettua datan määrää, mikä auttoi syväoppimismallia yleistämään paremmin. Yhteensä ilmakuvia tallennettiin 80 kappaletta. Kuvat tallennettiin ohjelman Tallenna kuva -toiminnolla 1024x768 resoluutiossa. Jotta mahdollistettiin helppo jatkokäsittely, kuvan nimi annettiin työnimen, sijainnin ja kokoluokan mukaisesti, esimerkiksi:

```
kukkarokivi_60.429094_22.088980_todellasuuri.png
```

# 8 Luokitteludatan esikäsittely

## 8.1 Annotointi ja sen eri formaatit

Annotoinnin tarkoituksena on merkitä lähtöaineiston kuvaan niin että siitä käy esille minkä luokan ilmentymiä kuvassa on, missä kohtaa ne sijaitsevat ja minkä kokoinen osa kuvasta esittää kyseessä olevan luokan ilmentymää. Tämä prosessin osa on esiehto syväoppimismallin opettamiselle. Tarkasti tehty annotointi parantaa mallin kykyä erotella ja paikantaa eri luokkien ilmentymiä kuvissa. (Ultralytics 2024.)

YOLOv8 malli käyttää TXT-formaattia, missä rajauslaatikon keskipiste sekä leveys ja korkeus tallennetaan suhteellisina lukuina suhteessa lähdekuvan korkeuteen ja leveyteen (Ultralytics 2024). Näin ollen kaikki arvot skaalataan välille 0-1. Ensimmäinen luku kertoo, että rajauslaatikon sisällä on luokkaan 0 (ensimmäinen luokka) kuuluva objekti:

```
0 0.489583 0.511806 0.328646 0.533796
```

Faster R-CNN ja useimmat muut objektintunnistusmallit käyttävät eri annotointiformaattia. Ne yleensä odottavat joko pascal VOC:ia, joka käyttää XML-tiedostoja tai COCO:a, joka käyttää JSON-tiedostoformaattia. Nämä formaatit eivät käytä suhteellisia keskipisteitä vaan pikselikoordinaatteja kuvaamaan objektien rajauslaatikot kuva-alueella.

## 8.2 Annotointi käytännössä

Erilaisia annotointiohjelmia on saatavilla useita ja yksi niistä on avoimen lähdekoodin sovellus LabelImg. Se mahdollistaa objektien rajaamisen ja luokittelemisen helposti graafisessa käyttöliittymässä. LabelImg asennettiin tutkimuskoneelle Pythonin 3.10 versiossa komennolla:

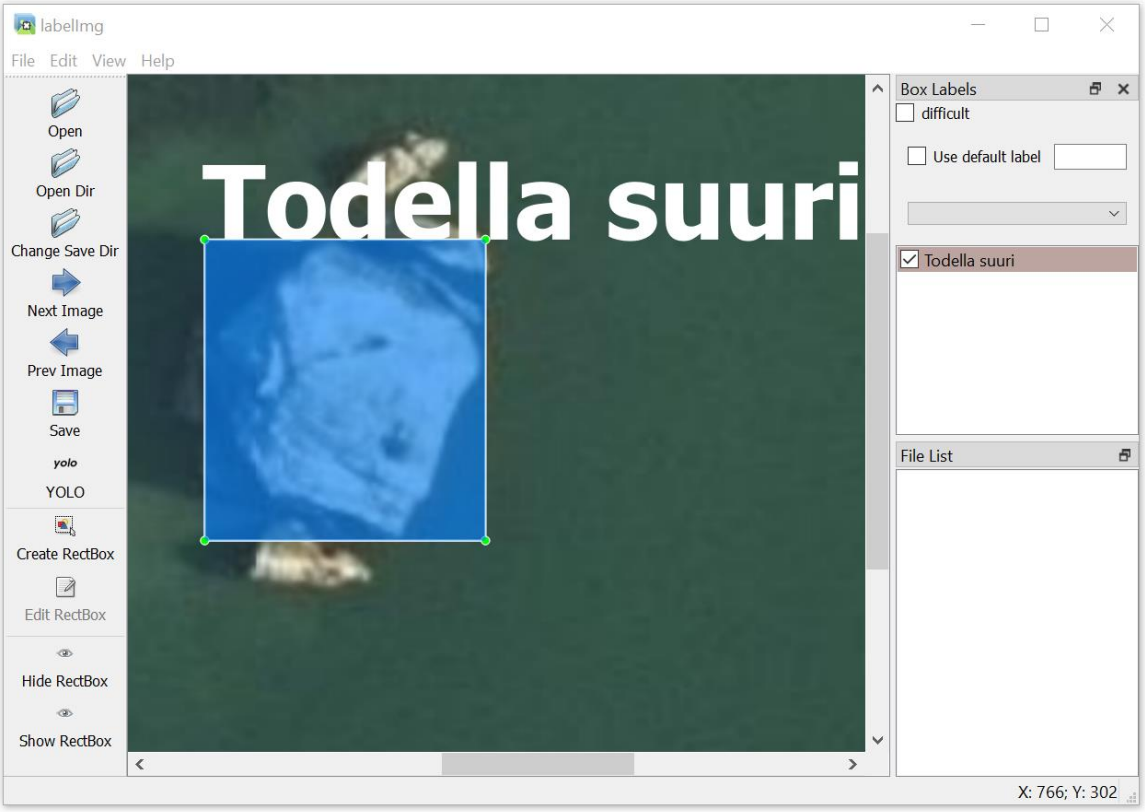
```
pip install labelImg
```

Annotointiohjelman käyttöönoton jälkeen ilmakuvat annotoitiin yksi kerrallaan manuaalisesti. Yhteen ilmakuvaan merkittiin vain yksi siirtolohkare. Jokaista ilmakuvaa ja siinä tunnistettua siirtolohkaretta vastaa yksi txt-tiedosto, esimerkiksi:

```
kukkarokivi_60.429094_22.088980_todellasuuri.txt
```

Kuvassa 16 nähdään annotointi käytännössä. Ilmakuvassa näkyvän siirtolohkareen päälle piirrettiin rajauslaatikko, niin että koko siirtolohkare on laatikon sisällä. Tämän jälkeen merkittiin, minkä luokan ilmentymää se edustaa. Annotointi oli yksi työn haastavimmista vaiheista. Vaati useita yrityksiä, jotta siirtolohkare ja erityisesti sen reunat saatiin sopimaan rajauslaatikon sisälle. Kokonaistyöajasta toteutuksen osalta annotointiin yhdessä

ilmakuvien valintaan kului arviolta noin puolet. Annotointiin liittyy näin ollen merkittävä inhimillisen virheen mahdollisuus, joka voi vaikuttaa koko mallin oppimiseen ja lopullisiin tuloksiin.




Kuva 16. Annotointi

## 8.3 Ristiinvarmistus ja kerrostaminen

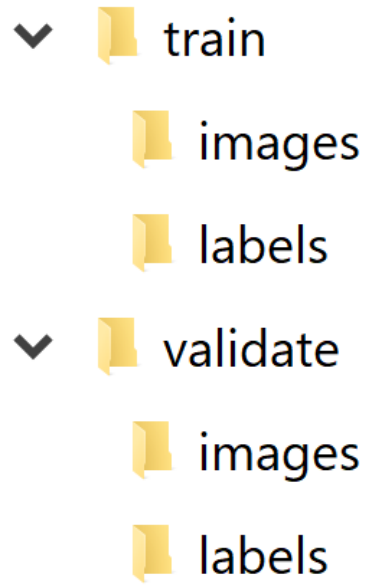
Kun annotointitiedot oli tallennettu, ne jaettiin kahteen osaan opetusta varten. Koska dataa oli vähän, tehtiin päätös ja noin 80 % sijoitettiin train-kansioon ja niitä käytettiin mallin kouluttamiseen. Validate-kansioon siirrettiin noin 20 %, ne, joita YOLOv8 käyttää mallin validointiin. Tämä jakaminen tehtiin Pythonin shuffle-moduulilla, jotta jakamisesta saatiin satunnainen.

Lisäksi jakaminen tehtiin kerrostetusti, niin että 80-20 % periaate pätee jokaisen luokan sisällä. Kokonaisluvuista johtuen vaihteluväli liikkui 67-79 % opetusdatasetin osalta ja 21-33 % validointidatasetin osalta. Tällä varmistettiin, ettei opetus- ja validointivaiheessa synny luokkakohtaisia vinoumia datan jakamisesta johtuen. Taulukossa 2 on esitetty nämä luokkakohtaiset lukumäärät, josta nähdään että normaali-luokka on edustettuna paremmin, jolloin malli todennäköisesti myös oppii sen piirteet paremmin.

| Luokka | Yhteensä kpl | Opetusdata kpl | Opetusdata % | Validointidata kpl | Validointidata % |
|---|---|---|---|---|---|
| Normaali | 57 | 45 | 79 % | 12 | 21 % |
| Suuri | 17 | 13 | 76 % | 4 | 24 % |
| Todellasuuri | 6 | 4 | 67 % | 2 | 33 % |
| Kaikki | 80 | 62 | 78 % | 18 | 23 % |

Taulukko 2. Luokkakohtaiset lukumäärät

Kuvassa 17 esitetään YOLOv8:n opetukseen käytetty kansiorakenne. Se on selkeä ja johdonmukainen, ja käytettäviä datasettejä voidaan helposti vaihtaa kansioita siirtämällä tai kopioimalla. Myös erillinen test-kansio voidaan tarvittaessa määrittää kuville, joilla halutaan testata mallin suorituskyky, kun varsinainen opetusprosessi on päättynyt.




Kuva 17. YOLOv8 mallin kansiorakenne

## 8.4 Data.yaml-tiedosto

YOLOv8 mallin opetusdatan konfiguraatio määritellään data.yaml-tiedostossa. Siinä kerrotaan missä sijainnissa on opetukseen ja validointiin käytettävät kuvat sekä montako luokkaa mallissa on ja mitkä ovat niiden nimet. Kuvassa 18 nähdään kolmen luokan mallin määrittely ja niitä vastaavat suhteelliset polut, joissa lähtöaineisto sijaitsee.

data.yaml

```
1  train: ../train/images
2  val: ../validate/images
3  nc: 3
4  names: ['normaali', 'suuri', 'todellasuuri']
```

Kuva 18. Data.yaml-tiedosto

# 9 Syväoppimismallin opettaminen

## 9.1 Koneoppimisalustan asentaminen

Opetusympäristö asennettiin Windows 10 koneelle, jossa oli käytössä NVIDIA GeForce RTX 3050 näytönohjain. Se sisältää 8 Gt GDDR6 muistia ja 2560 kappaletta CUDA-suoritinydintä. Koneelle asennettiin CUDA käytön mahdollistavat NVIDIAn CUDA Toolkit 11.8 ja cuDNN 9.1.

Koneoppimisalustan asentamisessa tärkeää oli varmistaa, että eri Python kirjastojen versiot ovat toistensa kanssa yhteensopivia. Lisäksi Python version tuli olla välillä 3.9–3.12. CUDA:n laskentatehoa hyödyntävä PyTorch sekä YOLOv8 asennettiin komennoilla:

```
pip install torchvision==0.17.1 torchaudio==2.2.1 torch==2.2.1 --index-url https://download.pytorch.org/whl/cu118

pip install ultralytics==8.2.45
```

## 9.2 Syväoppimismallin opettaminen ja parametrien säätäminen

Opetus käynnistyi sillä, että YOLOv8 latasi valmiiksi koulutetun nano-mallin (pretrained). Tällä saatiin nopeutettua opetusprosessia koska valmis malli sisältää paljon jo opittuja piirteitä. Opetuskertoja ajettiin useita, jotta löydettiin parhaiten toimivat hyperparametrit, esimerkiksi opetuskierrosten määrää piti kasvattaa arvosta 100 arvoon 1000. Erilaisia optimoijia testattiin, ja huomattiin että AdamW tuotti huonomman tuloksen $F_1$-arvotuksen osalta kuin SGD. Seed-parametrilla varmistettiin, että jokainen ajokerta oli toistettavissa. Mallin opettaminen käynnistettiin komennolla:

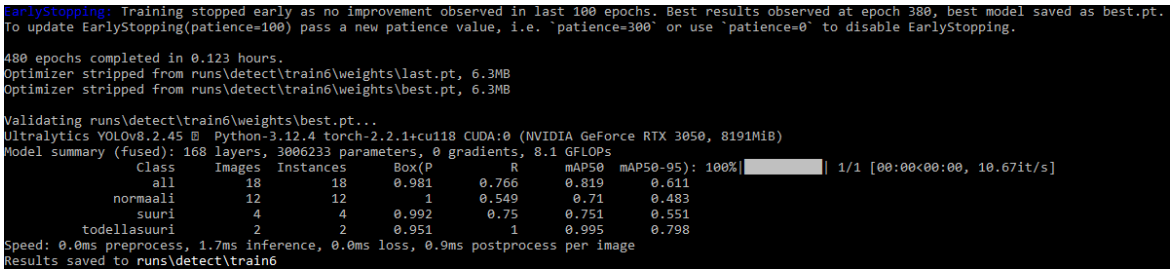
```
yolo task=detect data=data.yaml mode=train pretrained=true batch=-1 epochs=1000 patience=100 imgsz=640 rect=false lr0=0.01 optimizer=SGD verbose=true device=0 workers=1 cache=true amp=true seed=141283
```

Näytönohjaimen (device-muuttuja=0) muistin määrä koneella rajoitti eniten sitä, kuinka nopeasti opetusprosessi valmistui. Käyttämällä -1 eli automaattista laskentaa eräkoolle (batch), 1000 opetuskierrosten lukumäärälle (epochs), 640 mallin kuvakoolle (imgsz), false kuvasuhteen käsittelylle (rect), 0.01 oppimisnopeudelle (lr0) ja SGD optimoijalle (optimizer) opetus valmistui noin kahdeksassa minuutissa. Opetus siis valmistui huomattavan nopeasti, mutta toisaalta dataakin oli käytettävissä vähän.

Kaikki opetuksessa käytetyt hyperparametrit tallentuivat tiedostoon args.yaml. Ne mitä ei erikseen asetettu opetusprosessin käynnistysvaiheessa, saivat oletusarvot YOLOv8:n määrittelystä. Args.yaml-tiedostoon tallentui yhteensä 106 parametria, ja tätä dokumenttia voitiin myöhemmin tarkastella suhteessa opetuksen tuloksiin.

Mallin suorituskykyä arvioitiin säännöllisesti oppimisprosessin aikana, ja tarkat statistiikat tallennettiin (verbose). Kuvassa 19 näkyy oppimisprosessin statistiikkaa, jossa tärkeimmät indikaattorit mallin oppimisesta ovat tarkkuus (P) ja herkkyys (R). Paras tulos saavutettiin koulutuskierroksen 380 kohdalla, ja koska patience-muuttuja oli asetettu arvoon 100, päättyi suoritus kierrokseen 480. Yhteenveto mallista suhteessa validointidatasettiin oli seuraava:

- Normaaliluokan tarkkuus 100,0 % ja herkkyys 54,9 %
- Suuri-luokan tarkkuus 99,2 % ja herkkyys 75,0 %
- Todella suuri -luokan tarkkuus 95,1 % ja herkkyys 100,0 %.



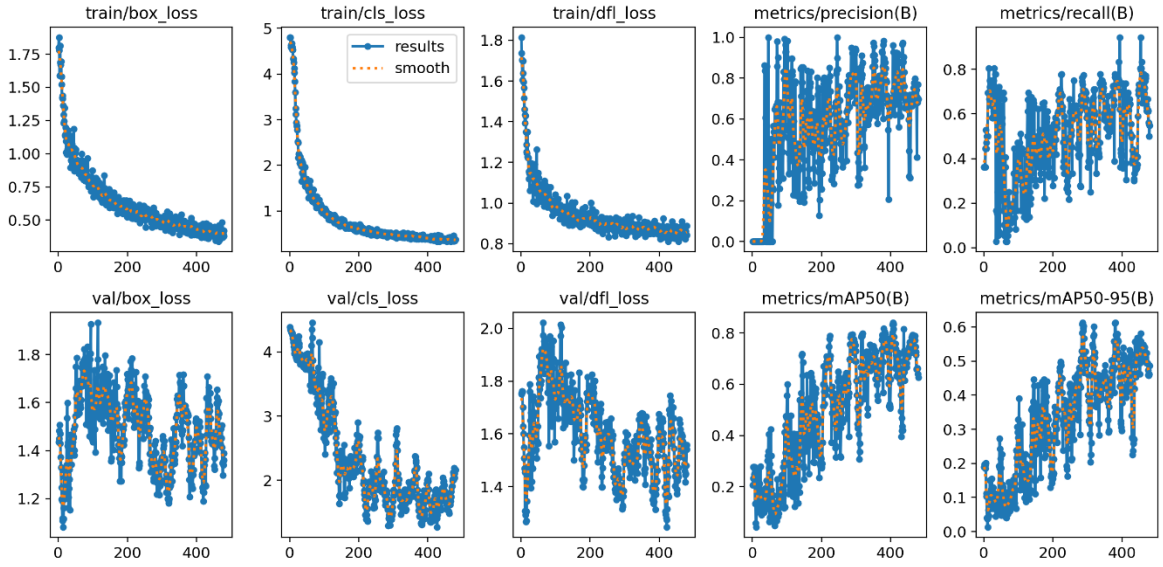
Kuva 19. Syväoppimismallin oppimisprosessi

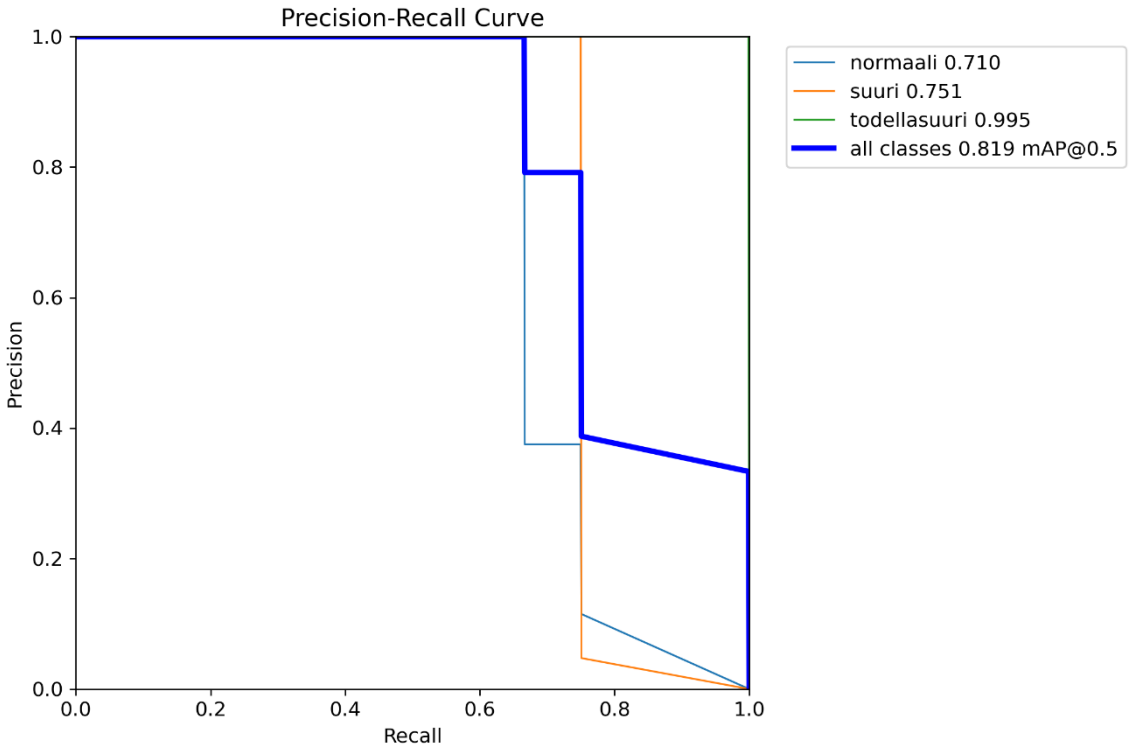
## 9.3 Opetuksen tulokset ja niiden arviointi

Oppimisprosessin jälkeen statistiikat analysoitiin, jotka YOLOv8 tuotti automaattisesti. Sekä tarkkuus että herkkyys kasvoivat opetuskierrosten edetessä eli malli oppi tunnistamaan oikeita luokan ilmentymiä kuvista. Kuvaajista huomattiin, että malli oppi opetusdatan piirteet nopeasti, mutta validointidatan kohdalla suorituskyky ei ollut yhtä hyvä, mikä voi viitata mahdolliseen ylisovittamiseen.

**Virhearvot**

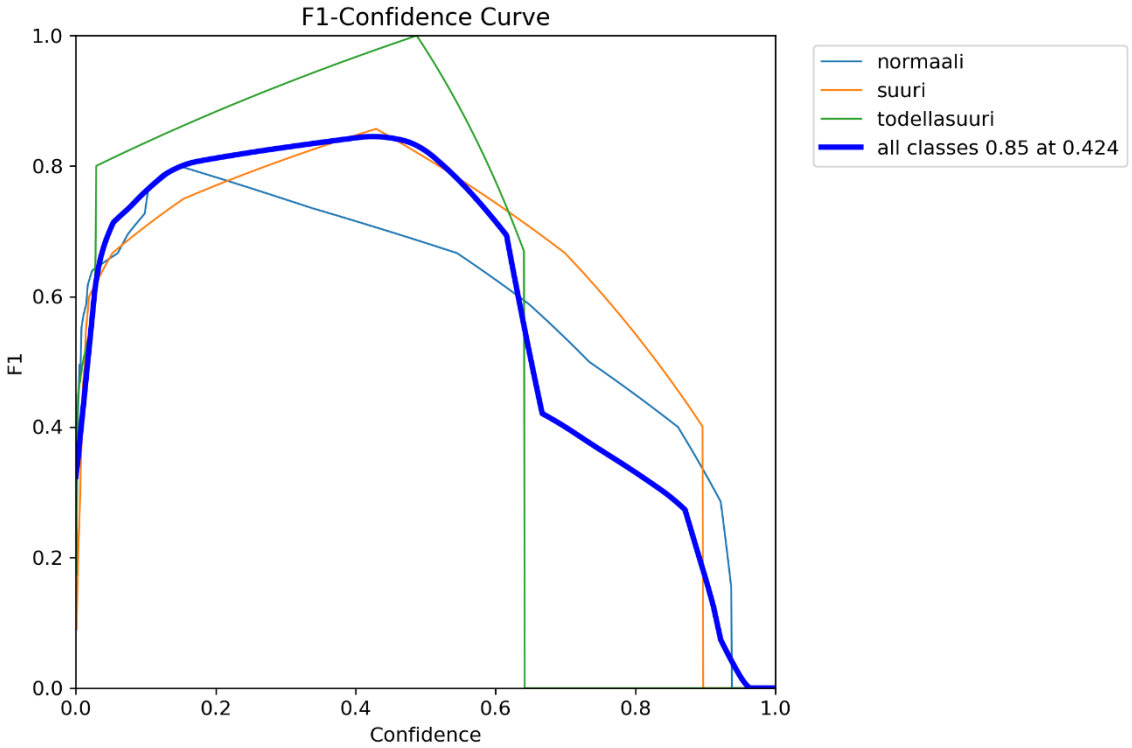
Erilaiset virhearvot (loss) kuten box_loss, cls_loss ja dfl_loss pienentyivät eli malli oppi datasta opetuskierrosten edetessä. Mean Average Precision -mittarit kasvoivat opetuskierrosten edetessä eli malli oppi tunnistamaan objektit oikeassa sijainnissa. mAP@0.5-mittari saavutti arvon 81,9 %.



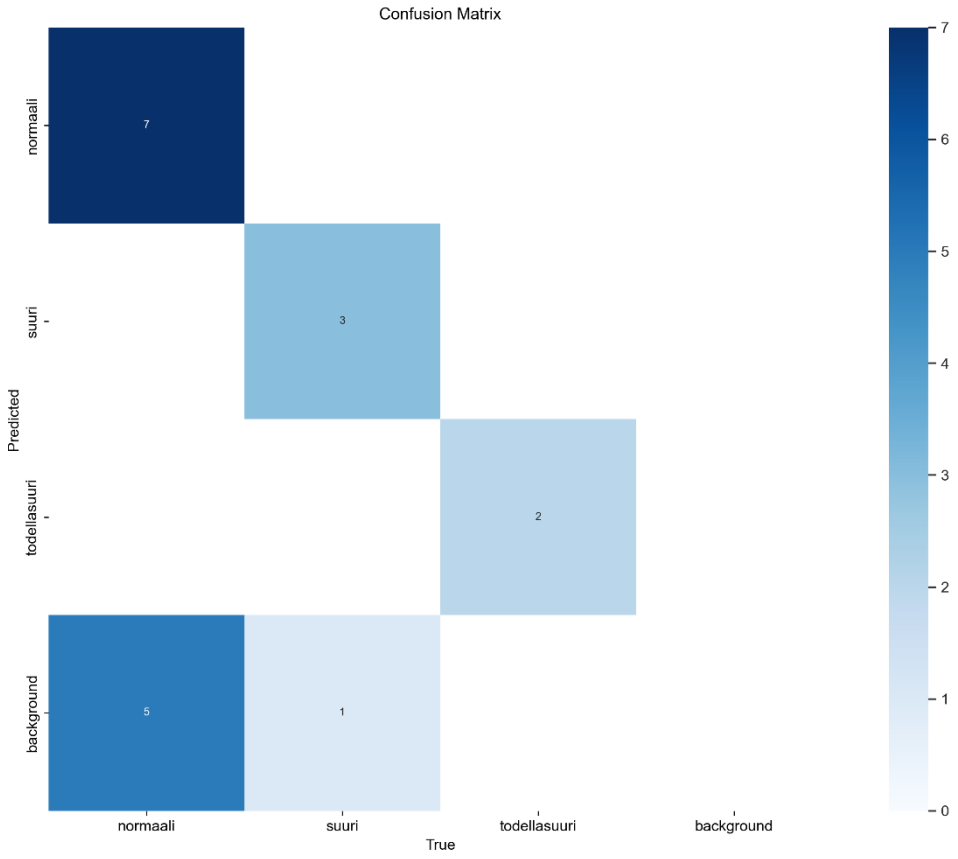

### $F_1$-arvotus

$F_1$ saavutti maksimiarvon 85 % kun ennusteen luottamusarvo oli 42,4 %. Tässä kohtaa mallin ennusteet olivat tasapainossa tarkkuuden ja herkkyyden suhteen. Yleisesti voidaan sanoa, että malli oppi kohtalaisen hyvin, ottaen huomioon määrällisesti pienen lähtöaineiston. Vaikka datamäärä oli pieni, mallin suorituskykyä paransi se, että opetuksen pohjaksi otettiin YOLOv8:n esiopetettu nano-malli joka on jo oppinut hyvin tunnistamaan reunoja ja ääriviivoja.




**Sekaannusmatriisi**

Sekaannusmatriisi osoitti että normaali-luokan osalta 7 ilmakuvaa 12 ilmakuvasta luokiteltiin validointivaiheessa oikein. Toisin sanoen 42 % luokiteltiin background-luokkaan eli virheen määrä oli lopullisessa mallissa vielä suuri. Background-luokka YOLOv8:ssa edustaa kuvaa, jossa ei ole minkään luokan ilmentymää.


Confusion Matrix
normaali
suuri
todellasuuri
background
Predicted
7
3
2
5
1
normaali
suuri
todellasuuri
background
True
7
6
5
4
3
2
1
0

# 10 Syväoppimismallin tuottaman ennusteen visualisointi

## 10.1 YOLOv8 mallin exportointi ONNX-formaattiin

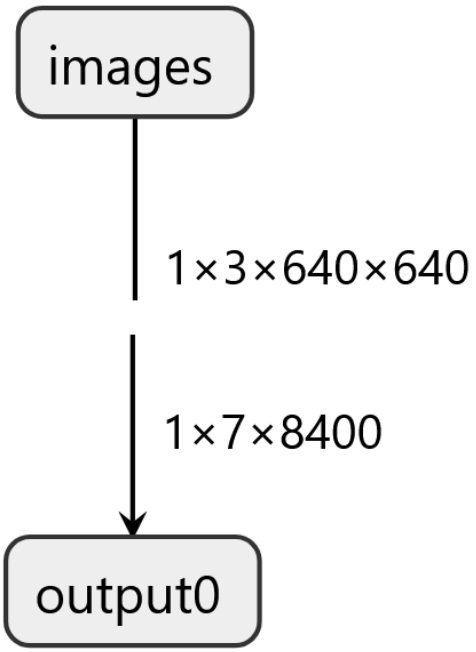
Syväoppimismallia haluttiin hyödyntää käyttäen ONNX Runtime -kirjastoa, tarvittaessa myös muissa koneoppimisympäristöissä. Tämän takia YOLOv8:n generoima malli muunnettiin ONNX-formaattiin. Muunnos suoritettiin komennolla:

```
yolo export model=best.pt format=onnx
```

Mallia tarkasteltiin tämän jälkeen Netron.app-verkkosivuston avulla, jotta voitiin todentaa, että muunnetun mallin rakenne oli oikea. Erityisesti tarkastettiin mallin input- ja output-tensorien muoto, jotka esitetään kuvasta 20. Todettiin että ne ovat oikein, eli muunnos onnistui.

Input-tensorin muoto on 1x3x640x640, joka tarkoittaa 4-uloitteista liukulukujen taulukkoa. Taulukon ensimmäinen dimensio tarkoittaa erien lukumäärää, mikä on tässä tapauksessa yksi koska mallille syötetään vain yksi kuva kerrallaan. Toinen dimensio ilmaisee väriarvojen (RGB) määrää, ja kaksi viimeistä dimensiota kuvapisteiden määrää.

Output-tensorin muoto 1x7x8400 tarkoittaa 3-ulotteista liukulukujen taulukkoa. Taulukon toinen dimensio sisältää ennusteiden attribuutit eli rajauslaatikon sijainnin, pituuden, korkeuden ja luokkatodennäköisyydet. Kolmas dimensio kuvaa ennusteiden lukumäärää.

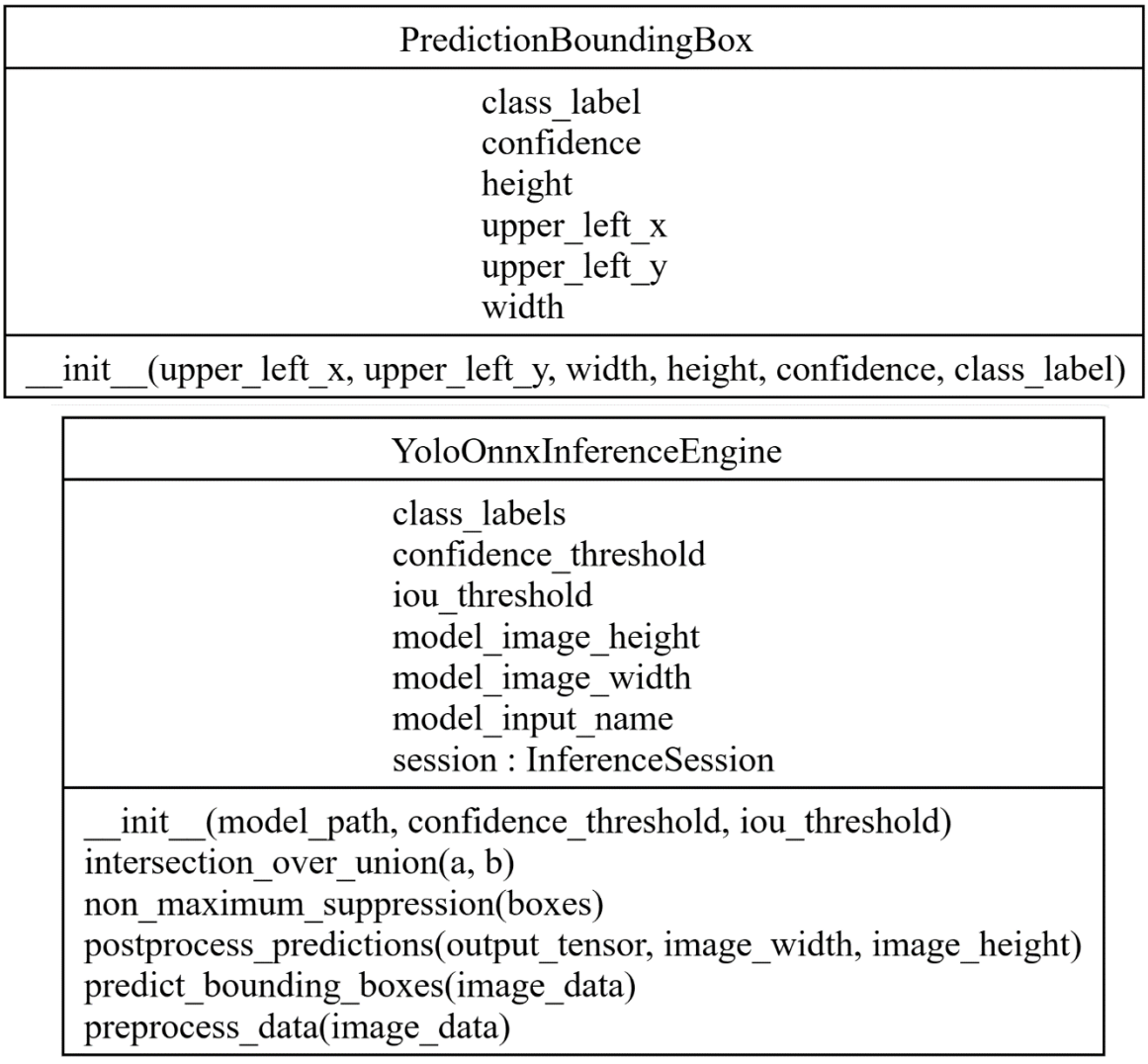



Kuva 20. Netron.app luoma kuvaus mallin input- ja output-tensoreista

## 10.2 Syväoppimismallin hyödyntämiskerros

Syväoppimismallin hyödyntämiseksi rakennettiin YoloOnnxInferenceEngine-luokka. Sen avulla voidaan suorittaa ennusteita antamalla sille syötteenä kuvia. Lisäksi rakennettiin apuluokka PredictionBoundingBox, jonka avulla tallennetaan tunnistetut rajauslaatikot. Luokat rakennettiin Visual Studio Codella käyttäen Python-ohjelmointikieltä.

Luokat sisältävät kuvassa 21 esitetyt funktiot ja toiminnallisuudet. Session on viittaus onnxruntime-kirjaston InferenceSession-funktion palauttamaan olioon, joka sisältää ladatun syväoppimismallin tiedot. Confidence- ja IoU-threshold parametrien avulla voidaan hallita ennusteiden rajaamista halutulle luottamusvälille.




Kuva 21. Hyödyntämiskerroksen luokkakaaviot

### 10.2.1 Input-tensorin luominen lähdekuvasta

Funktion preprocess_data avulla käsiteltiin syötteenä annettu kuvadata. Funktion avulla värit muunnettiin syväoppimismallin vaatimaan RGB-bittimuotoon, väriarvot skaalattiin välille 0-1, data skaalattiin mallin vaatimaan kokoon ja kanavat asetettiin oikeaan järjestykseen. Kuvassa 22 esitetään nämä tärkeät vaiheet datan esikäsittelyssä, joiden toteutus oli suoraviivaista käyttäen Pythonin OpenCV- ja NumPy-kirjastoja.

```
def preprocess_data(self, image_data):
    data = cv2.cvtColor(image_data, cv2.COLOR_BGRA2RGB).astype(numpy.float32) / 255.0
    data = cv2.resize(data, dsize=(self.model_image_width, self.model_image_height))
    return numpy.expand_dims(numpy.transpose(data, [2, 0, 1]), axis=0)
```

Kuva 22. Lähdedatan esikäsittely

### 10.2.2 Output-tensorin jälkikäsittely

Kun input-tensori oli valmisteltu ja ajettu syväoppimismallin läpi, saatiin tulosjoukkona output-tensori, joka vaati vielä jälkikäsittelyä, jotta rajauslaatikkojen tiedot voitiin tallentaa apuluokkaa hyödyntäen. Output-tensori sisältää tiedot jokaisesta rajauslaatikosta, jotka syväoppimismalli on ennustanut. Se on liukulukuja sisältävä taulukoiden taulukko, jonka koko voidaan esittää kuten kaavassa 7.

$$(4 + \text{mallin luokkien määrä}) \times \text{ennustettujen rajauslaatikkojen määrä} \qquad (7)$$

Luku neljä on vakio ja sisältää rajauslaatikon keskipisteen koordinaatit sekä sen leveyden ja korkeuden. Mallin luokkien määrä on tässä toteutuksessa kolme: normaali, suuri ja todella suuri siirtolohkare. Liukuluvut on tallennettu taulukon ensimmäiseen dimensioon seuraavassa järjestyksessä:

- keskipisteen x-koordinaatti
- keskipisteen y-koordinaatti
- rajauslaatikon leveys
- rajauslaatikon korkeus
- luokan 1 todennäköisyys
- luokan 2 todennäköisyys
- luokan 3 todennäköisyys.

Edellä mainituista määrityksistä saatiin johdettua kuvassa 23 esitetty funktio postprocess_predictions, joka toteutti halutun toiminnallisuuden. Lisäksi funktio skaalasi ennustetut rajauslaatikot oikein suhteessa alkuperäiseen kuvaan, sekä suodatti pois vähemmän todennäköiset ennustukset, confidence_threshold-muuttujan mukaisesti. Jos

ennustus sisälsi esimerkiksi arvojoukon normaali 60 %, suuri 5 %, todella suuri 3 % ja confidence_threshold oli 50 %, tällöin ainoastaan normaali 60 % ennustus lisättiin rajauslaatikkojen kokoelmaan. Tämä esikarsinta tehtiin, jotta vain oleellisesta tiedosta tallennettiin ennustusrajauslaatikkoja seuraavia vaiheita varten.

```
def postprocess_predictions(self, output_tensor, image_width, image_height):
    num_predictions = output_tensor.shape[1]
    x_scale = image_width / self.model_image_width
    y_scale = image_height / self.model_image_height
    boxes = []
    for i in range(num_predictions):
        class_confidences = output_tensor[4:, i]
        max_confidence = numpy.max(class_confidences)
        if max_confidence < self.confidence_threshold:
            continue
        boxes.append(PredictionBoundingBox(
            (output_tensor[0, i] - output_tensor[2, i] / 2) * x_scale,
            (output_tensor[1, i] - output_tensor[3, i] / 2) * y_scale,
            output_tensor[2, i] * x_scale,
            output_tensor[3, i] * y_scale,
            max_confidence, self.class_labels[numpy.argmax(class_confidences)]))
    return self.non_maximum_suppression(boxes)
```

Kuva 23. Postprocess_predictions-funktion logiikka

### 10.2.3 Rajauslaatikkojen jälkikäsittely

Kun rajauslaatikot oli saatu tallennettua listaan, joka sisältää PredictionBoundingBox-apuluokan ilmentymiä, niitä voitiin jälkikäsitellä. Tavoitteena oli piilottaa päällekkäin menevät rajauslaatikot, jotta visualisointi olisi mahdollisimman selkeä ja ymmärrettävä. Päällekkäisistä rajauslaatikoista poistettiin vähemmän todennäköinen ilmentymä, jos niiden välinen IoU-% ylitti annetun arvon iou_threshold.

NMS-algoritmin toteutus Pythonilla oli suoraviivaista kuten kuvasta 24 nähdään. Pythonin list comprehension -menetelmällä saatiin yhdellä koodirivillä muodostettua uusi lista rajauslaatikoista, jotka täyttävät annetut ehdot. Lopputuloksena jäljelle jäi vain ne rajauslaatikot, jotka olivat parhaiten tunnistettuja, ja joiden päällekkäisyys muiden tunnistettujen rajauslaatikkojen kanssa oli minimoitu.

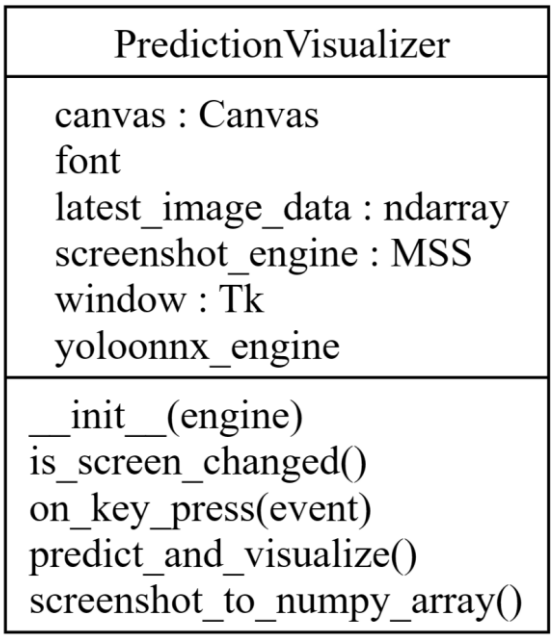
```python
def non_maximum_suppression(self, boxes):
    boxes.sort(key=lambda box: box.confidence, reverse=True)
    filtered_boxes = []
    while boxes:
        best_box = boxes.pop(0)
        filtered_boxes.append(best_box)
        boxes = [
            box for box in boxes
            if self.intersection_over_union(best_box, box) < self.iou_threshold
        ]
    return filtered_boxes
```

Kuva 24. NMS-algoritmin toteutus

## 10.3 Automaattinen tunnistaminen ja visualisointikerros

Siirtolohkareiden tunnistamisen visualisointia varten toteutettiin PredictionVisualizer-luokka. Se hyödyntää edellä kuvattua syväoppimismallin hyödyntämiskerrosta. Pythonin Tkinter-, MSS- ja cv2-kirjastoja käytettiin käyttöliittymän rakentamisessa ja ruutukaappauksen käsittelyssä.

Toimintalogiikka rakennettiin niin että jos näytön sisältö muuttuu, ruutukaappaus ajetaan syväoppimismallin läpi, ja saadaan tunnistetut rajauslaatikot. Rajauslaatikot esitettiin ruudulla laatikoina, joihin kirjoitettiin tunnistuksen todennäköisyys ja tunnistetun luokan nimi. Kuvassa 25 esitetään visualisointikerroksen luokkakaavio. Yoloonnx_engine on viittaus syväoppimismallin hyödyntämiskerrokseen.




Kuva 25. Visualisointikerroksen luokkakaavio

### 10.3.1 Kuvaruutukaappauksen ottaminen syötteeksi syväoppimismallille

Automaattinen tunnistaminen toimii parhaiten tilanteissa, joissa syötteenä annettu kuva vastaa leveydeltään ja korkeudeltaan mahdollisimman paljon syväoppimismallin dimensioita. Jos tunnistettava kohde on pieni ja lähdekuva suuri, tunnistamisen todennäköisyys pienenee. Työpöytäsovelluksen käyttäjän yleisin ruutukoko on 1920x1080 globaalien tilastojen mukaan, kuten kuvassa 26 nähdään.


Kuva 26. Ruudun resoluutiokoko (Statcounter GlobalStats)

Syväoppimismalli opetettiin input-tensorin koolla 640x640 pikseliä. Näin ollen lähdekuva on 3.0 kertaa suurempi leveyssuunnassa ja 1.6875 kertaa suurempi korkeussuunnassa. Tämä voi johtaa epätarkkuuteen tunnistuksessa ja toimia teknisenä rajoitteena.

Yksi ratkaisu tämän teknisen rajoitteen poistamiseksi olisi toteuttaa kuvaruutukaappauksen pilkkominen 640x640 kokoisiin ruutuihin. Pilkkomisessa pitäisi kuitenkin huomioida, ettei tunnistettavat kohteet katkeaisi ruutujen keskeltä. Tällaisen ratkaisun toteutus rajattiin sen tuoman lisäkompleksisuuden takia tämän opinnäytetyön ulkopuolelle.

Is_screen_changed-funktion avulla hallittiin, että kuvaruutukaappaus, ennusteet ja visualisoinnit päivitetään vain silloin kun näytön sisältö oli oleellisesti muuttunut. Raja-arvona käytettiin 160x160 pikselin muutosta, pois lukien rajauslaatikot, jotka maskattiin pois laskennasta. Näin saatiin toteutettua visualisointi, joka ei jatkuvasti vilkkunut eikä syöttänyt syväoppimismallille kuvaruutukaappausta, jossa olisi mukana myös rajauslaatikkojen grafiikka.

### 10.3.2 Rajauslaatikkojen esittäminen ruudulla

Kun rajauslaatikot olivat valmiit, ne esitetään ruudulla käyttäen indikaattoreita. Indikaattoreiden määrää ei rajattu. Jokainen indikaattori sisälsi rajauslaatikon, ennusteen luotettavuus %:n sekä ennustetun luokan nimen.

Indikaattorit päivitettiin kuvassa 27 näkyvällä tavalla. Jokaista laatikkoa kohden laskettiin vasemman yläkulman sekä oikean alakulman koordinaatit. Tähän kohtaan piirrettiin sininen suorakulmio, jonka sisälle kirjoitettiin valkoisella tekstillä luokan todennäköisyys prosentteina sekä luokan nimi.

```
for box in boxes:
    x0 = box.upper_left_x
    y0 = box.upper_left_y
    x1 = x0 + box.width
    y1 = y0 + box.height
    label = f"{box.confidence * 100:.0f}% {box.class_label}"
    self.canvas.create_rectangle(x0, y0, x1, y1, outline="darkblue", fill="black", width=2)
    self.canvas.create_rectangle(x0, y0-20, x1, y0-2, outline="darkblue", fill="darkblue", width=1)
    self.canvas.create_text(x0, y0, fill="white", font=self.font, anchor="sw", text=label)
```

Kuva 27. Rajauslaatikkojen ja visualisointi

Toteutuksessa haluttiin rikastaa käyttäjän ruutua niin että rajauslaatikot eivät estä koneen muuta käyttöä. Tämä toiminnallisuus saavutettiin Pythonin Tkinter-kirjaston avulla, jolla luotiin läpinäkyvä ikkuna sekä kangas, jonka päälle rajauslaatikot piirrettiin. Kuvassa 28 näkyy asetukset, joilla haluttu lopputulos saavutettiin. Läpinäkyvä ikkuna asetettiin kaikkein päällimmäiseksi ja kokoruudun kokoiseksi, ja mustaa väriä käytettiin maskina läpinäkyvyydelle.

```
self.window = tkinter.Tk()
self.window.title("YoloOnnxTkinterVisualizer")
self.window.attributes('-fullscreen', True, '-topmost', True)
self.window.wm_attributes('-transparentcolor', 'black')
self.canvas = tkinter.Canvas(self.window, bg='black', highlightthickness=0, bd=0)
self.canvas.pack(fill=tkinter.BOTH, expand=True)
```

Kuva 28. Asetukset, joilla saavutettiin läpinäkyvä vaikutelma

## 10.4 Sovelluksen testaaminen ja tulokset

Sovellusta ja syväoppimismallia testattiin Windows 10 64-bit käyttöjärjestelmässä kahdella eri resoluutiolla: 3840x2160 sekä 1920x1080. Testissä selattiin Google Earthin ilmakuvia alueelta, jossa tiedettiin olevan siirtolohkareita, ja joiden ilmakuvia ei käytetty syväoppimismallin opetuksessa opetusdatassa eikä validointidatassa. Sovelluksen Python lähdekoodi käännettiin exe-muotoon ja käynnistettiin komennolla:

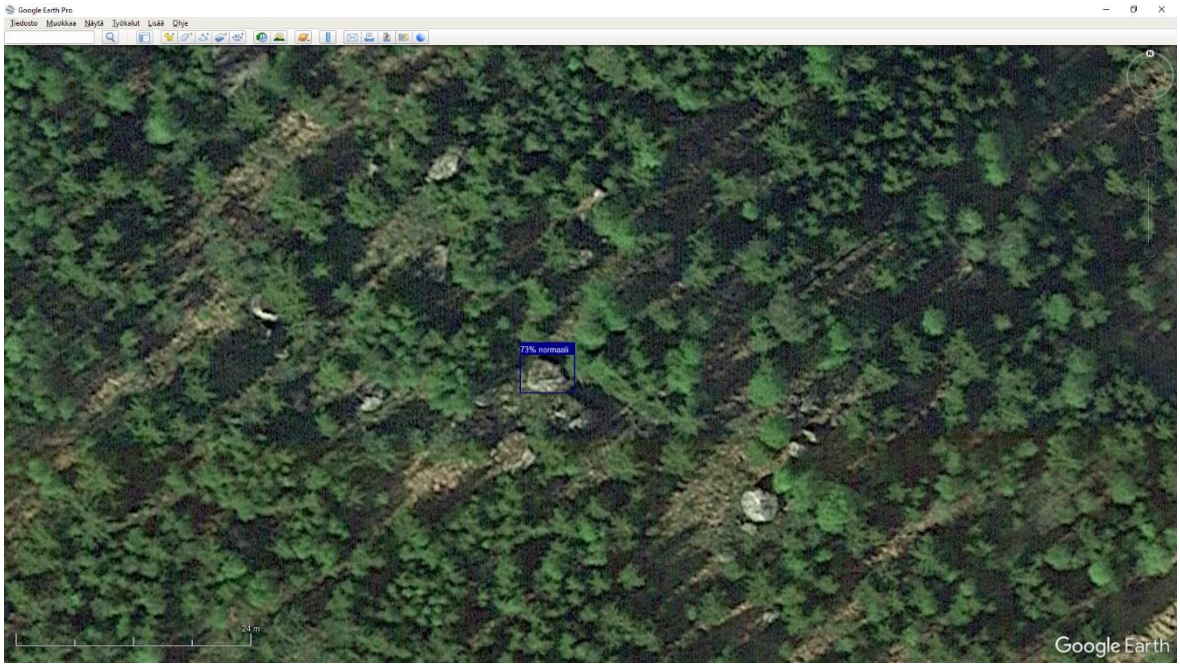
```
pyinstaller --onefile --windowed YoloOnnxTkinterVisualizer.py --
distpath=. && YoloOnnxTkinterVisualizer.exe best.onnx 0.5 0.7
```

Suuremmalla resoluutiolla selattaessa, indikaattorien päivitys kesti kauemmin, johtuen suuremmasta määrästä pikseleitä, joita sovelluslogiikka joutuu käsittelemään. Kaiken kaikkiaan prosessi kesti noin 650 millisekuntia per ruudun päivitys, josta ennustamisen ja jälkikäsittelyn osuus oli 160 millisekuntia. Pienemmällä resoluutiolla prosessi ja kuvan päivitys oli nopeampaa, noin 300 millisekuntia per ruudun päivitys, josta ennustamisen ja jälkikäsittelyn osuus oli 120 millisekuntia. Nämä suorituskykystatistiikat mitattiin hyödyntäen Pythonin time-kirjastoa. Koodia olisi voitu optimoida tehokkaammaksi, mutta se siirrettiin jatkokehitykseen.

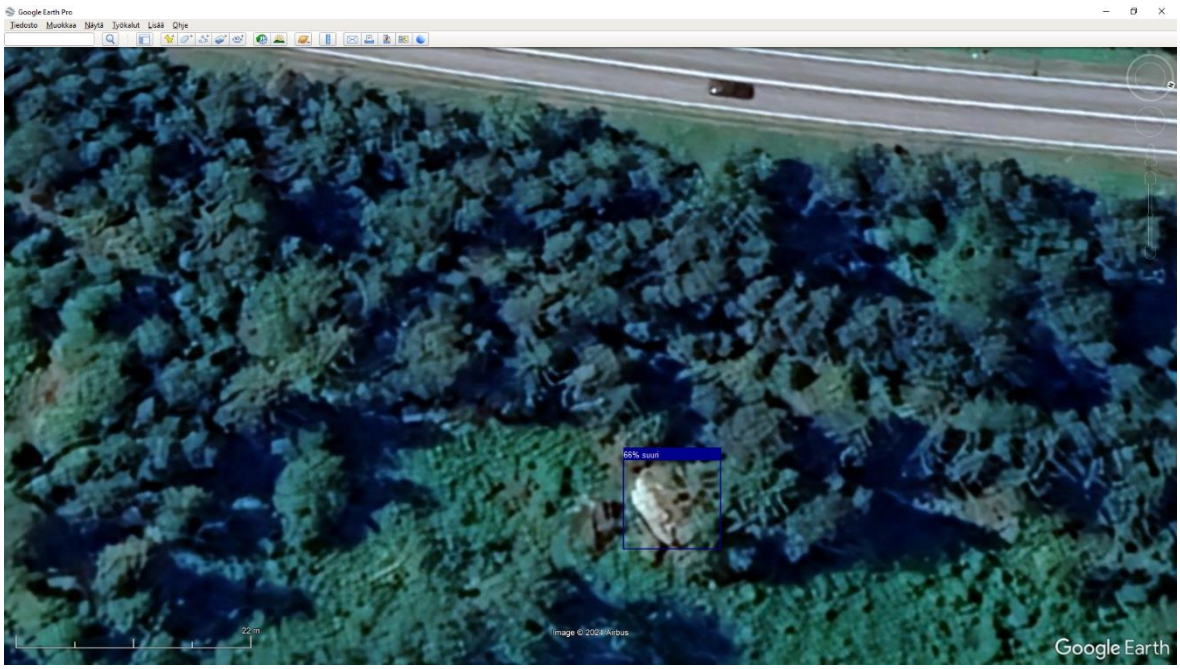
Pienemmällä resoluutiolla selattaessa, normaali-luokan siirtolohkareet tunnistettiin ja luokiteltiin oikein jo kauempaa. Suuremmalla resoluutiolla jouduttiin tarkentamaan karttaa lähemmäksi, jotta tunnistaminen tehtiin oikein. Tämä tekninen rajoite voidaan ratkaista tuomalla lisää dataa siirtolohkareista eri mittakaavoissa, jolloin malli oppii paremmin yleistämään näkemäänsä.

Yleisesti testit tuottivat positiivisen lopputuloksen: selailtaessa ilmakuvia Google Earth Pro -ohjelmalla YoloOnnxTkinterVisualizer-sovellus tunnisti ja luokitteli normaali-luokan siirtolohkareen 73 % todennäköisyydellä, ja visualisoi sen näkyville, kuten kuvasta 29 nähdään. Vaikka kuvassa näkyvällä alueella on useita muitakin siirtolohkareita, syväoppimismalli tunnisti oikein vain sen, joka todellisuudessa on riittävän korkea eli täyttää luokan määrittelyn. Kuvasta 30 nähdään oikein suureksi luokiteltu Pälkäneentien varrella sijaitseva siirtolohkare 66 % todennäköisyydellä.

Testattaessa löytyi paljon tilanteita, joissa ilmakuvassa näkyy siirtolohkare mutta malli ei osannut tunnistaa eikä luokitella niitä oikein. Lisäksi oli tilanteita, jossa siirtolohkare tunnistettiin oikein vain kartan ollessa käännettynä tiettyyn ilmansuuntaan. Nämä poikkeamat olivat odotettavia, koska opetusdatan määrä oli vain 80 ilmakuvaa.




Kuva 29. Normaali-luokan siirtolohkare tunnistettiin. Ilmakuva (Google, © 2024)




Kuva 30. Suuri-luokan siirtolohkare tunnistettiin. Ilmakuva (Google, © 2024)

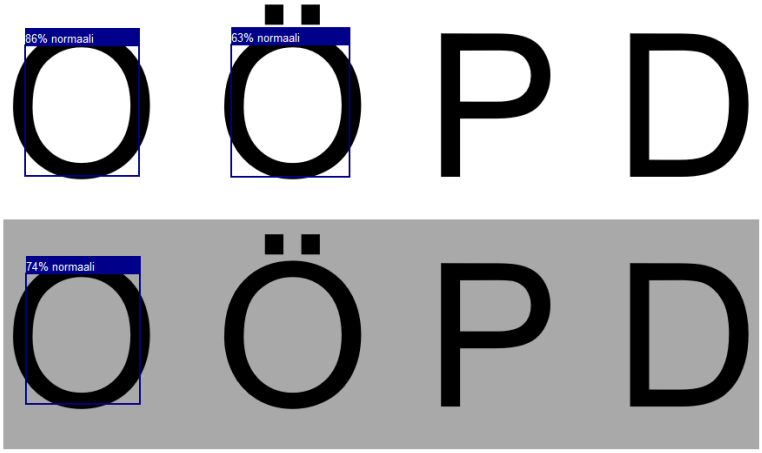
### 10.4.1 Syväoppimismallin tunnistamat piirteet

Syväoppimismalli oppi tunnistamaan siirtolohkareita, mutta se ei suoraan kerro mitä piirteitä se tunnisti oppiessaan. Toisin sanoen, mikä määrittelee siirtolohkareen ilmakuvassa ja miksi se tunnistettiin. Syväoppimismalli on kuin musta laatikko eikä siitä suoraan voi lukea syy-seuraussuhteita (Alpaydin 2021, 189). Syväoppivat mallit ovat monimutkaisen rakenteensa ja kytkentöjen lukumäärän takia erittäin vaikeasti tulkittavissa sen suhteen, miten malli tulkitsi syötteen, tehdessään siihen perustuen päätöksen (Kelleher 2020, 217).

Kirjoittajan oma hypoteesi merkitsevistä piirteistä on, että pyöreät mutta laajat, suhteellisen paksut tummat kaaret, joilla on vahva kontrasti suhteessa taustaan, voivat viitata siirtolohkareeseen. Nämä tummat kaaret edustavat varjoa, jonka siirtolohkare langettaa maahan. Varjot muodostavat tällä tavoin ohuemman tai paksumman ääriviivan ja indikaattorin mahdollisesta korkeudesta.

Tätä hypoteesia testattiin Word-ohjelmalla ja tuloksista huomattiin, että hypoteesi saattaa pitää paikkaansa. Lähes jokainen edellä kuvattua orgaanista pyöreyttä omaava kirjainmerkki luokiteltiin normaalikokoiseksi siirtolohkareeksi. Kun tausta muutettiin harmaaksi, reunojen kontrasti pieneni, ja ennusteiden todennäköisyydet pienenivät myös.

Nämä kuvassa 31 näkyvät niin sanotut väärät positiiviset ennusteet kertovat syväoppimismallin oppimista piirteistä (Alpaydin 2021, 92-93). Ulokkeita ja jyrkkiä kulmia omaavat muodot jäivät alle 50 % ennusterajan, kuten kuvassa näkyvät P- ja D-kirjaimet. Tällaisia muotoja harvoin esiintyykään siirtolohkareissa.




Kuva 31. Vääriä positiivisia ennusteita, jotka kertovat piirteistä, joita syväoppimismalli tunnisti

# 11 Yhteenveto ja pohdinta

## 11.1 Vastaukset tutkimuskysymyksiin

Ensimmäisenä tutkimuskysymyksenä oli selvittää kuinka ilmakuvat tulisi valita, jotta syväoppimismalli tunnistaisi siirtolohkareita luotettavasti. Toisin sanoen kuinka opetusdatan laatu tulisi varmistaa. Huomattiin että opetusdatan laatu on kriittinen vaihe mallin toimivuudessa. Ilmakuvat valittiin niin että siirtolohkareet pystyttiin silmämääräisesti tunnistamaan ja näin ollen myös rajaamaan kuvista oikein. Sellaisia kuvia ei käytetty, joissa siirtolohkareen sijaintia tai kokoa olisi joutunut arvaamaan. Google Earthin jälkikäsittelytoiminnot täytyi kytkeä pois, jotta ne eivät vaikuttaneet kuvien laatuun heikentävästi, koska esimerkiksi reunojen pehmennys poistaa kuvista yksityiskohtaisuutta. Ilmakuvien resoluutio on yleisesti vielä suhteellisen heikko, ja niissä on aina kohinaa, joka rajoittaa mallin tarkkuutta. Valaistus, varjot sekä kuvakulmat poikkeavat huomattavasti eri ilmakuvissa alueittain ja ajankohdittain, mutta niiden osalta vaihtelua sallittiin, jotta syväoppimismalli oppisi paremmin yleistämään.

Toisena tutkimuskysymyksenä oli tutkia miten siirtolohkareen rajaaminen ilmakuvien esikäsittelyvaiheessa vaikuttaa syväoppimismallin tuottamaan ennusteeseen. Tätä oli työn aikana vaikea todentaa, koska ilmakuvat olisi pitänyt rajata ja annotoida useaan kertaan eri tavalla, ja suorittaa mallin opetus eri rajauksilla. Yleisesti malli kuitenkin suoriutui kohtalaisen hyvin. mAP@0.5-mittari saavutti arvon 81,9 %, eli voidaan todeta, että rajaus oli onnistunut. $F_1$-mittari saavutti arvon 85 %, eli ilmakuvista opittiin kohtalaisen hyvin.

Kolmantena tutkimuskysymyksenä oli selvittää, mitkä tekijät vaikuttavat syväoppimismallin kykyyn erotella eri kokoisia siirtolohkareita ilmakuvissa. Luokkakohtainen vinouma ilmakuvien määrässä aiheutti sen, että malli tunnisti testikäytössä paremmin normaalit kuin suuret tai todella suuret siirtolohkareet, mutta suuria siirtolohkareita kuitenkin tunnistettiin. Sekaannusmatriisin perusteella todettiin, että validointivaiheessa normaali-luokka ennustettiin oikein 58 % tapauksista, suuri-luokka 75 % ja todella suuri -luokka 100 %. Lisäämällä ilmakuvien määrää suurista ja todella suurista siirtolohkareista syväoppimismalli oppisi yleistämään paremmin, jolloin kyky erotella eri kokoisia siirtolohkareita paranisi.

Neljäntenä tutkimuskysymyksenä oli tutkia, miten syväoppimismallin opettamista ja optimointia voidaan hienosäätää parempien tulosten saavuttamiseksi. Mallikohtaisista hyperparametreista huomattiin, että opetuskierrosten lukumäärä paransi tuloksia, samoin oppimisnopeus ja optimoija, sekä mallin koko 640x640, joka on suoraan verrannollinen käytössä olevan laskentatehon, erityisesti muistin määrään. Stokastisella gradienttilaskulla saavutettiin parempi tulos AdamW-optimoijaan verrattuna.

Viidentenä tutkimuskysymyksenä oli selvittää mitä rajoitteita syväoppimismallin visualisoinnissa työpöytäsovelluksessa havaitaan. Mallin visualisoinnissa huomattiin rajoitteita ja haasteita, jotka liittyvät korkearesoluutioisen näyttökuvan tehokkaaseen käsittelyyn syväoppimismallin avulla, sekä tulosten luotettavuuteen resoluutiosta johtuen. Pienemmällä resoluutiolla saavutettiin noin 300 millisekunnin päivitysnopeus per ruutu. Tämä ei ole lähelläkään reaaliaikaisuutta, mutta lisäämällä koneeseen laskentatehoa ja optimoimalla koodia se voidaan saavuttaa.

## 11.2 Pohdinta

Työn tavoitteet saavutettiin. Tutkittiin ja kehitettiin prosessi, jolla mahdollistetaan siirtolohkareiden automaattinen tunnistaminen ja luokittelu ilmakuvista. Teknisesti saatiin toteutettua ja testattua sekä syväoppimismalli että työpöytäsovellus, joka oli työn toinen tavoite. Todettiin että kyllä, siirtolohkareita voi tunnistaa ja luokitella automaattisesti ilmakuvista.

Huomattiin että syväoppimismalli oppi tunnistamaan siirtolohkareita ilmakuvista. $F_1$-mittari saavutti arvon 85 %, mikä ei vielä takaa luotettavia tuloksia, ja malli tekee myös vääriä positiivisia ja negatiivisia havaintoja. Opittiin kuinka paljon datan määrä ja laatu vaikuttavat syväoppimismallin laatuun. Tunnistaminen toimisi huomattavasti paremmin, jos ilmakuvien laatu kehittyisi tarkemmaksi korkeammalle resoluutiolle, jolloin pienien kohteiden kuten siirtolohkareiden tunnistaminen olisi myös helpompaa. Tällaista investointia ei kuitenkaan ole todennäköisesti odotettavissa, koska nykyinen kuvatarkkuus vastaa jo riittävän hyvin niihin tarpeisiin, joita varten ilmakuvaus on alun perin kehitetty. Näitä haasteita voidaan kuitenkin kiertää rikastamalla dataa tuomalla sitä esimerkiksi eri mittakaavoissa, asennoissa, valaistuksissa, jolloin malli oppii tunnistamaan ja yleistämään siirtolohkareita paremmin.

Tässä työssä tutkittiin siirtolohkareiden automaattista tunnistamista, mutta samaa prosessia voidaan hyödyntää missä tahansa alueella, jossa tarvitaan nopeaa ja automaattista esineiden tunnistamista lähtökuvasta. On tärkeä kuitenkin muistaa eettisestä näkökulmasta, että syväoppimismalli tekee vääriä ennusteita, ja vaikka dataa lisättäisiin, virheen mahdollisuus on kuitenkin aina olemassa. Tämän takia ratkaisua ei suositella toimintakriittisiin ympäristöihin sellaisenaan, vaan ihmisellä on aina oltava mahdollisuus tulkita mallin tekemiä ennusteita, ja säätää mallia tarvittaessa. Erityisesti jos prosessin datankeräysvaihetta jollain tavalla automatisoidaan, tulee huomioida yksityisyydensuoja ilmakuvien käsittelyssä.

Tekoäly kehittyy valtavan nopeasti. Silti, tiheissä metsissä ja puiden latvustojen alla piileksivät siirtolohkareet jäävät tekniikan kehittyessäkin edelleen odottamaan löytäjäänsä. Tämä kutsuu tutkijaa yhä uudestaan lähtemään koneen ääreltä ulos luontoon etsimään vastauksia.

## 11.3 Jatkokehittämisideat

Jatkokehitysmahdollisuutena olisi ilmakuvien ohjelmallinen tarkennus tekoälyn avulla, sekä ohjelmallinen datan rikastaminen, joilla parannetaan lähtödatan määrää keinotekoisesti. Ilmakuvadataa voidaan myös kerätä lisää eri kokoisista ja muotoisista siirtolohkareista sekä ilmakuvista mutta myös uusista lähteistä esimerkiksi dronekuvauksella. Näillä keinoilla voidaan parantaa syväoppimismallin tarkkuutta ja kykyä yleistää.

Voidaan myös tutkia tarkemmin sitä, miten syväoppimismalli tulkitsee sille annetut syötteet. Visualisoimalla neuroverkkojen osien aktivoitumista erilaisilla syötteillä voitaisiin saada selville mitkä piirteet ovat ratkaisevia mallin tuottamien ennusteiden kannalta. Tämän kaltainen tutkimus tulee jatkossa olemaan tärkeää myös eettisesti, kun pitää ymmärtää miksi tekoäly on päätynyt tiettyyn ratkaisuun tai ennusteeseen.

Lisäksi, YOLOsta on tämän opinnäytetyön kirjoittamisen aikana ilmestynyt jo kolme uutta versiota YOLOv9, YOLOv10 ja YOLO11. Niitä voidaan tutkia seuraavaksi, ja testata miten ne parantavat syväoppimismallien tehokkuutta, tarkkuutta ja mukautumiskykyä. YOLO11 mainostaa tarvitsevansa 22 % vähemmän parametreja mutta saavuttaa silti saman tarkkuuden kuin YOLOv8. (Ultralytics 2024.)

## Lähteet

Alpaydin, E. 2021. Koneoppiminen. Terra Cognita. Suomentanut Kimmo Pietiläinen. Alkuperäisteos: Machine Learning. Revised and updated edition.

Alvi, F. 2024. PyTorch vs TensorFlow. OpenCV. Viitattu 17.7.2024. Saatavissa https://opencv.org/blog/pytorch-vs-tensorflow/

Boyd, E. 2017. Microsoft and Facebook Create Open Ecosystem for AI Model Interoperability. Microsoft. Viitattu 17.7.2024. Saatavissa https://azure.microsoft.com/en-us/blog/microsoft-and-facebook-create-open-ecosystem-for-ai-model-interoperability/

ByteHide. 2023. WPF vs WinForms – Which One is Right for Your Project? Viitattu 8.5.2024. Saatavissa https://www.bytehide.com/blog/wpf-vs-winforms

Germanov A. 2023. How to Detect Objects in Images Using the YOLOv8 Neural Network. freeCodeCamp. Viitattu 7.5.2024. Saatavissa https://www.freecodecamp.org/news/how-to-detect-objects-in-images-using-yolov8/

Google. 2024. Google Earth. Viitattu 21.4.2024. Saatavissa https://earth.google.com/

He, K., Zhang, X., Ren, S., & Sun, J. 2014. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. arXiv. Viitattu 29.6.2024. Saatavissa https://arxiv.org/abs/1406.4729

Järvenpää, J. 2019. Python-ohjelmoinnin opas opettajalle. Edita Publishing.

Kananen, O. & Puolitaival, H. 2019. Tekoäly – Businekseen uudet työkalut. Alma Talent.

Kelleher, J. 2020. Syväoppiminen. Terra Cognita. Suomentanut Kimmo Pietiläinen. Alkuperäisteos: Deep Learning.

Kesäläinen, T. & Kejonen, A. 2019. Suomen lohkareet ja tarinakivet. Salakirjat.

Kolari, J & Kallio, A. 2023. Tekoäly 123 Matkaopas tulevaisuuteen. Docendo.

Korosuo, S. 2017. Suomalainen kiipeilyopas. Aula & Co.

Kummarikuntla, T. 2023. Choosing the Right Python GUI Framework: A Complete Guide. ToolJet. Viitattu 9.5.2024. Saatavissa https://blog.tooljet.com/python-gui-framework/

Lindroos, T. 2023. Kiipeily-lehti. Boulderalueiden ylläpito ja kehittäminen. Suomen kiipeilyliitto ry.

Louridas, P. 2021. Algoritmit. Terra Cognita. Suomentanut Kimmo Pietiläinen. Alkuperäisteos: Algorithms.

Lukka, K. 2001. Konstruktiivinen tutkimusote. Metodix. Viitattu 30.6.2024. Saatavissa https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/

Maanmittauslaitos. Kansalaisen karttapaikka. Viitattu 6.3.2024. Saatavissa https://asiointi.maanmittauslaitos.fi/karttapaikka/

Magnusson, P. 2023. Anchor-free object detection in surveillance applications. Master's Thesis, Department of Computer Science and Media Technology, Faculty of Technology and Society, Malmö University. Saatavissa https://www.diva-portal.org/smash/get/diva2:1774895/FULLTEXT01.pdf

Microsoft. 2023. Koneoppimismallien tulokset. Viitattu 27.10.2024. Saatavissa https://learn.microsoft.com/fi-fi/dynamics365/finance/finance-insights/confusion-matrix

Miettinen, s. 2006. GPS Käsikirja. Genimap Oy.

Niemelä, O. 2004. Maasto ja Kartta - kartanvalmistajan ja kartankäyttäjän käsikirja. Genimap Oy.

Oksanen J., Sirkiä O., Ahonen T., Ilves R., Pyysalo U., Ahokas E. 2016. Kansallisen maastotietokannan laatumalli. Maanmittauslaitos. Viitattu 29.6.2024. Saatavissa https://www.maanmittauslaitos.fi/sites/maanmittauslaitos.fi/files/attachments/2017/05/KMTK_korkeusmallit_laatukasikirja_2017-01-02.pdf

ONNX. 2024. Viitattu 29.6.2024. Saatavissa https://onnx.ai/

Patel, M. 2017. When two trends fuse: PyTorch and recommender systems. O'Reilly Media, Inc. Viitattu 7.5.2024. Saatavissa https://www.oreilly.com/content/when-two-trends-fuse-pytorch-and-recommender-systems/

Ramachandran, P. Zoph, B.  Le, Q.-V. 2017. Searching for Activation Functions. arXiv. Saatavissa https://arxiv.org/pdf/1710.05941v2

Ren S., He K., Girshick R., Sun J. 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv. Viitattu 29.6.2024. Saatavissa https://arxiv.org/abs/1506.01497

Sasaki, Y. 2007. The Truth of the F-measure. Viitattu 29.6.2024. Saatavissa https://nicolasshu.com/assets/pdf/Sasaki_2007_The%20Truth%20of%20the%20F-measure.pdf

Statcounter GlobalStats. 2024. Viitattu 20.4.2024. Saatavissa https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide/

TensorFlow. 2024. Viitattu 12.5.2024. Saatavissa https://www.tensorflow.org/

Terven J., Córdova-Esparza, D.-M., Romero-González, J.-A. 2023. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. Mach. Learn. Knowl. Extr. MDPI. Saatavissa https://www.researchgate.net/publication/375783101_A_Comprehensive_Review_of_YOLO_Architectures_in_Computer_Vision_From_YOLOv1_to_YOLOv8_and_YOLO-NAS

Ultralytics. 2024. Ultralytics YOLOv8 Documentation. Viitattu 28.4.2024. Saatavissa https://docs.ultralytics.com/models/yolov8