Joonas Tuominen

Konenäön hyödyntäminen pelien kenttärakennuksessa

Tradenomi

Tietojenkäsittely

Syksy 2020

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Tuominen Joonas

Työn nimi: Konenäön hyödyntäminen pelien kenttärakentamisessa

Tutkintonimike: Tradenomi (AMK), tietojenkäsittely

Asiasanat: Koneoppiminen, luokittelu, konenäkö, tunnistus, kuvio, kategorisointi, algoritmi, U-Net

Työn tavoitteena oli kuvata konenäön teoriaa ja miten sitä voidaan hyödyntää pelinkehityksessä. Ajatus näiden konseptien yhdistämiseen tuli tämän alueen vähäisen tutkimuksen vuoksi. Automaatio lisääntyy ja pelimoottorit tarjoavat automatisoituja ratkaisuja pelinkehitykseen. Koneoppimista kuitenkaan ei ole vielä yleisesti käytetty pelinkehityksessä, vaikka siihen on potentiaalia.

Koneopissa on neljä pääasiallista oppimisen konseptia: valvottu, valvomaton, vahvistava ja keskiohjattu. Työssä keskitytään valvottuun oppimiseen, joka sisältää konenäön malleja. Konenäön tarkoituksena on jäljitellä ihmisen näkökykyä ja ymmärrystä näkyvistä esineistä ja teksteistä. Alussa kone voi vain nähdä kuvan ykkösinä ja nollina, jotka on painettu kokoon kuvaksi. Työssä käytiin läpi kuvien leimaaminen, kategorisointi ja datasetin muodostaminen tekoälylle.

Työssä käytettiin paikan ja muodon saamiseksi koneopin mallia, joka tulkattiin pelimoottorille kolmiulotteiseksi muodoksi. Tarkoituksena oli luoda ”whitebox”-kenttä kaksiulotteisesta kuvasta, jossa on tiet ja rakennusten julkisivut. Tekoäly tuotti kaksiulotteisesta kuvasta maskillisen kuvan, jossa värilliset alueet vastaavat esineitä ja asioita. Maskillinen kuva tulostettiin pilkulla erotettuun listaan, joka annettiin tulkille. Tulkki käänsi pikselidatan kategorisoiduksi paikkadataksi, jossa yksi pikseli vastaa yhtä kuvion pistettä, joiden avulla muodostetaan kolmioita. Perspektiivin ongelmaa lähestyttiin kategorisoimalla dataa, jotta voidaan erotella, mitkä esineet rakennetaan ylöspäin. Esineet, jotka rakennetaan ylöspäin, rakennetaan tien rajapisteiden avulla. Näin kuviot ovat yhdessä ja saadaan kuvioihin syvyyttä. Kuviot ovat suhteellisen halpaa piirtää, koska niitä on vähän.

Konenäköä hyödynnetään pelinkehityksessä vähän, mutta tämä työ osoitti mahdolliset käyttömahdollisuudet. Työ on vasta prototyyppi, mutta näyttää potentiaalia. Peliyritykset tulevat varmasti panostamaan enemmän automatisaatioon tietyissä asioissa ja kehitystiimeihin halutaan koneopin asiantuntijoita.

KAMK • University of Applied Sciences

Abstract

Author(s): Tuominen Joonas

Title of the Publication: Usage of machine vision in game level building

Degree Title: Bachelor of Information Technology

Keywords: Machine learning, classification, machine vision, recognition, pattern, categorization, algorithm, U-Net

The aim of this work is to go through the theory of machine vision and how it can be applied in game development. The idea of combining these two concepts came when I couldn't find anything relating to appliances for game development. Automation is increasing in game development and game engines present automated solutions for game development. Machine learning in general isn't in common usage even though there is potential.

Machine learning contains 4 main learning concepts: Supervised, unsupervised, reinforcing and semi-supervised learning. In this work we will focus on supervised learning which contains machine vision models. Machine vision aims to mimic the human eyesight and comprehension of objects and writing. At the beginning the machine can only see the picture as zeroes and ones which have been compressed to an image. In this work we have gone through image classification, categorization and forming a dataset for artificial intelligence.

In this work the idea was to use machine learning model to get placement and shape of an object which is then translated to a three-dimensional model. The goal was to create a Whitebox version of the level from two-dimensional picture where there are road and front of buildings. Artificial intelligence created an RGB picture, where coloured zones correspond to object. Mask picture is printed to a comma separated list which is supplied to the translator. Translator then converts the data to categorized location data where one pixel is one vertex. These vertices are then connected by triangles which form shapes like roads. Data categorization helps us to determine which objects should face upwards and which should be flat. This helped us with the perspective problem in some extend. The models are cheap to render because there are few of them.

The usage of machine learning is seldom but this work should show the possible affordance. This work was just a prototype but show potential. Game companies will surely put more effort in automatization in certain fields and there will be demand of machine learning experts being part of development teams.

Sisällys

Symboliluettelo

CSV: Comma separated list. Tiedostomuoto, joka sisältää pilkulla eroteltuja arvoja listana.

Conda: Avoimen lähdekoodin paketinhallintajärjestelmä.

KH: Konvoluutioinen hermoverkko. Erityinen arkkitehtuuri keinotekoisesta neuroverkosta, joka on suunniteltu erityisesti kuvien käsittelyyn ja kuvien jakamiseen topologisesti kompakteihin osioihin, joista jokaista kuvaa käsittelee suodattaja.

Konenäkö: Kuvioiden tunnistamisen alalaji, joka erikoistuu kuvissa olevien esineiden ja asioiden tunnistamiseen.

Kuvio: Asia, jolle voidaan asettaa luokka tai kategoria, kuten kuva tai signaalin aaltomuoto.

Linux: Käyttöjärjestelmä, joka on enimmäkseen avoimena lähdekoodina.

Luokittelu: Ongelma, jossa ulostulevalle tiedolle voidaan määrittää kategoria, kuten ”Katu” tai ”Julkisivu”.

Matplotlib: Python-kirjasto, jolla voidaan luoda karttoja NumPyn tuottamille taulukoille.

MRI: Magnetic resonance imaging. Radiologian alaan kuuluva lääketieteellinen kuvantamismenetelmä, jota käytetään esimerkiksi röntgenkuvauksessa.

NER: Named entity recognition. Metodi, jonka avulla voidaan määrittää kuvista olioita, jotka opetetaan tekoälylle. Esimerkiksi kuvien värittäminen.

NumPy: Python-kirjasto, joka lisää tuen suurille, useamman ulottuvuuden taulukoille: Tuo myös korkean tason matemaattisia toimintoja.

OLY: Oikaistu lineaarinen yksikkö. 2011 kehitetty luokittelufunktio.

OpenCV: Avoimen konenäön ja koneopin sovellutuksien Python kirjasto.

Python: C-pohjainen ohjelmointikieli, joka sopii hyvin matemaattisesti pohjautuviin ohjelmistoihin ja kirjastoihin

PyTorch: Pythonilla kirjoitettu avoimen lähdekoodin koneoppimisen kirjasto.

RCNN: Regional Convolution neural network. Aluepohjaisesti toimiva konvoluutioinen neuroverkko, joka pyrkii paikantamaan esineitä ja asioita kuvista suorakulmioisiin alueisiin

Regressio: Ongelma, jossa ulostulevalla tiedolla on oikea arvo (numerollinen arvo), kuten eurot tai paino.

RGB: Värimalli, jossa sekoitetaan punaisen, vihreän ja sinisen väristä valoa.

TensorFlow: Alusta loppuun luotu avoimen lähdekoodin alusta koneoppimiselle.

Unity: Yksi suosituimmista pelimoottoreista, jolla voi luoda kaksiulotteisia ja kolmiulotteisia pelejä.

U-Net: U-mallinen konvoluutioinen hermoverkon malli, joka hyödyntää oikaistun lineaarisen yksikön luokittelufunktiota ja maksimaalista arvojen yhdistämistä.

VNL: Viereisten naapurien luokittelu. Luokittelutapa, jossa verrataan vieressä olevia arvojen samanarvoisuutta.

Whitebox: Yksinkertainen malli alueesta, joka sisältää vain kentän geometriaa ilman tekstuureja tai yksityiskohtia.

1 Johdanto

Koneoppiminen kuuluu tekoälyjen alaluokkaan, missä perimmäisenä tavoitteena on luoda ihmisen korviketta tutkimustiedon prosessointiin. Ihmisellä kuluu noin pari tuntia käydä läpi 20 kuvaa, kun taas tavallinen kotikone voisi käydä läpi tuhansia, jos sen opettaa oikein. Automatisoinnin tärkeys tulevaisuudessa on kiistämätön, ja sen tutkintaan käytetään enemmän aikaa kuin ennen. Pelinkehityksessä tekoälyä hyödynnetään jo ohjelmoinnissa kieliserverien muodossa. Kieliserverit auttavat ohjelmoijia kirjoittamaan koodia automaattisella täydennyksellä ja viittausluetteloilla.

Koneoppimisen ala on kasvamassa, ja sen käyttötarkoituksia ei ole vielä mitoitettu. Pelialan sovellutuksia on vielä harvassa, koska harvempi pelialaan erikoistunut miettii datacenter ratkaisuja videopelien kehityksessä. Tämä johtunee siitä, että koneoppimista hyödynnetään yleisesti tiedon prosessoinnissa ja statistisissa malleissa. Myös U-Net-malli, jota työssä käytetään, on suunniteltu lääketieteeseen löytämään syöpäkasvaimia. Tulevaisuus näyttää kuitenkin hyvältä peliteollisuudessa, koska koneoppimisesta on tullut muodikas teknologian ala. On odotettavissa, että suurempi osa kehittäjistä haluaa laittaa työpanoksensa koneoppimisen parantamiseksi ja laajentamiseksi. On hyvin mahdollista, että tulevaisuudessa tulee olemaan enemmän koneoppimisen ratkaisuja hyödynnettynä videopelien kehityksessä.

Työn tarkoitus on antaa esimerkkitapaus, miten koneoppimista voisi hyödyntää ja osoittaa sen hyödyllisyys tulevaisuudessa. Prototyypin ideana on luoda yksinkertainen kenttä kaksiulotteisesta kuvasta, jossa ei ole ylimääräisiä esineitä kuten puita tai autoja.

2 Konenäkö

Koneoppiminen on algoritmien ja tilastollisten mallien tutkimista, joita tietotekniset järjestelmät käyttävät tiettyjen tehtävien suorittamiseen ilman erityisiä ohjeita. Koneoppiminen on kasvava laskennallisten algoritmien ala, joka tähtää ihmisälyn jäljittelyyn (1). Koneoppimisen konsepti syntyi, kun Arthur Samuel loi vuonna 1952 tietokoneohjelman, joka osasi pelata tammea (2, s. 206-226). Nopeasti lisääntyvä informaatio ja sen digitalisaatio tarvitsee keinotekoisia älyllisiä järjestelmiä käsittelemään suurta datan määrää, jotta voitaisiin tehdä johtopäätöksiä ja tutkielmia nopeammin ja luomaan ratkaisuja aihekohtaisiin ongelmiin. Koneoppimisen yleistyminen alkoi lääketieteestä, koska haluttiin ennustaa potilaiden terveydentilaa tulevaisuudessa. Osa koneopin sovellutuksista keskittyy tilastoista ja kuvista ennustamiseen. MRI (magnetic resonance imaging) on radiologian alaan kuuluva lääketieteellinen kuvantamismenetelmä, jota käytetään esimerkiksi röntgenkuvauksessa. MRI:n yhteydessä voidaan hyödyntää koneoppia ennustamaan tai tunnistamaan syöpää potilaiden röntgenkuvista.

Konenäkö on yksi koneopin osa-alueista, jonka avulla tulkitaan kuvia ihmisen tavoin. Tekoäly ei vielä voi tulkita kuvia täysin ihmisen tasolla, mutta rajaamalla tietoa voi tekoälyllekin opettaa kuvien tulkintaa. Tässä työssä käytetään katukuvia, joissa on paljonkin ylimääräistä tietoa, kuten katuroskat ja taivas. Jotta voidaan optimoida tekoälyn opettamista ja tehdä siitä vähemmän raskasta, pitää kuvista luokitella näkyvät asiat. Opinnäytetyössä keskitytään rajaamaan talojen julkisivut, ihmiset ja tiet. Aikataulun vuoksi käytetään valmista datasettiä, joka on luokiteltu lähelle tarpeisiin vastaavaksi (3).

2.1 Valvottu oppiminen

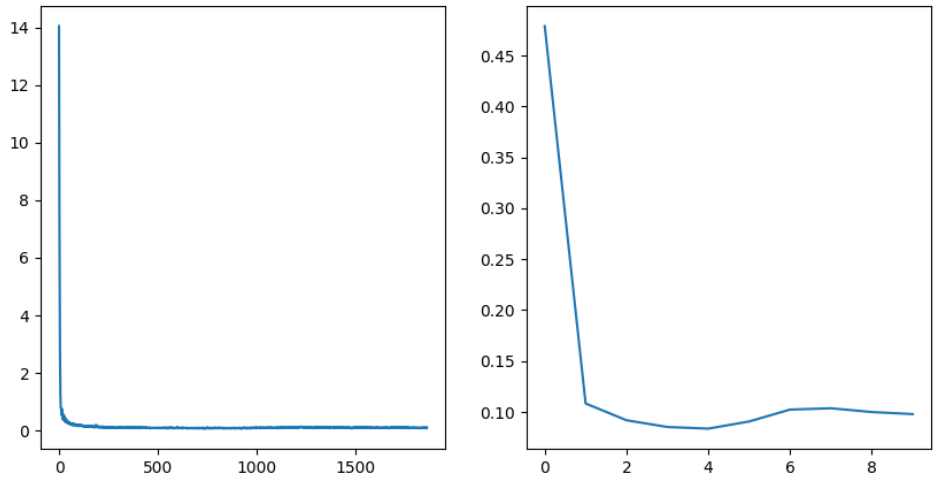
Koneoppimiseen liittyy neljä oppimisen mallia, jotka ovat valvottu, valvomaton, keskiohjattu ja vahvistava oppiminen. Tässä työssä keskitytään enimmäkseen valvottuun oppimiseen, jota käytetään kuvioiden ja kuvien tunnistukseen. Valvottuun oppimiseen liittyy kuvien luokittelu, oppimisen seuraaminen ja ennustaminen.

Valvotussa oppimisessa on syöte- ja ulostulomuuttujia, joihin voidaan käyttää algoritmeja, joiden avulla voidaan oppia kartoittava toiminto syötteestä ulostuloon. Tavoitteena on arvioida kartoit-

tava toiminto niin hyvin, että sitä voidaan hyödyntää ulostulon ennustuksessa, kun esitetään toinen syötemuuttuja. Valvottu oppiminen saa nimensä sen vuoksi, koska algoritmillinen oppiminen vaatii tietosarjan, jonka voi opettaa niin kuin opettaja valvoisi oppimisprosessia. Oppiminen päättyy, kun algoritmi saavuttaa hyväksyttävän tason (4.) Valvottuun oppimiseen liittyy kaksi ongelmaa: luokittelu ja regressio. Luokittelu on ongelma, jossa ulostulevalle tiedolle voidaan määrittää kategoria, kuten ”Katu” tai ”Julkisivu”. Regressio on ongelma, jossa ulostulevalla tiedolla on oikea arvo (numerollinen arvo), kuten eurot tai paino. Näiden avulla voidaan määritellä lähestymistapa tiedon käsittelyyn. Kuvioiden ja kuvien tunnistamisessa käytetään luokittelua. Opinnäytetyössä keskitytään enemmän luokitteluun, koska kuvien tieto ei ole reaalilukuja.

Luokittelun tapoja on erilaisia, kuten viereisten naapurien luokittelu (VNL) tai lineaarisen erottelijan analyysi (5). Luokittelu ja hermoverkot tähtäävät samaan lopputulokseen, mutta toimivat eri tavoin. Luokittelua voidaan hyödyntää hermoverkoissa oppimiseen, mutta hermoverkkoja ei voida hyödyntää luokittelufunktioissa. VNL perustuu nimensä mukaisesti viereisten naapurien arvojen vertailuun. Jos viereiset arvot ovat tarpeeksi lähellä toisiaan, laitetaan ne samaan luokkaan. Esimerkiksi sokerin ja myricetinin määrien vertailulla voidaan määritellä, onko viini punaista vai valkoista. Kuten valvotulle oppimiselle on ominaista, ennakkotapausten lukujen perusteella tehdään ennustaminen. Tässä projektissa olisi pystytty antamaan eri asioille luokka, joilla on sisäisiä arvoja, jotka erottavat ne toisistaan. Kuitenkin tämä lähestymistapa olisi ollut liian työläs, koska VNL ei opi edellisistä tapauksista, vaan luokittelee metodissa määritellyllä tavalla. Hermoverkoissa oppiminen on jatkuvaa, ja hermoverkko pystyy oppimisen jälkeen ennustamaan tapauksia, joissa on samanlaisuuksia opeteltuihin tapauksiin. Konvoluutioiset hermoverkot ei ole kuitenkaan optimoitu raakojen arvojen ennustamiseen, vaan kuvien käsittelyyn. Tätä selvennetään seuraavassa kappaleessa enemmän.

Oppimisen seuraamisessa täytyy tarkastella mallin ulostuloa ja verrata alkuperäisiin kuviin. Työssä käytettävässä hermoverkon mallissa voidaan tarkastella kuvan maskia. Maskin tarkkuus riippuu oppimisesta, joten luokitellun ja ennustetun kuvan vertailu keskenään antaa kehityksestä tarkan arvion. Mallista voidaan myös laskea oppimisen hävikki ja tarkkuus, jotka näytetään kaavioina kuvassa 1. Vasemmanpuolinen kaavio näyttää hävikin per kuva prosentteina (y-akseli) ja kuvien määrän (x-akseli). Oikeanpuolinen kaavio näyttää oppimisen häviön iteraatiossa prosentteina (y-akseli) ja senhetkisen iteraation (x-akseli).



Kuva 1. Mallin oppimisen määrä ja hävikki.

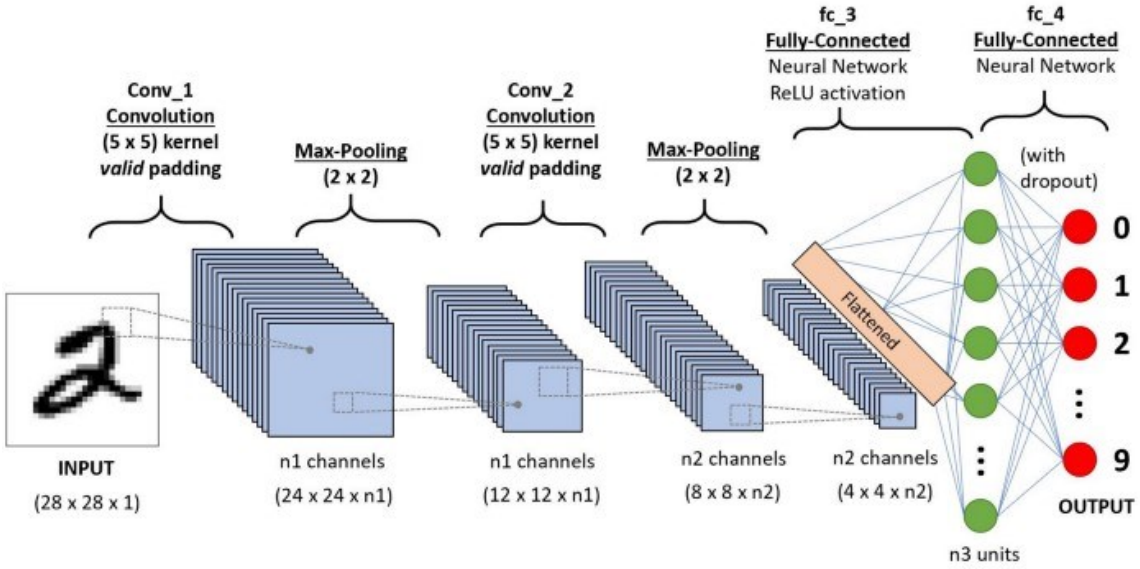
Kaavioista voidaan huomata, että ensimmäisessä iteraatiossa esiintyy suurin häviö, koska tekoäly aloittaa oppimisen. Tekoälyllä ei ole alustavaa tietoa ennen ensimmäistä iteraatiota, joten hävikki on normaalia. Tekoäly tarvitsee alustavaa tietoa oppiakseen tarkasti. Tämä heijastuu kaaviossa toisessa iteraatiossa, joissa hävikin määrä pienenee kymmenykseen. Myös kuvakohtaisessa hävikin laskussa hävikin käyrä putoaa rajusti ja tasoittuu yhden paikkeille. Myös iteraation hävikki tasoittuu kymmeneen prosenttiin, joskin pienellä vaihtelulla.

Oppimisen jälkeen mallia pyritään käyttämään ennustamiseen. Ennustamisessa pyritään ennakkotapausten eli opitun materiaalin pohjalta tekemään johtopäätöksiä. Esimerkiksi rajaamaan kuvista alueita, jossa on ihmisiä. Ennustamiseen käytettävä aika on murto-osa opettamiseen tarvittavassa ajasta, koska tekoälyllä on jo tiedot valmiina. Tekoäly vain implementoi ennakkotiedot käytäntöön. Myös ennustettavan materiaalin määrä on pienempi kuin opeteltavan materiaalin.

2.2 Konvoluutioinen hermoverkko

Konvoluutioinen hermoverkko (KH) on erityinen arkkitehtuuri keinotekoisesta neuroverkosta, joka on suunniteltu erityisesti kuvien käsittelyyn ja kuvien jakamiseen topologisesti kompakteihin osioihin, joista jokaista kuvaa käsittelee suodattaja. Kuvien jakamista tarvitaan kuvien kokojen vuoksi. Esimerkiksi 100x100 kuva sisältää 10000 pikseliä, ja jos ensimmäinen taso sisältää 1000

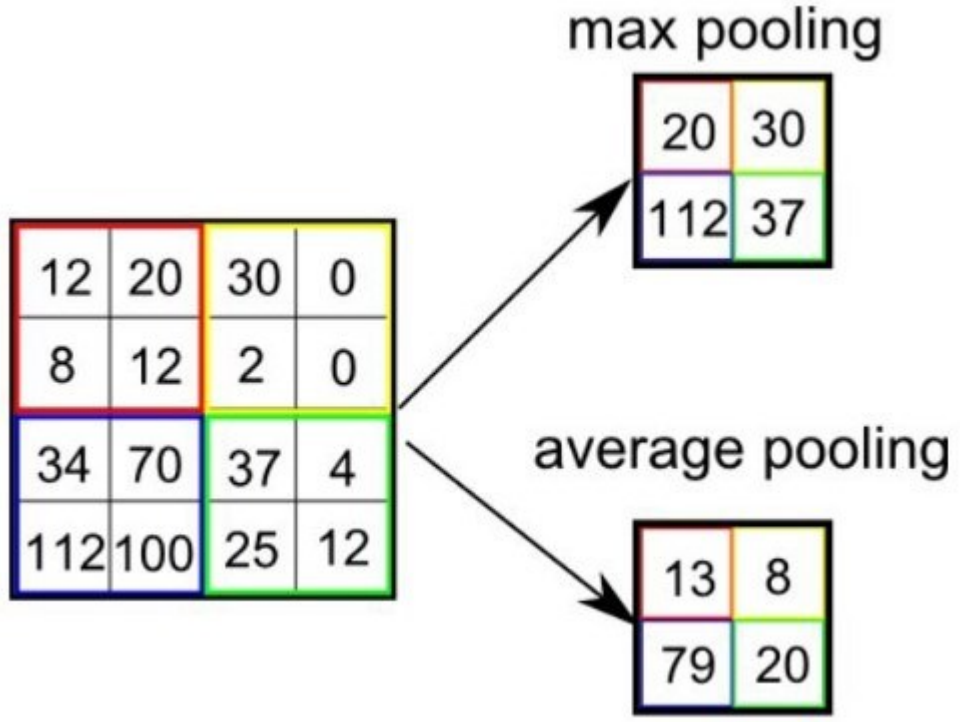
neuronia, tämä tarkoittaa 10 miljoonaa yhteyttä hermoverkossa. Tämä on todella raskasta ja työlästä käsitellä. KH myös käyttää todella paljon uudelleen painojansa, joten KH sisältää paljon vähemmän parametrejä kuin keinotekoinen neuroverkko (6). Painot ovat parametrejä, joilla voidaan määritellä sisääntulomuuttujan vaikutus ulostuloon (eli yhteyden voimakkuus kahden neuroverkon noodin välillä). Esimerkiksi kuvasta saadun hallitsevan piirteen vaikutus seuraavaan oppimiskertaan. KH koostuu kahdesta yksinkertaisesta elementistä: konvoluutiotasoista ja kerrosten yhdistämisistä. Kuva 2 antaa kuvan KH toiminnasta. Sen sijaan, että verrattaisiin naapureita ja niiden arvoja keskenään, pilkotaan kuvat pienempiin kerroksiin. Pienemmät kerrokset ovat nopeampia käsitellä ja niistä saatava tieto on tarkempaa vaikkakin resoluutio on pienempi. Myös KH oppii kuvista ja ennustaa tulevista alueet, joissa kohteet mahdollisesti sijaitsevat.



Kuva 2. Konvoluutioisen hermoverkon kaavio (7).

Sisääntulomuuttuja (kuva) syötetään ensimmäiselle noodille, joka on kanava 1. Kanava 1 ottaa osan kuvasta, jonka resoluutio on tietenkin pienempi kuin alkuperäisen kuvan. Kuvasta otetaan hallitsevat piirteet ja syötetään ne seuraavaan kanavaan. Tätä prosessia kutsutaan kerrosten yhdistämiseksi. Prosessia jatketaan, kunnes saavutetaan haluttu resoluutio. Haluttu resoluutio riippuu kerrosten määrästä. Tässä työssä käytetään neljää kerrosta, mikä on yleinen määrä U-Net-mallissa.

Kerrosten yhdistäminen ottaa kuvan hallitsevat piirteet ja yhdistää piirteiden arvot yhteen (7). Tämä prosessi tehdään siksi, että voidaan vähentää laskentaan tarvittavaa tehoa. Kuvan 2 viittauksessa havainnollistaa prosessia tarkemmin. Tässä projektissa käytetään maksimaalisten arvojen (max pooling) yhdistämistä.



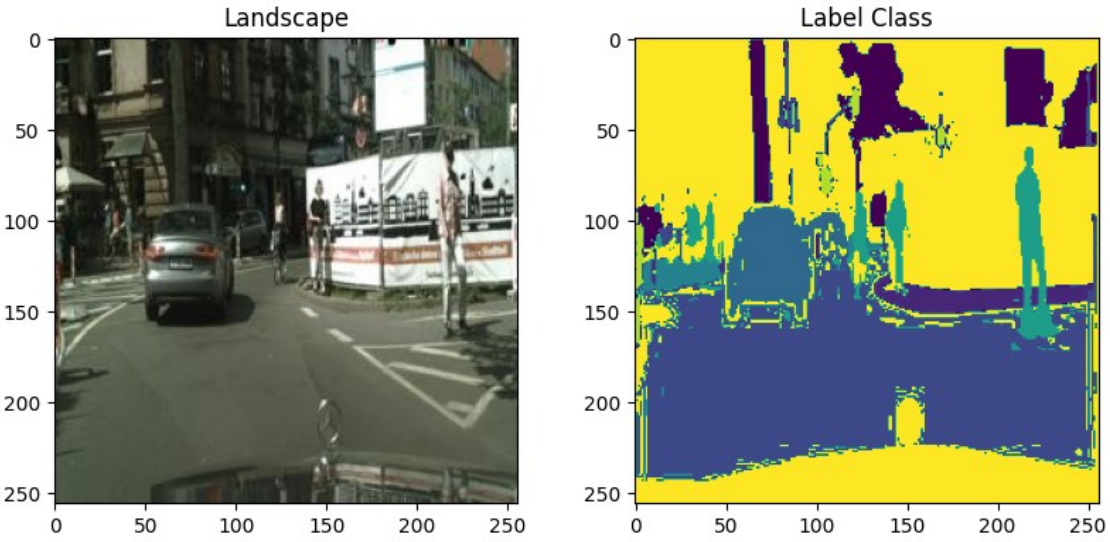
Kuva 3. Kerrosten yhdistäminen(7).

Kerrosten yhdistäminen on suodattaja, joka pienentää tilan kokoa konvoluutioidussa ominaisuudessa (7). Kuvassa 3 otetaan 2x2 osioita 24x24 kuvasta, jotka voidaan käsitellä kahdella tavalla. Keskiarvoisessa yhdistämisessä otetaan 2x2 osioiden arvojen keskiarvo. Maksimaalisten arvojen yhdistämisessä otetaan korkeimman luvun (eli hallitsevan ominaisuuden) ja annetaan sen seuraavalle noodille. Maksimaalista arvoa hyödynnetään U-Net-mallissa, kun jaotellaan kuvia. U-Net-mallin arvojen käsittely muistuttaakin hyvin paljon KH-mallia pienellä muokkauksella.

2.3 U-Net-malli

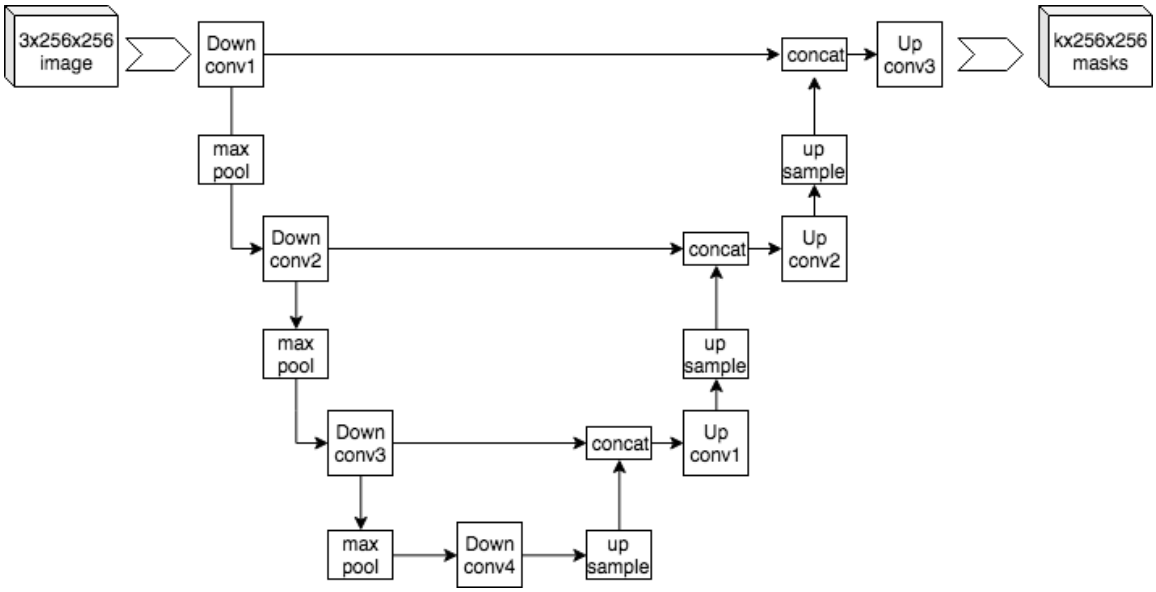
U-Net malli luotiin vuonna 2015 kirjoituksessa ”U-Net: Convolutional Networks for Biomedial Image Segmentation” (8). U-Net-mallin etuna verrattuna täysin konvoluuttisiin verkkoihin on nopeampi kuvien jaottelu ja luokittelu. U-Net hyödyntää myös oikaistua lineaarista yksikköä (OLY), joka on vuonna 2011 kehitetty luokittelufunktio. Yleisesti käytetympi ja vanhempi funktio on ”Softmax”, joka käyttää viimeistä neuroverkon tasoa ja painoparametriä laskeakseen ennustettua luokkaa. U-Net-mallissa puolestaan käytetään OLY:ä paitsi viimeisessä neuroverkon tasossa, mutta myös jokaisella piilossa olevalla verkon tasolla. Tämä näkyy U-Net-mallin kaaviossa (kuva 5) jokaisen ”max pool”-noodin kohdalla, joka hyödyntää OLY:ä. Molemmat algoritmit suoriutuvat aika samanlaisesti (OLY:n ollessa hieman nopeampi, mutta vähän epätarkempi)(9.) U-Net-mallin toteutus kirjoitetaan Pythonilla, joka on C-pohjainen ohjelmointikieli, joka sopii hyvin matemaattisesti pohjautuviin ohjelmistoihin ja kirjastoihin. Molemmissa funktioissa on etuutena ennalta määritelty painoparametri, joka laskee vaadittavan laskentatehon määrää.

Luokittelun keinona käytetään RGB maalausta, jossa opeteltavassa kuvassa on vain väritettyjä alueita. Alueiden värit vastaavat asioiden paikkoja ja muotoja, kuten kuvassa 4 on havainnollistettu. Tällä tavalla koneäly tietää, mitä sen pitää opetella ja osaa tulevaisuudessa rajata tavallisista kuvista alueita, joissa asiat ovat. Matplotlib on Python-kirjasto, jolla voidaan luoda karttoja NumPyn tuottamille luvuille. NumPy on Python-kirjasto, joka lisää tuen suurille, useamman ulottuvuuden taulukoille. Se tuo myös korkean tason matemaattisia toimintoja.



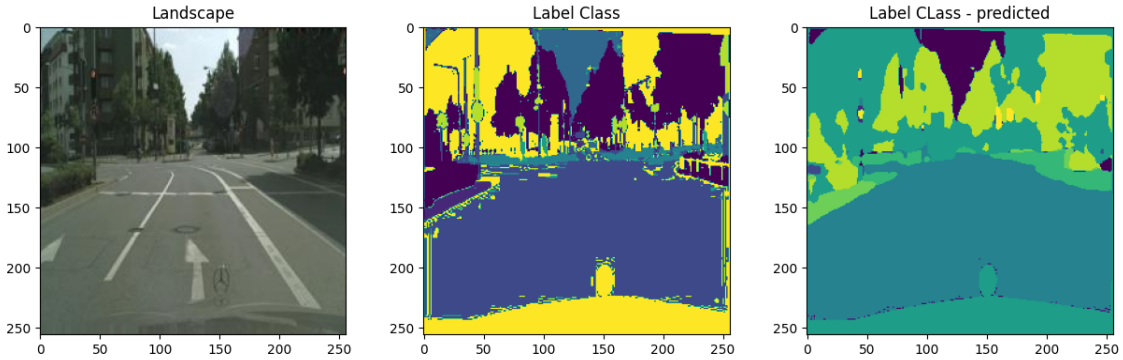
Kuva 4. Alkuperäinen kuva (vasen) ja luokiteltu kuva (oikea). Julkisivut on kuvattu keltaisella, kadut sinisellä, pylväät ja liikennevalot violetilla ja liikkuvat oliot vaaleanvihreällä.

Kuvat on luotu Matplotlibillä, jolle on annettu NumPy taulukko ohjeeksi. Näin saadaan kartan näköisiä kuvia, joihin voidaan laittaa riveittäin useampia kuvia. Kuvan oikeanpuoleinen kartta on se, mitä tekoälylle opetetaan (vasemmanpuoleinen on havainnollistamiseen.) Tämä on yksittäinen kuva siitä tietosarjasta, joka syötetään tekoälylle. Tämä kuva käy läpi konvoluutioverkon, jossa sitä käsitellään noodeittain. Tätä prosessia havainnollistaa kuva 5. Verkossa aluksi lisätään kanavien määrää, jotka pienentävät otantaa kuvasta. Eli prosessissa vähennetään jokaisen kanavan lisäyksen jälkeen pikselien määrää kuvista. Tämä prosessi vähentää ulottuvuutta kuvista ja vaadittavaa laskennallista kuormaa. Tämä toteutetaan U-mallisesti, josta malli saakin nimensä.



Kuva 5. U-Net-mallin konvoluutioverkko, jossa hyödynnetään kolmea 256x256 kuvaa.

TensorFlow on alusta loppuun luotu avoimen lähdekoodin alusta koneoppimiselle. Tämä toimii osana PyTorch-kirjastoa. Kuvasta 5 näkyy, miten edellisten otantojen syötteen TensorFlow-taulukoista otetaan isoimmat arvot ja annetaan ne seuraavalle konvoluutioverkon noodille. Koska kuvat ovat 256x256 resoluutiolla, tarvitsee niitä käsitellä vain neljä kertaa. Työssä pyritään pääsemään kuvissa 16x16 resoluutioon, jotta ne voidaan nopeasti käsitellä ja opettaa tekoälylle. Kun on päästy tavoitteeseen, on aika nostaa resoluutio takaisin alkuperäiseen. Prosessin aikana liitetään samalla korkeudella olevien noodien kuvien TensorFlow-taulukoiden arvot yhteen. Näin voidaan vahvistaa, että oikeat alueet ovat kyseessä ja mahdollisen ”virhekohinan” tai ison kontrastin aiheuttama virhe minimoiduksi. Prosessin loppupuolella saadaan aikaan 256x256 maskeja, jotka näyttävät kuvan 6 oikeanpuolisen kuvan tapaisilta. Tämä tulos tulee testiotannasta, jossa on referoinnin helpottamiseksi myös luokiteltu kuva. Luokiteltua kuvaa ei anneta tekoälylle, vaan alkuperäinen kuva.



Kuva 6. Alkuperäinen kuva (vasen), luokiteltu kuva (keskellä) ja ennustettu kuva (oikea)

Kuten kuvasta 6 näkyy, niin malli on hyvin lähellä oikeaa tilannetta. Pientä heittoa löytyy joistain kohdista, mutta pidemmän ajan jälkeen kuva olisi todennäköisesti tarkempi. Huomattavaa on myös, että tekoälyn luomat maskit ovat pehmeäreunaisia ja sisältävät vähän ylimääräistä ”virhekohinaa”. Puiden määrä kuvassa vaikeuttaa rakennusten julkisivujen hahmottamista ja tämä näkyykin tulkinnan vaiheessa, kun lopputulos viedään Unityn puolelle.

3 Tulkki ja Unity

Unity on pelimoottori, jolla voi luoda kaksiulotteisia ja kolmiulotteisia pelejä. Kun on saatu tekoäly opetettua ja saadaan mahdollisimman tarkka maski tuotettua, voidaan aloittaa tiedon viemisen prosessi Unityyn. CSV (comma separated list) on tiedostomuoto, joka sisältää pilkulla eroteltuja arvoja listana. Unity ei suoraan tue CSV tiedoston lukemista, mutta C#-kirjasto sisältää tarvittavat luku- ja jaottelumetodit toteutukselle. Tekoäly tulostaa kuvan pikselit suoraan CSV-tiedostoon, jonka voi sitten siirtää Unityn puolelle. Tiedoston koko on pieni, joten sen siirtäminen ei aiheuta hankaluuksia. Kuvat ovat muodossa RGB, joka on värimalli, jossa sekoitetaan punaisen, vihreän ja sinisen väristä valoa.

3.1 Tulkin toiminta

Koska kuvat ovat noin 256x256 kokoisia, tulee pikseleistä saatua dataa paljon. Jokaista pikseliä kohden tulee taulukko, joka sisältää RGB arvot (eli kolme arvoa). Näiden arvojen suoraan vienti pelimoottoriin ei ole ideaalista, joten on parempi, että arvot tallennetaan CSV-tiedostoon. CSV-tiedosto sisältää kaiken tiedon pilkuilla eroteltuna, joten sen lukeminen on helpompaa. Kuvan pikselidatan lukeminen aloitetaan kuvan vasemmasta yläkulmasta ja etenee rivin kerrallaan vasemmalta oikealle. Kuvien lukemiseen käytetään OpenCV:tä, joka on avoimen lähdekoodin konenäön Python-kirjasto. OpenCV sisältää koneopin ja konenäön sovelluksiin sopivia metodeja ja luokkia. Tässä työssä se toimii osana PyTorch-kirjastoa. Käytetään OpenCV:n metodeja kuvien lukemiseen ja tietojen tulostamiseen. OpenCV tulostaa väriarvot BGR muodossa, joten tulkinnan vaiheessa on hyvä muistaa tämä.

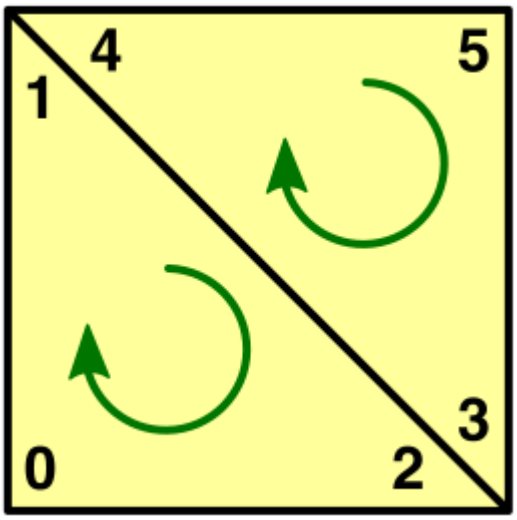
Jotta Unity osaisi käsitellä annettuja arvoja, on tehtävä tulkki tiedostojen arvoille. Tulkin toiminta on seuraavanlainen: lue pikselien väriarvot, yhdistä samanlaiset väriarvot muodostamalla kappaleita ja syötä ne kappaleiden luojalle. Pikselien arvojen lukeminen on helppoa, koska luvut ovat peräkkäin. Samanlaisten väriarvojen yhdistämiseen tarvitaan enemmän työtä. Edellisistä kuvista nähdään, miten tekoäly värittää maskit ja minkä muotoisia kappaleet ovat. Kuitenkin osa väreistä on lähellä toisiaan ja saman värisiä pikseleitä voi olla erillisinä kappaleina. Asiaa lähestytään tässä

työssä niin, että pikseli tarkastaa seuraavan pikselin väriarvot. Jos väriarvot ovat samat, niin voidaan pikseli olettaa kuuluvan samaan muotoon. Myös pystytään pikselidatasta luomaan kaksiulotteinen taulukko, josta voidaan tarkastella naapurit oikealta vasemmalle ja ylhäältä alas.

3.2 Muotojen luominen

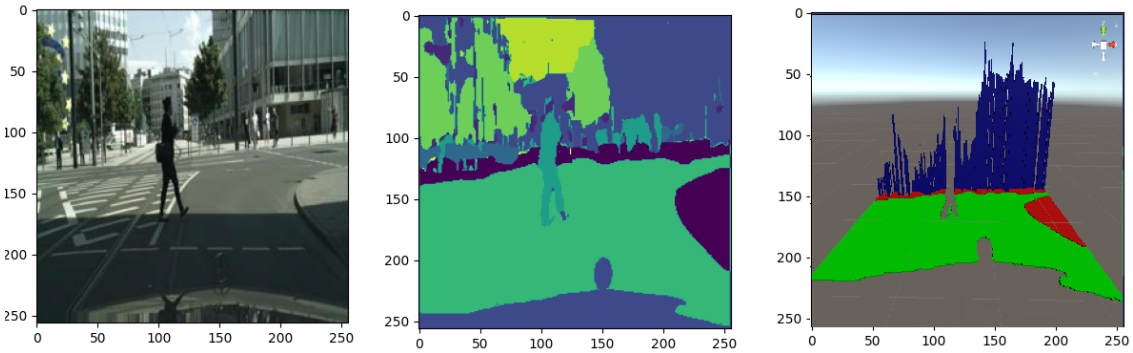
Kun pikselit on ryhmitelty, pitää niistä rakentaa muotoja. Muodot luodaan proseduraalisesti, jossa topologiana käytetään kolmioita. Jokainen pikseli, jota ei ole yhdistetty, vastaa yhtä muodon pistettä. Kuvan perspektiivi on keskeinen ongelma kaksiulotteisten kuvien muuntamisessa kolmiulotteiseksi. Tässä tapauksessa pyritään rakentamaan tie ensimmäiseksi, toiseksi jalkakäytävät, kolmanneksi rakennukset ja lopuksi muut. Tässä syynä on tien pieneneminen, joka auttaa hahmottamaan perspektiiviä. Kun tie on hahmoteltu, voidaan tien rajapisteistä jatkaa jalkakäytävät, koska ne ovat muodoiltaan hyvin samankaltaiset. Kun on saatu ”pohja” luotua kentälle, voidaan jalkakäytävien rajapisteistä jatkaa ylöspäin eli muodostaa rakennuksien julkisivut. Rakennusten julkisivut eivät ole yksityiskohtaisia, koska tekoälylle ei ole opetettu erottamaan erillisiä muotoja rakennuksista. Prototyypin merkeissä riittää ”whitebox”-versio kuvasta.

Edellisen kappaleen prosessi vaatii proseduraalisen luomisen. Yksinkertaisuudessaan kentän geometrian luominen vaatii pisteitä ja indeksejä. Pisteitä tarvitaan, koska pyritään muodostamaan geometrisiä muotoja kolmioiden avulla. Tätä havainnollistetaan kuvalla 7, joka näyttää indeksoinnin käytännössä. Indeksien luvut määrittyvät aloituspisteestä ja myötäpäivään kierrosta.



KUVA 7. Neliön etusivun muodostaminen kahdesta neliöstä indeksien kanssa (10.)

Myötäpäivään kierretään, koska halutaan mallin etupuolen näkyvän. Jos kierrettäisiin vastapäivään, niin näkyisi mallin takaosa. Vieressä olevan kolmion alkamispiste on edellisen kolmion loppupiste. Näin pelimoottori ymmärtää, että kolmiot ovat liittyneitä toisiinsa. Sama kuvio toistuu seuraavissakin neliöissä. Prototyypissä muodostetaan kuvan oliot riveittäin lukemalla pikselidatan koordinaatit, jotka on luetteloitu tulkinnan vaiheessa. Jokainen koordinaatti vastaa yhtä pistettä muodossa. Tie muodostetaan x- ja z- akselilla ja rakennusten julkisivut z- ja y-akselilla. Kuva 8 konkretisoi, miltä tämä näyttää. Rakennusten julkisivut rakennetaan teiden rajapisteiden avulla, joista sitten rakennetaan ylöspäin paikkadatan avulla. Koska kuvat luetaan ylhäältä alas, pitää paikkadata kääntää ylösalaisin, jotta voitaisiin saada jalkakäytävien ääripisteiden paikat ennen rakennuksien rakentamista.



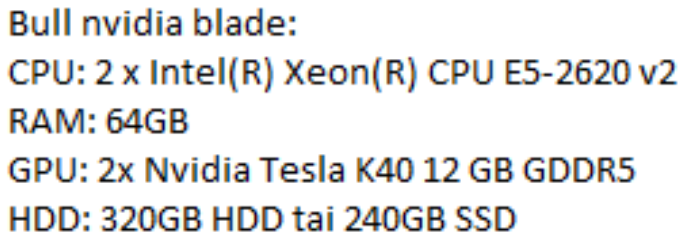
Kuva 8. Oikealta vasemmalle. Alkuperäinen, ennustettu ja muuntaminen kolmiulotteiseksi (Punainen jalkakäytävät, vihreä tie ja rakennukset sininen.)

Kuten kuvasta 7 voidaan nähdä, kuvan sisältäessä paljon pienempiä asioita tulee kuvan muuntamisessa kolmiulotteiseksi virheitä. Jalkakäytävät eivät ole muodostuneet kunnolla ja eivät tarjoa tarpeeksi syvyyttä kuvaan. Suurimpana syynä on ihmisen ja puiden oleminen jalkakäytävän ja tien edessä. Tämän vuoksi rakennukset tulevat tien eteen sen sijaan, että ne tulisivat viereen. Tie olisi jatkunut vielä eteenpäin, kunnes taustalla oleva rakennus olisi tullut vastaan. Myös tien koko on liian suuri verrattuna muuhun ympäristöön. Tarkoituksena on muodostaa ainoastaan rakennusten julkisivut laskentatehon säästämiseksi, koska pelaaja ei esimerkiksi pääse rakennuksiin sisälle. Rakennuksista olisi tullut onttoja, mutta halvempia piirtää.

Pisteitä tulee muotoon monta, koska luetaan kuvan jokainen pikseli. Muotojen monimutkaisuuden vuoksi on tärkeää säilyttää pisteitä, jotta muodot ovat lähellä todellisuutta. Mutta tämä aiheuttaa Unityn puolella ongelmia, jos käytetään 16-bittistä muodon pisteiden määrää. 16-bittisyys tukee vain 65535 pistettä yhdessä muodossa, mikä ylittyy helposti muodoissa. Onneksi Unity tukee 32-bittisyyttä pisteiden muodon formaattina, joka mahdollistaa 4 biljoonan pisteen olemista yhdessä muodossa (11). Tämä on kuitenkin raskaampi laskea ja ei ole tuettuna jokaisessa laitteessa (enimmäkseen Android-pohjaisissa käyttöjärjestelmissä).

4 Teorian toteutus ja projekti

Tämä kappale käsittelee projektin kulkua ja miten työssä olevia teoreettisia osuuksia käytettiin hyväksi. Projektissa tekoälyn opettaminen piti aloittaa nollatasolta eli ilman mitään edellisiä viittauksia tai ennustuksia. Tähän ongelmaan kuitenkin sai apua Kajaanin ammattikorkeakoululta, joka lainasi BULL- supertietokoneen yhtä NVIDIA bladea, jonka tekniset tiedot ovat kuvassa 9. Tämä mahdollisti nopeamman opettamisen, käyttöjärjestelmän ja pakettien asentamisen ja kulujen minimoinnin (sähkö ei ole ilmaista). Näin pystyttiin aloittamaan varsinainen projektin tekeminen sen sijaan, että aikaa olisi kulunut kehitysympäristön pystyttämiseen.



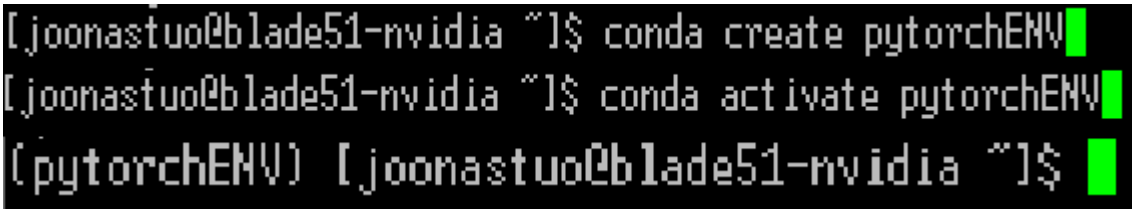
Kuva 9. BULL-supertietokoneen yhden bladen tekniset tiedot

Kuten nähdään, graafiset suorittimet ovat paljon tehokkaampia kuin keskinkertaisessa pelitietokoneessa. Myös SSD-vaihtoehto mahdollisti nopean pakettien asentamisen ja datan nopean siirtelyn paikasta toiseen. Käyttöjärjestelmänä toimi Centos7 (Red Hat Enterprise Linux 7), joka on Linux-pohjainen. Käyttöjärjestelmä on suunniteltu ja optimoitu palvelinkäyttöön, joka sopii tällekin projektille, koska ei pystyttäisi olemaan paikan päällä koronan vuoksi. Kaikki kommunikaatio toimi SSH (Secure shell) –yhteydellä, joka mahdollisti konsolin käytön palvelinympäristössä etäyhteydellä, joka mahdollisti työskentelyn kotona. Tämä rajoitti joitain työtapoja, joita käsitellään seuraavissa kappaleissa.

4.1 Työn ympäristö ja tekoälyn toteutus

Vaikka kehitysympäristössä oli vaadittavia paketteja valmiiksi asennettuna, piti kuitenkin koneoppimiseen tähdättyjä paketteja alkaa asentamaan. Projektin aloitusvaiheessa oli tietoa enimmäkseen teoreettisesti tekoälystä, ja kirjatkin käsittelivät enemmänkin algebraa kuin toteutusta. Työssä käytetään enemmän korkean tason kirjastoja ajan säästämiseksi. Funktioiden kirjoittaminen tyhjästä ja neuroverkon suunnittelu olisi vienyt enemmän kuin tähän työhön sallittu aika.

Työssä käytettiin PyTorch-kirjastoa, jonka asentaminen on helppoa, koska siitä on saatavilla valmis asennettava paketti. Tämä vaatii kuitenkin Pythonin ja joitakin muita kirjastoja. Projektinhallinnan ja käyttöjärjestelmän eheyden säilyttämiseksi valittiin käyttöön Conda, joka on avoimen lähdekoodin paketinhallintajärjestelmä. Condan ympäristö toimii osittain eristetysti. Jotta Condan voisi asentaa paketteja, pitää ensin luoda ympäristö. Sitten ympäristö pitää aktivoida, jolloin konsolinäkymään tulee suluissa ympäristön nimi. Kuva 10 visualisoi tämän tapahtumasarjan. Tämän jälkeen voidaan asentaa tarvittavia paketteja, kuten PyTorch-kirjasto.



Kuva 10. Ympäristön luominen ja aktivointi

Vaikka PyTorch-kirjasto on saatavilla käyttöjärjestelmän omasta paketinhallintajärjestelmässä (yum), pitää Nvidia bladen kortteja varten ladata tietty versio kirjastosta. Tällä hetkellä saatavat paketit ovat sopivia CUDA 10.+-korteille, mutta projektissa tarvitaan 10.0-versiota. Tämän version saa arkistosta, mutta tarvitaan Conda-järjestelmää sen asentamiseksi. Tämä olikin ensimmäinen syy Condan käyttöönottoon. Toisena syynä on käyttöjärjestelmän eheyden säilyttäminen. Alussa piti testata useita eri kehitysympäristön kokoonpanoja, mikä olisi syönyt kaiken tilan kovalevyltä. Condassa voi poistaa ympäristöjä helposti, joten levytilan pieni koko ei ole esteenä.

PyTorch tarjoaa muutamia valmiita tekoälyratkaisuja ja koneoppimisen malleja, jotka ovat valmiiksi koulutettuja. Työn käyttötarkoitukseen ei löytynyt valmiiksi koulutettuja malleja, joten piti kouluttaa oma. Haasteena on löytää sopiva koneoppimisen malli, koska vaihtoehtoja löytyy

useita. Työ vaatii kuvien tulkintaa ja asioiden ja esineiden paikkojen saamista. Nämä vaatimukset rajaavat koneoppimisen malleja runsaasti, eli halutaan konenäön malleja. Osa konenäön malleista perustuu yksittäisten esineiden tunnistukseen ja osa esineiden rajaamiseen ja paikallistamiseen. Työssä alun perin pyrittiin käyttämään RCNN-ratkaisua. RCNN on aluepohjaisesti toimiva konvoluutioinen neuroverkko, joka pyrkii paikantamaan esineitä ja asioita kuvista suorakulmioisiin alueisiin. Tämä aiheutti virheitä ja oli kallis opettaa. Myös ulostulomuuttujien käsittely olisi vaatinut enemmän lisätutkimusta. RCNN antaa ulostulona useamman maskin, jota ei tekemisen hetkellä ymmärretty. Myös valmista datasettiä ei löydetty mallille, mikä olisi aiheuttanut lisätöitä. Datasetit nimittäin sisältävät satoja, jopa tuhansia kuvia ja esineiden luokittelu kuvista olisi ollut paitsi työlästä, mutta myös aikaa vievää. Onneksi löytyy sivusto nimeltä Kaggle, jossa koneoppimisen kehittäjät ovat jakaneet valmiita datasettejä hyödynnettäväksi.

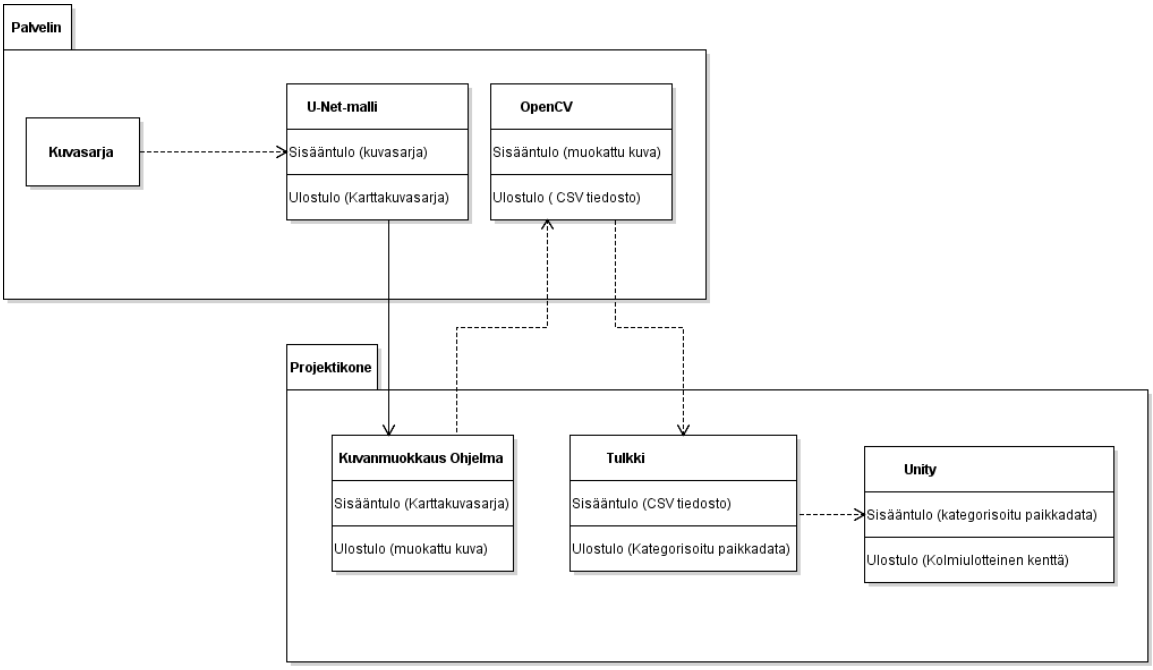
Työn tarpeisiin tarvitaan katukuvia, jotka on luokiteltu. Tarpeisiin löytyi tarpeeksi lähellä oleva datasetti, joka hyödynsi U-Net-mallia. Kuten edellisissä kappaleissa on todettu, U-Net-malli toimii hyvin samanlaisesti kuin RCNN, mutta ulostulona on yksi maski ja arkkitehtuuri on hieman erilainen. Yhden maskin laittaminen kuvaan on yksinkertaisempaa ja nopeampaa kuin useamman eri maskin käsittely ja liittäminen kuvaan. Tämä tuottaa kuvan 6 oikeanpuolisen kartan, jossa esineet ovat värikoodattuja. Tämä helpottaa seuraavaa työvaihetta, jossa kategorisoidaan pikselit. Työssä pyritään lajittelemaan rakennusten, teiden ja jalkakäytävien pikseleitä. Loput merkataan käyttämättömiksi ja pudotetaan pois pelimoottoriin viennin yhteydessä.

4.2 Datan vienti ja kentän luonti

Koska tekoäly isännöidään etäpalvelimella, pitää maskin tieto pakata lähetettävään muotoon. Koska maski tulostetaan tuloksena useamman kartan kuvaan, pitää sitä muokata. Kuva ladataan palvelimelta SSH-yhteydellä ja muokataan sopivalla kuvanmuokkausohjelmalla. Maskillinen kartta leikataan ja liitetään uuteen kuvaan, joka voidaan kääntää CSV-tiedostoksi. Tämä välivaihe olisi parempi hoitaa suoraan palvelimella, jotta voitaisiin vähentää tarpeetonta datan siirtelyä. Jotta kuvasta saadaan CSV tiedosto, pitää ladata OpenCV. OpenCV sisältää valmiita metodeja, joilla kuvan voi kääntää Numpy-taulukoksi, joissa pikselien väriarvot on ryhmitelty kaksiulotteiseksi taulukoksi. Tämä on helppo kirjoittaa suoraan CSV-tiedostoksi, koska Python sisältää CSV-kirjoittamisen valmiiksi ja arvot ovat jo ryhmiteltyinä. Arvot kirjoitetaan taulukon järjestyksessä,

jossa taulukoiden arvot erotetaan pilkuilla. Koska taulukko on kaksiulotteinen, voidaan määritellä rivien ja sarakkeiden määrä. Tämä kirjoitetaan tiedostoon rivinvaihtona, jotka kirjoitetaan aina, kun yhdestä rivistä loppuu sarakkeet. Tämä tuottaa pilkulla erotetun listan, jossa rivit on eroteltu rivivaihdoilla. Pikselien väriarvot on tulostettu BGR värimallin muodossa, joka pitää muistaa tulkinnan vaiheessa. CSV-tiedoston voi siirtää palvelimelta Unity projektiin, missä sen voi lukea ja kääntää kolmiulotteiseksi.

Jotta CSV-tiedoston tietoja voidaan käyttää, pitää se ensin tulkita. Työssä aloitetaan prosessi tiedoston lukemisella, joka on mahdollista ilman kolmannen osapuolen C#-kirjastoja. Data on hyvin yksinkertaista ja hankalin osuus onkin sen kategorisointi. Kuva 11 näyttää koko prosessin yleisesti. Koska lisättiin rivinvaihto jokaisen rivin loppuun, voidaan lukuvaiheessa määritellä rivien ja sarakkeiden määrä. Sarakkeiden määrä on arvojen määrä yhdellä rivillä ja rivien määrä on sidonnainen rivienvaihdon määrään. Näillä tiedoilla voidaan C#-puolella luoda kaksiulotteinen taulukko, jonne säilöä arvot. Tulkki palauttaakin kuvioiden luojalle kyseisen taulukon, jossa taulukon arvoina toimii merkkijono.



Kuva 11. Yleisdiagrammi kuvan muuntamisesta kolmiulotteiseksi

Kuvioiden luoja asettaa kolmiulotteiseen kenttään pisteitä, jotka muodostavat kuvion. Kategorisoimalla voidaan määritellä, pitääkö muodon pisteet asetella ylöspäin, kuten rakennusten julkisivut. Teiden ja jalkakäytävien pisteet voidaan olettaa olevan maan tasolla pienellä nousulla, mutta prototyypin tapauksessa ei pyritty simuloimaan tien pinnan muutoksia. Pisteet yhdistetään toisiinsa kolmioilla, jotta saadaan piirrettyä muoto moottorissa. Koska muodot ovat osittain hankalia ja eivät ole aina täysin suorakolmioita, on kolmioiden määrittäminen hankalaa. Ratkaisua tähän ongelmaan ei ole vielä löytynyt, joten pisteiden sijaan sijoitetaan kuutioita. Kuutiot ovat hyvin primitiivisiä ja halpoja piirtää, mutta niitä tulee olemaan tuhansia, mikä aiheuttaa monen eri kuvion piirtämisen. Unity tarjoaa valmiin metodin, jonka avulla voidaan yhdistää kuvioiden piirtämisen. Tämä laskee piirtämisen määrää ja vaatii vähemmän välimuistia. Ongelmana on pisteiden määrä yhdessä kuviossa, koska yhdessä kuutiossa on 8 pistettä. Eli kuvion alkuperäinen määrä kertaantuu kahdeksalla. Tämä aiheuttaa kuvioissa vääristymiä, jos kuvion pisteiden formaattia ei muuta 32 bittiseksi (kts. 3.2). Kuviot ovat omia pelin esineitä, joten niiden koon muuttaminen ja siirtäminen on mahdollista. Lopputuloksena on kuvan 8 oikeanpuolinen kuva, jossa vihreä on tie, punainen jalkakäytävä ja siniset ovat rakennusten julkisivut.

5 Pohdinta

Tekoälyn opettaminen on tehty helpommaksi, koska Pythonille on suunniteltu monia erilaisia kirjastoja, joissa on valmiina tarvittavat funktiot ja toiminnot. Työn aloittaminen alkoi enemmänkin koneoppimisen teorian tutkimisella, vaikka alhaisen tason matemaattisten funktioiden kirjoittamista ei tarvitsekaan. Enimmäkseen tarkoituksena oli kartoittaa konenäön malleja, jotta voitaisiin valita työn kannalta edullisin ja hyödyllisin malli. Työn alussa hyödynnettiinkin RCNN-mallia, kunnes huomattiin sen kalleus ja turhan tiedon määrä. Vaikka esineet saatiinkin hyvin rajattua kuvista, tuli mukana myös turhaa tietoa, kuten alueen koko. RCNN-malli pyrkii myös rajaamaan esineet laatikoihin, mikä olisi aiheuttanut väärentymää kuvioissa.

U-Net-malli sopii paremmin käyttötarkoitukseen keveytensä vuoksi. Kouluttamisessa OLY tarjoaa parempaa lopputulosta luokittelussa, koska se käy tulosta läpi joka noodissa. U-Net-mallin opettaminenkin on yksinkertaisempaa kuin RCNN. U-Net-malli tuottaa neuroverkon tuloksena yhden maskin, joka on yhdistelmä muita maskeja, kun taas RCNN tuo jokaisen maskin erikseen. Joten RCNN:n jokainen maski pitää käsitellä erikseen ja laittaa kuvaan, jotta saataisiin haluttu lopputulos. Tämä tekee siitä raskaamman käsitellä. Ennustamisen nopeudessa molemmat ovat hyvin lähellä toisiaan, mutta U-Net on hieman hitaampi. Tämä johtunee maskien yhdistämisestä jokaisessa kuvassa. Prototyypissä U-Net sopii paremmin, koska halutaan vain paikallistaa asiat kuvista. Kuitenkin olisi varmasti ollut parempi, että olisi otettu esineiden etäisyys huomioon, koska perspektiivin huomioon ottaminen on vaikeaa maskista. Tätä pyrittiin kompensoimaan jalkakäytävien avulla, joiden muoto antoi perspektiivin vahvuuden ja etäisyyden. U-Net-mallin nopea opettaminen (kaksi tuntia per 100 kuvaa) sopi paremmin opinnäytetyöhön tiukan aikataulun vuoksi. Kuitenkin yhden maskin ulostulo vaikeuttaa sisällä olevien maskien korjaamista, koska maskit tulostetaan yhteen kuvaan. Korjaaminen onnistuu vain kuvankäsittelyohjelmalla, joka vahingoittaa alkuperäistä konseptia automaattisesta kenttien luomisesta.

Ajatuksena oli muodostaa rakennukset jalkakäytävien viereen ja nostaa niitä kuvien y-akselin mukaisesti. Tässä ongelmana on valmiiksi valitut datasetit, joissa on luokiteltu ihmiset, lyhtypylväät, puut jne. Koska prototyypissä ei pyritä muodostamaan pienempiä esineitä, on nämä merkitty ohitettaviksi. Tämä aiheuttaa lopputuloksessa vääristymiä ja aukkoja, jotka hankaloittavat kentän

todellisen muodon luomista. Myös rakennuksien paikat ja muodot poikkeavat, koska jalkakäytävät ovat osittain ihmisten peitossa käytetyissä kuvissa. Jos kuvat olisi luokitellut itse, olisi lopputulos varmasti parempi.

Tulkkauksessa olisi auttanut, jos kuvan väriarvojen lisäksi olisi ollut etäisyys. Etäisyyden avulla olisi pystynyt suhteuttamaan esineiden kokoa ja luomaan realistisemman kentän. Ongelmana on U-Net-malli itsessään, joka ei ole suunniteltu mittaamaan etäisyyksiä. U-Net-malli tuo vain yhden maskin, joka on väritelty värikoodeittain, kuten kuvan 7 oikeanpuolinen kuva näyttää.

6 Yhteenveto

Koneoppimisen sovellutuksia on monia, ja tässä käytiin niistä yksi. Kentän luominen kaksiulotteisesta kuvasta on haasteellinen ja monimutkainen. Lähestymistapa, joka tässä käytiin, ei välttämättä ole optimaalisin ratkaisu ja tarkin. Muotojen pisteiden yhdistäminen vähentäisi laskemisen määrää ja muotojen jaottelu mahdollistaisi 16-bittisen formaatin. Perspektiivin jäljittely on suurin ongelma, joka kaksiulotteisen kuvan muuntamisessa kolmiulotteiseksi yleensä on.

Tulkin toiminta on yksinkertainen ja tarvitsisi vielä lisäkehitystä. Pikselidatan kategorisoimisen yhteydessä olisi hyvä saada myös etäisyyksien arvot. Tämän avulla pystyttäisiin käsittelemään perspektiiviä ja syvyyttä helpommin. U-Net-mallia ei ole suunniteltu tähän tarkoitukseen, vaan enemmänkin asioiden tunnistamiseen. Jos olisi käytetty omaa kategorisoitua dataa, niin lopputulos olisi tarkempi. Prototyypistä voidaan kuitenkin todeta, että kenttien automaattinen rakentaminen kuvan pohjalta on mahdollista. Tekoälyn pitempiaikainen opettaminen olisi johtanut tarkempaan ja parempaan lopputulokseen. Ennustuksista käy ilmi, että joitain sattumia on kuvioiden keskellä, joka hankaloittaa kuvioiden pisteiden yhdistämistä ja aiheuttaa vääristymiä.

Käytetyissä kuvissa olevien esineiden mittasuhteet ovat vääristyneitä perspektiivin vuoksi, mikä aiheutti suuria kokoeroja. Jokainen datasetin kuva oli kuvattu katsojan perspektiivistä, joka suurentaa mittasuhteita lähellä oleviin esineisiin ja toisinpäin. Datasetin ja oppimisen mallin tarkempi valinta olisi tuottanut paremman lopputuloksen, ja mittasuhteet olisivat lähempänä kuvan antamia suhteita. Paikkadatan jatkojalostamisella ja enemmällä tiedolla saataisiin tarkempi lopputulos. Kuitenkin prototyyppi on näyttänyt, että on mahdollista tuottaa kaksiulotteisista kuvista kolmiulotteisia kenttiä.

Lähdeluettelo

(1) Murphy KP. Machine Learning : A Probabilistic Perspective. Cambridge: MIT Press; 2012.

(2) A. L. Samuel. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development 2000;44(1.2):206-226.

(3) Gokul Karthik. Image-segmentation-with-UNet-PyTorch. 2020; Saatavilla: https://www.kaggle.com/gokulkarthik/image-segmentation-with-unet-pytorch. Viitattu Lokakuu 12, 2020.

(4) Browniee Jason. Supervised and Unsupervised Machine Learning Algorithms. 2019; . Viitattu Dec 12 , 2019.

(5) Berkeley University of California. Classification methods. 2010;Saatavilla : https://www.stat.berkeley.edu/~spector/s133/Class.html. Viitattu Lokakuu 22, 2020.

(6) Zaccone G, Karim R. Deep Learning with TensorFlow : Explore Neural Networks and Build Intelligent Systems with Python, 2nd Edition. Birmingham: Packt Publishing, Limited; 2018.

(7) Sumit Saha. Konvoluutioisen hermoverkon perusteet. 2018;Saatavilla : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Viitattu Lokakuu 22, 2020.

(8) Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. Cham: Springer International Publishing; 2015.

(9) Agarap AF. Deep Learning using Rectified Linear Units (ReLU). 2018 Maaliskuu 22,.

(10) Flick Jasper. Neliöiden muodostaminen. Saatavilla: https://catlikecoding.com/unity/tutorials/procedural-grid/. Viitattu Lokakuu 21, 2020.

(11) Unity. Mesh Indexformat. 2020; Saatavilla: https://docs.unity3d.com/ScriptReference/Mesh-indexFormat.html. Viitattu Marraskuu 7, 2020.