Kohteen tunnistus tekoälyn avulla - Oman tunnistusmallin kouluttaminen ja implementointi asiakkaan kohdejärjestelmään

Juha Rouvinen

Haaga-Helia ammattikorkeakoulu
Tradenomi
Opinnäytetyö
2024

Tiivistelmä

Tekijä(t)
Juha Rouvinen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Kohteen tunnistus tekoälyn avulla - Oman tunnistusmallin kouluttaminen ja implementointi asiakkaan kohdejärjestelmään

Sivu- ja liitesivumäärä
63 + 14

Tekoälyn nousevana aikakautena kyky kouluttaa ja käyttää omaa tietokonenäkömallia voi antaa yritykselle merkittävän kilpailuedun muihin toimijoihin nähden. Se voi myös avata mahdollisuuksia uusien tuotteiden ja palveluiden luomiselle tai olemassa olevien toimintojen merkittävälle tehostamiselle. Yksi tekoälyyn liittyvistä keskeisimmistä läpimurroista on ollut tietokonenäön ja erityisesti kohteentunnistuksen kehitys, joka mahdollistaa aivan uudenlaisen kommunikaation digitaalisen ja analogisen maailman välillä. Vaikka kohteentunnistamisella on valtava potentiaali erilaisissa käyttötapauksissa, on oman kohteentunnistusmallin luominen haastava ja monimutkainen prosessi.

Tämä opinnäytetyö toteutettiin asiakkaalle, ja työn keskeisinä tavoitteina oli luoda asiakkaalle kyky kouluttaa kohteiden tunnistamiseen kykeneviä neuroverkkoja suljetuissa ympäristöissä sekä kouluttaa kohteiden tunnistamiseen kykenevä neuroverkko asiakkaan tutkimusprojektille.

Tässä opinnäytetyössä kartoitettiin erilaisten kohteentunnistusmenetelmien eroja ja käytännön ratkaisuja. Kartoituksen avulla luotiin suunnitelma ja toiminnallinen määrittely, jonka avulla pystytään rakentamaan asiakkaan vaatimuksiin vastaava neuroverkon koulutusjärjestelmä ja asiakkaan tarpeisiin sopiva kohteita tunnistava neuroverkko. Kartoituksen osana tehtiin myös selvitystä siitä, mitkä ovat neuroverkon kyvykkyyden arviointiin sopivat parametrit ja koulutukseen vaikuttavat tekijät sekä millainen kohteentunnistusjärjestelmän arkkitehtuuri tulisi olla.

Tuloksena saatiin koulutettua neuroverkko, joka suurimmalta osin kykenee tunnistamaan kohteita asiakkaan vaatimilla tarkkuuksilla. Puutteita tarkkuudessa havaittiin esiintyvän lähinnä muutamien luokkien osalta, minkä analysointiin johtuvan koulutusmateriaalin vähyydestä näiden luokkien osalta. Tätä koulutusmateriaalin epätasapainoa ei pystytty korjaamaan täysin työn aikana, mutta työn aikana saatiin kehitettyä sopivat tekniikat ja menetelmät näiden puutteiden korjaamiseksi.

Tuloksista voidaan nostaa esiin paljon erilaisia näkökulmia ja kehityksen kohteita niin prosessikehityksen kuin teknisen kehityksenkin osalta. Suurin kehityksen kohde prosessi- ja valmistelumielessä havaittiin olevan jonkinlaisen koulutussuunnitelman tai strategian luominen jo siinä vaiheessa, kun uuden mallin koulutusta suunnitellaan. Tämä strategia voisi pitää sisällään käytännössä kaikki uuden mallin koulutuksen oleelliset vaiheet aina koulutusmateriaalin keräämisestä ja analysoinnista aina mallin testaamiseen ja tuotantoon viemiseen asti ainakin niiltä osin, kun on tarpeellista ja mahdollista kuvata. Tämä lähestymistapa mahdollistaisi johdonmukaisemman koulutusprosessin mallin koulutuksen eri vaiheissa.

Asiasanat
Tekoäly, konenäkö, neuroverkot,

Sisällys

1 Johdanto

Tekoälyn nousevana aikakautena kyky kouluttaa ja käyttää omaa tietokonenäkömallia voi antaa yritykselle merkittävän kilpailuedun muihin toimijoihin nähden. Se voi myös avata mahdollisuuksia uusien tuotteiden ja palveluiden luomiselle tai olemassa olevien toimintojen merkittävälle tehostamiselle.

Nopeasti muuttuvassa teknologisessa ympäristössä tekoälyn kehitys on yksi eturintaman kohteista, jossa uusia innovaatioita tapahtuu melkein viikoittain. Yksi tekoälyrintaman keskeisimmistä läpimurroista on ollut tietokonenäön ja erityisesti kohteentunnistuksen kehitys, joka mahdollistaa aivan uudenlaisen kommunikaation digitaalisen ja analogisen maailman välillä. Näillä aloilla tapahtunut kehitys tasoittaa tietä valtavalle määrälle uusia mahdollisuuksia, jotka ulottuvat kaikille elämän aloille aina jokapäiväisestä elämästä, teollisuuteen ja yksityiselämään saakka. (West, Allen. 2018.) Tietokonenäköä hyödyntävät ratkaisut tekevät tuloaan aina hienostuneista valvontajärjestelmistä lääketieteen kuvantamiseen saakka, ja jopa kaikkein yksityisimmän ja rakkaimman lomakuva-albumin tärkeimpien hetkien tunnistamiseen on tulossa omat ratkaisunsa. Ja tämä kaikki on vasta alkua. Yritykset tarvitsevat digitaalista tietoa käytännössä kaikilla osa-alueillaan aina logistiikasta markkinointiin ja henkilöstöstrategioiden luomiseen. (University of Liverpool 2021.) Tiedolla johtamisesta onkin tullut yksi merkittävimmistä tavoista hakea etuja kilpailijoihin nähden. Tekoälysovelluksista juuri konenäköön perustuvalla kohteentunnistuksella on merkittävä rooli tässä, koska se mahdollistaa ennennäkemättömän nopeuden ja tarkkuuden alun perin analogisen tiedon analysointiin ja tulkintaan, mikä auttaa yrityksiä tekemään parempia päätöksiä.

Valtavasta potentiaalistaan huolimatta oman tietokonenäkö- tai kohteentunnistusmallin luominen on haastava ja monimutkainen prosessi. Tämä lopputyö pyrkii kartoittamaan, millainen järjestelmäarkkitehtuuri vaaditaan oman kohteentunnistusmallin kehittämiseen ja erityisesti kouluttamiseen. Tarkoituksena on, että koko prosessi kartoitetaan toiminnallisten vaatimusten ja koulutustiedon keräämisestä aina lopullisen ratkaisun jalkauttamiseen työn toimeksiantajan ohjelmistokohdearkkitehtuuriin.

Yksi tässä lopputyössä käytetyistä avainkäsitteistä on datamanipulaatio ja sen käyttäminen lopullisen, toiminnallisen tuotteen saavuttamisessa. Datamanipulaatio on tekniikka, jolla voidaan kasvattaa kohteentunnistusmallin koulutusmateriaalia ja monipuolisuutta ilman että koulutusmateriaaliin kerätään jatkuvasti uutta tietoa. Tämä tekniikka ei ole ainoa käytössä ollut mallin koulutusta helpottava tai parantava tekniikka, ja siksi tässä työssä käsitellään myös muita lopputuotoksen tekemisessä käytettyjä tekniikoita, joilla voitaisiin tulevaisuudessa parantaa mallin koulutuksen tavoitteiden saavuttamista. Esimerkiksi opitun siirtämistä, jossa jo opetettua mallia jatko- tai hieno jalostetaan uudella koulutusmateriaalilla, on esitelty hyvin laajasti tämän työn osana. Omaa

kohteentunnistusmallia koulutettaessa on ensiarvoisen tärkeää käyttää erilaisia menetelmiä monipuolisesti, jotta koulutettu tunnistusmalli kykenisi mahdollisimman hyvin suoriutumaan tehtävästään tosimaailman tilanteissa.

Kyky luoda omia kohteentunnistusmalleja voi olla todellinen game-changer, sillä se avaa aivan uudet mahdollisuudet yrityksille luoda sellaisia innovaatioita, jotka sopivat erityisen hyvin yrityksen toimialaan ja muuhun tuotekehitykseen. Tämä voi puolestaan johtaa yrityksen omien prosessien huomattavaan tehostumiseen ja avata mahdollisuuksia useilla eri aloilla. (Sjödin, Palmie. 2021,4.)

Tämä lopputyö kartoittaa sitä pikkutarkkaa ja haastavaa työtä, jota oman konenäköön perustuvan kohteentunnistusmallin luominen vaatii, ja jonka mahdolliset palkinnot ovat suuret, mikäli tekijän hermot ja keskittyminen pysyvät kohdallaan opinnäytetyöprosessin loppuun asti.

Sanasto

Aktivaatiofunktio	On matemaattinen funktio, jota käytetään neuroverkoissa lineaarisuuden poistamiseksi yksittäisissä neuroneissa. (Szeliski 2021, 272)
Annotaatio	On prosessi, jossa tekoälyn kouluttamiseen käytettävistä kuvista erotellaan koulutuksen kannalta olennaiset asiat tai kohteet. (Orenda Minds 2019)
Automaattinen annotointi	Automatiikan avulla tapahtuva tekoälyn koulutusmateriaalin nimikointi. (Infosys BPM s.a)
Vasta-virta algoritmi	Algoritmi, jota käytetään myötä kytkettyjen neuroverkkojen kouluttamisessa. Algoritmi toimii kaksi-vaiheisesti, jossa ensimmäisessä vaiheessa koulutusmateriaali syötetään neuroverkolle tunnistettavaksi ja toisessa vaiheessa arvioidaan tunnistuksen hyvyyttä koko verkon läpi laskemalla häviöfuntio jokaiselle neuronille. (Coursesteach 2024)
Sarjan normalisointi	Tekniikka, jonka avulla parannetaan neuroverkon nopeutta, tehokkuutta ja vakautta. (Szeliski 2021, 277)
Tunnistuslaatikko	Yksi nimikointimenetelmä, jossa tekoälyn kouluttamisen kannalta olennaisen kohteen ympärille piirretään laatikko, joka kertoo kohteen sijainnin ja luokan kuvassa. (Subramanyam 2021)
Luokittelu	Kertoo tekoälymallille koulutuksen kannalta kiinnostavien kohteiden nimen, laadun, tyypin tai vastaavan tiedon, jonka perusteella kohde halutaan erotella tunnistettavasta materiaalista. (Brownlee 2020)
Convolutional Neural Networks (CNN)	Ovat erityiseesti kuvien käsittelyyn tarkoitettuja neuroverkkoja. Ne ovat erityisesti kehitetty tunnistamaan kuvista yksityiskohtia. (Mishra 2020)
Datamanipulaatio	Tekoälymallin koulutusmateriaalin, esimerkiksi kuvien manipulointi siten, että pienestä koulutusmateriaalin massasta saadaan tuotettua enemmän koulutusmateriaalia. Tällaisia tekniikoita ovat esimerkiksi rakeisuuden lisääminen, väritasapainon muokkaaminen, kuvien pilkkominen ja/tai yhdistely. (Dilgemani 2024.)
Koulutusmateriaali	Esimerkiksi PASCAL VOC, COCO, ImageNet, ovat yleisesti käytettyjä tieto/kuvakirjastoja, joita käytetään

	tekoälymallien koulutuksessa, arvioinnissa ja testauksessa. (Bansal 2019.)
Syväoppiva	Koneoppimisen/tekoälyn alakategoria, jossa neuroverkon kerrokset simuloivat sitä, miten ihmisen mieli/aivot käsittelevät tietoa. (Stanford University s.b.)
Deterministinen	Teoria, joka viittaa ajatukseen, jossa kaikki tapahtumat määräytyvät aikaisempien syiden / tapahtumien perusteella,
Diskriminoiva	Tiettyjen ominaisuuksien perusteella jonkin asian toiminteen tai muun sellaisen arvottaminen huonommaksi kuin jonkin toisen tekemä vastaava toiminta.
Gradienttimenetelmä	Menetelmän tavoite on minimoida neuroverkon sisäisitä parametreistä riippuvaa virhefunktoita, eli syötteiden ja verkon antamien tulosten välistä virhettä. (Szeliski 2021, 268.)
Häviöfunktio	Häviöfunktio, joka mittaa kuinka hyvin neuroverkko suoriutuu sille annetusta tehtävästä, esimerkiksi kohteen tunnistamisesta. (Parmar 2018)
Yli- ja alioppiminen	Tekoälymallin ei toivottavat tilat, käytännössä johtuvat huonosta koulutusmateriaalista, jolloin malli joko ei kykene ‘soveltamaan’ oppimaansa tai ei kykene tunnistamaan kohteita tai asioita materiaalista. (Szeliski 2021, 199-200.)
Perceptroni	Ihmisaivojen neuronin mallintamiseen kehitetty keinotekoinen neuroni, jonka tehtävänä on simuloida tai jäljitellä aivoissa olevia synapseja. Neuroni aktivoituu saamansa signaalin johdosta ja suorittaa signaalille sille asetutut toiminteet ennen signaalin lähettämistä eteenpäin. (Paananen. 2018, 7-8.)
Realistinen	Todellisuuden todenmukainen kuvaaminen, tapahtumien tai asioiden esittäminen siten kuten ne tosiasiassa ovat.

2 Kohteiden tunnistus neuroverkkojen avulla

Tässä kappaleessa esitellään keskeiset käsitteet, joita tarvitaan tietokonenäön ymmärtämiseen ja kohteentunnistusjärjestelmän rakenteen hahmottamiseen.

Ihmiset käsittävät näkemällä ympäröivän kolmiulotteisen maailman näennäisen helposti. Tätä voi havainnollistaa kuvittelemalla esimerkiksi kukkaruukun, jossa on kukkia. Ihminen kykenee erottelemaan sekä kokonaisuuden että yksityiskohdat: jokaisen kukan terälehden ja niiden muodot, läpinäkyvyyden kukkaan osuvien valonsäteiden avulla. Kukat erottuvat mielessämme selkeästi taustasta, osana asetelmaa. Katsoessamme koulun luokkakuvaa kykenemme helposti laskemaan ihmisten määrän sekä päättelemään heidän mielenmaisemaansa tai olotilaansa kasvojen ilmeiden avulla. Psykologit ovat pyrkineet jo vuosikymmeniä ymmärtämään kuinka ihmisen visuaalinen tunnistuskyky toimii, ja tämän ohessa onkin luotu muutamia erittäin hienostuneita optisia illuusioita. Täydellistä ymmärrystä tämän ihmisen hahmotuskyvyn toiminnasta ei ole kuitenkaan saatu luotua. (Szeliski 2021, 3.)

Tietojenkäsittelytieteen tutkijat ovat kehittäneet matemaatikkojen kanssa yhteistyössä menetelmiä, joiden avulla voidaan hahmottaa kolmiulotteisten kuvioita tai esineiden piirteitä. Viimeisten vuosikymmenten kehitys on ollut erittäin nopeaa, ja tällä hetkellä kykenemme kartoittamaan luotettavasti kolmiulotteisen mallin ympäristöstä käyttämällä useita satoja tai tuhansia kuvia hieman eri perspektiivistä. Mikäli tietystä kohteesta saadaan tarpeeksi kuvia, voidaan tämän tiedon avulla luoda kohteesta tarkka kolmiulotteinen mallinnus käyttämällä syvyyden määrittelyyn tarkoitettuja laskennallisia menetelmiä. Tällä hetkellä parhaimmat suuret tekoälymallit yhdistettynä kuvantunnistuskykyyn pystyvätkin kuvailemaan kuvassa näkemänsä asiat hyvin suurella tarkkuudella. (Szeliski 2021, 3.)

Tietokoneen näkökyky on hyvin monimutkainen ja vaativa aihe, koska meidän tietämyksemme siitä, miten visuaalinen hahmotuskyky oikeasti toimii ihmisissä, on vielä osittain hämärän peitossa. Tästä syystä joudumme turvautumaan fysiikan ja tietokonegrafiikan meille antamiin keinoihin silloin, kun pyrimme hakemaan ratkaisuja meille osittain tuntemattomiin haasteisiin. Toisin sanoen jonkin asian ratkaiseminen selkeästi ja elegantisti on hyvin vaikeaa, jos kaikkia muuttujia ei tunneta tai ei tiedetä mistä osista haluttu lopputulos oikeasti koostuu. Nykyisin käytettävät tietokonenäkömallit kykenevät havaitsemaan, mallintamaan ja ymmärtämään hyvin sitä, miten kohteista heijastuva valo hajautuu ilmakehässä, minkä jälkeen se otetaan vastaan kameran linssissä (tai ihmisen silmässä) ja lopulta esitetään projektiona tasaisella alustalla. (Szeliski 2021, 3.)

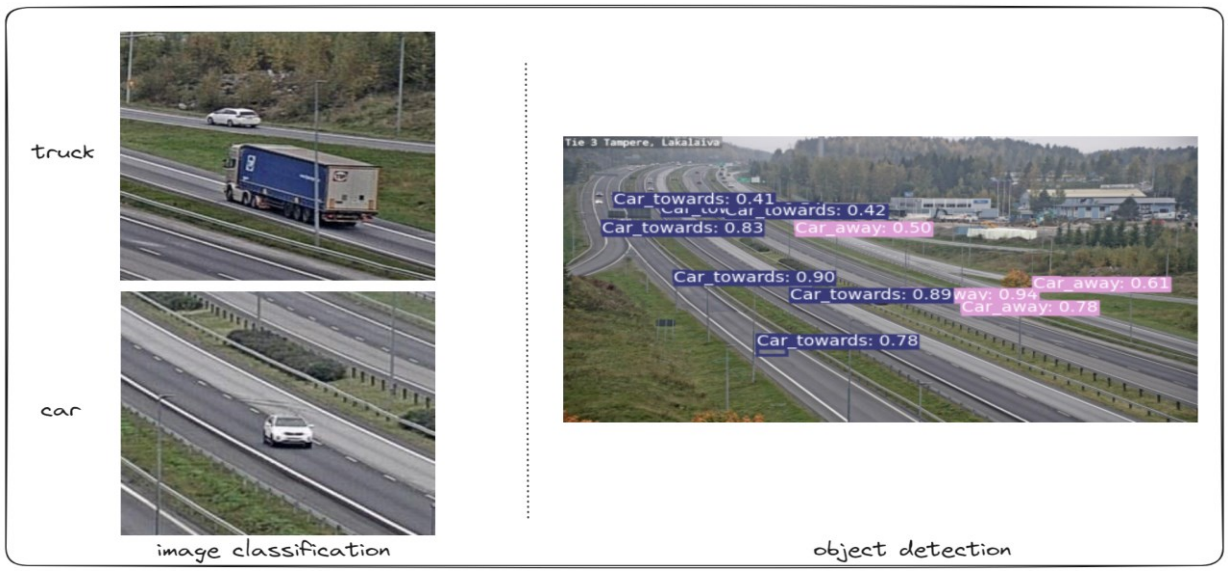
Toisaalta tietokonegrafiikan avulla pystytään luomaan mitä mielikuvituksellisempia ympäristöjä tai hahmoja sekä hyvin todentuntuisia henkilöitä tai tiloja. Tietokonenäkö ilmiönä onkin periaatteessa

käänteinen ilmiö: se pyrkii selittämään ympäröivän maailman yhden tai useamman kuvan avulla, käyttäen hyväksi siinä esiintyviä muotoja, valaistusta ja värien tasapainoa.

On kiehtovaa, kuinka vaivattomasti ihmiset ja eläimet kykenevät tekemään tämän toimenpiteen, kun taas tietokoneen algoritmit ovat kovin häiriö- ja virheherkkiä. Usein käykin niin että ihmiset, jotka eivät ole työskennelleet tietokonenäön parissa aliarvioivat tämän ongelman ratkaisemisen monimutkaisuutta. Tämä harhaluulo juontaa juurensa tekoälynkehittämisen alkuaikoihin, jolloin uskottiin, että kognitiiviset (logiikka ja suunnittelu) osat älykkyydestä olisivat luonnostaan paljon vaikeampia ratkaista kuin visuaalisen kyvykkyyden osat. (Zuo, Qian, Feng & Yin 2022, 2.)

2.1 Kohteiden tunnistus tietokonenäön avulla

Kohteen tunnistus on tietokonenäön alalaji, johon liittyy kohteen paikantaminen ja tunnistaminen kuvasta tai videosta. Toisin kun kuvan tunnistus, joka asettaa kokonaiselle kuvalle yksittäisen tunnistuksen, kohteentunnistus tunnistaa useita kohteita kuvasta sekä päättelee kohteiden paikan kuvassa tunnistuslaatikon avulla (Kuva 1). (Gidaris & Komodakis 2015, 1.) Erilaiset syväoppivat tunnistusmallit voidaan jakaa karkeasti erilaisiin luokkiin arkkitehtuurin ja käyttötavan mukaisesti. Arkkitehtuurin mukaisesti mallit voidaan jakaa yksi- tai kaksivaiheisiin malleihin ja käyttötavan mukaisesti mallit voidaan jakaa esimerkiksi kuvantunnistus-, kohteentunnistus- tai segmentointimalleihin. On kuitenkin huomattava, että tämän kaltainen jaottelu ei ole mitenkään vahvistettu ja termistöä käytetään usein hieman risteävästi. Mallien perusajatus on kuitenkin hyvin samankaltainen. Mallit ovat käytännössä neuroverkkoja, joiden sisäisiä painotuksia (weights) muutetaan koulutuksen aikana. Tällä tavoitellaan sitä, että kyseessä oleva neuroverkko kykenisi koulutuksen jälkeen päättelemään riittävällä tarkkuudella tunnistettavan kuvan tai kohteen. (Chen, Li, Tian & Zhang 2023, 1.)



Kuva 1 – Havaintokuva kuvan- ja kohteentunnistuksen eroista. (Mukaillen Cetinsoy 2022)

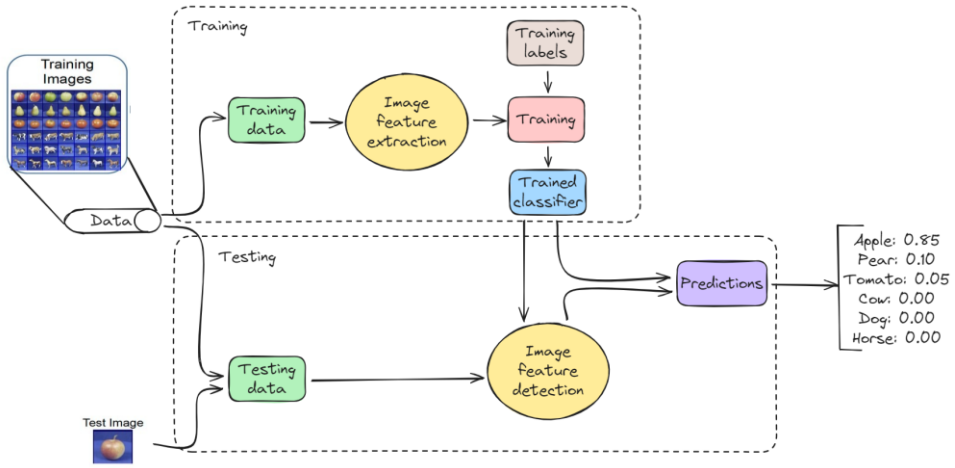
Syväoppivat kohteentunnistusmallit koulutetaan usein suurilla määrillä kuvia, jotka sisältävät tunnistettavien kohteiden tunnistuslaatikot sekä tunnistukset. Tällaiset mallit oppivat tunnistamaan ja luokittelemaan useita kohteita, kuten esimerkiksi autoja, koiria, ihmisiä ja niin edelleen. Tunnistuksen lisäksi mallit pystyvät ennustamaan opeteltujen kohteiden esiintymisen ja paikan uusissa kuvissa. (Gidaris & Komodakis 2015, 2.)

Kohteentunnistusprosessi voidaan jakaa eri vaiheisiin monin eri tavoin, kuitenkin tietyt vaiheet ovat yhteneviä kaikissa tapauksissa, niiden laajuus ja monimutkaisuus eroavat toisistaan riippuen kohteentunnistuksen projekteista. Prosessin pääpiirteet ovat kuitenkin seuraavanlaiset:

1. Ympäristön rakentaminen ja konfigurointi
2. Koulutus ja testausmateriaalin kerääminen, sekä valmistelu
3. Mallin arkkitehtuurin ja konfiguraation valitseminen, sekä sopivien parametrien asettaminen
4. Kohteentunnistusmallin kouluttaminen
5. Koulutetun kohteentunnistusmallin arviointi
6. Koulutetun kohteentunnistusmallin vieminen tuotantoon
7. Kohteentunnistusmallin käyttäminen/ylläpito tuotannossa

(Thakur, Nagrath, Jain & Saini. 2021, 12.)

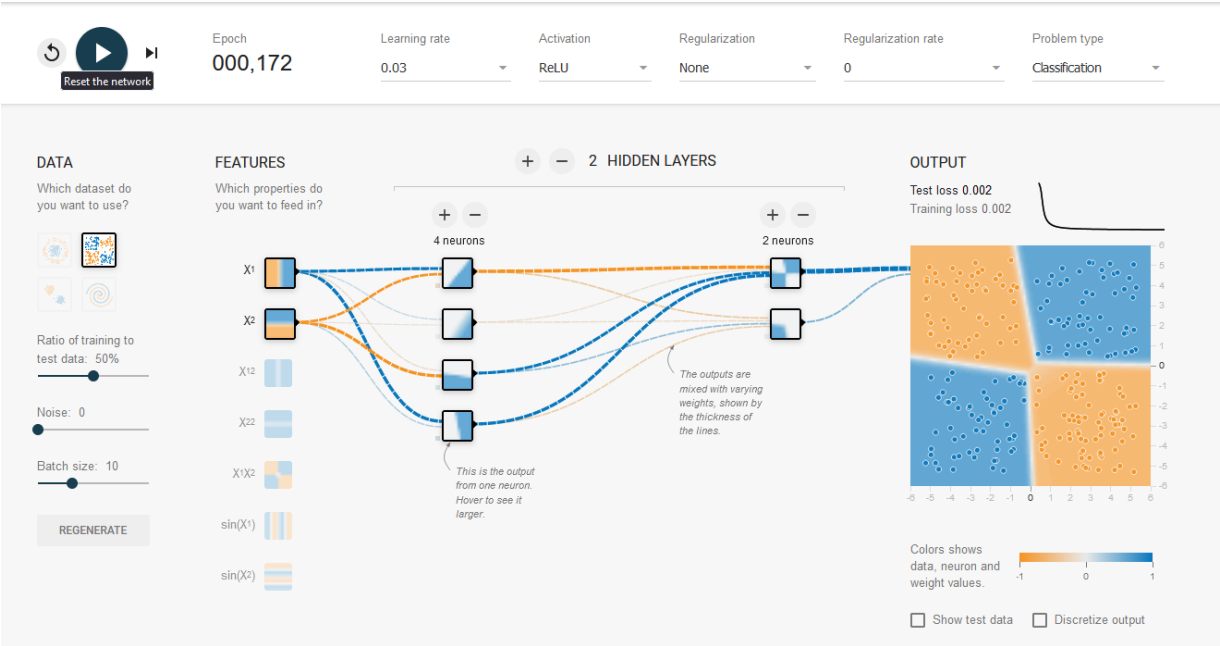
Kuvassa 2 esitetään kohteen- tai kuvantunnistus algoritmien toiminta yleisellä tasolla. Koulutuksen aikana koulutusmateriaalista eli kuvista haetaan oleellisia piirteitä ja samankaltaisuuksia, jotka muunnetaan numeraalisiksi algoritmeiksi, jotka tallennetaan opetettavaan malliin koulutuksen aikana. Koulutettavan mallin testauksen tai validoinnin aikana mallille syötetään samankaltaisia kuvia kuin koulutuksen aikana, mutta ei kuitenkaan samoja kuvia kuin koulutuksessa, ja mallin antamia vastauksia verrataan testikuvien tietoihin. (Stanford University s.a.)



Kuva 2 – Kohteentunnistuksen korkeantason prosessikuvaus (Mukaillen Thakur ym. 2021, 12; Stanford University s.a)

2.2 Neuroverkot koneoppimisen välineinä

Konenäkö- ja kohteentunnistusjärjestelmät toteutetaan usein neuroverkkojen avulla. Nykypäivänä kaikkein suosituimmat ihmisten aivoja jäljittelevät järjestelmät tunnetaan suoraviivaisina päätöksentekoverkkoina. Nämä järjestelmät tekevät päätöksiä ja päätelmiä käsittelemällä tietoa yhdensuuntaisesti hyväksikäyttäen numeroita. Samalla järjestelmät oppivat gradienttimenetelmää hyväksikäyttäen eli ne oppivat harjoittelun ja virheistä oppimisen kautta. Ennen neuroverkkojen kehittämistä, koneoppimistekniikat luottivat esiprosessointiin muuttujien löytämiseksi, joiden perusteella luokittelijat voitiin rakentaa. Syväoppivat neuroverkot luottavat taas päästä päähän tyyppiseen oppimiseen, jossa raakapikselit muunnetaan halutuiksi syötteiksi tai painotuksiksi. (Szeliski 2021, 268.) Kuvassa 3 on esimerkki yksinkertaisen neuroverkon rakenteesta. Jokaiselle neuroverkolle määritellään, kuinka paljon erilaisia piirteitä/ominaisuuksia (features) verkkoon otetaan sisään. Tämän jälkeen määritellään verkkoon kerrokset (layers), jotka käsittelevät ja tulkitsevat verkkoon syötettyjä piirteitä. Viimeisen kerroksen avulla (output) tuotetaan lopullinen tulkinta verkkoon sisään otetuista piirteistä/ominaisuuksista.



Kuva 3 – Esimerkki hyvin yksinkertaisesta neuroverkosta (Tensorflow s.a.)

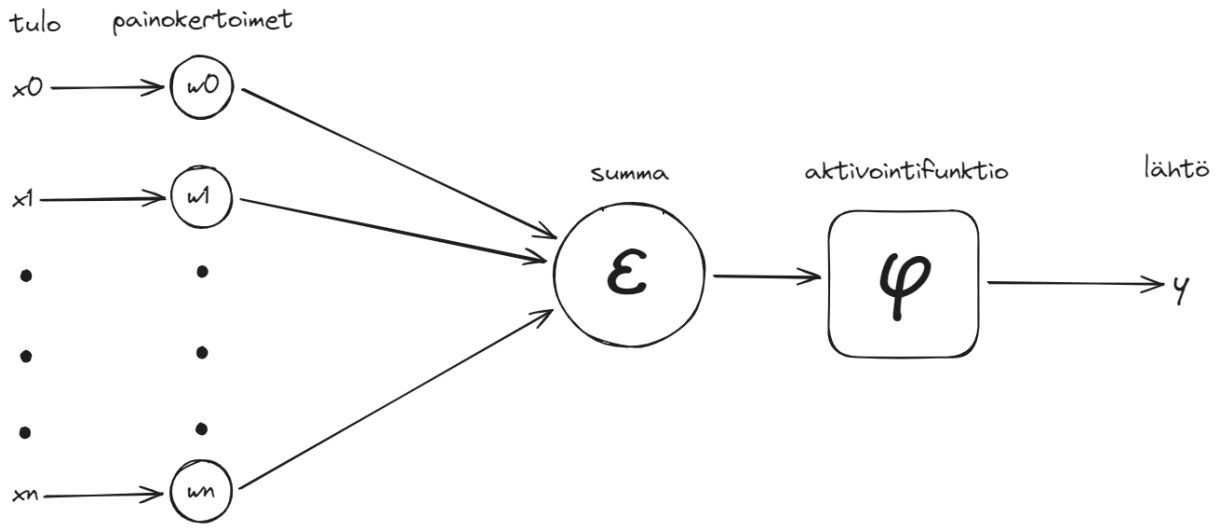
Syväoppivat neuroverkot (Deep Neural Networks - DNN) ovat usein eteenpäin syöttäviä laskentaverkkoja, jotka koostuvat tuhansista yksinkertaisista toisiinsa yhdistetyistä neuroneista. Nämä neuronit suorittavat laskentaa saamiensa syötteiden mukaisesti ja aktivoituvat tiettyjen funktion ehtojen mukaisesti. (Szeliski 2021, 269.)

Funktio voidaan määritellä seuraavasti: Sääntö, jonka avulla numero voidaan kuvata toisen uniikin numeron avulla. Toisin sanoen, jos aloittaa numerolla x ja lisää siihen funktion, saa tulokseksi numeron y. Kuvassa 4 on kuvattu yksinkertainen matemaattinen kaava funktiolle, joka lisää luvun kolme mihin tahansa numeroon. Joten jos lisätään numeron 2 funktioon, saadaan tulokseksi luku 5. Jos taas lisätään luku 5 kyseiseen funktioon, saadaan tulokseksi luku 8. Ja jos funktioon lisätään taas luku 8 saadaan tulokseksi luku 11. (Mathcentre 2009, 2.)

$$f(x) = x + 3.$$

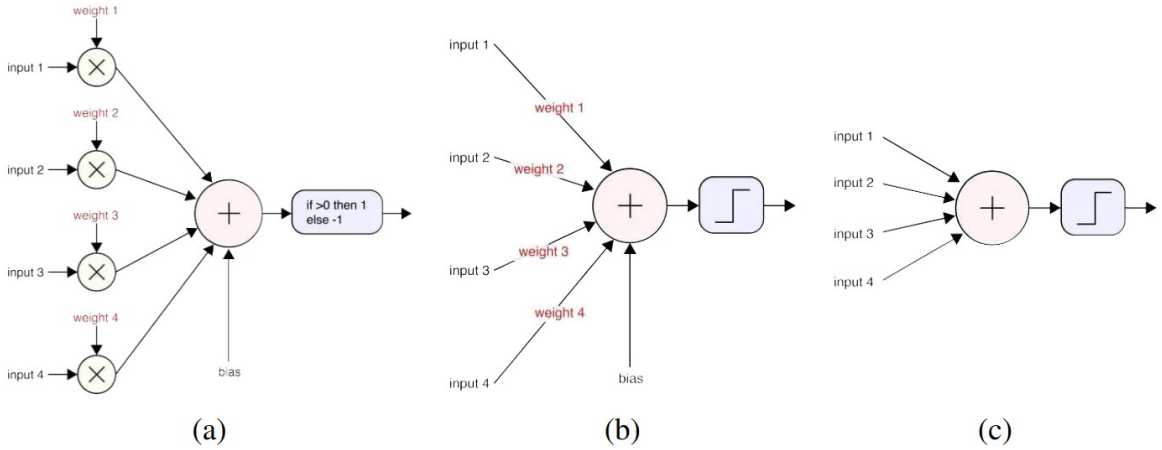
Kuva 4 – Funktio matemaattisesti aukikirjoitettuna. (Mukaillen Mathcentre 2009, 2)

Ihmisaivojen neuronin mallintamiseen kehitettiin keinotekoinen neuroni (engl. perceptroni), joka kuvaa oikean neuronin syttymistä tiettyjen tulosignaalien ja painoarvojen funktiona. On huomioitava, että keinotekoinen neuroni ei kykene käsittelemään tietoa yhtä tehokkaasta kuin sen ihmisaivojen vastine, joten on hyvin epätodennäköistä, että keinotekoisia neuroneja käyttämällä saataisiin luotua ihmisälyn kaltaista suorituskykyä. Tästä huolimatta on varmaa, että keinotekoisia neuroneita hyväksikäyttämällä saadaan luotua erittäin tehokkaita neuroverkkoja, jotka kykenevät ratkaisemaan hyvin monimutkaisia ongelmia. Keinotekoisen neuronin rakenne on kuvattu kuvassa 5, jossa on kuvattuna tulovektorit X_n, painovektorit (painotukset) W_n, summausfunktio $\sum$ ja aktivointifunktio φ. Neuroni toimii seuraavasti: jokainen tulovektorin elementti X_n kerrotaan sitä vastaavan painotuksen W_n elementin kanssa, ja summataan yhteen summausfunktiolla $\sum$. Tämän jälkeen painotettu summa syötetään aktivointifunktioon φ, joka antaa lähtöön arvon 0 tai 1, täten kuvastaen neuronin syttymistä. (Paananen. 2018, 7-8.)



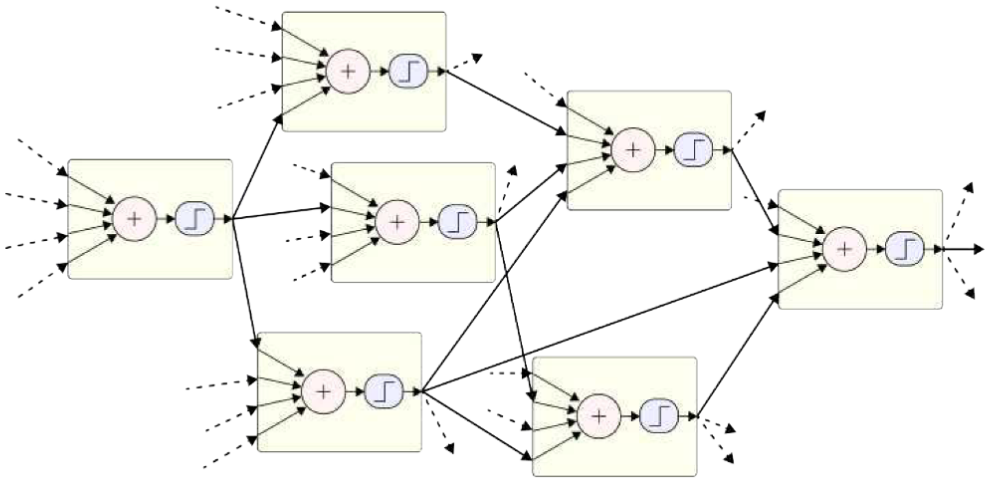
Kuva 5 – N-tuloisen neuronin malli (Mukaillen Paananen. 2018. 8)

Kuvassa 6 on kuvattuna kolme erilaista neuronin toimintaperiaatetta. Kohdassa (a) on neuroni, joka saa muuttuvat painotukset syötteiden mukaisesti. Kohdassa (b) on neuroni, jonka syötteet ovat painotettu pysyvästi. Kohdassa (c) on neuroni, jossa painotukset on jätetty huomioimatta.



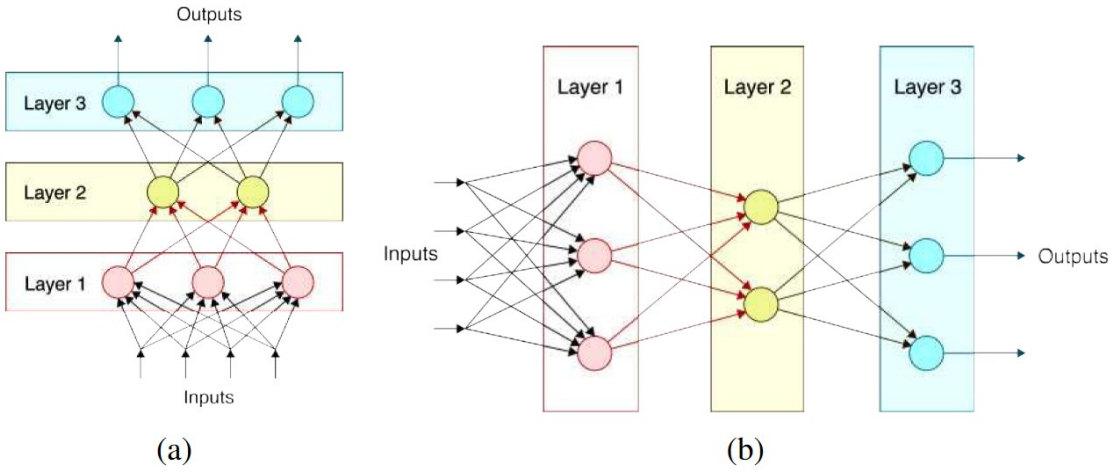
Kuva 6 – Kuvaus erilaisista neuronien toimintaperiaatteista. (Glassner 2021a; Glassner 2021b, Glassner 2021c)

Kuvassa 7 on kuvattu neuroverkko, jossa yksittäiset neuronit ovat yhdistetty epäsäännöllisesti toisiinsa.



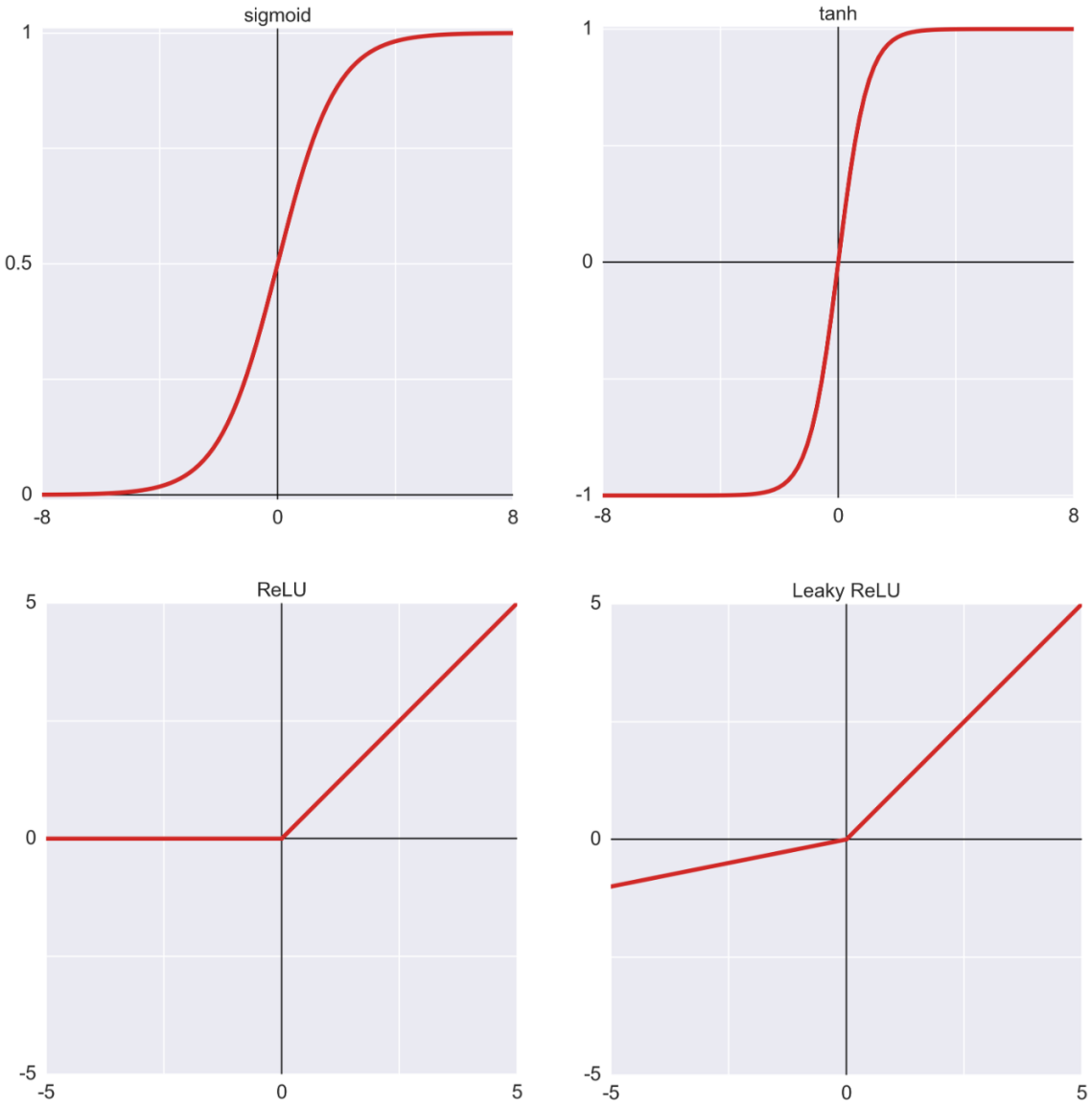
Kuva 7 – monitasoinen verkko, jossa neuronin laskennan tulos on syötetty yhteen tai useampaan neuroniin. (Glassner 2021d)

Kuvassa 8 on kuvattu kaksi erilaista neuroverkon esitystapaa, jossa yksittäiset neuronit ovat yhdistetty toisiinsa perättäisten kerrosten kautta.



Kuva 8 – Kaksi eri neuroverkon esitystapaa: (a) Syötteet alimmaisena ja tuotokset ylimpänä, (b) Syötteet vasemmalla ja tuotokset oikealla. (Glassner 2021e; Glassner 2021f)

Aktivaatiofuntiot ovat neuroneissa käytettyjä funktioita, jotka määrittävät kuinka ja miten kyseisen neuronin tulisi aktivoitua tietyssä tilanteessa. (Knok, Pap, Hrncic 2019, 4) Kuvassa 9 on kuvattu useimmin neuroneissa käytettyjen funktioiden aktivointiperiaatteet kuvaajien avulla.

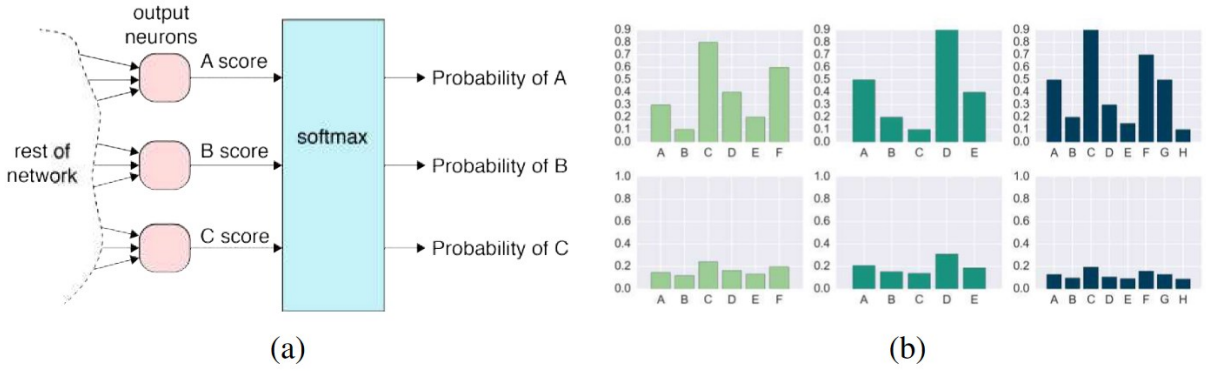


Kuva 9 – Usein käytetyt neuronin aktivointifunktiot kuvaajina (Glassner 2021g; Glassner 2021h; Glassner 2021i; Glassner 2021j)

Monet aikaisemmin käytetyt neuroverkot käyttivät neuroneissaan sigmoidisia[1] tai logistisia regressiofunktioita. Uudemmat neuroverkot käyttävät niin kutsuttua Rectified Linear Unit (ReLU) -aktivointia tai sen erilaisia variantteja, jotka ovat lineaariseen regressioon perustuvia funktioita. Neuroverkon viimeisessä kerroksessa, joka tekee luokittelun, käytetään useasti softmax-funktiota. Viimeisen kerroksen softmax-funktio muuntaa aktivoinnit todennäköisyyksiksi, joiden perusteella luokittelu voidaan suorittaa. (Szeliski 2021, 274.) Kuvassa 10 esitetään softmax-kerros

[1] Matemaattinen funktio, jolla on kyseiselle funktiolle tyypillinen ”S” -kirjaimen muotoinen kuvaaja, kun se piirretään tasolle.

yksinkertaistettuna kaaviona. Kaaviossa neuroverkon läpi tulleet tulokset syötetään softmax -kerrokseen, joka muuntaa tulokset arvioiksi todennäköisyydestä siitä mikä verkkoon annettu syöte on.



Kuva 10 – Kuvaus miten softmax (a) kerros muuntaa neuroverkon tuottamat aktivoinnit (score) tunnistusluokkien todennäköisyyksiksi (b) (Glassner 2021k; Glassner 2021l)

2.3 Neuroverkkojen kouluttaminen

Neuroverkko voidaan kouluttaa joko valvottuna tai ilman valvontaa. Valvotussa koulutuksessa neuroverkolle annetaan haluttu syöte ja sitä vastaava haluttu vaste, kun taas valvomattomassa koulutuksessa neuroverkolle annetaan tilastolliset esimerkit ilman näitä vastaavia vasteita. (Szeliski 2021, 239.) Tässä kappaleessa määritellään seuraavat neuroverkon kouluttamisen kannalta oleellisimmat tekniikat: Koulutusmateriaalin lisääminen, sarjan normalisointi, häviöfunktio, vastavirta-algoritmi ja koulutuksen siirtäminen. Neuroverkkoa koulutettaessa näitä tekniikoita voidaan käyttää joko yksittäisinä tekniikoina tai erilaisina ryhminä riippuen tilanteesta.

Neuroverkon ylioppimisella tarkoitetaan tilannetta, jossa neuroverkko oppii koulutusmateriaalin perusteella tunnistamaan kaikki mahdolliset piirteet, mukaan lukien erilaiset kohinat ja muut häiriöt. Tämä näkyy käytännössä siten, että neuroverkko suoriutuu hyvin koulutusmateriaalin tunnistamisesta/käsittelystä, mutta huonosti testi- tai validointimateriaalin tunnistamisesta/käsittelystä. Neuroverkon ylioppimiseen johtavia syitä on useita, mutta yhtenä yleisimpänä syynä voidaan pitää koulutusmateriaalin vähyys tai huono laatu, ja joissain tilanteissa jopa molemmat. (Szeliski 2021, 199-202.; McCoubrey, Elbadawi, Orlu, Gaisford & Basit 2021, 5-6.)

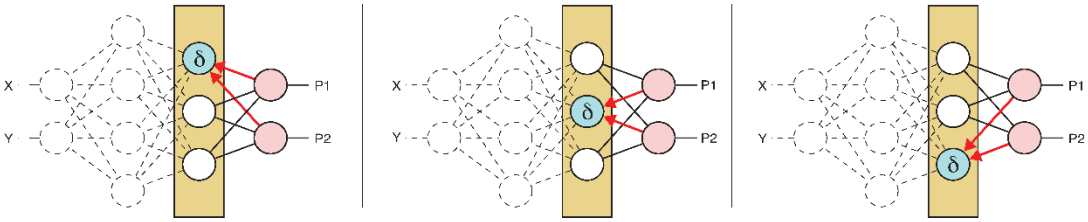
Yksi tehokkaimmista tavoista vähentää koulutettavan neuroverkon ylioppimista on *koulutusmateriaalin lisääminen* monimuotoistamalla jo olemassa olevaa koulutusmateriaalia. Tätä tekniikkaa voidaan kutsua myös koulutusmateriaalin monimuotoistamiseksi. Sen on todettu olevan erityisen tehokas kohteen luokitteluun/tunnistamiseen tarkoitettujen neuroverkkojen koulutuksessa, koska

näissä tapauksissa usein koulutusmateriaalin lisääminen muilla keinoilla vaatii huomattavan paljon aikaa ja vaivaa. (Szeliski 2021, 275.)

Neuroverkon koulutuksessa käytettävien painotusten optimointi voi olla hyvin monimutkaista ja hidasta toimintaa. Yksi klassisimmista ongelmista neuroverkon kouluttamisessa on tapaus, jossa käytettävät gradientit vaihtelevat suuresti koulutuksen aikana. Vaikka joissain tapauksissa on mahdollista ennaltaehkäistä gradienttien suuria muutoksia rajoittamalla yksittäisten elementtien muutoksia gradienteissa, on suositeltavampaa tehdä toimenpiteitä järjestelmätasolla. Usein syynä gradienttien suuriin muutoksiin neuroverkoissa on se, että peräkkäisten kerrosten väliset painotukset tai aktivoinnit joutuvat epätasapainoon. Tällöin mahdollisuus painotuksen ’yliampumiseen’ gradientin laskeutumisen aikana on hyvin todennäköistä, ellei laskeutuminen tapahdu hyvin pienin askelin. (Szeliski 2021, 277-278.)

Sarjan normalisoinnin perusidea on skaalata uudelleen yksikön aktivoinnit siten, että yksikön varianssi ja keskiarvo otetaan huomioon. Toinen tapa tehdä sarjan normalisointia on muokata tasojen aktivointeja siten, että painotuksen standardi ja painotuksen vektorin suunta erotetaan omiksi muuttujikseen. Tätä menetelmää kutsutaan tarkemmin painotuksien normalisoinniksi. Jotta neuroverkon painotuksia voidaan optimoida, tulee verkolle ensin määrittää *häviöfunktio* (Loss function), jota pyritään pienentämään koulutuksen aikana. Luokitusta varten useat neuroverkot käyttävät viimeisenä kerroksenaan softmax-funktiota käyttävää kerrosta. (Szeliski 2021, 279-280.)

Vastavirta-algoritmi hyödyntää differentiaalilaskentaa neuroverkon painotusten optimoinnissa. Käytännössä vastavirta-algoritmi käyttää ennalta määritettyä gradienttimenetelmää etsiäkseen optimiarvot neuroverkon neuroneiden välisille painotuksille. (Kallio 2021, 19.) Kuvassa 11 on kuvattu vastavirta algoritmin toimintaperiaate yhden neuronikerroksen kohdalla, kun häviöfunktion tuottamat häviöt kirjataan jokaiseen kerroksen neuroniin.

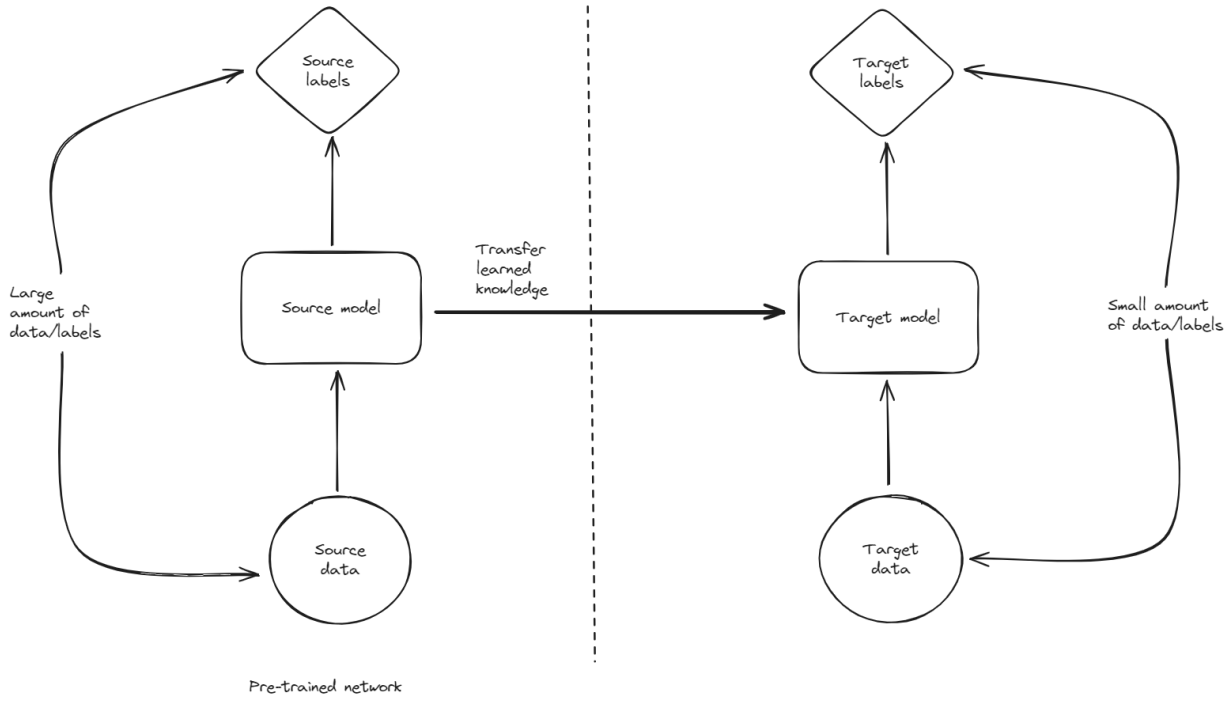


Kuva 11 – Vastavirta-algoritmin toimintaperiaate yhden neuronikerroksen kohdalla. (Glassner 2021m)

Käytännössä vastavirta-algoritmi kirjaa jokaiseen yksikköön (neuroniin) häviön, joka on riippuvainen yksikön painotuksista ja tuottamista virheistä (Szeliski 2021, 280). Kun neuroverkkoa koulutetaan tunnistamaan jokin tietty asia, esine tai muu vastaava, kutsutaan tätä toimintaa mukautetun

neuroverkon kouluttamiseksi tai custom neuroverkon kouluttamiseksi. Tällaisessa tapauksessa kehittäjät alkavat kehittämään omaa tunnistamiseen kykenevää neuroverkkoa joko aivan tyhjästä tai niin sanotusti siirtämällä koulutusta. Tunnistamiseen kykenevää neuroverkkoa harvoin koulutetaan täysin tyhjästä, koska tämä vaatii hyvin paljon koulutusmateriaalia sekä hyvää syväoppimisen arkkitehtuurien ja algoritmien tuntemusta. Tästä syystä useimmiten käytetään koulutuksen siirtämistä, jossa mukautetun neuroverkon koulutuksessa käytetään hyväksi jo aikaisemmin koulutettuja neuroverkkoja, jotka pystyvät tekemään kohteen tunnistusta. (Zhao, Zhou & Chen 2020, 4.)

Koulutuksen siirtämisessä lähestymistapa on se, että koulutettava neuroverkko alustetaan jo aikaisemmin koulutetun neuroverkon painotuksilla, jonka jälkeen koulutettavaa neuroverkkoa hienosäädetään kouluttamalla sitä uudella kyseiseen tunnistustehtävään valikoidulla koulutusmateriaalilla. Koulutuksen siirtämisellä (Kuva 12) on mahdollista säästää hyvin paljon aikaa neuroverkon koulutuksessa. Tutkimuksissa on kuitenkin saatu viitteitä siitä, että aina koulutuksen siirtämisellä ei ole pelkästään positiivisia vaikutuksia. Koulutuksen siirtäminen voi joissain tilanteissa viedä mahdollisesti yhtä paljon aikaa kuin neuroverkon kouluttaminen niin sanotusti tyhjästä, koska mallin hienosäätöön ja neuroverkon ylioppimisen välttämiseen joudutaan käyttämään hyvin paljon aikaa. (Zhao ym. 2020, 5.)



Kuva 12 – Koulutuksen siirtäminen (Mukaillen Nummela 2022, 14; Alom ym. 2019, 45)

Kohteen tunnistuksen yleisimpinä käyttötapoina voidaan pitää esimerkiksi autonomisia ajoneuvoja, valvonta- ja turvallisuusratkaisuja, kuvanhakuratkaisuja tai vaikkapa kehittyneempiä ihmisen ja

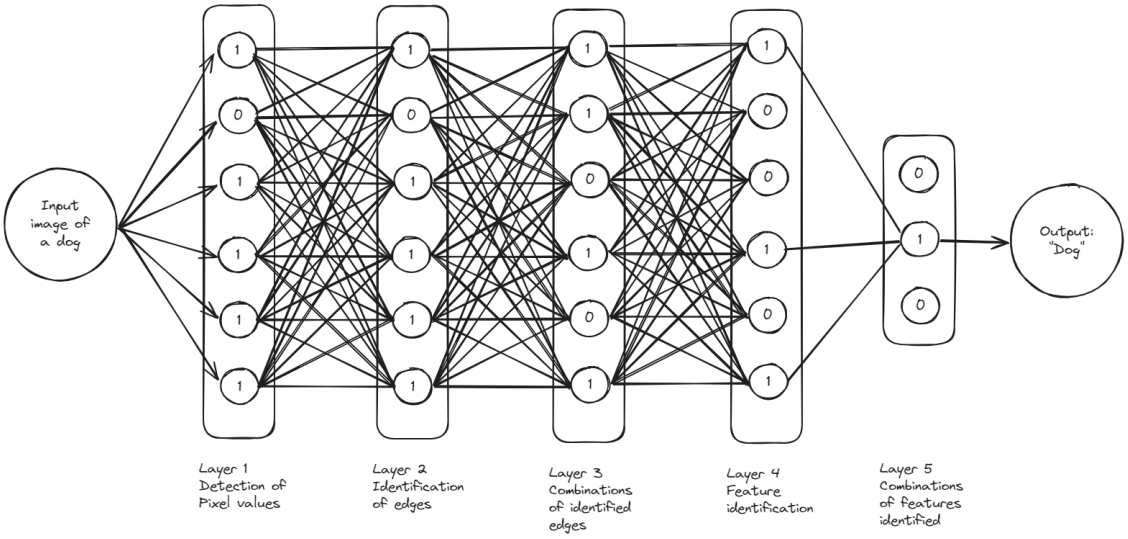
tietokoneen väliseen yhteyden pitoon luotuja järjestelmiä. Jokainen käyttötapa on usein erilaisten muuttujien ja kompromissien summa. Siihen vaikuttavat esimerkiksi tunnistusnopeus, tunnistustarkkuus ja laskentateho. (Tasnim& Qi 2023, 2-3.)

2.4 Kohteentunnistukseen käytettävät neuroverkot

Tässä kappaleessa esitellään kohteentunnistuksessa käytettävien neuroverkkojen keskeisimmät perusteet, modernien neuroverkkojen perusarkkitehtuurit sekä millä menetelmillä neuroverkkojen kyvykkyyksiä voidaan arvioida.

Kohteen löytäminen on olennainen osa tietokonenäköä. Tätä kykyä käytetään erilaisten kohteiden tai kohteiden ominaisuuksien hahmottamiseen joko kuvista tai videolta. Erilaisia kohteentunnistuksen käyttötapoja ovat esimerkiksi kasvojen löytäminen ja kasvontunnistus, kotiautomaatio, itseohjautuvat ajoneuvot, ihmisten laskenta, maatalouden järjestelmät, liikenteenseuranta, sotilas- ja puolustusjärjestelmät, urheilutapahtumat, teollisuus, robotiikka, ilmailuteollisuus sekä lääketiede. (Anand & Divyakant 2020, 2.)

Kohteentunnistus toimii siten, että annettua tehtävää, esimerkiksi kuvaa, verrataan tiedossa olevaan koulutuksen kautta hankittuun tietoon. Tämän avulla pystytään erottelemaan erilaiset kohteet toisistaan sekä nimeämään ja osoittamaan esimerkiksi kuvassa olevat erilaiset kohteet. Kohteentunnistaminen on prosessi, jossa tosielämän kohteita etsitään esimerkiksi kuvista ennalta tiedossa olevien luokkien tai parametrien, kuten autot, moottoripyörät, televisiot, lentokoneet, junat, kukat, hedelmät ja niin edelleen, perusteella (Kuva 13). Kohteentunnistus kykenee ideaalitilanteessa löytämään ja osoittamaan kuvasta useita kohteita. (Anand & Divyakant 2020, 2.)



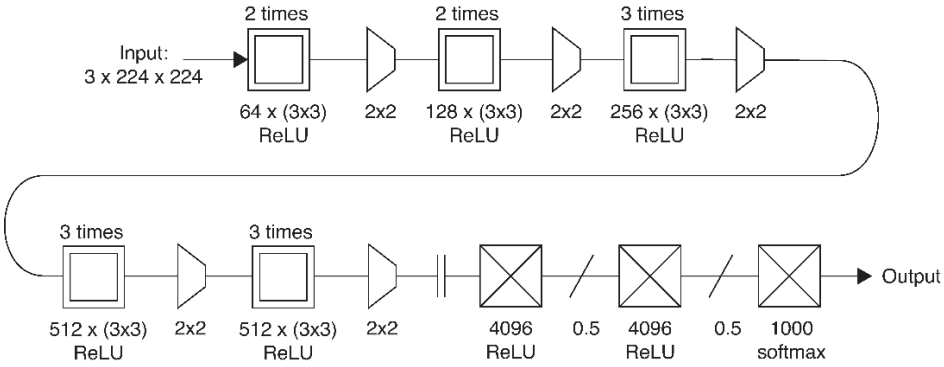
Kuva 13 – Havaintokuva neuroverkon rakenteesta ja tunnistuksen vaiheista (Mukaillen Shukla 2019)

Kohteen löytäminen ja tunnistaminen on ollut viime vuosina tieteellisen tutkimuksen kiinnostuksen kohteena, koska tarve erilaisten tunnistusjärjestelmien käytölle on kasvanut. Tästä syystä useita erilaisia kohteen löytämisen ja tunnistamisen menetelmiä tutkitaan, kehitetään ja käytetään aktiivisesti. Aikaisemmat kohteen löytämisessä ja tunnistamisessa käytetyt tekniikat vaativat paljon käsin tehtävää asiantuntijatyötä, mutta konvoluuttisten neuroverkkojen (Convolutial Neural Networks - CNN) yleistymisen myötä tarve vaativalle käsityölle on vähentynyt merkittävästi. CNN on keskeinen konsepti kohteen löytämisessä kuvasta neuroverkon avulla. Syväoppimiseen kykenevät neuroverkot kykenevät huomattavasti parempaan tunnistustarkkuuteen kuin aikaisemmin käytetyt tunnistustekniikat. (Anand & Divyakant 2020, 2.)

Konvoluuttisten neuroverkkojen (CNN) perusmallia voidaan myös kutsua nimellä VGG16 (Kuva 14). Konvoluuttisten neuroverkkojen kerroksia kutsutaan piirrekartoiksi tai piirrevektoreiksi. Näiden piirrekarttojen tai vektoreiden avulla kuvasta luodaan piirreavaruus, jolle suoritetaan erilaisia suodatus- ja yhdistämistoimenpiteitä. (Anand& Divyakant 2020, 2.; Szeliski 2021, 298.) Kohteen löytämisen tekniikka parantuu huomattavasti, kun tekniikkaan lisätään alueellinen tunnistamisen ominaisuuksia (Region-CNN tai R-CNN), joka toimii käytännössä täysin erilaisella mekanismilla kuin aikaisemmat metodit (Ammar, Koubaa, Ahmed, Saad & Benjdira 2021, 9.) Alueellisen tunnistamiseen kykenevät CNN metodit parantuivat huomattavasti, kun R-CNN, Fast R-CNN ja Faster R-CNN metodit kehitettiin. Regressioluokitteluun liittyvät menetelmät sisältävät muun muassa SSD ja YOLO menetelmät. Nämä menetelmät on suunniteltu siten, että ne analysoivat kohteita kuvista ja antavat ennusteita oikeista vastauksista. (Anand & Divyakant 2020, 2.)

Konvolutiaalinen neuroverkko (Convolutional Neural Network) on yksi syväoppimiseen kykenevistä verkkomuodoista. Konvolutionaalisessa neuroverkossa hyödynnetään monikerros-perceptronin luokitteluominaisuuksia sekä konvoluutio-operaation mahdollistamaa kykyä eri ominaisuuksien erotteluun ja tunnistamiseen. Konvoluuttisessa neuroverkossa käytetään usein erilaisia kerroksia muokkaamaan ja erottelemaan syötteenä saadusta kuvasta erilaisia ominaisuuksia. (Paananen 2018, 12.)

Vaikka modernien neuroverkkojen sisäinen *arkkitehtuuri* voi vaihdella suuresti, ovat ne usein samankaltaisia toistensa kanssa arkkitehtuurisesti. Arkkitehtuurisesti neuroverkot sisältävät usein samankaltaisen kerrosrakenteen, jossa on konvoluuttinen kerros, näytteistyskerros ja luokittelukerros (Kuva 14). Eri arkkitehtuurit koostuvat usein useista konvoluuttisista- ja näytteistyskerroksista, joita seuraa luokittelukerros. (Alom ym. 2019. 14.)



Kuva 14 – VGG16 neuroverkon rakenne (Glassner 2021n)

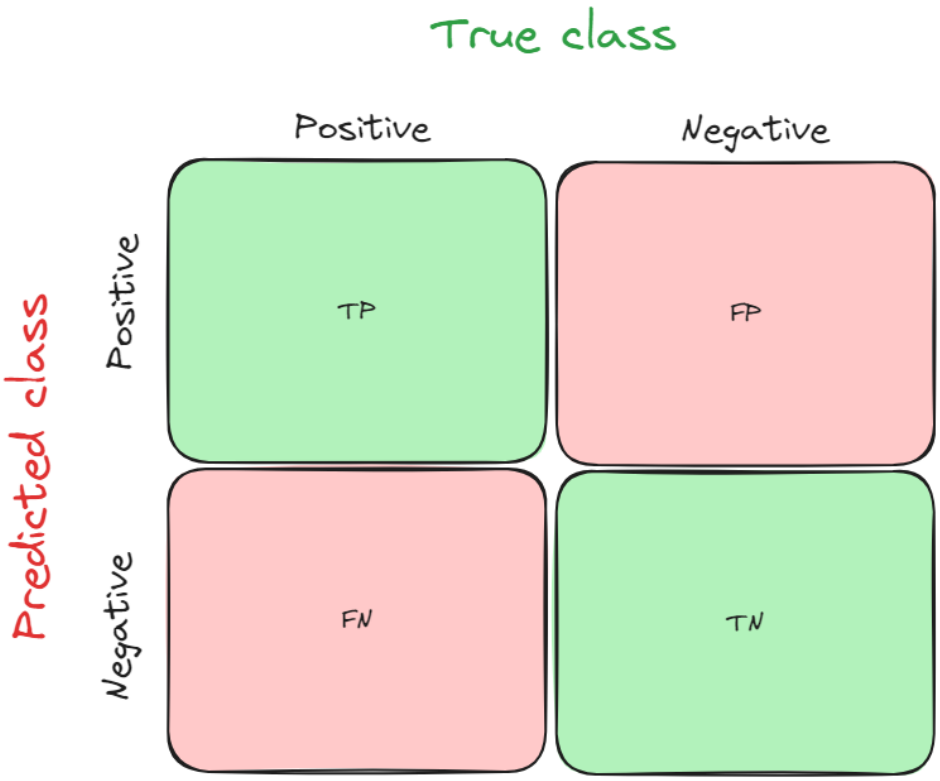
Nykyiset neuroverkkoihin perustuvat kohteentunnistusjärjestelmät voidaan jakaa karkeasti yksi- ja kaksivaiheisiin järjestelmiin. Yksivaiheisia tunnistusjärjestelmiä ovat esimerkiksi You Only Look Once (YOLO) ja sen eri variaatiot, Single Shot MultiBox Detector (SSD) ja sen variaatiot sekä RetinaNet. Yksivaiheiset tunnistusjärjestelmät palauttavat suoraan arvioidun kohteen luokituksen ja paikan esimerkiksi kuvassa. Kaksivaiheisia tunnistusjärjestelmiä ovat esimerkiksi Fast Region-Based Convolutional Network (Fast R-CNN) ja Faster R-CNN. Kaksivaiheiset tunnistusjärjestelmät käyttävät ensimmäisessä ehdotusmenetelmää, jonka perusteella löydetyt piirteet ja kohteet tunnistetaan ja luokitellaan toisessa vaiheessa. Normaalisti kaksivaiheiset järjestelmät ovat tarkempia kohteiden tunnistamisessa verrattuna yksivaiheisiin järjestelmiin. Tarkkuuden kääntöpuolena on kuitenkin huomattava hitaus yksivaiheisiin järjestelmiin verrattuna. (Zhao ym. 2020, 2-3.)

Neuroverkkoja koulutettaessa ja niiden *kyvykkyyksiä* arvioitaessa käytetään usein erilaisten arvojen ja mittausten antamaa tulosta vain yhden lukeman tai arvon sijaan. Keskeisiä tarkasteltavia arvoja ovat Alsharabin (2023, 8-9.) mukaan:

- *Sisäinen tarkkuus* (Precision) ja *herkkyys* (Recall): Sisäinen tarkkuus mittaa oikeiden ennusteiden määrää kaikista tuotetuista ennusteista, kun taas herkkyys mittaa oikeiden ennusteiden määrää oikeista tapauksista.
- *Tarkkuuden keskiarvo* (Average Precision): Mittaa sisäisen tarkkuuden keskiarvoa eri herkkyyden asteilla.
- *Ennusteen virheellisyys* (Intersection over Union, IoU): Mittaa tunnistetun kohteen ennustetun paikan ja todellisen paikan välistä eroa, jonka perusteella voidaan päätellä, onko ennuste oikea- vai vääräpositiivinen tunnistus.
- *Kuvaa sekunnissa* (Frame per Second, FPS): Mittaa prosessointi nopeutta, joka on erittäin tärkeää reaaliaikaisille järjestelmille.
- *Tunnistusaika* (Interference time): Mittaa aikaa kuinka kauan neuroverkolta kestää yhden kuvan kohteiden tunnistamiseen ja luokitteluun.
- *Tarkkuuden keskiarvon* keskiarvo (Mean Average Precision, mAP): Mittaa kaikkien tunnistettavien luokkien tarkkuuden keskiarvoa.
- *Tarkkuus vs. Nopeus* (Accuracy vs. Speed Trade-off): Mittaa neuroverkon tarkkuuden ja prosessointi nopeuden välistä eroa, antaen kuvan kuinka optimaalinen neuroverkko on näiden arvojen suhteen.
- *Toimivuus* (Robustness): Mittaa Kuinka hyvin neuroverkko selviää haastavista tehtävistä, kuten kohteiden tunnistamisesta vääristyneistä tai sotkuisista kuvista.

- *Muistin käyttö* (Memory footprint): Mittaa kuinka paljon neuroverkko käyttää resursseja eli esimerkiksi muistia ja prosessorin tehoa.
- *Tehon käyttö* (Power efficiency): Mittaa kuinka paljon neuroverkko/järjestelmä tarvitsee tehoa (ts. sähköä) toimiakseen, tärkeää sellaisissa järjestelmissä, joissa käytettävä teho on rajallista.

Yksi merkittävistä menetelmistä neuroverkon suorituskyvyn mittaamiseen on sekaannusmatriisin käyttäminen. Käytännössä sekaannusmatriisi voidaan kuvata NxN matriisina (Kuva 15), jossa on esitetty todellisten luokkien ja ennustettujen luokkien oikeat ja väärät arviot. Matriisin avulla pystytään visualisoimaan helposti neuroverkon kykyä tunnistaa eri kohteita. (Karimi 2021, 1.)



Kuva 15 – Sekaannusmatriisi (TP (True Positive): Oikea positiivinen, FP (False Positive): Väärä positiivinen, FN (False Negative): Väärä negatiivinen, TN (True Negative): Oikea negatiivinen) (Mukaillen Karimi 2021, 1)

3 Oman neuroverkon kouluttaminen

Tässä kappaleessa esitellään yleisimpien neuroverkkojen perusteet, miten kohteentunnistusjärjestelmän arkkitehtuuria tulisi tarkastella, sekä tässä työssä käytetyn YOLO-neuroverkon erityispiirteet. Tähän työhön tutkittavaksi valittiin seuraavat kohteen tunnistukseen kykenevät neuroverkot: R-CNN, Fast R-CNN, Faster R-CNN, YOLO, SSD, RetinaNet ja DETR.

Region-based *Convolutional Neural Network (R-CNN)* on tunnistamiseen kykenevä neuroverkko, joka käyttää kaksivaiheista tunnistusprosessia:

1. Alueellinen ehdotus (Region Proposals), jossa algoritmi rakentaa tunnistuslaatikot tunnistettavaan kuvaan, tähän toimintoon käytettään esimerkiksi Selective Search algoritmia.
2. Tunnistus (Identification), jossa jokainen alueellinen ehdotus ajetaan ConvNet (Convolutional Neural Network) verkon läpi, jonka tuloksena verkko antaa löytämänsä luokitukset kyseiseltä alueelta.

Vaikka R-CNN on huomattavasti muita myöhemmin esiteltäviä menetelmiä tarkempi, on tarkkuuden kääntöpuolena hitaus, koska jokainen yksittäinen alueellinen ehdotus ajetaan yksitellen neuroverkon läpi. (Gidaris & Komodakis 2015, 1-2.; Tasnim& Qi 2023, 4.)

Fast R-CNN on kehitetty vastaamaan R-CNN:n hitauden luomiin haasteisiin. Fast R-CNN:ssä on muokattu alkuperäistä neuroverkon arkkitehtuuria siten, että Fast R-CNN käyttää yhteistä ConvNet verkkoa koko kuvan tunnistamiseen. Fast R-CNN käyttää silti alueellisen ehdotuksen menetelmää, mutta sen sijaan että se ajaisi jokaisen alueen yksitellen neuroverkon läpi, Fast R-CNN ajaa koko kuvan kerralla neuroverkon läpi ja tämän jälkeen käyttää Region of Interest (RoI) pooling -tekniikkaa kiinnostavien kohteiden löytämiseen kuvasta. Tämä tekniikka pienentää huomattavasti tarvittavaa laskentaa ja siten nopeuttaa huomattavasti tunnistamista. (Girshick 2015, 1-2.; Tasnim & Qi 2023, 4.)

Faster R-CNN puolestaan parantaa jo aikaisemmin mainittua Fast R-CNN arkkitehtuuria lisäämällä Region Proposal Network:n (RPN) arkkitehtuuriin mukaan. RPN mahdollistaa koko kuvan kohteiden löytämisen yhdellä tunnistuskerralla, jolloin tarvittavan laskennan määrä pienenee entisestään. Käytännössä RPN on ConvNet neuroverkko, joka luo tunnistuslaatikot kohteille ja tunnistaa kohteet samalla kerralla. (Chaudhuri 2023, 6-7.; Tasnim & Qi 2023, 4.)

YOLO (You Only Look Once) lähestyy tunnistamisen haastetta hieman eri lailla kuin R-CNN mallit. YOLO arkkitehtuurissa kuva jaetaan ruudukoihin, ja jokaisen ruudun solun paikannuslaatikot ja tunnistukset luodaan yhdellä neuroverkon tunnistuskerralla. Tämä menetelmä tekee YOLO:sta hyvin nopean, jopa niin nopean että se kykenee lähes reaaliaikaiseen tunnistamiseen. Vaikka YOLO

ei ollut yhtä tarkka kuin R-CNN mallit silloin kun se julkaistiin, sen kehitystä on jatkettu ja tämän kehityksen ansiosta se on hyvin varteenotettava tunnistusmalli nopeutensa ja tarkkuutensa ansiosta. (Redmon, Divvala, Girshick & Farhadi 2016, 1-2.; Tasnim & Qi 2023, 4.)

Työn toiminnallisessa osassa käytetään YOLO arkkitehtuurin mukaista neuroverkkoa ja neuroverkon koulutinta, joka on kuvattu tarkemmin kappaleessa 3.2.

SSD on yksivaiheista tunnistusta käyttävä menetelmä, jossa on poistettu tarve käyttää erillistä alueellisen ehdotuksen mekanismia. Se yhdistää kohteenluokituksen ja kohdistuslaatikon muodostuksen yhteen kuvankäsittelykertaan. SSD koostuu useista konvoluuttisista kerroksista, joissa on käytetty erikokoisia tunnistusalueita erilaisten kohteiden löytämiseksi. Tämän johdosta SSD on nopeampi tunnistamaan kohteita kuin kaksivaiheista tunnistusta käyttävät järjestelmät. Nopeuden kääntöpuolena on kuitenkin kaksivaiheisia tunnistimia huonompi tunnistustarkkuus varsinkin pienten kohteiden osalta. (Tasnim & Qi 2023, 4.)

RetinaNet on yksivaiheista tunnistusta käyttävä menetelmä, joka pyrkii vastaamaan korostuneiden ja taustalla olevien kohteiden haasteeseen käyttämällä polttopisteen häviön (Focal Loss) menetelmää. Tämä menetelmä auttaa neuroverkkoa seulomaan koulutuksen aikana helpot negatiiviset tapaukset pois, jolloin neuroverkko voi keskittyä koulutuksen aikana haastavampiin positiivisiin tunnistuksiin. RetinaNet käyttää myös Feature Pyramid Network (FPN) menetelmää käsitelläkseen erikokoisia tunnistettavia kohteita, täten saavuttaen kohtuullisen hyvän tarkkuuden erilaisten kohteiden osalta. (Tasnim & Qi 2023, 5.)

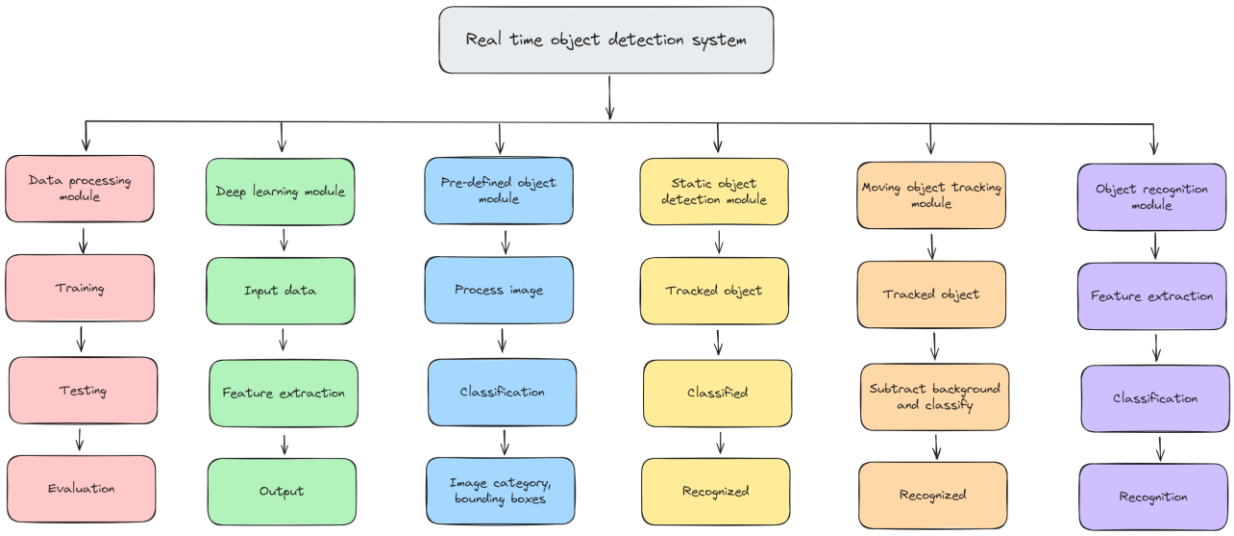
DETR on tunnistusmenetelmä, joka käyttää muuntajia mallintamaan eri kuvan alueiden välisiä relaatioita. DETR käsittelee kohteen tunnistusta ennusteongelmana, jossa tavoitteena on ennustaa tietty määrä tunnistuslaatikoita ja niitä vastaavat luokitukset. DETR arkkitehtuuri koostuu CNN rangasta, jota seuraa muuntajien joukko, joka käsittelee kuvan piirteet ja tekee kohteen haun. DETR menetelmässä ei käytetä ollenkaan ankkurointilaatikoita, non-maximum suppression (NMS) menetelmää tai heuristisia menetelmiä, joita perinteisesti käytetään kohteentunnistusmenetelmissä. Vaikka DETR kykeneekin erittäin vertailukelpoisiin tarkkuuksiin, sen selkeänä huonona puolena on tunnistamisen hitaus, varsinkin verrattuna yksivaiheisiin tunnistimiin, kuten YOLO:on tai SSD:hen. (Tasnim & Qi 2023, 5.)

Liitteessä 1 – (Eri tunnistusmenetelmien välinen kyvykkyys vertailu) on esitelty tarkemmin eri tunnistusmenetelmien välisiä eroja käytetyn kehyksen, nopeuden, tarkkuuden ja tärkeimpien ominaisuuksien suhteen.

3.1 Kohteentunnistusalgoritmi osana ohjelmistoarkkitehtuuria

Ohjelmistoarkkitehtuurin tehtävänä on kuvata sitä, miten ohjelmisto voidaan purkaa pienempiin osiin, miten nämä osat on liitetty yhteen ja miten osat kommunikoivat keskenään. Huonosti tai väärin ymmärretty arkkitehtuuri on hyvin usein syynä erilaisiin ohjelmistovirheisiin tai ongelmiin. Applikaatio- ja alustatason ohjelmistolle käytetään erilaisia arkkitehtuurisia rakenteita, riippuen hieman missä vaiheessa ohjelmistokehitystä ollaan. Nämä rakenteet voidaan jakaa karkeasti seuraaviin luokkiin: Konseptuaalinen arkkitehtuuri, kommunikaatioarkkitehtuuri, suoritusarkkitehtuuri ja koodiarkkitehtuuri. Jokainen luokka kuvaa järjestelmää hieman erilaisesta näkökulmasta. (Soni, Nord & Hofmeister 1995, 1-5.)

Kohteentunnistusjärjestelmän pääasiallinen tarkoitus on tunnistaa kohde tai kohteita kuvasta tai videolta sekä esittää tunnistus jollakin menetelmällä. Lisäksi toiminnalliset määrittelyt kuvaavat mitä järjestelmän odotetaan tekevän tai miten sen odotetaan toimivan. Kohteentunnistusjärjestelmällä voidaan olettaa olevan ainakin seuraavat oleelliset toiminnalliset osa-alueet: tiedonkäsittelymoduuli, syväoppiva moduuli, staattisen kohteen tunnistusmoduuli, liikkuvan kohteen tunnistusmoduuli, ennalta-määritetyn kohteen moduuli ja kohteen luokitusmoduuli (Kuva 16). Järjestelmän perusperiaatteena on se, että järjestelmä saa syötteenä kameran ottaman staattisen tai liikkuvan kuvan, vertaa saatua syötettä sille koulutettuun materiaaliin, hakee koulutusmateriaalista syötettä vastaavan luokituksen ja luo kohteen ympärille tunnistuslaatikon ja lähettää tunnistustuloksen rajapintaan. (Ullah, Wahab, Choi & Khan 2022, 4.)



Kuva 16 – Toiminnallisen määrittelyn mukaiset järjestelmämoduulit (Mukaillen Ullah ym. 2022, 5)

3.2 You Only Look Once (YOLO)

Tämän opinnäytetyön toiminnallisessa osiossa käytetty You Only Look Once (YOLO) arkkitehtuurin mukainen kohteentunnistus toimii hieman erilaisella periaatteella kuin R-CNN. R-CNN käyttää alueellisen ehdotuksen (region proposal) metodia luodakseen mahdolliset tunnistuslaatikot kuvaan ja tämän jälkeen luokitella laatikoissa mahdollisesti olevat kohteet. Luokittelun jälkeen suoritetaan jälkikäsittely, jossa tunnistuslaatikoiden koko säädetään mahdollisimman todenmukaiseksi, poistetaan päällekkäiset tunnistuslaatikot ja uudelleen pisteytetään eli luokitellaan jäljelle jääneet tunnistuslaatikot. Tämä prosessi on monimutkainen ja siten hidas. Lisäksi se on vaikea optimoida, koska jokainen yksittäinen komponentti pitää kouluttaa erikseen. (Redmon ym. 2016, 1.)

YOLO arkkitehtuurin mukainen järjestelmä kykenee kouluttamaan mallin kokonaisilla kuvilla ja suoraan optimoimaan tunnistukseen vaatiman laskennan. Tällaisella yhtenäistetyllä mallilla on monia etuja verrattuna muihin kohteentunnistusarkkitehtuureihin. YOLO arkkitehtuurissa kohteen tunnistus ja luokittelu on ajateltu yksittäisenä regressio-ongelmana, jolloin tarvetta monimutkaiselle usean kanavan järjestelylle ei enää ole. YOLO arkkitehtuurissa neuroverkon läpi syötetään uusi kuva, jolloin samalla luodaan ennuste tunnistuksista. Tämä mahdollistaa sen, että YOLO mallit kykenevät tunnistamaan kohteita reaaliaikaisesti esimerkiksi videoista alle 25 ms latenssilla. Lisäksi YOLO mallit kykenevät hyvin korkeaan keskivertotarkkuuteen (mean average precision) verrattuna moniin muihin reaaliaikaisiin tunnistusjärjestelmiin. (Redmon ym. 2016, 1.)

Lisäksi YOLO malli on kykenevä päättelemään tunnistettavasta kuvasta myös kontekstuaalisia vihjeitä. Tämä onnistuu siksi, koska YOLO malli ’näkee’ koko kuvan koulutuksen ja testauksen aikana tallentuu kohteiden mukana niihin liittyvää kontekstuaalista informaatiota. Kontekstiymmärryksen ansiosta YOLO mallit tekevät yleisesti puolet vähemmän virheellisiä tunnistuksia taustakohinaan verrattuna esimerkiksi Fast R-CNN malleihin. Näistä syistä johtuen YOLO mallit ovat yleensä hyvin käyttökelpoisia myös tosielämän tilanteissa. YOLO mallien yleisenä heikkoutena voidaan pitää pienten kohteiden kuvasta löytämisen vaikeutta ja täten myös näiden kohteiden tunnistamisen vaikeutta (Redmon ym. 2016, 1.).

YOLO neuroverkon arkkitehtuurissa perustana on konvoluutioneuroverkko, jossa verkon ensimmäiset kerrokset hakevat kuvasta pääpiirteet ja täysin yhdistetyt myöhemmät kerrokset hakevat löydetyille piirteille todennäköisyydet ja paikkakoordinaatit. Alun perin YOLO neuroverkon arkkitehtuuri perustui muokattuun GoogLeNet mallin arkkitehtuuriin, jossa alkuperäisen 22 kerroksen sijaan käytettiin 24 konvoluuttista kerrosta ja kahta täysin yhdistettyä kerrosta. (Redmon ym. 2016, 2-3.)

Perusarkkitehtuurin lisäksi YOLO arkkitehtuuriin suunniteltiin vielä kevyemmän arkkitehtuurin neuroverkko nimeltään YOLOv1-tiny (Liite 2), jonka tarkoitus oli mahdollistaa erittäin nopea kohteen tunnistus. Tässä arkkitehtuurissa käytettiin alkuperäistä vähemmän konvoluuttisia kerroksia (yhdeksää kerrosta alkuperäisen 24 sijaan), sekä myös pienempää määrää suodattimia näissä kerroksissa. YOLO arkkitehtuurin mukaisesti neuroverkosta saadaan ulos 7x7x30 ennuste (prediction) muuttuja (tensor). (Redmon ym. 2016, 3.)

YOLO v3:n perusajatuksena oli parantaa aikaisemmin luotua arkkitehtuuria erilaisissa tutkimuksissa ja julkaisuissa esitettyjen parannusten avulla (Redmon & Farhadi 2018, 1). Tässä kappaleessa on esitetty oleellisimmat YOLOv3 – arkkitehtuurin erityispiirteet muihin neuroverkkoihin verrattuna, nämä piirteet ovat: Täsmällisen tunnistuslaatikon kehitys, tunnistusluokkien ennustaminen ja tunnistusten tekeminen eri luokkien välillä.

YOLO9000 arkkitehtuurissa *tunnistuslaatikoiden* paikkaa ennustettiin eri ulottuvuuksien ryhmiin perustuvina. Tämä toimi siten, että neuroverkko ennusti neljä koordinaattia jokaiselle tunnistuslaatikolle (t_x, t_y, t_w, t_h). Kun tähän tietoon yhdistetään ennustetun laatikon asema (c_x, c_y) suhteutettuna kuvan vasempaan yläreunaan, pystytään ennustetun laatikon muoto ja asema ennustamaan seuraavan kaavan (Kuva 17) mukaisesti. (Redmon & Farhadi 2018, 1.)

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Kuva 17 – Tunnistuslaatikon geometrian laskenta (Mukaillen Redmon & Farhadi 2018, 1)

Mallin koulutuksen aikana mallin hyvyyttä mitattiin virhehäviöiden neliöjuurien summana. YOLOv3 malli ennustaa jokaiselle objektiivisuus pisteytyksen käyttäen logistista regressiota. Tämä pisteytys luku on yksi (1) silloin kun ennustettu tunnistuslaatikko peittää asetetun ’totuuslaatikon’ (gound truth object) täydellisemmin kuin aikaisemmat tunnistuslaatikot. Mikäli aikaisempi tunnistuslaatikko ei peitä todellista totuuslaatikkoa paremmin kuin ennustettu tunnistuslaatikko, vertaillaan ylittääkö aikaisempi tunnistuslaatikko asetetun raja-arvon (esimerkiksi 0.5) jolloin uusin tunnistus automaattisesti hylätään. Jokaiseen edellä mainittuun tunnistuslaatikkoon luodaan *ennuste tunnistettavista kohteista*, jotka kyseessä oleva laatikko sisältää. Muista kohteentunnistusjärjestelmistä eroten YOLO arkkitehtuurissa ei käytetä viimeisenä kerroksen softmax kerrosta, vaan itsenäisiä

logiikkaan perustuvia luokittelijoita. Mallin koulutuksen aikana arkkitehtuuri käyttää binääristä ristientropian häviöön (binary cross-entropy loss) perustuvaa luokituksen määritystä. Tämä menetelmä on usein hyvä silloin, kun mallille koulutetaan monimutkaisempia tunnistushaasteita. Esimerkiksi softmax:ia käytettäessä oletus on, että jokainen tunnistuslaatikko sisältää vain yhden tunnistusluokan, eikä limittäisyyttä esiinny. Todellisuudessahan näin ei ole, jolloin ristientropiaan perustuvan luokituksen käyttö on perusteltavaa. (Redmon & Farhadi 2018, 1.)

YOLOv3 arkkitehtuuri luo tunnistuslaatikoita kolmella eri skaalalla. Järjestelmä käyttää eri piirteiden havaitsemiseen pyramidiverkkojen konseptia, samalla tavalla kuin R-CNN arkkitehtuurissakin. Näiden piirteiden hakemisen jälkeen YOLO käyttää useita konvoluuttisia kerroksia sopivien tensoreiden muodostamiseen ja tunnistusten löytämiseen. Viimeisestä kerroksesta saadaan tuloksena ulos 3D-tensori, joka sisältää tunnistuslaatikon, hyvyysarvion ja luokituksen, tämä auttaa tekemään *tunnistuksen eri luokkien välillä*. Tämän jälkeen järjestelmä hakee aikaisemmista konvoluuttisista kerroksista piirrekartat (feature-map), yhdistelee ja suurentaa saadun tuloksen kaksinkertaiseksi. Tämä metodi auttaa saamaan semanttista oheistietoa tunnistuksen tueksi. Suurentamisen jälkeen saatu tensori ajetaan jälleen useiden konvoluuttisten kerrosten läpi, jonka tuloksena saadaan jälleen tensori, jonka koko on nyt kaksinkertainen alkuperäiseen verrattuna. YOLOv3 käyttää uudelleen suunniteltua runkoverkkoa (backbone) lopullisten tunnistusten hakemiseen, jossa käytetään useita peräkkäisiä 3x3 ja 1x1 konvoluuttisia kerroksia, sekä oikoreittejä erityispiirteiden tunnistamiseen. Tämä runkoverkko on nimeltään Darknet-53. Edellä mainitun runkoverkon etuna aikaisempaan on sen tehokkuus ja nopeus. Näistä syistä esimerkiksi mallin koulukseen käytettävän grafiikkakiihdyttimen resurssien käyttö on tehokkaampaa. (Redmon & Farhadi 2018, 2-3.)

YOLOv4 perustuu alkuperäiseen YOLO arkkitehtuuriin ja siitä paranneltuun v3 arkkitehtuuriin. Tässä versiossa on kuitenkin useita merkittäviä parannuksia, jotka tehostavat tämän mallin toimintaa ja koulutusta merkittävästi edellisiin verrattuna. (Bochkovskiy, Wang & Liao 2020, 1.) Yksi suunnittelun perusteista oli luoda nopeasti toimiva kohteentunnistin tuotantojärjestelmiä varten sekä optimoida samanaikaisten rinnakkaisten laskentojen määrää. Myös mallin koulutuksen helpottaminen sekä mahdollisuus käyttää kaupallisesti saatavilla olevia grafiikkakiihdyttimiä mallin kouluttamiseen ovat olleet tärkeitä periaatteita tämän mallin suunnitellussa. (Bochkovskiy ym. 2020, 1.) Tässä kappaleessa esitellään keskeisimmät muutokset YOLOv4:n ja YOLOv3:n välillä, joita ovat datamanipulaatio ja arkkitehtuurimuutokset.

Toteutuksessa yhtenä merkittävänä osana on ollut *datamanipulaation* toteuttaminen järjestelmään. Tämän tarkoituksena on lisätä koulutusmateriaalin monimuotoisuutta, jotta koulutettava malli kykenisi tunnistamaan paremmin eri lähteistä saatuja materiaaleja. Esimerkiksi kuvametrinen käsittely ja geometriset käsittelyt ovat kaksi ehkä yleisimmin käytettyä datamanipulaation menetelmää, jotka

auttavat koulutusmateriaalin rikastamisessa. Kuvametrisessä käsittelyssä kuvan kirkkautta, valoisuutta, kontrastia, saturaatiota ja/tai rakeisuutta muutetaan, jotta tuloksena on hieman alkuperäisestä erilainen kuva. Geometrisessa käsittelyssä voidaan muuttaa esimerkiksi kuvan kokoa, leikata kuvaa, kääntää tai pyörittää kuvaa, jolloin taas saadaan tuloksena alkuperäisestä muuttunut ’uusi’ kuva. Muita kehityksen kohteina olleita osa-alueita oli esimerkiksi semanttisen epätasapainon ongelman mitätöinti koulutusmateriaalissa. Tätä ongelmaa voidaan lähestyä monenlaisten eri ratkaisuiden kautta, mutta tässä toteutuksessa sovelletaan polttopistehäviön ja luokitusverkon tiedonjakamisen periaatteita. Edellä mainittujen lisäksi tunnistuslaatikoiden muodon ja paikan laskentaan on tuotu uusi muuttuja (IoU loss), jonka avulla pystytään häivyttämään aikaisemmissa versioissa ollut ongelma, jossa häviöt kasvavat, kun tunnistuksia uudelleen skaalataan neuroverkkojen välillä. (Bochkovskiy ym. 2020, 2-3.)

Lisäksi YOLOv4 *arkkitehtuuria* on muutettu aikaisemmasta siten, että tavoitteena on ollut parantaa mallin tiettyjä ominaisuuksia, kuten havaintoaluetta, kohteiden piirteiden havaitsemista sekä jälkikäsittelyä (Bochkovskiy ym. 2020, 4). Moduulit, joita käytetään reseptiivisessä havainnoinnissa, on määritelty uudestaan. Tässä versiossa voidaan käyttää erikoismoduuleita kuten SPP, ASPP ja RFB, joita käytetään määrittelemään, kuinka suuren osan kuvasta neuroverkko näkee ja prosessoi kerrallaan. Uusina moduuleina ASPP, joka käyttää kuvan tarkasteluun konvoluutioita eri suodatuksien avulla, jolloin neuroverkko ikään kuin näkee kuvan kauempaa. RFB tekee samaa asiaa, mutta vielä tehokkaammin ja antaa mahdollisuuden entistä suurempaan kuvan tarkasteluun. Tämä auttaa neuroverkkoa tunnistamaan erikokoisia kohteita entistä helpommin, ilman kovinkaan suurta laskentatehon lisäystä. Huomio moduuleissa käytetään kahta tekniikkaa, jotka helpottavat mallia näkemään ja keskittymään tärkeimmät osat koulutusmateriaalista, samaan tapaan kuin ihmiset kiinnittävät huomiota tiettyihin asioihin ympärillään. Kanavan laajuinen huomio (kuten SE) keskittyy erilaisten piirteiden huomioimiseen, kun taas pistemäisen huomion moduulit (kuten SAM) hakevat tiettyjä pieniä yksityiskohtia koulutusmateriaalista. Piirteiden yhdistelyssä käytettiin aikaisemmin melko suoraviivaisia menetelmiä matalan tason ja korkean tason yhdistelyyn. Tässä arkkitehtuurissa käytetään monitasoisia ennuste tekniikoita, kuten SFAM, ASFF ja BiFPN näiden piirteiden löytämiseen ja yhdistelyyn. (Bochkovskiy ym. 2020, 4.)

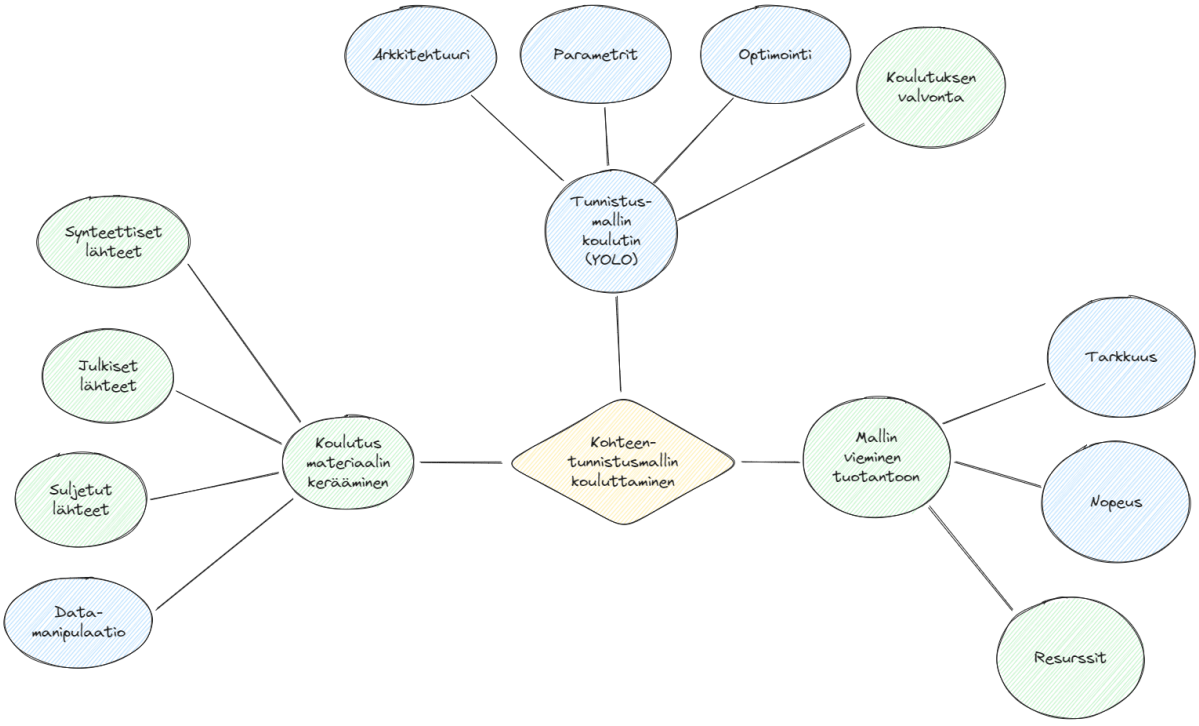
Aktivointifunktiot ovat uudelleen määritelty YOLOv4 arkkitehtuurissa. Nämä funktiot määrittelevät milloin neuroverkon yksittäinen neuroni pitäisi aktivoida, jolloin mallin koulutuksessa tarvittavien gradienttien laskeminen helpottuu huomattavasti. Arkkitehtuurin mukaisesti voidaan käyttää joko perinteistä ReLU aktivointia tai sen muita variantteja (kuten LReLU ja PReLU) estämään mallin koulutuksen hidastumista silloin kun mallia on koulutettu useita koulutuskierroksia. Jälkikäsittelymenetelmiä on edelleen kehitetty, siten että sen jälkeen, kun neuroverkko on luonut karkean arvion tunnistettavista kohteista, suodatetaan päällekkäiset tunnistuslaatikot pois ja pidetään kaikkein

paras tunnistuslaatikko, lisäksi parhaan tunnistuslaatikon laskentaa on päivitetty käyttämällä Soft NMS ja DIoU NMS menetelmiä. (Bochkovskiy ym. 2020, 5.)

3.3 Tietoperustan käyttäminen empiirisessä osassa

Tämän toiminnallisen opinnäytetyön tarkoituksena on kuvata tarvittavaa järjestelmäarkkitehtuuria, kun tarkoituksena on oman kohteentunnistusmallin kouluttamiseen, siten että koko prosessi kartoitetaan aina tarvittavan koulutustiedon keräämisestä aina lopullisen mallin jalkauttamiseen kohdearkkitehtuuriin. Edellisissä kappaleissa kuvattu tietoperusta luo omalta osaltaan perustukset tämän arkkitehtuurin rakentamiselle siten, että tietoperustassa kuvataan tekniset vaatimukset, sekä menetelmät oman kohteentunnistus mallin kouluttamiseen ja käyttämiseen. Tämä on erittäin tärkeä osa kokonaisuutta, koska ilman sitä ei luonnollisesti koko tunnistusmallia voitaisi kouluttaa.

Tietoperusta ei kuitenkaan kuvaa sitä kokonaisuutta mitä kaikkea oikeasti vaaditaan, kun oman kohteentunnistusmallin koulutusta suunnitellaan, aina koulutusmateriaalin keräämisestä ja mallin viemisestä tuotantoon. Tämän kokonaisuuden kartoittamiseen tällä työllä pyritään vastaamaan. Kuvassa 18 on esitetty yksinkertaistettuna Mindmap -karttana tärkeimmät osiot kohteentunnistusmallin luomisesta. Kuvassa sinisellä värillä merkitty osiot ovat niitä, joihin tietoperusta antaa vastauksia, vihreällä värillä osiot, joita jouduttiin tutkimaan/kehittämään työn aikana.



Kuva 18 – Mindmap kohteentunnistusmallin luomisesta.

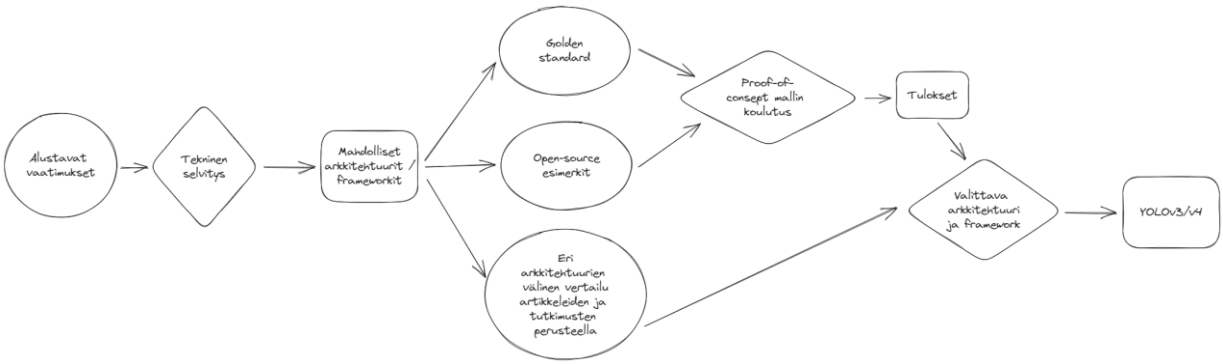
3.4 Toteutuksen suunnittelu

Tässä kappaleessa esitellään asiakkaan lähtötilanne työlle ja esitellään työn suunnittelun prosessi. Asiakkaana tällä lopputyölle on suomalainen pirkanmaalainen ohjelmistotalo X, joka toimii teollisuuden, puolustus, ohjelmistokonsultoinnin ja kyberturvallisuuden aloilla.

Asiakkaan tarve työlle oli kaksi osainen:

1. Luoda kyky tunnistaa ajoneuvojen tyyppi ja liikesuunta liikennekamerasta saatavasta kuvasta riittävällä tarkkuudella
2. Rakentaa kyky kouluttaa omia kohteiden tunnistusmalleja paikallisessa ympäristössä, josta ei ole yhteyttä internettiin.

Lähtötilanteessa kartoitettiin asiakkaan X tarpeita ja tavoitteita kohteidentunnistusjärjestelmälle. Tämän jälkeen, kun alustavat vaatimukset ja tekniset rajoitteet olivat selvillä, aloitettiin erilaisten teknisten ratkaisuiden kartoittaminen. Referenssinä käytettiin muutamia erilaisia olemassa olevia palveluita (Roboflow, Ultralytics), joiden avulla selvitettiin millaisia mahdollisia teknisiä haasteita tai rajoitteita toteutuksessa voisi tulla eteen. Lisäksi tehtiin tutkimusta erilaisista kuvantunnistusmenetelmistä, arkkitehtuureista, joiden avulla järjestelmä voitaisiin toteuttaa sekä millaisia kuvamääriä tunnistusmallin kouluttaminen voisi vaatia. Kuvassa 19 on esitetty periaatekuvaus prosessista, jonka avulla lopullisessa toteutuksessa käytetty arkkitehtuuri ja framework valittiin.



Kuva 19 – Periaatekuva prosessista, jolla käytettävä arkkitehtuuri ja framework valittiin toteutukseen.

Alustavan tutkimuksen perusteella käytettäväksi arkkitehtuuriksi valikoitui YOLO (You Only Look Once) arkkitehtuurin mukainen järjestelmä käytännössä kolmesta syystä:

1. Nopeus – YOLO arkkitehtuurin mukainen tunnistin on huomattavasti nopeampi kuin muut vastaavat kohteen tunnistusarkkitehtuurien mukaiset järjestelmät, kuitenkin siten että tarkkuus on riittävällä tasolla nopeuteen suhteutettuna

2. Esimerkkejä saatavilla suhteellisen helposti – YOLO arkkitehtuuriin nojaavia erilaisia avoimen lähdekoodin kouluttimia ja tunnistimia on helposti saatavilla avoimista lähteistä.
3. YOLO arkkitehtuurin mukainen ’Golden standard’ saatavilla – YOLO arkkitehtuuriin nojaava tuotantovalmis järjestelmä (Ultralytics) on saatavilla, jolloin voidaan tehdä vertailua käytettävän arkkitehtuurin yhteen parhaista esimerkeistä.

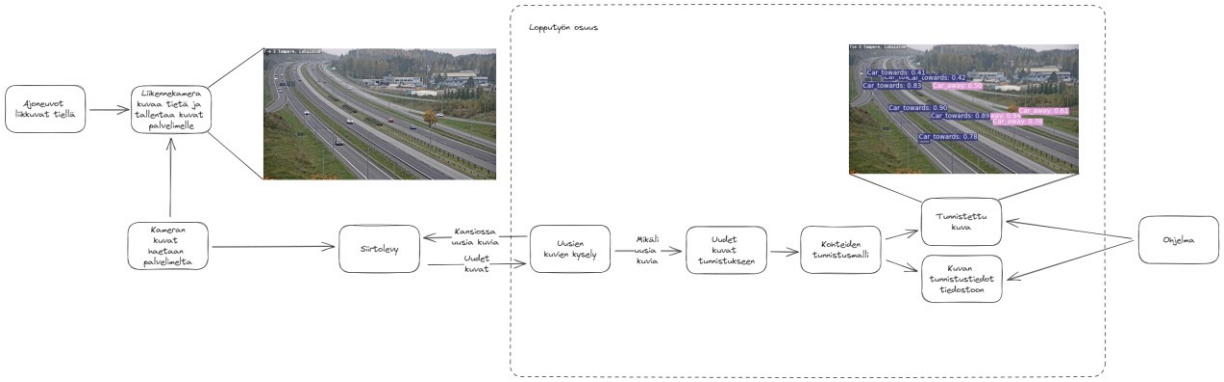
Kun oli saavutettu riittävä tekninen ymmärrys siitä mitä pitäisi tehdä ja siitä että asiakkaan vaatimukset täyttävä järjestelmä olisi teknisesti mahdollista toteuttaa, aloitettiin toiminnallisen määrittelyn kirjoittaminen, johon kirjattiin käytettävä tunnistustekniikka, arkkitehtuuri, vaatimukset, rajoitteet ja riskit.

3.5 Toteutuksen tavoitteiden kartoitus ja sopiminen

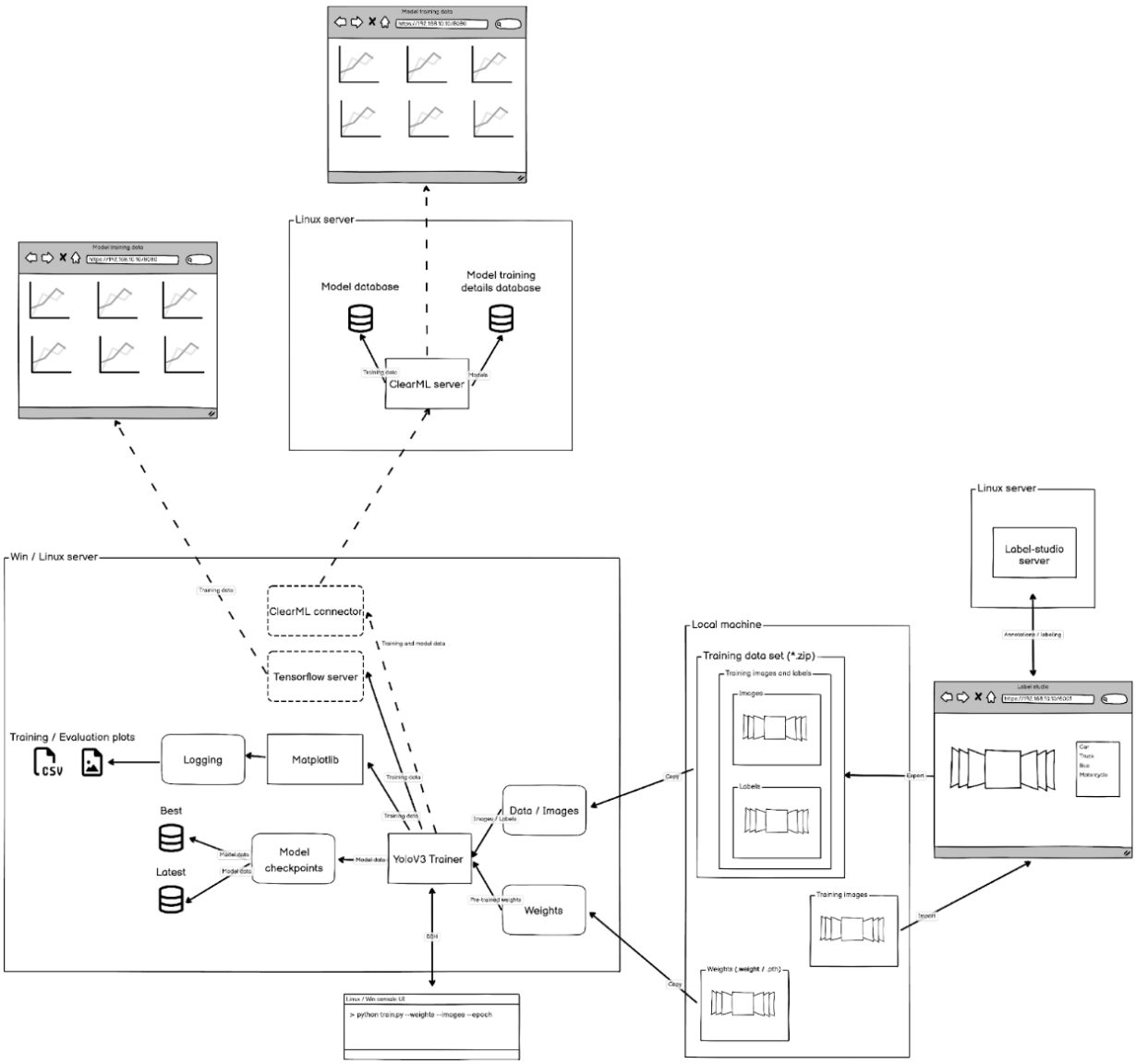
Tehtävälle toteutukselle kirjattiin toiminnallisessa määrittelyssä seuraavat tavoitteet:

1. Mallin on kyettävä tunnistamaan kohteet ja niiden aspektit/liikesuunnat vähintään mAP_0.5>=65% tarkkuudella.
2. Mallin on pystyttävä tunnistamaan kohteet f1 -arvolla >= 0.65%
3. Mallin on pystyttävä luokittelemaan ajoneuvot seuraavien kategorioiden mukaisesti:
    a. auto/pakettiauto (car)
    b. rekka (truck)
    c. bussi (bus)
    d. moottoripyörä (motorcycle)
    e. maatalouskone (farm vehicle)
4. Mallin on pystyttävä tunnistamaan ajoneuvon suunta/aspekti seuraavien kategorioiden mukaisesti:
    a. away
    b. towards
    c. left
    d. right
5. Mallin on pystyttävä tunnistamaan ja luokittelemaan y kuvaa x minuutissa
6. Mallin kouluttamisessa käytetyt avoimen lähdekoodin työkalut dokumentoidaan ja mikäli näitä työkaluja on muokattu tai päivitetty työn mahdollistamiseksi, tehostamiseksi tai muutoin, toimitetaan päivitetyt versiot asiakkaalle.
7. Kuvantunnistusmallin koulutinta on pystyttävä käyttämään myös suljetussa ympäristössä, jossa ei ole internet yhteyttä.

Kuvassa 20 on esitetty korkeantason periaatekuvaus kohteentunnistuksen prosessista asiakkaan kohdearkkitehtuurissa ja kuvassa 21 on esitetty periaatekuvaus kohteentunnistukseen kykenevän neuroverkon kouluttimen arkkitehtuurista.



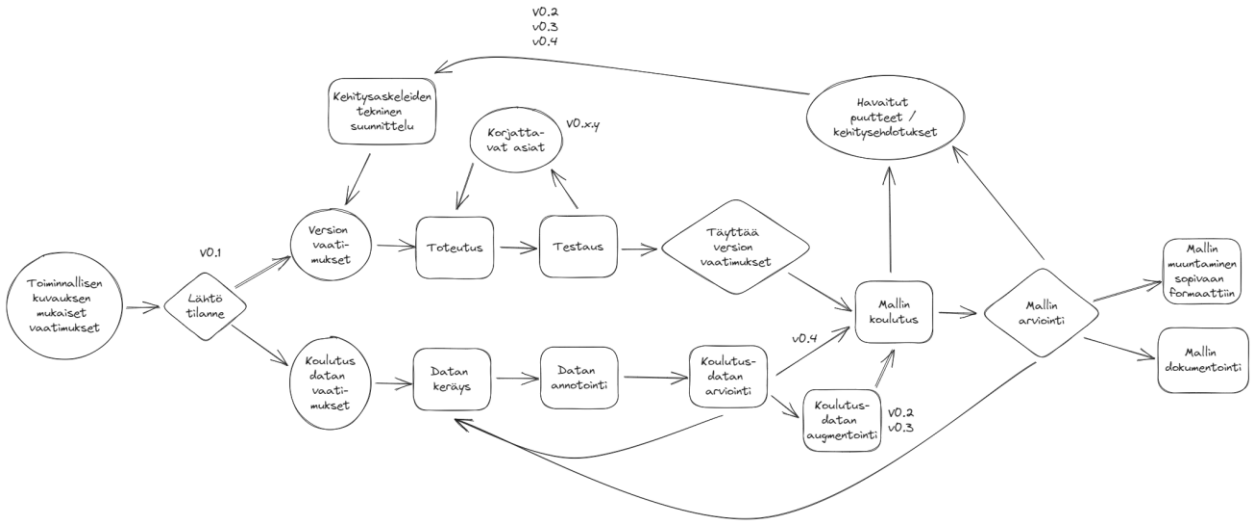
Kuva 20 – Periaatekuva kohteentunnistuksen prosessista asiakkaan kohdearkkitehtuurissa.



Kuva 21 – Periaatekuva tunnistusmallin kouluttimen arkkitehtuurista

3.6 Työn toteutusprosessi

Käytännössä kohteentunnistusmallin kouluttimen ja mallin kouluttamisen prosessi muotoutui siten, että ajatuksena oli kehittää mallin koulutinta ja mallia askeleittain eteenpäin mukaillen ketterää kehitysmallia (Kuva 2222). Uusia ominaisuuksia ja mallin koulutusmateriaalia tuotettiin sitä mukaa kun kokemusta karttui ja mahdollisia puutteita tuli esiin. Samalla ajatuksena oli pyrkiä tekemään toteutus ’pienimmän vaivan’ periaatteella, siten että kehitys lopetetaan heti kun mallin minimivaatimukset saavutetaan. Kun prosessia suunniteltiin, oli arvattavissa, että vaatimusten mukaiseen toteutukseen tarvittaisiin enemmän kuin yksi kehityskierros, mutta kehityskierrosten lopullista määrää ei osattu oikein arvioida. Suunnitteluvaiheessa paras arvaus oli, että kehityskierroksia voitaisiin tarvita ainakin kolme-kuusi kappaletta.



Kuva 22 – Periaatekuva tunnistusmallin kouluttimen kehitysprosessista ja mallin kouluttamisen prosessista.

Ensimmäisen kehitysvaiheen tavoitteina oli luoda hyvin yksinkertainen koulutussilmukka, jossa ei ole juurikaan mitään ylimääräisiä työkaluja tai kuvankäsittelymahdollisuuksia. Ainoana poikkeuksena toteutettiin tuki erikokoisille kuville, jotta koulutusmateriaalin kuvia ei tarvitsisi erikseen muuntaa sopivaan kokoon koulutinta varten. Ensimmäisen vaiheen kouluttimen pääasialliset tavoitteet olivat löytää sopivat tukityökalut koulutusmateriaalin nimikointia ja muuntamista varten, sekä prosessit tarvittavien kirjastojen hakemiseen, lataamiseen ja tuomiseen suljettuun ympäristöön (Vaatimus 7). Liitteessä 3 on kuvattuna millainen arkkitehtuurinen ratkaisu versioon 0.1 toteutettiin käynnistysrutiinin ja koulutussilmukan osalta.

Ensimmäisen version koulutussilmukka (Liite 3 – Kuva 2) luotiin pääosin olemassa olevien esimerkkien avulla, siten että silmukkaan toteutettiin vain hyvin yksinkertainen mallin koulutus, evaluointi ja tallennus (Kuva 23). Silmukkaan toteutettiin vain yksi parametrien optimointitapa

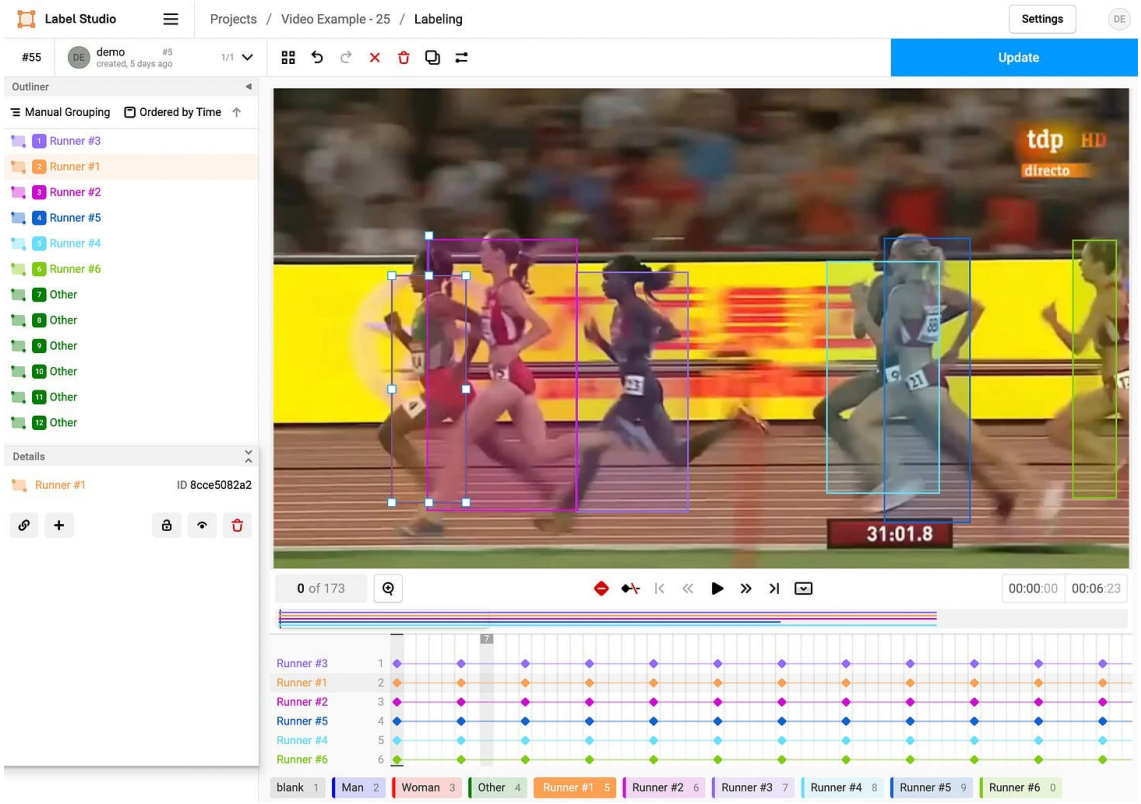
(Stochastic gradient descent - SGD), joka on käytännössä yleisin ja monikäyttöisin optimointitapa kohteentunnistuksessa. Silmukassa ei toteutettu mitään muuta lokitietojen keräystapaa, kuin tulostettavien rivien tallentaminen txt-tiedostoon.

```
151         for epoch in range(1, args.epochs+1):
152
153             print("\n---- Training Model ----")
154
155             model.train()  # Set model to training mode
156
157             for batch_i, (_, imgs, targets) in enumerate(tqdm.tqdm(dataloader, desc=f"Training Epoch {epoch}")):
158                 batches_done = len(dataloader) * epoch + batch_i
159
160                 imgs = imgs.to(device, non_blocking=True)
161                 targets = targets.to(device)
162
163                 outputs = model(imgs)
164
165                 loss, loss_components = compute_loss(outputs, targets, model)
166
167                 loss.backward()
168
169                 ###############
170                 # Run optimizer
171                 ###############
172
173                 if batches_done % model.hyperparams['subdivisions'] == 0:
174                     # Adapt learning rate
175                     # Get learning rate defined in cfg
176                     lr = model.hyperparams['learning_rate']
177                     if batches_done < model.hyperparams['burn_in']:
178                         # Burn in
179                         lr *= (batches_done / model.hyperparams['burn_in'])
180                     else:
181                         # Set and parse the learning rate to the steps defined in the cfg
182                         for threshold, value in model.hyperparams['lr_steps']:
183                             if batches_done > threshold:
184                                 lr *= value
185                     # Log the learning rate
186                     logger.scalar_summary("train/learning_rate", lr, batches_done)
187                     # Set learning rate
188                     for g in optimizer.param_groups:
189                         g['lr'] = lr
190
191                     # Run optimizer
192                     optimizer.step()
193                     # Reset gradients
194                     optimizer.zero_grad()
```

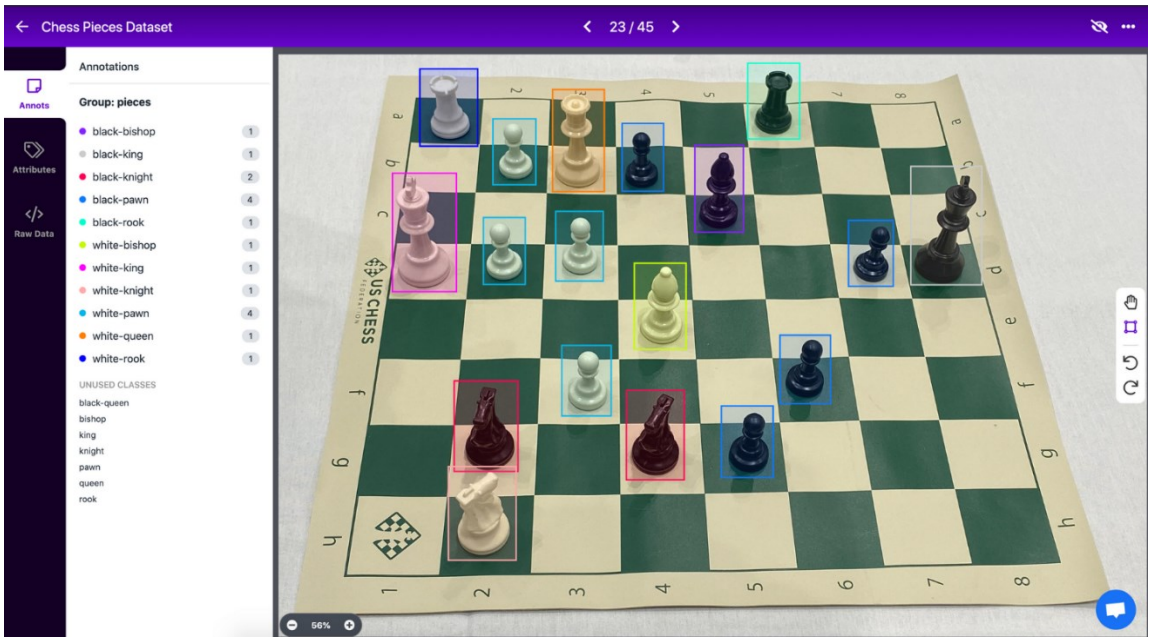
Kuva 23 – Version 0.1 koulutussilmukka

Koulutusmateriaalin nimikointiin valikoitui työkaluksi LabelStudio (Kuva 24), koska sillä oli testatuista työkaluista selkein dokumentaatio, avoimen lähdekoodin tuote ja käytettävissä täysin suljetussa ympäristössä (Vaatimus 7). Tuotteella oli myös suhteellisen hyvät jatkokehitys mahdollisuudet semi/automaattisen nimikoinnin kannalta. Koulutusmateriaalin nimikointiin, manipulointiin ja

analysointiin aputyökaluksi valikoitui Roboflow:n kuvien nimikointityökalu (Kuva 25), joka on käytettävissä ilmaiseksi web-selaimen kautta.



Kuva 24 –LabelStudio:n käyttöliittymä (Labelstudio 2022)



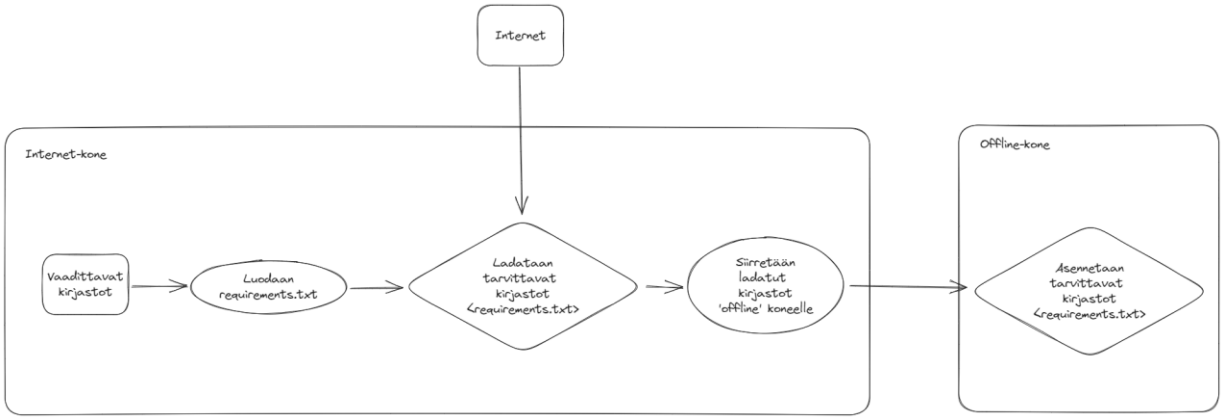
Kuva 25 – Kuva Roboflow:n käyttöliittymä (Roboflow 2024)

Koulutusmateriaalia tuotettiin ensimmäiseen koulutuskierrokseen noin 800 kuvaa, joiden määrää kasvatettiin Roboflown manipulointityökalujen avulla kaksinkertaiseksi käyttämällä kuvien muuntamista mustavalkoiseksi (Kuva 26), lisäämällä kohinaa (Kuva 27) ja sumentamalla kuvaa.

Kuva 26 – Koulutusmateriaalin kuva, joka on muutettu mustavalkoiseksi

Kuva 27 – Koulutusmateriaalin kuva, johon lisätty rakeisuutta.

Tarvittavien kirjastojen lataaminen ja vieminen suljettuun ympäristöön (Vaatimus 7) osoittautui kohtuullisen suoraviivaiseksi operaatioksi, jossa vaadittava tiedosto tarvittavista kirjastoista luodaan joko käsin, käyttämällä Pythonin omaa kirjastojen hallintaskriptiä (pip) tai luomalla kyseinen tiedosto ohjelmistoympäristön (esimerkiksi PyCharm) kautta. Kirjastotiedoston luomisen jälkeen ladataan tarvittavat kirjastot koneella, jolla on pääsy internettiin. Lataamisen jälkeen ladatut tiedostot pakataan esim. *.zip tai *.tar.gz formaattiin ja siirretään suljettuun ympäristöön (Kuva 28). Tämän jälkeen siirretty tiedosto puretaan ja kirjastot asennetaan suljettuun ympäristöön hyväksikäyttämällä Pythonin pip -kirjastojen hallintatyökalua.



Kuva 28 – Python kirjastojen lataamisen prosessi

Version 0.2 tavoitteiksi muodostuivat koulutussilmukan parantaminen, koulutuksen monitoroinnin parantaminen ja koulutusmateriaalin kasvattaminen.

Koulutussilmukan parantaminen keskittyi tässä versiossa lähinnä siihen, että miten koulutuksen aikana muodostuvia malleja voitaisiin vertailla keskenään. Versiossa 0.1 mallien välinen vertailu tehtiin vain vertailemalla tarkkuuden keskiarvoja (mAP – Mean Average Precision) toisiinsa. Vaikkakin tämä menetelmä onkin hyvin toimiva ei se ole koska mallin hyvyyteen vaikuttavat muutkin arvot, kuten Recall, Precision ja F1. Ratkaisuna tähän ongelmaan päätin soveltaa Ultralytics:n ratkaisussa käytettyä menetelmää, jossa mallin hyvyys lasketaan arvojen (Precision, Recall, mAP@0.5, mAP@0.5:0.95) painotettuna keskiarvona.

Sovelletussa versiossa mallille lasketaan koulutuksen aikana kaksi hyvyyden arvoa Training Fitness ja Model Fitness, siten että Training Fitness lasketaan mallin koulutuksen aikana saatavien häviöiden (IOU Loss, Class Loss, Object Loss, Loss) painotettuna keskiarvona (Kuva 29) ja Model Fitness lasketaan mallin evaluoinnin tuloksena saatavien arvojen (Precision, Recall, mAP@0.5, F1 ja AP Class) painotettuna keskiarvona (Vaatimus 1 ja 2)(Kuva 30).

```
239    def training_fitness(x, w):
240        # Model training fitness as a weighted combination of training metrics
241        #w = [0.25, 0.25, 0.25, 0.25]  # weights for [IOU, Class, Object, Loss]
242        return (x[:, :4] * w).sum(1)
```

Kuva 29 – Training Fitness laskenta

```
def fitness(x, w):
    # Model fitness as a weighted combination of metrics
    #w = [0.0, 0.05, 0.85, 0.1, 0.0]  # weights for [P, R, mAP@0.5, f1, ap class]
    return (x[:, :5] * w).sum(1)
```

Kuva 30 – Model Fitness laskenta

Näiden saatujen tulosten perusteella koulutussilmukkaan määriteltiin ehdot, siten että mikäli Training Fitness arvo on koulutuksen aikana tallennettua aikaisempaa arvoa parempi, aloitetaan mallin evaluointi automaattisesti. Tämän lisäksi mallin evaluointi suoritetaan aina silloin kun käyttäjä on määritellyt ’Evaluate interval’ -parametrissa. Mallin hyvyyden laskennan lisäksi versioon 0.2 luotiin kyky seurata mallin koulutusta (Liite 4), joko Tensorboard:n ja/tai ClearML-sovellusten avulla, sekä luotiin lokitietojen tallentaminen *.csv tiedostoihin paremman luettavuuden ja analysoinnin vuoksi (Vaatimus 7).

Tensorboard ja ClearML valikoituivat käyttöön siksi että Tensorboard on suhteellisen yksinkertainen ja helppokäyttöinen ohjelma, jolla voidaan seurata monia mallin koulutuksen parametrejä ja se on ajettavissa paikallisesti mallin koulutusta tehtävällä koneella. Kun taas ClearML valikoitui käytettäväksi siksi, että siinä oli kohtuullisen hyvä ja selkeä suljetun ympäristön asennuksen tuki, sekä sisäänrakennettu tuki PyTorch-kehyksen parametrien talteen ottamiseen ja raportointiin palvelimelle (Vaatimus 7). Koulutusmateriaalin osalta materiaalin määrää nostettiin 2000 kuvaan, jonka kokonaismäärää jälleen kasvatettiin kaksinkertaiseksi Roboflown manipulointityökalujen avulla.

Version 0.3 keskeisimpinä kehityskohteina aluksi olivat manipulointityökalut, kuten kuvien kierto, muuntaminen mustavalkoiseksi, kyllästäminen, valotus ja sumentaminen. Näiden lisäksi kehityskohteina olivat koulutussilmukan parannukset automaattisen koulutussetin koon laskeminen, paremman koulutuksen lämmittelyn (warmup loop) luominen, paremman vikojen raportoinnin ja lokitietojen luominen, sekä koulutuksen parametrien kirjoittaminen kuvatiedostoon. Näiden lisäksi luotiin tarvittavia aputyökaluja, kuten koulutussetin analysaattori (dataset_analyzer), koulutussetin jakaja (splitter). Liitteessä 5 kuvataan millainen arkkitehtuurinen ratkaisu versioon 0.3 toteutettiin käynnistysrutiinin ja koulutussilmukan osalta, sekä mitä osia oleellisimmat muutokset koskivat.

Koulutusmateriaalin määrää nostettiin hieman yli 8000 kuvaan, jonka määrää jälleen kaksinkertaistettiin Roboflown manipulointityökalujen avulla. Tällä kertaa kuvien mustavalkoisuutta, rakeisuutta tai sumeutta ei muunnettu, vaan kuville tehtiin leikkaus ja yhdistelyä (mosaic ja cutmix), jotta mallin tarkkuutta saataisiin parannettua pienten kohteiden osalta (Vaatimus 1 ja 2).

Mallin koulutuksen aikana kävi selväksi, että lisääntyvän koulutusmateriaalin takia vapaasti käytettävissä olevat koulutusresurssit (esimerkiksi Google Colab) eivät enää käytännössä riitä. Teknisesti resurssit ovat yhtä toimivia kuin ennenkin, mutta grafiikkakiihdyttimen muistin vähäisyyden takia (esimerkiksi Google Colab:ssa käytettävissä NVIDIA T4 – 16GB / FP32 - 8.1 TFLOPS) mallin kouluttaminen alkoi viemään huomattavasti aikaa. Lisäksi mallien huomattiin tekevän ylioppimista, jota jouduttiin korjaamaan erinäisillä toimenpiteillä.

Edellä mainituista syistä ensimmäisten koulutuskierrosten jälkeen jouduttiin tekemään seuraavia jatkokehityksiä versioon 0.3:

1. lisätään optimointitapojen (optimizers) määrää
2. luodaan mahdollisuus käyttää erilaisia oppimiskertoimen päivittäjiä (learn rate schedulers)
3. yksinkertaistetaan käytettävää mallia huomattavasti, ottamalla käyttöön yolo-arkkitehtuurissa määritelty ’tiny’-neuroverkko, jonka pohjalle oma neuroverkko jatkokoulutetaan.
4. optimoidaan koulutusmoduulin koodia tarvittavilta osin.

Lisäksi kartoitettiin sopivia pilvessä olevia palveluita, jotka tarjoavat grafiikkakiihdyttimillä varustettuja koneita tekoälyn koulutusta varten. Perusajatuksena oli löytää suhteellisen halpa palveluntarjoaja, joka kuitenkin tarjoaa mahdollisuuden valita konesalin sijainnin, sekä tarjoaa suojattuja virtuaalikoneita (secure cloud tai vastaava) käyttöön. Palveluntarjoajia on luonnollisesti olemassa paljon, kuten Google, Amazon, Microsoft suurimmista mainitakseni. Mutta myös paljon pienempiä toimijoita, joiden palvelut ovat edelle kerrottuja suuria toimijoita halvempia, kuten Akamai, Runpod, Linode, Vast.ai, Jarvislabs, LeaderGPU, Xesktop, ArkaneCloud, Runpod. Näistä käytettäväksi valikoitui Runpod, lähinnä siitä syystä, että sillä oli tarjolla palvelimia Euroopassa, dokumentaatio oli kohtuullisen hyvällä tasolla.

Käytännössä tämä pilvipalvelun käyttäminen ei sinänsä ole työn kannalta tarvittava askel, lähinnä grafiikkakiihdyttimen kapasiteetin lisäys oli haettu ominaisuus, mutta mallin kouluttaminen pilviympäristössä oli tällä kertaa vain käytännöllisempää. Koska tällöin pystyin paremmin valvomaan ja ohjaamaan mallin kouluttamista myös kotoa, iltaisin ja viikonloppuisin. Pilvipalvelussa käytetyt menetelmät ovat aivan hyvin toistettavissa tai tuotavissa suljettuun ympäristöön, joten sinällään tästä näkökulmasta ei ole väliä kummassa ympäristössä toimittiin.

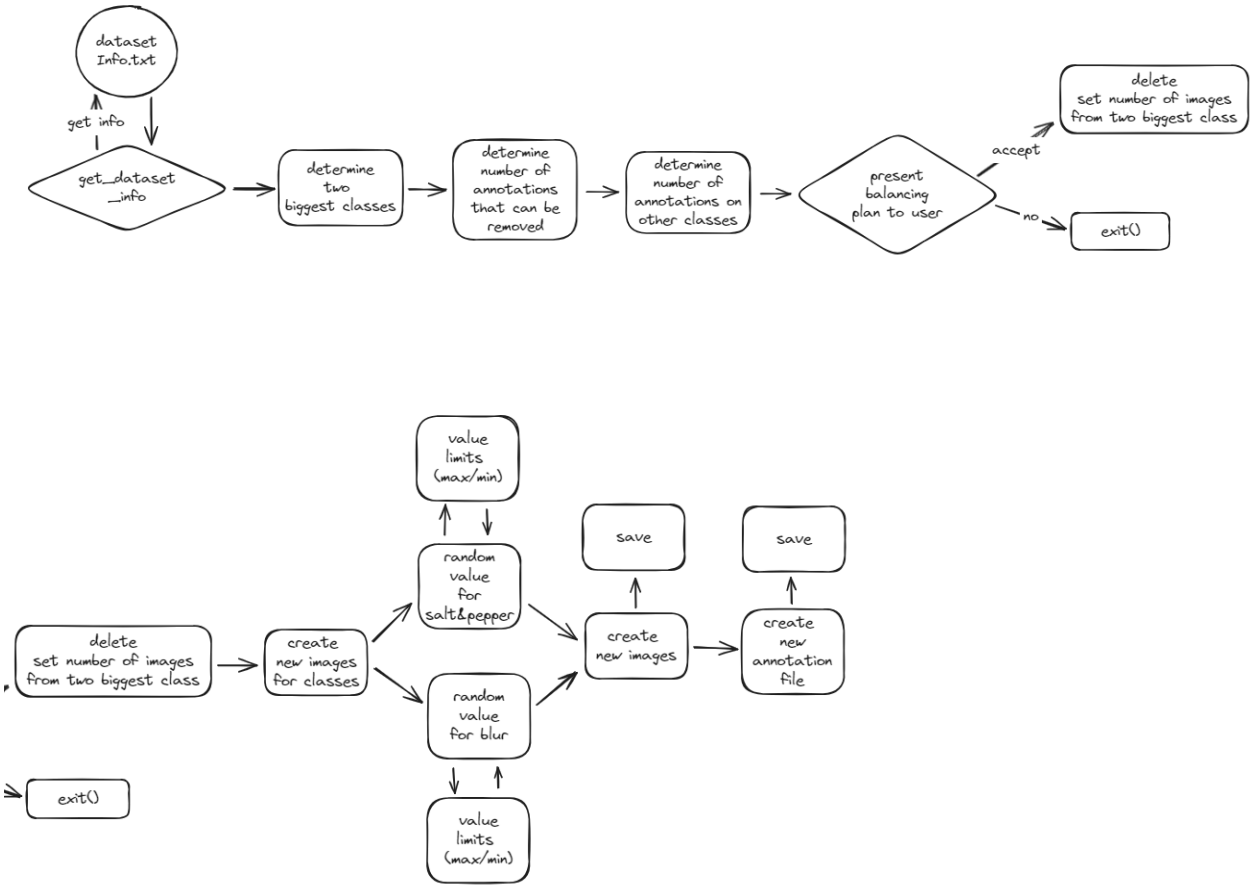
Koulutuksen aikana saatiin edellä mainittujen muutosten avulla tuotettua malleja, joiden tavoitearvot alkoivat lähestyä määrittelyssä asetettuja tavoitteita. Malleissa kuitenkin havaittiin olevan kohtuullisen suuri tunnistuksen epätasapaino eri luokkien välillä. Vaikka mallin kokonaistarkkuus oli usein luokkaa >0.6 mAP, saattoi yksittäisen luokan tarkkuus olla huonoimmillaan vain hieman yli 0.3 mAP (Vaatimus 1). Tätä tunnistuksen epätasapainoa pyrittiin ensin tunnistamaan ja analysoimaan koulutusmateriaalista *datasetAnalyzer.py* skriptin avulla (Kuva 31), joka lukee koulutusmateriaalista kaikki nimikoinnit ja tulostaa tämän jälkeen jokaisen luokan nimikointien määrän.

```python
import os

'''
This code analyzes a dataset by counting the number of annotations for each class.
It prompts the user to provide the path to the dataset folder, reads the classes from a "classes.txt" file,
and initializes a list to store the number of annotations for each class.
It then reads the label files from a "labels" directory and updates the annotation count for each class.
Finally, it writes the results to a "datasetInfo.txt" file, displaying the number of annotations for each
class and the total number of annotations in the dataset.
'''

# Prompt user for dataset folder path
path = input("Give dataset folder path:")

# Define paths for classes file and labels directory
classes_file = path + "/classes.txt"
labels_dir = path + "/labels"

# Initialize lists for classes and annotations
classes = []
annotations = []

# Read classes file and store classes in a list
f = open(classes_file, "r")
for line in f:
    line_to_list = line.replace( _old: "\n",  _new: "")
    classes.append(line_to_list)
    annotations.append(0)
f.close()

# Read label files and update annotation count for each class
for filename in os.listdir(labels_dir):
    f = os.path.join(labels_dir, filename)
    if os.path.isfile(f) and filename.endswith(".txt"):
        file = open(f, "r")
        for line in file:
            annotation_number = line.split(" ")[0]
            annotations[int(annotation_number)] += 1
        file.close()

# Write results to datasetInfo.txt file
f = open(path + "/datasetInfo.txt", "w")
print("Number of annotations in dataset by class:\n")

for i in range(len(classes)):
    print(classes[i] + "-" + str(annotations[i]))
    f.write(classes[i] + "-" + str(annotations[i]) + "\n")
f.close()

# Print total number of annotations and file path of datasetInfo.txt
print(f"\nTotal number of annotations: {sum(annotations)}")
print(f'Results written into {path + "/datasetInfo.txt"}\n')
```

Kuva 31 – datasetAnalyzer.py:n koodi

Kun koulutusmateriaalin epätasapaino oli saatu selvitettyä datasetAnalyzer skriptillä, kokeiltiin tasapainottaa koulutusmateriaalia luomalla *datasetBalancer.py* skripti (Kuva 32), joka käyttää hyväkseen datasetAnalyzer:n luomaa kuvausta koulutusmateriaalin epätasapainosta. DatasetBalancer

hakee tuotetusta kuvauksesta kaksi suurinta nimikointiluokkaa ja laskee tämän jälkeen koulutusmateriaalista kaikki koulutuskuvat, joissa on vain yksi jompaankumpaan suurimpaan luokkaan kuuluva kohde. Tämän jälkeen datasetAnalyzer tuottaa kuvauksen siitä, kuinka monta kuvaa kahdesta suurimmasta nimikointiluokasta voidaan poistaa ja kuinka paljon kuvia pienempiin luokkiin tulisi lisätä. Mikäli käyttäjä hyväksyy datasetAnalyzer:n ehdotuksen ohjelma poistaa koulutusmateriaalista laskemansa määrän kahden suurimman luokan kuvista ja lisää pienempiin luokkiin uusia kuvia luomalla olemassa olevista kuvista datamanipulaation avulla uusia kuvia. Uusien kuvien luonti tapahtuu käytännössä lisäämällä olemassa oleviin kuviin kohinaa ja muuttamalla kuvan tarkkuutta satunnaisotannan avulla.



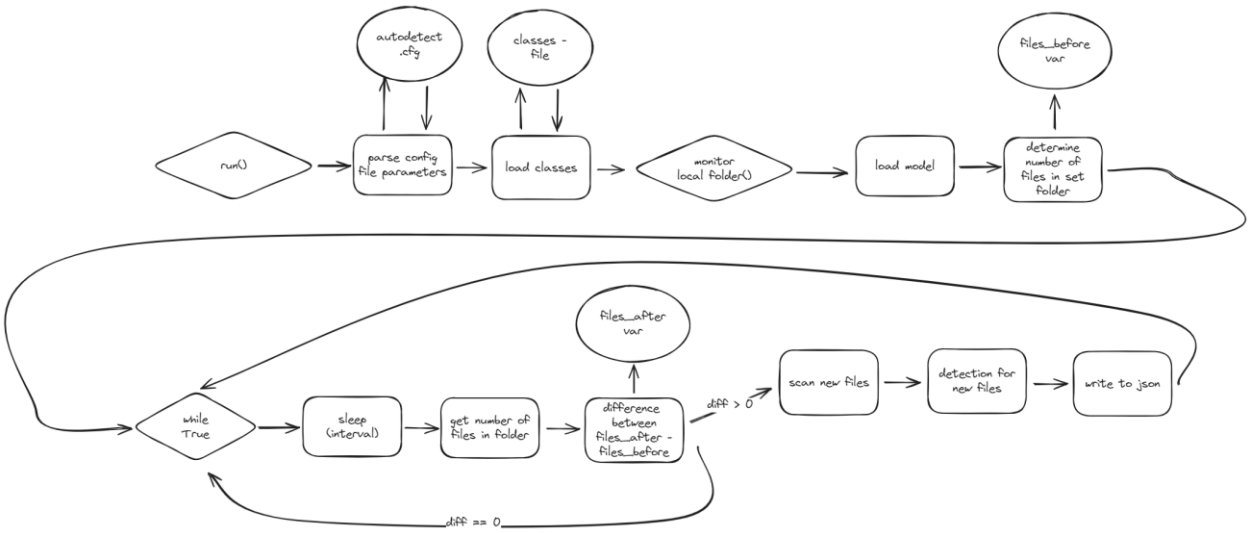
Kuva 32 –datasetBalancer.py:n toiminta prosessi

Tämän prosessin avulla koulutusmateriaalia saadaan tasapainotettua, mutta kääntöpuolena on se, että mikäli kahta suurinta nimikointiluokkaa ei saada tarpeeksi pienennettyä, kasvaa koulutusmateriaalin kokonaiskoko suhteellisen suureksi. Tässäkin tapauksessa kävi juuri niin että reilusta 8000 kuvan alkuperäismäärä kasvoi tämän tasapainotusoperaation jälkeen reiluun 61 000 kuvaan. Tästä johtuen mallin kouluttaminen vei jo kohtuullisen paljon aikaa ja vaati suhteellisen paljon resursseja.

Tällä koulutusmateriaalilla saatiin kuitenkin tuotettua jo malli, joka oli huomattavasti paremmin tasapainossa edellisiin verrattuna ja lähestulkoon täytti asiakkaan vaatimukset.

Sopivan mallin luomisen jälkeen sovimme asiakkaan kanssa kohdeympäristön implementaation luonteesta, jolloin sen osalta työ myös pääsi alkamaan. Samalla tuli esiin toive siitä, että olisi hienoa, jos malli pystyisi tunnistamaan kohteita myös videolta. Vaikka tätä ei alkuperäisessä suunnitelmassa ollut luvattu, niin sekin luvattiin toteuttaa lähinnä siksi että sillä on suuri merkitys asiakkaalle, sekä siksi että toteutuksen osalta ei suuria haasteita ollut nähtävissä enää tässä vaiheessa.

Kohdeympäristön implementaatioksi sovittiin hyvin yksinkertainen ratkaisu, jossa skripti lukee kohdehakemistoa määräajoin ja mikäli se havaitsee hakemistossa uusia kuvia, suoritetaan kuvilla tunnistus, tämän jälkeen tulokset kirjoitetaan json-tiedostoon, josta tiedot voidaan ottaa jatkokäyttöön (Kuva 33).



Kuva 33 –autoDetect.py:n toimintaprosessi

Kohteiden tunnistamiseen videolta luotiin vain hyvin yksinkertainen prototyyppinen skripti, jossa skriptille syötetään tunnistusmallin tiedot, luokat ja video, jolta tunnistus tulisi tehdä. Tämän jälkeen skripti esittää videolla tunnistettavat kohteet.

Koska koulutusmateriaalin tasapainotus kasvattamalla koulutusmateriaalin määrää ei aina ole paras tapa parantaa eri luokkien välistä epätasapainoa, varsinkin kun luokkia on kohtuullisen monta. Tämä johtuu yksinkertaisesti siitä, että koulutusmateriaalin koko alkaa kasvamaan kohtuuttoman suureksi, joka johtaa siihen, että koulutukseen tarvittavia grafiikkakiihdyttimiä tarvitaan koko ajan enemmän (joko useita pienempiä yksiköitä tai yksi suuren muistin omaava yksikkö) tai vaihtoehtoisesti kouluttamisen kesto kasvaa huomattavasti.

Tästä syystä oli miestäni tarpeellista etsiä myös muita tapoja parantaa mallien välistä tarkkuuden tasapainoa, kuin vain edellä mainitulla tavalla. Tähän oli nähtävissä kaksi suoraviivaista ratkaisua:

1. Parannetaan datasetBalancer:n laskentaa siten että skripti ottaa laskennassa huomioon myös sellaiset koulutusmateriaalit, jossa on useita eri luokkia samassa kuvassa ja laskee näiden perusteella vähennettävien ja lisättävien määrän.
2. Implementoidaan sellainen materiaalin lataaja (dataloader) koulutussilmukkaan, joka pystyy ottamaan huomioon eri luokkien väliset epätasapainot ja syöttää koulutukseen koulutuskuvia tasapainotetusti.

Näistä kahdesta vaihtoehdosta käytännöllisemmältä vaikutti vaihtoehto kaksi, koska erilaiset materiaalin lataajat kuuluvat PyTorch:n kirjastoihin, joten niiden implementointi, monipuolisuus ja jatkokäytön mahdollisuudet olivat huomattavasti vaihtoehtoa yksi paremmat.

Näiden ominaisuuksien tehokas käyttö kuitenkin vaatii myös muutoksia muihin koodin osiin, lähinnä parempien manipulointityökalujen muodossa, sekä analysointityökalujen parantamista. Näistä syitä versioon 0.4 päätettiin tehdä vielä seuraavat muutokset ja päivitykset:

1. Implementoidaan seuraavat koulutusmateriaalin lataajat: SequentialSampler, RandomSampler, SubsetRandomSampler, WeightedRandomSampler, BatchSampler
2. Luodaan menetelmä koulutusmateriaalin epätasapainon päättelyyn
3. Implementoidaan parannetut kuvamanipulaation välineet
4. Parannetaan optimoijien ja ajastimien parametrien käyttöä
5. Lisätään grafiikkakiihdyttimen muistin valvonta
6. Implementoidaan sekaannusmatriisi analysoinnin tueksi
7. Parannetaan koulutussilmukan tehokkuutta koodia optimoimalla
8. Luodaan yksinkertaiset testausmenetelmät PyTest:n avulla

Materiaalin lataajien implementoinnissa käytettiin apuna PyTorch:n dokumentaatiota sekä vapaasti käytettävää YOLOv4 koodia. Sinällään itse implementaatio on melko suoraviivainen prosessi, kuitenkin suurempaa mietintää aiheutti tapa, miten koulutusmateriaalin epätasapaino pitäisi laskea. Laskennassa päädyttiin hyvin yksinkertaiseen menetelmään, jossa jokaisen luokan kuvien määrä ilmoitetaan suhteutettuna koko koulutusmassan kuvien määrään. Lisäksi uusien kuvamanipulaation työkalujen implementoinnin takia suurin osa vanhasta koulutusmateriaalin latausskriptistä jouduttiin kirjoittamaan uudestaan, jotta lataaja pystyisi tukemaan uusia kuvan manipuloinnin työkaluja, kuten HSV (Hue, Saturation, Value), Mosaic, Letterbox, Random perspective, Cutout.

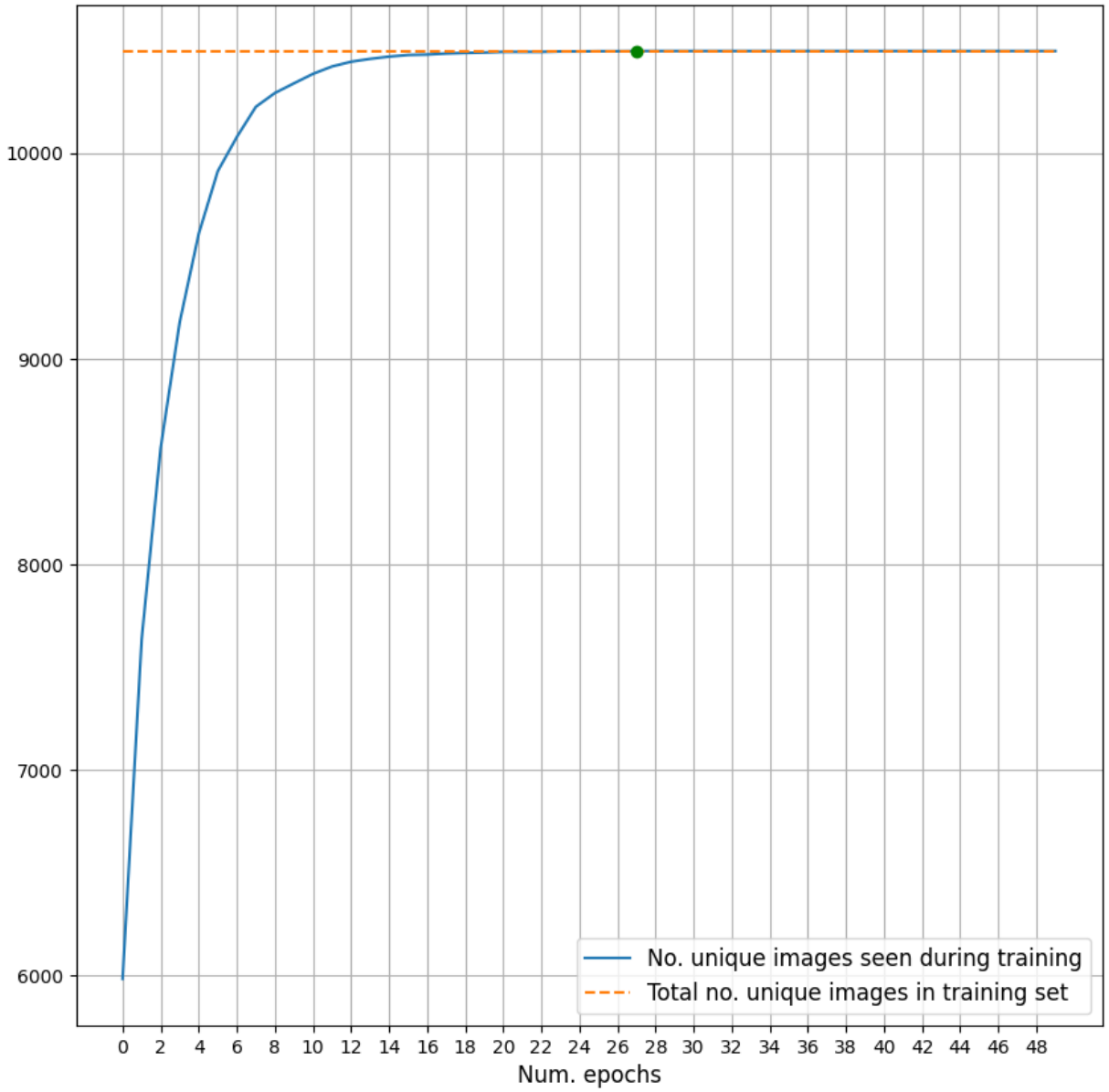
Kun edellä mainituista koulutusmateriaalin lataajista käytetään erityisesti WeightedRandomSampler:ia päästään erityisen mielenkiintoisen asian ääreen liittyen koulutusmateriaalin käyttöön mallin kouluttamisessa. Perinteinen ajatusmallihan on se, että mallia koulutettaessa koulutusmateriaalin

kuvia syötetään mallille siten että koko koulutusmateriaalin massa käydään yhden koulutuskierroksen (epoch) aikana läpi. Kuitenkin käytettäessä WeightedRandomSampler:ia koulutusmateriaalia syötetään mallille, siten että vähemmän kuvia sisältäviä kuvia syötetään mallille suhteessa enemmän kuin sellaisia luokkia, joiden kuvia on paljon. Tämä johtaa käytännössä tilanteeseen, jossa yhden koulutuskierroksen jälkeen malli on saattanut nähdä yhden luokan kuvia 10 ja ei ollenkaan toisen luokan kuvia, jolloin perinteinen ajatusmalli koulutusmateriaalin läpikäynnistä ei enää toimikkaan. Tästä syystä oli tarpeen saada ymmärrys siitä, kuinka monta koulutuskierrosta tarvittaisiin, jotta todennäköisyyksien mukaan voitaisiin olettaa, että koulutettava malli on nähnyt kaikki koulutusmateriaalin kuvat. Tätä kysymystä pyrittiin ratkaisemaan kahdella eri tapaa:

1. Käyttämällä yksinkertaista skriptiä, jolla simuloidaan koulutusmateriaalin lataamista ja käyttämistä mallin koulutukseen ja laskemalla, kuinka monta koulutuskierrosta kestäisi, jotta kaikki koulutusmateriaalin kuvat ovat käyty läpi (Kuva 34). Tuloksena oli, että arviolta noin joka 23 - 27 koulutuskierros kaikki koulutusmateriaalin kuvat ovat käyty läpi (Kuva 35).

```python
num_epochs = 50

unique_idxs_seen = set()
cumulative_unique_images_per_epoch = {}

for i in range(num_epochs):
    class_0_batch_counts, class_1_batch_counts,class_2_batch_counts,class_3_batch_counts,
        dl, with_outputs=False
    )
    unique_idxs_seen.update(idxs_seen)
    cumulative_unique_images_per_epoch[i] = len(unique_idxs_seen)

#print(len(unique_idxs_seen))
#print(cumulative_unique_images_per_epoch)
target_epoch = list(cumulative_unique_images_per_epoch.values()).index(len(ds))

fig, ax = plt.subplots(1, figsize=(10, 10))

ax.plot(
    cumulative_unique_images_per_epoch.keys(),
    cumulative_unique_images_per_epoch.values(),
    label="No. unique images seen during training",
)
ax.plot(
    cumulative_unique_images_per_epoch.keys(),
    [len(ds)] * len(cumulative_unique_images_per_epoch),
    label="Total no. unique images in training set",
    linestyle="--",
)

ax.plot(target_epoch, len(ds), "go")
ax.legend(fontsize=12)
ax.set_xlabel("Num. epochs", fontsize=12)
ax.grid()

ax.set_xticks(
    np.arange(50, step=2),
)
fig.show()
```

Kuva 34 – Esimerkki skriptistä, jonka avulla laskettiin, montako koulutuskierrosta tarvitaan, jotta koulutettava neuroverkko on nähnyt koko koulutusmateriaalin.

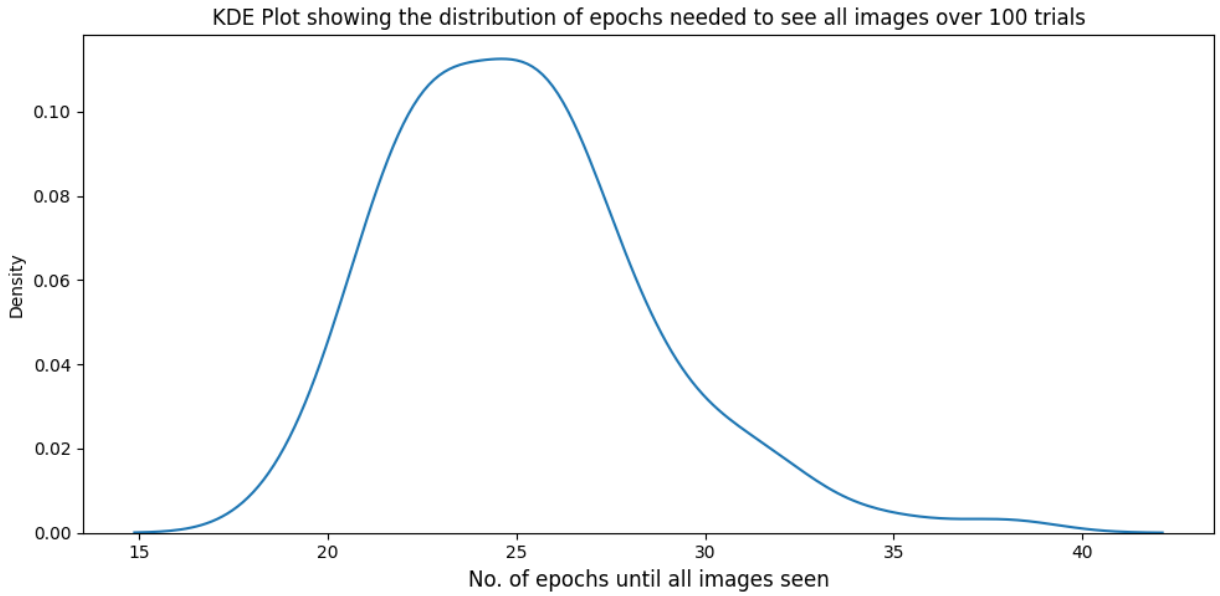


Kuva 35 – Esimerkki edellisen kuvan skriptin antamasta arviosta monenko koulutuskierroksen jälkeen koulutettava neuroverkko on nähnyt koko koulutusmateriaalin.

2. Varmentamalla edellisessä kohdassa saatu vastaus käyttämällä Monte Carlo -simulaatiota 50 koulutuskierroksen ajolla 100 yritystä (Kuva 36). Tämän tuloksena saatiin samansuuntaisia tuloksia kuin edellisessäkin kohdassa, eli noin 23-25 koulutuskierroksen kohdalla neuroverkko olisi nähnyt kaikki koulutusmateriaalin kuvat (Kuva 37).

```
import seaborn as sns
num_trials = 100
num_epochs_needed = []
num_epochs = 50

for t in range(num_trials):
    unique_idxs_seen = set()
    for i in range(num_epochs):
        class_0_batch_counts, class_1_batch_counts,class_2_batch_counts,class_3_batch_counts,class_4_batch_counts,
        unique_idxs_seen.update(idxs_seen)
        if len(unique_idxs_seen) == len(ds):
            num_epochs_needed.append(i)
            break

fig, ax = plt.subplots(figsize=(10, 5))
sns.kdeplot(num_epochs_needed)
ax.set_xlabel('No. of epochs until all images seen', fontsize=12)
ax.set_title('KDE Plot showing the distribution of epochs needed to see all images over 100 trials')
fig.tight_layout()
fig.show()
```

Kuva 36 – Esimerkki Monte Carlo -simulaatiosta, jolla pyrittiin löytämään arvio monenko koulutuskierroksen jälkeen, koulutettava malli on nähnyt koko koulutusmateriaalin.



Kuva 37 – Esimerkki Monte Carlo -simulaation vastauksesta, jossa esitetään monenko koulutuskierroksen jälkeen, koulutettava neuroverkko on nähnyt koko koulutusmateriaalin.

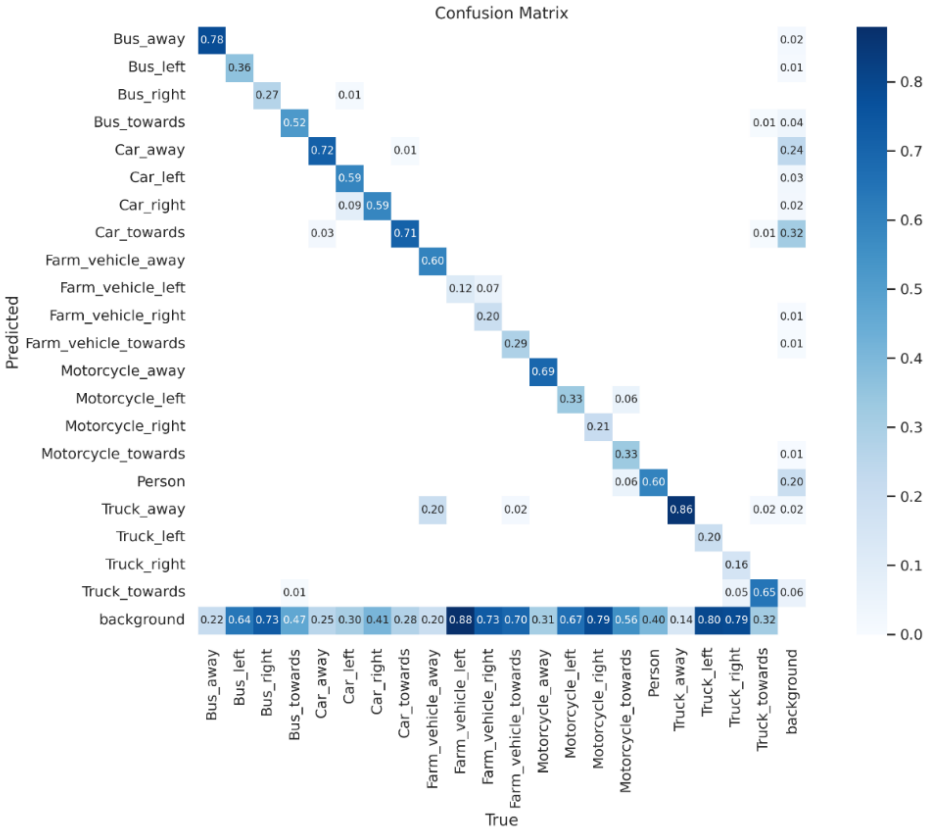
Näiden testien jälkeen oli helppoa tehdä luotettava arvio siitä, montako koulutuskierrosta tarvittiin, jotta koulutettava neuroverkko olisi nähnyt kaikki koulutusmateriaalin kuvat ainakin kerran.

Optimoijien ja ajastimien parametrien osalta päädyttiin vain yhtenäistämään asioita, siten että koodi hakee tässä versiossa koulutuksessa käytettävät hyperparametrit omasta konfiguraatiotiedostostaan mallin tiedoston sijaan. Eli aikaisemmin mallin kaikki parametrit ovat olleet samassa tiedostossa, ja nyt mallin neuroverkon rakenne ja kouluttamisen parametrit ovat omina tiedostoinaan.

Tämä muutos tehtiin lähinnä siksi, että jatkossa on helpompi hallita ja muuttaa koulutuksen parametrejä ja mallin arkkitehtuuria, koska tiedostot ovat nyt eriytetty. Lisäksi lisääntynyt koulutusparametrien määrä (versiossa 0.4 – 34 kpl) on nyt helpompi käsitellä ja päivittää jatkossa, mikäli parametrejä on tarve lisätä tai muuttaa.

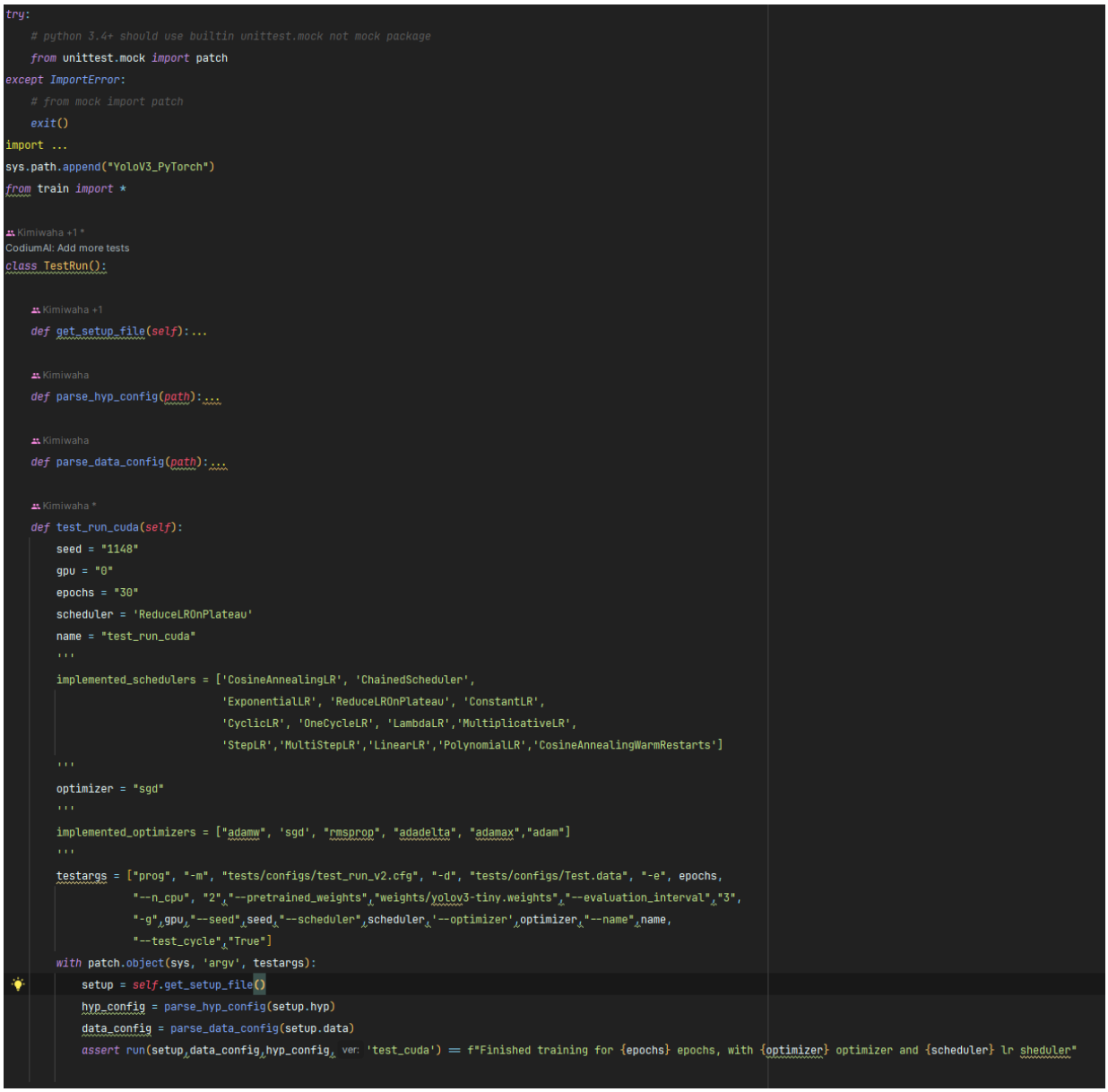
Grafiikkakiihdyttimen muistin valvonta ei sinällään ollut mikään valtavan suuri työ ja sen merkitys on lähinnä kosmeettinen siinä mielessä, että tämän tarkoitus on lähinnä auttaa vikojen selvittämisessä, varsinkin niissä tilanteissa, joissa mallin gradientit ’räjähtävät’ tai koulutusmateriaalin lataus epäonnistuu muistiylivuotojen takia.

Sekaannusmatriisi implementoitiin lähinnä siitä syystä, että mielestäni oli hyvä saada jokin mittari sille, miten hyvin malli pystyi tunnistamaan tietyn luokan mukaisia kohteita. Sekaannusmatriisin (Kuva 38) implementointi oli hieman monimutkainen prosessi, jossa jouduttiin jälleen tukeutumaan olemassa oleviin implementointeihin ja rakentamaan näiden pohjalta oma ratkaisu. Tässä toteutuksessa käytettiin hyväksi matplotlib -kirjaston lisäkirjastoa seaborn, jonka avulla matriisi piirretään. Suurimmaksi käytännön ongelmaksi muodostui matplotlib-kirjaston bugi, jossa kirjastoon sisäinen muistinsiivous ei toimi tai ehdi siivoamaan vanhoja kuvia pois muistista, jolloin tapahtuu muistivuotoja ja lopulta koneen välimuisti loppuu ja koodin suorittaminen keskeytyy. Tämän ongelman havaitsemiseen ja korjaamiseen meni yllättävän paljon aikaa ja lopulta toimivan ratkaisun löytäminen vaati kohtuullisen paljon raakaa kokeilua ja testausta. Lopputuloksena on kuitenkin toimiva ja tehokas laskenta sekaannusmatriisille.



Kuva 38 – Esimerkki sekaannusmatriirista

Koulutussilmukan tehokkuutta parannettiin lähinnä koodia profiloimalla ja yksinkertaistamalla, sekä poistamalla turhia muuttujia ja kirjastoja. Koodin monimutkaistuessa ja sekaannusmatriisin testauksen vaikeuksien takia tuli tarpeeseen luoda yksinkertaiset testaustyökalut PyTest:n avulla. Testauksessa lähinnä on mahdollista testata koko koulutussilmukan toiminta, siten että testauksessa määritellään käytettävä testausmateriaali, optimoija, ajastin ja koulutuskierrosten määrä (Kuva 39). Tämän jälkeen testausskripti käynnistää kouluttimen ja ilmoittaa mikäli koulutin, pystyy suorittamaan kaikki käsketyt koulutuskierrokset. Liitteessä 6 on esitelty käynnistysprosessin ja koulutussilmukan oleellisimmat arkkitehtuuriset muutokset versiossa 0.4.



```
try:
    # python 3.4+ should use builtin unittest.mock not mock package
    from unittest.mock import patch
except ImportError:
    # from mock import patch
    exit()
import ...
sys.path.append("YoloV3_PyTorch")
from train import *

Kimiwaha +1 *
CodiumAI: Add more tests
class TestRun():

    Kimiwaha +1
    def get_setup_file(self):...

    Kimiwaha
    def parse_hyp_config(path):...

    Kimiwaha
    def parse_data_config(path):...

    Kimiwaha *
    def test_run_cuda(self):
        seed = "1148"
        gpu = "0"
        epochs = "30"
        scheduler = 'ReduceLROnPlateau'
        name = "test_run_cuda"
        '''
        implemented_schedulers = ['CosineAnnealingLR', 'ChainedScheduler',
                                  'ExponentialLR', 'ReduceLROnPlateau', 'ConstantLR',
                                  'CyclicLR', 'OneCycleLR', 'LambdaLR','MultiplicativeLR',
                                  'StepLR','MultiStepLR','LinearLR','PolynomialLR','CosineAnnealingWarmRestarts']
        '''
        optimizer = "sgd"
        '''
        implemented_optimizers = ["adamw", 'sgd', "rmsprop", "adadelta", "adamax","adam"]
        '''
        testargs = ["prog", "-m", "tests/configs/test_run_v2.cfg", "-d", "tests/configs/Test.data", "-e", epochs,
                    "--n_cpu", "2","--pretrained_weights","weights/yolov3-tiny.weights","--evaluation_interval","3",
                    "-g",gpu,"--seed",seed,"--scheduler",scheduler,'--optimizer',optimizer,"--name",name,
                    "--test_cycle","True"]
        with patch.object(sys, 'argv', testargs):
            setup = self.get_setup_file()
            hyp_config = parse_hyp_config(setup.hyp)
            data_config = parse_data_config(setup.data)
            assert run(setup,data_config,hyp_config, ver: 'test_cuda') == f"Finished training for {epochs} epochs, with {optimizer} optimizer and {scheduler} lr sheduler"
```

Kuva 39 – Esimerkki yksinkertaisesta testausskriptistä, jonka avulla voidaan testata kouluttimen toiminta

4 Tulokset

Edellisessä kappaleessa kuvatun toteutuksen avulla saatiin koulutettua malli, joka suurimmalta osin kykenee tunnistamaan kohteita vaadituilla tarkkuuksilla. Puutteita tarkkuudessa esiintyy lähinnä muutamien tiettyjen luokkien osalta. Tämä johtuu koulutusmateriaalin vähyydestä näiden luokkien osalta. Tätä koulutusmateriaalin epätasapainoa ei siis pystytty korjaamaan täysin edellisessä kappaleessa esitettyjen toimenpiteiden avulla. Kuitenkin näitä puutteita voidaan korjata käytännössä kahdella tapaa: lisäämällä koulutuskierroksia sopivan koulutusdatan lataajan avulla ja/tai lisäämällä kyseisten luokkien koulutusmateriaalia.

Mallin suorituskykytestit tehtiin suoraviivaisesti siten, että tulkittavaa materiaalia kerättiin tarvittava määrä, jonka jälkeen mallille annettiin tehtäväksi tunnistaa kaikki kuvat. Käytännössä testit osoittivat, että malli kykenee tunnistamaan vaaditun määrän kuvia noin ¼ vaaditusta ajasta. On kuitenkin huomioitavaa, että käytännössä malli vaatii grafiikkakiihdyttimen resursseja saavuttaakseen hyväksyttävän suorituskyvyn. Pelkällä prosessorilla tunnistusta tehtäessä tunnistusaika oli keskimäärin noin kaksinkertainen vaadittuun verrattuna.

Mallin kouluttamisessa käytetyt työkalut ovat esitetty tämän työn ohessa. Myös niihin tehdyt muutokset on dokumentoitu, joko ohjelman käyttöohjeessa tai tässä dokumentissa. Käytännössä mitään oleellisia muutoksia käytettyihin avoimen lähdekoodin työkaluihin ei tämän työn aikana tehty, lähinnä jotain kosmeettisia muutoksia käyttäjähallintaan tai tietojen tallentamiseen, mutta nämäkin ovat dokumentoitu ko. ohjelmien käyttöohjeissa (Vaatimus 6). Käytännössä kaikki tämän työn ohessa tehdyt työkalut ovat käytettävissä myös suljetuissa ympäristöissä, sillä oletuksella että kyseessä olevaan ympäristöön voidaan tuoda ulkoverkosta (internet tai vast.) tietyt kouluttimen ja tunnistimen tarvitsemat kirjastot (Vaatimus 7).

Toteutuksen aikana suurimmat kohdatut haasteet liittyivät seuraaviin tekijöihin:

ClearML ei ollut ennakko-odotuksista huolimatta niin käytännöllinen työkalu koulutuksen monitorointiin kuin oli alun perin oletettu. Tämä johtui lähinnä siitä, että ClearML:n sisäinen tiedonhallinta varsinkin mallien säilömisen osalta ei ole kaikkein helpoimmin hallittavissa. Käytännössä ClearML:n arkkitehtuuri ei suoraan tue esimerkiksi vanhan mallin ylikirjoitusta, vaan vanha malli pitää ensin löytää kannasta ja sitten poistaa kannasta ja vasta tämän jälkeen tallennetaan uusi malli kantaan. Tämä vaihe aiheutti usein kohtuullisen pitkiä viiveitä mallin tallentamisen ja poistamisen välillä, joka johti muutamia kertoja siihen, että malleja puskuroitiin muistiin, joka taas aiheutti välimuistin täyttymisiä. Mikäli kantaa ei siivottu aina mallin tallentamisen aikana, aiheutui usein tilanteita, jossa kantaan tallennettiin hyvin paljon malleihin liittyvää tietoa (lähinnä mallien kuvauksia), joka johti usein siihen, että ClearML:n palvelin koneelta loppui tallennustila kesken. Näistä

syistä johtuen mallien tallentamista ja koulutusten seuraamista ei työn loppuvaiheessa enää tehty ClearML:llä vaan Tensorboard:lla ja tallentamalla mallit suoraan koneelle, jolla koulutus tehtiin.

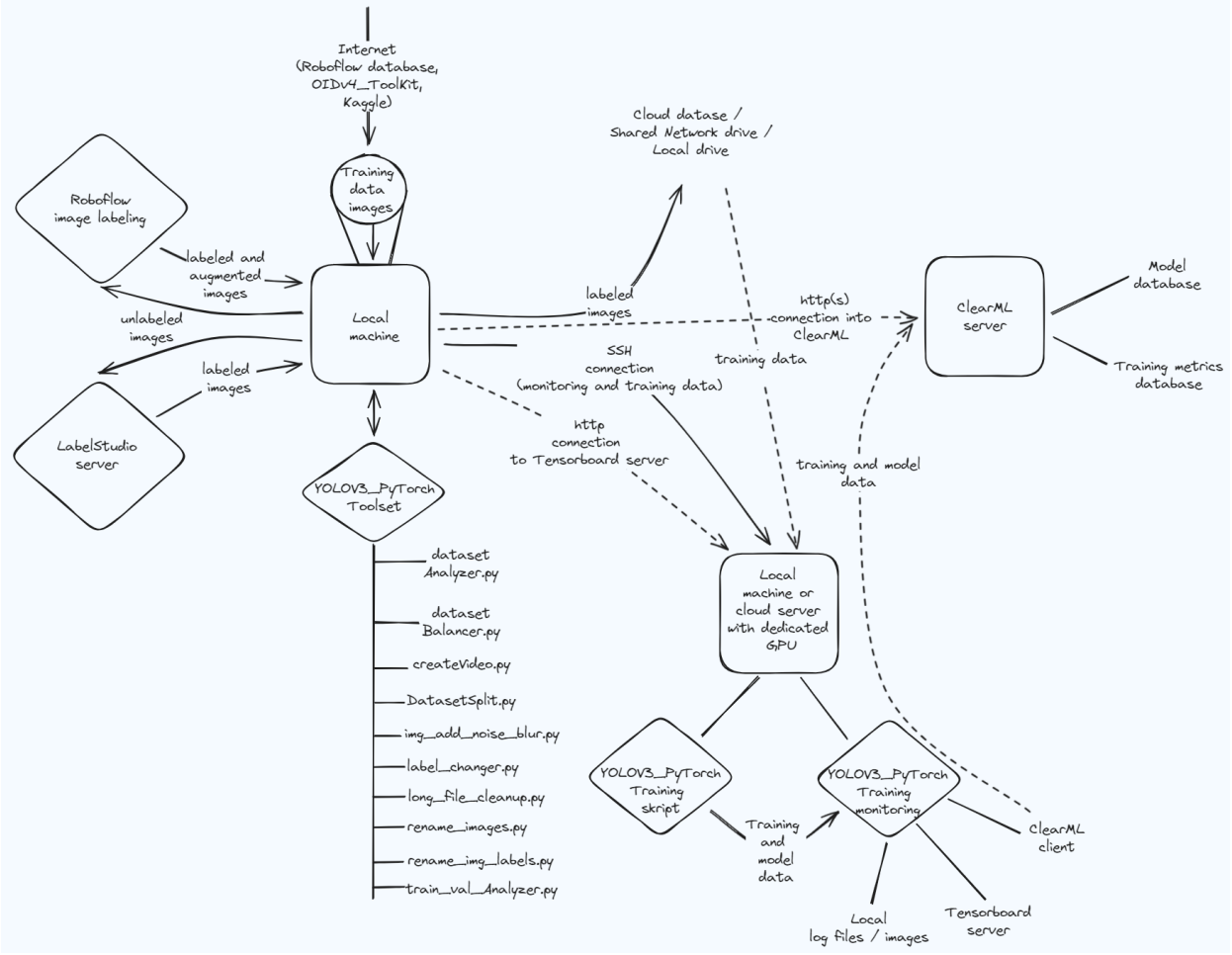
Toinen kohdattu haaste oli jo aikaisemmin mainittu koulutusmateriaalin epätasapaino, jonka korjaaminen vei oletettua huomattavasti enemmän aikaa. Tämä ongelma on sinänsä enemmänkin riippuvainen koulutusmateriaalista kuin itse kouluttimesta, mutta kuitenkin on mielestäni hieman epärealistista olettaa, että koulutusmateriaalin voidaan aina olettaa olevan erittäin hyvin tasapainossa, joten tekniset muutokset ja parannukset tämän asian parantamiseksi olivat paikallaan. Kuitenkin on myös todettava, että tätä ongelmaa ei ehkä osattu ottaa tarpeeksi hyvin huomioon työtä suunniteltaessa, jolloin ongelmasta pääsi tulemaan ehkä turhankin vaikuttava.

Kolmantena haasteena voidaan pitää tehdyn koodin optimoinnin ongelmaa kouluttimen ja tunnistimen osalta siinä mielessä, että varsinkin kouluttimen osalta työtä valmisteltaessa ei osattu ottaa ehkä tarpeeksi hyvin huomioon, että kouluttimenkin optimointi olisi tärkeää. Tämä ongelma johtaa juurensa siihen tosiasiaan, että koulutusmateriaalin määrä kasvoi koulutuksen tietyssä vaiheessa erityisen suureksi. Kun koulutusmateriaalin epätasapainoa pyrittiin korjaamaan, kouluttimen optimaalinen käyttö koodin tasolla korostui.

Toteutuksen osalta suurimpina puutteina voidaan pitää jo aikaisemmin mainittuja mallin tarkkuuden suorituskyvyn puutteita tiettyjen luokkien osalta. Muutoin koodillisesti ehkä suurimpia mainittavia puutteita ovat koodin kommentoinnin puutteellisuus, koodin siivous ja perusteellisempi dokumentointi, mikäli koodia aiotaan käyttää tuotantokäytössä.

4.1 Koulutusjärjestelmän yleisarkkitehtuuri

Koulutusjärjestelmän yleisarkkitehtuuri (Kuva 40) pyrittiin pitämään alusta alkaen mahdollisimman yksinkertaisena, jotta kokonaisuus olisi mahdollisimman helppo ottaa käyttöön myös suljetussa ympäristössä. Lisäksi yksinkertaisuutta puolsi työn määrä, jota piti joka tapauksessa pyrkiä rajoittamaan, joten monimutkaisuus korkealla arkkitehtuuritasolla ei ollut tavoitteen mukaista. Alkuperäisessä suunnitelman mukaisessa arkkitehtuurissa pysyttiinkin kohtuullisen hyvin, käytännössä vain ClearML:n käytöstä luovuttiin käytännön syistä ja tämän tilalle ei varsinaisesti rakennettu muuta uutta tai korvaavaa toiminnallisuutta, paitsi laajennettiin Tensorboard:n kykyä esittää aikaista enemmän tietoa mallista ja koulutuksen tilanteesta. Toinen selkeä muutos / lisäys alkuperäiseen suunnitelmaan oli koulutusdatan tallentaminen pilvipalveluun, jolloin sen käyttäminen ja ylläpitäminen oli hieman helpompaa.



Kuva 40 – Mallin koulutusjärjestelmän yleisarkkitehtuuri

4.2 Kouluttimen sisäinen arkkitehtuuri

Kouluttimen sisäinen arkkitehtuuri koki työn aikana hyvin paljon muutoksia, jotka ovat tarkemmin kuvattu aikaisemmissa kappaleissa. Suurimmat muutoksia aiheuttavat työt olivat kouluttimen ajastimien, optimoijien ja koulutusmateriaalin lataajien päivitykset, sekä kuvien manipuloinnin implementointi. Nämä työt eivät olleet varsinaisesti suunniteltu tehtäväksi tässä vaiheessa vaan mahdollisesti vasta seuraavassa kehitysvaiheessa, mikäli asiakas olisi niin halunnut. Kuitenkin mallin koulutuksen haasteiden takia nämä vaiheet jouduttiinkin tekemään jo tässä vaiheessa, jolloin alkuperäinen työmäärä kasvoi huomattavasti. Käytännössä alkuperäinen suunnitelma oli luoda mallin kouluttimesta YOLOv3 arkkitehtuurin mukainen, kuitenkin näiden tehtyjen muutosten myötä mallin kouluttimen arkkitehtuuri on lähempänä YOLOv4:n arkkitehtuurin mukaista arkkitehtuuria.

4.3 Tunnistimen yleisarkkitehtuuri

Tunnistin noudattelee suunnitelman mukaista arkkitehtuuria, joka perustuu YOLOv3:n arkkitehtuuriin. Muutoksia on lähinnä tehty optimoinnin kannalta, sekä tarvittavilta osin datan lataajan osalta kuin on tarvittu, jotta tunnistin kykenee käyttämään YOLOv3 ja YOLOv4:n mukaisia malleja tunnistamiseen (Liite 7 ja Liite 8).

4.4 Yhteenveto vaatimuksista ja toteutuksesta

Tehtyä toteutusta testattiin eri tavoin kehityksen aikana (Kuva 41). Kehityskierroksen aikana koodia testattiin pääosin ensin yksikkötestien avulla ja kun kehityskierroksen eri vaiheet olivat läpäisseet omat yksikkötestinsä, aloitettiin koko koulutusjärjestelmän testaus. Koko järjestelmää testattiin kouluttamalla yksinkertaista neuroverkkoa, jonka tehtävänä oli tunnistaa kaksi eri kohdetta (head ja helmet). Tämän testauskoulutuksen aikana tarkkailtiin seuraavia asioita:

- Järjestelmä laskee lämmittelykierrosten määrän oikein
- Järjestelmä laskee maksimi iteraatioiden määrän oikein
- Järjestelmä päivittää oppimiskerrointa (learning rate) asetettujen parametrien mukaisesti
- Järjestelmä tuottaa halutut raportit ja lokitiedostot automaattisesti
- Järjestelmä tallentaa neuroverkosta koulutuksen aloituksessa käytetyt painotukset, viimeisimmät painotukset, sekä parhaimman tunnistuksen tuottaneet painotukset
- Järjestelmä suorittaa koulutuksen tarkkailua automaattisesti ja käynnistää neuroverkon evaluoinnin määräajoin, sekä aina kun koulutuksen hyvyysarvo on parempi kuin paras siihen mennessä.

Kun järjestelmän toimivuus oli todennettu testauskoulutuksessa, aloitettiin varsinainen neuroverkon koulutus oikealla koulutusmateriaalilla. Tämän koulutuksen aikana tarkkailtiin pääosin koulutettavan neuroverkon hyvyyttä kouluttimen tuottamien raporttien perusteella, sekä automaattisesti ja määräajoin ajettavien neuroverkon evaluointi testien avulla. Mallin evaluointitesti tuotti joka kerta seuraavat mallin hyvyyttä/kyvykkyyttä kuvaavat arvot:

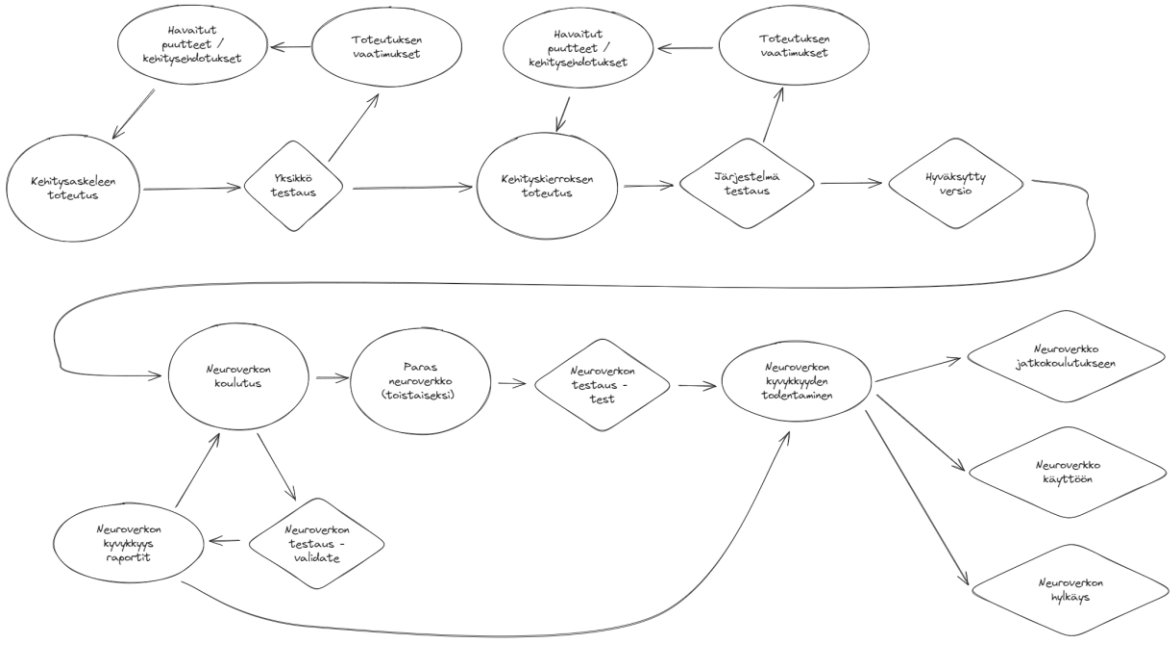
- Mean average precision, mAP
- Average precision / tunnistusluokka
- F1 arvo
- Precision
- Recall
- Training fitness
- Model fitness
- Sekaannusmatriisi

Kun neuroverkon koulutus oli saatu päätökseen, valittiin koulutuksen aikana parhaat tulokset antanut neuroverkko varmistustesteihin. Varmistustesteissä neuroverkkoa testattiin kuvasarjalla, jonka kuvia verkko ei ollut aikaisemmin nähnyt. Tämän jälkeen varmistustestien tuloksia verrattiin koulutuksen aikana saatuihin mallin hyvyyttä/kyvykkyyttä kuvaaviin arvoihin ja luotiin suunnitelma siitä käytetäänkö saatua mallia jatkokoulutukseen vai hylätäänkö malli ja aloitetaan koulutus uudestaan uusilla parametreillä.

Videolta tunnistamisen hyvyyttä (käytännössä nopeutta) testattiin vapaasti saatavilla olevilla videoilla liikennevirroista, sekä siten että asiakkaan kuvamateriaalista koostettiin kuvia yhdistelemällä muutaman minuutin pituinen video. Tämän testauksen tavoite oli kaksiosainen:

- Testata videolta tunnistamisen teknistä kyvykkyyttä, eli kuinka nopeasti koodi pystyy käsittelemään videota
- Testata halutun neuroverkon nopeutta vaativassa tunnistustapauksessa, jossa kuvia syötetään nopeana kuvavirtana

Neuroverkon referenssinä tässä testauksessa käytettiin valmiiksi koulutettua neuroverkko (YOLOv3-tiny tai YOLOv4-tiny), jonka avulla pyrittiin erottelemaan, johtuiko mahdollinen tunnistamisen hitaus tunnistimen koodin hitaudesta vai itse mallin hitaudesta. YOLO-tiny-neuroverkot ovat koulutettu nopeaa tunnistamista (kuten videolta tunnistamista) vaativiin tilanteisiin, jolloin näiden verkkojen käyttäminen referenssinä oli hyvin luontevaa. Tämän testauksen (Kuva 41) perusteella tehtiin päätelmiä siitä pitäisikö videon tunnistimen koodia itsessään parantaa vai johtuiko mahdollinen hitaus koulutetusta neuroverkosta.



Kuva 41 – Prosessikuvaus kouluttimen testauksesta ja neuroverkon testauksesta

Taulukossa 1 on esitetty yhteenveto siitä mitä eroja suunnitellun toteutuksen ja lopullisen toteutuksen välillä on keskittyen keskeisiin eroihin toteutuksien välillä.

Taulukko 1 – Suunnitellun ja lopullisen toteutuksen väliset erot keskeisiltä osin.

Ominaisuus	**Suunniteltu toteutus**	**Lopullinen toteutus**
Piirteiden löytäminen	Darknet-53	CSPDarknet-53
Kohteen paikan etsin	Ankkureihin perustuva (Dimensio klusterointi)	Ankkureihin perustuva
Ankkurointilaatikoiden määrä	Yksi jokaista totuuskohdetta kohti.	Useita laatikoita yhtä totuuskohdetta kohti.
IoU Kynnys	Yksi arvo (0.5)	Yksi arvo (0.2)
Kuva-manipulaatio	Rotation, Saturation, Exposure, Hue	Rotation, Exposure, HSV-Hue, HSV-Saturation, HSV-Value, Translation, Scale, Shear, Perspective, Flip (Up-Down + Left-Right), Mosaic, Mixup
Koulutuksen lämmittely	Yksinkertainen, perustuen manuaaliseen arvoon	Laskettu automaattisesti perustuen kuvajoukkojen ja koulutuskierrosten määrään.
Optimoijat	SGD	SGD, ADAM, ADAMW, RMSPROP, ADADELTA, ADAMAX
Ajastimet	Ei	CosineAnnealingLR, ChainedScheduler, ExponentialLR, ReduceLROnPlateau, ConstantLR, CyclicLR, OneCycleLR, LambdaLR, MultiplicativeLR, StepLR, MultiStepLR, LinearLR,

Ominaisuus	Suunniteltu toteutus	Lopullinen toteutus
		PolynomialLR, CosineAnnealingWarmRestarts
Koulutusmateriaalin näytteistimet	Ei	SequentialSampler, RandomSampler, SubsetRandomSampler, WeightedRandomSampler, BatchSampler
Skaalaaja	Ei	Gradscaler
Testaus automaatio	Ei	Basic scenarios for each optimizer
Monitorointi	Matplotlib, Tensorboard, ClearML	Matplotlib, Tensorboard, ClearML
Kuvajoukkojen koko	Manuaalinen	Automatisoitu
GPU Muistin hallinta	Ei	Kuvajoukon koon rajoitin, aktiivinen valvonta
Tunnistus	Kuva	Kuva ja video

Taulukossa 2 on esitetty se, miten asiakaan vaatimukset saatiin täytettyä lopullisessa toteutuksessa.

Taulukko 2 – Vaatimusten toteutuminen lopullisessa toteutuksessa

Nro	Vaatimus	Toteutuminen (Kyllä, Ei, Osittain)	Huomiot
1	Mallin on kyettävä tunnistamaan kohteet ja niiden aspektit/liikesuunnat vähintään mAP_0.5>=65% tarkkuudella.	**Osittain**	Osa tunnistusluokista jää vaaditun tarkkuuden alle.
2	Mallin on pystyttävä tunnistamaan kohteet f1 -arvolla >= 0.65%	**Osittain**	Osa tunnistusluokista jää vaaditun tarkkuuden alle.

Nro	Vaatimus	Toteutuminen (Kyllä, Ei, Osittain)	Huomiot
3	Mallin on pystyttävä luokittelemaan ajoneuvot seuraavien kategorioiden mukaisesti: • auto/pakettiauto (car) • rekka (truck) • bussi (bus) • moottoripyörä (motorcycle) • maatalouskone (farm vehicle)	Kyllä	Malli kykenee erottelemaan ajoneuvot haluttujen kategorioiden mukaisesti.
4	Mallin on pystyttävä tunnistamaan ajoneuvon suunta/aspekti seuraavien kategorioiden mukaisesti: • away • towards • left • right	Kyllä	Malli kykenee erottelemaan ajoneuvot haluttujen kategorioiden mukaisesti.
5	Mallin on pystyttävä tunnistamaan ja luokittelemaan y kuvaa x minuutissa	Kyllä	Suorituskykytesteissä saavutettiin riittävä suorituskyky
6	Mallin kouluttamisessa käytetyt avoimen lähdekoodin työkalut dokumentoidaan ja mikäli näitä työkaluja on muokattu tai päivitetty työn mahdollistamiseksi, tehostamiseksi tai muutoin, toimitetaan päivitetyt versiot asiakkaalle.	Kyllä	Työkaluja ei tarvinnut muokata työn aikana, käytetyt työkalut ovat mainittu tämän dokumentin kappaleessa x.
7	Kuvantunnistusmallin koulutinta on pystyttävä käyttämään myös suljetussa ympäristössä, jossa ei ole internet yhteyttä.	Kyllä	Koulutin ja sen tarvitsemat kirjastot, sekä aputyökalut voidaan viedä suljettuun ympäristöön.

5 Pohdinta

Suurin osa tavoitteista saatiin toteutettua. Työn aikana on myös noussut esiin melko paljon erilaisia näkökulmia ja kehityksen kohteita niin prosessikehityksen kuin teknisen kehityksenkin osalta. Suurin kehityksen kohde prosessi- ja valmistelumielessä olisi mielestäni jonkinlaisen koulutussuunnitelman tai strategian luominen jo siinä vaiheessa, kun uuden mallin koulutusta suunnitellaan. Tämä strategia voisi pitää sisällään käytännössä kaikki uuden mallin koulutuksen oleelliset vaiheet aina koulutusmateriaalin keräämisestä ja analysoinnista, aina mallin testaamiseen ja tuotantoon viemiseen asti ainakin niiltä osin, kun on tarpeellista ja mahdollista kuvata. Tämä lähestymistapa mahdollistaisi ehkä johdonmukaisemman koulutusprosessin mallin koulutuksen eri vaiheissa.

Koulutusmateriaalin nimikoinnin osalta kenties keskeisin jatkokehityksen kohde olisi puoliautomaattisen nimikoinnin kehittäminen käytettävään työkaluun (esimerkiksi LabelStudio:n), joka jo käytännössä tukee tätä toiminnallisuutta. Tällöin työ olisi lähinnä sopivan infrastruktuurin rakentaminen tämän toiminnallisuuden ympärille ja käyttöön ottamisen ohjeistaminen. Tämä vaihe säästäisi huomattavasti aikaa siinä tilanteessa, kun koulutusmateriaalia on tarve kerätä paljon, jolloin käytännössä ainoa luonnollinen tapa edetä on inkrementaalinen lähestymistapa siinä mielessä, että kun ensimmäinen kohtuullisen hyvä malli saadaan koulutettua, voidaan sitä käyttää koulutusmateriaalin jatkojalostamiseen automaattisen tunnistamisen apuna.

Teknisesti mahdollisia kehityskohteita löytyy hieman enemmän. Yksi tärkeä kehityksen kohde voisi olla parempien koulutuskuvien manipulointityökalujen implementointi. Näiden avulla olisi mahdollista saada helpommin monimuotoisuutta koulutusmateriaaliin sekä mahdollisesti hallinnoida nykyistä toteutusta paremmin sitä, missä tilanteessa kuvia muokataan. Lisäksi koulutusmateriaalin lataajaan (dataloader) jatkokehitys siten, että lataaja jatkossa pitäisi kirjaa siitä millaisia koulutuskuvia/luokkia mallin kouluttamisessa on käytetty, jotta tasapainoa voitaisiin säätää koulutuksen aikana. Tähän liittyen myös jo koulutuksen alkuvaiheessa olisi ehkä hyvä tehdä nykyistä tarkastelua laajempi tarkastus koulutusmateriaalille, jotta mahdolliseen epätasapainoon voitaisiin puuttua ennen koulutuksen aloittamista. Myös koulutusmateriaalin ja tarvittavien konfiguraatioiden hallintaan ja lataamiseen voisi tehdä parannuksia, siten että jatkossa näiden hallintaan käytettäisiin esimerkiksi json tai yaml pohjaista formaattia. Tämä poistaisi melko suuren määrän tekstipohjaista parsintaa, johon YOLOv3 version konfiguraatiot kirjoitetaan. Lisäksi olisi hyvä tarkastella onko tarvetta mahdollistaa mallin kouluttaminen monella grafiikkakiihdyttimellä samaan aikaan, jolloin mallin koulutusaikaa saadaan vähennettyä huomattavasti. Tämän vaihtoehdon loogisuus on toki hyvin riippuvainen käytössä olevista resursseista ja niiden kapasiteetista, mutta vaihtoehtona tutkimisen arvoinen.

Lisäksi mallin koulutuksen seurantaan ja monitorointiin voisi olla hyvä tehdä päivityksiä ClearML:ssä havaittujen ominaisuuksien vuoksi. Varteenotettavia vaihtoehtoja voisivat olla esimerkiksi MLflow, Weights and Biases tai Tensorboard:n päivitys tukemaan vielä nykyistä suurempaa tietomäärää. Lisäksi lokitietojen keräämiseen liittyen olisi hyvä korjata tai päivittää Matplotlib kirjastoon liittyviä muistivuotoja ja tarvittaessa päivittää koko lokitietojen kerääminen käyttämään Seaborn -kirjastoa.

Edellä mainittujen kohtien lisäksi olisi hyvä pohtia uusien tunnistusmallien arkkitehtuurien määrittelyä, jotta mahdollistettaisiin erilaisiin käyttötarkoituksiin soveltuvien mallien tuotanto.

Lähteet

Alom Z, Taha T M., Yakopcic C, Westberg S, Sidike P, Nasrin S, Hasan M, Van Essen B C., Awwal A A. S., Asari V K. 2019. A State-of-the-Art Survey on Deep Learning Theory and Architectures, s.14-45.

Alsharabi N. 2023. Real-Time Object Detection Overview: Advancements, Challenges, and Applications. Luettavissa: https://www.researchgate.net/publication/375599607_Real-Time_Object_Detection_Overview_Advancements_Challenges_and_Applications. Luettu: 02.02.2024

Ammar A, Koubaa A, Ahmed M, Saad A, Benjdira B. 2021. Aerial Images Processing for Car Detection using Convolutional Neural Networks: Comparison between Faster R-CNN and YoloV3, s. 9.

Anand J, Divyakant M. 2020. A Comparative Study of Various Object Detection Algorithms and Performance Analysis. Luettavissa: https://www.researchgate.net/publication/346346964_A_Comparative_Study_of_Various_Object_Detection_Algorithms_and_Performance_Analysis. Luettu: 30.01.2024

Bansal Ankit. 2019. Class Lists for well-known Object Detection Datasets. Luettavissa: https://medium.com/@a7b/class-lists-for-well-known-object-detection-datasets-27be67e5db39. Luettu: 19.03.2024

Bochkovskiy. A, Wang C-Y, Liao H-Y M. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection, s. 1-5.

Brownlee Jason. 2020. 4 Types of Classification Tasks in Machine Learning. Luettavissa: https://machinelearningmastery.com/types-of-classification-in-machine-learning/. Luettu: 19.03.2024

Cetinsoy A. 2022. Democratizing Object Detection. Luettavissa: https://blog.bigml.com/2022/06/09/democratizing-object-detection/. Luettu: 20.01.2024

Chaudhuri A. 2023. Smart Traffic Management of Vehicles using Faster R-CNN based Deep Learning Method, s. 6-7.

Chen W, Li Y, Tian Z, Zhang F. 2023. 2D and 3D object detection algorithms from images: A Survey, s. 1.

Coursesteach. 2024. Deep Learning (Part 27)-Backpropagation Intuition. Luettavissa: https://medium.com/@Coursesteach/deep-learning-part-27-backpropagation-intuition-657275fd1960. Luettu: 19.03.2024

Dilgemani Cem. 2024. What is Data Augmentation? Techniques & Examples in 2024. Luettavissa: https://research.aimultiple.com/data-augmentation/. Luettu: 19.03.2024

Gidaris S, Komodakis N. 2015. Object detection via a multi-region & semantic segmentation-aware CNN model, s. 1-2.

Girshick. 2015. Fast R-CNN, s. 1-2.

Glassner A 2021a. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-2.png. Luettu: 30.01.2024

Glassner A 2021b. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-4.png. Luettu: 30.01.2024

Glassner A 2021c. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-6.png. Luettu: 30.01.2024

Glassner A 2021d. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-7.png. Luettu: 30.01.2024

Glassner A 2021e. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-10.png. Luettu: 30.01.2024

Glassner A 2021f. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-11.png. Luettu: 30.01.2024

Glassner A 2021g. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-23.png. Luettu: 30.01.2024

Glassner A 2021h. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-24.png. Luettu: 30.01.2024

Glassner A 2021i. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-31.png. Luettu: 30.01.2024

Glassner A 2021j. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-32.png. Luettu: 30.01.2024

Glassner A 2021k. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-36.png. Luettu: 30.01.2024

Glassner A 2021l. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/13-37.png. Luettu: 30.01.2024

Glassner A 2021m. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/14-24.png. Luettu: 30.01.2024

Glassner A 2021n. Deep Learning - A Visual Approach. Luettavissa: https://github.com/blueberry-music/Deep-Learning-A-Visual-Approach/blob/main/Figures/Images/17-10.png. Luettu: 30.01.2024

Infosys BPM s.a. Automated data labelling versus manual data labelling. Luettavissa: https://www.infosysbpm.com/blogs/annotation-services/automated-data-labelling-vs-manual-data-labelling.html. Luettu: 19.03.2024

Kallio R. 2021. Neuroverkon opettaminen vastavirta-algoritmilla. Luettavissa: https://urn.fi/URN:NBN:fi-fe2021052030929. Luettu: 30.01.2024

Karimi Z. 2021. Confusion Matrix. Luettavissa: https://www.researchgate.net/publication/355096788_Confusion_Matrix. Luettu: 03.02.2024

Knok Z, Pap K, Hrncic M. 2019. Implementation of intelligent model for pneumonia detection, s. 4.

Labelstudio. 2022. Video Object Tracking & New Labeling UI—Label Studio 1.6 Release. Luettavissa: https://labelstudio.substack.com/. Luettu: 18.03.2024

Mathcentre. 2009. Introduction to functions. Luettavissa: https://www.mathcentre.ac.uk/resources/uploaded/mc-ty-introfns-2009-1.pdf. Luettu: 27.02.2024

McCoubrey L E., Elbadawi M, Orlu M, Gaisford S, Basit A W. 2021. Harnessing machine learning for development of microbiome therapeutics, s. 5-6.

Mishra Manyank. 2020. Convolutional Neural Networks, Explained. Luettavissa: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939. Luettu: 19.03.2024

Nummela T. 2022. OBJECT DETECTION WITH NVIDIA JETSON NANO. Luettavissa: https://urn.fi/URN:NBN:fi:amk-202204276128. Luettu: 05.02.2024

Orenda Minds 2019. What Is Annotation In AI / ML? Luettavissa: https://medium.com/@orenda-minds/what-is-annotation-in-ai-ml-df63af8599e8. Luettu: 19.03.2024

Paananen V. 2018. Neuroverkkojen FPGA-toteutus. Luettavissa: https://urn.fi/URN:NBN:fi:oulu-201805312377. Luettu: 30.01.2024

Parmar Ravindra. 2018. Common Loss functions in machine learning. Luettavissa: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23. Luettu: 19.03.2024

Redmon J, Divvala S, Girshick R, Farhadi A. 2016. You Only Look Once: Unified, Real-Time Object Detection, s. 1.

Redmon J, Farhadi A. 2018. YOLOv3: An Incremental Improvement, s. 1-3.

Roboflow. 2024. How to Label Image Data for Computer Vision Models. Luettavissa: https://blog.roboflow.com/tips-for-how-to-label-images/. Luettu: 18.03.2024

Shukla L. 2019. Designing Your Neural Networks. Luettavissa: https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed. Luettu: 20.01.2024

Sjödin D, Palmie M. 2021. How AI capabilities enable business model innovation: Scaling AI through co-evolutionary processes and feedback loops, s. 4.

Soni D. Nord R L. Hofmeister C. 1995. Software Architecture in Industrial Applications, s. 1-5.

Stanford University s.a. What is Computer Vision? Luettavissa: https://ai.stanford.edu/~syyeung/cvweb/tutorial4.html. Luettu: 21.01.2024

Stanford University s.b. Convolutional Neural Network. Luettavissa: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. Luettu: 19.03.2024

Subramanyam Vineeth S. 2021. Basics of Bounding Boxes. Luettavissa: https://medium.com/analytics-vidhya/basics-of-bounding-boxes-94e583b5e16c. Luettu: 19.03.2024

Szeliski R. 2021. Computer Vision: Algorithms and Applications. Luettavissa: https://szeliski.org/Book/. Luettu: 06.12.2023

Tasnim S, Qi W. 2023. Progress in Object Detection: An In-Depth Analysis of Methods and Use Cases. Luettavissa: https://www.researchgate.net/publication/372797767_Progress_in_Object_Detection_An_In-Depth_Analysis_of_Methods_and_Use_Cases. Luettu: 02.02.2024

Tensorflow s.a. Tinker with a Neural Network Right Here in Your Browser. Luettavissa: http://playground.tensorflow.org/. Luettu: 27.02.2024

Thakur N, Nagrath P, Jain R, Saini D. 2021. Object Detection in Deep Surveillance. Luettavissa: https://www.researchgate.net/publication/355895501_Object_Detection_in_Deep_Surveillance. Luettu: 21.01.2024

Ullah I, Wahab F, Choi A, Khan R A. 2022.Design and implementation of real-time object detection system based on single-shoot detector and OpenCV, s. 4-5.

University of Liverpool 2021. Why artificial intelligence is so important in today's world. Luettavissa: https://online.liverpool.ac.uk/why-artificial-intelligence-is-so-important-in-todays-world/. Luettu: 19.01.2024

West M. D, Allen J R. 2018. How artificial intelligence is transforming the world. Luettavissa: https://www.brookings.edu/articles/how-artificial-intelligence-is-transforming-the-world/. Luettu. 19.01.2024

Zhao K, Zhou Y, Chen X. 2020. Object detection: training from scratch, s. 2-5.

Zuo C, Qian J, Feng S, Yin W. 2022. Deep learning in optical metrology: a review, s. 2.

Liitteet

Liite 1. Eri tunnistusmenetelmien välinen kyvykkyys korkean tason vertailuna

Taulukossa 1 esitellään eri tunnistusmenetelmien välisiä eroja käytetyn kehyksen, nopeuden, tarkkuuden ja pääpiirteiden suhteen.

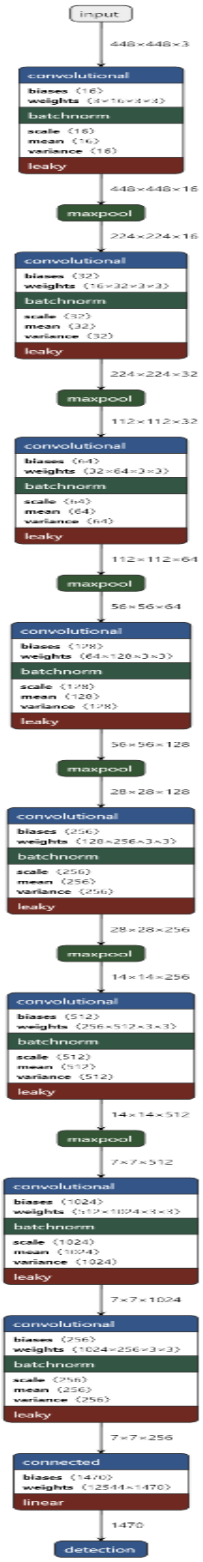
Taulukko 3 – Eri tunnistusmenetelmien välinen kyvykkyys korkean tason vertailuna (Mukaillen Alsharabi. 2023. 10)

Menetelmä	Vuosi	Kehys	Nopeus	Tarkkuus	Pääpiirteet
YOLO (You Only Look Once)	2016	Darknet, YOLOv3, YOLOv4	Very Fast	Moderate to High	Single pass, real-time processing
SSD (Single Shot MultiBox Detector)	2016	Caffe, TensorFlow	Fast	Moderate to High	Multi-scale feature maps, anchor boxes
Faster R-CNN (Region Convolutional Neural Network)	2015	TensorFlow, PyTorch	Moderate	High	Region Proposal Network (RPN) for object proposals
RetinaNet	2017	TensorFlow, PyTorch	Moderate	High	Focal Loss for handling class imbalance
EfficientDet	2019	TensorFlow, PyTorch	Moderate to Fast	High	Scalable and efficient architecture
CenterNet	2019	PyTorch	Fast	Moderate to High	Detects objects as points and regresses to bounding boxes
Detectron2	2019	PyTorch	Moderate to Fast	High	Flexible framework with state-of-the-art models
YOLOv5	2020	PyTorch	Very Fast	High	Efficient architecture, focus on speed
HTC (Hybrid Task Cascade)	2019	TensorFlow, PyTorch	Moderate to Fast	High	Multi-task framework for improved accuracy

Menetelmä	Vuosi	Kehys	Nopeus	Tarkkuus	Pääpiirteet
Sparse R-CNN	2021	PyTorch	Fast	High	Utilizes sparsity for efficient inference
Deformable DETR	2021	PyTorch	Fast	High	Utilizes deformable self-attention
RepPoints	2021	PyTorch	Fast	Moderate	Representing object as points
YOLOX	2021	PyTorch	Very Fast	Moderate to High	SOTA speed-accuracy tradeoff
Sparse R-CNN	2021	PyTorch	Fast	High	Utilizes sparsity for efficient inference

Liite 2. YOLOv1 – tiny neuroverkon arkkitehtuuri

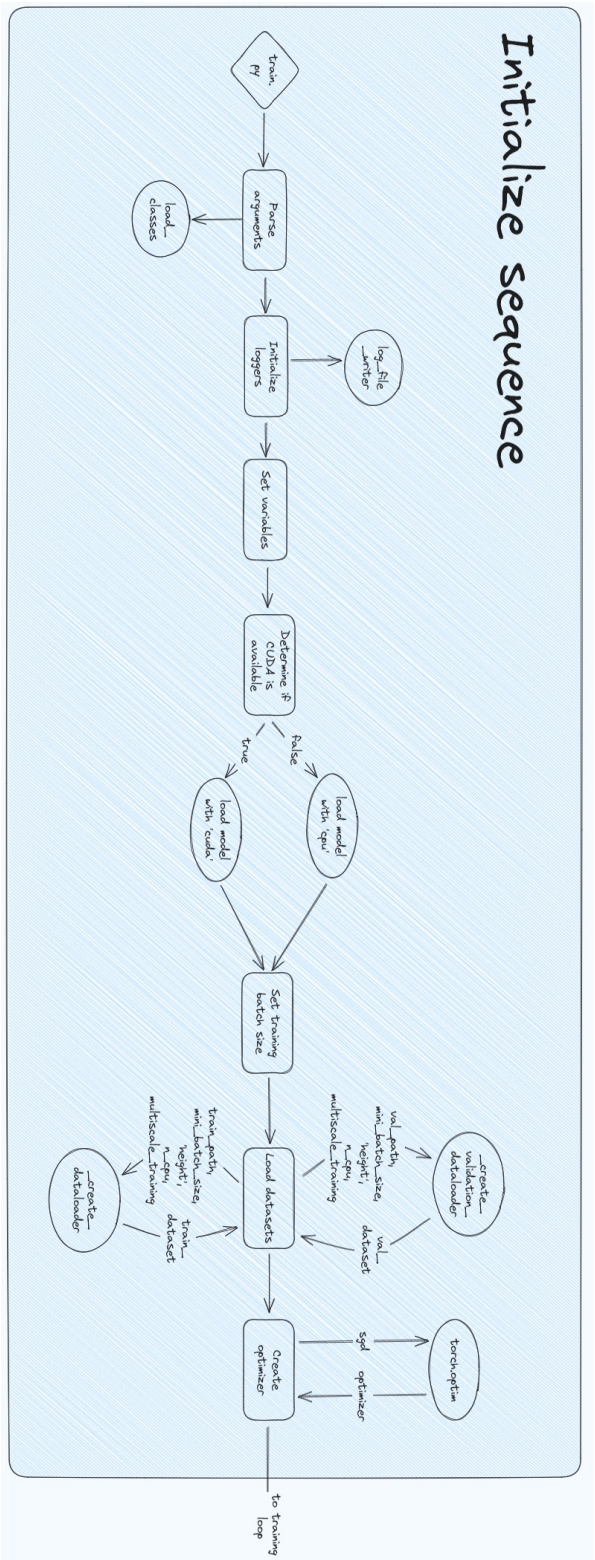
Kuvassa 1 esitellään YOLOv1-tiny arkkitehtuuri, joka toimii perustana myöhemmille YOLO-arkkitehtuuriin perustuville neuroverkoille.



Kuva 1 – YOLOv1-tiny arkkitehtuuri. Mukaillen (Redmon, Divvala, Girshick, Farhadi. 2016, 3)

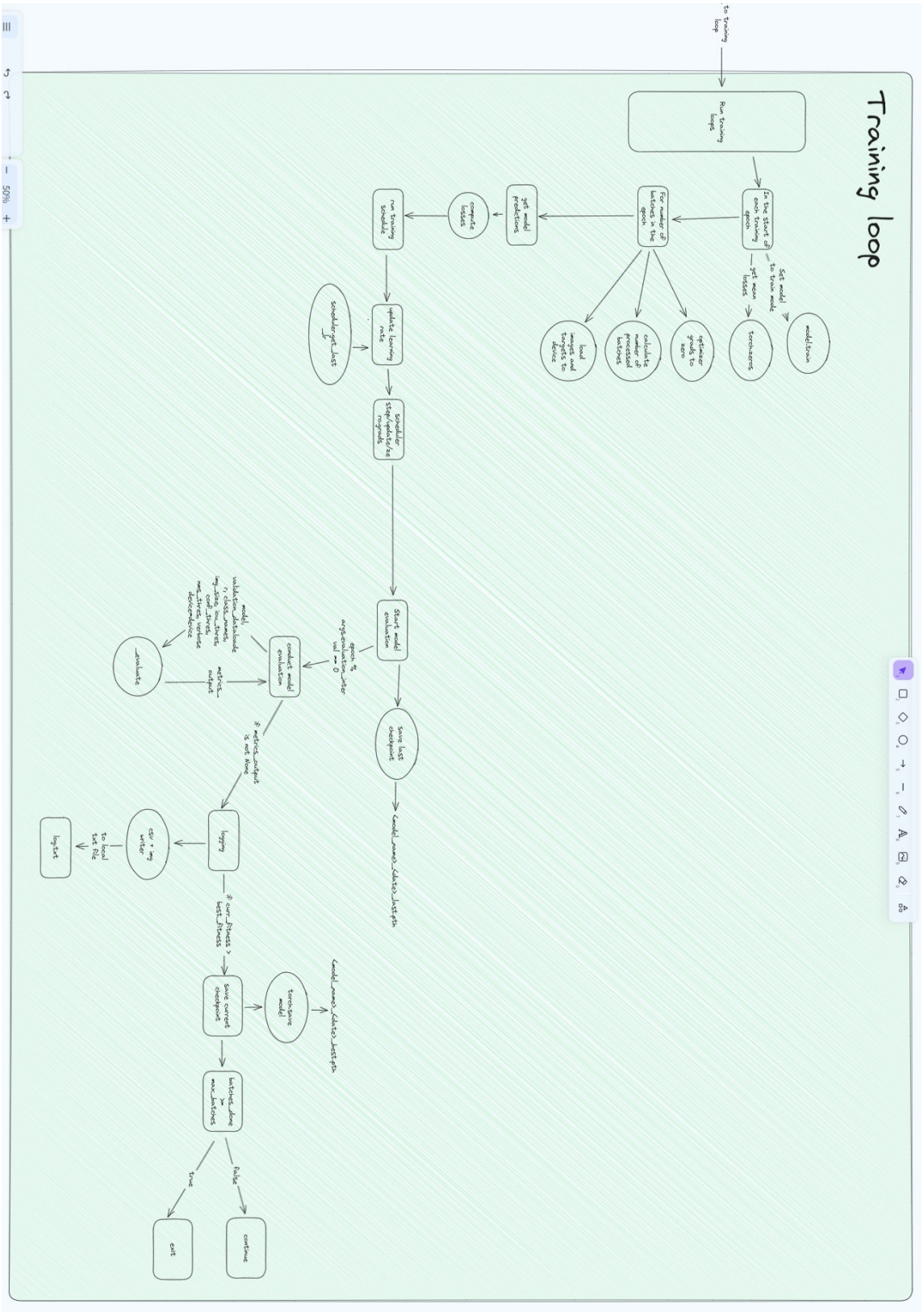
Liite 3. Neuroverkon kouluttimen V0.1 arkkitehtuurikuvaus

Kuvassa 1 on kuvattuna version 0.1 kohteentunnistusmallin kouluttimen käynnistysrutiinin arkkitehtuuri ja prosessit.



Kuva 1 – V0.1 kohteentunnistusmallin kouluttimen käynnistysrutiinin arkkitehtuurikuvaus

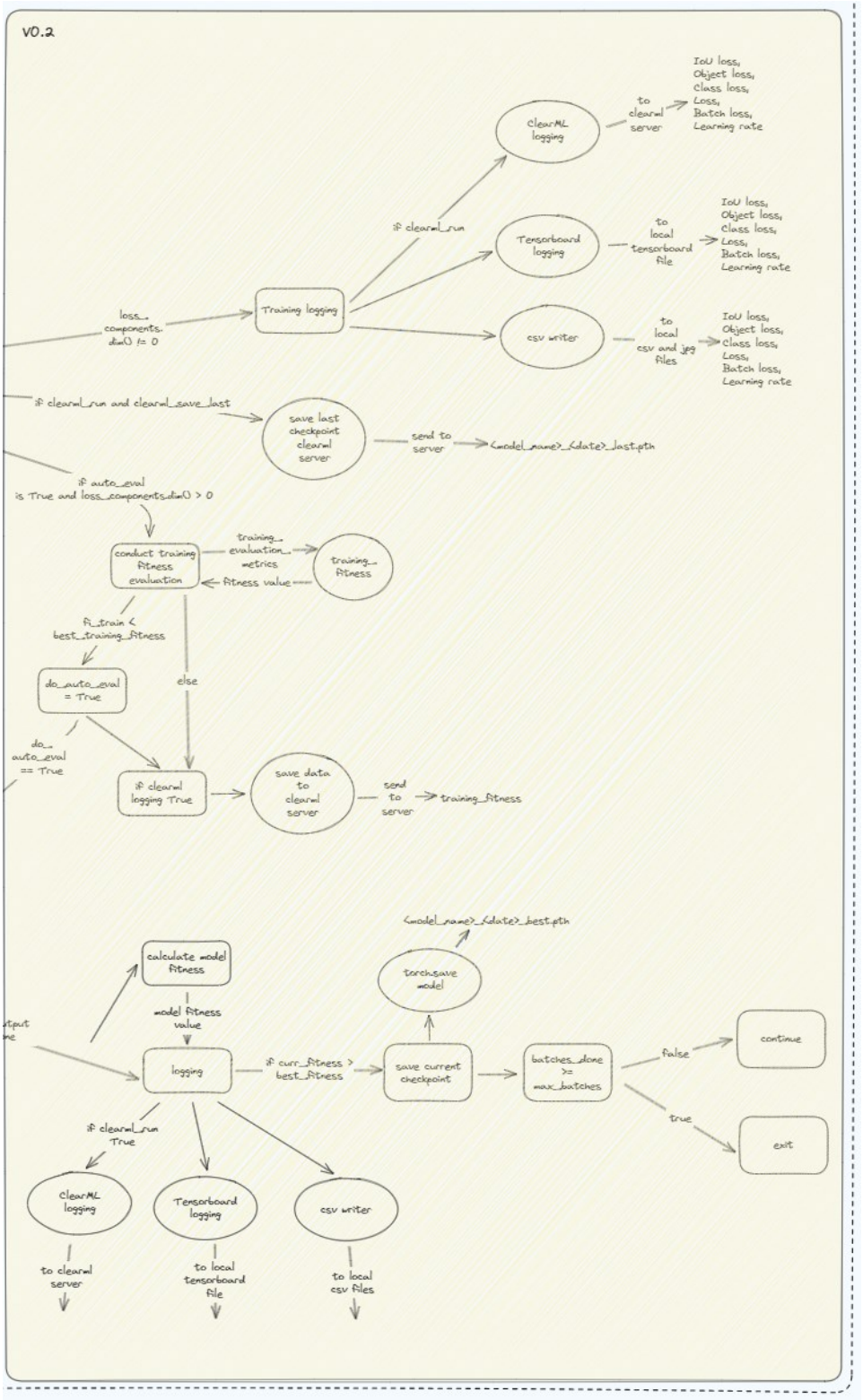
Kuvassa 2 on kuvattuna version 0.1 kohteentunnistusmallin kouluttimen koulutussilmukan arkkitehtuuri ja prosessit.



Kuva 2 –V0.1 kohteentunnistusmallin kouluttimen koulutussilmukan arkkitehtuurikuvaus

Liite 4. Neuroverkon kouluttimen V0.2 arkkitehtuurin muutokset

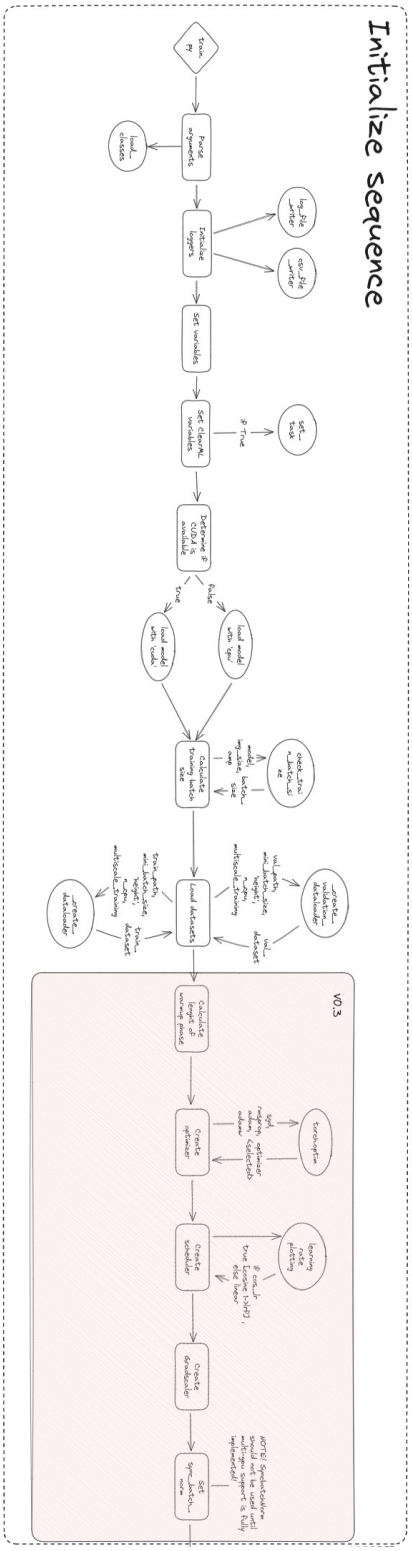
Kuvassa 1 on kuvattuna version 0.2 kohteentunnistusmallin kouluttimen koulutussilmukan arkkitehtuuriin tehdyt keskeiset muutokset verrattuna edelliseen kehitysversioon.



Kuva 1 – Version 0.2 muutokset koulutussilmukan arkkitehtuuriin.

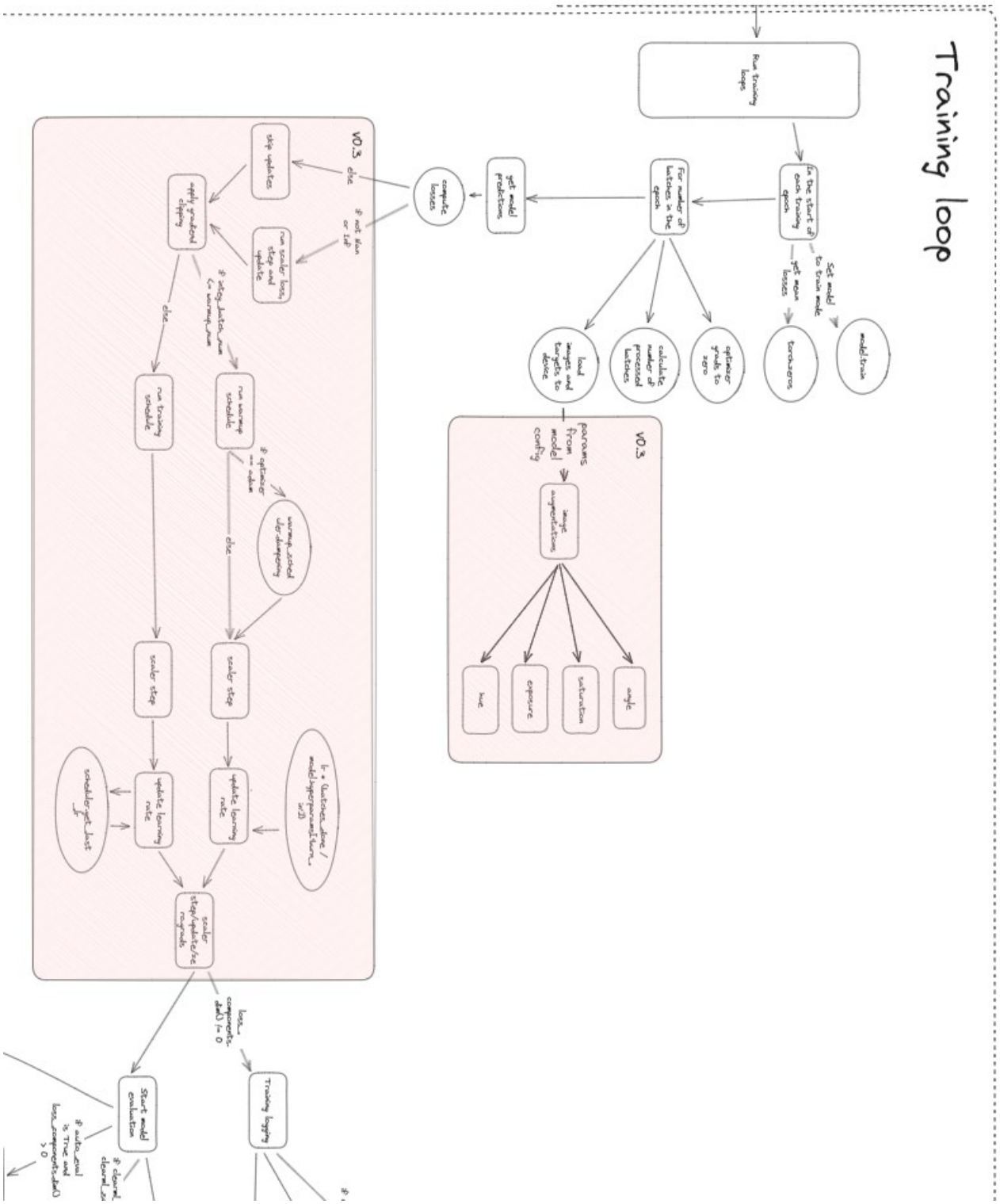
Liite 5. Neuroverkon kouluttimen V0.3 arkkitehtuurin muutokset

Kuvassa 1 on kuvattuna version 0.3 kohteentunnistusmallin kouluttimen käynnistysosion arkkitehtuuriin tehdyt keskeiset muutokset verrattuna edelliseen kehitysversioon.

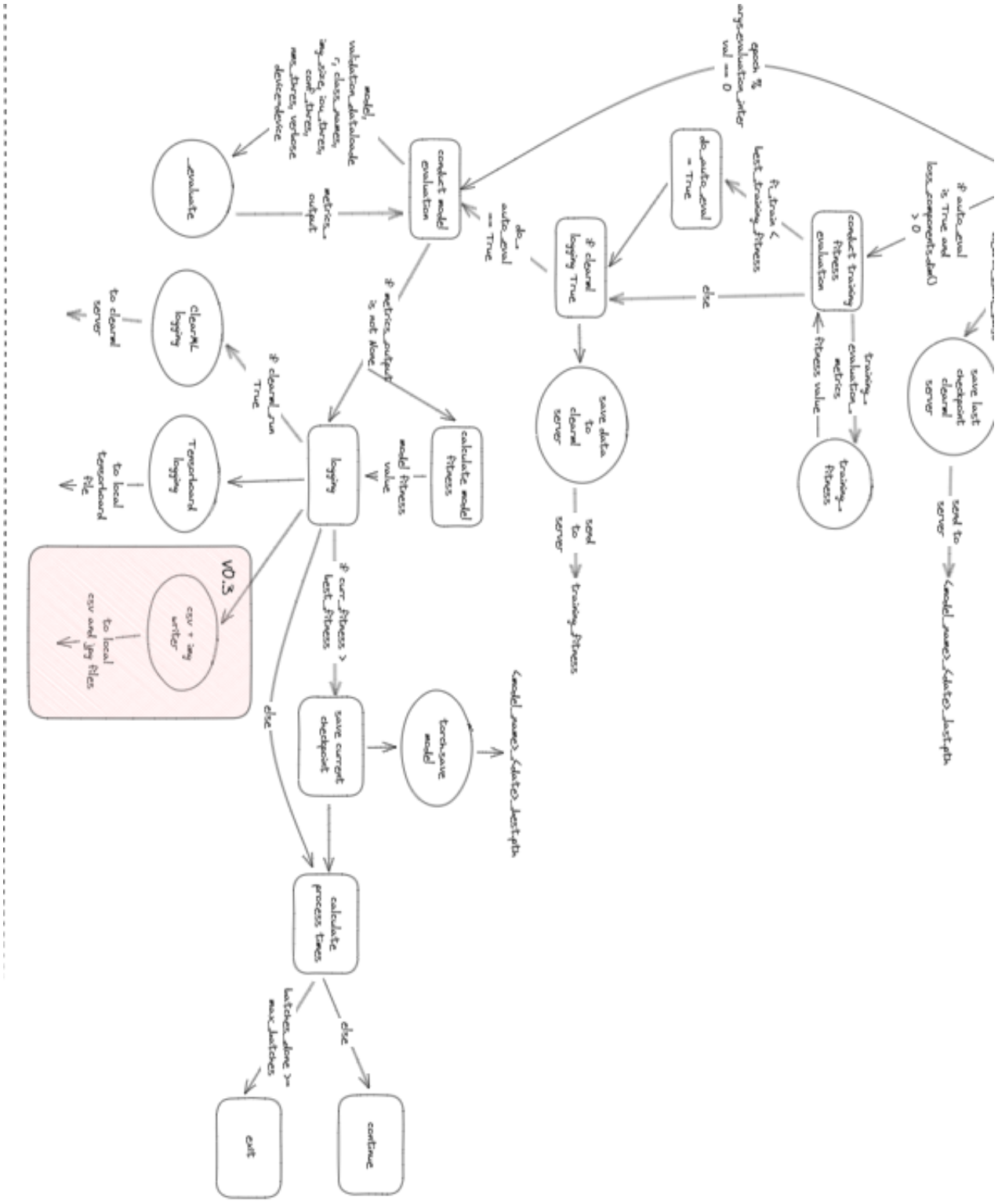


Kuva 1 – Version 0.3 muutokset arkkitehtuuriin – käynnistysosion muutokset

Kuvissa 2 ja 3 on kuvattuna version 0.3 kohteentunnistusmallin kouluttimen koulutussilmukan arkkitehtuuriin tehdyt keskeiset muutokset verrattuna edelliseen kehitysversioon.



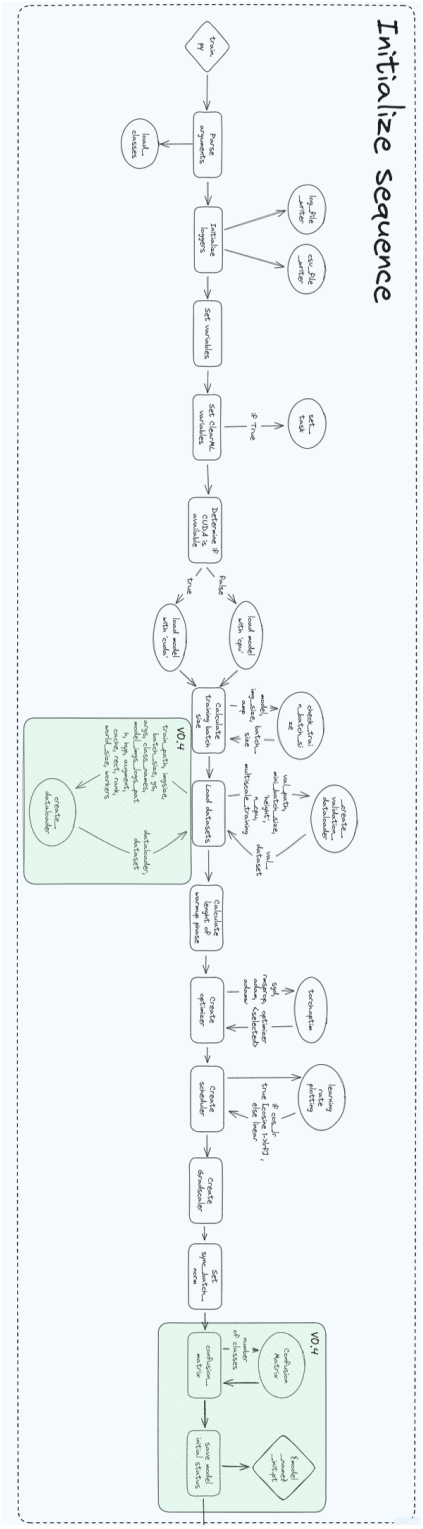
Kuva 2 – Version 0.3 muutokset arkkitehtuuriin – koulutussilmukan muutokset 1/2



Kuva 3 – Version 0.3 muutokset arkkitehtuuriin – koulutussilmukan muutokset 2/2

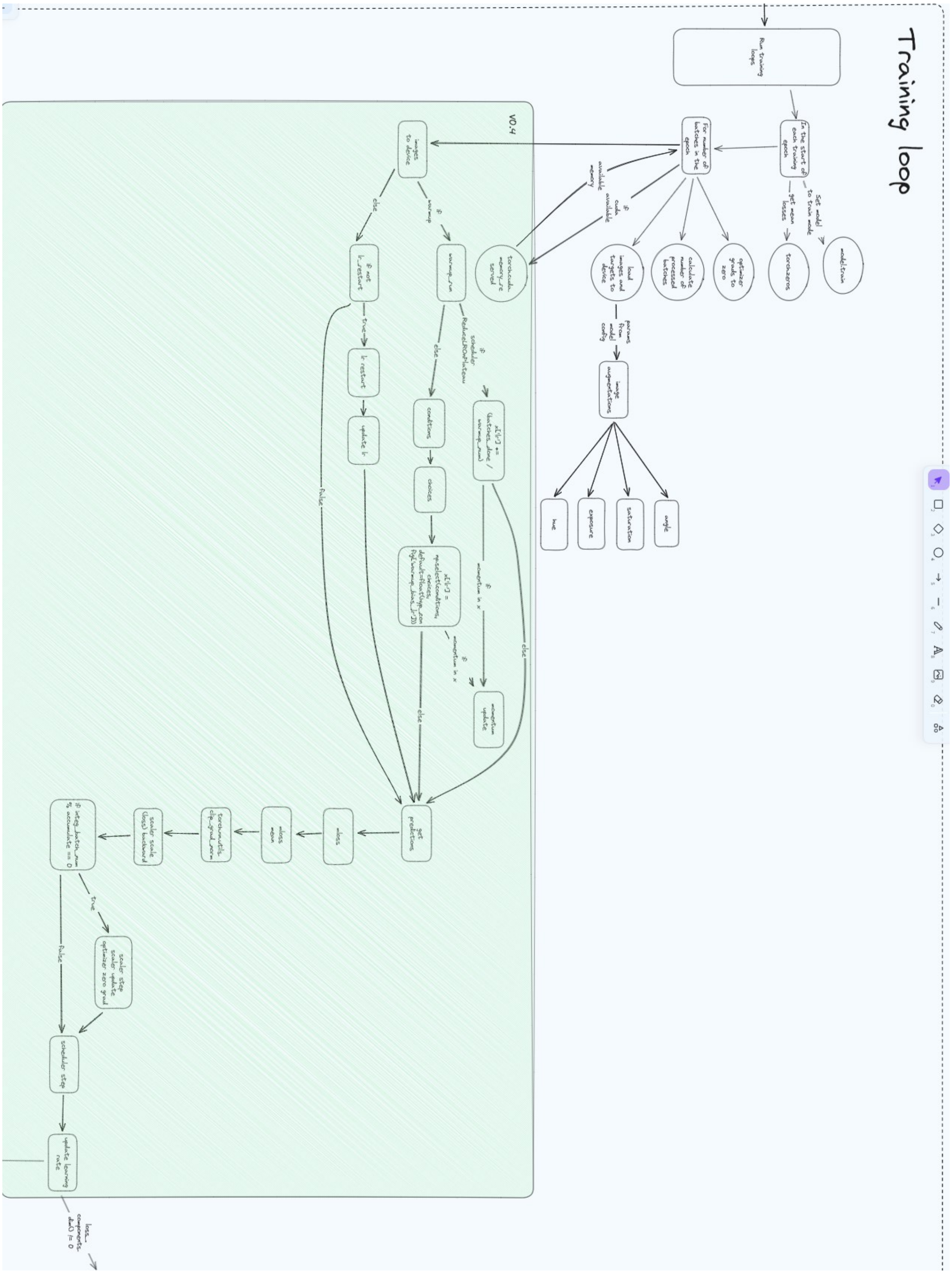
Liite 6. Neuroverkon kouluttimen V0.4 arkkitehtuurin muutokset

Kuvassa 1 on kuvattuna version 0.4 kohteentunnistusmallin kouluttimen käynnistysrutiinin arkkitehtuuriin tehdyt keskeiset muutokset verrattuna edelliseen kehitysversioon.

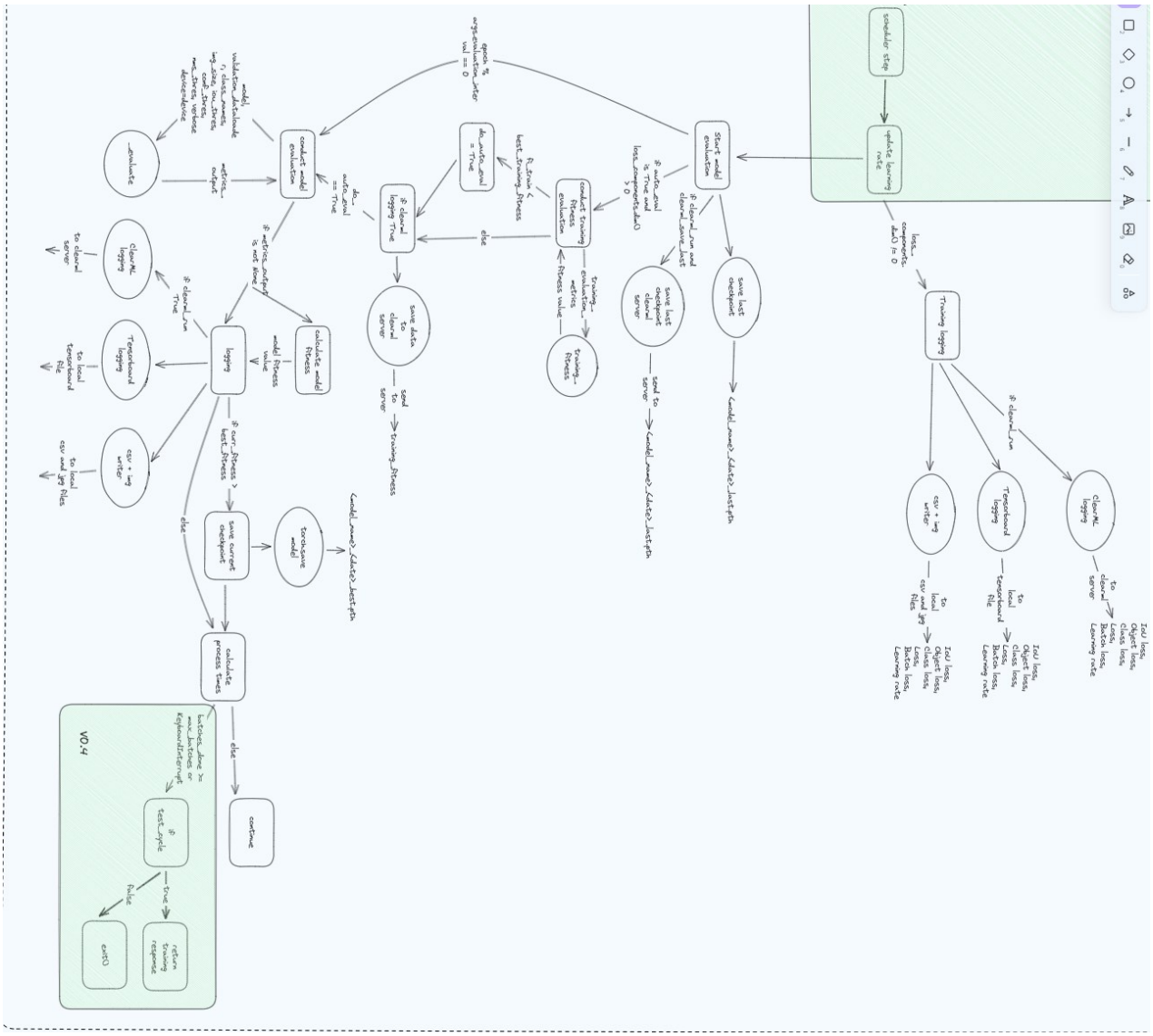


Kuva 1 – Version 0.4 muutokset arkkitehtuuriin – käynnistysosion muutokset

Kuvissa 2 ja 3 on kuvattuna version 0.4 kohteentunnistusmallin kouluttimen käynnistysrutiinin arkkitehtuuriin tehdyt keskeiset muutokset verrattuna edelliseen kehitysversioon.



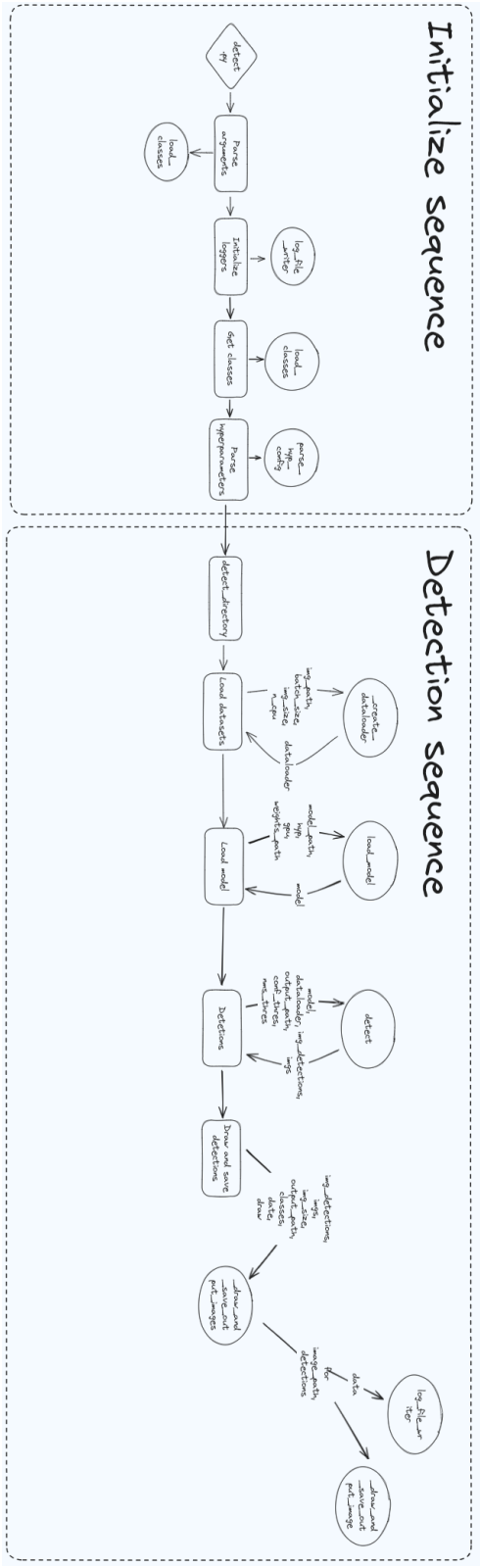
Kuva 2 – Version 0.4 muutokset arkkitehtuuriin – koulutussilmukan muutokset 1/2



Kuva 3 – Version 0.4 muutokset arkkitehtuuriin – koulutussilmukan muutokset 2/2

Liite 7. Tunnistimen sisäinen arkkitehtuuri – kuvien tunnistaminen

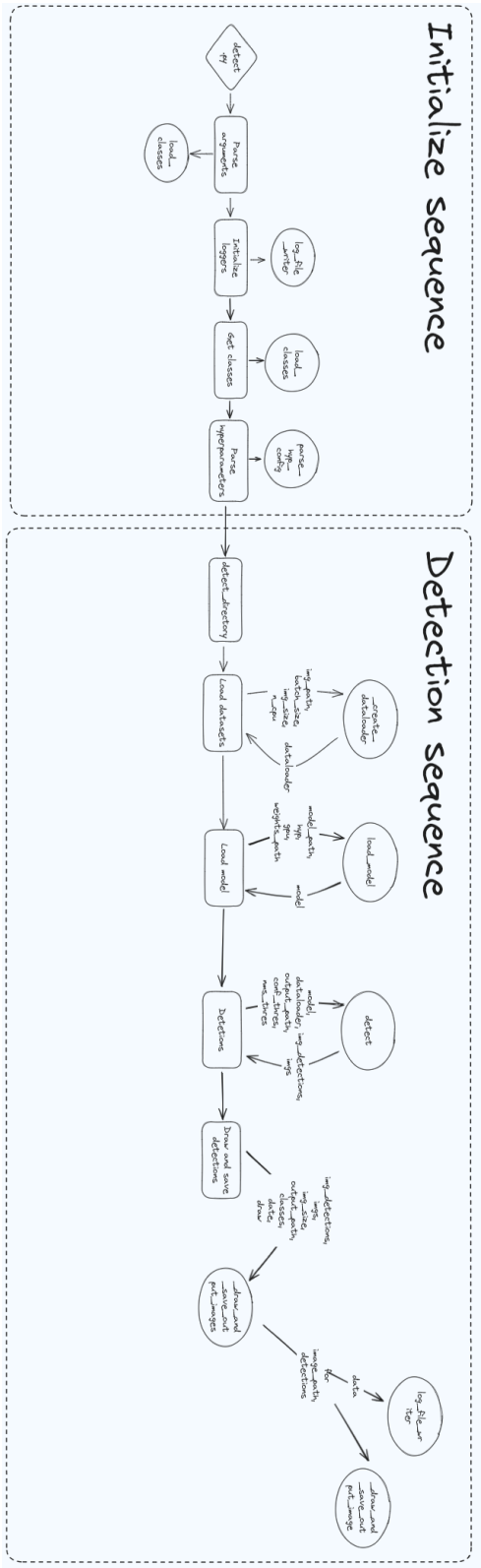
Kuvassa 1 on kuvattuna kohteentunnistimen arkkitehtuuri silloin kun kohteita tunnistetaan kuvista.



Kuva 1 – Tunnistimen sisäinen arkkitehtuuri – kuvien tunnistaminen

Liite 8. Tunnistimen sisäinen arkkitehtuuri – videolta tunnistaminen

Kuvassa 1 on kuvattuna kohteentunnistimen arkkitehtuuri silloin kun kohteita tunnistetaan videolta.



Kuva 1 – Tunnistimen sisäinen arkkitehtuuri – videolta tunnistaminen