Tampereen yliopisto

Tobias Halfar

EVOLUTIIVINEN NEUROVERKKOARKKITEHTUURIHAKU

Geneettisen algoritmin käyttö neuroverkon rakenteen ja hyperparametrien optimoinnissa

Master's Thesis
Department of Signal Processing, Tampere University
Tarkastaja: Prof. Tapio Elomaa
Heinäkuu 2022

TIIVISTELMÄ

Tobias Halfar: Evolutiivinen neuroverkkoarkkitehtuurihaku
Master's Thesis
Tampereen yliopisto
MSc (Tech), Electrical Engineering
Heinäkuu 2022

Tämä diplomityö käsittelee geneettisellä algoritmilla toteutettua neuroverkkoarkkitehtuurihakua. Diplomityön tarkoitus on luoda katsaus evolutääristen algoritmien toimintaan, sekä käyttää geneettistä algoritmia neuroverkkoarkkitehtuurin optimoimiseen. Työssä käsitellään esitetyn algoritmin toimivuutta kolmessa erillisessä tapauksessa. Näissä tavoitteena on optimoida sellainen neuroverkkoarkkitehtuuri joka kykenee pienikokoisista kuvista määrittämään, montako ihmistä kussakin kuvassa on.

Diplomityö rakentuu allekirjoittaneen TTY:n kaggle kisoja varten luotuun geneettisen algoritmin laajennukseen, johon ideoita on saatu alan oppikirjoista ja artikkeleista. Tutkimushypoteesina on, että elitistiseen valintaan perustuvaa geneettistä algoritmia, jossa on useita ajoja satunnaisilla aloituspopulaatioilla ja edellisten ajojen populaatiot käytössä risteytyksessä, saadaan täysin satunnaista neuroverkkoarkkitehtuurihakua parempia tuloksia. Tämä useiden ajojen satunnaisten aloituspopulaatioiden tarkoitus oli lisätä kokonaispopulaation diversiteettiä ja estää käytetyn elitistisen geneettisen algoritmin liian nopeaa konvergoitumista yhden ratkaisun ympärille.

Tulokset tukivat hypoteesiä. Geneettisellä algoritmilla toteutettu neuroverkkoarkkitehtuurihaku antoi kaikissa tapauksissa satunnaista neuroverkkoarkkitehtuurihakua parempia tuloksia.

Avainsanat: Evolutääriset algoritmit, Geneettinen algoritmi, Neuroverkkoarkkitehtuurihaku, Konvoluutioneuroverkko

ABSTRACT

Tobias Halfar: Evolutionary neural architecture search
Thesis type
Tampere University
Degree Programme
July 2022

This master thesis does cover a neural architecture search using genetic algorithm. Main hypothesis is that it is possible to implement a neural architecture search using elitist selection in genetic algorithm that have many runs that shares their population gives better results than fully randomized neural architecture search. This algorithm is used in three cases where the goal was to count the number of people in pictures using convolutional neural network and search for each case the best neural network architecture. Results did support the hypothesis.

Keywords: Evolutionary algorithms, Genetic algorithm, Neural architecture search, Convolutional neural network

ALKUSANAT

Haluan kiittää professori **Tapio Elomaata** kärsivällisestä ohjauksesta ja aikataulupaineiden ymmärtämisestä. Kiitän professori **Joni Kämäräistä** lämpimän humaanista tukemisesta opiskelijoita kohtaan ja antoisista keskusteluhetkistä, sekä ystävällisestä kannustuksesta uran aloitukseen.

Tampereella, 31. heinäkuuta 2022

Tobias Halfar

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

EA	Evolutiivinen Algoritmi
EO	Evolutiivinen Ohjelmointi
ES	Evolutiiviset strategiat
GA	Geneettinen Algoritmi
NAH	Neuroverkkoarkkitehtuurihaku

1. JOHDANTO

Tämä opinnäytetyö käsittelee evolutiivisella algoritmilla toteutettua neuroverkkoarkkitehtuurihakua. Tarpeena oli kouluttaa konvoluutioneuroverkko joka pystyisi mahdollisimman hyvin määrittämään kuvista montako ihmistä on eri kokoisten koppien sisällä. Aineiston ollessa suhteellisen pieni ja etenkin kuvien koon ollessa pieni, on mahdollista toteuttaa geneettisellä algoritmillä toimiva neuroverkkoarkkitehtuurihaku. Opinnäytetyö koostuu kuudesta luvusta johdannon lisäksi.

Luvussa *evolutääriset algoritmit* tehdään nopea yleiskatsaus eri evolutäärisiin algoritmeihin ja niiden historiaan, sekä käyttökohteisiin. Erityisesti keskitytään käsittelemään tarkemmin geneettisten algoritmien yleisiä toimintaperiaatteita. Luvussa *Evolutiiviset neuroverkkoarkkitehtuurihaut* käydään yleisesti lävitse eri neuroverkkoarkkitehtuurihakuja, sekä nopea katsaus muutamaan yleisimmin käytettyyn neuroverkkoarkkitehtuuriin. Luvussa *toteutettu evolutiivinen neuroverkkoarkkitehtuurihaku* esitellään tässä opinnäytetyössää toteutettua evolutiivisen neuroverkkoarkkitehtuurihaun toimintaa. Pohjana oli TTYn Kagglen koneoppimiskisoissa menestyksellisesti toteutettu geneettiseen algoritmiin perustuva tukivektorikoneen hyperparametrien optimointi [1]. Tätä geneettistä algoritmia laajennettiin toteuttamaan neuroverkkoarkkitehtuurihaku konvoluutioneuroverkoille. Toteututussa algoritmissa erikoispiirteenä oli populaation diversiteetin ylläpito käyttämällä monia peräkkäisiä ajoja satunnaisilla aloituspopulaatioilla. Näin pyrittiin tasapainottamaan elitististä valintamekanismia. Käytetty algoritmi myös ei käyttänyt yksilöiden poistamista populaatiosta, vaan puolet kunkin jälkeläisen geenistä otettiin kaikkien aiempien yksilöiden muodostamasta populaatiosta. Toinen puoli uusien yksiöiden geeneistä otettiin elitistisellä valinnalla sen hetkisestä sukupolvesta. Luvussa *tulokset* käydään lävitse kolme erillaista tapausta joihin evolutiivista neuroverkkoarkkitehtuurihakua sovellettiin. Tapaukset koostuivat erikokoisista kopeista, joihin kuhunkin mahtui 0-1, 0-4 tai 0-6 ihmistä. Toteutetun evolutiivisen neuroverkkoarkkitehtuurihaun tuloksia verrattiin satunnaisesti toteutettuun neuroverkkoarkkitehtuurihakuun. Luvussa *Yhteenveto* tiivistetään opinnäytetyön tulokset.

Opinnäytetyön hypoteesina on, että käyttämällä elitististä valintaa ja monta peräkkäistä ajoa jotka kukin käyttävät aikaisempien ajojen tuloksia satunnaisen aloituspopulaation jälkeen, saadaan täysin satunnaista neuroverkkoarkkitehtuurihakua parempia tuloksia. Tulokset tukevat tätä hypoteesia. Etenkin 0-1 kopin tapauksessa, missä käytössä oli eniten

laskuaikaa, joka mahdollisti suurimman populaation käytön.

2. EVOLUTÄÄRISET ALGORITMIT

Evolutäärinen algoritmi(EA, evolutionary algorithms) on optimointialgoritmi jossa matkitaan biologian evoluutiota optimointiongelmien ratkaisemiseksi. Yhteistä evolutiivisille algoritmeille on populaatio joka koostuu yksilöistä, joilla kullakin on oma *kelpoisuutensa* (fitness) joka määritellään optimoitavan ongelman suhteen. Kaikki yksilöt esittävät eri mahdollisia ratkaisuja annetulle optimointiongelmalle. Yksilöt jotka saavuttavat korkeamman kelpoisuusarvon valitaan muodostamaan uusia populaation yksilöitä rekombinaation ja/tai mutaation kautta. Tämän prosessin kautta populaation keskimääräinen kelpoisuus kasvaa algoritmin iteraatioiden myötä suuremmaksi. Tätä iteratiivista prosessia jatketaan tarpeen mukaisesti, kunnes ollaan saavutettu halutun kelpoisuuden omaava ratkaisu. Kuvattu prosessi matkii evoluutiota ja sen keskeisimmät ajurit ovat variontioperaattorit (mutaatio ja rekombinaatio) ja valinta (mitkä yksilöt otetaan variontioperaattorin käsittelyyn).

EA:t ovat olleet tietojenkäsittelyn alkuajoista lähtien tutkimuksen kohteena. Siksi näille algoritmeille on luontaisesti kehittynyt monta eri jaottelua. Alla esitetään kolme EA paradigmaa jotka olivat alan kehityksen alusta lähtien olleet tutkimuksen kohteena. Näitä paradigmoja yhdistää mutaatio variontioperaattorina, sekä valinta joka ajaa populaatiota kohti parempaa ratkaisua. Myöhemmin on todettu näiden lähestymistapojen olevan erityyppisiä esityksiä pohjimmiltaan samoista EA:sta. Käytännön sovelluksissa on tärkeää valita sellainen EA paradigma joka sopii parhaiten annettuun ongelmaan. [2] [3]

2.1 Evolutiivinen ohjelmointi

Evolutiivinen ohjelmointi (EO) (evolutionary programming) on paradigma jossa käytetään evoluutiota *äärellisten automaattien (ÄA)* (finite state machine) optimointiin. Tässä lähestymistavassa uusi sukupolvi luodaan mutatoimalla jokaista edellisen sukupolven yksilöä. Vaihtoehtoisesti uusi sukupolvi luodaan rekombinaation kautta. Suuret simulointiajot ovat osoittaneet sopivan rekombinaatio-operaattorin kehittämisen olevan vaikea toteuttaa. Tällä menetelmällä on saavutettu silti hyviä tuloksia pelkästään mutaatio-operaattoria käyttäen. Yhteistä näille on, että sukupolvi jonka yksilöiden määrä on N, kaikki yksilöt muodostavat uuden yksilön mutaatio-operaattoria käyttäen. Näin syntyneestä $2N$ yksilöstä, jossa myös edellisen sukupolven yksilöt mukana, valitaan N kappaletta parhaan kelpoisuusarvon omaavaa yksilöä muodostamaan seuraava sukupolvi. [4]

2.2 Evolutiiviset strategiat

Evolutiiviset Strategiat (ES) (evolutionary strategies) paradigmassa yksilöä esittää N:n alkion pituinen reaaliarvoinen vektori, joka esittää ratkaisua optimointiongelmaan. Yksinkertainen esimerkki ES strategiasta on $ES(1 + \lambda)$ jossa yhdestä yksilöstä mutataatio-operaattorilla tuotetaan λ kappaletta jälkeläisiä. Näistä $1 + \lambda$ yksilöstä jossa on vanhempi mukana valitaan parhaan kelpoisuuden omaava yksilö muodostamaan uuden sukupolven vanhempi. ES mutaatio-operaattorin valintana on yleensä normaalijakauma $\mathcal{N}(0, \sigma)$, joka summataan mutatoitavan yksilöä kuvaavan vektorin eri indekseihin. ES kehittämisen alusta lähtien huomattiin evoluution edistymisen riippuvan hyvin herkästi mutaatio-operaattorin σ valinnasta. Tämä johti ratkaisuun, jossa yksilöä kuvataan $2N$ pituisella reaaaliarvoisella vektorilla. Tässä esityksessä $N + 1, \cdots, 2N$ indeksit vektorista kuvaavat alkupään indeksien $1, \cdots, N$ mutaatio-operaattorin σ arvoa kullekin indeksille. Näiden σ parametrien säätämiseen liittyy nimellä "1:5"metaheuristiikka. Jos yksittäisen indeksin mutaatio-operaattori $G(0, \sigma_i)$ tuottaa yli 20% mutaatioista vanhempaa parempia kelpoisuuksia, pitää kyseisen σ_i arvoa kasvattaa. Jos taas mutaation tuottamista jälkeläisistä alle 20% tuottaa vanhempaa parempia kelpoisuuksia, täytyy kyseistä σ_i parametria pienentää. $ES(\mu+\lambda)$ strategiassa sukupolven kelpoisuudeltaan μ parasta valitaan muodostamaan seuraava sukupolvi siten, että kustakin yksilöstä luodaan mutaatio-operaattorilla λ uutta jälkeläistä. [5]

2.3 Evolutiivisten algoritmien historiaa

Ensimmäisiä artikkeleita joissa evoluutiota käsiteltiin optimointiongelmana oli Sewall Wrightin kirjoittama [6]. Artikkelissa esitettiin esimerkkinä yksinkertaistuksena evoluutio kolmiulotteisena optimointiongelmana, jossa kaksi akselia kuvaavat eri mahdollisuuksia yhdistää geenit ja kolmas kuvaa kelpoisuutta. Keskeistä oli kuvata evoluutiota optimointiongelmana jolla on monta erillään olevaa kelpoisuushuippua joiden ympärille lajit muodostavat klusteriryppäitä. Artikkelissa käsiteltiin Mendelin perinnöllisyysteoriaa tilastollisin menetelmin. Optimointiongelmissa usein toistuva kelpoisuushuippujen välissä olevat alhaisen kelpoisuuden alueet ja niistä ylitse pääseminen oli esitetty artikkelissa. Artikkelin yleisenä tuloksena oli, että evoluutio riippuu seuraavista keskeisistä tekijöistä. Uusien populaatioon syntyvien geenien mutaatio on tärkeä evoluutiota eteenpäin ajava voima, joka pakottaa populaation kokeilemaan eri variaatioita jo löydetyistä ratkaisuista. Silti mutaatio ei saa olla liian suurta, koska se tuhoaisi aiempien geenien ratkaisut. Valinta ajaa populaatiota kohti parempia ratkaisuja, mutta liian suuri valintapaine tuhoaa populaation sisäisen variaation. Populaation sisäsiittoisuus, eli aiempien löydettyjen ratkaisujen yhdistely uusien ratkaisujen löytämiseksi on myös tärkeä tekijä. Liian suuri sisäsiittoisuus kuitenkin johtaa lajin sukupuuttoon. [6]

Tietojenkäsittelyn isänä pidetty Turing esitti 1948 artikkelissaan [7] evoluutiota matkivan

algoritmin käyttämistä älykkään tietokoneen rakentamisessa. Artikkelissa käsitellään epäorganisoidun tilakoneen opettamista evoluutiolla. Käytännössä artikkeli oli ensimmäinen esitys EO paradigman mahdollisuuksista. Artikkelin lopussa on myös lyhyt maininta GA toimintaperiaatteesta. [7]

Bremermann [8] vuonna 1962 esitti karkean arvion, minkä rajan tunnetut luonnonlait antavat laskennalle. Tässä artikkelissa arvioidaan energia-massa dualiteetin ja Heisenbergin epätarkkuusperiaatteen antavan laskennalle rajan mitä voidaan grammalla materiaa toteuttaa: $2 \cdot 10^{47} \frac{bit}{s \cdot g}$. Tämä *Bremermann rajana* tunnettu luku on suuri. Silti esimerkiksi 160 pituinen binääreistä koostuvan vektorin eri mahdollisia arvoja on $2^{160} \approx 1.5 \cdot 10^{48}$ verran. Tämä luonnonlakien antama yläraja laskennolle ja jo lyhyenkin binääriarvoista koostuvan vektorin kombinatorinen $O(2^n)$ räjähdys herättivät kysymyksen, miten luonto pystyy optimoimaan DNA:n eri arvoja. Bremermannin raja estää käytännössä yksinkertaisen haun kaikkien mahdollisten binäärikombinaatioiden lävitse. Luonnon vastaus tähän ongelmaan on evoluutio. Artikkelissa Bremermann esitti, että pelkästään yksittäisten binäärien muuttaminen suvuttomassa ilman rekombinaatio-operaattoria tapahtuvassa lisääntymisessä johtaa nopeasti stagnaatioon. Yksinkertaiset yksisoluiset organismit lisääntyvät suvuttomasti ja niissä vain mutaatio toimii variontioperaattorina. Kaikki monimutkaisemmat organismit käyttävät suvullista lisääntymistä. Suvullinen lisääntyminen lisää mutaation rinnalle rekombinaatio-operaattorin. Bremermann kokeili GA pohjaista menetelmää lineaaristen epäyhtälöiden optimointiongelmien ratkaisuun. Menetelmä toimi ja toimi lähtökohdalle GA paradigman kehittämiseen eteenpäin vaikeiden optimointiongelmien ratkaisemiseen. [9]

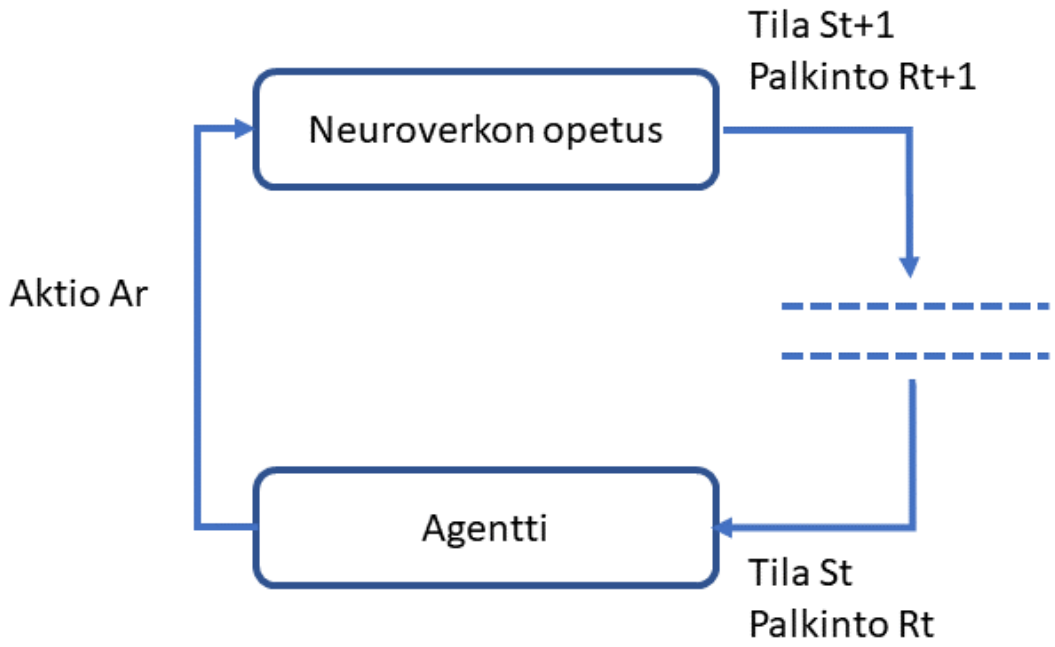
2.4 Neuroverkkoarkkitehtuurihaku

Neuroverkkoarkkitehtuurihaulla NAH (neural architecture search NAS) tarkoitetaan automatisoitunutta järjestelmällistä tapaa hakea käsillä olevaan tarpeeseen sopiva neuroverkon arkkitehtuuri ja sen hyperparatmetrit. Yleisimmät tavat toteuttaa NAH ovat vahvistusoppimiseen perustuva NAH, bayesilaiseen optimointiin perustuva NAH, sekä evoluution käyttöön perustuva NAH. Tämä opinnäytetyö käsittelee evolutiivisella algoritmilla toteutettua NAH:ia. [10]

Vahvistusoppiva NAH (reinforcement learning NAS) käyttää pohjalla vahvistusoppimista. Siinä tavoitefunktiona toimii haetun neuroverkon suorituskyky annettuun ongelmaan. Tässä lähestymistavassa usein käytetään pienempää osajoukkoa opetusaineistosta jolle haetaan optimi neuroverkkoarkkitehtuuri. Syy tälle, on suuremmalla opetusaineistolle tarvittava laskentateho. Jokaisen yksittäisen verkon vastavirta-algoritmin ajo vaatii paljon laskentatehtoa. Ensin haetaan samankaltaiseen, mutta pienempään opetusaineistoon sopiva neuroverkkoarkkitehtuuri. Tämä yleensä yleistyy myös suurempiin aineistoihin. Esimerkkinä voitaisiin pitää CIFAR-10 opetusaineistoon opetettua verkkoa, jonka ra-

kenne toimii myös suuremmalla COCO opetusaineistolla. [11], [12], [13]

Tässä lähestymistavassa yksi yleisesti käytetty metodi on Q-oppiminen.



Kuva 2.1. *Vahvistusoppimisen periaate*

Siinä agentin operaatioavaruus on yksittäisen löydetyn neuroverkkoarkkitehtuuria kuvaavat hyperparametrit. Nämä opetetaan erikseen käyttäen vastavirta-algoritmia ja saatuna palkintona pidetään kyseisen neuroverkon kelpoisuutta annetun ongelman suhteen. Jokainen koulutettu neuroverkko tallennetaan Q-tauluun. Tätä Q-taulun informaatiota hyväksikäyttäen agentti valitsee seuraavan siirron, käyttäen hyväksi kaikkia edellisten koulutettujen neuroverkkojen kelpoisuutta. Tämän metodin toimintatapa on esitetty kuvassa 2.1. Keskeinen rajoitus on yksittäisten neuroverkkojen koulutus joka vaatii laskentatehoa. Yksi tapa kiertää tämä rajoite, on käyttää jo koulutettujen neuroverkojen tasojen painoja uusia neuroverkkoarkkitehtuureja etsiessä. [14]

Baysilaisen optimoinnin NAH (bayesian optimization NAS) perustuu Baysesilaiseen mallintamiseen. Tässä lähestymistavassa oletuksena on, että eri neuroverkon hyperparametrien tulokset seuraavat gaussinjakaumaa. Aluksi määritetään hakuavaruus hyperparametrien suhteen. Tämän jälkeen valitaan lähtökohdaksi satunnaisesti eri hakuavaruuden arvoja, joille kullekin selvitetään neuroverkon haluttu hyvyysfunktion arvo. Kun hakuavaruudesta on saatu tarpeeksi iso määrä satunnaisia näytteitä hyvyysfunktion arvoineen, lähdetään hakemaan uusia alkioita käyttäen bayesin teoreemaa. [15]

Evolutäärinen NAH (evolutionary NAS) käyttää evolutiivista algoritmia sopivan neuroverkkoarkkitehtuurin ja hyperparametrien etsimiseen [16]. Tätä lähestymistapaa on käytetty tässä opinnäytetyössä ja sen tarkempi esitys on luvuissa 2.5 ja 4.2.1.

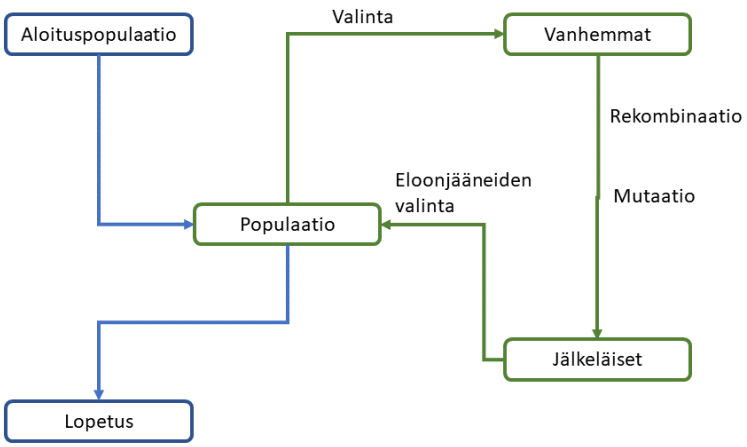
2.5 Geneettiset Algoritmit

Geneettiset algoritmit (GA) (genetic algorithms) paradigmassa yksilöä esitetään yleensä binääriarvoisella vektorilla. Tätä vektoria nimitetään geeniksi. Kuhunkin geeniin on binäärinä esitetty optimoitavan ongelman parametrien mahdollisia arvoja. Nämä arvot voivat olla esimerkiksi binääriesityksenä annettuja kokonaislukuja ja/tai liukulukuja, sekä mahdollisia valintoja äärellisestä määrästä etukäteen annettuja valintamahdollisuuksia. GA:n paradigmassa käsitellään erikseen geenien evoluutiota ja geenien esittämien ratkaisujen kelpoisuuden laskemista. GA:ssa geenejä kutsutaan *genotyypiksi* ja kyseisen genotyypin koodaamaa yksilöää *fenotyypiksi*. Nämä erotellaan toisistaan, koska GA:ssa evoluutio, *valinta*, *rekombinaatio* ja *mutaatio* kohdistuvat genotyyppiin. Fenotyyppi on genotyypistä yksiselitteisesti purettu esitys. [17]

Esimerkiksi luvun 42 binääriesitys on

$$42 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = \{1, 0, 1, 0, 1, 0\}$$

Tässä esimerkissä lukua 42 voidaan pitää fenotyyppinä ja binääriesitystä $\{1, 0, 1, 0, 1, 0\}$ vastaavana genotyyppinä.



Kuva 2.2. *Geneettisen algoritmin vuokaavio*

Kuvassa 2.2 on hahmoteltu vuokaavioina GA:n toimintaperiaatetta. Rekombinaatio yhdistää kaksi tai useampaa populaation yksilöä yhdeksi tai useammaksi jälkeläiseksi. Osaan jälkeläisistää kohdistetaan mutaatio-operaattori, joka tuottaa hieman muuttuneet genotyypin. Riippuen koodaustavasta nämä voivat olla fenotyyppeinä joko kaukana toisistaan tai hyvin samanlaisia. Rekombinaatiota ja mutaatio-operaattoria sanotaan GA:n *variaatio-operaattoreiksi*. Nämä tuottavat populaatioon uusia yksilöitä ja ylläpitävät popu-

laation diversiteettiä. Valintaoperaattori taas ajaa populaation keskimääräistä kelpoisuutta ylöspäin, määrittämällä mitkä yksilöt populaatiosta pääsevät lisääntymään. Huomattavaa on, että valintaoperaattori toimii fenotyyppien maailmassa jossa yksilöille on ongelman suhteen määrätty kelpoisuus. Rekombinaatio ja mutaatio-operaattori ovat yleensä genotyyppien maailmassa stokastisesti toimivia operaattoreita. [2]

Seuraavissa aliluvuissa on esitelty tarkemmin GA:n komponenttien toimintaa.

2.5.1 Kelpoisuus

Kelpoisuus on GA:n suorittaman optimoinnin tavoite. Kelpoisuutta usein myös kutsutaan *kelpoisuusfunktioksi*. Yleensä GA:n tavoitteena on joko minimoida tai maksimoida kelpoisuusfunktion arvo. Yksinkertainen esimerkki kelpoisuusfunktiosta on maksimoida funktion $f(x) = x^2$ arvoa annetulla fenotyyppiavaruudella. Tällöin yksi mahdollinen genotyyppi olisi kuuden binäärin pituinen geeni jolla esitetään kokonaislukuja. Aiemman esimerkin $42 = \{1, 0, 1, 0, 1, 0\}$ kelpoisuus olisi tällöin $f(42) = 42^2 = 1764$.

Kelpoisuuksia voi olla hyvinkin erillaisia. Tämän opinnäytetyön NAH:in on yksi kelpoisuudesta, joka on haettu ratkaistavan ongelman suhteen. NAH:illa haettujen neuroverkkojen tarkkuus:

$$tarkkuus = \frac{\#oikein}{\#kaikki}$$

jossa $\#oikein$ on neuroverkon oikein määrittämien kuvien lukumäärä $\#kaikki$en kuvien joukosta. Kelpoisuutta käytetään valinnan ja elonjääneiden valinnassa muodostettaessa uutta populaatiota. [2]

2.5.2 Populaatio

Populaatio pitää sisällään mahdollisia GA:n ratkaistavan ongelman ratkaisuja. Se muodostuu genotyypeistä, jotka ovat GA:n variontioperaattorien ja valintaoperaattorien tuottamia. Yksinkertaisissa GA:n malleissa populaatiota voidaan kuvata sen yksilöiden määrällä. Monimutkaisimmissa GA:issa populaatiota voidaan kuvata myös avaruudellisena rakenteena, jossa eri genotyyppien välisiä suhteita tai ryppäitä kuvataan tunnusluvuilla. Tällä karkeasti haetaan luonnon evoluutiossa esiintyviä eri lajien alatyyppejä vastaavaa esitystä. Suuressa osassa GA:n toteutuksissa populaation koko pidetään vakiona ja se ei muutu evoluution tuottamien uusien populaatioiden välillä. Valintaoperaattorit kohdistuvat populaatioon. Ne muodostavat edellisestä populaatiosta uusia yksilöitä ongelmaan soveltuvaa valintaoperaattoria käyttäen. Esimerkiksi valintaoperaattori voi valita edellisen populaation parhaat yksilöt lisääntymään. Huomattavaa on, että valintaoperaattori käsittelee populaatiota, kun taas variontioperaattorit operoivat yksittäisten genotyyppien tasolla.

Diversiteetti kuvaa paljonko erityyppisiä genotyyppejä populaatio sisältää. Diversiteetille ei ole yksittäistä yleiskelpoista määritystä, vaan se on suoraan GA sovelluksesta kiinni. Entropia on yksi mahdollinen valinta kuvaamaan populaation diversiteettiä.

Tässä opinnäytetyössä käsiteltyyn NAH eroaa normaalisti käytettyihin GA populaation koon suhteen. Tässä opinnäytetyössä käytetty NAH käyttää rekombinaatio-operaattorissa myös kaikkien aiemmin laskettujen yksilöiden kokonaispopulaatiota. Näin käsittelyssä olevasta populaation lisäksi myös kaikki edellisten sukupolvien yksilöillä on mahdollisuus tuottaa uusia yksilöitä. Diversiteettinä tässä opinnäytetyössä esitettyyn NAH:iin oli sukupolvien kelpoisuuksien keskijakauma.

2.5.3 Valinta

Valintaoperaattori valitsee mitkä yksilöt käsittelyn alla olevasta populaatiosta muodostavat uusia yksilöitä rekombinaatio-operaattoria käyttäen. Valintaoperaattori ajaa populaation keskimääräistä kelpoisuutta eteenpäin. Yleensä valinta on stokastinen operaattori.

Kelpoisuuteen verrannollinen valinta KVV (fitness proportional selection FPS) on suoraan kelpoisuuteen verrannollinen tapa valita populaation lisääntymään pääsevät yksilöt. Siinä yksittäisen yksilön valintatodennäköisyys on:

$$P_{KVV}(i) = \frac{f_i}{\Sigma_{j=1}^{\mu} f_j}$$

jossa f on yksilön kelpoisuus ja μ koko populaation koko.

Tämän valintaoperaattorin heikkouksia on nopea konvergoituminen, mikäli löydetään kelpoisuudeltaan suuri yksilö verrattuna muihin yksilöihin. Myöskin heikkoutena on, mikäli kelpoisuuksien arvot ovat lähellä toisiaan. Tällöin evoluution paine on liian pieni ajamaan populaation kelpoisuutta eteenpäin. Tämä voidaan havaita populaation kelpoisuuden keskiarvon kasvaessa vain hyvin hitaasti sukupolvien välillä. [9]

Sijoitusvalinta (ranking selection) pyrkii kiertämään KVV heikkoudet. Tässä valintaoperaattorissa kunkin populaation yksilöt ensin järjestetetään niiden kelpoisuuksien suhteen järjestykseen. Lisääntymään pääsevät yksilöt valitaan stokastisesti siten, että valintatodennäköisyys on suhteessa yksilön sijoitukseen populaation sisällä. Valintatodennäköisyys voi olla lineaarisessa suhteessa yksilön sijoitukseen populaation sisällä, tai se voi riippua ekspotentiaalisesti yksilön sijoituksesta. Huomioitavaa on, että populaation kaikkien yksilöiden valintatodennäköisyyksien summan tulisi olla yksi, jotta kyseessä olisi todennäköisyysavaruus. [18]

Turnajaisvalinta (tournament selection) Edelliset esitetyt valintaoperaattorit perustuvat siihen, että tiedossa on koko käsiteltävän populaation kelpoisuudet. Näin ei kaikissa GA:n mahdollisissa käyttötapauksissa ole. Kelpoisuuden määrittäminen yksilölle voi olla las-

kennallisesti raskasta. Näin käy esimerkiksi NAH:in tapauksessa, jossa yksilön kelpoisuuden selvittäminen vaatii laskenanllisesti raskaan vastavirta-algoritmin käyttämistä neuroverkon opettamisessa. Turnajaisvalinnassa koko populaation kelpoisuutta ei tarvitse olla etukäteen tiedossa. Turnajaisvalinnassa tarvitaan oletuksena, että kahden yksilön välillä voidaan kelpoisuudeltaan suurempi yksilö selvittää.

Algorithm 1 Turnajaisvalinnan algoritmi

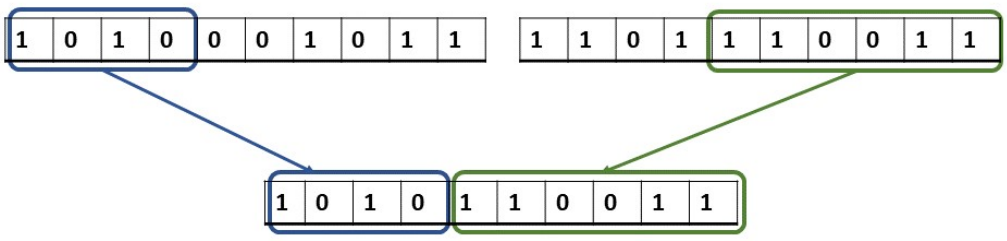
Require: $k \geq 2$
  $\alpha = 0$
  while $\alpha \leq \mu$ **do**
    Valitse k kappaletta yksilöitä satunnaisesti populaatiosta
    Valitse valituista yksilöistä kelpoisuudeltaan paras yksilö i
    Uusi lisääntymispopulaatio $\leftarrow i$
    $\alpha = \alpha + 1$

Tässä algoritmissa k on turnajaisiin osallistuvien yksilöiden määrä ja μ on populaation koko. Turjanaisvalinnan tuloksena on uusi populaatio johon seuraavaksi kohdistetaan variaatio-operaattorit rekombinaatio ja mutaatio. Muuttamalla parametrin k arvoa voidaan evoluutiopaineen arvoa muuttaa. [19]

2.5.4 Rekombinaatio

Rekombinaatio-operaattori yhdistää kahden tai useamman yksilön genotyyppiesityksen toisiinsa. Tässä aliluvussa keskitytään pelkästään binäärivektorina esitettyyn genotyyppiesitykseen. Rekombinaatiossa otetaan valintaoperaattorin valitsemia yksilöitä ja käytetään niiden genotyyppiesityksiä uuden yksilön tuottamiseen. Yleisimmin käytetyt rekombinaatio-operaattorit esitellään seuraavaksi.

Yhden pisteen risteytys (one-point crossover) kaksi lisääntymään valittua yksilön geeneistä otetaan kummastakin osa, käyttämällä satunnaisesti valittua katkaisukohtaa. [20]



Kuva 2.3. *Yhden pisteen risteytys*

Kuvassa 2.3 on esitetty esimerkkinä yksi mahdollinen tapaus yhdenpisteen rekombinaatiosta.

L-pisteen risteytys (l-points crossover) on laajennus yhdenpisteen risteytyksestä. Siinä valitaan satunnaisesti monta katkaisukohtaa joiden väliin vuorotellen kopioidaan kummankin vanhemman geeneistä lohkot. L-pisteen risteytys mahdollistaa myös enemmän kuin kahden vanhemman käytön risteytykseen. L-pisteen risteytystä on käytetty kolmen leikkauspisteen tapauksena tässä opinnäytetyössä.

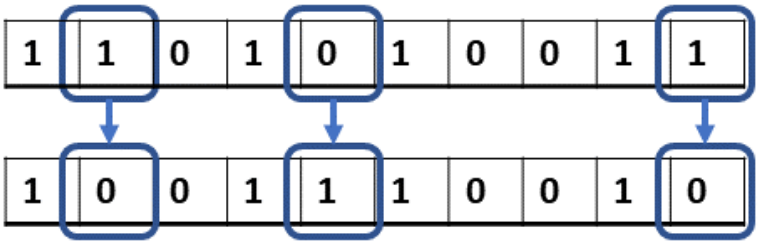
Yhtenäinen risteytys käytetään yleensä tasaista satunnaisjakaumaa välillä $p \leftarrow [0, 1]$ kaikille uuden yksilön binäärivektorin kohtiin. Mikäli $p \leq 0.5$ valitaan tähän geenivektorin kohtaan ensimmäisen yksilön binääri. Muutoin valitaan toisen lisääntyvän yksilön kyseisen geenivektorin alkio.

Näillä rekombinaatio-operaattorin valinnoilla on eri etuja. Yhdenpisteen risteytys, sekä L-pisteen risteytys pitävät lähellä toisiaan olevien binäärivektorin arvoja samoina. Tämä on jossain tapauksissa edullista. Esimerkiksi tässä opinnäytetyössä käsitellyssä NAH:issa on osa fenotyypin informaatiosta koodattu enemmän kuin kahden vaihtoehdon valintana. Tämä voi olla oleellista yksilön kelpoisuuden kannalta, että jälkeläinen saa tämän moneen binäärin jonoon koodatun fenotyypin ominaisuuden itseensä. Yhdenpisteen risteytyksen ongelmana on, että uuteen yksilöön kopioituu vasempaan reunaan vain ensimmäisen vanhemman geenejä ja oikeaan reunaan vain toisen vanhemman geenejä. Tätä kutsutaan *sijaintiharhaksi* (positional bias). Yhtenäisellä risteytyksellä ei tätä sijaintiharhaa ole, vaan se yleensä siirtää 50% geeneistä eteenpäin kummastakin vanhemmasta, sillä oletuksella että kummankin vanhemmat mahdollisuus siirtää geeninsä eteenpäin on 50%. Kuhunkin käsillä olevaan ongelmaan sopii yleensä yksi rekombinaatio-operaattori paremmin, kuin toinen. [21] Tällaiset vajavaisuudet liittyvät keskeisesti optimointiongelmien ratkomiseen. Ei ole yleispäteviä tehokkaasti toimivia menetelmiä. Tämä tunnetaan myös "No free lunch"teoreemana. [22]

2.5.5 Mutaatio

Mutaatio on rekombinaatio-operaattorin lisäksi toinen variaatio-operaattori. Sen tarkoitus on muuttaa jo löydettyjen yksilöiden genotyyppiä uusien mahdollisesti parempien ratkaisujen toivossa. Yleisin käytössä oleva mutaatio-operaattori perustuu tasaiseen todennäköisyysjakaumaan. Valinnan ja rekombinaatio-operaattorin tuottamalle uudelle sukupolven kullakin yksilöllä on etukäteen määritetty todennäköisyys p_{yks} mutantoitua. Mutaatio toteutetaan binääriesityksen tapauksessa muuttamalla kunkin binääri vastakkaiseksi etukäteen määritetyllä todennäköisyydellä p_{mut}. Geenin binääriesityksen ollessa L pituinen, johtaa se keskimäärin yhdellä yksilöllä $\approx L \cdot p_{mut}$ binäärin vaihtumiseen vastakkaiseksi.

Kuvassa 2.4 on esitetty tapaus, missä on $L = 10$ ja $p_{mut} = 0.3$. Yksilön mutaatiomahdollisuuden p_{yks} ja yhden bitin vastakkaiseksi vaihtumisen p_{mut} arvojen valinta seuraa yleensä käytössä olevan sovelluksen tarpeista. Jos halutaan populaatio jossa kaikilla yksilöillä on suuri kelpoisuus, valitaan p_{mut} ja p_{yks} pienet arvot. Mikäli taas halutaan löytää



Kuva 2.4. *Esimerkki mutaatiosta*

yksi kelpoisuudeltaan korkea yksilö, käytetään näille suurempia arvoja. [2]

2.5.6 Eloonjääneiden valinta

Eloonjääneiden valinnan tarkoituksena on hallita GA:n käsittelemän populaation kokoa. Jokaisella kieroksella μ yksilöitä tuottaa γ verran jälkeläisiä jotka lisätään populaation. Jotta populaation koko pysyisi samana, on yksilöitä poistettava.

Ikäpohjainen korvaamisen (age-based replacement) strategiassa eloonjäävät yksilöt valitaan ikänsä mukaisesti. Tämä tarkoittaa sitä, että ne yksilöt jotka ovat olleet tietyn ennalta määrätyn sukupolvien ajan populaation mukana, poistetaan populaatiosta. Yksinkertaisimmassa GA:ssa jokainen valinnan ja rekombinaation lävitse tuottamat yksilöt muodostavat kokonaisuudessa seuraavan populaation. Vaihtoehtona tälle on käyttää satunnaista korvaamista. Tässä strategiassa populaatiosta valitaan satunnaisesti yksilöitä jotka poistetaan. Tämän satunnaisen korvaamisen strategia perustuu siihen, että populaation jokaisella yksilöllä on sama todennäköisyys poistua populaatiosta. Tällöin jos populaation koko on vakio, pysyy populaation kelpoisuuden keskiarvo keskimäärin samana. Satunnaisen korvaamisen stategian on havaittu lisäävän populaation keskimääräistä varianssia. [23]

Kelpoisuuteen perustuva korvaaminen (fitness-based replacement) eloonjääneiden valintamekanismi perustuu yksilöiden kelpoisuuksiin. *Korvaa heikoimmat* (replace worst) perustuu siihen, että ennaltamäärätty määrä kelpoisuudeltaan huonoimpia yksilöitä poistetaan populaatiosta. Tämä yleensä johtaa populaation kelpoisuuden keskiarvon nopeaan nousemiseen, on haasteena sen liian nopea konvergoituminen parhaan yksilön esittämän ratkaisun ympärille. Tämän vuoksi korvaa heikoimmat strategiassa on yleeensä käytössä suuri populaatio. [2]

3. EVOLUTIIVISET NEUROVERKKOARKKITEHTUURIHAUT

Tämä opinnäytetyö käsittelee ohjattua oppimista suorittavia neuroverkkoja. Ohjattua oppimista käyttävissä neuroverkoissa käytetään *vastavirta-algoritmia* (backpropagation) neuroverkon neuronien sisääntulojen painojen optimoinnissa. Vastavirta-algoritmissa käytetään tietoa opetusaineiston luokista, tai regressio-ongelmissa numeerista arvoa. Neuroverkon ulostuloa verrataan tähän tiedettyyn luokkaan/arvoon neuroverkon loppupäässä. Käyttäen moniulotteisen derivaatan ketjusääntöä, voidaan neuroverkon ulostulon erotus oikeaan haluttuun arvoon paloitella kunkin neuronin sisääntulon painokertoimien suhteen erilleen. Tällä tiedolla voidaan neuronien sisääntulojen painokertoimia muuttaa siten, että neuroverkon tuottama virhe validointijoukkoon pienenee. [24]

Ongelmana vain on, että emme etukäteen voi tietää minkälainen neuroverkkoarkkitehtuuri on sopiva kuhunkin käsillä olevaan ongelmaan. Nykytilanteessa itse sopivan neuroverkkoarkkitehtuurin löytäminen ja sen hyperparametrien optimointi on koneoppimisen sovelluksissa suurimpia kulueriä. Tämä vaatii aiheen asiantuntijalta kokemusta ja tietotaitoa, rakentaa ongelmaan sopiva neuroverkko. Yleensä tämä on aikaa vievä prosessi, koska monta erillaista neuroverkkoarkkitehtuuria joudutaan käymään käsin lävitse ja toistamaan monta iteraatio parantaen. Ongelmaa pahentaa vielä se, että neuroverkkojen opetus on käytännössä stokastinen prosessi, koska neuroverkkojen neuronien sisääntulojen painokertoimet ovat yleensä eri satunnaisjakaumilla alustettu. Joten sama neuroverkko voi antaa eri tuloksia täysin samaan aineistoon, eri opetuskerroilla. Neuroverkkojen arkkitehtuurien ja hyperparametrien optimointia voidaan pitää erittäin vaikeana osittain ratkaisemattomana ongelmana. Syynä tähän on se, että ei ole löydetty mitään vastavirta-algoritmin tapaista yleistä keinoa määrittää miten neuroverkon virheen gradientti toimii neuroverkon arkkitehtuurin suhteen. [25]

Neuroverkkoarkkitehtuurihaku

Tähän ongelmaa on haettu erillaisia ratkaisuja. Yksi ratkaisutyyppi on käyttää evolutiivisia algoritmeja sopivan neuroverkon arkkitehtuurin hakemisessa ja hyperparametrien optimoinnissa. Tämä opinnäytetyö on tehty yhden tällaisen geneettistä algoritmia hyväksikäyttävän menetelmän ympärille. Aiheeseen liittyvien viitteiden ja tutkimusten määrä on viime vuosina kasvanut ekspotentiaalisesti [26]. Kyseessä on yksi keskeinen ko-

neoppimisen kehityssuunta, jossa evoluutiota käyttäen kasvatatetaan ongelmaan sopivia neuroverkkoja. Suurimpia rajotteita on GA:n käytössä sen vaatima hyvin suuri laskentateho. Tämä johtuu siitä, että jokainen yksilön kelpoisuus pitää selvittää opettamalla vastavirta-algoritmilla käytössä olevaan opetusaineistoon neuroverkko. Mikäli opetusaineisto on suuri, muodostuu tämän lähestymistavan pullonkaulaksi yksittäisen yksilön kelpoisuuden selvitys. Jotta tämä GA:han perustuva NAH toimisi, on kyettävä selvittämään mahdollisimman suuren yksilömäärän kelpoisuudet. [26]

Evolutiivisia algoritmeja on käytetty neuroverkkojen neuronien painojen optimoinnissa rinnan neuroverkon arkkitehtuurin ja parametrien optimoinnin kanssa. Tätä suuntausta sanotaan *neuroevoluutioksi* (neuroevolution). Tämän lähestymistavan suuri ongelma on, että vastavirta-algoritmi on kertaluokkia nopeampi neuroverkon sisääntulojen painojen optimoinnissa. Nykyinen tutkimus NAH suhteen on pääosin keskittynyt miten neuroverkkojen arkkitehtuuria ja hyperparametreja voidaan optimoida. Itse opetus tässä lähestymistavassa on hoidettua vastavirta-algoritmia käyttäen.

Seuraavissa aliluvuissa käydään pintapuolisesti muutama evolutiivisilla algoritmeilla toteutettuja NAH:ja läpi. Lähtökohtana on käytetty kokooma-artikkelia [26], josta on poimittu tämän opinnäytetyön kannalta mielenkiintoisimman artikkelit.

3.1 Geneettinen konvoluutioneuroverkko

Tämä aliluku käsittelee geneettisellä algoritmillä toteutettua NAH:ia joka on kuvailtu artikkelissa *Genetic CNN* [27]. Aluksi käsittelen lyhyesti kronologisessa järjestyksessä muutamia virstanpylväitä konvoluutioneuroverkkojen kehityksessä.

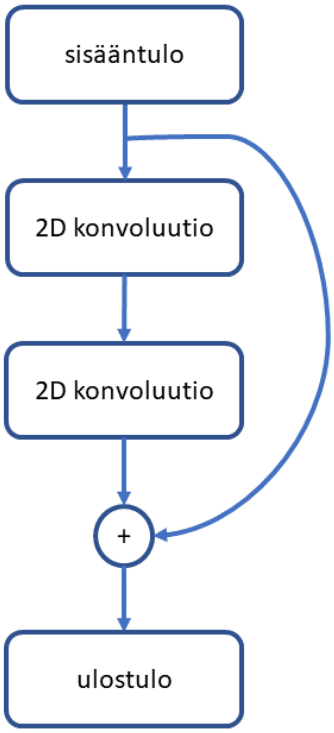
3.1.1 VGG arkkitehtuuri

Vuonna 2014 pidetty *ImageNet Large Scale Visual Recognition Challenge* [28] koneoppimisen kilpailussa sai objektien paikan määrittämisessä ensimmäisen sijan, ja objektien tunnistamisessa toisen sijan *VGG* niminen konvoluutioneuroverkkoon perustuva neuroverkkoarkkitehtuuri. VGG arkkitehtuuri perustuu monen peräkkäisen konvoluutiokerroksen käyttöön, joilla saavutettiin suurempi avaruudellinen kernelin koko, kuin vain käyttäen yhtä konvoluutiokerrosta. Kisan aikoihin suurin osa konvoluutioneuroverkoista perustui peräkkäisiin konvoluutiokerroksiin ja *koontikerroksiin* (pooling layer). VGG:n arkkitehtuurissa käytettiin monta peräkkäistä konvoluutiokerrosta joiden kerneleiden koko oli $3 \cdot 3$. Näillä saavutettiin pienemmällä parametrimäärällä saman kokoinen kerneli-ikkuna, kuin kasvattamalla vain yhden konvoluutiokerroksen kernelin kokoa. VGG arkkitehtuureja on konvoluutiokerrosten määrän suhteen nimetty VGG-16 ja VGG-19. Nämä Imagenet kuvapankkiin opetetut verkot löytyvät *Tensorflow* nimisessä neuroverkkojen laskentaohjelmistossa suoraan. Näitä valmiiksi opetettuja verkkoja käytetään vieläkin useiden konenäön

ongelmien ratkaisuun, poistamalla vain VGG verkon lopusta viimeiset tiheistä neuroneista koostuvat kerrokset. [29]

3.1.2 ResNet arkkitehtuuri

ImageNet kisassa vuonna 2015 kisan voitti Microsoftin joukkue, joka käytti ResNet nimistä erittäin syvää residuaalista konvoluutioverkkoarkkitehtuuria [28]. Tässä neuroverkkoarkkitehtuurissa *hiipuva virheen gradientti* (vanishing gradient problem) oli ratkaistu residuaaliyhteyksillä konvoluutiokerrosten välillä. Tämän yhteyden käyttö auttoi hiipuvan virheen gradientin ongelmassa, koska neuroverkon peräkkäisten konvoluutiokerrosten välillä on myös yhteys, joka syöttää dataa eteenpäin. Aikaisemmin oli keskeisin este syvempien konvoluutioneuroverkkoarkkitehtuurien kehittämisessä mainittu hiipuvan virheen gradientti. ResNet oli ensimmäisiä askeleita päästä kertaluokkaa enemmän konvoluutiokerroksia sisältävien neuroverkkojen käyttöön. [30]

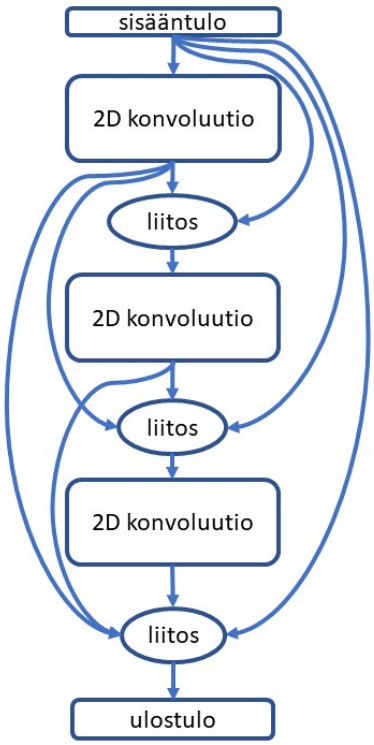


Kuva 3.1. *ResNet arkkitehtuurin yksi lohko*

Kuvassa 3.1 nähdään yksittäisen lohkon toimintaperiaate, jossa sisääntulosta on myös suora yhteys ulostuloon, konvoluutiokerrosten lisäksi. ResNet arkkitehtuurin menestys lisäsi kiinnostusta tämän tyyppisiin rinnakkaisiin yhteyksiin konvoluutiokerrosten välillä.

3.1.3 DenseNet arkkitehtuuri

ResNet arkkitehtuurin jälkeen luonnollisesti heräsi kysymys, voidaanko tätä tapaa kehittää eteenpäin kohti paremmin toimivia konvoluutioneuroverkkoarkkitehtuureja. Tähän kysymykseen vastasi *DenseNet* arkkitehtuuri vuonna 2017. Kyseissä arkkitehtuurissa verkko koostuu erillisistä *lohkoista* (block) jotka koostuvat peräkkäisistä konvoluutiokerroksista. Jokaisen lohkon sisällä olevalla konvoluutiokerroksilla on yhteys jokaiseen sitä edeltävään konvoluutiokerrokseen, jotka yhdistetään *yhteenliitoksella* (concatenation), erona ResNet arkkitehtuuriin. Lohkon lopussa on yksi konvoluutiotaso ja sen jälkeen koontikerros joka syöttää syötteen seuraavalle lohkolle. Tämä verkon rakenne mahdollisti yhteyden jokaisen lohkon konvoluutiokerrosten välillä lohkon sisällä. Etuja tällä arkkitehtuurilla on, että vastavirta-algoritmilla opettaessa verkkoa lyhentää tämä eri konvoluutiokerrosten yhteyksiä optimoitavaan virheeseen. Myöskin parametrien käyttö on tehokkaampaa, koska neuroverkon saamaa informaation ei tarvitse kulkea jokaisen konvoluutiokerroksen lävitse erikseen. Tiheiden yhteyksien käyttö myös estää perinteisten konvoluutioneuroverkkojen ongelmaa, jossa jokaisen kerroksen pitää pystyä myös välittämään olennainen informaatio seuraavilla kerroksille. DenseNet tyyppiseessä arkkitehtuurissa jokainen yksittäinen konvoluutiokerros pääsee optimoimaan itseään juuri neuroverkon kokonaistehokkuuden kannalta olennaisia ominainsuuksia, koska eri *ominaisuuskartat* (feature map) yhdistetään toisiinsa summaamisen sijaan ennen seuraavalle lohkolle siirtämistä. [31]



Kuva 3.2. *DenseNet arkkitehtuurin yksi lohko*

Kuvasta 3.2 voidaan nähdä yksittäisen DenseNet arkkitehtuurin toimintaperiaate. Kaikilta

konvoluutiokerroksilta on suora yhteys aikaisempiin ja myöhempiin konvoluutiokerroksiin.

DenseNet arkkitehtuuri oli julkistamisaikanaan parhaiten suoriutuva konvoluutioneuroverkko. Tämä herätti kysymyksen, voisiko konvoluutioneuroverkkojen arkkitehtuuria käsitellä yksittäisista konvoluutiotasoista koostuvista lohkoista, joita liitetään peräkkäin yhteen. Sekä voitaisiinko yksittäisen lohkon sisäistä arkkitehtuuria optimoida kullekin käyttökohteelle automaattisesti.

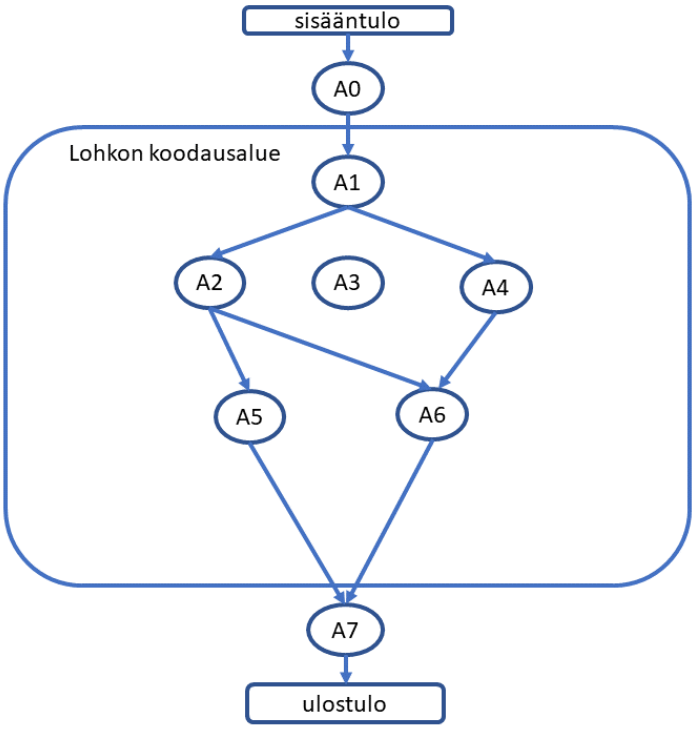
3.1.4 Lohkojen sisällön koodaaminen binääriesitykseen ja geneettisen algoritmin käyttö näiden optimoinnissa

Edellä huomattiin, että parametrien käytön suhteen ja ilmaisuvoimaisten konvoluutioneuroverkkojen kehityksessä keskityttiin yksittäisten, monista konvoluutiokerroksista koostuviin, lohkoihin. VGG arkkitehtuurissa lohkot sisälsivät useita peräkkäisiä konvoluutiokerroksia. ResNet arkkitehtuurissa lohkojen sisällä oli lohkon alusta yhteys lohkon ulostuloon. DenseNet arkkitehtuurissa jokainen lohkon sisäinen konvoluutiokerros on yhdistetty aiempiin kerroksiin. Näiden lähestymistapojen ongelmana on, että neuroverkkoarkkitehtuuri koostuu peräkkäisistä samantyyppisistä lohkoista joita vain on yhdistelty peräkkäin. Olisiko mahdollista optimoida jokaisen yksittäisen lohkon rakennetta käsillä olevan ongelman mukaisesti automaattisesti, ilman käsin optimointia?

GA:n lähestymistavassa jokainen yksilö esitetään binääriarvoista koostuvana geeninä. Geeniin on etukäteen määritetty kullekin lohkon esitykselle oma alueensa. Alueen sisäisesti koodataan peräkkäin, yksittäisen konvoluutiokerroksen mahdolliset yhteydet sitä aikaisempiin. Etukäteen täytyy määrittää, kuinka monta mahdollista konvoluutiokerrosta voi yhdessä lohkossa maksimissaan olla. Tämän koodauksen etuja on, että sillä pystytään koodaamaan genotyyppiin kaikki edellisissä aliluvun neuroverkkoarkkitehtuurien lohkojen sisäinen rakenne.

Kuvassa 3.3 on esitetty yksi mahdollinen lohkon sisältö, missä on lohkon sisäisessä koodausalueessa kuusi mahdollista konvoluutiokerrosta ulostulokerroksen lisäksi. Binääriesityksessä ensimmäinen osio on pituudeltaan yksi bitti. Tämä koodaa, onko A1 konvoluutiokerroksella mahdollisesti yhteys sisäätulon jälkeiseen A0 konvoluutiokerrokseen. Seuraava osio on kahden bitin pituinen, joka ilmaisee ensimmäisessä kohdassa onko tällä konvoluutiokerroksella yhteys sisääntulon A0 konvoluutiokerrokseen ja seuraava onko yhteys A1 konvoluutiokerrokseen. Jokaisen osion osuus kasvaa yhdellä bitillä, koska aiempia mahdollisia yhteyksiä on yksi enemmän. Näin muodostuu aritmeettinen sarja, jonka summa:

$$1 + 2 + 3 + \ldots + n = \frac{K_s(K_s - 1)}{2} \tag{3.1}$$



Kuva 3.3. *Yksi esimerkki lohkon koodaamisesta jonka binääriesitys: 1-01-000-0100-00100-001010-0000011*

jossa termi K_s on kyseiselle lohkolle annettujen konvoluutiokerrosten määrä. Koska yksittäisen lohkon eri mahdollisten kombinaatioiden määrä kasvaa rajusti $2^{\frac{K_s(K_s-1)}{2}}$, on kaikkien mahdollisten lohkojen läpikäynti käytännössä mahdotonta. Tässä koodauksessa on ongelmana, että osa koodauksista vastaa sellaisia lohkoja joiden sisäinen rakenne ei ole järkevä. Tällaiset tapaukset ovat niitä, missä konvoluutiokerroksilla on yhteys aiempiin, mutta ei mihinkään seuraavaan kerrokseen. Artikkelissa tällaiset tapaukset jätettiin vain neuroverkon opettamisessa ulkopuolelle, huomiomatta. Tapaukset joissa sisääntulevaan konvoluutiokerroksella ei ole mitään yhteyttä ulostulevaan konvoluutiokerrokseen, luotiin vain suora yhteys sisääntulevaan A0 konvoluutiokerroksen ja ulostulevan AN kerroksen välille. Tämän koodaustavan etu on, että sillä pystytään esittämään kaikki tapaukset joissa on myös pienempi määrä konvoluutiokerroksia. Keskeistä on, että tällä koodaustavalla voidaan esittää VGG, ResNet sekä DenseNet arkkitehtuurien sisäiset lohkot.

Käsitellyssä artikkelissa tämä lähestymistapa saavutti ResNet tasoisen tarkkuuden CIFAR-10 virheessä. Myöskin menetelmän löytämät neuroverkkoarkkitehtuurit toimivat kohtuullisen hyvin ILSVRC-2012 datasetille. Vaikka tämä menetelmä ei saavuttanut tälle CIFAR-10 huomattavasti haastavammalle datasetille yhtä hyviä tuloksia kuten ResNet, on huomattavaa että löydettyjen neuroverkkojen kerrosten määrä oli huomattavasti pienempi. [27]

Artikkelin menetelmä on rajoittunut, koska siinä optimoitiin pelkästään yhden lohkon si-

säisiä yhteyksiä. Muita keskeisiä hyperparametreja, kuten optimoijaa, aktivointifunktioita, opetusnopeutta tai ylioppimista rajoittavan dropoutin käyttö oli jätetty menetelmässä ulkopuolelle. Mielenkiintoisen artikkelista tekee sen, että siinä esitettiin tapa jolla voidaan yksittäisiä lohkoja koodata geeniin.

4. TOTEUTETTU EVOLUTIIVINEN NEUROVERKKOARKKITEHTUURIHAKU

Tehtävänä on määrittää ihmisten määrä eri kokoisissa kopeissa katossa olevan kalansilmäobjektiivin avulla. Haasteena on kuvien pieni koko ja suhteellisen pieni opetusaineisto rajatulla määrällä eri koppeja. Kuvien koko on $72 \cdot 54$ RGB pikseliä. Kuvien pieni koko ei mahdollista jo valmiiksi opetettujen neuroverkkojen käyttöä. Tähän ongelmaan käytettiin tässä opinnäytetyössä esitettyä evolutiivista neuroverkkoarkitehtuurihakua. Näitä evolutiivisella algoritmilla toteutettuja neuroverkkoarkkitehtuurihakuja toteutettiin kaksi eri versiota kummallekin koppityypille.

Koppityyppejä oli montaa eri tyyppiä. Koppien ihmiskapasiteetti oli 0-1 ja toisessa 0-4, sekä viimeisessä 0-6 ihmistä.

4.1 Kuvien käsin luokittelu

Koko aineisto koottiin itse eri kopeista joissa oli kussakin Rasberry pi tietokone liitettynä kalansilmäobjektiiviin. Kuvia otettiin 1min välein mikäli kopissa havaittiin liikettä. Muutoin kuvia otettiin 10min välein, mikäli kopissa ei havaittu liikkeentunnistimella liikettä. Koko projekti toteutettiin itse alusta lähtien. Tämä tarkoitti kuvien käsin luokittelua.

Tätä varten toteutin Pythonilla skriptin joka luki OpenCV kirjastoa käyttämällä ensin kuvat Numpyn RGB-matriisiksi. Neuroverkkojen opetusta varten haluttiin pitää sekä opetusdata että validointidata erillään. Tätä tarkoitusta varten kukin koppi josta kuvia otettiin merkittiin omalla numerotunnisteella. Kuvien käsin luokittelussa tämä tieto otettiin omaan kokonaislukuja sisältävään listaan ylös. Ohjelma tulosti käyttäjälle kaksi eri kuvaa käsin luokittelua varten. Toinen kuvista oli saatu skaalaamalla kuva $700 \cdot 600$ kokoiseksi käyttäen OpenCV kirjaston bilineaarista interpolaatiota. Koska osassa kuvista oli kontrasti huono heikon valaistuksen takia käytettiin rinnalla toista kuvaa. Tämä kuva oli saatu ensin muuttamalla alkuperäinen RGB-kuva harmaasävykuvaksi. Sen jälkeen tähän kuvaan toteutettiin histogrammin tasaaminen. Lopuksi kuva myös skaalattiin $700 \cdot 600$ kokoiseksi käyttäen bilineaarista interpolaatiota. Ohjelman käyttäjälle näytettiin molempia kuvia rinnakkain ja ohjelma otti syötteenä virhetarkistuksineen sisään kokonaisnumeron 0-6 väliltä. Näitä tietotoja, kopin tunnistusnumero ja käsin kuvista määritetyn ihmismäärän mukaisesti oh-

jelma tallensi omana hakemistoonsa kuvan. Tämän kuvan tiedoston nimessä oli kopin tunnistusnumero, tarkka `Unix timestampilla` muodostettu aika, sekä ihmisten määrä kopissa.

Koska projekti eteni tätä opinnäytetyön kirjoittamisen ohella, lisääntyi käsin luokiteltujen kuvien määrä joka viikko. Tämä aiheutti ongelmia, koska aineisto oli hyvin epätasapainoittunut luokkien suhteen.

Ongelmaa kierettiin sillä, että aikaisemmin NAH:hilla löydettyjä opetettuja verkkoja käytettiin annotoinnin apuvälineenä. NAH:in kelpoisuudeksi asetettiin alussa suoraan oikein luokiteltujen kuvien prosenttiosuus koko validaatioaineistosta. Koska eniten kuvia oli opinnäytetyön alkuvaiheessa tapauksissa joissa oli 0-2 ihmistä, oli verkko tehokkainta opettaa näille tapauksille. Näin koulutettuja verkkoja pystyttiin käyttämään jatkossa suodattamaan pois 0-2 ihmisen tapaukset, jotta pystyttiin keskittymään opetuksen kannalta kiinnostaviin ja harvinaisiin 3-6 ihmisen tapauksiin. Tämä oli keskeistä siksi, koska allekirjoittanut joutui itse luokittelemaan käsin kuvat ja kuvia kerääntyi joka viikko yli 10 000 kappaletta. Ilman tätä aiemmin koulutettujen neuroverkkojen käyttöä annotoinnin suodattimena olisi työ muuttunut pidemmän päälle liian aikaa vieväksi.

4.2 Toteutettu evolutiivinen neuroverkkoarkkitehtuurihaku käyttäen 2D konvoluutioverkkoa

Tässä opinnnäytetyössä käytettiin kahta eri rinnakkain toimivaa NAHia. Tämä johtui siitä, että tulosten perusteella vaikutti parhaita tuloksia antavan käyttää kummallekin koppityypille (0-4 ja 0-6 kopit) omaa toisesta riippumatonta NAHit. Molemmat NAHit pyörivät omilla RTX 3090 GPUlla varustetuissa koneissa.

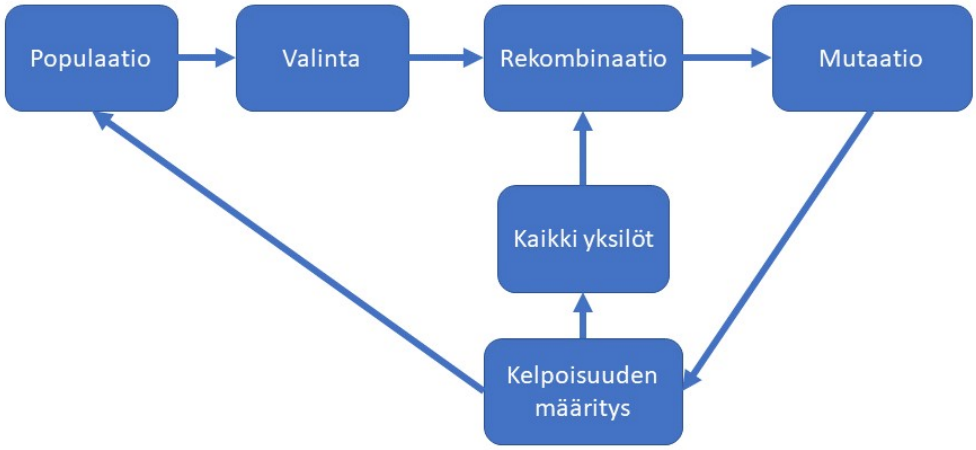
4.2.1 Evolutiivisen neuroverkkoarkkitehtuurihaun geneettisen algoritmin toimintaperiaate

NAH:eissa käytettiin allekirjoittaneen itse kehitetyä versiota GA:sta. Tämä perustui omien kokeilujen tuloksiin TTY:n kursseilla ja kisoihin liittyen [1]. Kyseessä on hyvin elitistinen GA:han perustuva NAH. NAH:in ensimmäisen sukupolven yksilöt koostuvat satunnaisesti alustetuista binäärivektoreista. Koska suurin osa näistä satunnaisista geeneistä eivät muodosta tehokasta neuroverkkoa tarvitaan jokin tapa karsia nämä epäkelvot yksilöt pois. Yksi tapa alustaa NAH on käyttää kertaluokkaa suurempaa populaatiota aloitussukupolvelle, kuin tätä seuraavien sukupolvien populaation koot ovat.

Käytössä olevan datan jako opetus ja validointiaineistoon tapahtui käyttämällä scikit-learning GroupKFold algoritmia. Opetusaineisto jaettiin kuvien päivämäärien mukaan omiin ryhmiinsä. Scikit-learn algoritmi pyrkii optimoimaan aineistoa siten, että luokkien suhteet ovat

mahdollisimman samat. Tällä algoritmillä varmistetaan, että eri datan jaotteluilla ei ole opetus/validointi datassa samoja päiviä samaan aikaan. NAH:in toiminnaksi valittiin kahden jaottelun käyttö. Näitä vuoroteltiin ja NAH:in annettiin hakea tarkkuudeltaan parhaat verkot. [32]

Käytetyn NAH:in keskeisiä periaatteita oli, että jokainen yksilö jonka kelpoisuus oli selvitetty pidettiin tallessa. Yksittäisen NAH:in aloituspopulaatio oli alustettu tasaisella satunnaisjakaumalla.



Kuva 4.1. *NAH toimintaperiaate yhdelle ajolle*

Kuvassa 4.1 voidaan nähdä yksittäisen NAH:in toimintaperiaate.

Populaatio Tämä on kyseisen sukupolven populaatio. Mikäli kyseessä on aloituspopulaatio, on se tasaisesta satunnaisjakaumasta valittu binääreistä koostuva geeni. Tässä tapauksessa jokaiselle yksilölle lasketaan ensin kelpoisuus.

Valinta Tämä on valintaoperaattori. Käytetyssä NAH:issa versiossa valinta koostui kahdesta erillisestä osasta. Ensimmäisenä selvitetään koko populaation kaikkien yksilöiden kelpoisuus. Se on suoraan kyseisen yksilön validaatiojoukon oikein arvattujen prosenttiosuus. Tämän jälkeen populaation kaikki alkiot järjestetään laskevaan järjestykseen kyseisen prosenttiosuuden suhteen. Tässä järjestetyssä listassa parhaat yksilöt pääsevät lisääntymään eliittikertoimen mukaan. Loput etukäteen määritetystä määrästä parhaita yksilöitä pääsevät lisääntymään. Risteytykseen osallistuva toinen yksilö valitaan kaikista aiemmin selvitettyjen yksilöiden joukosta käyttäen yhtälöä (4.2). Tämä kelpoisuuskerroin suoraan määritti millä todennäköisyydellä kukin yksilö pääsi lisääntymään käsiteltävän sukupolven parhaiden/eliittiyksilöiden kanssa.

Rekombinaatio Rekombinaatio-operaattoria käytettiin 3. pisteen risteytysoperaattoria. Tästä operaattorista on selittävä kuva 4.2.

Mutaatio NAH otti myös atribuuttina todennäköisyyden, millä rekombinaation tuottamat yksilöt altistuivat mutaatioperaattorille. Tämän tarkoitus oli luoda diversiteettiä kaikkien yksilöiden populaatioon. Yksilöiden mutaatio toteutettiin muuttamalla binääriarvoltaan vastakkaiseksi etukäteen annetulla tasaisella todennäköisyysjakaumalla yksilön yksittäiset geenin kohdat.

Kelpoisuuden määritys Tässä yksilön geeni dekoodattiin 2D konvoluutioarkkitehtuuriksi. Tämä arkkitehtuuri syötettiin GPU:n laskettavaksi etukäteen määritetylle ajalle epokkeja. Tuloksena oli opinnäytetyön alussa suora prosenttiosuus neuroverkon oikein arvaamista ihmismääristä. Myöhemmässä vaiheessa, kun opetusaineistoa oli tarpeeksi käytettiin tasoitettua luokkatodennäköisyyttä.

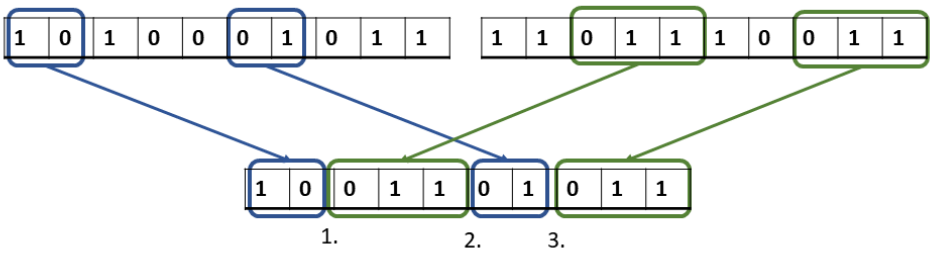
Kaikki yksilöt Jokainen yksilö jonka kelpoisuus oli määritetty lisättiin tallennettuun datamatriisiin. Tässä matriisissa oli kunkin yksilön geeni ja kelpoisuus.

Luokitellun aineiston ollessa tarpeeksi suuri etenkin tärkeiden ja harvinaisten 3-6 ihmisten tapauksien suhteen, muutettiin NAH:in kelpoisuudeksi *tasoitettu luokkatodennäköisyys*:

$$\text{Tasoitettu luokkatodennäköisyys} = \frac{1}{n}\sum_{i=0}^{n} f_{tarkkuus}(i) \tag{4.1}$$

jossa n, on kaikkien luokkien määrä ja i on yksittäinen luokka (ihmismäärä) ja $f_{tarkkuus}$ kyseisen luokan tarkkuus.

Tässä menetelmässä käytettiin kolmen pisteen risteytysoperaattoria. Tällä risteytysoperaattorilla valitaan kahdesta lisääntyvästä yksilöstä satunnaisesti vuorotellen kolme rajaa, joiden jälkeen kopiodaan vuorotellen lisääntyvien yksilöiden geenejä. Menetelmän pohjaideana on, että hyvän kelpoisuuden tuottavassa geenissä on hyviä geenin binäärien peräkkäisiä osajoukkoja. Silti näitä osia ei voida tietää, joten risteytysoperaatiossa on järkevää ottaa satunnaisesti leikkauskohtia. Normaalisti käytetään yhden pisteen risteytysoperaatiota. Tässä työssä havaittiin, että kolmen pisteen risteytysoperaattori antoi parempia tuloksia ja ajoi evoluutiota nopeammin eteenpäin.



Kuva 4.2. *Kolmen pisteen risteytysoperaatio*

Tässä NAH:issa versiossa lisääntymään pääsivät ennalta listassa määritelty määrä yksi-

löitä, jotka kukin tuottivat kolme jälkeläistä. Myöskin erillisessä listassa on määritelty montako yksilöä kyseisen populaation eliittiyksilöt pääsevät tuottamaan. Tälle ajolle ne olivat $[32, 16, 8, 4]$. Tämä tarkoittaa, että kyseisen sukupolven parhaan kelpoisuuden saanut yksilö tuottaa 32 jälkeläistä, seuraava 16, jne. Kaikki yksilöt joiden kelpoisuus on määritetty tallennetaan omaan listaansa kelpoisuusarvonsa kanssa. Kunkin populaation lisääntymiseen valitut yksilöistä muodostettiin uusia yksilöitä käyttäen 3. pisteen risteytysoperaatiota. Tähän valittiin kaikista määritettyjen geenien kelpoisuuudesta painotetulla satunnaisella otannalla käyttäen yhtälöä

$$\text{otantakerroin} = e^{w \cdot acc}. \tag{4.2}$$

jossa w on *wheel* vakio, joka määritetään kullekin NAH:in ajolle erikseen ja acc on kyseisen yksilön kelpoisuus.

Koska kelpoisuus on prosenttiosuus validointijoukon oikeista arvauksista, saadaan käyttämällä ekspotentiaalista otantakerrointa ajettua evoluutiota tehokkaasti eteenpäin. Syy tälle on se, että vertaamalla kahden eri yksilön otantakertoimia, joka myös kertoo suoraan kyseisten yksilöiden keskinäiset mahdollisuudet tulla valituksi, on voimassa:

$$\frac{e^{w \cdot acc_1}}{e^{w \cdot acc_2}} = e^{w \cdot (acc_1 - acc_2)} \tag{4.3}$$

jossa w on etukäteen annettu vakio, acc_1 on ensimmäisen yksilön kelpoisuus ja acc_2 toisen yksilön kelpoisuus.

Tämä pakottaa GA:ta käyttämään hyväksi sellaisia geenejä jotka on havaittu aiemmin tuottavan korkeita kelpoisuuksia. Matalamman arvon kelpoisuuden omaavat geenit jäävät näin vähälle käytölle. Esimerkkinä tapaus missä $acc_1 = 0.86$ ja $acc_2 = 0.80$ ja $wheel = 42$:

$$\frac{e^{42 \cdot 0.86}}{e^{42 \cdot 0.8}} = e^{42 \cdot (0.86 - 0.80)} = e^{42 \cdot 0.06} \approx 12.4$$

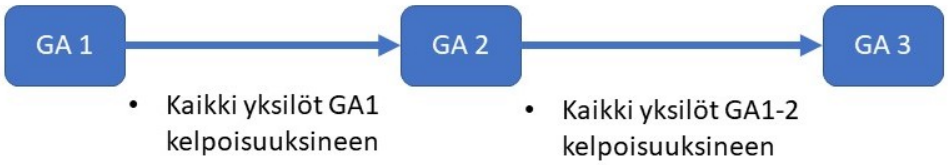
mutta taas eri kymmenyksellä olevilla eroilla, jossa $acc_2 = 0.7$:

$$\frac{e^{42 \cdot 0.86}}{e^{42 \cdot 0.7}} = e^{42 \cdot (0.86 - 0.70)} = e^{42 \cdot 0.16} \approx 828.8$$

Tämä menetelmä, jossa yksilöiden otantakerroin on ekspotentiaalisesti määritetty, havaittiin hyvin tehokkaaksi tavaksi ajaa NAH:ia eteenpäin.

Ongelmia tässä elitistisessä menettelyssä on erittäin nopea konvergoituminen tiettyyn arvoon. Tätä vastaan käytetään eri ajoille erikseen asetettua mutaatiomahdollisuutta, jossa mutaation kokeville yksilöille 4% binääreistä käännetään vastakkaiseksi. Myöskin mikä-

li populaation kelpoisuuden keskihajonta alittaa etukäteen määritetyn kynnysarvon, mutatoidaan koko populaatio. Aiempien kokemusten perusteella tällainen koko populaation mutatoiminen GA:n konvergoituessa samoihin arvoihin yleensä johti hiukan parempien ratkaisujen löytämiseen. Ajatuksena oli pohjustaa sellaista NAH:ia jossa toistetaan laskentoajan puitteissa sopiva määrä eri ajokertoja. Kullakin on mahdollisimman suuri aloituspopulaatio, jota elitistisellä valinnalla ajetaan kohti minimiä. Parhaat verkot tallennetaan ja ajoa toistetaan aina uusilla suurilla aloituspopulaation määrillä. Tässä menetelmässä on keskeistä, että aloituspopulaation jälkeen käytetään myös kaikkien aikaisempien ajojen määritettyjä geenien kelpoisuuksia. Näin tuodaan jokaisella uudella satunnaisella aloituspopulaation geeneillä uusia mahdollisia hyviä tuloksia tuottavia kohtia geenissä. Tässä on huomattavaa se, että jokaisen aloituspopulaation parhaan kelpoisuuden pääsevät lisääntymään seuraavilla sukupolvilla kaikkien ajojen aiemmin selvitettyjen geenien kanssa.



Kuva 4.3. *Monen NAH ajon yhdistäminen*

Tämä menetelmä tuotti lisää diversiteettiä kaikkien yksilöiden populaation. Johtuen siitä, että jokaisella NAH:in ajokerralla ensimmäinen populaatio on satunnaisesti alustettu. Tästä populaatiosta parhaan kelpoisuuden saaneet yksilöt pääsevät lisääntymään jo kaikkien aiemmin kelpoisuudeltaan määritettyjen yksilöiden kanssa.

4.3 Evolutiivisen neuroverkkoarkkitehtuurin geenin koodaaminen

NAH perustui suoraan GA:n käyttöön. Käytössä oli eteenpäin syöttävä lineaarinen konvoluutioneuroverkko. Kukin yksilö ratkaisi annettua luokitteluongelmaa, jossa piti koppityypistä riippuen määrittää kuvasta onko ihmisiä 0-4 vai 0-6 kappaletta tai 0-1. Optimoijan käyttämä virhe oli kategorinen ristientropia.

4.3.1 Neuronien määrän koodaus geeniin

Tässä NAH:in versiossa oli listassa ilmoitettu montako neuronia oli kullakin konvoluutiokerroksen tasolla. Esimerkkinä lista $[8, 8, 8, 8, 8]$ kuvaa viiden konvoluutiokerroksen syvyistä neuroverkkoa jossa kullekin neuronien määrälle varattu 8 bittiä. Neuronien määrä

kullekin konvoluutiokerrokselle oli ilmoitettu kokonaislukujen binäärinotaatiolla jossa määrä:

$$\text{Neuronien määrä} = 1 + \sum_{K=0}^{N} v[k] \cdot 2^k, \tag{4.4}$$

missä $v[k]$ on niiden geenin kyseisissä indekseissä olevien bittien $\{0, 1\}$ arvo jotka kuvaavat kunkin kerroksen konvoluutioneuroneiden määrää.

Tämä listan käyttämien bittien määrän ilmaisuun mahdollisti hakuavaruuden rajaamisen, koska yleensä toimivissa konvoluutioverkoissa *ominaisuuskartat* (featuremap) kasvavat kohti neuroverkon loppupäätä samalla kun kerrosten resoluutio pienenee . Esimerkiksi $[6, 6, 7, 7, 8]$ lista kuvaa hakuavaruutta, jossa vastaavien konvoluutiokerrosten neuronien määrä on:

$$\{(1-64), (1-64), (1-128), (1-128), (1-256)\}.$$

Myös lisäämällä yhtälöön (4.4) jokin kerroin eteen, on mahdollista rajata hakualuetta pienentämällä yksittäisen neuronin määrän resoluutiota pienemmäksi. Tällöin pienempi määrä bittejä riittää kuvaamaan saman hakualueen.

4.3.2 Hyperparametrien koodaus geeniin

Neuroverkkojen toiminnan kannalta keskeistä on, käytetty *optimoija* (optimization algorithm) ja sen *opetusnopeus* (learning rate), sekä mahdolliset *opetusnopeuden aikatauluttajat* (learning rate scheduler).

Ensimmäiset kolme bittiä koodasivat käytetyn optimoijan seuraavanlaisesti:

Geeni	Optimoija
0,0,0	SGD
0,1,0	Adam
0,0,1	Adadelta
1,0,0	Adagrad
1,1,0	Adamax
0,1,1	Nadam
1,1,1	RMSprop

Taulukko 4.1. *Optimoijan koodaus geeniin*

Koska kolmeen bittiin mahtuu kahdeksan erillaista kombinaatiota, ohjelma valitsi satun-

naisesti yhden näistä jos kohtasi tapauksen $''1, 0, 1''$ ja korvasin kyseisen kohdan geenistä vastaavalla bittiesityksellä. Samaa Pythonin sanakirjaan perustuvaa avainsanojen määritystä geenistä on käytetty myös muissa vastaavissa kohdissa, jotta koodi olisi helposti laajennettavissa avainsanojen poistolla/vaihdolla havaittujen tulosten mukaan.

Seuraavat bitit ilmaisivat millä kertoimella optimoijan opetusnopeutta (learning rate) kerrottiin. Tämä oli koodattu käyttäen binäärilukujen liukulukuesitystä:

$$\begin{aligned} H &= \sum_{k=0}^{P} v[k] \cdot 2^{-k-1} \\ LR &= LR_{std} \cdot H \end{aligned} \tag{4.5}$$

jossa P kuvaa montako bittiä varattu luvun esittämiseen ja LR_{std} on vakio-opetusnopeus. Huomattavaa on, että tässä koodauksessa käytetään geometrisen sarjan tietoa:

$$\sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k = 2 \Rightarrow 1 + \sum_{k=1}^{\infty} \left(\frac{1}{2}\right)^k = 1 + 1 \Rightarrow \sum_{k=1}^{\infty} \left(\frac{1}{2}\right)^k = 1. \tag{4.6}$$

Tämä takaa sen, että kaikki murtoluvut välillä $[0, 1]$ on mahdollista ilmoittaa tällä koodauksella, kunhan tarpeeksi bittejä käytössä.

Neuroverkkojen ylioppimista vastaan on käytössä monta eri tekniikkaa. Tälle NAH:ille valittiin Dropout-kerroksen käyttö. Tämä kerros opetusvaiheessa nollaa tietyllä prosenttiosuudella kyseisen kerroksen neuroneiden toiminnan. Tällä tekniikalla voidaan pakottaa verkko oppimaan oleellisia asioita saamastaan datasta. Etenkin tämän opinnäytetyön pienellä aineistolla on ylioppiminen yksi keskeinen haaste. Tälle NAH:ille varattiin 7 bittiä, joiden koodauksessa käytettiin yhtälöä (4.6). Samaa dropout arvoa käytettiin koko neuroverkon jokaiselle tasolle. [33]

Jokaiselle konvoluutiotasolle varattiin yksi bitti ilmaisemaan onko kyseinen kerros päällä vai ei. Mikäli kyseinen bitti oli tilassa 0 ohitettiin tämä kerros kokonaan ja siirryttiin konvoluutionneuroneita kuvaavassa listassa seuraavan alkion käsittelyyn.

Seuraavat kolme bittiä ilmaisivat mitä neuronien alustajaa käytettiin. Tätä samaa alustusta käytettiin kaikkien neuroneiden alustuksessa hakuavaruuden rajaamisen vuoksi. Neuronien painojen alustus on tärkeää neuroverkon oppimisen kannalta. Alustuksia on montaa eri tyyppiä. On mahdollista alustaa neuronien painot käyttäen normaalijakaumaa. Myös mahdollista alustaa neuronien painot käyttäen tasajakaumaa. Myös kehittyneempiä menetelmiä on kehitetty neuronien painojen alustukseen. Yksin näistä on Xavier alustukset jotka tunnetaan myös nimellä *GlorotNormal* ja *GlorotUniform*. Myös *HeNormal* ja *HeUniform* alustuksilla on mahdollista saada täysin satunnaista alustusta parempia tuloksia.

[34] Käytetyt alustusten koodaus on esitetty taulukossa 4.2.

Geeni	Alustaja
0,0,0	RandomNormal
0,1,0	RandomUniform
0,0,1	TruncatedNormal
1,0,0	GlorotNormal
1,1,0	GlorotUniform
0,1,1	HeNormal
1,0,1	HeUniform
1,1,1	VarianceScaling

Taulukko 4.2. *Alustajan koodaus geeniin*

Jokaiselle konvoluutiokerrokselle oli myös ilmaistu erikseen käyttäen kolmea bittiä kyseisen tason aktivointifunktio. Aktivointifunktioiden määrää myös rajoitettiin ajojen tulosten perusteella, koska havaittiin etteivät kaikki aktivointifunktiot tuota hyviä tuloksia. Tämä toteutettiin samoin, kuten luvun alussa esitetty sanakirjan käyttö. Mikäli geenin konvoluutiokerroksen aktivointifunktion kohdalla on sellainen binäärinpätkä, joka ei sisälly NAH:in alussa esitettyyn sanakirjaan, arvotaan jokin toinen sanakirjaan sisältyvä binäärinjono tähän kohdalle. Käytössä olleet aktivointifunktiot on esitetty taulukossa 4.3.

Geeni	Aktivointifunktio
0,0,0	Relu
0,1,0	Sigmoid
0,0,1	Linear
1,0,0	SoftPlus
1,1,0	SoftSign
0,1,1	tanh
1,0,1	selu
1,1,1	elu

Taulukko 4.3. *Alustajan koodaus geeniin*

Jokaiselle konvoluutiokerrokselle oli myös kahdella bitillä koodattu kernelin koko seuraavanlaisesti:

Yhdellä bitillä koodattiin, onko kyseisellä konvoluutiokerroksella BatchNormalization niminen kerros joka auttaa verkon optimoinnissa katoavan virhegradientin suhteen. Tämä taso normalisoi saamansa sisääntulon. Normalisoimalla sisääntulo arvojen $[0, 1]$ välille, pysyvät seuraavan konvoluutioneuronikerroksen virheen gradientti vastavirta-algoritmelle

Geeni	Kernelin koko
0,0	1
0,1	3
1,0	5
1,1	7

Taulukko 4.4. *Kernelin koon koodaus geeniin*

sopivana kertaluokkana. Tällä pyritään kiertämään pieneksi muuttuva virheen gradientti vastavirta-algoritmin käytössä. Etukäteen ei voida tietää, onko neuroverkon rakenteen kannalta optimeinta jokaisella konvoluutiokerroksella olla normalisoiva kerros. Tässä toteutuksessa GA:n annettiin valita normalisaatiokerroksen käyttö kullekin tasolle erikseen.[35]

Konvoluutioneuroverkoilla on yleensä tapana jokaisen konvoluutikerroksen jälkeen puristaa dataa kasaan. Tähän on olemassa monta eri tapaa. Tälle NAH:in ajolle käytettiin yhden bitin koodausta, joka määritti minkälaista datan pienennystasoa käytettiin:

Geeni	Yhdistämistason tyyppi
0	AveragePooling2D
1	MaxPooling2D

Taulukko 4.5. *Yhdistämistason koodaus geeniin*

Koska tällä NAH:illa ajolla oli mahdollista hypätä konvoluutiotasojen ylitse, on haasteena hyvin matalien verkkojen tapaukset. Tällöin konvoluutiokerroksia on vähän, jolloin ominaisuuskartat ovat kooltaan suhteellisen suuria. Ongelmaksi muodostuu neuroverkon pohjalla olevat täysin yhdistyneet neuronien kerrokset. Parametrien määrä kasvaa liian suureksi jotta GA:n toteutus olisi mahdollista, koska GA:n toimivuuden vaatimuksena on yhden yksilön kelpoisuuden suhteellisen lyhyt laskenta-aika.

Tämä ongelma kierettiin käyttämäällä ennen mahdollisia täysin yhdistyneitä neuroneita ns. *globaalia yhdistämistasoa* (global pooling). Tässä lähestymistavassa käytetään konvoluutioneuroverkon lopussa kutakin ominaisuuskartan keski-arvoa tai maksimiarvoa. Tällä menetelmällä voidaan myös vähentää ylioppimisen ongelmaa, joka etenkin tämän opinnäytetyön kaltaisessa ongelmassa pienellä opetusaineistolla vaikeus. Geeniin oli kuitenkin koodattu mahdollisuus lisätä neuroverkon loppuun 0, 1 tai 2 kappaletta tiheiden yhdistyneiden neuronien kerroksia. Ajoissa nämä usein jäivät pelkästään globaaleja yhdistämistasoja ennen lopun luokitteluneuroneja, käyttävien neuroverkkojen tuloksista jälkeen. Syy tähän lienee yliopppiminen. [36]

Tämän jälkeen tällä NAH:in ajolla oli listattu vielä lopun tiheiden kerrosten neuronien määrä listalla. Käsittely oli sama kuin konvoluutiotasojen neuronien määrittämisessä (4.4).

Geeni	Yhdistämistason tyyppi
0	GlobalAveragePooling2D
1	GlobalMaxPooling2D

Taulukko 4.6. *Globaalin yhdistämistason koodaus geeniin*

Tälle NAH:ille tiheiden neuronien määrien raja-arvot olivat 1-128 neuronia jokaiselle tiheälle tasolle joita oli kaksi. Näille tasoille oli myös yksi bitti, joka määritti onko kyseinen taso päällä vai ei. Nämä tiheiden neuronien kerrosten aktivointifunktiot olivat myös koodattu kuten taulukossa 4.3.

Viimeisenä verkossa oli yksi tiheiden neuronien kerros, jolle aktivointifunktioksi oli määritetty *softmax*. Näitä neuroneita oli etukäteen määritetty määrä. Koska 0-4 ihmisen kopin tapauksessa, on 5 eri luokkaa, oli neuroneiden määrä luonnollisesti viisi. Taas 0-6 ihmisen kopin tapauksessa viimeisen kerroksen neuronien määrä oli seitsemän. Softmax aktivointifunktio antaa tuloksena suoraan todennäköisyyden, millä tämä löydetty konvoluutioneuroverkko pitää annettua näytettä kuhunkin luokkaan kuuluvana.

5. TULOKSET

Tässä luvussa käsitellään eri toteutettujen NAH:ien tuloksia.

5.1 Evolutiivinen neuroverkkoarkkitehtuurihaku 0-4 ihmisen tapaukselle

Tässä NAH:in versiossa käsiteltiin tapausta, missä kopissa voi olla 0-4 ihmistä. Aloituspopulaation koko oli tälle ajolle 1000.

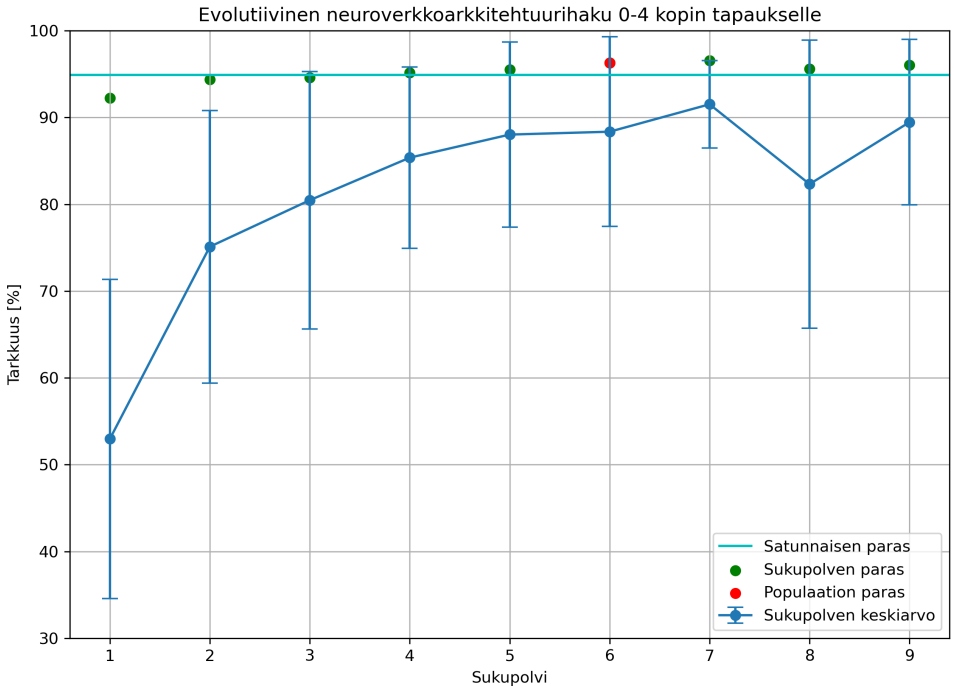
Parametri	Arvo
Yhden yksilön mutantoitumistodennäköisyys	15%
Mutaation yksittäisen geenin flippauksen todennäköisyys	4%
Eliittikerroin	[32, 16, 8, 4]
Konvoluutiokerrosten neuronien bittimäärää	[4, 4, 4, 4, 4] * 8
Tiheiden kerrosten neuronien bittimäärä	[4, 4] * 8
Wheel kerroin	42
Yhden sukupolven lisääntyvät yksilöt	75
Muiden kuin eliittiyksilöiden jälkeläisten lukumäärä	3
Sukupolvien määrä	9

Taulukko 5.1. *Evolutiivisen neuroverkkoarkkitehtuurihaun GA hyperparametrit*

Taulukossa 5.1 on konvoluutiokerrosten ja tiheiden neuronikerrosten esityksen resoluutiota tiivistetty vakiokertoimella 8. Näin on saavutettu hakualue $8 - 128$, jossa resoluutio on 8 neuronia. Tätä evolutiiviista neuroverkkoarkkitehtuurihakua verrattiin täysin satunnaisesti alustettuun, samaa geenin koodausta käyttäneeseen populaatioon jossa yksilöiden lukumäärä oli sama.

Tuloksista voidaan havaita, että tulokset ovat selvästi parempia kuin täysin satunnaisen haun tapauksessa. Myöskin kuvasta 5.1 on juuri sen sukupolven kohdalla, missä on löydetty paras yksilö myöskin mutantoitu koko sukupolvi. Tällä kertaa tämä koko sukupolven mutatoituminen keskihajonnon alittaessa etukäteen asetetun kynnysarvon ei parantanut seuraavien sukupolvien tuloksia. Myös voidaan havaita, että 8. sukupolven keskihajonta

on aiempaa suurempi.



Kuva 5.1. *Neljän ihmisen kopin NAH ajon tulokset*

Ajon tyyppi	Yksilöt	Paras yksilö	Keskiarvo	Keskihajonta
NAH	2680	96.55%	81.51%	12.45%
Satunnainen	2680	93.67%	54.63%	16.90%

Taulukko 5.2. *NAH ja satunnaisen populaation tilastoluvut*

Taulukossa 5.2 on toteutetun NAH:in arvot laskettu kaikkien sukupolvien vastaavien lukujen keskiarvoista. Kun verrataan täysin satunnaisen haun ja toteutetun NAH:in tuloksia, havaitaan toteutetun menetelmän antavan parempia tuloksia. Huomattavaa on myös, että tässä ei käytetty kuvassa 4.3 esitettyä menetelmää, jossa käytetään aikaisempien ajojen yksilöiden tuloksia. Tämän ajon löytämä paras neuroverkkoarkkitehtuuri on esitelty taulukossa 5.3.

Hyperparametri	Arvo
Satsi (Batch)	16
Opetusnopeus	0.003906
Opetusnopeuden eksponentti	-0.16
Optimoija	Adamax
Alustus	heUniform
Dropout	0.12
Tason tyyppi, aktivointifunktio	Neuronien määrä, kernelin koko
2D Konvoluutio, Relu	104, (3,3)
2D Yhdistäminen, keskiarvo	–
2D Konvoluutio, Sigmoid	28, (7,7)
2D Yhdistäminen, keskiarvo	–
2D Konvoluutio, Sigmoid	52, (5,5)
2D Yhdistäminen, maksimi	–
2D Konvoluutio, Tanh	40, (3,3)
2D Yhdistäminen, maksimi	–
2D Globaali yhdistäminen, maksimi	–

Taulukko 5.3. *NAH löytämä paras neuroverkkoarkkitehtuuri ja hyperparametrit 0-4 ihmisen tapauksessa*

5.2 Evolutiivinen neuroverkkoarkkitehtuurihaku 0-1 ihmisten tapaukselle

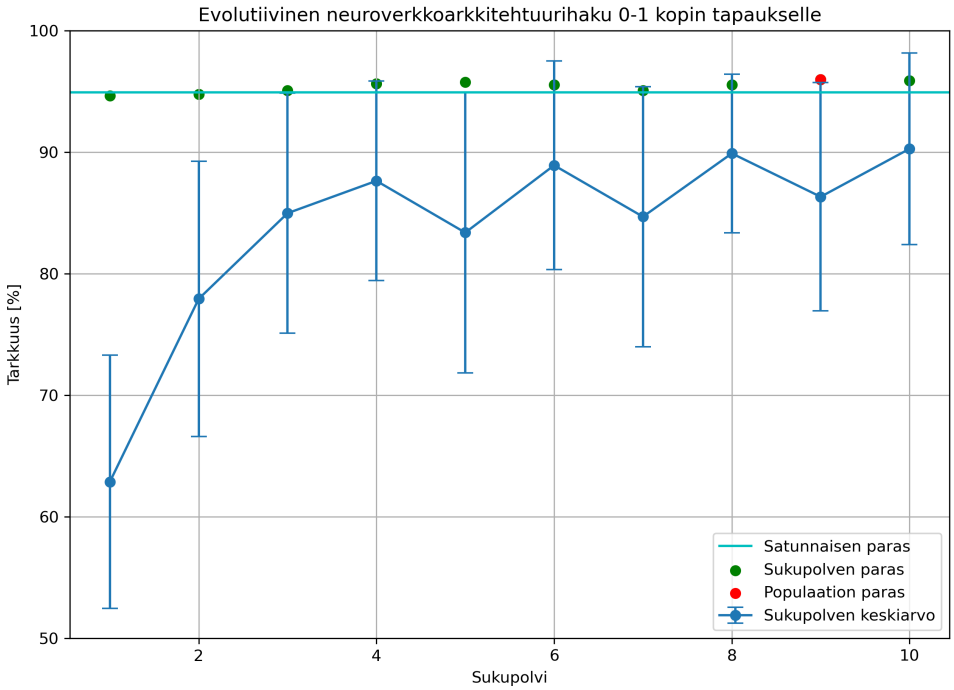
Tässä tapauksessa käytössä oli koppi, jonne mahtui maksimissaan yksi ihminen. Tavoitteena oli selvittää kuvista, onko kopin sisällä ihmistä vai ei. Tälle ajolle käytettiin toiminta-periaatteelta luvussa 4.3 esitetty ajoa, jossa aloituspopulaation jälkeen oli käytössä kaikki aiemmat yksilöt, joiden kelpoisuus oli määritetty. Optimoitavaksi virhefunktioksi valittiin tässä tapauksessa binäärinen ristientropia. Ajoja tehtiin kolmelle satunnaiselle aloituspopulaatiolle.

Data käyttö	0-ihmistä	1-ihminen
Opetus	334	571
Validointi	353	544
Yhteensä	687	1115

Taulukko 5.4. *Yhden tapauksessa käytössä ollut data*

Tässä ajossa taulukossa 5.4 ollut data oli jaettu K-group foldia käyttäen päivämäärien

mukaan opetus ja validointi joukoiksi.



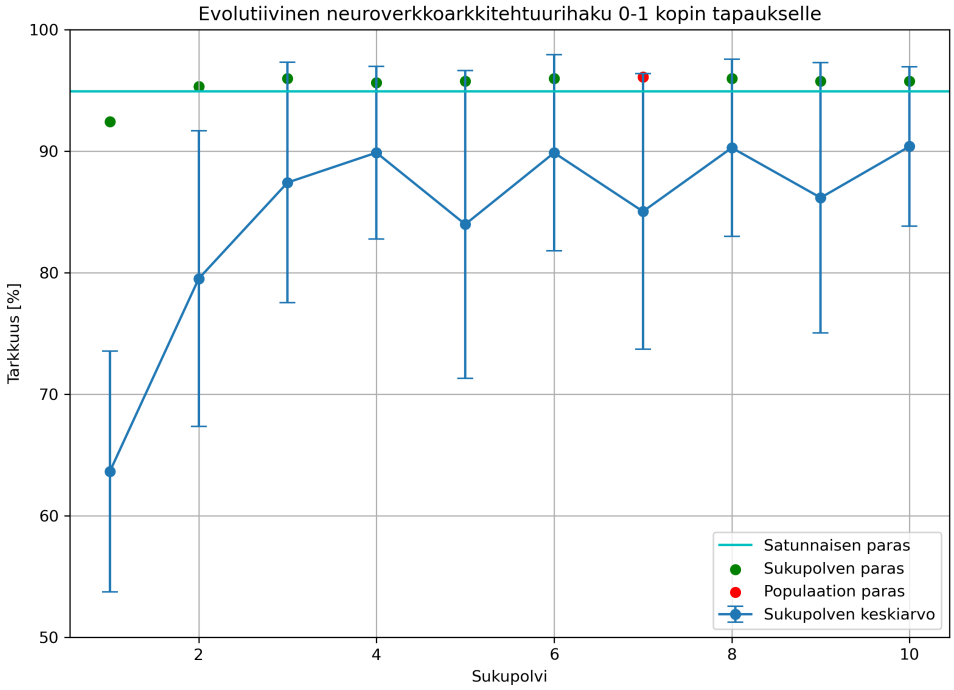
Kuva 5.2. *Yhden ihmisen kopin ensimmäinen NAH ajo*

Kuvasta 5.2 voidaan havaita, että tällä ajolla paras yksilö kullakin sukupolvella oli käytännössä sama. Mahdollisesti käytössä ollut aloituspopulaatio sisälsi huonoja geenejä, koska GA nopeasti divergoitui, ajon parhaan yksilön ollessa 95.99%. Käytössä ollut aineisto oli suhteellisen pieni. Yhden yksilön kelpoisuuteen kului keskimäärin 0.5 minuuttia. Mikäli kuvassa 4.3 esitetty periaate toimisi, täytyisi seuraavien ajojen tuottaa parempia tuloksia. Syynä tälle on se, että jokaisella seuraavalla ajolla on käytössä kaikkien aiempien ajojen geneettinen aineisto risteytysoperaattoria käyttäessä.

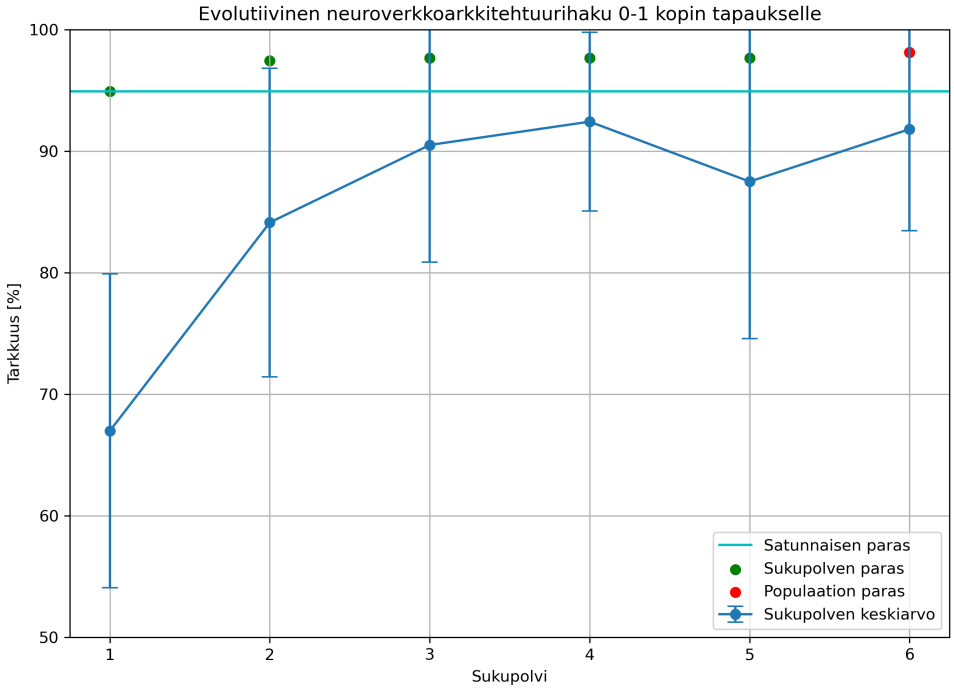
Seuraavan ajon kuvasta 5.3 voidaan havaita toisen sukupolven jälkeen populaatioiden kelpoisuuden keskiarvon olevan ensimmäistä NAH:in ajoa korkeampia. Myöskin huomattavaa on, että 5, 7, 9 populaatiot oli tuotettu kokonaan mutantoimalla kyseinen populaatio. Tämä johtui keskihajonnan jäädessä alle 10% raja-arvon, jonka seurauksena koko seuraava sukupolvi oli mutatoitu.

Viimeisellä ajolla jonka tulokset on esitelty kuvassa 5.4 voidaan havaita, että toisesta sukupolvesta eteenpäin tulokset olivat aikaisempia ajoja parempia. Tämä ajo valitettavasti jäi kesken rajallisen tietokoneajan takia. Tälle ajolle mutantoitiin koko populaatio sukupolvella 5. Ajo tuotti näistä kolmesta ajosta parhaan tuloksen, vaikka sukupolvien määrä oli edeltäviä pienempi.

Taulukosta 5.5 voidaan havaita, että kunkin yksittäisen ajon paras yksilö kasvaa. Etenkin



Kuva 5.3. *Yhden ihmisen kopin toinen NAH ajo*



Kuva 5.4. *Yhden ihmisen kopin kolmas NAH ajo*

viimeinen ajo tuotti parhaan yksilön. Nämä tulokset tukevat tämän opinnäytetyön hypo-

Ajon tyyppi	Yksilöt	Aloitussukupolvi	Paras yksilö	Keskiarvo	Keskihajonta
GA1	2911	1000	95.99%	83.69%	9.45%
GA2	2911	1000	96.10%	93.00%	9.61%
GA3	2365	1000	98.12%	85.56 %	10.65%

Taulukko 5.5. *Ajojen vertailut yhden ihmisen tapaukselle*

teesia, että käytössä ollut evolutiivinen NAH tuottaa parempia tuloksia kuin satunnainen NAH. Näiden kolmen NAH:in ajon paras löytämä neuroverkkoarkkitehtuuri on esitetty taulukossa 5.6.

Hyperparametri	Arvo
Satsi (Batch)	24
Opetusnopeus	0.003438
Opetusnopeuden eksponentti	0.0
Optimoija	Adam
Alustus	heUniform
Dropout	0.0
Tason tyyppi, aktivointifunktio	Neuronien määrä, kernelin koko
2D Konvoluutio, Relu	80, (3,3)
2D Yhdistäminen, keskiarvo	–
2D Konvoluutio, Relu	40, (3,3)
2D Yhdistäminen, Maksimi	–
2D Konvoluutio, Selu	48, (5,5)
2D Yhdistäminen, Maksimi	–
2D Konvoluutio, Softplus	44, (5,5)
2D Yhdistäminen, Maksimi	–
2D Globaali yhdistäminen, maksimi	–
Tiheää kerros, Elu	8, –
Tiheää kerros, Sigmoid	40, –

Taulukko 5.6. *NAH löytämä paras neuroverkkoarkkitehtuuri ja hyperparametrit 0-1 ihmisen tapauksessa*

Koska kunkin NAH:in ajon ensimmäinen sukupolvi oli satunnainen, pystyttiin näitä käyttämään tulosten vertailuun satunnaisen ja evolutiivisen neuroverkkoarkkitehtuurihaun välillä. Taulukossa 5.7 on vertailu tehty siten, että otettu satunnaisen ensimmäisten sukupolvien tilastolliset luvut, sekä näiden jälkeen GA:lla toteutetut sukupolvien tilastolliset ajot. Huomionarvoista on, että tällä vertailumenetelmällä on yksilöiden määrät eri. Tämä on valitettavaa, koska käytännön tarpeiden takia käytössä ollut tietokoneaika oli rajallista.

Ajon tyyppi	Yksilöiden määrä	Paras yksilö	Keskiarvo	Keskihajonta
GA1-GA3	5187	98.12%	87.41%	9.90%
Satunnainen	3000	94.92%	64.50%	11.08%

Taulukko 5.7. *Satunnaisen NAH ja evolutiivisen NAH vertailu*

5.2.1 Evolutiivinen neuroverkkoarkkitehtuurihaku 0-6 ihmisen tapaukselle

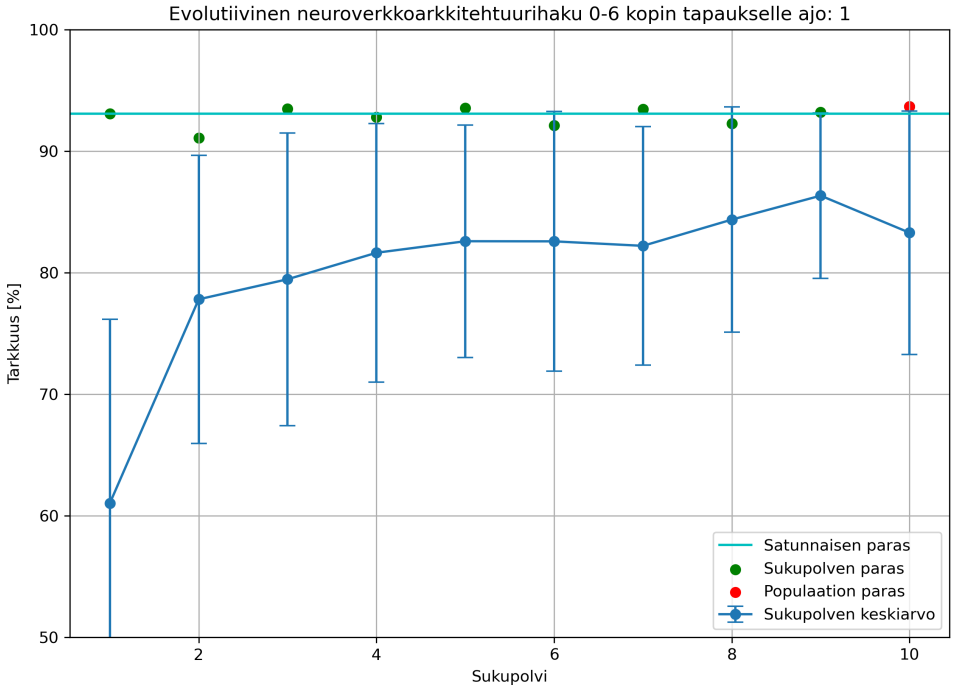
Tässä tapauksessa käytössä oli koppi johon mahtui 0-6 ihmistä sisälle. Tälle NAH:ille oli käytössä olevan laskentoajan takia tehtävä rajoitteita. Satunnaisten aloituspopulaatioiden kooksi rajattiin 500 yksilöä. Geenistä pudotettiin pois ne kohdat jotka kuvasivat vastavirta-algoritmin optimoijan opetusnopeuden vähennys, sekä satsit (patches). Yhtä verkkoa opetettiin vain 10 eepokin ajan. Muutoin käytössä oli luvussa 4.2 esitetty tyyppinen kolmen aloituksen ajo. Taulukossa 5.8 on esitetty tämän NAH:in kunkin kolmen ajon hyperparametrit. Kohdassa yksilön mutantoitumistodennäköisyys on esitetty kunkin yksittäisen kyseinen arvo. Tarkoituksena oli tätä arvoa kasvattamalla peräkkäisillä ajoilla pyrkiä korkeammalla mutaatiotodennäköisyydellä löytämään parempia parhaita yksilöitä.

Parametri	Arvo
Yhden yksilön mutantoitumistodennäköisyys	[10%, 20%, 30%]
Mutaation yksittäisen geenin flippauksen todennäköisyys	4%
Eliittikerroin	[8, 4]
Konvoluutiokerrosten neuronien bittimäärää	[4, 4, 4, 4, 4] * 8
Tiheiden kerrosten neuronien bittimäärä	[4, 4] * 8
Wheel kerroin	42
Yhden sukupolven lisääntyvät yksilöt	30
Muiden kuin eliittiyksilöiden jälkeläisten lukumäärä	2
Sukupolvien määrä	10

Taulukko 5.8. *Evolutiivisen neuroverkkoarkkitehtuurihaun GA hyperparametrit 0-6 kopin tapauksella*

Kuvasta 5.5 voidaan havaita ensimmäisen ajokerran konvergoituvan nopeasti parhaiden yksilöiden suhteen. Ajon lopussa löydettiin tämän ajon paras yksilö. Satunnainen ensimmäisellä populaatiolla löydettiin kunkin kolmen NAH:in ajon satunnaisista paras yksilö.

Kuvassa 5.6 havaitaan populaatioiden keskiaron olevan hieman korkeammalla ensimmäisen sukupolven jälkeen. Huomattavaa myös on, että tällä ajolla löytyi näiden kolmen ajon paras yksilö kuudennen sukupolven kohdalla.



Kuva 5.5. *0-6 ihmisen kopin ensimmäinen NAH ajo*

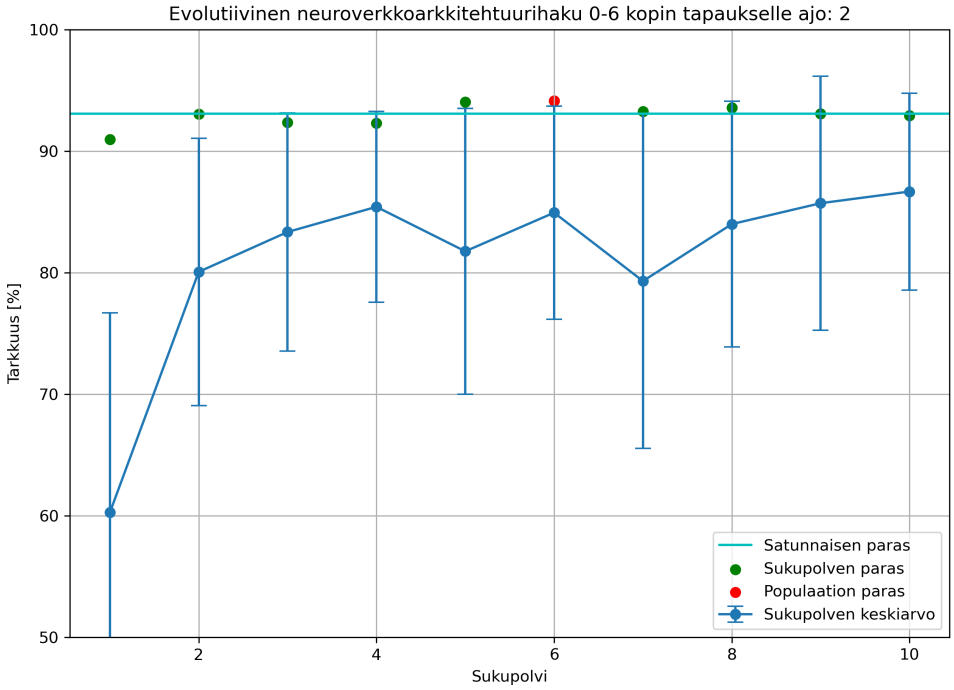
Kuvassa 5.7 havaitaan, että sukupolvien keskiarvot ovat hieman aikaisempaa kahta ajoa suurempia. Tällä ajon paras yksilö löytyi viimeisellää sukupolvella.

Näiden kolmen ajon tulokset on koottu taulukkoon 5.9. Tästä voidaan havaita, että jokainen evolutiivinen NAH:in ajo tuotti hieman parempia tuloksia, kuin täysin satunnaiset NAH:it. Vaikka yksittäisen yksilön todennäköisyyttä mutaatioon kasvatettiin kullakin ajolla, ei tämä näkynyt keskihajonnan kasvamisena. Mahdollinen syy tälle oli, että aloituspopulaation koko (500) muodosti melkein puolet yhden ajon yksilöiden lukumäärästä. Tämän tapauksen evolutiivinen NAH ei tuottanut yhtä selkeää eroa satunnaiseen NAH:iin. Syynä tälle oli oletettavasti sukupolvien yksilöiden liian pieni määrä (68) ja aikaisempia ajoja paljon pienemmät eliittiyksilöiden lukumäärä.

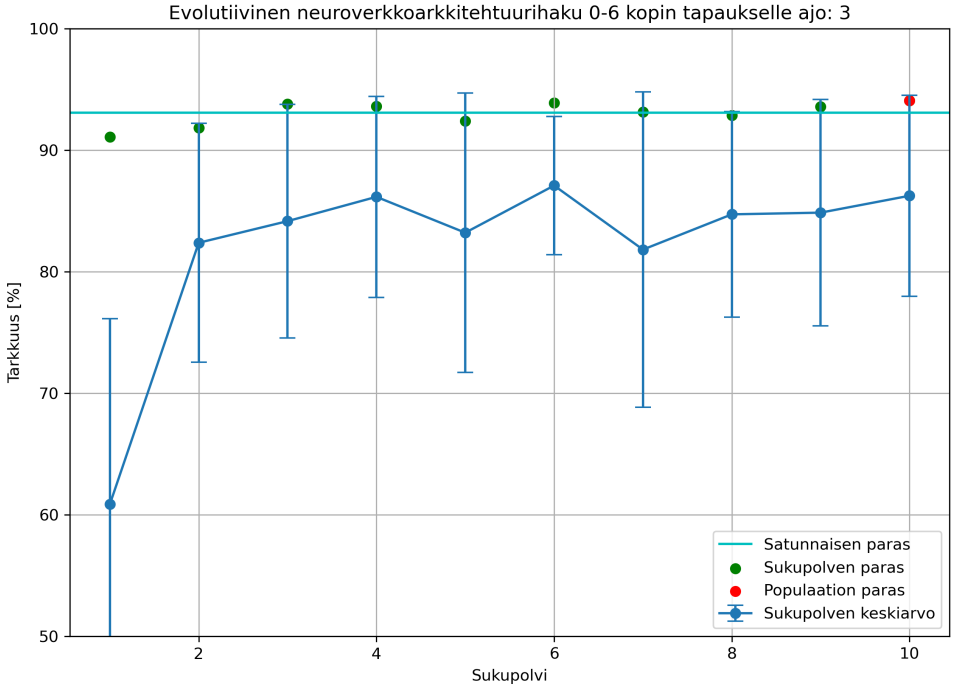
Ajon tyyppi	Yksilöiden määrä	Paras yksilö	Keskiarvo	Keskihajonta
GA1	1112	93.69%	80.13%	10.58%
GA2	1112	94.16%	81.15%	10.87%
GA3	1112	94.07%	82.14%	9.92%
Satunnainen	1500	93.07%	60.71%	14.83%

Taulukko 5.9. *Satunnaisen NAH:in ja evolutiivisen NAH:in vertailu 0-6 kopin tapauksella*

Paras löydetty neuroverkkoarkkitehtuuri on esitetty taulukossa 5.10.



Kuva 5.6. *0-6 ihmisen kopin toinen NAH ajo*



Kuva 5.7. *0-6 ihmisen kopin kolmas NAH ajo*

Hyperparametri	Arvo
Optimoija	Adam
Alustus	heNormal
Dropout	0.38
Tason tyyppi, aktivointifunktio	Neuronien määrä, kernelin koko
2D Konvoluutio, elu	128, (3,3)
2D Yhdistäminen, maksimi	–
2D Konvoluutio, elu	60, (1,1)
2D Yhdistäminen, keskiarvo	–
2D Konvoluutio, relu	52, (7,7)
2D Yhdistäminen, keskiarvo	–
2D Konvoluutio, elu	24, (5,5)
2D Yhdistäminen, maksimi	–
2D Konvoluutio, elu	28, (3,3)
2D Yhdistäminen, maksimi	–
2D Globaali yhdistäminen, maksimi	–
Tiheä kerros, relu	40, –
Tiheä kerros, tanh	72, –

Taulukko 5.10. *NAH löytämä paras neuroverkkoarkkitehtuuri ja hyperparametrit 0-6 ihmisen tapauksessa*

6. YHTEENVETO

Tämä työ käsitteli geneettisellä algoritmilla toteutettua neuroverkkoarkkitehtuurihakua. Työn motivaationa oli käytössä ollut pieni aineisto, joka mahdollisti geneettisen algoritmin käytön yksittäisen yksilön kelpoisuuden selvittämisen kestäessä geenin pituudesta riippuen keskimäärin $0.5 - 3$ min. Työssä esitetty evolutiivinen neuroverkkarkkitehtuurihaku oli itse toteutettu jatkokehittäen TTY:n Kaggle kisoissa menestyksellisesti toiminutta algoritmia [1]. Keskeisimpiä piirteitä oli käyttää peräkkäisiä ajoja, jotka kukin käyttivät kaikkia edellisissä ajoissa löytyneitä yksilöiden geenejä. Lisäksi käytettiin elitististä valintaa, joka mahdollisti parhaiden löydettyjen yksilöiden geenien jatkokehittämisen. Yksittäisen sukupolven lisääntymään päässet yksilöt valittiin kelpoisuusjärjestyksen mukaisesti. Uudet yksilöt muodostuivat geeneiltään puolittain käsittelyssä olevan sukupolven lisääntymään valituista yksilöistä. Toinen osuus uusien yksilöiden geeneistä tuli kaavan (4.2) antaman todennäköisyyden mukaan koko populaatiosta. Tämä johti hyvin elitistiseen nopeaan sukupolven konvergoitumiseen. Vastaelementtinä elitistiselle lisääntymispaineella toimi jokaisen ajon ensimmäisen täysin satunnaisista yksilöistä koostuvan aloitusopulaation käyttö. Tämä lisäsi käytössä olleeseen populaation geenipooliin diversiteettiä. Lisäksi liian nopeaa konvergoitumista vastaan käytettiin koko sukupolven mutantoitumista tapauksissa missä sukupolven kelpoisuuden keskihajonta alitti etukäteen annetun raja-arvon.

Opetusaineistona oli kolme erillaista tapausta. Ensimmäisessä 5.1 käytettiin yhtä ajoa. Tämän tarkoitus oli varmistaa käytetyn algoritmin toimivuus, vertaamalla sitä satunnaiseen populaatioon. Taulukossa 5.2 voidaan havaita, että toteutettu NAH oli satunnaista hakua parempia tuloksia antava. Ongelmana tässä ajossa oli kunkin populaation parhaiden yksilöiden nopea konvergoituminen raja-arvoon. Tämä oletettavasti johtui liian elitistisestä valinnasta.

Seuraavalla ajolla 5.2 käytettiin useiden peräkkäisten ajojen tekniikkaa. Taulukosta 5.5 voidaan havaita kunkin ajon tuottavan tuloksena aina kelpoisuudeltaan kasvavia yksilöitä. Tämä tuki opinnäytetyön hypoteesia, että toistamalla useita peräkkäisiä ajoja käyttäen satunnaisia aloituspopulaatioita on mahdollista saavuttaa satunnaista NAH:ia parempia tuloksia. Tämä tulos vahvistuu taulukon 5.7 vertailussa, jossa käytetty algoritmi saavutti selkeästi satunnaista hakua parempia tuloksia, niin parhaan yksilön kuin etsittyjen ratkaisujen keskiarvon suhteen.

Viimeisen ajon 5.2.1 tulokset olivat heikompia opinnäytetyön hypoteesin kannalta. Syynä voidaan pitää ajanpuutteen takia kahta edellistä ajoa pienempää populaatiokokoa. Taulukossa 5.9 voidaan havaita käytetyn algoritmin antavan vain marginaalisesti satunnaista hakua parempia tuloksia. Myös tässä tapauksessa toinen ajo antoi parhaan yksilön, päinvastoin kuin 5.2. Asiaan todennäköisesti vaikutti myös ajojen välillä kasvatettu yksilön mutaatiomahdollisuus. Tämän tarkoitus oli pyrkiä löytämään nopeammin mutaatiooperaattoria käyttäen jo löydetyistä yksilöistä parempia ratkaisuja ajanpuutteen takia.

Tässä opinnäytetyössä esitetty geneettisella algoritmilla toteutettu neuroverkkoarkkitehtuurihaku toimi jokaisessa tapauksessa paremmin kuin vastaava satunnainen neuroverkkoarkkitehtuurihaku. Tämä vahvisti opinnäytetyön kohteena olevaa hypoteesia. Käyttämällä peräkkäisiä ajoja joissa kussakin on satunnaisesti alustettu aloituspopulaatio ja elitististä valintaa, on mahdollista ylläpitää koko populaation geeniaineiston diversiteettiä.

LÄHTEET

[1] Halfar, T. *TAU Vehicle Type Recognition Competition*. 2019. URL: https://www.kaggle.com/competitions/vehicle/discussion/121883#696262 (viitattu 30. 07. 2022).

[2] Eiben, A. E., Smith, J. E. et al. *Introduction to evolutionary computing*. Vol. 53. Springer, 2015.

[3] Jong, K. A. D. *Evolutionary computation: a unified approach*. 1st. The MIT Press, 2002. ISBN: 9780262041942; 0262041944.

[4] Fogel, L., Owens, A. ja Walsh, M. *Artificial intelligence through simulated evolution*. Chichester, WS, UK: Wiley, 1966.

[5] Schwefel, H.-P. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Vol. 26. ISR. Basel/Stuttgart: Birkhaeuser, 1977, s. 390.

[6] Wright, S. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. *Proc. of the 6th International Congress on Genetics* 1 (1932), s. 356–366.

[7] Turing, A. M. Intelligent machinery (1948).

[8] Bremermann, H. J. et al. Optimization through evolution and recombination. *Self-organizing systems* 93 (1962), s. 106.

[9] Holland, J. H. *Adaptation in Natural and Artificial Systems*. second edition, 1992. Ann Arbor, MI: University of Michigan Press, 1975.

[10] Hutter, F., Kotthoff, L. ja Vanschoren, J. *Automated Machine Learning: Methods, Systems, Challenges*. 1st. Springer Publishing Company, Incorporated, 2019. ISBN: 3030053172.

[11] Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images (2009), s. 32–33. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[12] Zoph, B., Vasudevan, V., Shlens, J. ja Le, Q. V. *Learning Transferable Architectures for Scalable Image Recognition*. 2017. DOI: 10.48550/ARXIV.1707.07012. URL: https://arxiv.org/abs/1707.07012.

[13] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L. ja Dollár, P. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[14] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V. ja Dean, J. *Efficient Neural Architecture Search via Parameter Sharing*. 2018. DOI: 10.48550/ARXIV.1802.03268. URL: https://arxiv.org/abs/1802.03268.

[15] White, C., Neiswanger, W. ja Savani, Y. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search (2019). DOI: 10.48550/ARXIV.1910.11858. URL: https://arxiv.org/abs/1910.11858.

[16] Real, E., Aggarwal, A., Huang, Y. ja Le, Q. V. *Regularized Evolution for Image Classifier Architecture Search*. 2018. DOI: 10.48550/ARXIV.1802.01548. URL: https://arxiv.org/abs/1802.01548.

[17] Sheppard, C. *Genetic Algorithms with Python*. Clinton Sheppard, 2018. ISBN: 9781732029804. URL: https://books.google.fi/books?id=3jNqtAEACAAJ.

[18] Baker, J. E. Reducing Bias and Inefficiency in the Selection Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. Cambridge, Massachusetts, USA: L. Erlbaum Associates Inc., 1987, s. 14–21. ISBN: 0805801588.

[19] Bäck, T. Generalized Convergence Models for Tournament- and (mu, lambda)-Selection. *Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 15-19, 1995*. Toim. L. J. Eshelman. Morgan Kaufmann, 1995, s. 2–8.

[20] De Jong, K. A. *An analysis of the behavior of a class of genetic adaptive systems*. University Microfilms International, 1975.

[21] Eshelman, L. J., Caruana, R. ja Schaffer, J. D. Biases in the Crossover Landscape. *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989*. Toim. J. D. Schaffer. Morgan Kaufmann, 1989, s. 10–19.

[22] Wolpert, D. H. ja Macready, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1.1 (1997), s. 67–82.

[23] De Jong, K. A. ja Sarma, J. Generation Gaps Revisited. *Foundations of Genetic Algorithms*. Toim. L. D. WHITLEY. Vol. 2. Foundations of Genetic Algorithms. Elsevier, 1993, s. 19–28. DOI: https://doi.org/10.1016/B978-0-08-094832-4.50007-6. URL: https://www.sciencedirect.com/science/article/pii/B9780080948324500076.

[24] Kearney, J. K., Thompson, W. B. ja Boley, D. L. Optical Flow Estimation: An Error Analysis of Gradient-Based Methods with Local Optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.2 (1987), s. 229–244. DOI: 10.1109/TPAMI.1987.4767897.

[25] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[26] Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G. G. ja Tan, K. C. A Survey on Evolutionary Neural Architecture Search. *IEEE Transactions on Neural Networks and Learning Systems* (2021), s. 1–21. DOI: 10.1109/tnnls.2021.3100554. URL: https://doi.org/10.1109%2Ftnnls.2021.3100554.

[27] Xie, L. ja Yuille, A. *Genetic CNN*. 2017. DOI: 10.48550/ARXIV.1703.01513. URL: https://arxiv.org/abs/1703.01513.

[28] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. ja Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115.3 (2015), s. 211–252. DOI: 10.1007/s11263-015-0816-y.

[29] Simonyan, K. ja Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[30] He, K., Zhang, X., Ren, S. ja Sun, J. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[31] Huang, G., Liu, Z., Maaten, L. van der ja Weinberger, K. Q. *Densely Connected Convolutional Networks*. 2016. DOI: 10.48550/ARXIV.1608.06993. URL: https://arxiv.org/abs/1608.06993.

[32] *K-fold iterator variant with non-overlapping groups.* https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GroupKFold.html. Accessed: 2022-05-16.

[33] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. ja Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15.56 (2014), s. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[34] Datta, L. *A Survey on Activation Functions and their relation with Xavier and He Normal Initialization*. 2020. DOI: 10.48550/ARXIV.2004.06632. URL: https://arxiv.org/abs/2004.06632.

[35] Ioffe, S. ja Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015).

[36] Lin, M., Chen, Q. ja Yan, S. *Network In Network*. cite arxiv:1312.4400Comment: 10 pages, 4 figures, for iclr2014. 2013. URL: http://arxiv.org/abs/1312.4400.