TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

OSKARI POHJA
GEENIEKSPRESSIOLUOKITTIMEN TOTEUTUS KERAS-KIRJASTOLLA

Kandidaatintyö

Tarkastaja: Sami Hyrynsalmi
04.03.2018

TIIVISTELMÄ

OSKARI POHJA: Geenieksпressioluokittimen
toteutus Keras-kirjastolla
Tampereen teknillinen yliopisto
Kandidaatinyö, 25 sivua
Maaliskuu 2018
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Sami Hyrynsalmi
Avainsanat:Geenieksпressioluokitin, Konvolutiivinen neuroverkko, DeepChrome, Keras

Konvolutiivisten neuroverkkojen sovellutuksien määrä on kasvussa muiden koneoppimismenetelmien tavoin. Tämä kandidaatintyö pyrkii tutkimaan konvolutiivisen neuroverkkoarkkitehtuurin toteutusta Python-kielisellä Keras-kirjastolla. Toteutettava neuroverkko toimii luokittimena geenien korkealle ja matalalle aktiivisuudelle histonimodikaatiodatasta. Toteuttu arkkitehtuuri pohjautuu jo tunnettuun DeepChrome-arkkitehtuuriin.

Työ jakatuu kolmeen osaan. Ensimmäinen osa tutkii työssä vaadittavaa teoreettista taustaa, esittellen luokittimen rakenteen osineen. Toinen osa tutkii luokittimen toteutusta Keras-kirjastolla, tutkien toteutuksen yksityiskohtia pohjautuen ensimmäisen osan teoreettiseen taustaan. Kolmas osa käsittelee luokittimen optimointia ja tulosten validointia, missä luokittimelle tehdään optimointi etsien sille parhaan suorituskyvyn toteuttavat parametrit käyttäen ristiinhakua ja -validointia.

Tutkimus osoittaa Keraksen soveltuvan DeepChrome-luokittimen toteutukseen, sekä tutkitun luokittimen olevan suorituskyvyltään vastaava kuin alkuperäinen Lua-kielinen DeepChrome-toteutus vastaavalla data-aineistoilla. Tutkimus vahvistaa alkuperäisen DeepChrome-toteutuksen parametrivalinnat optimaaliselle luokittimen suorituskyvylle.

SISÄLLYS

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

Konvoluutio	Kahden funktion välille määritelty operaatio, joka tuottaa kolmannen funktion.
Kääreluokka	Luokka, joka toteuttaa rajapintamäärittelyn käyttäen toisen luokan toteutusta.
Luokitin	Järjestelmä, joka suorittaa kuvauksen näytearvoista luokka-arvoksi tai arvoiksi.
Luokitusvirhe	Luokitus, jonka tulos on väärä tai väärien lukitusten osuus kokonaisluokituksista.
Temporaalinen	Ajan hetkestä riippuva.
Ylioppiminen	Luokitin on harjoitettu liiaksi harjoitusdatalle, jolloin luokitin suoriutuu heikosti yleisestä luokitustehtävästä samantyyppiselle datalle.

1. JOHDANTO

Tekoäly ja sen osa-alue koneoppiminen ovat viime vuosina olleet kasvavan huomion kohteena akateemisessa maailmassa, mediassa ja politiikassa. Yksi osoitus tästä on pääministerin johtaman Tutkimus- ja innovaationeuvoston lausunto, jossa todetaan tekoälyn olevan *"yksi globaaleista megatrendeistä"* ja että *"[Suomen] Hallitus päätti puolivälitarkastelun yhteydessä elokuussa 2017 käynnistää tekoälyohjelman, jonka tavoitteena on, että Suomi on kärkimaa, joka kykenee soveltamaan tekoälyä nopeammin kuin kilpailijansa" [1].* Yksi mahdollinen syy kasvaavaan kiinnostukseen on ollut niin sanottujen syväoppivien neuroverkkojärjestelmien suorituskyvyn kasvu, joka on johtanut tehokkaiden sovellusten lisääntymiseen. Syitä suorituskyvyn kasvuun ovat muun muassa hyödynnettävän datan suuri lisääntyminen yhteiskuntien digitalisaation mukana, pilvipalveluiden tuoman laskentakapasiteetin kasvu, sekä neuroverkkojen toteuttamiseen suunnattujen ohjelmistokirjastojen kehittyminen ja yleistyminen [2].

Tämä kandidaatintyö käsittelee Keraksen — yksi edellä mainituista ohjelmistokirjastoista — käyttöä toimivan syväoppivan neuroverkon toteuttamiseksi. Työssä tutkitaan Keraksen hyödyntämistä syväoppivan konvolutiivisen neuroverkon toteuttamisessa. Konvolutiivisten neuroverkkojen erityisiä sovellusalueita ovat hahmontunnistus, suosittelujärjestelmät, sekä luonnollisen kielen käsittely. Keras tarjoaa rajapinnan konvolutiivisen neuroverkon toteuttamiseksi Python-ohjelmointikielellä, pyrkien mahdollistamaan neuroverkkojen nopean toteuttamisen ja jatkuvan testaamisen [5].

Toteutettava neuroverkko suorittaa luokituksia geenien aktiivisuudelle hyödyntäen syötteenään geenien histonimodifikaatiodataa. Toteutettu työ noudattaa toisella neuroverkkokirjastolla ja ohjelmointikielellä toteutetun *DeepChromen* arkkitehtuuria [3]. Työn luonne on siis ohjelmistotekninen, tutkien Kerksen käytön yksityiskohtia DeepChrome-arkkitehtuurin järjestelmää toteutettaessa.

Luku 2 käsittelee luokittimen toteutuksen teoreettista taustaa, arkkitehtuuria ja työn luokitteluongelmaa. Luku 3 käsittelee toteutusta käyttäen Keraksen rajapintaa. Luku 4 käsittelee toteutetun luokittimen suorituskyvyn validointia sekä sen

parametrien optimointia, listaten työssä optimoitavat muuttujat. Luku 5 esittelee työn tulokset. Luvussa 6 esitellään työn yhteenveto.

2. TEOREETTINEN TAUSTA

2.1 Luokitteluongelma

Toteutettavan luokittimen tehtävänä on luokitella alaluvussa 2.2 esiteltävästä geenin histonimodifikaatiodatasta geenin aktiivisuus joko *korkeaksi* tai *matalaksi*. Toisin sanoen, histonimodifikaatiosyötteet kartoitetaan kaksijakoisesti joko korkean tai matalan aktiivisuuden luokkaan.

Luokittimen toteuttaminen aloitetaan sen rakenteen määrittelyllä. Rakennemäärittelyn jälkeen luokitin käännetään, jolloin se voidaan kouluttaa käytettevällä datalla. Koulutetulla luokittimella voidaan suorittaa luokituksia tyypiltään koulutusdataa vastaavalle datalle. Luokittimen rakenne on kuvattu alaluvussa 2.3 ja koulutus- ja luokitusprosessi luvussa 3.

Luokittimen suorituskykyä P mitataan luokittimen suorittamien oikeiden luokituksien osuudella kaikista luokituksista. Luokitus on oikea, kun luokittimen antama luokitus vastaa koulutusdatan syötteessä määriteltyä luokkaa. Kaavalla

$$P = 1 - avg(\mathbf{y} - \mathbf{y}_0) \tag{2.1}$$

voidaan arvioida luokittimen suorituskykyä, kun vektori $\mathbf{y}$ sisältää luokittimen suorittamien luokitusten luokitusarvot ja vektori $\mathbf{y}_0$ sisältää oikeat luokitusarvot syötteille. Luokitusarvo on kokonaisluku, joko *0* tai *1*.

Luokittimen toimintaa säätelevät parametrit optimoidaan suorituskyvyn suhteen käyttäen ristiinhakua ja -validointa, jotka esitellään luvussa 4. Luokittimen lopulliseksi parametreiksi valitaan parhaan suorituskyvyn tuottava parameterijoukko.

2.2 Histonimodifikaatiodata syötteenä

Geenieksпressio kuvaa geenin ilmentymistä. Geenin ilmentyminen voi joko vaimentua tai lisääntyä, mitä kuvataan geenin aktiivisuudella matalaksi tai korkeaksi. Geenin aktiivisuuteen vaikuttavat epigeneettiset muutokset, kuten erilaisten histonimodifikaatioiden ilmentymät [11].

Tässä työssä geeni kuvataan sekvenssinä, joka sisältää tiedon geenin histonimodifikaatiota indikoivista histonisignaaleista [3]. Histonimodifikaatiodata on samassa muodossa kuin DeepChromen totetuksessa. Käytetty syötedata on koostettu *REMC*-tietokannasta kerätystä geenisekvenssidatasta. Syötedata on koottu valitsemalla geenin transkriptioalueen aloituskohdan ympäriltä 10 000 emäsparia (viisi tuhatta paria aloituskohdan molemmin puolin) jakaen ne 100 yhtäsuureen *emäsparilohkoon*, jolloin jokainen lohko sisältää 100 vierekkäistä emäsparia. Lohko sisältää tiedon emäsparisekvenssinsä histonimodifikaatisignaaleista, esitettynä positiivisena kokonaislukuna.

Alkuperäisestä datasta on valittu syötteeseen mukaan viisi olennaisinta histonimodifikaatiotyyppiä [3, s. 4]:

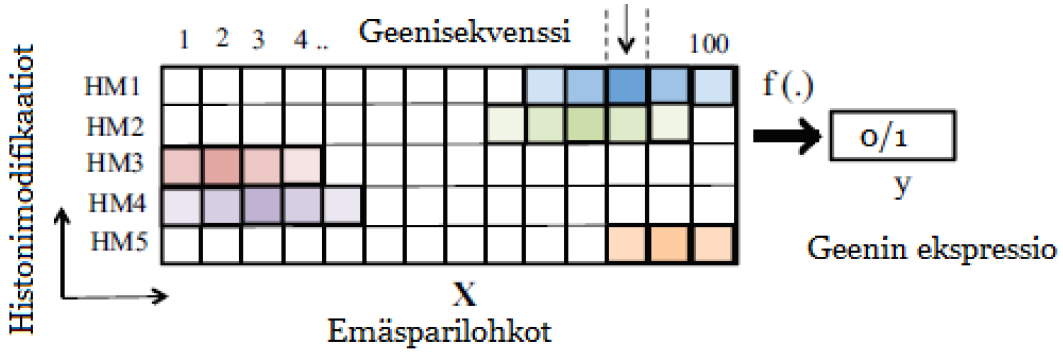
- H3K4me3,
- H3K4me1,
- H3K36me3,
- H3K9me3, ja
- H3K27me3.

Näin olleen syötteenä toimii 5×100 suuruisia tauluja sisältävä vektori, jossa taulujen sarakkeet kuvaavat emäsparilohkoa geenisekvensissä ja rivi histonimodifikaatiotyyppiä. Taulujen alkiot ovat kokonaislukuja, jotka kuvaavat alkion riviä vastaavan histonimodifikaation signaalin arvoa alkion saraketta vastaavassa emäsparilohkossa.

Syötettä vastaava luokitus geenin korkeaksi tai matalaksi ekspressioksi on diskretisoitu arvoiksi 0 ja 1, arvon 0 kuvatessa matalaa ekspressiota ja arvon 1 korkeaa ekspressiota. Diskretisointi on toteutettu asettamalla kynnysarvo geenityypin kaikkien geeninäytteiden ekspressioarvojen mediaaniin [3]. Näin ollen puolet näytteistä on korkeaekspressioisia ja puolet matalaekspressioisia. Syötteitä vastaavat luokitukset annetaan luokittimelle koulutusvaiheessa (alaluku 3.4) vektorina. Näytteen histonimodifikaatiotaulu ja sitä vastaava luokitus ovat omissa vektoreissaan samalla järjestyspaikalla. Kuva 2.1 havainnollistaa datan rakennetta.

DeepChromen esitellyt tutkimus koulutti luokittimensa geeneille, jotka on näytteistetty 56 eri solutyypistä [3]. Tämän työn luokitin koulutettin vain yhdelle edellä mainitun REMC-tietokannan solutyypille: *E047, CD8_Naive_Primary-_Cell*.

Tässä työssä käytetään 15 485 geeninäytettä aktiivisuusluokituksineen tallennettuna CSV-tiedostoihin. Data jaetaan kolmeen yhtäsuureen joukkoon: *koulutus*-,



Kuva 2.1 *Käytetyn datan rakenne [3, s.2]*

validointi- ja *testijoukkoon.* Syötettäessä luokittimelle, validointi- ja testijoukosta on poistettu syötteitä vastaavat luokat, mikä mallintaa luokitustilannetta, jossa syötteen oikeaa luokkaa ei tiedetä.

Määritelty luokitin koulutetaan käyttäen koulutusjoukkoa ja validointijoukkoa. Koulutusjoukko sisältää syötteet sekä syötteitä vastaavat luokitukset. Järjestelmä käyttää koulutusjoukkoa luokittimen sisäisten parametrien säätöön koulutusvaiheessa. Validointijoukkoa käytetään koulutusvaiheen aikana arvoimaan luokittimen suorituskykyä muuttamatta sen parametreja.

Koulutusvaiheen jälkeen luokittimen suorituskykyä arvoidaan käyttäen syötteenä testijoukkoa. Luokittimen antamia luokituksia verrataan testijoukossa määriteltyihin luokkiin. Tällä prosessilla varmistetaan se, että luokittimen suorituskykyä arvoidaan datalla, jota sille ei ole aiemmin syötetty.

2.3 Luokittimen arkkitehtuuri

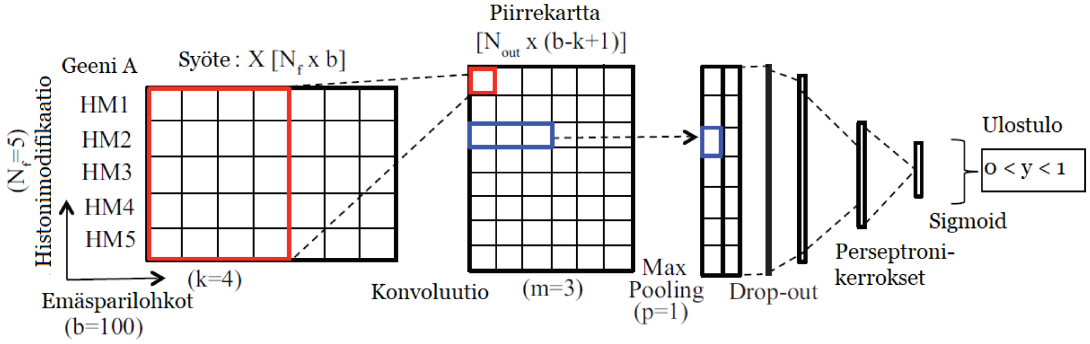
Työssä toteutettu järjestelmä jäljittelee DeepChrome-arkitehtuuria. DeepChrome koostuu peräkkäisistä luokittimen logiikkaa toteuttavista kerroskomponenteista [3]. Arkkitehtuuri on kuusikerroksinen ja sekventaalinen. Tässä yhteydessä sekventialisuus tarkoittaa rakennetta, jossa kerroskomponentit syöttävät oman ulostulonsa seuraavalle syötteeksi, poislukien viimeinen kerros, joka toimii koko luokittimen ulostulona. Sekventialisuutta voidaan kuvata kaavalla

$$\mathbf{y}_0 = (f_7 \circ f_6 \circ f_5 \circ f_4 \circ f_3 \circ f_2 \circ f_1)(\mathbf{X}) \tag{2.2}$$

jossa funktioita vastaavat kerrokset ovat:

1. konvoluutiokerros
2. "MaxPooling"-kerros
3. "Flatten"-kerros
4. "Dropout"-kerros
5. 1. täysinkytketty perseptronikerros
6. 2. täysinkytketty perseptronikerros
7. ulostulokerros

Hahmotelma arkkitehtuurin toimintalogiikasta on esitelty kuvassa 2.2.



Kuva 2.2 *DeepChrome-arkkitehtuurimallin toimintalogiikka [3, s.4]*

Järjestelmän sisäisten parametrien, kuten konvoluutiokerroksen ja perseptronikerroksien kertoimet ovat koulutusvaiheessa säädettäviä arvoja. Järjestelmän kertoimia muutetaan koulutusvaiheessa käyttäen *optimointialgoritmtia*, kuten *Stochastic Gradient Descent* tai *Adam*, joka pyrkii minimoimaan kohdefunktionsa, *loss*-funktion.

2.4 Luokittimen komponenttien looginen toiminnallisuus

Tämän luvun alaluvuissa annetut esittelyt ja perustelut pohjautuvat DeepChrome-työhön [3]. Alaluku 2.4.1 esittelee konvoluutiokerroksen teoreettisen taustan, alaluvun 2.4.2 esitellessä muiden luokittimen kerroksien teoreettisen taustan.

Kerrosten teoreettista taustaa selvitetään luvussa vain sen verran, kuin luokittimen Keras-toteutuksen kannalta on oleellista. Kerroksien toiminnallisuus sisältää paljon yksityiskohtia, joita ei tässä työssä käsitellä taustan laajuden vuoksi.

2.4.1 Konvoluutiokerros

Konvoluutiokerros toimii luokittimen ensimmäisenä kerroksena saaden syötteenään alaluvussa 2.2 esiteltyä histonimodifikaatiodataa. Konvoluutiokerros suorittaa konvoluutioita syötteellä ja kerroksen suodinmatriiseilla, antaen ulostulonaan piirrekarttatauluja.

Konvoluutiokerroksen toiminnallisuus koostuu kolmesta osatekijästä:

- **Suotimet**, eli joukko konvoluutiomatriiseja. Matriisin arvot ovat konvoluutiokertoimia syöttelle. Oppimisvaiheessa järjestelmä säätää näitä arvoja. Matriisien arvot säilyvät samana riippumatta konvoluutioikkunan sijainnista.
- **Konvoluutioikkuna**, joka määrittää suotimien ulottovuuden. Konvoluutioikkuna rajaa kunkin konvoluutioon valittavan alijoukon syötematriisista.
- **Aktivointifunktio**, joka ottaa syötteenään konvoluution tuloksen. Aktivointifunktion ulostulo asetetaan *piirrekarttaan*.

Konvoluutiokerros suorittaa konvoluutio-operaatiot syötematriisin alkioille jokaista suodinmatriisia vasten konvoluutioikkunan alueelta, liu'uttaen konvoluutioikkunaa syötematriisin sijaintiulottouvuuden yli. Konvoluutioikkunan liu'utus tarkoittaa ikkunan siirtämistä sekvenssijaksojen suunnassa hypyn h määrän. Jokaisen konvoluution ulostuloarvo syötetään aktivointifunktiolle, jonka tulos tallennetaan *piirrekarttaan*.

Järjestelmän konvoluutiokerros on *temporaalinen*, jossa konvoluutioikkunaa liu'utetaan vain yhden ulottovuuden yli [3, s.4]. Järjestelmässä temporaalisen konvoluution ikkuna kattaa syötteen kaikki rivit (histonimodifikaatiotyypit) ja konvoluutioikkunan pituus k määrittelee ikkunan suuruden sekvenssijaksojen suunnassa, joka on myös konvoluutioikkunan liukumasuunta.

Järjestelmän rakentamisessa tarvittavia parametreja konvoluutiokerrokselle ovat konvoluutioikkunan pituus k, liukuman pituus h, syötetaulun täydennyksen tyyppi ja suotimien määrä N_{out}. Nämä parametrit vaikuttavat ulostulona annettavien piirrekarttojen kokoon sekä määrään [4, s. 327–335]. Konvoluutiokerros tuottaa ulostulonaan piirrekartan kooltaan $N_{out} \times (b-k+1)$, jossa N_{out} on suotimien lukumäärä, b syötteen pituus (emäsparilohkojen lukumäärä) ja k konvoluutioikkunan pituus, kun $h = 1$ [3].

2.4.2 Muut kerrokset

MaxPooling-kerroksen tehtävänä on suodattaa konvoluutiokerroksen tuottamista piirretauluista suurimmat arvot vähentäen piirretaulun kokoa ja laskentakuormaa, samalla tuoden esiin merkittävimmät arvot. Kuten konvoluutio, pooling on myös temporaalinen: operaatio suoritetaan sekvenessijaksojen suunnassa, joka on säilynyt piirrekarttoihin alkuperäisestä syötteestä. MaxPooling valitsee ulostuloonsa suurimman arvon valitulta alueelta, *pooling*-ikkunasta. Kuten konvoluutiossa, ikkunaa liu'utetaan sekvenssijaksojen suuntaisesti. Pooling-ikkuna kattaa m-pituisen alueen jokaiselta piirrekartan riviltä, tuottaen ulostulonaan $N_{out} \times \left\lfloor \frac{(b-k-1)}{m} \right\rfloor$ kokoisen matriisin, missä b ja k ovat konvoluutiokerroksen parametreja kun konvoluutio- ja poolingikkunnan siirron suuruus on 1.

Flatten-kerros uudelleenjärjestelee MaxPooling-kerroksen ulostulomatriisin yksiuloitteiseksi vektoriksi, joka voidaan myöhemmin antaa perseptronikerrokselle syötteenä. *Dropout*-kerros muuttaa osasta flatten-kerroksen ulostulovektorin alkioista arvon nollaksi, minkä päämääränä on hallita järjestelmän ylioppimista.

Perseptronikerrokset koostuvat kukin N_{per} *perseptronista*, joilla on omat syötevektorin kokoiset kerroinvektorinsa. Perseptronikerroksia on arkkitehtuurissa kolme. Ensimmäinen perseptronikerros saa Dropout-kerroksen tuottaman vektorin syötteekseen. Perseptronikerroksen jokainen perseptroni saa syötteenään syötevektorin alkiot suorittaen pistetulon syötteen ja kerroinvektorin välillä syöttäen pistetulon *aktivointifunktiolle*, jonka ulostulo on perseptronin ulostulo. Jokaisen perseptronin ulostulo tallennettaan *ulostulovektoriin*, kooltaan N_{per}. Perseptronin ulostulo uvoidaan esittää kaavalla

$$y = f_{activation}(\mathbf{w} * \mathbf{x}) \tag{2.3}$$

missä y on perseptronin ulostulo, $f_{activation}$ on perseptronin aktivointifunktio, $\boldsymbol{w}$ on perseptronin kerroinvektori ja $\boldsymbol{x}$ on kerroksen syötevektori. Järjestelmä säätää koulutusvaiheessa kerroinvektorin $\boldsymbol{w}$ arvoja. Järjestelmän toinen perseptronikerros saa syötteenään ensimmäisen kerroksen ulostulovektorin suorittaen vastaavan operaation syötteellään kuin ensimmäinen kerros.

Järjestelmän viimeinen kerros, *luokituskerros*, on yksittäinen perseptroni joka suorittaa toisen perseptronikerroksen ulostulovektorille kaavan 2.3 operaation. Luokitinkerroksen aktivointivointifunktio *sigmoid* asettaa ulostulon arvojen 0 ja 1 välille, antaen todennäköisyyden geenin korkealle aktivointiasteelle, joka on myös luokittimen ulostulo.

3. LUOKITTIMEN TOTEUTUS KÄYTTÄEN KERASTA

Keras on Python-ohjelmointikielellä toteutettu kirjasto, joka tarjoaa rajapinnat neuroverkkojärjestelmien toteutukseen. Keras on modulaarinen, eli järjestelemä kootaan erillisistä kerrosobjekteista. Tässä luvussa ja sen alaluvuissa esitellään *DeepChrome*-arkkitehtuurin toteutus käyttäen Keraksen rajapintoja.

Keraksessa ei itsessään toteuteta raskasta matriisilaskentaa; vaativassa laskennassa hyödynnetään toisia kirjastoja. Vaihtoehtoja matriisilaskentaa toteuttavaksi kirjastoksi on kolme: TensorFlow, Thaeno ja CNTK. Keraksen rajapinta säilyy samana huolimatta kirjastovalinnasta, jolloin se ei vaikuta neuroverkon määrittelyyn ja toteutukseen ohjelmoijan kannalta, mistä johtuen niiden eroja ei tässä työssä käsitellä.

Luokittimen määrittelyn ja käytön pääsasialliset vaiheet koostuvat:

1. neuroverkon rakenteen määrittelystä käyttäen *Sequential*- tai *Model* -malliluokkia ja luokittimen rakennetta kuvavaavia kerrosobjekteja,
2. mallin kääntämisestä *compile*-metodilla,
3. mallin kouluttamisesta *fit*-metodilla käyttäen syötteistä ja niitä vastaavista luokittelusta koostuvaa opetusdataa, ja
4. ennusteen tai luokittelun suorittamisesta syötteelle *predict*-metodilla.

Tämän luvun alaluvut sisältävät toteutuksessa vaadittujen metodien esittelyt olennaisine parametreineen. Käytetyt tietotyypit ovat Python-ohjelmointikielen tietotyyppejä. Työn toteutuksessa luokittimelle on pyritty löytämään optimaaliset parametrit käyttäen ristiinvalidointia ja -sovitusta. Optimoinnin yhteydessä käytetyt parametrit on lueteltu taulussa 4.2.

Alaluvussa 2.4 esitellyt luokittimen komponentit määritellään Keraksessa kerrosobjekteina eli *layereinä*. Keraksen kerroskomponentit sijoitetaan luokitinrajapinnan tarjoavaan luokitinolioon, joka esitellään alaluvussa 3.1.

3.1 Sequential-luokitinmalli ja luokittimen rakenteen määrittely

Keraksessa luokitin toteutetaan rakentamalla ja lisäämällä alaluvussa 3.2 esiteltävien kerrosluokkien instansseja (*kerrosobjektieja*) luokitinmalli-luokan instassin (*luokitin-objektiin*). Keras tarjoaa kaksi luokitinmalliluokkaa: *Sequential* ja *Model*, joilla molemmilla voidaan toteuttaa suunniteltu luokitin.

Luokitinmallin tehtävänä on tarjota:

- säiliö kerrosoliolle,
- välitys kerrosten syötteille ja ulostuloille oikeassa muodossa,
- rajapintakutsu luokittimen hyperparametrien asettamiselle,
- rajapintakutsu luokittimen kouluttamiselle, ja
- rajapintakutsu luokitusten tekemiselle.

Sequential-luokka on valittu työhön koska sen tarjoama rajapinta soveltuu sekventiaalisen mallin (kaava 2.2) luomiseen, eikä ohjelmoijan tarvitse huolehtia lisättävien kerroksien syötteen rakenteen määrittelystä. Keraksessa Sequential-luokitinobjekti luodaan kutsumalla luokan rakentajaa

```
model = Sequential()
```

joka palauttaa tyhjän luokitinobjektin.

Luotuun malliin voidaan lisätä kerrosobjekteja luokitinmallin rakentajassa ja luokitinmallin *add*-metodissa. Kerroksien looginen järjestys malliobjektissa vastaa niiden syöttöjärjestystä. Sequential-luokan rakentajassa lisätään kerrosobjektit parametreina. Esimerkiksi rakentajakutsu

```
model = Sequential([
    Dense(32, input_shape=(784,0), Activation('relu')),
    Dense(10, Activation('softmax')
]),
```

palauttaa luokitinobjektin, joka sisältää kaksi määriteltyä Dense-kerrosobjektia.

Kerrosobjekteja voidaan lisätä luokitinmalliobjektiin myös add-metodilla

```
model = Sequential()
model.add(Dense(32, input_shape=(784,),
        activation='relu'))
model.add(Dense(10, activation = 'softmax')),
```

jolle syötetään parametreina kerrosobjekti. Kerrosobjektin parametrit syötetään kunkin kerrosluokan rakentajalle.

Luokittimelle on määriteltävä syötteen muoto. Tämä toteutetaan luokittimen ensimmäisen kerroksen rakentajassa antamalla syötteen muodon määrittävä *input_shape*-parametri. Kaksi ja kolmiuloitteisille kerroksille voidaan syötteen muoto määritellä vaihtoehtoisesti *input_dim*-parametrillä, mutta luokittimen ensimmäisen kerroksen ollessa yksiuloitteinen konvoluutiokerros käytetään *input_shape* -parametriä, eikä *input_dim*-parametriä käsitellä tämän työn puitteissa.

3.2 Luokitinkerrokset

Luokittimen kerroskomponentit määritellään ja luodaan Keraksessa kutsumalla kerrosluokan rakentajaa, joka palauttaa kerroksen toteuttavan objektin. Kerroksen toiminnallisuutta määrittelevät parametrit syötetään kerrosluokan rakentajalle.

Alaluvut 3.2.1–3.2 esittelevät työn luokittimen toteutuksessa tarvitut kerrosluokat tarvittavine parametreineen. Esiteltyjen parametrien tietotyypit vastaavat Python-ohjelmointikielessä määriteltyjä tyyppejä.

3.2.1 Konvoluutiokerros

Alaluvussa 2.4.1 kuvattu temporaalinen konvoluutio toteutaan Keraksessa käyttäen Conv1D-kerrosluokkaa. Kerrosobjekti luodaan kutsumalla Conv1D-luokan rakentajaa

```
keras.layers.Conv1D(filters, kernel_size, ... )
```

asettaen sille taulukossa 3.1 esitellyt parametrit.

Taulukko 3.1 *Työssä tarvittavat Conv1D-luokan parametrit*

Nimi	**Tyyppi**	**Pakollinen**	**Vakioarvo**	**Kuvaus**
filters	Integer	Kyllä	Pakollinen parametri	Konvoluutiosuodinten lukumäärä
kernel_size	Tuple (Integer, Integer)	Kyllä	Ei	Konvoluutioikkunan pituus
strides	Integer, Tuple tai List	Ei	1	Konvoluutioikkunan askeleen pituus
padding	String	Ei	'valid'	Syötteen täytetyyppi
activation	String	Ei	None	Aktivointifunktio

Koska Conv1D on luokittimen ensimmäinen kerros, on sen rakentajassa määriteltävä syötteen muoto. Alaluvussa 2.2 todettiin syötteen olevan 5x100-muotoinen matriisi, joten luokan rakentajalle tulee asettaa `input_shape=(5,100)` -parametri.

Parametrien vaikutus kerroksen toiminnallisuuten on kuvattu alaluvussa 2.4.1. Parametri *filters* vastaa muuttujaa N_{out}, parametri *kernel_size* vastaa muuttujaa k. Parametrien *filters* ja *kernel_size* arvot optimoitiin ristiinhaulla arvojoukosta, joka on esitetty taulukossa 4.2. Parametrivalinnassa noudatettiin DeepChromen alkuperäistä toteutusta. Parametri *padding* asetetaan arvoon *'valid'*, jolloin syötedataan ei lisätä täytekenttiä. Aktivointifunktioksi asetettiin *'ReLU'*, eli *Rectified linear units* ja muuttujaa h vastaava parametri *strides* asetettin arvoon *1* [6].

3.2.2 MaxPooling-kerros

MaxPooling1D-luokka toteuttaa alaluvussa 2.4.2 kuvatun Pooling-operaation. Järjestelmässä käytetään *MaxPooling1D* -luokkaa, koska edellinen konvoluutiokerros ei poista datan temporaalisuutta. [3, s. 5]

Kerrosobjekti luodaan kutsumalla sen rakentajaa

```
keras.layers.MaxPooling1D(pool_size=2, strides=None,
padding='valid')
```

asettaen sille taulukossa 3.2 esitellyt parametrit.

Taulukko 3.2 *Työssä tarvittavat MaxPooling-luokan parametrit.*

Nimi	**Tyyppi**	**Pakollinen**	**Vakioarvo**	**Kuvaus**
pool_size	Integer	Ei	2	Pooling-ikkunan koko
strides	Integer	Ei	None	Pooling-ikkunan hypyn pituus
padding	String	Ei	'valid'	Syötteen täytetyyppi

Parametrien arvot asettiin vastaamaan DeepChrome-toteutuksen arvoja [3]. Parametri *pool_size* vastaa alaluvun 2.4.2 arvoa m, määritellen pooling-ikkunan koon. Vastaavasti parametri *strides* vastaa Pooling-ikkunan siirron suuruutta ja se asetettiin arvoon 1, jolloin pooling ikkunaa siirretään yksi hyppy sekvenssijaksojen suunnassa jokaisen pooling-operaation päätteeksi. Syötteeseen ei lisätä täytettä, joten parametrin *padding* arvoksi asetettiin *'valid'* [6]. Parametri *pool_size* optimoitiin ristiinhaussa arvojoukosta, joka on esitetty taulussa 4.2.

3.2.3 Flatten-kerros

Flatten-kerros järjestelee sisääntulonaan saamansa MaxPooling-kerroksen tiivistämät piirrekartat yksiulotteiseksi vektoriksi, jotta se voidaan syöttää perseptronikerroksille. Flatten-kerrosobjekti luodaan kutsumalla luokan rakentajaa

```
keras.layers.Flatten()
```

jolle ei aseteta lainkaan parametreja.

3.2.4 Dropout -kerros

Dropout-luokka toteuttaa alaluvussa 2.4.2 esitellyn Dropout-kerroksen. Dropout-objekti luodaan kutsumalla sen luokan rakentajaa

```
keras.layers.Dropout(rate, ... )
```

asettaen taulukossa 3.3 kuvatut parametrit.

Taulukko 3.3 *Työssä tarvittavat Dropout-luokan parametrit*

Nimi	Tyyppi	Pakollinen	Vakioarvo	Kuvaus
rate	Float, väliltä 0-1	Kyllä	Pakollinen parametri	Nollattavien syötteiden osuus syötteestä, väliltä 0-1.

Kuten DeepChrome-toteutuksessa, puolet kerroksen syötteen arvoista asetetaan nollaan, joten parametrin *rate* arvoksi asetetaan *0.5* [6].

3.2.5 Dense-kerrokset

Toteutettava arkkitehtuuri sisältää kaksi täysinkytkettyä perseptronikerrosta. Keraksen *Dense*-luokka toteuttaa perseptronikerroksen, jonka teoreettinen tausta on esitelty alaluvussa 2.4.2.

Dense-objekti luodaan kutsumalla luokan rakentajaa

```
keras.layers.Dense(units, ... )
```

asettaen sille taulukossa 3.4 esitellyt parametrit.

Taulukko 3.4 *Työssä tarvittavat Dense-luokan parametrit*

Nimi	Tyyppi	Pakollinen	Vakioarvo	Kuvaus
units	Integer	Kyllä	Pakollinen parametri	Perseptronien lukumäärä kerroksessa.
activation	String	Ei	None	Perseptronien aktivointifunktio
use_bias	Boolean	Ei	True	Bias-vektorin käytön aktivointi

Parametri *units* vastaa alaluvussa 2.4.2 esitettyä muuttujaa N_{per}, eli perseptronien määrää kerroksessa. Kerrokselle asetetut parametrit seuraavat DeepChromen alkuperäistä toteutusta [6]. Ensimmäisen perseptronikerroksen *units*-parametri asetettiin arvoon 625, toisen kerroksen vastaava parametri asetettin arvoon 125. Molemmille kerroksille määritellään työn totetuksessa aktivointifunktioksi *'ReLU'*[3].

3.2.6 Luokituskerros

Alaluvussa 2.4.2 esitelty luokituskerros toteutetaan työssä käyttäen alaluvussa 3.2.5 esiteltyä *Dense*-luokkaa. Luokituskerroksen ulostulo on luokittimen ulostulo, todennäköisyys arvona väliltä 0-1.

Luokituskerros toteutetaan asettamalla aktivointifunktioparametrin *activation* arvoksi *'sigmoid'*. Kerroksessa on vain yksi perseptroni, joten parametri *units* asetetaan arvoon *1*. Toteutus eroaa alkuperäisestä *DeepChromen*-toteuksesta, jossa luokituskerroksessa on kaksi perseptronia käyttäen *'Softmax'*-aktivointifunktiota [6]. *Deep-Chromen*-ulostulokerros antaa todennäköisyyden molemmille luokille, *geenin korkealle* ja *matalalle* aktiivisuudelle. Lopputuloksen kannalta eroa ei ole, koska luokitus on binäärinen, jolloin pelkkä todennäköisyys geenin korkealle aktivaatiolle riittää luokitusvalinnan tekemiseksi. Työssä luokituskerros on totetutettu kuvatulla tavalla, koska ulostulon käsittely on mielekkäämpää ohjelmointitoteutusta tehdessä, kun käytetään luvussa 4.2 esiteltyä kääreluokkaa.

3.3 Luokittimen kääntäminen

Luokittimen kerroksien määrittelyn jälkeen tulee malli kääntää asettaen luokittimen *parametrit*. Parametreilla määritellään mallille optimointifunktio, loss-funktio ja luokituksen metriikka.

Optimointifunktioparametrilla määritellään algoritmi, joka optimoi luokittimen kertoimet koulutusvaiheessa. Keraksessa on käytettävissä lukuisia optimointiluokkia, joita ei tässä työssä tarkastella pidemmälle. Keraksen dokumentaatio tarjoaa listan käytettävistä optimointifunktiosta [5]. Loss-funktioparametri määrittelee optimoinnin kohdefunktion. Luokituksen metriikka -parametrilla määritellään loukittimen suorituskyvyn mittaustapa.

Hyperparametrit määritellään luokitinmallin *compile*-metodissa

```
model. compile(optimizer='SGD',
               loss='binary_entropy',
               metrics=['accuracy'])
```

jolle asetetaan taulukossa 3.5 esitellyt parametrit.

Taulukko 3.5 *Työssä tarvittavat Compile-metodin parameterit*

Nimi	**Tyyppi**	**Pakollinen**	**Vakioarvo**	**Kuvaus**
optimizer	string tai optimizer-objekti	Kyllä	Pakollinen parametri	Optimisointifunktion nimi string-objektina tai optimizer-objekti
loss	string tai optimizer-objekti	Kyllä	Pakollinen parametri	Lossfunktion nimi string-objektina tai loss-objekti.
metrics	List-objekti	Ei	None	Lista luokittimen arviointiin käytettävästä metriikasta

Työssä asettiin luokittimen optimizer-arvoksi *'SGD'* eli *stochastic gradient descent* -algoritmi sen vakioparametreilla, kuten DeepChromen alkuperäisessä toteutuksessa [3]. Loss-funktioksi valittiin *'binary_ crossentropy'*, sen soveltuessa binääriseen todennäköisyysluokitukseen [9]. Luokittimen metriikaksi asetettiin *['accuracy']*, joka arvioi luokittimen suorituskyvyn oikeiden luokituksien ja luokituksien kokonaismäärän suhteena.

3.4 Luokittimen opettaminen

Luokan määrittelyn jälkeen voidaan luokitin kouluttaa *fit*-metodilla. Fit käyttää suorituksen aikana *compile*-metodissa määriteltyä *optimizer*-algoritmia ja *loss*-funktiota optimoiden luokittimen kerrosten sisäiset parametrit siten, että määritelty metriikka antaa parhaan tuloksen opetusdatalla.

Fit on Sequential-luokan metodi, jota kutsutaan luokitinmalliobjektin vitteen kautta

```
model.fit( ... )
```

asettaen taulukossa 3.6 esitellyt parametrit.

Taulukko 3.6 *Työssä tarvittavat Fit-metodin parametrit*

Nimi	**Tyyppi**	**Pakollinen**	**Vakioarvo**	**Kuvaus**
x	Numpy-array	Ei	None	Syötearvot sisältävä taulu
y	Numpy-array	Ei	None	Syötteitä vastaavat luokitukset sisältävä taulu
batch_size	Int tai None	Ei	32	Syötteiden lukumäärä ennen gradientin päivitystä
epochs	Int	Ei	1	Koulutusiteraatioden määrä, eli arvo, joka määrittää kuinka monesti harjoitusdata ajetaan luokittimessa
validation _split	Float väliltä 0-1	Ei	0.0	Validointiin käytettävän syötejoukon suuruden osuus opetusdatasta
validation _data	Tuple (syötteet, luokitukset)	Ei	None	Validointiin käytettävä syötejoukko, ei käytetä koulutuksessa. Vaihtoehtoinen *validation_ split*-paramterille.

Fit-metodi ottaa parametreinaan opetusdatan syötteet parametrissa x ja syötettä vastaavien luokitusten listan parametrissa y. Syötteiden ja syötteitä vastaavien luokitusten tulee olla omissa tauluissaan, samassa järjestyksessä. Työssä käytetyn syötedatan muoto on määritelty alaluvussa 2.2. Syötedatana annetun taulun koon on vastattava *input_ size*-parametria, joka on määritelty luokittimen ensimäisessä kerroksessa.

Koulutuksen aikana Keras syöttää luokittimelle parametrin *batch_ size* kokoisen syöte-erän, jonka jälkeen luokittimen kerrosten sisäisiä parametreja säädetään valitun optimointialgoritmin ja loss-funktion mukaisesti [5].

Jokaisen opetusjakson jälkeen Keras arvioi luokittimen suorituskykyä käyttäen opetuksen ulkopuolelle jätettyä datajoukkoa. Tämä validointijoukko erotetaan koulutusdatasta ottaen *validation_ split*:in suuruinen osuus tai vaihtoehtoisesti käyttämällä erikseen määritettyä validointidatajoukkoa, joka on määritelty *validation_ data*-parametrissa. Työssä käytetään *validation_ split* parametria ar-

volla 0.33, jolloin *koulutusdata*, *validointidata* ja *testidata* sisältävät yhtäsuuren määrän näytteitä.

Luokittimen koulutuksen vaiheet ovat seuraavat:

1. Ladataan syötetaulu x ja luokitustaulu y.
2. Käytettäessä validation_split parametria, valitaan satunnainen osuus koulutusdatasta validointidataksi. Käytettäessä validation_set parametria, ladataan parametrissa määritelty validointidata. Ladattua validointidataa ei käytetä vaiheissa 3-4.
3. Syötetään syötetalusta *batch_size*:n määrittämä määrä syötteitä luokittimelle.
4. Säädetään luokittimen kertoimet käyttäen annettua optimointialgoritmia, lossfunktiota.
5. Toistetaan vaiheet 4-5, kunnes koko koulutusdata on syötetty luokittimelle.
6. Arvoidaan luokittimen suorotuskyky osavaiheen päätteksi, käyttäen validointidataa.
7. Toistetaan vaiheet 3-7 parametrin *epochs* määrän.

Koko opetusdatajoukko syötetään luokittimelle koulutusvaiheessa useasti. Yhtä tällaistä itearaatiota voidaan kutsua *koulutusjaksoksi*. Opetusvaiheessa suoritettavien koulutusjaksojen määrä määritellään *epochs*-parametrilla. Koulutusjaksojen määrää on syytä säädellä, sillä riskinä liian monella koulutusjaksolla on ylioppiminen, kun taaseen liian pienellä jaksomäärällä ei päästä parhaaseen mahdolliseen suorituskykyyn [8].

3.5 Luokittelu

Opetusvaiheen jälkeen voidaan luokittimella toteuttaa luokituksia syötteelle, jonka luokka ei ole tiedossa. Luokitukset suoritetaan kutsumalla luokitinmalliluokan metodia *predict*

```
model.predict(x, ... )
```

määritellen sille taulukossa 3.7 esitellyt parametrit.

Taulukko 3.7 *Työssä tarvittavat Predict-metodin parametrit*

Nimi	**Tyyppi**	**Pakollinen**	**Vakioarvo**	**Kuvaus**
x	Numpy-array	Kyllä	Pakollinen parametri	Syötearvot sisältävä taulu
batch_size	Ei	None	None	Syöte-erän suuruus

Parametri x sisältää syötteet, joille luokittelut suoritetaan *batch_size*:n määrittämän kokoisissa erissä. Metodi palauttaa numpy-taulun, joka sisältää arvion syötteen geenien korkean aktivaation todennäköisyydelle. Koska toteutetun järjestelmän ulostulokerros on yhden perspetronin *sigmoid*-aktivointifunktiota käyttävä kerros, on järjestelmän ennuste geenin aktiivisuudelle väliltä 0–1.

4. LUOKITTIMEN OPTIMOINTI JA TULOSTEN VALIDOINTI

4.1 Ristiinvalidointi ja ristiinhaku

Optimaalisen luokittimen suorituskyvyn saavuttamiseksi luokittimen parametreille tulee suorittaa *ristiinhaku* ja suorituskyvyn validointiin *ristiinvalidointi*.

Ristiinvalidoinnissa pyrkimyksenä on suorituskyvyn todentaminen datalla, jota luokittimelle ei ole vielä syötetty, jolloin simuloidaan tilannetta, jossa syötteen todellista luokkaa ei tiedetä. Ristiinvalidoinnissa kuitenkin tiedetään todelliset luokat, jotka ovat piilossa luokittimelta jolloin suorituskykyä voidaan arvioida. Ristiinvalidoinnissa harjoitusdata jaetaan n:ään osaan. Tämän jälkeen luokitin koulutetaan *n-1* osalla koulutusdatasta. Kouluttamisen jälkeen suoritetaan tulosten validointi jäljelle jääneellä harjoitusdatan osalla, jota kutsutaan ristiinvalidoinnin yhteydessä validointidataksi. Validointidata syötetään luokittimelle luokiteltavaksi, jolloin luokitin ei huomioi validointisyötteen todellista luokkaa. Luokitustuloksia verrataan todellisiin luokkiin. Näistä kahdesta joukosta voidaan laskea luokittimelle prosentuaalinen tarkkuus jakamalla oikeiden luokitusten lukumäärä validointijoukon koolla. Luotettavuuden lisäämiseksi ristiinvalidointiprosessi suoritetaan k-kertaa, valittaessa validointidataksi joka kerralla eri harjoitusdatan joukko. Tätä prosessia kutsutaan k-kertaiseksi ristiinvalidoinniksi [10].

K-kertaisen ristiinvalidoinnin ollessa suorituskyvyn mittari, voidaan optimoida luokittimen suorituskyky säätämällä parametrijoukko sellaiseksi, joka antaa parhaan tuloksen k-kertaisesta validoinnista. *Ristiinhauksi* määritellään tässä työssä operaatio, jossa luokittimen kaikista säädettävistä parametreista luodaan kaikki kombinaatiot ja niille suoritetaan koulutus ja validointi k-kertaisella ristiinvalidoinnilla. Ristiin haun tulokseksi valitaan se kombinaatio, joka tuottaa parhaan ristiinvalidoinnin keskiarvotuloksen. Huomionarvoista tässä menetelmässä on, että valitun hyperparametrijoukon suuruden mukana kasvaa myös ristiinvalidointien määrä, jolloin myös suoritusaika pitenee [10].

4.2 Ristiinvalidointi Keraksessa Sci-Kit Learn kääreluokalla

Edellä kuvatut menetelmät ovat tärkeitä työkaluja luokittimen suorituskyvyn arvioinnissa ja suunnittelussa. Keras ei itsessään tarjoa rajapintaa ristiinvalidoinnille ja ristiinhaulle. Sci-Kit Learn -kirjasto tarjoaa *GridSearchCV*-funktion samanaikaiseen ristiinvalidointiin ja -hakuun. Keras tarjoaa kääreluokan, joka mahdollistaa Keras-luokittimien käytön Sci-Kit Learn-rajapinnassa.

Kääreluokan rakentaja

```
keras.wrappers.scikit_learn.KerasClassifier(build_fn=None,
        **sk_params)
```

palauttaa sklearn-yhteensopivan objektin, jonka rajapinnan toteuttaa Keraksen luokitinmalliobjekti.

Parametrille *build_fn* tulee antaa arvoksi joko

- funktio, joka palauttaa Keraksen Sequential- tai Model-tyyppisen luokitinmalliobjektin, tai
- objekti, joka toteuttaa *call*-metodin (ei käsitellä työssä laajemmin).

Parametrille ***sk_params* annetaan lista *fit* ja *predict* metodeille syötettävistä paremetreistä. Työssä tätä paremetria ei kuitenkaan käytetä, vaan ristiinhaussa käytettävät parametrit syötetään *GridSearchCV*-metodille, joka syöttää arvoitavat parametrit luokittimen rakentavalle *buildfn*-metodin *fitparams*-parametrille.

GridSearchCV-metodia

```
sklearn.model_selection.GridSearchCV(estimator,
        param_grid, ... )
```

kutsuttaessa sille tulee asettaa taulukon 4.1 esittelemät parametrit.

Taulukko 4.1 *Käytetyt GridSearcCV-metodin parametrit*

Nimi	**Tyyppi**	**Pakollinen**	**Vakioarvo**	**Kuvaus**
estimator	sklearn estimator	Kyllä	Pakollinen parametri	Sklearn-kirjaston luokitinrajapinnan toteuttava objekti
param_grid	Dictionary	Kyllä	Pakollinen parametri	Dictionary, jonka avaimet ovat parametrien nimiä ja arvot listoja ristiinhaettaevista parametrien arvoista
scoring	Funktio	Ei	None	Luokittimen metriikan toteuttava funktio
cv	Integer	Ei	3	Ristiinvalidointiin jaettujen harjoitusdatan osien määrä, alaluvun 4.1 muuttuja k

Scoring-parametrilla määritellään ristiinhaussa ja ristiinvalidoinissa käytettävä metriikka. *DeepChromen* metriikkana käytetään *Area under curve* -arvoa (AUC), joten parametrin arvoksi tulee asettaa sitä vastaava arvo *'roc_ auc'* [3].

4.3 Ristiinhaussa sovitettavat parametrit

Ristiinhaussa optimoitavat kerroskomponentttien parametrit ovat taulukossa 4.2.

Taulukko 4.2 *Ristiinhaussa sovitettavat parametrit*

Kerros	**Parametri**	**Sovitettavat arvot**
Conv1D	filters	20, 50 , 100
Conv1D	kernel_size	5, 10
MaxPooling1D	pool_size	2, 5

Metodin *fit* parametreille *epoch* etsittiin optimaalinen arvo joukosta {16, 32, 64}. Luokittimen parhaan suorituskyvyn parametrijoukko on esitelty luvussa 5.

5. TULOKSET

Työssä toteutettu luokitin pääsee REMC-tietokannan E047 CD8_Naive_Primary_-Cells näytesarjan kanssa AUC-arvoon 0.89 [3]. Alkuperäinen DeepChrome pääsee 0.83 AUC-pisteeseen, joten voidaan olettaa toteutetun luokittimen suorituskyvyn olevan DeepChromen tasoa. GridSearchCV:n tuloksena, edellä mainitun parhaan tuloksen antaneet hyperparametrit taulukossa 5.1.

Taulukko 5.1 *GridSearchCV:n valitsemat hyperparametrit työn luokittimelle*

Nimi	Arvo
Konvoluutiokerroksen suotimien lukumäärä	50
Konvoluutiokerroksen ikkunan pituus	10 elementtiä
Pooling-ikkunan pituus	5
Epochs	32

DeepChromeen valitut parhaan tuloksen tuottavat hyperparametrit taulukossa 5.2

Taulukko 5.2 *Deep Chromen hyperparametrit*

Nimi	Arvo
Konvoluutiokerroksen suotimien lukumäärä	50
Konvoluutiokerroksen ikkunan pituus	10 elementtiä
Pooling-kerroksen ikkunan pituus	5

Tuloksista nähdään, että luokittimilla on samat parhaat tulokset tuottavat hyperparametrien arvot. Suorituskyvyn ja valittujen hyperparametrien ollessa samat toteutetulla luokittimella kuin DeepChromella, on vahva näyttö siitä että DeepChrome-luokittimen Keras-toteutus on onnistunut.

6. YHTEENVETO

Työn tavoitteena oli tutkia geenien aktiivisuusluokittimen toteutusta käyttäen Kerasta ja histonimodifikaatiodataa. Lopputulokseen päästiin toteuttamalla DeepChrome-arkkitehtuuri, jäljitellen sen rakennetta ja parametrijoukkoja käyttäen Keraksen rajapintoja. Luokittimen suorituskyky validoitiin ja parametrit optimoitiin käyttämällä sklearn-kirjaston ristiinvalidointi- ja ristiinhakurajapintaa. Saadut tulokset osoittavat, että toteutus Keras-kirjastolla on onnistunut, luokittimen suorituskyvyn vastatessa alkuperäistä DeepChrome-toteutusta.

LÄHTEET

[1] Valtioneuvoston lehdistötiedote, "Tutkimus- ja innovaationeuvosto: Suomi tarttuu tekoälyn mahdollisuuksiin", 13.12.2017, http://valtioneuvosto.fi/artikkeli/-/asset_publisher/1410845/tutkimus-ja-innovaationeuvosto-suomi-tarttuu-tekoalyn-mahdollisuuksiin

[2] G. Seif, I'll tell you why Deep Learning is so popular and in demand, Medium, 06.01.2018 https://medium.com/swlh/ill-tell-you-why-deep-learning-is-so-popular-and-in-demand-5aca72628780

[3] S. Ritambhara, J. Lanchantin, G. Robins, Y. Qi, "DeepChrome: Deep-learning for predicting gene expression from histone modifications", *Bioinformatics*, vol. 32, no. 17, pp. i639-i648, 2016.

[4] I. Goodfellow, Y. Bengio A. Courville, "Deep Learning", 1. painos, MIT Press, 2016, Avaivable: http://www.deeplearningbook.org

[5] Keras-dokumentaatio. Viitattu 12/2017. www.keras.io

[6] DeepChromen alkuperäinen toteutus Lua-ohjelmointikielellä. Julkaistu tutkimuksen "DeepChrome: Deep-learning for predicting gene expression from histone modifications" (Ritambhara & et. al) yhteydessä. https://github.com/QData/DeepChrome

[7] Scikit-learn kirjaston dokumentaatio. http://scikit-learn.org/stable/modules/classes.html

[8] G. Panchal, A. Ganatra, P. Shah, D. Panchal "Determination of over-learning and over-fitting problem in back propagation neural network", International Journal on Soft Computing (IJSC), No.2, 2011 https://pdfs.semanticscholar.org/bc17/b61de9f618d88d4fd8a145ead4c87c60ccbd.pdf

[9] Useita tuntemattomia kirjoittajia, "ML Cheat Sheet" http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

[10] C. Hsu, C. Chang, C. Lin, "A Practical Guide to Support Vector Classification", sivut 5-7, National Taiwan University https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[11] U. Toikkanen, "Epigenetiikasta etsitään uusia työkaluja", Lääkärilehti 16-17/2014 http://www.laakarilehti.fi/ajassa/ajankohtaista/epigenetiikasta-etsitaan-uusia-tyokaluja/