Niko Kahilainen

Aimbottien havaitseminen keinotekoisilla neuroverkoilla

Tietotekniikan pro gradu -tutkielma

8. joulukuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Niko Kahilainen

Yhteystiedot: niko.s.kahilainen@student.jyu.fi

Ohjaaja: Ilkka Pölönen

Työn nimi: Aimbottien havaitseminen keinotekoisilla neuroverkoilla

Title in English: Aimbot detection using neural networks

Työ: Pro gradu -tutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 37+0

Tiivistelmä: FPS-peleissä aimbottien käyttäjillä on epäreilu etu suhteessa rehellisiin pelaajiin, mutta huijausohjelmien käytön havaitseminen perinteisin keinoin on jatkuvaa kissa ja hiiri -leikkiä. Ratkaisuksi tähän on usein ehdotettu ja sovellettukin erilaisia koneoppimisen menetelmiä, sillä peleistä on helppo kerätä monenlaista dataa pelaajien käytöksestä. Tässä tutkielmassa kerätään aineistoa pelisessiosta, ja tutkitaan miten yksinkertainen keinotekoinen neuroverkko onnistuu erottamaan huijausohjelman käyttöä rehellisestä pelaamisesta. Mallin suorituskykyä verrattiin aikaisempaan vastaavaan tutkimukseen aiheesta.

Avainsanat: aimbot, peli, huijaus, neuroverkko, koneoppiminen

Abstract: Aimbot users in FPS games have an unfair advantage compared to honest players, but cheat detection using traditional methods has spiraled into a neverending game of cat and mouse. One solution to this is the application of machine learning methods, since games make it easy to collect different kinds of data from player behavior. In this thesis, we gather data from live game sessions, and investigate how a simple neural network is able to differentiate cheat usage from honest gameplay. The performance of the model was also compared to previous relevant literature.

Keywords: aimbot, game, cheat, neural network, machine learning

Esipuhe

Kiitokset ohjaajalleni Ilkka Pölöselle tutkielman ohjauksesta ja kannustuksesta. Suuret kiitokset myös ystävilleni tutkielmaan osallistumisesta ja kannustamisesta.

Jyväskylässä 8. joulukuuta 2023

Niko Kahilainen

Termiluettelo

FPS-peli	*First-Person Shooter*, ensimmäisen persoonan räiskintäpeli. Peligenre, jossa pelaaja näkee pelimaailman ensimmäisestä persoonasta.
Deathmatch	Yleinen pelimuoto FPS-peleissä, jossa tavoitteena on päihittää mahdollisimman monta pelaajaa ennen erän loppumista.
Aimbot	Yleinen termi huijausohjelmalle, joka avustaa pelaajaa tähtäämisessä. Aimbotiksi lasketaan lukuisia huijauksia, esim. sellaiset huijausohjelmat, jotka seuraavat annettua kohdetta automaattisesti, mitätöivät aseen rekyylin, ampuvat heti, kun tähtäinlinjalla havaitaan vihollinen ym.
RNN	*Recurrent Neural Network*, takaisinkytketty neuroverkko.
CNN	*Convolutional Neural Network*, konvoluutioneuroverkko.

Kuviot

Taulukot

Sisällys

1 Johdanto

Verkkomoninpelien yhteydessä huijaaminen on ollut ongelmana niin pitkään kuin peligenre on ollut olemassa. Verkkopeleissä tapahtuvaan huijaamiseen liittyy useimmiten kolmannen osapuolen ohjelmisto, joka mahdollistaa huijaavalle pelaajalle epäreilun edun. Huijausohjelmisto voi esimerkiksi paljastaa normaalisti salaista tai piilotettua tietoa, kuten toisten pelaajien sijainteja, automatisoida toimintoja, tai jopa manipuloida pelitilaa. Koska pelinkehittäjät haluavat varmistaa kaikille pelaajille reilun pelikokemuksen, on heillä syytä kehittää keinoja huijaamisen havaitsemiseen ja estämiseen.

Huijausohjelmistojen havaitseminen ja estäminen ei ole kuitenkaan yksinkertainen tehtävä. Huijausohjelmistot toimivat yleensä hyödyntämällä erilaisia tietoturvahaavoittuvuuksia pelissä tai muussa pelin ajamiseen tarvittavassa ohjelmistossa. Sitä myötä kun huijaamisen mahdollistava ongelma korjataan, huijausohjelmistojen kehittäjät keksivät uuden metodin mikä mahdollistaa huijaamisen. Kyseessä on siis jatkuva kissa ja hiiri -leikki, jolle ei näy loppua huijauksenestonperinteisin menetelmin.

Lukuiset koneoppimismenetelmät ovat osoittautuneet hyvin soveltuviksi erilaisiin luokitteluja tunnistustehtäviin, kuten konenäköön ja puheentunnistukseen. Koneoppimismenetelmiä on sovellettu myös peleissä huijauksien havaitsemiseen, esimerkiksi Valven syväoppimista hyödyntävä VacNet (McDonald 2018). Koneoppimismenetelmien ytimessä on aineisto, josta malli oppii tunnistamaan erityispiirteitä ilman eksplisiittistä opettamista. Eräs syy koneoppimismenetelmien kasvaneeseen menestykseen aineistojen keräämisen helpottuminen. Esimerkiksi verkkopelien kontekstissa erilaisten tietojen ja tilastojen kerääminen pelipalvelimella on helppoa.

Tässä tutkielmassa pyritään vastaamaan seuraavaan tutkimuskysymykseen: Kuinka luotettavasti aimbotin käyttöä FPS-pelissä voidaan havaita tutkimalla pelaajan näkökulman liikettä tappojen yhteydessä hyödyntäen yksinkertaista keinotekoista neuroverkkoa? Tutkielmaa varten luodaan aineiston keräämisen mahdollistava ohjelmisto ja kerätään aineisto todellisilta pelaajilta. Aineisto sitten prosessoidaan, sovelletaan keinotekoiselle neuroverkolla, jonka jälkeen mallin suorituskykyä arvioidaan eri mittareilla.

Tutkielma koostuu seuraavasti. Luvussa 2 käsitellään huijaamista verkkopeleissä tarkemmin, miten se on mahdollista ja mitkä ovat yleiset motivaatiot sen takana. Luvussa 2 käsitellään myös tutkielmassa käytettävää kohdepeliä ja käydään läpi lyhyt kirjallisuuskatsaus vastaaviin tutkimuksiin. Luvussa 3 käsitellään tutkielmassa hyödynnettävää koneoppimismenetelmää ja sen yksityiskohtia. Luvussa 4 kuvataan miten aineisto kerättiin ja prosessoitiin, sekä koneoppimismenetelmän asetelma. Luvussa 5 käydään läpi tutkielman tulokset, ja lopuksi luvussa 6 pohditaan tuloksia ja verrastetaan niitä aikaisempien tutkimuksien tuloksiin.

2 Huijaaminen peleissä ja kohdepeli

Tämä luku avaa huijaamista tietokonepelien kontekstissa. Ensiksi määritellään mitä huijaaminen tietokonepeleissä on, miksi sitä tehdään, ja mikä sen mahdollistaa. Tämän jälkeen annetaan pieni esimerkki siitä, miten moninpelaaminen verkon välityksellä mahdollisestaan kohdepelissä. Tämä jälkeen käydään tutkielmassa käytetty kohdepeli pikaisesti. Luku päättyy pieneen kirjallisuuskatsaukseen vastaavista tutkimuksista.

2.1 Huijaaminen peleissä

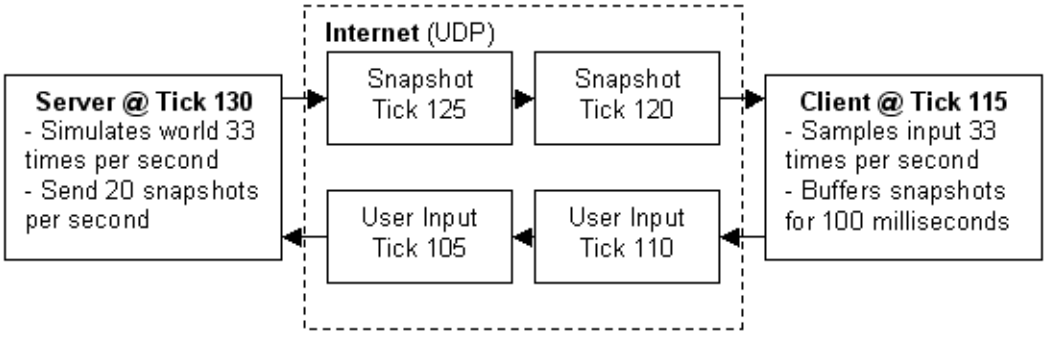
Verkkopelien suosion myötä on myös kilpailullisten pelien suosio kasvanut huomattavasti. Kilpailullisten pelien määritelmä on häilyvä, mutta niiden tärkein piirre on kuitenkin se, että pelaajat ovat suorassa kilpailussa toisiaan vastaan, joko pelaaja vastaan pelaaja tai joukkue vastaan joukkue. Kasvavan suosion ja kovemman kilpailun takia huijaaminen kilpailullisissa peleissä on suuri ongelma. Pelaajat huijaavat peleissä siitä saadun koetun hyödyn perusteella, vaikka huijaaminen saattaa samalla haitata muiden pelaajien pelikokemusta huomattavasti (Chen ja Ong 2018).

Maisterintutkielmassaan Lehtonen 2020 jakoi huijausmenetelmät peleissä kahteen kategoriaan, pehmeisiin (engl. *soft*) ja koviin (engl. *hard*) menetelmiin. Pehmeät menetelmät hyödyntävät edun hankkimiseen peliin sisäänrakennettuja mekaniikoita ja järjestelmiä semmoisin keinoin, joita pelinkehittäjä ei ole osannut ennustaa. Näitä voi esimerkiksi olla ohjelmointi- tai suunnitteluvirheet. Esimerkkinä suunnitteluvirheestä on roolipeleissä olevat pelinsisäiset kaupat, jossa pelaajat voivat myydä tavaroita suuremmalla summalla kuin minkä ne maksaisivat ostettaessa. Pehmeitä huijauksia on hankala estää ja havaita, koska ne eivät vaadi pelin muokkaamista. Kovat huijaukset hyödyntävät kolmannen osapuolen ohjelmia, jotka pyrkivät muuttamaan tai lukemaan pelitilaa aktiivisesti. Kovat huijaukset voivat esimerkiksi paljastaa pelaajalle normaalisti salaista tietoa, automatisoida tiettyjä toimintoja tai muuttaa pelitilaa mielivaltaisesti. Kun pelien kontekstissa puhutaan huijaamisesta, puhutaan yleensä juurikin kovista huijauksista. Huijauksenesto-ohjelmia kehitetään pääasiassa kovien huijauksien havaitsemiseen ja estämiseen.

Yksi esimerkki huijaukseen mahdollistavasta tietoturvaongelmasta on DLL-injektio. DLL-tiedostot, eli *Dynamic Link Library* -tiedostot ovat Windows-käyttöjärjestelmässä käytettäviä tiedostoja, joilla voidaan ladata ohjelmistoon ulkoisia kirjastoja, toiminnallisuutta tai jopa muuttaa ohjelman käyttäytymistä. DLL-injektio toimii DLL-injektorin avulla, joka avaa kohdeprosessin, eli tässä tapauksessa pelin, ja kirjoittaa DLL-tiedoston muistiosoitteen kohdeprosessin muistiin. Tämä johtaa siihen, että kohdeprosessi suorittaa DLL-tiedoston sisältämän koodin. Täten huijausohjelmalla voi olla pääsy prosessin muistiin, mikä mahdollistaa huijaamisen.

2.2 Asiakas-palvelin-arkkitehtuuri peleissä

Tämän tutkielman kohdepelissä ja monessa muussa verkkopelissä hyödynnetään asiakas-palvelin-arkkitehtuuria moninpelin mahdollistamiseksi internetin yli. Pelien kontekstissa tämä tarkoittaa yleisesti sitä, että peliä ajava palvelin on ainut auktoriteetti pelitilan suhteen. Asiakasohjelmat, eli tässä tapauksessa pelaajat, voivat vaikuttaa pelin tilaan rajoitetusti vain pelipalvelimen sallimin keinoin. Yleisimmin tämä tarkoittaa sitä, että pelipalvelin vastaanottaa pelaajilta vain syötteeseen liittyvää dataa, jonka palvelin sitten prosessoi, ja lähettää tämän tiedon perusteella muuttuneen pelitilan takaisin pelaajille. Koska yksikään pelaaja ei voi suoranaisesti vaikuttaa pelitilaan, mahdollisuudet huijaamiselle vähenevät. Kuvio 1 kuvaa miten pelipalvelin ja asiakasohjelmat toimivat keskenään kohdepelissä.



Kuvio 1. Havainnekuva verkkoarkkitehtuurista Source-pelimoottorissa (“Source Multiplayer Networking - Valve Developer Community”, n.d.)

Verkkoviiveen ja prosessointinopeuksien takia pelipalvelin ja asiakasohjelma eivät voi päi-

vittää pelitilaa reaaliaikaisesti. Peleissä pelitilan päivitys tapahtuu yleensä diskreetein askelin sovituin väliajoin. Kohdepelissä ja muissa Source-pelimoottorin peleissä yksittäistä palvelimen tilaa kutsutaan termillä *tick*, ja pelitilan päivitysnopeutta kutsutaan termillä *tickrate*. Yhden päivityksen aikana pelipalvelin siis prosessoi muilta pelaajilta saamansa syötteen, simuloi mahdolliset fysiikat, tekee tarkastuksen pelisäännöistä ja päivittää pelitilan pelaajille ("Source Multiplayer Networking - Valve Developer Community", n.d.). Nopeatempoiset pelit, joissa millisekuntienkin viivellä voi olla vaikutusta pelikokemukseen, tähtäävät yleensä korkeisiin päivitysnopeuksiin. Pelin päivitysnopeuden kasvattaminen kuitenkin johtaa suurempaan kuormaan palvelimella ja suurempaan verkkokaistan käyttöön.

Kohdepelin virallisilla palvelimilla pävitysnopeus on 64. Tämä tarkoittaa sitä, että pelipalvelin ja asiakasohjelma vaihtavat tietoja 64 kertaa sekunnissa. Pelissä on myös mahdollista isännöidä omia pelipalvelimia, joissa päivitysnopeuden voi asettaa jopa 128 kertaan sekunnissa, johtaen sujuvampaan pelikokemukseen. Tämä luku on poikkeuksellisen suuri jopa kohdepelin genressä.

2.3 Kohdepeli

Tämän tutkielman kohdepeli on Counter-Strike: Global Offensive (lyh. CS: GO). CS: GO on Valven vuonna 2012 julkaisema suosittu ensimmäisen persoonan räiskintäpeli (engl. *First Person Shooter, FPS*), ja se on samalla viimeisin jatko-osa Counter-Strike-pelisarjalle.[1]

Pelin kilpailullisessa pelimuodossa kaksi viiden pelaajan joukkuetta, terroristit ja vasta-terroristit, pelaavat toisiaan vastaan. Terroristien tehtävä on asettaa pommi pelikentälle asetettuun pommipaikkaan, ja suojata sitä vastaterroristeilta pommin räjähtämiseen asti. Terroristit voittavat myös, jos saavat eliminoitua kaikki vastapuolen pelaajat. Vasta-terroristien tehtävä on taas puolustaa pommipaikkoja terroristeilta, ja he voittavat erän jos onnistuvat kuluttamaan erän ajan loppuun ennen pommin asettamista, eliminoilla vastapuolen pelaajat ennen pommin asettamista, tai purkamalla asetetun pommin. Pelin 16:sta erän kohdalla joukkueet vaihtavat puolia, ja ensimmäinen joukkue joka saavuttaa 16 erävoittoa voittaa pelin.

1. Peli sai päivityksen 27.9.2023, jonka yhteydessä myös pelin nimi muuttui. Peli tunnetaan nyt nimellä Counter-Strike 2

Kuvio 2. Kuvakaappaus kohdepelistä

2.4 Aikaisempi tutkimus

Tutkielmaa varten haettiin aikaisempaa tutkimusta kirjallisuuskatsauksella. Tähän hyödynnettiin tieteellisen kirjallisuuden hakukoneista IEEE exploreria, scopusta, ja google scholaria. Näissä hakukoineissa käytetiin hakutermeinä mm. game, cheat, cheat detection, aimbot ja machine learning.

Löydetyt artikkelit keskittyivät suurimmilta osin kahteen eri verkkopeligenreen; Massiivisiin verkkoroolipeleihin (eng. *Massively multiplayer online Role-playing game, MMORPG*), ja räiskintäpelehin, kuten myös Kotkov ja kumppanit ovat todenneet (Kotkov, Pandey ja Semenov 2018). Syynä tähän on todennäköisesti peligenrejen suosio, ja huijausohjelmien käytöstä saatu koettu etu. Esimerkiksi verkkoroolipeleissä, joissa pelaajien on käytettävä merkittävä määrä aikaa hahmojensa tai tiliensä kehittämiseen, on ongelmana ns. bottaaminen, eli kolmannen osapuolen ohjelman käyttäminen pelaamisen täydelliseen automatisointiin. Bottaamisella yleensä tavoitellaan pelitavaroiden tai valuutan keräämistä, tai pelitilien kasvattamista myyntitarkoitukseen pelin harmailla markkinoilla. FPS-pelien kohdalla huijausohjelmilla tavoitellaan vain pelaajan kykyjen keinotekoista parantamista esimerkiksi paljastamalla vihollispelaajien sijaintia tai tähtäyksen avustamista.

Motivaatio koneoppimismenetelmien hyödyntämisessä huijausohjelmien havaitsemiseen verk-

kopeleissä on kaksipuolinen. Ensinnäkin, huijausohjelmat yrittävät parhaansa mukaan imitoida ihmispelaajia noudattamalla pelaajalle asetettuja sääntöjä. Toiseksi perinteinen huijauksenhavaitseminen on jatkuvaa kissa ja hiiri -leikkiä pelin kehittäjien ja huijausohjelmien kehittäjien välillä. Joissakin tapauksissa tämä on myös mahdollinen yksityisyysongelma. Esimerkiksi Electronic Frontier Foundation on leimannut Blizzardin huijauksenestojärjestelmän Wardenin haittaohjelmana, sillä se skannasi pelaajien tietokoneilta mahdollisesti arkaluontoista tietoa (McSherry 2005). Toinen esimerkki yksityisyysongelmasta on ring-0-huijauksenestojärjestelmät. Riot Gamesin julkaisema FPS-peli Valorant hyödyntää huijauksenestoon Vanguard nimistä järjestelmää. Vanguardin osana on kernel-tason ajuri, joka teoriassa mahdollistaa alhaisimman tason hallinnan tietokoneessa. Koska Vanguard vaatii ajurin käynnistämisen tietokoneen käynnistämisen yhteydessä, on ajurin pelätty mahdollistavan erilaisia tietoturvariskejä käyttäjän tietokoneella (Orland 2020). VacNetin, eli Valven oman syväoppimista hyödyntävän huijauksenestoratkaisun kehittäjä John McDonald totesi luennossaan, että niin pitkään kuin pelaajalla on minkäänlainen fyysinen pääsy peliä pyörittävään alustaan, on sillä mahdollista suorittaa huijausohjelmiakin (McDonald 2018).

Koneoppimismenetelmien ytimessä on menetelmän kouluttamiseen käytetty aineisto, sekä sen määrä ja laatu. Kirjallisuuskatsauksessa nousi esille laaja kirjo erikaltaisia aineistoja. Aineistojen keruuseen käytettiin kahta eri metodia; joko nauhoittamalla tapahtumia suoraan pelipalvelimelta, tai hyödyntämällä pelin tarjoamaa replay-toimintoa.

Yu ym. 2012 tutkivat aimbottien havaitsemista mittaamalla pelaajan kursorin kiihtyvyyttä tähtäyksen yhteydessä, sekä kuinka pitkään pelaaja tähtää kohdettaan yhtäjaksoisesti. Näitä kahta muuttujaa arvioitiin Kolmogorov-Smirnovin testillä. Tutkimusta varten hyödynnettiin avoimen lähdekoodin peliä Cube. Aineiston keräämiseen osallistui 21 henkilöä, joista kukin pelasi peliä kuuden 10 minuuttia kestäneen pelierän ajan. Osallistujat pelasivat ensimmäiset neljä pelierää ilman huijausohjelmaa, ja viimeiset kaksi huijausohjelma päällä. Tutkimuksen tuloksena huijaava pelaaja havaittiin luotettavimmin tutkimalla pelkästään kohteen tähtäämisen ajankestoa, jolla saavutetiin jopa 93,65 % tarkkuus.

Alayed, Frangoudes ja Neuman 2013 keräsivät aineistoonsa poikkeuksellisesti 16 eri muuttujaa. Näihin kuuluivat mm. pelaajan tähtäys- ja osumatarkkuus, liike, liikeen muutos, ja käytetty ase. Aineisto kerättiin kahden pelaajan toimesta 18 pelierän ajalta, yhteensä noin

460 minuutin ajalta, hyödyntäen Trojan Battles nimistä peliä. Pelaajat vuorottelivat huijausohjelmiston käyttöä, poislukien kaksi erää, jossa ei käytetty huijauksia. Aineistossa oli myös erikseen luokiteltu huijausohjelman eri asetuksien käyttö. Mielenkiintoisesti tutkimuksessa käsiteltiin moniluokkaista analyysiä eri huijausohjelmien käytön perusteella, ja tämä tieto nosti mallien tarkkuutta. Aimbotin havaitsemisessa saavutettiin parhaimmillaan 98,2 % tarkkuus.

Galli ym. 2011 keräsivät Unreal Tournament 3 -pelistä pelaajan sijainti- ja liiketietoa, pelaajan etäisyyttä kohteeseen ja tähtäystarkkuutta. Näillä tiedoilla jalostettiin "kehys", joka sisälsi 30 sekunnin ajalta tietoa pelaajan toiminnasta. Aineisto sovellettiin sitten viidelle eri koneoppimismenetelmälle; Bayesin luokitin, päätöspuut, satunnainen metsä, keinotekoinen neuroverkko, ja tukivektorikone. Tutkimusta varten kerättiin oma aineisto kahden osallistujan toimesta, jotka pelasivat kaksi 20 minuutin pelierää tekoälypelaajia vastaan. Pelaajat saivat kytkeä pelin aikana huijausohjelmisto päälle tai pois oman tahtonsa mukaan. Kerätyn datan esikäsittelyn jälkeen dataa oli 250 alkiota miltein täydellisellä huijaava/oikea pelaaja-jakaumalla. Jokainen menetelmä saavutti yli 90% tarkkuuden, ja vahvimman tuloksen saivat Bayesin luokitin ja tukivektorikoneet.

Pinto, Pimenta ja Novais 2021 kehittivät ohjelman, joka tallensi suoraan pelaajan tietokoneelta syötedataa, kuten hiiren liikettä ja näppäimistön painalluksia. Koneoppimismenetelmän hyödynnettiin konvoluutioneuroverkkoja. Tutkimusta varten kerättiin poikkeuksellisen suuri aineisto, ja aineiston keräämiseen hyödynnettiin peliä Counter-Strike: Global Offensive. Aineiston keruuseen osallistui 118 henkilö, joista 8 huijasivat hyödyntämällä joko triggerbottia tai aimbottia. Keruun jälkeen aineistoa oli 490 tunnin verran, ja menetelmällä saavutettiin vakuuttava noin 99 % tarkkuus.

3 Koneoppiminen

Tässä luvussa alustetaan ensin koneoppimisen perusteita yleisellä tasolla. Tämän jälkeen perehdytään syvemmin tutkielmassa hyödynnettävää menetelmään, keinotekoisiin neuroverkkoihin.

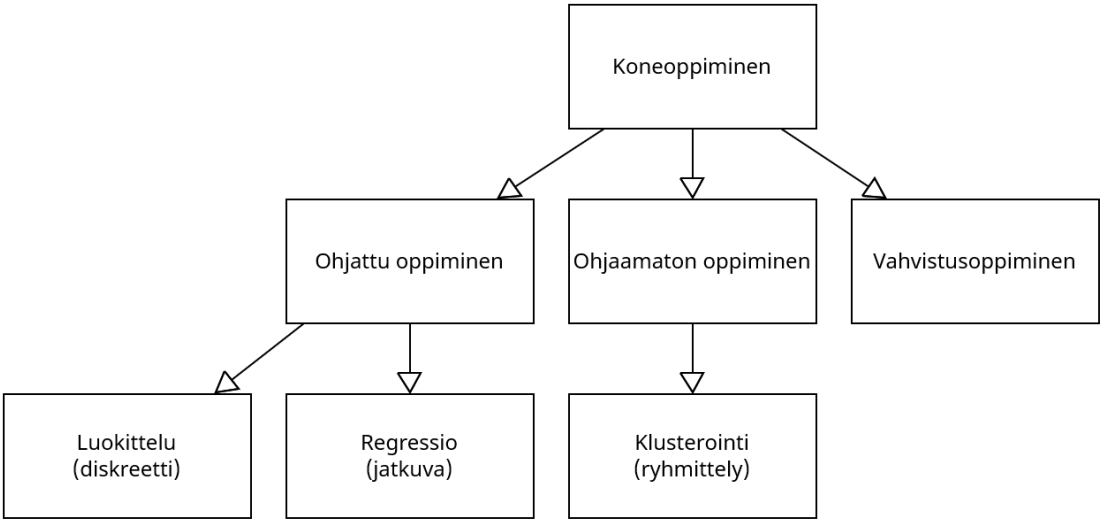
3.1 Koneoppiminen yleisesti

Koneoppimisella pyritään vastaamaan kysymykseen, kuinka rakentaa tietojärjestelmiä, jotka oppivat kokemuksen perusteella. Koneoppimen sijoittuu tietotekniikan ja tilastotieteiden välimaastoon, ja on mm. tekoälyn kehittämisen ytimessä (Jordan ja Mitchell 2015).

Koneoppimisen ytimessä on koneoppimismallin koulutukseen käytetty aineisto, josta malli oppii tunnistamaan siinä olevia piirteitä ja luokittelemaan niiden perusteella lopputuloksia ilman eksplisiittistä opettamista. Tekniikan kehityksen johdosta erilaisten aineistojen kerääminen on helpottunut, mikä on johtanut siihen että saatavilla olevat aineistot ovat laajempia ja kattavampia. Tämä taas on johtanut koneoppimismenetelmien suosion nousuun, sillä niillä pystyy melko vaivattomasti analysoimaan suuriakin aineistoja läpi (Jordan ja Mitchell 2015).

Koneoppimismenetelmät voidaan jakaa kolmeen eri lähestymistapaan käytetyn aineiston perusteella; ohjattuun oppimiseen, ohjaamattomaan oppimiseen, ja vahvistusoppimiseen. Ohjatussa oppimisessa opetusaineisto on ennalta luokiteltu, ja mallin tehtäväksi tulee tehdä syötteen perusteella oikea ennuste. Ohjaamaton oppiminen on ohjatun oppimisen vastakohta, sen opetusaineistoa ei olla erikseen luokiteltu. Ohjaamattoman oppimisen tavoitteena ei välttämättä ole ennusteiden tekeminen, vaan aineiston rakenteen tutkiminen ja sen erinäisten piirteiden löytäminen. Vahvistusoppiminen istuu ohjatun ja ohjaamattoman oppimisen välissä, siinä malli oppii iteratiivisesti ympäristöstä saamansa palautteen perusteella; malli saa ympäristöltään palautetta, oliko tulos tai ennuste hyvä vai ei, ja tämän avulla säätää parametrejaan parempian tuloksien toivossa. Tämän tutkielman aineisto on ennalta luokiteltu, joten siihen sovelletaan ohjatun oppimisen menetelmää.

Kuten menetelmän nimestä voi päätellä, koneoppimismallit tulee opettaa aineistonsa pe-



Kuvio 3. Koneoppimisen luokat

rusteella. Mahdollisimman pätevän mallin saamiseksi mallin tuloksia tulee arvioida. Tämä tapahtuu yleensä jakamalla mallin opettava aineisto kahteen osuuteen, opetusaineistoon ja arviointiaineistoon. Käytetystä menetelmästä riippuen mallia hienosäädetään iteratiivisesti saatujen tuloksien perusteella, verraten mallin tuloksia arviointiaineistoon. Tämä opetusprossessi eroaa eri menetelmien välillä. Lopullisen mallin arviointiin voi käyttää lukuisia eri metriikoita, luokitteluongelmissa esimerkiksi sekaannusmatriiseja ja luokitteluraportteja.

3.2 Keinotekoiset neuroverkot

Tässä tutkielmassa hyödynnetään koneoppimisen menetelmänä eteenpäinkytkettyä keinotekoista neuroverkkoa. Seuraavissa kappaleissa tarkastellaan neuroverkkojen toimintaperusteita, alkaen neuroverkon neuroverkkojen perusyksiköstä, eli neuronista. Tämän jälkeen käsitellään miten neuronit toimivat yhdessä neuroverkkona, ja miten neuroverkon koulutus tapahtuu. Kappaleiden tiedot on koottu lähteistä Lindholm ym. 2022 Goodfellow, Bengio ja Courville 2016, Tuominen ym. 2019, ja Hursti, Puuppo ja Pölönen, n.d.

3.2.1 Yksittäisen neuronin toiminta

Keinotekoiset neuroverkot koostuvat kerroksittain yksittäisistä neuroneista. Yksinkertaisimmillaan yksittäisen neuronin tehtävä on ottaa vastaan syötteitä prosessoitavaksi ja tuottaa yksittäinen ulostulo. Toisin sanoen yksittäinen neuroni on yksinkertainen luokitin, joka vahvistaa tai heikentää saamiaan signaaleita. Neuronin syötteinä voi toimia esimerkiksi muiden neuronien tulokset tai jokin datan ominaisuus. Syötteet voidaan esittää syötevektorina.

$$\mathbf{x} = (x_1, x_2, ..., x_n) \in R^n.$$

Jokaiselle syötteelle annetaan lisäksi oma painoarvo, joka käytännössä määrittelee sen, miten merkittävä kyseinen signaali on tuloksen laskennassa. Syötteen pieni paino viittaa siihen, että signaalin vaikutus neuronin tulokseen on pieni, ja suuri arvo taas sitä, että signaalilla on suuri vaikutus tulokseen. Syötteiden painoarvot voidaan myös esittää vektorimuodossa $\mathbf{w} = (w_1, w_2, ..., w_n) \in R^n$. Neuroni laskee näiden arvojen perusteella painotetun summa $\Sigma_{i=1}^{n} w_i x_i = w \cdot x$. Tämän summauksen jälkeen lisätään vielä neuronikohtainen vakiotermi θ, joka mahdollistaa neuronille joustavuutta ja toimii eräänlaisena säätöparametrina. Lopuksi yksittäisen neuronin summausvaiheen voi yksinkertaistaa seuraavaan muotoon

$$s = \Sigma_{i=1}^{n} w_i x_i + \theta = \mathbf{w} \cdot \mathbf{x} + \theta.$$

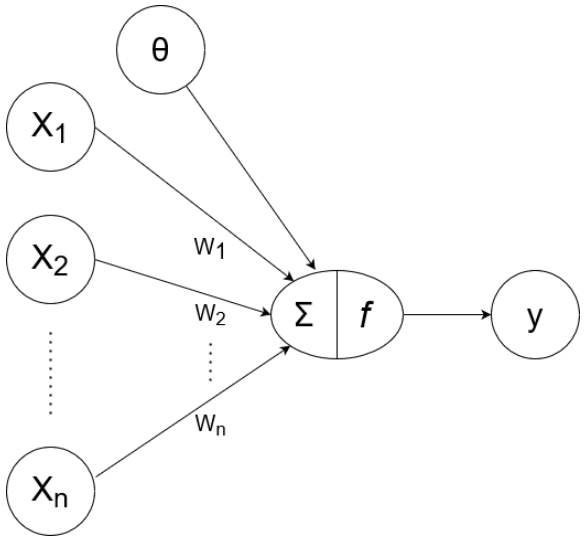
Summausvaiheen jälkeen meillä on käsissä lineaarinen funktio, joka soveltuu huonosti kompleksisten ongelmien ratkomiseen. Tämän ongelman ratkaisemiseksi neuronin tulos syötetään aktivaatiofunktiolle, jonka tehtävä on tuoda epälineaarisuutta neuronin tulokseen. Aktivaatiofunktioita on monia erilaisia, ja niillä jokaisella on hieman erilaiset ominaisuudet. Esimerkiksi ReLu, *Rectified Linear Unit*, palauttaa saamansa syötteen jos se on positiivinen, muuten se palauttaa nollan. Toinen yleinen aktivaatiofunktio on Sigmoid σ, joka tuottaa arvoja nollan ja ykkösen välillä.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$ReLu(x) = max(0, x)$$

Aktivaatiofunktion lisäämisen jälkeen yksittäisen neuronin kaavio voidaan esittää seuraavassa muodossa.

$$y = f(s) = f(\Sigma_{i=1}^{n} w_i x_i + \theta) = f(\mathbf{w} \cdot \mathbf{x} + \theta).$$



Kuvio 4. Havainnekuva keinotekoisesta neuronista

3.2.2 Neuroverkon toiminta

Yksittäinen neuroni ei vielä sovellu kompleksisten ongelmien ratkomiseen. Täten neuroneista muodostetaan kerroksia, jotka voidaan tiivistää kolmeen eri luokkaan, syötekerrokseen, piilokerrokseen ja tuloskerrokseen. Syötekerroksen tehtävä on ottaa vastaan syöte ja välittää se seuraaville kerroksille. Piilokerros on ns. välittäjäkerros, eli se ottaa edellisen kerroksen syötteet vastaan, prosessoi sen, ja lähettää signaalinsa eteenpäin seuraavalle kerrokselle. Keinotekoisissa neuroverkoissa voi olla mielivaltainen määrä erikokoisia piilokerroksia. Kun puhutaan syvistä neuroverkoista tai syväoppimisesta, tarkoitetaan sillä yleisesti keinotekoista neuroverkkoa, jolla on suuri määrä piilokerroksia. Viimeisenä tuloskerroksesta tulkitaan mallin tulos.

Neuroverkon yksittäisen piilokerroksen voi esittää matemaattisessa muodossa.

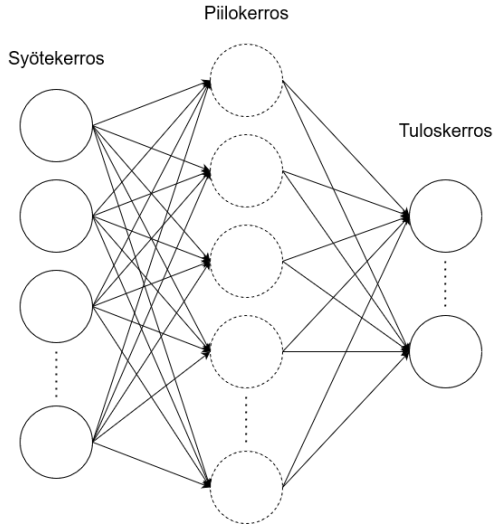
$$g(W^T \mathbf{x} + \theta).$$

missä W on matriisi kerroksen painoista $W = [w_1, w_2, ..., w_i]$, ja i kerroksen neuronimäärä.

Vastaavasti kaksikerroksinen piilokerroksen voidaan esittää seuraavassa muodossa.

$$F(x) = f(W_2^T g(W_1^T \mathbf{x} + \theta_1) + \theta_2)$$

Kuvasta 5 voi havainnoida yksinkertaisen, kaksikerroksisen eteenpäin kytketyn neuroverkon (engl. *Feed-Forward Neural Network, FNN*) rakenteen. Syötekerrosta ei yleensä lasketa mukaan kerroksiin. Tämänkaltaisessa neuroverkossa jokaisen kerroksen tulokset ovat kytköksissä seuraavan kerroksen syötteisiin. Tiedon virtaaminen tapahtuu syötekerroksesta piilokerroksiin, ja piilokerroksien kautta tuloskerrokseen.



Kuvio 5. Havainnekuva keinotekoisen neuroverkon rakenteesta

Eteenpäinkytketyn neuroverkon lisäksi on olemassa muunkaltaisia neuroverkkoja. Muut neuroverkkotyypit poikkeavat eteenpäinkytketyistä neuroverkoista esimerkiksi tiedon kulun suhteen, tai piilokerroksien ominaisuuksien perusteella. Mainitsemisen arvoisia esimerkkejä ovat takaisinkytketyt neuroverkot (engl. *Recurrent Neural Network, RNN*) ja konvoluutioneuroverkot (engl. *Convolutional Neural Network, CNN*). Takaisinkytkettyjen neuroverkkojen erikoispiirre on se, että neuronit voivat olla kytkeyttyinä itseensä tai aikaisempiin neuroneihin, mahdollistaen datan "kiertämisen"verkossa. Takaisinkytketyt neuroverkot soveltuvat erityisen hyvin esimerkiksi aikasarja-analyysiin. Konvoluutionverkkojen erikoispiirre on

konvoluutiokerrokset, jotka toimivat eräänlaisena ikkunointimekanismina. Esimerkiksi kuvatunnistusongelmissa konvoluutiokerros käy läpi kuvan pieni osio kerrallaan, yrittäen tunnistaa syötteestä piirteitä pieni osio kerrallaan.

3.2.3 Keinotekoisen neuroverkon koulutus

Keinotekoisen neuroverkon kouluttamista varten on ensin määriteltävä kustannusfunktio. Kustannusfunktion tehtävä on toimia metriikkana siitä, kuinka paljon mallin antama tulos eroaa oikeasta tuloksesta. Kustannusfunktioita on olemassa paljon erilaisia, ja osa niistä soveltuu tiettyyn tehtäviin paremmin kuin toiset. Regressio-ongelmiin voidaan esimerkiksi soveltaa pienimmän neliösumman menetelmää (engl. *Least squares*), ja luokitteluongelmissa ristientropiaa (engl. *Cross-Entropy*).

Mallin koulutuksen tehtävä on minimoida kustannusfunktio. Tämä tapahtuu käytännössä säätämällä verkon painoarvoja ja vakiotermejä siten, että kustannusfunktion tulos saadaan mahdollisimman pieneksi, eli löydetään globaali minimi. Kustannusfunktion minimointiin käytetään yleensä gradienttimenetelmiä.

Neuroverkkoja koulutetaan vastavirta-algoritmilla (engl. *Backpropagation*). Algoritmi alustaa ensin neuroverkon jokaisen neuronin satunnaisilla painoarvoilla ja vakiotermeillä. Alustuksen jälkeen mallille aletaan syöttämään koulutusdataa, ja mallin tulosta verrataan haluttuun tulokseen kustannusfunktion avulla. Kustannusfunktion tuloksen perusteella lasketaan virheen gradientti, jota käytetään verkon painojen päivittämiseen tuloskerroksesta takaisinpäin.

Yksinkertaisimmillaan gradienttimenetelmän voi selittää seuraavin keinoin. Kuvitellaan kolmiulotteinen, kumpuileva maasto, jossa on huippuja ja laaksoja. Tästä maastosta halutaan löytää laakson pohja, tai syvimmän laakson pohja, eli toisinsanoen kustannusfunktion minimi. Gradienttimenetelmällä siirrytään aina jyrkintä suuntaa, eli gradienttia kohti askel kerrallaan, kunnes laakson pohja on saavutettu, tai toisin sanoen parametrit konvergoituvat. On syytä huomioida, että tämä esimerkki pohjautuu tilanteeseen, jossa optimoitavia parametreja on vain muutama. Koska neuroverkolla optimoitavia parametreja on kaikki verkon painot ja vakiotermit, minimin etsiminen tapahtuu oikeasti n-ulotteisessa avaruudessa.

Neuroverkkojen koulutuksessa voidaan kohdata mm. seuraavia ongelmia.

- Katoava gradientti. Kun mallin parametreja optimoidaan syvemmälle taaksepäin, gradientti voi heiketä exponentiaalisti. Mallin varhaisemmat kerrokset saavat vain pieniä päivityksiä, mikä voi johtaa siihen, että malli oppii heikosti tai ei lainkaan.
- Räjähtävä gradientti. Vastakohta katoavasta gradientista, jossa gradientti kasvaakin nopeasti, päivittäen painoja suurin määrin johtaen epävakaaseen oppimiseen.
- Alisovittaminen ja ylisovittaminen. Alisovittamisessa malli jää liian yksinkertaiseksi, eikä pysty oppimaan datan syvempiä piirteitä. Ylisovittaminen taas tarkoittaa sitä, että malli on käytännössä opetellut aineistonsa ulkoa, eikä suoriudu hyvin ennennäkemättömän datan kanssa.
- Globaali ja lokaalit minimit. Kuten aikaisemmin mainittiin, gradienttimenetelmissä koulutus tapahtuu moniulotteisessa avaruudessa. Tällä avaruudella voi olla useita lokaaleja minimiarvoja, mutta vain yksi globaali minimi. Haasteksi nousee päätellä se, onko malli saavuttanut oikean globaalin minimin, vaiko jonkun lokaalin minimin.

Koulutuksen ongelmia voi ratkoa monesta eri näkökulmasta. Modernit koneoppimiskirjastot kuitenkin tarjoavat enemmän tai vähemmän automaattisia keinoja ongelmien ratkaisemiseen käyttämällä erilaisia optimointifunktioita.

mallin koulutusalgoritmin toimintaa ohjataan hyperparametreiksi kutsutuilla parametreilla. Neuroverkkojen tapauksessa näihin kuuluu esimerkiksi piilokerroksien määrä ja koot, neuroneissa käytetty aktivaatiofunktio, L2-arvo, ratkaisija ja oppimisnopeus. Näitä hyperparametreja säätämällä pyritään optimoimaan mallin suorityskyky, ja koulutuksen tehokkuus. Modernit koneoppimiskirjastot tarjoavat myös keinoja hyperparametrien optimointiin.

4 Menetelmä ja aineisto

Tässä luvussa esitellään tutkielmassa käytetyn aineiston luonne. Ensin käydään läpi miten aineisto kerättiin, jonka jälkeen avataan miten aineiston prosessoitiin, jotta se sopisi koneoppimismenetelmälle. Tämän jälkeen käydään lyhyesti läpi neuroverkon rakenne ja miten siihen päädyttiin. Viimeiseksi avataan hieman niitä mittareita, joiden avulla menetelmän suorituskykyä arvioidaan.

4.1 Aineiston kerääminen

Koska tutkielmaa varten ei löytynyt toivotunkaltaisia avoimia aineistoja, suoritettiin aineiston keruu itsenäisesti.

Aineiston keruuta varten kehitettiin ensin SourceMod-lisäosa. Lisäosa mahdollisti pelaajan näkökulman poikkiakselin (engl, *pitch*) ja pystyakselin (engl, *yaw*) suunnan (katso havainnekuva 6) tallentamisen suoraan pelipalvelimelta tappojen ympärillä, puoli sekuntia ennen ja puoli sekuntia jälkeen csv-tiedostoon. Pelipalvelin ajoi peliä 128 tickin päivitysnopeudella, joten jokaisesta taposta koostuva sekvenssi sisälsi 128 arvoparia. Lisäosan lähdekoodi on saatavilla julkisesta GitHub-repositoriosta (Kahilainen 2023).

Kuvio 6. Havainnekuva näkökulman liikkeestä

Rehellisen aineiston keräämiseen osallistui yhteensä 12 henkilöä, tutkija mukaanlukien. Ennen keräämistä osallistujia pyydettiin täyttämään vapaamuotoinen kuvaus kokemuksestaan kohdepelistä. Tämän jälkeen osallistuja liittyi tutkijan ylläpitämälle pelipalvelimelle pelaamaan deathmatch-pelimuotoa tietokoneen ohjaamia hahmoja vastaan 15 minuutiksi. Esikyselyn perusteella osallistujien kokemustason kirjo oli melko laaja, osallistujista neljällä oli yli 1000 pelituntia, kolmella yli 100 ja lopuilla näiden kahden välissä. Osallistujista muutama osallistui useammalle keruusessiolle.

Huijausaineiston kerääminen tapahtui samoin tavoin kuin rehellisen aineiston kerääminen. Kirjoittaja pelasi itse ylläpitämällä pelipalvelimellaan deathmatch-pelimuotoa tietokonepelaajia vastaan, tosin nyt hyödyntämällä aimbottia. Aimbotin mahdollisti avoimen lähdekoodin huijausohjelmisto Osiris, jonka lähdekoodi löytyy GitHub-palvelusta (Krupiński 2023). Ohjelmisto injektoitiin peliin itse kehitetyllä DLL-injektorilla. Osiris mahdollisti aimbotin laajan konfiguroimisen, joten sille asetettin seuraavat säännöt

- Aimbotti tähtää vain näkyvissä oleviin kohteisiin, eli ei tähtää esim. seinän takana olevaan viholliseen.
- Aimbotti tähtää vain pelaajan näkökentällä oleviin kohteisiin.
- Aimbotti tähtää ja lukitsee eniten vahinkoa aiheuttavaan ruuminosaan kohteessa. Tämä käytännössä johti siihen, että valtaosa tapoista on pääosumia, tosin tätä tilastoa ei erikseen tallennettu.
- Aimbotti hallitsee aseen rekyylin automaattisesti.
- Aimbotti ei ota huomioon liikkumisen tuomaa tarkkuuden menetystä.

Osiris salli myös tähtäämisen nopeuden säätämisen. Täten puolet huijausainestosta kerättiin siten, että ohjelma tähtäsi välittömästi vihollista kohti, ja puolet maltillisemmalla, "ihmismäisemmällä"reaktionopeudella.

Aineistoa kertyi yhteensä 4108 alkiota. Aineiston tarkemman jakauman voi nähdä taulukosta 1.

Ase	Rehellinen	Huijattu
AK-47	1132	1258
AWP	247	249
Desert Eagle	166	165
M4A4	95	100
M4A1-S	75	89
SG 553	58	58
G3SG1	28	40
AUG	24	25
Negev	24	24
Bizon	22	27
UMP-45	22	22
SSG 08	17	17
P90	13	13
M249	12	12
CZ75-Auto	11	12
MP7	7	0
SCAR-20	6	0
Dual Elites	5	0
Galil	5	0
Nova	5	0
Glock	4	0
MAG-7	4	0
XM1014	4	0
Famas	3	0
MP9	3	0
P250	2	0
Tec-9	2	0
USP-S	1	0
Yhteensä	1997	2111

Taulukko 1. Aineiston asejakauma

4.2 Aineiston prosessointi

Koska meitä kiinnostaa tutkia katselukulman muutosta, laskettiin aineiston pitch- ja yaw-arvoille muutosarvot ΔP_n ja ΔY_n,

$$\Delta P_n = P_n - P_{n-1} \tag{4.1}$$

$$\Delta Y_n = Y_n - Y_{n-1} \tag{4.2}$$

missä P_n ja Y_n ovat pelaajan näkökulman pitch- ja yaw-arvot hetkellä n. Tässä kohtaa 128 pariarvon sekvenssi kutistui 127 pariarvon sekvenssiin. Seuraavaksi sekvenssistä tiputettiin viimeiset 32 alkiota pois parempien tuloksien toivossa, sillä sekvenssin viimeisellä neljännessekunnilla ei uskottu olevan enää merkittävää painoa aineistossa. Tässä kohtaa yksittäinen tappoinstanssi on kutistunut 95 pariarvon sekvenssiin.

Koska neuroverkot ovat alttiita aineiston piirteiden skaalautumiselle, aineisto skaalattiin lukujen -1 ja 1 välille. Peli sallii pitch-arvolle ääriarvot $[-89, 89]$ ja yaw-arvolle ääriarvot $[-180, 180]$. Täten muutosarvoille laskettiin skaalatut arvot ΔYs_n ja ΔPs_n seuraavilla kaavoilla.

$$\Delta Ps_n = \frac{\Delta P_n}{89} \tag{4.3}$$

$$\Delta Ys_n = \frac{\Delta Y_n}{180} \tag{4.4}$$

Lopuksi yksittäinen instanssi kasattiin vektoriin kaavion 4.5 mukaisesti.

$$[\Delta Ps_1, \Delta Ys_1, \Delta Ps_2, \Delta Ys_2, ..., \Delta Ps_{95}, \Delta Ys_{95}] \tag{4.5}$$

Koneoppimismenetelmän koulutusta varten aineisto jaettiin harjoitus- ja testi-osiin jaolla 75/25, eli aineistosta 75% käytettiin mallin koulutukseen ja 25% mallin verifioimiseen.

4.3 Koneoppimismalli

Tässä tutkielmassa käytettiin koneoppimismallina eteenpäin kytkettyä monikerroksista neuroverkkoa. Tämä toteutettiin Python-ohjelmointikielellä Scikit-learn-kirjaston avulla, tarkemmin MLPClassifier()-metodin avulla (“1.17. Neural network models (supervised)”, n.d.). Neuroverkon hyperparametrien valintaan hyödynnettiin k(10)-kertaista ristiinvalidointia Scikit-learning GridSearchCV-metodilla (“sklearn.model_selection.GridSearchCV”, n.d.), ja metodin tuloksen perusteella parhaat parametrit löytyvät taulukosta 2.

Hyperparametri	Valittu arvo
Ratkaisija	Adam
Alpha	0.0001
Aktivaatiofunktio	Relu
Piilokerrokset	(190, 95)
Max iter	20 000

Taulukko 2. Mallin hyperparametrit

4.4 Tuloksien validointi

Koneoppimismallin tarkkuuden validointiin käytetään luokitteluraporttia ja ROC-käyrää. Luokitteluraportti muodostetaan sekaannusmatriisin tunnusluvuista, ja sen avulla saamme tarkempaa tietoa mallin kyvystä luokitella tulokset oikein.

Sekaannusmatriisi koostuu neljästä tunnusluvusta

- Todelliset positiiviset, TP. Positiivinen tulos ennustettiin oikein positiiviseksi.
- Todelliset negatiiviset, TN. Negatiivinen tulos ennustettiin oikein negatiiviseksi.
- Väärä positiivinen, FP. Negatiivinen tulos ennustettiin virheellisesti positiiviseksi.
- Väärä negatiivinen, FN. Positiivinen tulos ennustettiin virheellisesti negatiiviseksi.

Luokitteluraporttiin muodostetaan näistä neljästä tunnusluvusta luokittain (rehellinen pelaaja ja huijaava pelaaja) kolme tunnuslukua, joiden avulla mallin oikeellisuutta validoidaan: Tarkkuus, saanti, ja F1-arvo.

	Ennustettu posiitivinen	Ennustettu negatiivinen
Oikea positiivinen	TP	FN
Oikea negatiivinen	FP	TN

Taulukko 3. Sekaannusmatriisin malli

Tarkkuus on luokitteluraportin yksinkertaisin tunnusluku. Sillä selviää oikeiden positiivisten osuus positiivisista ennusteista luokassaan

$$Tarkkuus = \frac{TP}{TP+FP} \tag{4.6}$$

Saanti on vuorostaan se osuus positiivisista ennusteista, jotka ennustettiin oikein

$$Saanti = \frac{TP}{TP+FN} \tag{4.7}$$

Lopuksi F1-arvo on tarkkuuden ja saannin harmoninen keskiarvo. F1-arvo vastaa yksinkertaisuudessaan kysymykseen, kuinka suuri prosenttiosuus positiivisista ennusteista oli oikein.

$$F1-arvo = 2 * \frac{(tarkkuus * saanti)}{(tarkkuus + saanti)} \tag{4.8}$$

Mallin suorituskykyä voidaan visualisoidaan myös ROC-käyrällä (engl. *Receiver Operating Characteristic-curve*), joka kertoo mallin suorituskyvystä luokittajan eri kynnysarvoilla. ROC-käyrän varten pitää vielä määritellä yksi uusi tilasto, False Positive Rate, eli FPR. FPR kuvaa osuutta todellisista negatiivisista veikkauksista, jotka malli ennusti virheellisesti positiivisiksi.

$$FPR = \frac{FP}{FP+TN} \tag{4.9}$$

ROC-käyrä piirretään asettamalla saanti kuvaajan y-akselille ja FPR kuvaajan x-akselille. Jokaiselle kynnysarvolle lasketaan saanti ja FPR, ja näistä pisteistä muodostetaan kuvaajaan käyrä. Täydellisen mallin tapauksessa (kaikki todelliset positiiviset tapaukset tunnistetaan oikein, eikä yhtään todellista negatiivista tapausta luokiteltu virheellisesti positiiviseksi,

saanti = 1, FPR = 0) käyrä kulkisi kuvaajan vasemman yläkulman kautta. Täysin satunnaisen mallin tapauksessa käyrä kulkisi suoraan vasemmasta alakulmasta oikeaan yläkulmaan. Todellisuudessa käyrä sijoittuu näiden kahden ääripään välimaastoon.

ROC-käyrän tuloksen voi arvioida myös numeerisesti AUC:lla, *Area Under the Curve*. AUC on ROC-käyrän alle jäävä pinta-ala, joka kertoo mallin suorituskyvyn. AUC:n arvo on välillä 1-0, jossa täydellisen mallin AUC olisi 1, ja täysin satunnaisen mallin AUC olisi 0.5.

5 Tulokset

Tässä luvussa esitellään tutkielmassa luodun mallin suorituskykyä aikaisempien lukujen metriikoiden perusteella.

Testausaineistossa oli mukana 513 rehellistä tappoa, ja 514 huijauskeinoin saatua tappoa. Sekaannusmatriisista 4 näemme mallin ennusteen. Malli ennusti todellisia positiivisia, eli rehellisiä tappoja 447. Vääriä negatiivisia, eli huijatuiksi luokiteltuja rehellisiä tappoja oli 66. Vääriä positiivisia, eli rehelliseksi luokiteltuja huijattuja tappoja oli 116. Todellisia negatiivisia, eli huijatuksi tulkittuja huijattuja tappoja oli 398.

	Ennustettu rehellinen pelaaja	Ennustettu huijaava pelaaja
Oikea rehellinen pelaaja	447	66
Oikea huijaava pelaaja	116	398

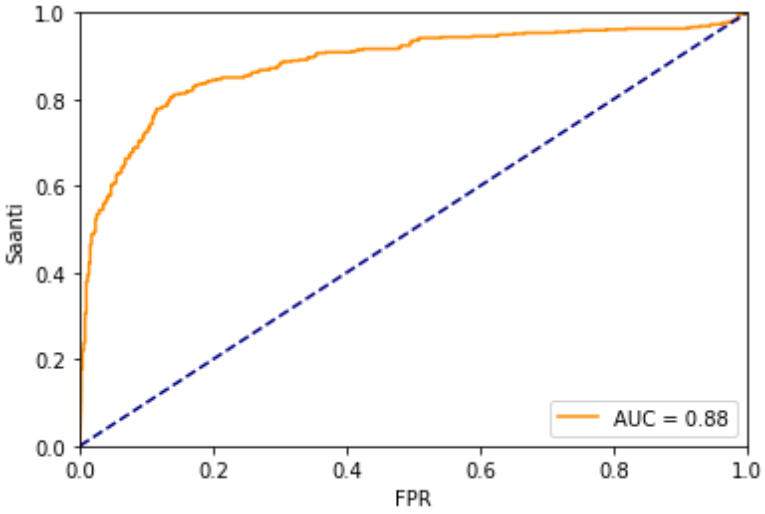
Taulukko 4. Mallin tuloksien sekaannusmatriisi

Näiden lukujen perusteella laskettiin tuloksille myös luokitteluraportti. Tarkkuuksiksi tuli oikeilla pelaajilla 0.79, ja huijaavalla pelaajalla 0.86. Saanti on oikealla pelaajalla 0.87, ja huijaavalla pelaajalla 0.77. Näiden kahden luokan välillä pienin ero on F1-arvossa, oikeilla pelaajille 0.83, ja huijaavalla pelaajalla 0.81.

Luokka	Tarkkuus	Saanti	F1-arvo	N
Oikea pelaaja	0.79	0.87	0.83	513
Huijaava pelaaja	0.86	0.77	0.81	514
Tarkkuus			0.82	1027
macro avg	0.82	0.82	0.82	1027
weighted avg	0.82	0.82	0.82	1027

Taulukko 5. Mallin tuloksien luokitteluraportti

Mallin ROC-käyrän voi nähdä kuvasta 7. Käyrän AUC on 0.88.



Kuvio 7. ROC-käyrä

6 Pohdinta

Tässä luvussa vastataan tutkielman tuloksen perusteella tutkimuskysymykseen, ja verrataan tuloksia aikaisempiin tutkimuksiin. Tämän jälkeen pohditaan tuloksiin vaikuttaneita seikkoja, tuloksien arvoa, ja mahdollisia kehityskohteita tuloksien parantamiseen.

6.1 Tutkimuskysymys

Tämän tutkielman tavoitteena oli selvittää, kuinka luotettavasti yksinkertaisella neuroverkolla voi havaita aimbotin käyttöä tulkitsemalla pelkästään pelaajan näkökulman dataa tappojen ympärillä yksinkertaista keinotekoista neuroverkkoa hyödyntäen. Aikaisemman luvun tulokset antavat vahvaa viitettä teorian potentiaalista; Malli antaa oikean ennusteen noin 80 % ajasta. Luku on lupaava, mutta siinä on vielä parantamisen varaa. Pelin kontekstissa, jossa yhden kilpailullisen erän aikana pelaaja voi kerätä kymmeniä tappoja, nousee virhetuomion riski huomattavan suureksi.

6.2 Vertailu aikaisempaan tutkimukseen

Luvussa 2 esiteltyihin aikaisempiin tutkimuksiin verrattuna, tämä tutkielma saavutti heikomman tuloksen. Jokainen mainittu tutkimus saavutti yli 90%:n tarkkuuden, josta tämä tutkielma jäi hieman vajaaksi. Suoraa vertailua muihin tutkimuksiin on vaikeahko tehdä, sillä jokaisessa aineiston luonne ja käytetyt menetelmät poikkeavat toisistaan. Mainituista tutkimuksista Pinto, Pimenta ja Novais 2021 myötäili kuitenkin eniten tätä tutkielmaa menetelmiensä osalta. Kerätty data hiiren liikkeestä oli käytännössä sama asia kuin tähän tutkielmaan käytetty data, tosin kerätty eri lähteestä.

Eräs konkreettinen ero muihin mainittuihin tutkimuksiin on se, miten tulkittava ainesto muodostettiin. Usea tutkimus kokosi data-alkion, jolla pyrittiin lähinnä vastaamaan kysymykseen "Käytettiinkö tämän ajanjakson aikana huijausohjelmaa". Tämä alkio saattoi olla esimerkiksi kokoelma prosessoituja tilastollisia tunnuslukuja tietyltä aikaväliltä, tai pitkä aikajakso syötteitä, kun taas tämän tutkielman tavoitteena oli selvittää, onko yksittäinen tappotapahtuma

saavutettu huijausohjelmia hyödyntäen vai ei.

6.3 Muita mietteitä

Tutkielmaa voi hyödyntää tulevissa tutkimuksissa, joissa käsitellään erityisesti aimbottien havaitsemista koneoppimismenetelmin. Tutkielma jättää myös taakseen avointa lähdekoodia SourceMod-lisäosan muodossa, jota yhteisö voi hyödyntää tai jatkokehittää omien aineistojen keräämistä varten. Tutkielma toimii hyvin myös ohjenuorana vastaavien omien aineistojen käsittelyyn ja mallien opettamiseen. Tutkimuksessa tuotettua mallia voisi myös konkreettisesti hyödyntää palvelimella jatkuvasti pyörivänä huijauksenhavaitsemisjärjestelmänä. Pienellä jatkokehityksellä olisi mahdollista saada SourceMod-lisäosa ja malli puhumaan keskenään, ja tuloksia voisi täten mahdollisesti käyttää esimerkiksi online-turnauksissa työkaluna kilpailullisen rehellisyyden varmistamiseksi.

Tämä tutkielma sai alkunsa kirjoittajan kokemuksesta kohdepelistä ja tiedosta siitä, miten pelin kehittäjät yrittävät ratkaista pelin huijausongelmaa. Tutkielman menetelmä otti inspiraationsa kohdepelin syväoppimista hyödyntävä havaitsemijärjestelmä VacNetistä. Vacnetin syöteavaruus koostuu "atomeista". Yksittäinen atomi vastasi aseen yksittäisiä laukauksia, ja se sisältää tiedon pelaajan näkökulman muutoksesta, käytetystä aseesta, laukauksen tuloksesta (ohi, osuma, pääosuma) ja etäisyydestä kohteeseen jos osuttiin. Mallin syöte oli täten 140 kuvatunkaltaista atomia, ja mallin tehtävä oli näiden tietojen perusteella päätellä käyttikö pelaaja huijausohjelmia vai ei (McDonald 2018). Tässä tutkielmassa metodi yksinkertaistettiin ja suoraviivaistettiin tutkimaan pelkästään näkökulman muutoksia tappojen ympärillä yksittäisten laukauksien sijasta.

Tutkielmassa on muutamat asiat mitkä tekisin toisin. Aloitetaan aineiston keräämiseen kehitetystä SourceMod-lisäosasta. Lisäosan olisi sopivampi laskea liikkeen muutosarvot suoraan, sillä aineiston jälkiprosessointi söi aineistosta yhden arvoparin verran dataa jokaista alkoita kohti. Lisäksi lisäosa ei nykyisellä toteutuksellaan välttämättä sovellu laajempaan käyttöön, sillä aineiston nauhoitus saattaa mennä sekaisin, mikäli pelisession aikana pelaajia poistuu ja liittyy takaisin palvelimelle.

Toiseksi, koneoppimismenetelmää hyödyntävien menetelmien ja tätä myötä myös aihetta

tutkivien tutkielmien heikkous tulee aina olemaan käytetty aineisto. Tämä tutkielma ei ole tämän suhteen mikään poikkeus. Aineistoa olisi voinut kerätä aktiivisemmin ja paljon enemmän. Vaikka aineistoa saatiin kuitenkin melkoisen laajalta kirjolta, aineistossa ei ole esimerkiksi ammattilaistason pelaajalta kerättyä dataa. Tällä datalla olisi ollut todennäköisesti suurin mahdollisuus kasvattaa väärien positiivisten määrää. Toinen silmäänpistävä heikkous aineistossa on sen asejakaumassa; puolet aineistosta on yhdellä aseella, AK-47. Tämä ase on normaalin kilpailullisen erän aikana saatavilla vain toisella joukkueella. AK-47 on tosin myös yksi pelin suosituimmista aseista, joten vinouma ei välttämättä ole niin kriittinen. Lisäsyynä vinoumaan myös se, että pelaajat aloittivat oletuksena kyseisellä aseella. Syy miksi olisin kaivannut enemmän vaihtelua aineistoon tämänkin suhteen on se, että pelin arsenaalia käytetään hyvin erilaisesti. Tarkkuuskivääreitä (esim. AWP, Scout) käytetään passiivisemmissa rooleissa; niillä pidetään kulmia ja käytäviä kiinni, eli niillä katselukulman liike on todennäköisesti erittäin staattista, tosin erittäin nopeilla liikkeillä kohdetta kohti. Pelin automaattiaseilla pelattaisiin enemmän aktiivista roolia, ja niillä ammutaan yleensä sarjaa vihollista kohti. Koska pelissä on jokaisella aseella uniikki rekyylikuvio sarjatulella ampuessa, tieto käytetystä aseesta olisi tuonut arvokasta lisätietoa malliin. Vielä yksi aineiston heikkous on huijausaineistossa, ja koska se kerättiin vain yhden henkilön toimesta. Huijausohjelmia käytetään varmasti laajemmalla kirjolla kuin tässä aineistossa, eikä kirjoittajalla ollut kokemusta huijausohjelman käytöstä. Tätä tosin yritettiin lievittää keräämällä huijausaineisto kaksilla eri säännöillä.

Tutkimusta voisi näiden huomioden lisäksi jatkaa hyödyntämällä muita neuroverkkokonfiguraatiota tai koneoppimismenetelmiä. Esimerkiksi takaisinkytkettyjen neuroverkkojen ikkunointimekanismi vaikuttaisi lupaavalta ominaisuudelta vielä tarkempien ennusteiden saamiseen, sillä yhden tapon aikana tapahtuneessa liikeradassa kaikki syötteet eivät välttämättä ole huijausohjelman tekemisiä.

7 Yhteenveto

Tässä tutkielmassa keskityttiin aimbotin käytön havaitsemiseen suositussa FPS-pelissä yksinkertaisen neuroverkon avulla, hyödyntäen suoraan pelipalvelimelta kerättyä dataa. Tutkielman teoreettinen osa alkoi määrittelemällä mitä huijaaminen tarkoittaa pelien kontekstissa, josta jatkettiin relevantin peleissä käytetyn verkko-arkkitehtuurin avaamisella. Tämän jälkeen esiteltiin tutkimuksessa käytettyä peliä, sekä vastaavaa aikaisempaa tutkimusta. Seuraavaksi avattiin keinotekoisen neuroverkon konseptia, toimintaperiaatteita sekä koulutukseen liittyviä haasteita. Tutkielman empiiristä tutkimusta varten luotiin ensin SourceMod-lisäosa, joka mahdollisti käytettävän aineiston keräämisen pelisessioilta. Rehellisen aineiston keruuseen osallistui 12 henkilöä, ja huijausaineisto kerättiin oikeaa huijausohjelmaa hyödyntäen kirjoittajan toimesta. Aineiston perusteella koulutettiin yksinkertainen eteenpäinkytketty neuroverkko, jonka suorituskykyä arvosteltiin eri menetelmin. Tutkimuskysymys oli täten seuraava; kuinka luotettavasti aimbotin käyttöä FPS-pelissä voidaan havaita tutkimalla pelaajan näkökulman liikettä tappojen yhteydessä hyödyntäen yksinkertaista keinotekoista neuroverkkoa?

Tutkielmassa tuotettu keinotekoinen neuroverkko saavutti noin 82 % tarkkuuden. Vaikka tulos on kaukana täydellisestä, antaa se viitettä siitä, että metodi on käyttökelpoinen. Tarkkoja juurisyitä mallin suorituskykyyn on vaikea päätellä. Koska aineiston data on aikasarjassa, paremman suorityskyvyn malli oltaisiin todennäköisesti saatu aikaan kehittyneimmillä neuroverkkomalleilla, kuten takaisinkytketyillä neuroverkoilla tai konvoluutioneuroverkoilla.

Loppujen lopuksi tutkielmalla tuettiin ja edistettiin aiheesta jo löytyvää tutkimusta. Tämä tutkielma jättää jälkeensä avointa lähdekoodia aineistojen keräämiseen vastaavia tutkimuksia varten kohdepelissä. Samalla tutkielma toimii hyvin ohjenuorana aineiston soveltamiseen keinotekoisia neuroverkkoja varten, ja tarjoaa näkökulmia miten mallia voitaisiin luoda ja miten sen suorituskykyä voitaisiin mahdollisesti parantaa.

Lähteet

"1.17. Neural network models (supervised)". Scikit-learn. n.d. Viitattu 7. syyskuuta 2023. https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

Alayed, Hashem, Fotos Frangoudes ja Clifford Neuman. 2013. "Behavioral-based cheating detection in online first person shooters using machine learning techniques". Teoksessa *2013 IEEE Conference on Computational Inteligence in Games (CIG),* 1–8. ISSN: 2325-4289. Elokuu. https://doi.org/10.1109/CIG.2013.6633617.

Chen, Vivian Hsueh Hua, ja Jeremy Ong. 2018. "The rationalization process of online game cheating behaviors". Publisher: Routledge _eprint: https://doi.org/10.1080/1369118X.2016.1271898, *Information, Communication & Society* 21, numero 2 (1. helmikuuta 2018): 273–287. ISSN: 1369-118X, viitattu 5. kesäkuuta 2023. https://doi.org/10.1080/1369118X.2016.1271898. https://doi.org/10.1080/1369118X.2016.1271898.

Galli, L., D. Loiacono, L. Cardamone ja P. L. Lanzi. 2011. "A cheating detection framework for Unreal Tournament III: A machine learning approach". Teoksessa *2011 IEEE Conference on Computational Intelligence and Games (CIG'11),* 266–272. ISSN: 2325-4289. Elokuu. https://doi.org/10.1109/CIG.2011.6032016.

Goodfellow, Ian, Yoshua Bengio ja Aaron Courville. 2016. *Deep Learning*. Http://www.deeplearningbook.org. MIT Press.

Hursti, Mikko, Joonas Puuppo ja Ilkka Pölönen. n.d. "Tekoälyä valmistavaan teollisuuteen -verkkokurssi". Yhteistyössä Anna-Maria Raita-Hakola, Kimmo Riihiaho ja Leevi Lind. Viitattu 6. syyskuuta 2023. https://tim.jyu.fi/view/kurssit/tie/aiaddva/kurssi/mainos.

Jordan, M. I., ja T. M. Mitchell. 2015. "Machine learning: Trends, perspectives, and prospects". *Science* 349, numero 6245 (17. heinäkuuta 2015): 255–260. ISSN: 0036-8075, 1095-9203, viitattu 5. kesäkuuta 2023. https://doi.org/10.1126/science.aaa8415. https://www.science.org/doi/10.1126/science.aaa8415.

Kahilainen, Niko. 2023. *Sourcemod viewangle logger.* Viitattu 5. syyskuuta 2023. https://github.com/Kahiis/Sourcemod-viewangle-logger.

Kotkov, Denis, Gaurav Pandey ja Alexander Semenov. 2018. “Gaming Bot Detection: A Systematic Literature Review”. Teoksessa *Computational Data and Social Networks,* toimittanut Xuemin Chen, Arunabha Sen, Wei Wayne Li ja My T. Thai, 11280:247–258. Cham: Springer International Publishing. ISBN: 978-3-030-04647-7 978-3-030-04648-4, viitattu 17. helmikuuta 2019. https://doi.org/10.1007/978-3-030-04648-4_21. http://link.springer.com/10.1007/978-3-030-04648-4_21.

Krupiński, Daniel. 2023. *Osiris.* Viitattu 5. syyskuuta 2023. https://github.com/danielkrupinski/Osiris.

Lehtonen, Samuli Johannes. 2020. “Comparative Study of Anti-cheat Methods in Video Games”. Tutkielma, Helsingin yliopisto.

Lindholm, Andreas, Niklas Wahlström, Fredrik Lindsten ja Thomas B. Schön. 2022. *Machine Learning - A First Course for Engineers and Scientists.* Cambridge University Press. https://smlbook.org.

McDonald, John. 2018. “Robocalypse Now: Using Deep Learning to Combat Cheating in ’Counter-Strike: Global Offensive’”. Viitattu 9. joulukuuta 2020. https://www.gdcvault.com/play/1024994/Robocalypse-Now-Using-Deep-Learning.

McSherry, Corynne. 2005. “A New Gaming Feature: Spyware”. Electronic Frontier Foundation, 20. lokakuuta 2005. Viitattu 14. kesäkuuta 2023. https://www.eff.org/deeplinks/2005/10/new-gaming-feature-spyware.

Orland, Kyle. 2020. “Ring 0 of fire: Does Riot Games’ new anti-cheat measure go too far?” Ars Technica, 14. huhtikuuta 2020. Viitattu 14. kesäkuuta 2023. https://arstechnica.com/gaming/2020/04/ring-0-of-fire-does-riot-games-new-anti-cheat-measure-go-too-far/.

Pinto, José Pedro, André Pimenta ja Paulo Novais. 2021. “Deep learning and multivariate time series for cheat detection in video games” [kielellä en]. *Machine Learning* 110, numero 11 (joulukuu): 3037–3057. ISSN: 1573-0565, viitattu 21. huhtikuuta 2022. https://doi.org/10.1007/s10994-021-06055-x. https://doi.org/10.1007/s10994-021-06055-x.

“sklearn.model_selection.GridSearchCV”. Scikit-learn. n.d. Viitattu 6. syyskuuta 2023. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

“Source Multiplayer Networking - Valve Developer Community”. n.d. Viitattu 5. kesäkuuta 2023. https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.

Tuominen, Heli, Pekka Neittaanmäki, Esko Niinimäki, Ilkka Pölönen, Ilkka Rautiainen, Sami Äyrämö, Toni Ruohonen ym. 2019. *Tekoälyn perusteita ja sovelluksia.* Accepted: 2019-07-03T12:15:07Z. ISBN: 978-951-39-7796-2, viitattu 26. elokuuta 2023. https://jyx.jyu.fi/handle/123456789/64975.

Yu, Su-Yang, Nils Hammerla, Jeff Yan ja Peter Andras. 2012. “A statistical aimbot detection method for online FPS games”. Teoksessa *The 2012 International Joint Conference on Neural Networks (IJCNN),* 1–8. ISSN: 2161-4407. Kesäkuu. https://doi.org/10.1109/IJCNN.2012.6252489.