Martin Johannes Nilsen
Ole Jonas Liahagen
Simon Årdal

Bacheloroppgave

Got Your Back: Bruk av maskinlæring til å klassifisere sittestillinger i sanntid

Bacheloroppgave i Dataingeniør
Veileder: Elise Klæbo Vonstad

Mai 2021

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

NTNU

Kunnskap for en bedre verden

Martin Johannes Nilsen
Ole Jonas Liahagen
Simon Årdal

Got Your Back: Bruk av maskinlæring til å klassifisere sittestillinger i sanntid

Bacheloroppgave i Dataingeniør
Veileder: Elise Klæbo Vonstad

Bacheloroppgave
Mai 2021

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

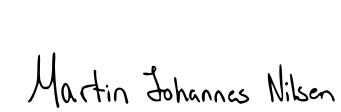
Forord

Denne bacheloroppgaven er utført av tre dataingeniørstudenter ved NTNU, våren 2021. Oppgaven er gitt og veiledet av Elise Klæbo Vonstad, stipendiat ved NTNU IDI/AIT. Oppgavestiller ønsket å undersøke i hvor stor grad man kan ta i bruk maskinlæring og data fra bluetooth-sensorer for å klassifisere sittestillinger, noe som senere kan tas i bruk ved forskning på rygghelse og ved forebygging av ryggplager.

Oppgaven ble valgt fordi den inkluderte å jobbe med spennende teknologier som blant annet databehandling i Python, maskinlæring og arbeid med sensorer for å kunne lage et produkt som potensielt sett kan bidra til å bedre befolkningens rygghelse.

Prosjektet har vært givende, ikke bare fra et teknologisk perspektiv, men også fra et empirisk, ingeniørfaglig og vitenskaplig perspektiv. Det å planlegge, gjennomføre og diskutere egne vitenskaplige metoder og fremgangsmåter har vært krevende, og det å stå på egne ben for å trekke slutninger mellom hypotese, teori og resultater for så å diskutere og konkludere kan være utfordrende. Likevel er den utviklede løsningen og sluttresultatet noe samtlige av gruppens medlemmer er godt fornøyde med og stolte av.

Avslutningsvis ønsker vi å rette en stor takk til vår veileder for gode veiledningsmøter og hjelp ved behov. Videre vil vi også uttrykke vår takknemlighet ovenfor medstudenter og andre ansatte ved NTNU som har hjulpet oss med å spille inn data. Til slutt ønsker vi å takke Janne Birgitte Block Børke ved St. Olavs Hospital for helsefaglig informasjon som førte til en revurdering av oppgaven, slik at den ble mer faglig korrekt.



Martin Johannes Nilsen
Trondheim, 20.05.21

Ole Jonas Liahagen
Trondheim, 20.05.21

Simon Årdal
Trondheim, 20.05.21

Oppgavetekst

Ved begynnelsen av bachelorprosjektet var oppgaveteksten gitt som følger:

> **Hensikten med oppgaven:**
> Hensikten med oppgaven er å utvikle et system som bruker sensorteknologi og maskinlæring for å analysere holdningen i rygg/nakke når man sitter. Da vil man kunne gi feedback til brukeren om å huske å sitte riktig for å unngå problemer med rygg/nakke f.eks. på hjemmekontor.
>
> **Kort beskrivelse av oppgaveforslag:**
> Akselerometerdata kan brukes til å analysere posisjonen til kroppsdeler over tid ved hjelp av maskinlæring. Oppgaven vil innebære utprøving og testing av maskinlæringsalgoritmer for å identifisere dårlig holdning i rygg/nakke ved hjelp av slik sensordata, og gi tilbakemelding til brukeren om å huske å sitte riktig, evt hva som burde forbedres. Et slikt system vil kunne hjelpe mange som sitter på et kontor å unngå plager og sykemeldinger som følge av dårlig holdning.
>
> Studenten(e) står fritt til å velge teknologi og algoritmer.

Etter et møte 29.01.2021 med Janne-Birgitte Block Børke, forskningskoordinator og sjef for klinikk for fysikalsk medisin og rehabilitering ved St. Olavs Hospital i Trondheim, ble det imidlertid fastslått at denne oppgaveteksten tar utgangspunkt i en teori som moderne forskning etterhvert har avkreftet, nemlig at man kan skille mellom *riktige* og *gale* sittestillinger. Hun kunne fortelle at rygghelseproblemer er svært individuelt og genavhengig. Med andre ord er ikke én bestemt sittestilling en universell god stilling - den kan være god for en person og dårlig for en annen. Det som dog er etablert er at *variasjon* i sittestilling er en svært viktig faktor. Som følge av dette, ble det foreslått å flytte fokuset vekk fra klassifisering av gode og dårlige sittestillinger, og heller fokusere på en overvåking av variasjon. Møtereferat fra møtet med Børke er vedlagt (Vedlegg D).

Oppgaven vil fortsatt innebære å ta i bruk maskinlæringsmodeller som trenes på sensordata til å gjenkjenne sittestillinger, og den fokuserer fortsatt på å implementere et system som tar i bruk en slik modell. Forskjellen ligger altså i at systemet skal benyttes til å overvåke varians i sittestilling i stedet for å gi tilbakemelding om god/dårlig sittestilling.

Sammendrag

Dårlig rygghelse er et stort problem i dagens samfunn, og kan legge store hemninger på en persons liv. En viktig faktor for å opprettholde god rygghelse er variasjon i måten en sitter på over tid. I denne oppgaven har det blitt implementert et system som tar i bruk maskinlæring for å kartlegge variasjon i en persons sittestilling. Det har derfor både vært fokus på systemutvikling og forskning på maskinlæringsalgoritmer - samt å benytte forskningsresultatene direkte i det ferdige resultatet.

Løsningen er en skrivebordsapplikasjon utviklet i React og bygget som en lokal applikasjon med rammeverket Electron. Applikasjonen kommuniserer med en Flask server som tar seg av sensorkommunikasjon, klassifisering og andre endepunktkall. Data fra sensorene blir i bakgrunnen fôret til en maskinlæringsmodell som med stor treffsikkerhet klassifiserer sittestillinger i sanntid.

Fire forskjellige maskinlæringsalgoritmer er blitt implementert; Random Forest Classifier, Long Short-Term Memory, Artificial Neural Network og Convolutional Neural Network. Samtlige er trent på egenprodusert treningsdata, samt testet på egne testsett og viser gode treffrater.

Abstract

Back pain is a common problem in today's society, and can have severely negative impact on a persons life. A vital factor concerning back-health is posture variation - which is something modern back-pain research tries to address. In this bachelor-thesis we have implemented a system which uses machine learning to track a person's posture-variation while sitting, and conducted our own machine learning research on the data collected. The results we have achieved from the research have been used directly in our system application.

Our solution is a desktop application, developed in React and built using the Electron framework. The application uses a Flask server which handles sensor communication, posture classification and other requests from the client. Realtime-data from the sensors are forwarded to a machine learning model, which is able to accurately classify the user's current sitting posture.

Four different machine learning algorithms have been implemented; Random Forest Classifier, Long Short-term Memory network, Artificial Neural Network and Convolutional Neural Network. All the algorithms are trained using our self-produced dataset, and are achieving high accuracy at predicting sitting postures both in realtime and using recorded test-data.

Innhold

Figurer

Tabeller

Akronymer

AI Artificial Intelligence 5

ANN Artificial Neural Network 2, 20, 33, 37

AR Activity Recognition 1

CNN Convolutional Neural Network 2, 12, 20, 37, 41

IMU Inertial Measurement Unit 3

KNN k-Nearest Neighbors 2

LSTM Long Short-Term Memory 2, 14, 20, 25, 39

MLP Multilayer perceptron 7, 8

PWA Progressive Web Application 26

RFC Random Forest Classifier 2, 16–18, 20, 25, 38, 41

RNN Recurrent Neural Network 2, 13, 14, 20

SVM Support Vector Machine 2, 45

WCAG Web Content Accessibility Guidelines 26

Ordliste

feature En kolonne med data som beskriver en karakteristikk ved datasettet. 1, 5, 37, 38

forsterket læring Maskinlæring med en agent som opererer i et gitt miljø, og lærer basert på belønning/straff gitt for valgene den tar. 5

gradient descent En metode for å finne lokale minimum for deriverbare funksjoner. 9, 10

ikke-veiledet læring Maskinlæring med datasett som ikke inneholder en fasit på hva som er riktig output, gitt en input. 5

klynge En gruppering av data basert på like features eller forekomst sammen. 5

konvolusjonelt nevralt nettverk En type nevralt nettverk med en bevegende kjerne i et gitt antall dimensjoner. 13, 25

kunstig nevralt nettverk En type maskinlæringsmodell med et gitt antall lag med nevroner for å etterlikne biologiske nettverk. 12, 25

pooling En teknikk for å redusere en matrises dimensjon samtidig som at en ønsker bevare dataens kvalitet. 13

prediksjon Resultatet av en gjennomgang av en maskinlæringsalgoritme. Dette vil være svaret på input man sender inn. Om dette er en klassifiseringsmodell, vil man her få ut hvilken klasse modellen tror dataene tilhører. 9

stride En parameter som beskriver hvor langt en CNN-kjerne skal bevege seg for hver iterasjon. 12

veiledet læring Maskinlæring med datasett som inneholder en klasse som indikerer ønsket output gitt en input. 5, 19

1 Introduksjon

Studier utført av eksperter innenfor kiropraktikk viser at opptil 80 prosent av befolkningen vil oppleve ryggplager i løpet av livet. Dette fører til millioner av tapte arbeidstimer og generelt lavere livskvalitet hos de det rammer [3]. Videre er smerter i korsryggen også den viktigste årsaken til sykefravær og uførhet i Europa [40]. Denne oppgaven har som formål å hjelpe personer som allerede sliter med ryggrelaterte plager på grunn av stillesittende arbeid, og å virke forebyggende hos personer som enda ikke er rammet. Produktet som utvikles har som formål å kartlegge variasjon i sittestillinger ved bruk av en applikasjon og trente modeller utviklet i dette prosjektet, noe som kan tas i bruk i videre forskning på området.

1.1 Problemstilling

Det konkrete målet for dette prosjektet er å utvikle en klassifiseringsmodell som klarer å klassifisere sittestillinger ved hjelp av sensorer fra Life Performance Research [23]. Dette innebærer å se på hvor høy nøyaktighet en klarer oppnå med ulike maskinlæringsmodeller, og videre hvor mange sensorer en må ta i bruk for å oppnå best mulig nøyaktighet.

I den sammenheng ønsker vi å besvare forskningsspørsmålene:

- *Hvilken nøyaktighet kan man oppnå ved klassifisering av sittestillinger basert på maskinlæringsalgoritmer og posisjonsdata fra IMU-sensorer?*
- *I hvor stor grad påvirkes klassifiseringsnøyaktigheten av antall sensorer?*

1.2 Tidligere relevant arbeid

I denne seksjonen vil det bli presentert relevant arbeid i de fagområdene som oppgaven går inn under, som aktivitetsgjenkjenning (AR) og maskinlæring. Det har ifølge Ramamurthy & Roy [29] vært en økende trend de siste årene å ta i bruk maskinlæringsteknikker for å finne mønstre i aktivitetsgjenkjennings-relaterte datasett. Spekteret strekker seg fra modeller der features er nøye utplukket av forskere, til i nyere tid dype nevrale nettverk der modellene selv velger hvilke features de skal vektlegge. Aktivitetsgjenkjenning forblir et utfordrende problem på grunn av den komplekse og kaotiske variasjonen en finner i dataens natur. Som presentert i visjonsdokumentet er det blitt utført en rekke prosjekter av liknende karakter de siste årene, som PosturePal [13] og forskningsartikkelen til Gupta *et al.* [9] for å nevne noen. Maskinlæring er heller ikke et uutforsket fagfelt, og har blitt spesielt populært de siste tiårene. Domenet aktiv læring derimot, som er en underkategori av aktivitetsgjenkjenning, har vokst frem i nyere tid. Her er det et fokus på at uklassifisert treningsdata blir manuelt annotert av et menneske, og at den annoterte dataen blir brukt til å trene maskinlæringsmodeller som igjen skal kunne klassifisere usett data. Avslutningsvis er forskning på hvordan en AR-modell påvirkes av introduksjonen av en ny sensor i et system av Bannach *et al.* [4] av relevans.

Når det kommer til metoderelatert arbeid er det innen aktivitetsgjenkjenning primært blitt tatt i bruk maskinlæringsalgoritmer som Random Forest Classifier, k-Nearest Neighbors og Support Vector Machine, i tillegg til dype maskinlæringmodeller som ANN, CNN, RNN og LSTM i nyere tid [29]. En omfattende beskrivelse av dyp læring og effekten av å anvende dette på tidsseriedata er omtalt i Wang *et al.* [38]. På samme tid forskes det på bruken av ANN, CNN og RNN for aktivitetsdata, og som beskrevet i Hammerla *et al.* [10] konkluderes det med at RNN er den av teknikkene med høyst presisjon. Videre har det vist seg at CNN med flere lag og pooling vil kunne prestere like godt, om ikke bedre enn RNN i gitte tilfeller [31]. Som beskrevet i Panwar *et al.* [26] ser en også at LSTM kan hevde seg med CNN når tatt i bruk på sensordata.

1.3 Rapportstruktur

Strukturen til dette prosjektet er delt inn på følgende måte:

Kapittel 1 Introduksjon gir en introduksjon til hva som ligger til grunn for oppgaven. Videre følger tidligere relevant arbeid som har blitt lest og hentet inspirasjon fra, før problemstillingen blir presentert.

Kapittel 2 Teori tar for seg grunnleggende teori man bør forstå for å kunne tolke de resultatene som blir presentert i rapporten. Elementer som hva slags type data som blir brukt og hvordan maskinlæringsmodellene fungerer etc. er relevant her.

Kapittel 3 Valg av teknologi og metode inneholder en beskrivelse av valg av programmeringsspråk, rammeverk og verktøy som har blitt brukt for å komme frem til de resultatene en har fått. I tillegg er prosessen og arbeidsmetodikk beskrevet i dette kapittelet.

Kapittel 4 Resultater presenterer de vitenskapelige, ingeniørfaglige og administrative resultatene som har fremkommet i prosjektet.

Kapittel 5 Diskusjon inneholder drøfting rundt prosessen og de resultatene som har blitt presentert. Dette innebærer drøfting av styrker og svakheter med resultatene en har fått og hvordan en har fått disse.

Kapittel 6 Konklusjon og videre arbeid inneholder de konklusjoner som kan trekkes basert på drøftingen i forrige kapittel, samt problemstillingen og kravene i visjonsdokumentet.

2 Teori

2.1 Data og IMU

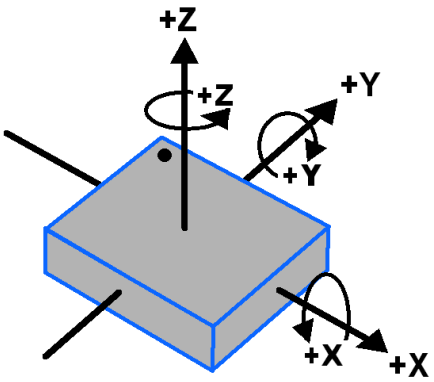
For å kunne klassifisere sittestillinger med maskinlæringsmodeller, trengs data som gir god informasjon om hvordan brukeren sitter. Slik informasjon kan oppdrives ved å ta i bruk instrumenter som akselerometer, gyroskop og/eller magnetometer. I dette prosjektet kommer den innspilte dataen fra mindre sensorer med en Inertial Measurement Unit (IMU) installert, som kommuniserer med datamaskinen over bluetooth. IMUen inneholder instrumenter som akselerometer, gyroskop og magnetometer, og returnerer felter som lineær og angulær akaselerasjon i tre dimensjoner, gyroskop i tre dimensjoner og quaternions i fire dimensjoner.

2.1.1 Akselerometer

Akselerometer er elektromagnetiske enheter som registrerer endringer i fart eller krefter indusert av tyngdekraft. Det kan brukes for å måle helning, vibrasjon og støt, og brukes i økende grad i bærbar elektronikk. Dens måleenhet er $meter/sekund^2 (m/s^2)$ eller g-krefter. Akselerometere kan måle langs én, to eller tre akser - og man kan skille mellom lineære akselerometere og angulære akselerometere. Sistenevnte registrerer vinkelhastigheten om tre akser.

2.1.2 Gyroskopdata

Data fra et gyroskopinstrument kan i likhet med akselerometerdata måle langs de tre aksene x, y og z, som sammen beskriver sensorens orientasjon. Dette gjøres ved hjelp av tre rotasjonsakser kalt gimbaler samt en rotor i sentrum av gyroskopet. Rotoren forholder seg til rotasjonsretningen satt, uavhengig av gimbalenes orientasjon. Dataen sendt er gimbalenes orientasjon i forhold til rotoren i sentrum.



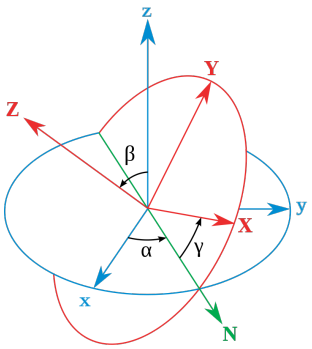
Figur 2.1: Gyroskopmodul[1]

[1] Hentet 05.05.21 fra *https://www.electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module*

2.1.3 Euler-vinkler

Eulervinkler er tre vinkler introdusert av Leonhard Euler som kan brukes for å beskrive rotasjoner (orienteringen av et fast legeme i forhold til et fiksert koordinatsystem). De kan defineres ved geometri eller som en sammensetning av enkle rotasjoner. Den geometriske definisjonen av Eulervinkler tilsier at et legeme kan oppnå enhver rotert tilstand som følge av tre sammensatte, enkle rotasjoner. Disse enkle rotasjonene kan enten være rundt xyz-aksene til det originale koordinatsystemet, eller xyz-aksene tilhørende det roterende legemet.

Eulervinkler er en av flere måter å representere rotasjoner. Andre brukte måter for å representere rotasjoner er rotasjonsmatriser og quaternions. Det er mulig å veksle mellom disse konvensjonene, og bytte fra en til en annen. Eulervinkler er enkle og og intuitive å bruke, og kan godt brukes til analyse og kontroll.



Figur 2.2: Geometrisk definisjon av Eulervinkler[2]

2.1.4 Quaternions

Quaternions kan som beskrevet av LaValle [17] i likhet med eulervinkler brukes til å beskrive rotasjon i tre dimensjoner. For å gjøre dette introduserer man fire variabler a, b, c, og d. Disse kan ses på som del av en firedimensjonal vektor. Det betyr at $a, b, c, d \in \mathbb{R}^4$. Quaternioner har en ikke-kommutativ algebra som anses som enklere og mer kosteffektiv enn utregninger med vanlige rotasjonsmatriser, og er dermed mye brukt innen spillgrafikk og andre områder hvor rotasjon i tre dimensjoner foregår.

Grunnen til at quaternions sees på som et bedre alternativ enn matriser til utregninger på rotasjoner i tre dimensjoner er deres likheter til komplekse tall, og deres gode egenskaper til å utregne rotasjoner i to dimensjoner. Man kan ofte se en quaternion skrevet som $q = a + bi + cj + dk$, hvor i, j og k er imaginære tall som oppfyller samme likning som det imaginære tallet i ($i^2 = j^2 = k^2 = -1$).

[2] Hentet 20.04.21 fra *https://en.wikipedia.org/wiki/Euler_angles*

Quaternions kan også skrives som et ordnet par $q = (s, v)$, hvor s kalles skalar-delen av quaternionen og v er en tredimensjonal vektor. Det er til slutt også mulig å presentere en quaternion slik:

$$q = (s, V) = s + v\hat{q}$$

Satt opp mot et generelt komplekst tall $z = a + bi$ er likheten slående. Det er nettopp denne likheten man utnytter ved rotasjoner ved hjelp av quaternions. Med komplekse tall kan en rotasjon av en vektor gjøres ved å gange en vektor i det komplekse planet med en såkalt rotor, som er et komplekst tall definert ved $rotor = cos\theta + i * sin\theta$.

Det viser seg at rotasjonsquaternions ser veldig like ut, grunnet deres liknende struktur. En generell rotasjonsquaternion defineres ved

$$q = (cos\frac{1}{2}\theta, sin\frac{1}{2}\hat{v}),$$

noe som til slutt gir mye enklere beregninger enn om man holder seg innenfor domenet til rotasjonsmatriser.

2.2 Maskinlæring

Maskinlæring er et felt innen kunstig intelligens (AI), som gir systemer muligheten til å automatisk lære og forbedre seg basert på erfaring som ikke har blitt eksplisitt programmert [18]. For å trene og tilpasse modeller, som er hovedessensen i maskinlæring, må en aller først ha observasjoner eller data for å utforske potensielt underliggende mønstre. Disse mønstrene er ifølge Loukas [18] ikke annet enn funksjoner og beslutningsgrenser. Hva som gjøres videre for å finne disse mønstrene avhenger så av dataformatet, og derav hvilken underkategori av maskinlæring en har med å gjøre. Det er vanlig å dele opp maskinlæring i tre hovedområder: Veiledet læring, Ikke-veiledet læring og Forsterket læring.

Veiledet læring handler om å la modellene trene på datasett der det er forhåndsdefinert hva som er riktig output for hver input. Målet er å lage en modell som klarer å gi riktig output på data den aldri har sett før. For å få til dette må en som beskrevet i Su [35] lage en funksjon (modell) som korrekt klassifiserer riktig klasse fra datasettets andre egenskaper, som igjen kan bli brukt til å klassifisere ny data. Typiske eksempler på dette er klassifisering eller regresjon.

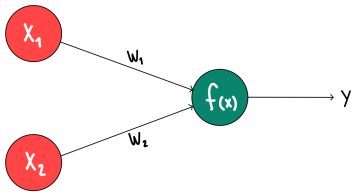
Ikke-veiledet læring er en mer utforskende tilnærming. Denne tilnærmingen baserer seg på data som ikke har en tilhørende fasit på hva som er riktig klasse, dette blir modellens jobb å løse. Modellen må selv lære seg hvordan den skal klassifisere. For å få til dette er det vanlig at modellene grupperer data basert på gitte features, og brukes derfor ofte i klyngeanalyse med store datamengder der en ønsker å gruppere data som ved første øyekast ikke har klare sammenhenger.

Forsterket læring går ut på å få en agent til å utføre de riktige handlingene i et gitt miljø. For en slik modell er det vanlig at agenten får en belønning for riktige valg, og straff ved feil valg. Denne formen for læring brukes for eksempel til å lære maskiner å spille spill som Atari Breakout [21] og sjakk [32] bedre enn mennesker.

2.3 Kunstig Nevralt Nettverk

Nevrale nettverk, eller kunstige nevrale nettverk, er en underkategori av maskinlæring som benytter et eller flere *nodelag* for å emulere et biologisk nevralt nettverk, og den elektriske aktiviteten i hjernen og i nervesystemet [24]. Virkemåten til hvert nevron kan tilnærmes med perceptron-modellen, først foreslått av Frank Rosenblatt i 1958 og senere raffinert og analysert av Minsky & Papert [20] i 1969.

2.3.1 Perceptron-modellen



Figur 2.3: Perceptron-modellen

Perceptron-modellen gir en matematisk tilnærming til nevron-sammenligningen, vist ved Figur 2.3. I dette eksempelet får nevronet (noden) to inputvariabler, x_1 og x_2. Noden i seg selv behandler disse ved bruk av en aktiveringsfunksjon. Outputen fra noden blir resultatet fra funksjonsberegningen $f(x)$.

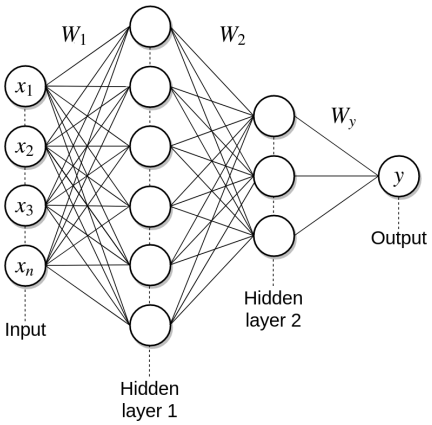
For at nevronet skal kunne lære, trengs også variabler som kan justeres for å påvirke output (verdien av $f(x)$). Man bruker derfor *vekter* (w_1 og w_2), som multipliseres med inputvariablene for å kunne gjøre nettopp dette. For å håndtere tilfeller hvor inputvariablene er lik null, har man også *biaser*, b, som sikrer at inputverdiene i disse tilfellene ikke blir neglisjert.

Rent matematisk har man at

$$\hat{y} = \sum_{i=1}^{n} x_i w_i + b,$$

hvor y er vist i Figur 2.3 som output fra aktiveringsfunksjonen $f(x)$.

2.3.2 Multilayer perceptron (MLP)



Figur 2.4: Multilayer perceptron-modell[3]

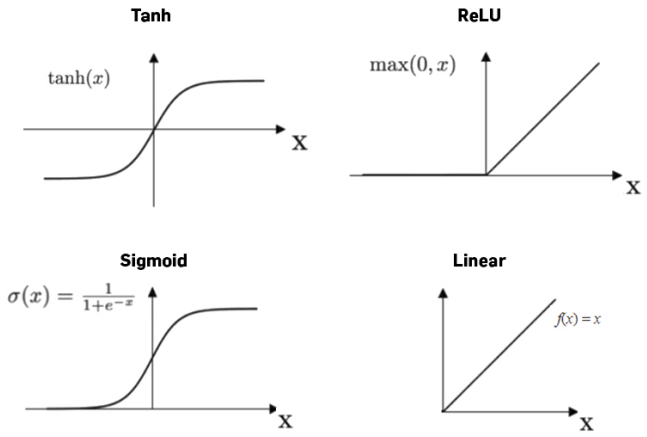
For å være i stand til å tilnærme komplekse matematiske modeller og problemstillinger kan man slå sammen flere perceptroner til en Multilayer perceptron (MLP)-modell, som er et nettverk av enkle perceptroner. Nodene blir organiserte i vertikale lag (*layers*), hvor man kan skille mellom input-laget, de gjemte lagene og output-laget. De gjemte lagene tar både input fra forrige lag, og produserer output som blir gitt som input til neste lag. Nettverket kjennetegnes som et *dypt nevralt nettverk* dersom man har to eller flere gjemte lag. Input-laget er det første laget, mens output-laget er det siste.

Det viser seg at et nevralt nettverk bestående av bare et lag med noder er nok for å tilnærme en hvilken som helst funksjon, selv om laget må være urimelig stort og ikke nødvendigvis får til å lære eller generalisere skikkelig [8]. Med *tilnærming av en hvilken som helst funksjon* menes det at ved å benytte nok noder kan man alltid finne et nevralt nettverk som har en output, $g(x)$ som tilfredsstiller $|g(x) - f(x)| < \epsilon$ for all input, x. Tilnærmingen vil med andre ord være tilfredsstillende for hver mulige input.

Selv om teoremet sier at ett lag er nok, er det vanlig å bruke flere lag i praksis for å unngå problemer ved trening som for eksempel overtrening.

[3]Hentet 05.05.21 fra *https://www.researchgate.net/figure/A-fully-connected-neural-network-with-two-hidden-layers_fig4_323218981*

2.3.3 Aktiveringsfunksjoner



Figur 2.5: Aktiveringsfunksjoner[4]

I en Multilayer perceptron (MLP)-modell vil hver node ha en funksjon ($f(x)$ i Figur 2.3) som behandler produktet til input og vekter. Målet med denne funksjonen er å kontrollere Z, hvor $Z = x*w+b$, slik at man holder kontroll på nodenes verdier utover i nettverket. Det finnes flere typer aktiveringsfunksjoner, som kan være passende til hver sine brukstilfeller.

Sigmoid er gitt ved

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

og gir en verdi mellom 0 og 1, som gjør den passende til binær klassifisering. Desto høyere input-tall, desto nærmere 1 blir output, og desto lavere input-tall, desto nærmere 0 blir output.

tanh er gitt ved

$$\frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

og er veldig lik sigmoid, men ettersom den gir en verdi mellom -1 og 1 vil gjennomsnittet konvergere mot 0, som gjør dataen mer sentrert rundt origo og lett å jobbe videre med.

ReLu er gitt ved

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

og er en lineær aktiveringsfunksjon. Den er en av de mest brukte aktiveringsfunksjonene på grunn av dens simplisitet, allsidighet og gode ytelse.

Softmax er gitt ved

$$S(y)_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}},$$

[4]Hentet 21.04.21 fra *https://docs.paperspace.com/machine-learning/wiki/activation-function*

hvor y er inputvektor, og y_i er i-ende element i vektoren. Denne er ofte brukt i siste lag ved multiklasse-klassifisering, ettersom den normaliserer output ved det aktuelle laget, og distribuerer en sannsynlighetsfordeling utover de aktuelle klassene.

2.3.4 Tapsfunksjoner og gradient descent

Etter at node-lagene i nettverket har behandlet sin input via aktiveringsfunksjoner og sendt resultatet videre til neste lag, vil outputen fra det siste laget være modellens estimering for hva den tror sagt input er. Denne outputen, $\hat{y}$, blir modellens *prediksjon*. Denne prediksjonen kan ha ulik grad av korrekthet, fra helt feil til helt riktig. Uansett hva outputen blir, må det *evalueres*. Man må sammenligne predikeringen opp mot fasitsvaret, og gjøre tiltak dersom det viser seg at predikeringen ikke er riktig eller tilfredsstillende. Denne sammenligningen gjøres ved bruk av en *tapsfunksjon*. En tapsfunksjon er med andre ord en funksjon som måler hvor langt unna modellens prediksjoner det reelle svaret er. I praksis vil *tapet* (en verdi for hvor feil modellen tar), være et gjennomsnitt - og derav én enkel verdi. På denne måten kan man observere hvor feil modellen tar underveis i treningen som en helhet.

I praksis vil en tapsfunksjon være avhengig av et nettverks vekter. Vi har at

$$C(W_1, W_2, W_3, ...W_n),$$

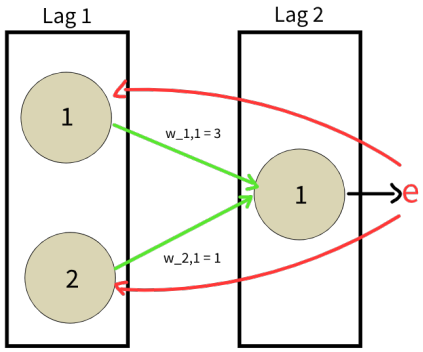
hvor C er selve tapsfunksjonen som tar inn n vekter. Spørsmålet blir da hvordan man endrer disse vektene slik at tapet, altså $C(W_1, W_2, W_3, ..., W_n)$, blir minst mulig.

På grunn av n antall vekter, vil tapsfunksjonen C være n-dimensjonal. På grunn av beregningsmessige begrensninger, vil ikke C være beregnbar dersom n blir svært stor ($n > 100, 1000, ...$). Man kan heller benytte *gradient descent*, som er en type optimaliseringsalgoritme som iterativt beveger seg nedover i et n-dimensjonalt plan og finner lokale minimum.

2.3.5 Tilbakepropagering og vanishing gradient problemet

Teorien om tilbakepropagering er hentet fra boka 'Make Your Own Neural Network' av Rashid [30]. For at nevrale nettverk skal lære, må de ha en evne til å forbedre seg. Dette gjøres gjennom tilbakepropagering. Tilbakepropagering tar utgangspunkt i tapsfunksjonen beskrevet tidligere og tar med seg den utregnede feilen bakover i nettverket. Dette gjøres ved å vektet fordele feilen på de tilkoblede foregående nodene slik at en node med høy innvirkning (stort vekttall) får en større del av feilen enn en med mindre vekttall. Figur 2.6 viser et tenkt nettverk med vektene $w_{1,1}$ og $w_{2,1}$. Feilen fordeles på hver node avhengig av koblingen til den etterfølgende nodens vekt. Å finne feilen for en node i lag 1 koblet til node 1 i output laget ville blitt gjort slik:

$$e_{1,x} = \frac{w_{x,1}}{w_{1,1} + w_{2,1} + w_{3,1} + ... + w_{n,1}}, 1 \leq x \leq n$$



Figur 2.6: Tilbakepropagering av feil

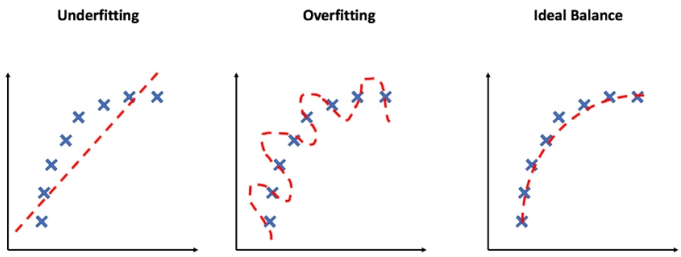
Om vektene fra Figur 2.6 skulle brukes, ville vi fått 3/4 av feilen til $node_{1,1}$ og 1/4 av feilen til $node_{1,2}$. Feilen som blir tildelt hver vekt, tas med til sin tilhørende node og blir feilen det foregående laget bruker til sin fordeling av feil. Det er denne rekursive bruken av feilen bakover i nettverket som gir tilbakepropagering navnet sitt.

Tilbakepropageringen kan føre til et problem kalt *vanishing gradient* problemet i nettverk som benytter seg av *gradient descent* metoden. Problemet oppstår som følge av at feilen som utregnes i output-laget fordeles utover for mange noder på vei bakover i nettverket og dermed fører til en svært liten feilverdi for noder plassert tidlig i nettverket. Dette gjør at tidligere noder og deres vekter oppdateres svært sakte, noe som igjen fører til tregere læring for nettverket i sin helhet.

2.3.6 Over- og undertrening

Når man trener en maskinlæringsmodell, trener man først modellen på et treningssett, før man benytter modellen til å predikere på usett data. Ved tilfeller hvor modellen presterer svært godt ved trening (opp mot 99 prosent nøyaktighet), men dårlig ved usett data er det stor sannsynlighet for at modellen har overtrent. I disse tilfellene vil ikke modellen generalisere godt nok. I tråd med prinsippene fra boken *The Signal and the Noise* [33] kan man skille mellom signal og bråk (eng: noise), hvor signalet er det underliggende mønsteret man ønsker maskinlæringsalgoritmen skal kunne etterligne, og bråk er irrelevant og distraherende data i et datasett. Sagt med andre ord; bråk forstyrrer signalet. Målet er derfor at algoritmen skal være i stand til å separere signalet og bråket fra hverandre. Det motsatte vil føre til at modellen vil gjøre prediksjoner basert på bråket fra treningssettet, og prestere dårlig på usett data.

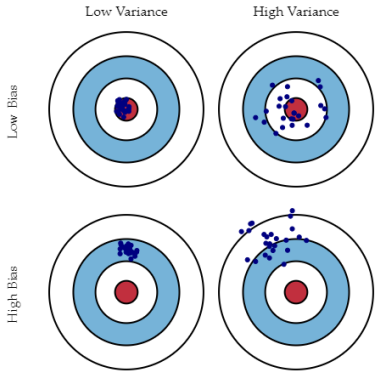
Det motsatte av overtrening er *undertrening*, som kan forekomme dersom modellen ikke er i stand til å oppfatte det underliggende signalet.



Figur 2.7: Grafiske fremstillinger av undertrening, overtrening og en ideell balanse[5]

2.3.7 Bias-varians kompromisset

Bias-varians kompromisset er et kompromiss mellom en modells evne til å minimere bias og varians. Med bias menes differansen mellom en modells gjennomsnittlige prediksjoner og det riktige svaret den prøver å predikere. Modeller med høy bias ser bort fra relevant treningsdata slik at modellen blir oversimplifisert, og man sier at modellen blir undertrent. Med varians menes hvor variert en modell vil predikere et gitt datapunkt. Modeller med høy varians vil være dårlig på å skille mellom bråk og signal - og vil derfor ikke være i stand til å generalisere på data den ikke har sett før. Modellen blir med andre ord overtrent. Et tilstrekkelig godt nok kompromiss mellom bias og varians er nødvendig for at en maskinlæringsmodell skal være i stand til å prestere tilfredsstillende.



Figur 2.8: Sammenhengen mellom bias og varians[6]

Fra Figur 2.8 ser man at et kompromiss mellom lav varians og lav bias er ønskelig for å få en modell til å prestere best mulig. Det betyr at modellen ikke kan være for simpel, med for få parametere, eller for kompleks - med for mange parametere.

En matematisk fremstilling kan også demonstreres. Si man har et fasitsvar, y, som kan oppnås ved $f(x)$ og en gitt input, x. Vi antar altså en sammenheng $y = f(x) + e$ hvor $e = 0$. Vi ønsker en modell, $\hat{f}(x)$, som skal etterligne $f(x)$ ved f.eks. lineær regresjon. Vi forventer at feilen ved en gitt input x er

$$Err(x) = E[(Y - \hat{f}(x))^2]$$

[5]Hentet 20.05.21 fra *https://subscription.packtpub.com/book/data/9781838556334/7/ch07lvl1sec82/underfitting-and-overfitting*

[6]Hentet 21.04.20 fra *https://www.kdnuggets.com/2016/08/bias-variance-tradeoff-overview.html*

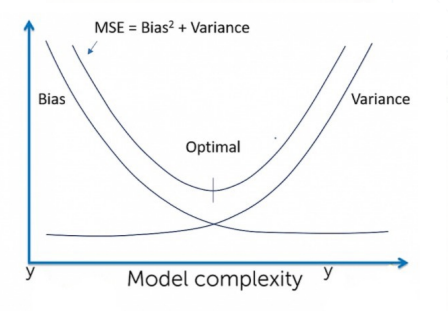
Som kan skrives som

$$Err(x) = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma_e^2$$

Som er

$$Err(x) = Bias^2 + Varians + irreduserbar feil$$

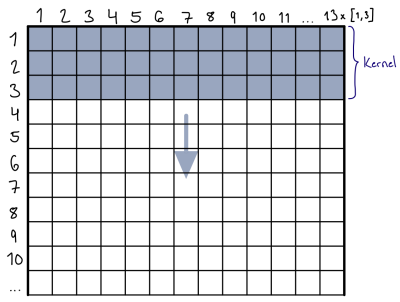
I og med at den totale feilen er gitt ved summen av bias og varians, kan en grafisk sammenheng fremstilles.



Figur 2.9: Grafisk fremstilling av bias-varians kompromisset[7]

2.4 Konvolusjonelt Nevralt Nettverk

Konvolusjonelle nevrale nettverk (eng: CNN) viderefører prinsippene av et kunstig nevralt nettverk, men introduserer i tillegg en kjerne som flytter seg i et gitt antall dimensjoner. For tidsserie-data, der dataen er en todimensjonal matrise, kan en ta i bruk et endimensjonalt konvolusjonelt nevralt nettverk. Her finner man en en todimensjonal kjerne som flytter seg i én retning, som illustrert i Figur 2.10. Kjernen har en lengde lik antall sensorer mulitiplisert med tretten datakolonner, og en bredde (*kernel_size*) som representerer hvor mange rader en vil slå sammen. I tillegg defineres en parameter *stride* som beskriver hvor langt den flytter seg i bredderetning for hver iterasjon.

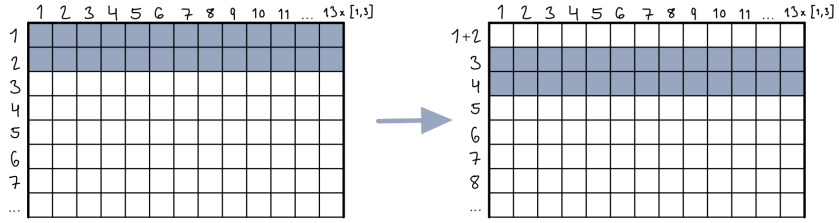


Figur 2.10: Illustrasjon av et konvolusjonelt nevralt nettverk med en todimensjonal kjerne

[7] Hentet 22.04.21 fra *https://www.analyticsvidhya.com/blog/2020/12/a-measure-of-bias-and-variance-an-experiment/*

2.4.1 Pooling

Et prinsipp som også er aktuelt å ta i bruk på et konvolusjonelt nevralt nettverk er pooling. Dette er en teknikk for å redusere matrisens dimensjon, mens man samtidig prøver bevare kvaliteten på dataen. Den mest brukte teknikken er MaxPooling, der en for hver iterasjon tar max av et antall rader for hver kolonne. Hvor mange rader som tas av gangen styres av parameteren *pool_size*. En har også her parameteren *stride*, som bestemmer hvor mange rader en skal gå for hver iterasjon. For pooling er default *stride* lik *pool_size* som i vårt tilfelle er 2, som illustrert i Figur 2.11. Kjernen itererer altså to og to rader nedover og tar maksverdi av hver kolonne over disse to radene. En annen mulighet er å ta i bruk AveragePooling, der den tar gjennomsnittet av de to verdiene. Ved å ta gjennomsnittet istedenfor maksverdi vil en bedre ivareta informasjon fra også de mindre utslagsgivende elementene i en blokk, mens MaxPooling legger mer vekt på større utslag.



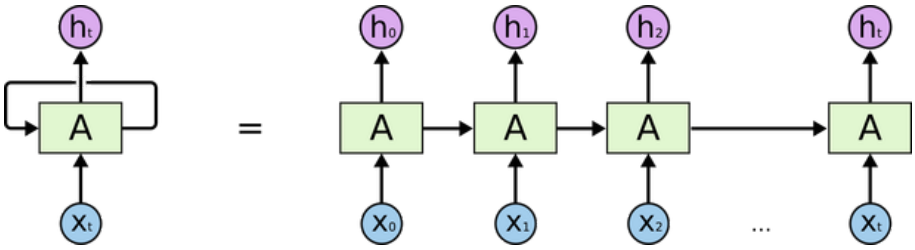
Figur 2.11: Pooling over to og to rader

2.5 Recurrent Neural Network

Standard nevrale nettverk tar ikke hensyn til tidligere hendelser når de klassifiserer. Det eneste modellen vil basere sin klassifisering på er input-data den får tilsendt akkurat i det øyeblikket, mens det i mange situasjoner vil være hensiktsmessig å gjøre klassifiseringer basert på tidligere hendelser. Dersom man for eksempel skal prøve å klassifisere hva som skjer på et gitt punkt i en film, er det svært vanskelig uten å vite hva som har skjedd før i filmen. Et annet eksempel er når man leser en setning; et ord i en setning gir bare mening dersom man forstår de tidligere ordene, og dermed konteksten ordet blir brukt i.

Et Recurrent Neural Network (RNN) er en type nevralt nettverk som prøver å løse denne problemstillingen. I dette nettverket vil ikke klassifiseringer bli gjort basert på bare øyeblikkelig input-data, men også bli påvirket av input fra tidligere celler.

Figur 2.12 viser tankegangen bak et RNN. Fra og med data-input nr. 2 ($X_1 ... X_t$) blir hver klassifisering utført basert på tidligere hendelser i tillegg til ny input. Et RNN kan bli sett på som flere versjoner av samme nettverk, hvor hver versjon sender en melding til neste. Det er nettopp denne meldingen som er forskjellen fra vanlige nevrale nettverk, og skal bidra til at nettverket tar hensyn til tidligere hendelser når klassifiseringer blir foretatt.



Figur 2.12: Et utrullet recurrent neural network[8]

Det er også denne lenke-strukturen som gjør den passende for sekvensiell data.

2.5.1 Utfordringer med RNN

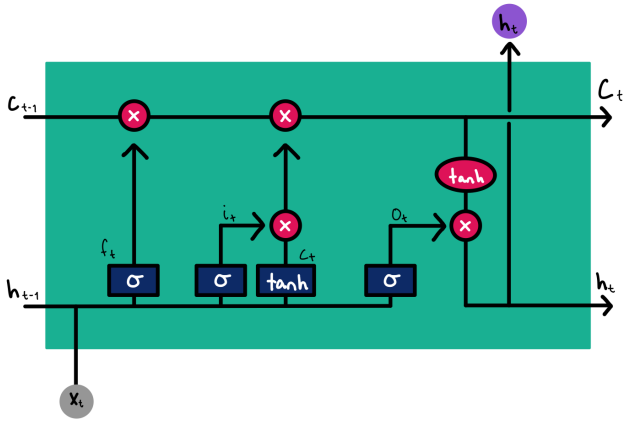
Det viser seg at RNN fungerer godt ved korte sekvenser med data, men har som beskrevet av Fridman [7] problemer med å huske dersom sekvensene blir for lange. Det vil si, de kan få problemer med å bære informasjon over lengre tid. Grunnen til at RNN lider av dette problemet er det man kaller for det forsvinnende gradientproblemet ved tilbakepropagering, som er nevnt i delkapittel 2.3.5. Gradienter er verdier som brukes til å oppdatere nettverkets vekter, og problemet oppstår som følge av at gradienten blir svært liten etterhvert som man går bakover i tid for å oppdatere vektene. Større gradienter vil bety hyppigere justering av vektene, og desto mer vil nettverket lære. Samtidig, dersom en har for små (neglisjerbare) gradienter vil nettverket slite med å justere vektene riktig, og dermed lære mindre. RNN-LSTM-nettverket er en type RNN som skal prøve å løse dette problemet.

2.5.2 LSTM

RNN-LSTM er en videreføring av et Recurrent Neural Network som evner å huske lengre tilbake i tid enn et standard RNN. Forskjellen er at de har komplekse, interne porter som er i stand til å kontrollere informasjonsflyt.

Kjernekonseptet bak en LSTM-celle (Figur 2.13) er at den skal være i stand til å finne ut av hvilken informasjon som er relevant for læring, og fokusere på denne. Irrelevant informasjon som ikke utgjør et godt klassifiseringsgrunnlag filtreres bort. Denne filtreringen av irrelevant informasjon er det portene som står for; glemme-porten, input-porten og output-porten styrer informasjonsflyten gjennom cellen. Med andre ord er det LSTM-cellen som fungerer som 'minnet' i et nettverk, og er i stand til å holde på informasjon gjennom prosesseringen av en hel sekvens med vilkårlig lengde. Tidligere hendelser kan derfor ha innvirkning på senere input, og problemet med korttidsminne ved vanlige RNN-nettverk blir derfor løst.

[8]Hentet 25.04.21 fra *https://colah.github.io/posts/2015-08-Understanding-LSTMs/*



Figur 2.13: Eksempel på en LSTM-celle

2.5.3 LSTM-Portene

Glemmeporten (f_t) bestemmer hvilken informasjon som beholdes eller forkastes. Informasjon fra forrige hidden layer (h_{t-1}) og nåværende data-input (X_t) blir input til en sigmoid-funksjon, som produserer en verdi mellom 0 og 1. Desto nærmere verdien kommer 0, desto mer skal glemmes. Disse verdiene blir punktvis multiplisert med forrige celle-tilstand (C_{t-1}) for å avgjøre hva som skal glemmes og ikke.

Inputporten (i_t) oppdaterer selve cellen. På inputporten benyttes både en sigmoid-funksjon og en tanh-funksjon. Input til begge funksjonene er input fra tidligere hidden layer (h_{t-1}) og nåværende data-input (X_t). Output fra sigmoid-funksjonen bestemmer hvilke verdier som skal oppdateres; desto nærmere 1, desto viktigere er verdiene. Tanh-funksjonen brukes for å justere nettverket. Outputen fra sigmoid-funksjonen (i_t) og outputen fra tanh-funksjonen ($\tilde{C}_t$) blir deretter multiplisert. Outputen fra sigmoid-funksjonen bestemmer altså hvilken som er verdt å huske fra tanh-funksjonen. Denne outputen blir punktvis addert med outputen fra glemmeporten og tidligere celle-tilstand. Dette resultatet blir den nye celle-tilstanden. Vi har altså at:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t,$$

hvor C_t er den nye celle-tilstanden.

Outputporten bestemmer hva den nye hidder-layer verdien skal bli. På denne porten blir nåværende data-input (X_t) og forrige hidden-layer verdi (h_{t-1}) prosessert av også her en sigmoid-funksjon. Denne outputen blir multiplisert med outputen fra en tanh-funksjon som prosesserer den nye celle-tilstanden (C_t). Output fra denne multiplikasjonen blir den nye hidden-layer verdien. Vi har altså at:

$$h_t = o_t * tanh(C_t),$$

hvor h_t er den nye hidden-layer tilstanden.

For å oppsummere består et LSTM-nettverk av en rekke typer porter som skal styre informasjonsflyten og sørge for at relevante tidligere hendelser blir tatt med videre. For

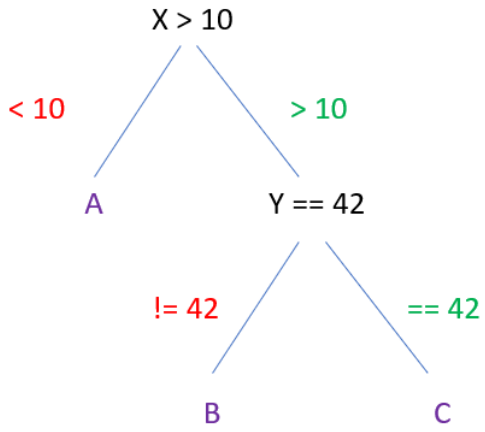
det første har en glemmeporten, som bestemmer hva som er relevant å beholde av tidligere data. For det andre har man inputporten, som bestemmer hva som er relevant å beholde fra nåværende tidspunkt. Den tredje porten er outputporten, som bestemmer hva neste hidden layer-tilstand skal være. Disse tre portene står for filtreringen av informasjon basert på relevans og viktigheten av å ha den med videre i nettverket.

2.6 Random Forest Classification

Random Forest Classifier (RFC) er en maskinlæringsalgoritme som brukes til kvalitativ analyse av data. Den brukes ofte i maskinlæringsproblemer hvor målet er å dele data inn i klynger basert på dens egenskaper. Det er en videreføring av mer enkle beslutningstrær som benytter seg av ulike metoder for å redusere varians og overtilpasning.

2.6.1 Beslutningstre

Et beslutningstre er en vanlig maskinlæringsalgoritme som egner seg godt til klassifisering med flere ulike klasser. Det kan også fungere godt for regresjon, men vi holder oss til multiklasse-klassifisering i denne oppgaven. Formålet med et beslutningstre er å dele opp dataene ved hjelp av ulike spørsmål eller parametere som kan splitte dem fra hverandre og til slutt klassifisere dem. For vår RFC modell bruker vi CART-modellen for hvert individuelle beslutningstre [27]. Dette er et binærtre som splitter dataene på spørsmål som kan besvares som sanne eller usanne. Et eksempel på et slikt tre kan man se i Figur 4.3.



Figur 2.14: Et generelt beslutningstre

Splittingen av dataene skjer rekursivt nedover i treet så lenge en gitt sluttbetingelse ikke er møtt. Ofte er denne sluttbetingelsen satt til å være slik at man stopper splittingen av noder når man ikke lenger oppnår en like stor grad av informasjonsutbytte, eller at man har nådd en tilfredsstillende grad av sikkerhet i klassifiseringen i den gjeldende noden. Informasjonsutbyttet kan sees på som et mål på hvor mye en splittelse av data senker usikkerheten i noden. I denne oppgaven har maskinlæringsbiblioteket scikit-learn blitt brukt til å lage trær. Scikit-learn bruker gini-urenhet som standard verdi til å måle informasjonsutbytte. Dette er et mål på hvor ren en gitt node er. Formelen for gini-urenhet er

gitt ved:

$$Gini = \sum_{i=1}^{J} p(i) * (1 - p(i)),$$

hvor $p(i)$ er sannsynligheten for å plukke ut et datapunkt fra klassen i den gitte noden og J er det totale antallet klasser.

Dette kan brukes til å regne ut informasjonsutbyttet ved splitting av dataene på et vilkårlig attributt. En lav gini-urenhet tilsvarer en god splittelse. For å finne hvilket attributt det er best å splitte på, regner vi ut gini-verdien til de to nye nodene som splittelsen ville ført til. Dette gjøres ved en vektet utregning. For å finne informasjonsutbyttet bruker man de vekta versjonene av gini-urenhetene til de resulterende nodene og subtraherer dem fra den originale nodens gini-urenhet. La U være informasjonsutbyttet man får av en split, G være gini-urenheten til en gitt node, N være totalt antall datapunkter og n et utvalg av de totale datapunktene. Da har en følgende formel:

$$U = G_{start} - (G_{venstre} * (n_{venstre}/N)) - (G_{hoeyre} * (n_{hoeyre}/N))$$

Man går gjennom alle attributtene helt til man finner den som gir det høyeste utbyttet og bruker denne.

2.6.2 Bagging og bootstrapping

Når man trener beslutningstre-modeller hver for seg kan de få svært ulike resultater med bare små endringer i datasettene de trener seg på [15]. Dette er et problem ettersom man helst vil ha mest mulig konsekvente svar og utfall fra en maskinlæringsmodell. For å prøve å redusere dette problemet benytter en RFC-modell seg av bagging. Bagging går ut på å trene et antall ulike modeller på hvert sitt utvalg av et felles datasett. Deretter går man over hvert enkelt resultat og teller opp hvilket utfall som var det vanligste. Det vil være det vanligste svaret som til slutt blir valgt som det endelige svaret. Under følger en mer detaljert gjennomgang:

La D være et tilfeldig datasett og n være antall modeller som skal trenes. La også D' være et tilfeldig utvalg av datapunkter fra det opprinnelige datasettet D. Hvert datasett D' bygges opp av individuelle tilfeldig utvalgte datapunkter med tilbakelegging fra D. Disse utvalgene gis så til hver sin modell. Dette kalles bootstrapping. Modellen trener så basert på det gitte datasettet. Når alle modellene har trent på hvert sitt datasett, sender man inn testdata. Dataene sendes til hver sin modell som legger klassifiseringene i en tabell. Til slutt velges klassifiseringen med flest forekomster som det endelige svaret. Denne vil med en større sannsynlighet være det riktige svaret enn om man hadde gått for én enkelt modell sitt svar. Dette fordi beslutningstrær er sensitive til endringer i datasettet, som fører til at to ganske like datasett kan gi helt forskjellig svar. Bagging reduserer altså variansen. Ettersom å legge til flere trær reduserer variansen, vil modellen bare bli bedre jo flere trær man legger til, frem til et visst punkt hvor forbedringen til slutt flater ut og videre økning av antall bare øker utregningskostnaden [6].

2.6.3 RFC - kombinér metodene

RFC kombinerer de ovennevnte teknikkene, i tillegg til en siste som fullfører hele modellen. Man ser kanskje for seg at dersom man gror mange nok trær og benytter bagging, så vil dette være tilstrekkelig for å unngå overtilpasning til datasettet, men det viser seg at dette ikke nødvendigvis er tilfellet. Se for deg at det finnes en veldig sterk predikator (attributt som splitter dataen veldig godt) samt andre ikke fullt så gode predikatorer. Siden beslutningstrær bruker grådig utvelgelse av hvilket attributt man skal splitte på, vil denne predikatoren mest sannsynlig være førstevalget til de fleste trærne. Dette fører til at man får mange relativt like trestrukturer, tross bruken av bagging og bootstrapping. For å løse dette problemet, unnlater man å la modellen vurdere alle predikatorene ved hver node. Det vanligste er å la hver node få et utvalg av $\sqrt{N}$ predikatorer som den kan velge mellom, hvor N er det totale settet med predikatorer [15]. Dette vil føre til at trærne blir mindre like og en unngår overtilpasning ved dette spesialtilfellet [15].

For å oppsummere går man gjennom følgende steg for å lage og ta i bruk en RFC-modell. For det første tar en i bruk bootstrapping for å lage n antall utvalg av det totale datasettet. Videre gis hver av de n ulike tremodellene et av de n utvalgene av datasettet. Modellen trenes så med tilfeldig valgte predikatorer, før en sender inn data til den ferdigtrente modellen og bruker den mest rapporterte klassifiseringen som det endelige svaret.

2.7 Systemutviklingsmetodikk

Systemutviklingsprosjekter kan variere i både form og størrelse, og kan involvere alt fra en til flere hundre utviklere som jobber på samme prosjekt. Tidlig historisk systemutvikling benyttet *vannfallsmodellen*, en prosess hvor hver fase i utviklingen (design, utvikling, testing, utgivelse) utføres sekvensielt. Dette fører imidlertid ofte til krevende og dyre omveltninger dersom kravene til systemet blir endret underveis i utviklingsprosessen.

For å bøte med denne problemstillingen, ble *The Agile Manifesto* [37] etablert i 2001, med hovedprinsippene

- Personer og samspill fremfor prosesser og verktøy
- Programvare som virker fremfor omfattende dokumentasjon
- Samarbeid med kunden fremfor kontraktsforhandlinger
- Å reagere på endringer fremfor å følge en plan

i fokus. Dette ble utgangspunktet for flere *agile* systemutviklingsmetodikker, som baserte seg på å være *iterative* for å være i stand til å kunne reagere på kravendringer underveis i utviklingsprosessen. Blant de mest populære metodene finner man bl.a. Scrum, Kanban og XP (Extreme Programming).

3 Valg av teknologi og metode

3.1 Teknologi

3.1.1 Frontend-rammeverk

Ved valg av frontend-rammeverk for utviklingen av systemets brukergrensesnitt ble det vektlagt kvaliteter som effektivitet, fleksibilitet og tilgjengelighet for sluttbruker. Et alternativ var å lage systemet som en skrivebordsapplikasjon ved å bruke et python GUI-rammeverk som f.eks. PyQt5 eller Tkinter. Selv om dette kunne vært aktuelt med tanke på sensorkommunikasjon som kunne ha foregått rett i applikasjonen, ble det stilt spørsmål om et slikt gui-rammeverk var like kraftig og fleksibelt som f.eks. React. Dessuten måtte et slikt rammeverk læres fra bunnen av, noe som hadde tatt tid og ført til et potensielt dårligere resultat, i og med at det er ny teknologi som gruppen ikke var vant med å bruke. I startfasen av prosjektet ble det diskutert om systemet skulle kunne benyttes på mobile enheter. Et alternativ var å benytte React, et kraftig og fleksibelt frontend-rammeverk med potensiale for å kjøre på mobile enheter. React var også et rammeverk samtlige av gruppens medlemmer var kjent med fra før, og håndterer godt. Dessuten kan man bygge et React-prosjekt som en skrivebordsapplikasjon via rammeverket Electron. På grunn av allerede etablert kjennskap til React, samt dets fleksibilitet og evne til å lage svært kapable og gode grensesnitt, falt valget på React.

3.1.2 Backend-rammeverk

For å ha muligheten til å kommunisere med sensorene via Openzen-APIet er man avhengig av å kunne kjøre python-kode. Ettersom det ble bestemt at React (som kjører JavaScript/TypeScript) skulle benyttes som frontend-rammeverk, trengte man en måte å få kjørt Python-filer på. To alternativer ble raskt etablert, hvor den ene gikk ut på å benytte seg av operativsystemets pipelines, som er en forbindelse mellom to kommuniserende prosesser. Det andre alternativet gikk ut på å kjøre lokale nettverkskall mot localhost, via en Python-server. Ettersom pipelines satte visse begrensninger og var, etter vår mening, for lite fleksibelt for vårt bruk, ble lokal Python-server en bedre løsning med tanke på dets fleksibilitet.

3.1.2.1 Maskinlæring

I avsnitt 3.2 blir de forskjellige maskinlæringskategoriene; veiledet, ikke-veiledet og forsterket læring presentert. Oppgavens natur passer best under kategorien veiledet læring ettersom det er ønskelig å modellere en sammenheng mellom inputdata og klassifikasjoner. Etter møtet med Børke (Vedlegg D), hvor det ble etablert at en ønsket å se på variasjon i sittestillinger, ble det bestemt at det skulle klassifiseres på ni ulike stillinger. Dette medfører at problemet er av typen multiklasse-klassifisering, og ikke binær, som kunne fungert om en kun skulle klassifisert stillinger som gode eller dårlige. Typiske maskinlæringsalgo-

ritmer for multiklasse-klassifisering er nevnt i delkapittel 3.2. Etter testing fant vi ut at Random Forest Classifier (RFC) presterte best på vår data av de klassiske maskinlæringsalgoritmene.

Videre ble det funnet flere forskningsartikler som benytter seg av nevrale nettverk for menneskelig aktivitetsgjenkjenning (eng: human activity recognition). Artiklene av Murad & Pyun [22] og Pienar & Malekian [28] har funnet at nevrale nettverk presterer svært godt på denne typen oppgaver, og har oppnådd svært høye nøyaktigheter. Nettverkene som går igjen i disse er ANN, RNN-LSTM og CNN. Siden disse nevrale nettverkene har oppnådd gode resultater på liknende oppgaver, bestemte vi oss for at disse også var verdt å utforske for vårt prosjekt.

3.2 Metode

3.2.1 Prosess

Ved prosjektets oppstart ble det etablert at prosjektet skulle deles inn i fem *faser* som skulle fungere som en veiledende fremdriftsplan. Disse fasene var

1. **Oppstartsfase (11. - 22. Jan)**
   Gjøremål: Oppstartsmøte, maskinlæringsteori, opprette visjonsdokument, installere nødvendige programmer
2. **Datainnspilling- og behandlingsfase (25. Jan - 11. Feb)**
   Gjøremål: Spille inn og behandle data slik at den kan fôres til en maskinlæringsmodell. Møte med fysioterapeut fra St. Olavs for å diskutere forskningsgrunnlag for oppgaven.
3. **Realtime-klassifisering og backend (11. Feb - 7. Mar)**
   Gjøremål: Få til realtime-klassifisering, utforske maskinlæringsalgoritmer og opprette backend-funksjonalitet som iverksetter og stopper realtime-klassifisering samtidig som sensordata blir samlet inn og behandlet.
4. **Frontend (8. Mar - 12. Apr)**
   Gjøremål: Implementering av klientapplikasjon i React
5. **Utbedring av applikasjonen og forskning (13. Apr - 2. Mai)**
   Gjøremål: Forbedringer av systemet, brukertester, forskning på algoritmer og dokumentasjon.
6. **Dokumentasjon (3. Mai - 18. Mai)**
   Gjøremål: Dokumentasjon og fokus på hovedrapport.

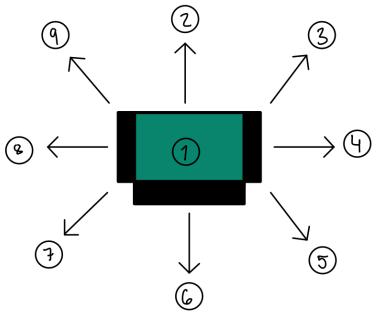
Det er verdt å merke seg at selv om en til enhver tid skulle være inne i en fase, vil man ved å se på vedlagt Gantt-diagram kunne se at det er noe overlapp. Det ble også inkludert to dagers margin før innleveringsfrist.

3.2.2 Datainnsamling- og prosessering

En maskinlæringsmodell trenger data for å kunne trenes opp til å kjenne igjen mønstre og være i stand til å klassifisere ny usett data. Det ble forsøkt å oppdrive allerede eksisterende datasett, men det ble raskt konkludert med at det beste var å spille inn data selv. Datasettene ble spilt inn med sensorer festet på kroppen til forskjellige testpersoner, som sender data over bluetooth. Videre må dataene behandles på riktig måte, og annoteres før den kan sendes inn til maskinlæringsmodellene. Her har man altså to arbeidsprosesser; en datainnsamlingsprosess, og en databehandlingsprosess.

3.2.2.1 Datainnsamling

Det ble raskt etablert at det er ønskelig med et relativt stort datasett, ettersom dette er nødvendig for at modellene skal kunne generalisere godt nok. Nevrale nettverk er spesielt avhengige av større datamengder for å generalisere og oppnå god treffsikkerhet. Innledningsvis i prosjektet ble det spilt inn trenings- og testsett på gruppens tre medlemmer. Det ble bestemt at for å kunne måle variasjon var det ønskelig med en del posisjoner, og valget falt på ni posisjoner, som vist i Figur 3.1.



Figur 3.1: Illustrasjon av sittestillinger

Det er en del ting en må ta hensyn til når det kommer til innspillingen. For det første måtte en kalibrere sensorene riktig slik at samtlige sensorer hadde samme relative nullpunkt i forhold til orientering. For det andre ble det erfart at sensor-id, som er viktig for å kunne skille mellom sensorenes plassering på ryggen, ikke ble lagret på sensorene med mindre man aktivt gjorde dette i innspillingsverktøyet. For det tredje ble det oppdaget at ved høyere frekvenser ble det et relativt ujevnt antall rader, opptil 10% differanse fra sensor til sensor. Dette fordi sensorene benytter seg av upålitelig overføring, altså at de sender data hyppig og håper på at den kommer frem, noe støy kan hindre den i å gjøre. Derfor falt valget på 50Hz, som ga en god avveining mellom tap av data underveis og innspillingshastighet. Det ble også tatt høyde for at andre bluetooth-enheter kunne være forstyrrende, så disse ble fjernet under innspilling, og en satt igjen med et neglisjerbart tap av data. Når en hadde dette på plass fant en videre testpersoner både ved NTNU og av bekjente, og det ble spilt inn totalt ti treningssett og ti testsett.

Testpersonene ble rekruttert av veileder innad på NTNU, og av gruppens bekjente. På grunn av den pågående pandemien var det ikke like lett å få tak i testpersoner fra kretser utenfor disse. Testpersonene var friske voksne mellom 20-50 år, med en median på 23 år og en relativt balansert kjønnsfordeling. Studiet er blitt godkjent av Norsk senter for forskningsdata (NSD), og samtykkeskjema for hver av deltakerne har blitt skrevet under og tatt vare på etter deres retningslinjer. Avslutningsvis er alle testpersonene anonymisert med identifikasjonsnummer av hensyn til personvern.

3.2.2.2 Databehandling

Rådata

Fra sensorene får en produsert følgende datafelter:

- Akselerometerdata (x,y,z-akse)
- Gyroskopdata (x,y,z-akse)
- Magnetometerdata (x,y,z-akse)
- Eulervinkler (x,y,z-akse)
- Quaternions (4 dimensjoner)
- Trykk (kPa)
- Temperatur
- Høydemeter og hevebevegelse

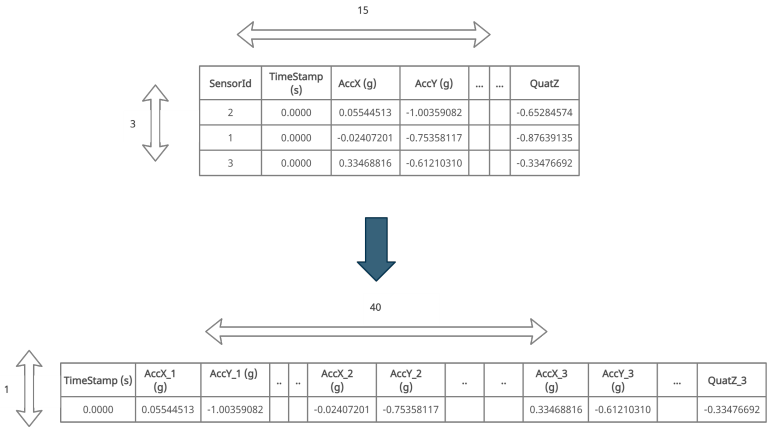
I tillegg til disse feltene sender også sensorene med sensor-id, timestamp og frame-nummer for hver rad. Dette er for å kunne identifisere sensorene, samt hvor hendelsen befinner seg i forhold til andre rader med data. Av datafeltene ble akselerometerdata (både lineær og angulær akselerasjon), gyroskopdata og quaternions inkludert ettersom de ble sett på som mest aktuelle for å beskrive hvordan man sitter. Magnetometerdata kan på grunn av diverse kilder til magnetfelt gi forskjellige verdier alt etter hvor man befinner seg. Det ble derfor luket ut ettersom det ikke er ønskelig at hvor man sitter skal ha innvirkning på klassifiseringen, samtidig som at det er ønskelig å kunne ha på seg for eksempel hodetelefoner eller sitte med telefon uten at dette skal ha noen påvirkning. Videre ble quaternions valgt for å beskrive rotasjon fremfor Eulervinkler, ettersom produsenten av sensorene sterkt anbefalte bruk av quaternions fremfor eulervinkler på grunn av risiko for potensiell gimbal-låsing [19]. Figur 3.2 viser de aktuelle feltene for vår brukssituasjon.

SensorId	TimeStamp (s)	AccX (g)	AccY (g)	AccZ (g)	GyroX (deg/s)	GyroY (deg/s)	GyroZ (deg/s)	LinAccX	LinAccY	LinAccZ	QuatW	QuatX	QuatY	QuatZ

Figur 3.2: Utvalgte felter fra sensordata

Flere sensorer og sammenslåing av rader

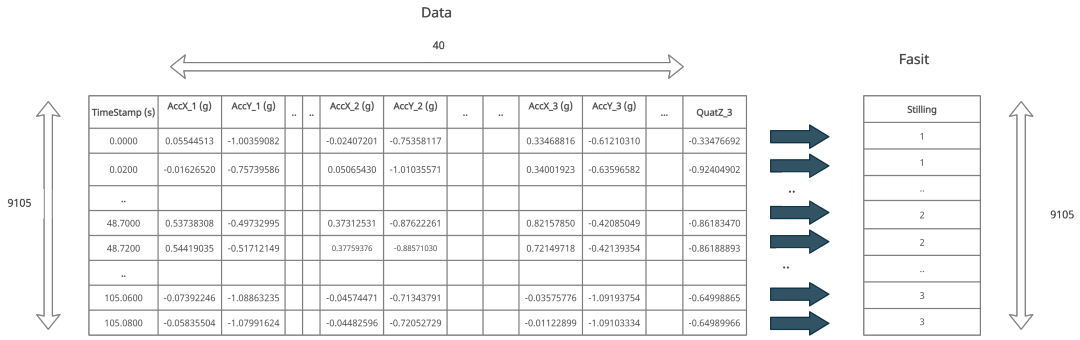
Dersom det er ønskelig å benytte seg av flere sensorer på samme tid, er man avhengig av at samtlige av sensorene er synkronisert på timestamp, og dermed har rader som viser data fra samme tidspunkt. Ved å benytte seg av Openzens synkroniseringsfunksjon var dette oppnåelig under innspilling, og man fikk dermed rådata fra tre forskjellige sensorer på forskjellige posisjoner på samme tid. Disse tre radene kunne dermed slås sammen til én rad på samme timestamp, slik at flere sensorer kunne benyttes sammen for å klassifisere sittestillinger.



Figur 3.3: Sammenslåing av sensordata fra flere sensorer

Annotering

For at en veiledet maskinlæringsalgoritme skal trene må man i tillegg til selve dataen også ha et fasitsvar for hvilken stilling man sitter i for hver sensormåling. For å tilegne hver sensormåling riktig sittestilling ble videoannoteringsverktøyet Anvil [2] benyttet. Ved å annotere opptak kan man generere en tekstfil som forteller hvilken stilling testpersonen sitter i til enhver tid under opptak. Videre kan disse tidspunktene fra denne tekstfilen sammenlignes med tidspunktene fra sensormålingene, og på denne måten produsere en fasittabell med riktig sittestilling for hvert tidspunkt. Denne prosessen er illustrert i Figur 3.4, hvor en tabell med timestamp og data som utgjør testgrunnlaget, ofte omtalt som x_test, og en fasittabell som ofte omtales som y_test, sammen utgjør et testsett.



Figur 3.4: Sensordatamålinger og tilsvarende sittestilling

3.3 Arbeids- og rollefordeling

Under prosjektetperioden har gruppen strevet etter en så jevn fordeling av arbeidsoppgaver som mulig, med mulighet for ekstra arbeid på eget initiativ. Arbeid har også blitt gjennomført i plenum ved større oppgaver slik at samtlige til enhver tid er oppdatert på hvordan gruppen ligger an og har god kontroll på oppgavedomenet. Et godt eksempel på dette er arbeidet med databehandling, hvor det ble jobbet sammen slik at alle hadde samme grunnleggende forståelse for hvordan den preprosesserte dataen så ut. Parprogrammering er også en god teknikk hentet fra de agile arbeidsmetodikkene, og det ble under flere deler av prosjektet sett på som en god løsning at et flertall av gruppens tre medlemmer programmerte sammen.

Underveis ble roller dynamisk fordelt etter behov. Teamet gjennomførte et godt teoretisk forarbeid i starten av prosjektet slik at alle skulle være i stand til å takle de ulike arbeidsoppgavene som måtte gjøres, og oppgavene ble løpende fordelt underveis. Dette sørget for god flyt i arbeidet hvor man ikke trengte å vente på at andre teammedlemmer ble ferdig med sitt arbeid for å kunne utføre egen arbeidsoppgave.

3.4 Systemutvikling

Systemutviklingsfasen ble originalt planlagt å vare i en måned. Før selve implementeringen ble iverksatt, ble et utkast av brukergrensesnittet skissert med skisseringsverktøyet Figma [34], i tillegg til at user-stories ble laget for kravdokumentasjon. Dette førte til at gruppen hadde et klart bilde på hva som skulle implementeres, hvordan det skulle se ut og fungere, og hvordan det måtte implementeres.

Ved valg av systemutviklingsmetodikk falt valget på Kanban. En agil systemutviklingsmetodikk ble ansett som nødvendig for å kunne respondere på kravendringer underveis i utviklingsprosessen. Scrum var et annet alternativ, men gruppens interne kommunikasjon og arbeidsfordeling fungerte såpass godt at Scrums roller, seremonier og artefakter ble ansett som unødvendig, og hadde dessuten ført til mer unødvendig arbeid. Et annet argument her er at det kun er tre medlemmer i gruppen, og Scrum sine ekstra elementer er mer et verktøy for å forbedre kommunikasjonsflyten i større team.

Ved å benytte et Kanban-board på Github ble arbeidsoppgaver fordelt innad i gruppen etter ønske. For å begrense kompleksitet og minske arbeidsmengden ved implementering av klientprogrammet, ble det fokusert på å implementere gjenbrukbare komponenter som kunne benyttes flere ganger. Serverens endepunkter ble kontinuerlig implementert parallelt med klientapplikasjonen, og fulgte den sekvensielle rekkefølgen for tilkobling, kommunikasjon og frakobling av sensorer. Først fikset man tilkobling av sensorer, så starting/stopping av klassifisering, før en avsluttet med å fikse frakobling av sensorer. Ytterligere ønsket grensesnittfunksjonalitet ble så utviklet etter at liveklassifisering hadde blitt implementert og grundig testet. På denne måten ble det en agil prosess med utvikling av komponenter, testing, og integrering.

4 Resultater

4.1 Ingeniørfaglige resultater

I oppstarten av prosjektet ble funksjonelle og ikke-funksjonelle krav til produktet beskrevet i et visjonsdokument. De fleste av disse er gjennomført, mens andre har blitt forkastet til fordel for mer fokus på områder som ble ansett som viktigere. Det er verdt å merke seg at gruppen har gått bort fra målet om å bestemme om en sittestilling er god eller dårlig, etter innspill fra Børke (Vedlegg D) som førte til en omformulering av oppgaveteksten. I denne seksjonen følger en oversikt over målene og status i forhold til disse.

4.1.1 Presentasjon av data

Ved prosjektets start ble det definert som krav at en skulle ha en eller flere maskinlæringsmodeller som var i stand til å klassifisere sittestillinger, i tillegg til å skulle kunne fremstille trender grafisk og eventuelt vise til grad av god/dårlig sittestilling. Teamet står ved slutten av prosjektet igjen med fire modeller, én som tar i bruk Random Forest Classifier, et standard kunstig nevralt nettverk, et konvolusjonelt nevralt nettverk og et nettverk av typen Long Short-Term Memory. Disse blir videre presentert under vitenskapelige resultater. Ved hjelp av den utviklede applikasjonen kan en spille inn data, som videre blir klassifisert i sanntid og presentert grafisk for å kunne se på distribusjon og variasjon over tid. Sammen med presentasjonen av klassifiseringene kan en også se kommentarer skrevet av brukeren selv etter avsluttet sesjon. På denne måten kan en se på sammenhengen mellom variasjon av sittestilling og hvordan brukeren opplever ryggen sin. Avslutningsvis er det siste målet blitt forkastet, som nevnt fordi oppgaven ble endret ettersom forskning innen fagfeltet tilsier at det er et mål uten hensikt. Løsningen på dette ble utarbeidet sammen med oppgavestiller, og dette er blant annet grunnen til at ideen om å knytte brukerens egen opplevelse av ryggen sin opp mot variasjon av sittestilling i løpet av innspillingen, ble til.

4.1.2 Backend

Det ble i utgangspunktet satt mål som omhandlet å lagre informasjon om brukeren (sensordata) i en database. Det ble imidlertid senere bestemt at det var mer ønskelig å utvikle produktet som en lokal skrivebordsapplikasjon, som medførte at det var unødvendig å ha noen form for sentralisert lagring. En kunne istedenfor lagre data lokalt i filer, som både ga mer mening gitt hensikten til applikasjonen og medførte mindre hensyn å ta i forhold til personvern og GDPR. Videre ga det da heller ikke mening å ha autentisering i form av innlogging på en nettside, og dette kravet ble forkastet. Dette ga gruppen bedre tid til å jobbe med mer ønsket funksjonalitet som rapportering fra brukeren og mer oversiktlige grafiske fremstillinger i applikasjonen.

4.1.3 Frontend

Når det kommer til brukergrensesnittet ble det lagt vekt på spesielt tre områder; applikasjonen skulle gi en indikator på sittestilling i sanntid og være lett å bruke, den skulle vise historikk over holdning for en gitt periode, samt gi tilbakemelding til brukeren ved dårlig holdning. Det ble utviklet en Progressive Web Application (PWA) som resultat av at gruppen ville benytte seg av React som frontend-rammeverk, samtidig som at applikasjonen skulle være lokal. Applikasjonen har funksjonalitet for å starte og stoppe opptak av data, samt å klassifisere i sanntid basert på data den mottar. Videre blir klassifikasjonene så brukt til å plotte grafer om brukerens sittestilling over tid. Disse to kravene ansees derfor som oppnådde.

Etter dialog med oppgavestiller, ble det besluttet at målene som omhandlet å varsle brukeren ved dårlig holdning, samt å gi belønninger til bruker var lite hensiktsmessige etter oppklaringen fra fysioterapeut Børke (Vedlegg D), og endring av oppgaven som følge av dette møtet. Funksjonalitet som omhandlet rapportering fra bruker ble samtidig innført som krav, ettersom dette ble ansett som nødvendig for å kunne se sammenhenger mellom variasjon i sittestillinger og hvordan det faktisk påvirker ens rygghelse.

4.1.4 Ikke-funksjonelle egenskaper og andre krav

Blant ikke-funksjonelle egenskaper og krav finner man at produktet måtte ha høy nøyaktighet på klassifiseringen, samtidig som at pålitelighet, oversiktlighet og fleksibilitet var sentrale punkter. Når det kommer til nøyaktighet har teamet oppnådd en gjennomsnittlig treffsikkerhet på over 80% for samtlige av modellene, og med noen nærmere 85% og oppover i gjennomsnitt over ni testsett. Dette kravet ansees dermed som oppnådd, selv om en med maskinlæring ikke kan være helt sikker på at den beste modellen er funnet. Dette blir videre diskutert i kapittel 5. Underveis i utviklingen av applikasjonen har det blitt utført testing av datainnsamling for å sikre pålitelighet i koblingen mellom applikasjonen og sensorene, i tillegg til brukertester for å forsikre at produktet både er fleksibelt, intuitivt og oversiktlig. Under disse brukertestene har elementer som tilgjengelighet (WCAG) og Universell Utforming blitt vektlagt, og ting som tabbing og fokus har blitt utbedret. I tillegg var det i første test tegn til usikkerhet om hvordan en skulle tolke klassifiseringene i diagrammene, samt sensorenes plassering på kroppen. I kjølvannet av denne testen ble oppkoplingsprosessen gjennomgått og designet på nytt, med klar beskjed til brukeren om hvor hver sensor skal plasseres. Videre ble nye tooltips utviklet med klassifiseringens tidspunkt og tekstlige beskrivelse i fokus, istedenfor et tall som ikke sier brukeren stort. Dette ble testet i neste brukertest, og gir oss grunn til å kunne beskrive disse kravene som oppnådd.

4.2 Administrative resultater

4.2.1 Planlegging og tidsbruk

Som nevnt i kapittel 3.2.1 om prosessen, ble prosjektet delt opp i seks faser. Det ble i løpet av første prosjektuke satt opp et Gantt-diagram (vedlagt i prosessdokumentasjonen) som beskriver en tenkt arbeidsfordeling. Ettersom gruppen hadde et fag i tillegg til bacheloroppgaven de første åtte ukene, ble det bestemt at en skulle sette av noen dager i uken til dette. Etter endt eksamensperiode i uke 11 kunne en øke arbeidsmengden og fokusere fullt på dette prosjektet. Gantt-dragrammet viser en flytende fremgang i løpet av prosjektet, og selv om noen av fasene sklir litt over i hverandre har det fortsatt vært én fase i fokus til enhver tid.

Når det kommer til faktisk timebruk samsvarer dette veldig godt med Gantt-diagrammet. I praksis var det gode overganger mellom hver fase, noe som kan argumenteres for at skyldes hvordan fasene har blitt strukturert. For det første måtte en ha data om en i det hele tatt skulle kunne utføre noen form for maskinlæring. Dette ble spilt inn, og ved å spille inn data ble en kjent med sensorene. Videre var det naturlig å se på hvordan en kobler til sensorenes API, samt lage en enkel form for maskinlæringsmodell som en kunne teste på. I det en hadde klart klassifisere i sanntid, var det naturlig å begynne utvikle brukergrensesnittet. Når applikasjonen nærmet seg ferdig kunne en igjen ta opp og konsentrere seg om forskningen. Avslutningsvis var forskningen og dokumentasjon av denne sentral. Fasene ble derav naturlig til, og bidro til en god arbeidsflyt gjennom hele prosjektet.

Videre hadde en også god nytte av å ha litt overlapp i fasene. Her kan det blant annet trekkes frem hvordan en ved avslutningen av applikasjonsutviklingen kunne ha noen medlemmer som jobbet med brukertester og justeringer der, mens en annen jobbet med maskinlæringsmodellene. Denne dynamiske arbeidsformen bidro til at en fikk utnyttet ressursene i gruppen bedre, og variasjon når en hadde jobbet relativt ensformig med utvikling av brukergrensesnitt i en måned. Allikevel kan det argumenteres for at en skulle hatt enda mer overlapp når det kom til dokumentasjon, der gruppen kunne hatt nytte av å skrevet teori-delen av hovedrapporten samtidig som man satte seg inn i maskinlæringsteorien fra starten av. Dette har ikke bydd på store problemer under dokumentasjonsfasen, men dersom det hadde blitt gjort kunne man sluppet å måtte repetere teorien på nytt for teoriskrivingen sin del. Da kunne en heller ha skrevet på andre deler av rapporten når resultatene var klare, istedenfor å bruke tid på å skrive teori i slutten av prosjektet. En ser altså at det kan være ønskelig med konkrete faser som setter en arbeidsoppgave i fokus til enhver tid, samt noe overlapp for effektiv tidsbruk og god utnyttelse av ressursene i gruppen.

4.2.2 Systemutviklingsprosessen

Prosjektets fjerde fase var satt av til systemutvikling. Det ble satt av en måned til denne fasen, i tillegg til finpussing og brukertester som skulle overlappe med neste fase. Som nevnt

i kapittel 3.4 var det naturlig å bygge systemutviklingsfasen opp i forhold til den naturlige rekkefølgen sensorkommunikasjon foregår i. Etter å ha skissert og designet grunnstrukturen, var det naturlig å se på oppkobling mot sensorene. Endepunkter ble laget kontinuerlig ettersom en fikk bruk for dem i frontend, og sørget for at de ble laget på riktig måte fra starten av. Videre var det naturlig å få til å starte og stoppe klassifiseringen, og få denne presentert visuelt. Allerede etter to uker var dette på plass, og en begynte se på visualisering i grafer og historikk. Under et veiledningsmøte ble det ytret et ønske om at man må fylle ut en beskrivelse av status etter opptak, og dette ble utviklet mot slutten av denne fasen. Avslutningsvis hadde en brukertester og justerte på de tingene som ble avdekket der. Kanban-boardet i GitHub fungerte hele tiden som en oversikt over hva gruppemedlemmene jobbet med, og her kunne en legge inn diverse feil eller mangler. Kombinasjonen av denne oversikten og brukertester med justeringer underveis sørget for at tidsaspektet i fasen ble holdt.

4.3 Vitenskapelige resultater

Basert på problemstillingen gitt for dette prosjektet, har de ulike maskinlæringsalgoritmene blitt trent på data fra både en, to og tre sensorer. I dette kapittelet vil det bli redegjort for hvordan datasettene er blitt produsert og formatet på disse, i tillegg til resultater fra de ulike maskinlæringsmodellene. Ettersom nevrale nettverk starer treningen med tilfeldig initierte vekter er disse blitt kjørt tre ganger for hver sensormengde, for å kunne undersøke i hvor stor grad de ulike nettverkenes resultater varierer for hver kjøring. Resultatene fra hver maskinlæringsalgoritme vil bli videre diskutert i kapittel 5 - Diskusjon.

4.3.1 Data

Som nevnt tidligere i kapittel 3 - metoder og valg av teknologi, ble det spilt inn ti treningssett og ti testsett fra ti forskjellige personer for å få et variert datagrunnlag å trene maskinlæringsalgoritmene på. I dette underkapittelet følger informasjon om all innspilt data for dette prosjektet.

4.3.1.1 Treningsdata

De tre første treningssettene ble spilt inn i samme omgang med to minutter i hver sittestilling. Senere ble det besluttet å øke til tre minutter da en innså at en kunne spille inn data med samtlige sensorer, og kutte vekk de en ikke ønsket for å teste med færre enn tre sensorer. Dette ga også muligheten til å spille inn mer data på kortere tid. De fleste datasettene er spilt inn med 50 Hz for å inngå et kompromiss mellom innspillingshastighet og pålitelig overføring ved innspilling. To av settene er imidlertid spilt inn med 10 Hz, og ble ikke inkludert i treningen av modellene på grunn av usikkerhet til hvor nøyaktig det var spilt inn. Mye bevegelse og en annoteringsvideo som frøs er årsakene til dette. Videre ble det også valgt å ikke trene på treningssett 010 for å se hvordan algoritmene ville prestere

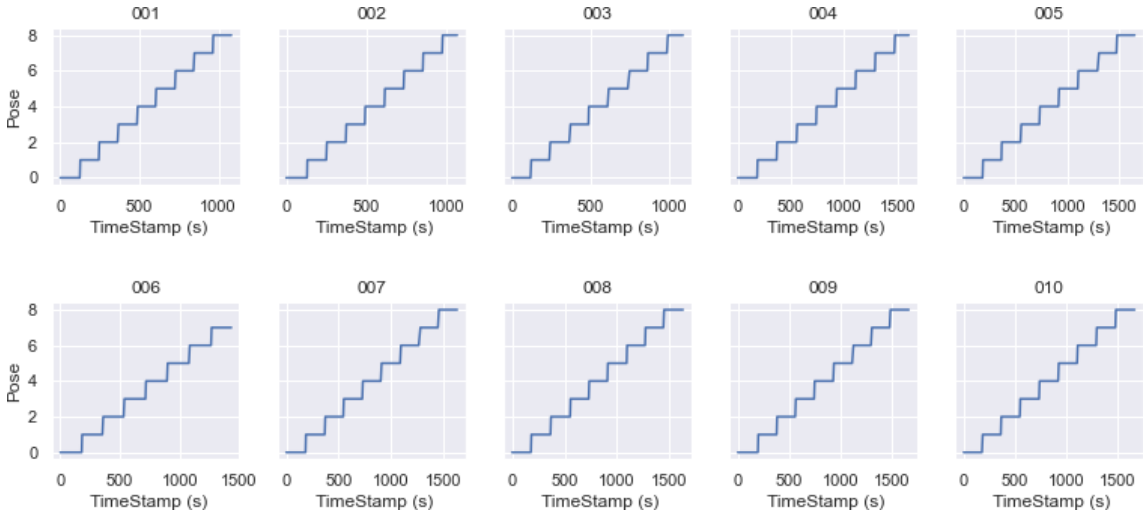
på data fra en helt ukjent person. Treningsdata som ikke ble trent på er merkert med grå bakgrunn i Tabell 4.1. Samtlige av de ti treningssettene er inkludert i prosjektstrukturen.

Treningsdata

Id	Antall rader	Innspillingsfrekvens	Minutter i hver posisjon
001	162297	50 Hz	2
002	160666	50 Hz	2
003	165067	50 Hz	2
004	248393	50 Hz	3
005	249085	50 Hz	3
006	243684	50 Hz	3
007	49042	10 Hz	3
008	49006	10 Hz	3
009	249877	50 Hz	3
010	250582	50 Hz	3

Tabell 4.1: Informasjon om innspilt treningsdata. Radene med grå bakgrunn er treningsdata som ble spilt inn, men som algoritmene ikke ble trent på

For å ha en så systematisk innspilling av treningsdata som mulig, i tillegg til å holde treningsgrunnlaget lett å holde oversikt over, ble det besluttet å spille inn samtlige treningssett med samme rekkefølge på sittestillingene som vist i Figur 4.1. Dette skulle ikke bringe med seg noen negativ effekt ettersom en kan stokke om radene under trening av maskinlæringsmodellene. En systematisk rekkefølge som var lett å følge over en halvtime med innspilling var prioritert for å prøve begrense feilkilder og forenkle prosessen for testpersonene, og ble derfor tatt i bruk på alle treningssettene.



Figur 4.1: Illustrasjon av rekkefølgen på sittestillingene under innspilte treningssett

4.3.1.2 Testdata

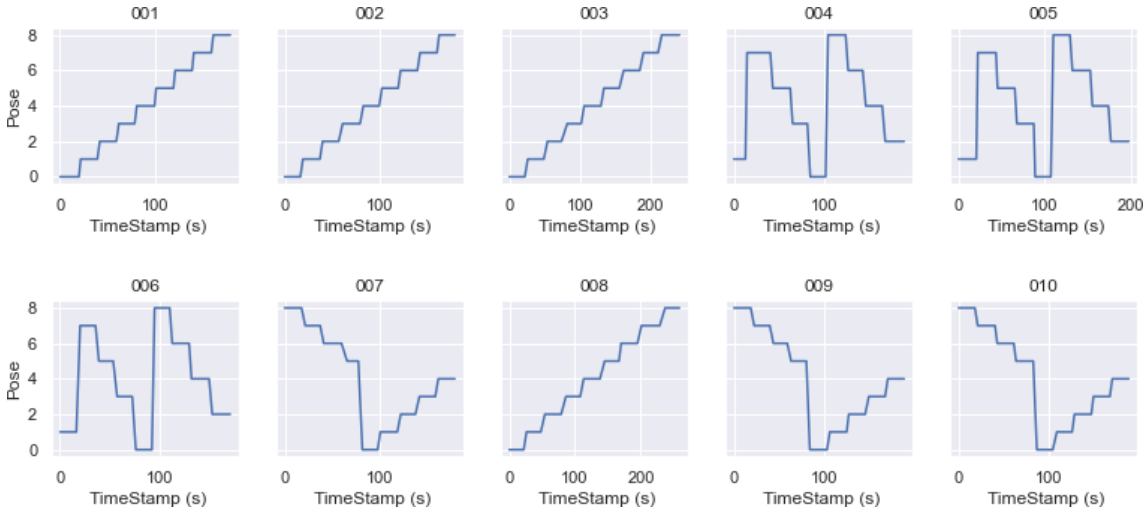
I motsetning til treningsdataen ble ikke testdataen spilt inn i samme rekkefølge - det var heller ønskelig å kunne observere hvordan modellene predikerer på testdata hvor rekkefølgen på sittestillingene er variert slik at en kan forsikre seg om at modellen ikke kun har lært seg rekkefølgen, men den faktiske sittestillingen basert på mønstre i datapunktene. En oversikt over de innspilte testsettene kan en se illustrert i Tabell 4.2.

Testdata

Id	Antall rader	Innspillingsfrekvens	Sekunder i hver posisjon
001	27061	50 Hz	20
002	27317	50 Hz	20
003	36221	50 Hz	20
004	28586	50 Hz	20
005	29788	50 Hz	20
006	25722	50 Hz	20
007	5410	10 Hz	20
008	3897	5 Hz	20
009	28698	50 Hz	20
010	28643	50 Hz	20

Tabell 4.2: Informasjon om innspilt testsdata

Forskjellige innspillingsfrekvenser ble også benyttet, med ett testsett på 5 Hz, ett på 10 Hz og resten på 50 Hz. En oppsummering av all innspilt testdata og rekkefølgen på sittestillinger under innspillingen av disse er gitt i Tabell 4.2 og Figur 4.2. I likhet med treningsdataen kan også all testdata finnes som CSV-filer i prosjektstrukturen.

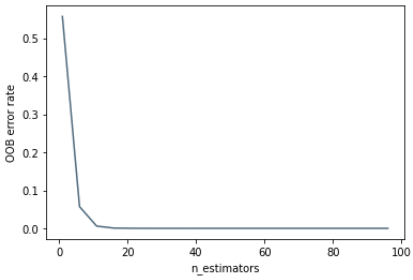


Figur 4.2: Illustrasjon av rekkefølgen på sittestillingene under innspilte testsett

4.3.2 RFC resultater

4.3.2.1 Antall trær

Fra teorien har vi at hver RFC-modell vil konvergere mot en øvre nøyaktighet ved et gitt antall trær. Dette antallet ble funnet ved hjelp av scikit-learn sitt OOB[9] feilestimat. Som vi ser av figuren konvergerer feilen ganske fort med dette testsettet og vi får bare minimale gevinster etter rundt 40 trær. Vi fant at den maksimale gevinsten for settet vårt lå på rundt 150 trær. Dermed blir det brukt 150 trær for alle modellene i dette kapittelet.



Figur 4.3: Out of bag error for økende antall trær

4.3.2.2 Treffrate for klassifisering med ulikt antall sensorer

Testsett / Antall sensorer	1	2	3
001	87.25%	80.09%	94.93%
002	45.50%	71.44%	76.92%
003	35.33%	79.06%	89.29%
005	77.22%	69.97%	78.08%
006	79.45%	79.97%	85.91%
007	58.77%	72.94%	85.22%
008	78.77%	35.57%	44.51%
009	70.51%	62.91%	88.49%
010	49.80%	77.74%	82.25%
Gjennomsnitt	64.74%	69.97%	80.62%

Tabellen viser oss at testsett 008 stikker seg negativt ut, og dette er grunnet feilkilder som blir tatt opp i diskusjonskapittelet av rapporten. Dermed kan det diskuteres om dette settet er bra nok til å kjøre tester på, men det ble inkludert for å illustrere hva som kan skje om dataene ikke samles feilfritt. Modellen klarer seg forøvrig en del bedre om man ser bort ifra testsett 008. Modellen med 1, 2 og 3 sensorer får da gjennomsnittlige treffrater på henholdsvis: 62.98%, 74.27% og 85.14%.

[9] OOB står for out of bag error, og er et mål på hvor godt modellen klarer å treffe på data fra treningssettet som ikke ble tatt med i et gitt tre.

4.3.3 ANN

Antall sensorer	**1**				**2**				**3**			
Kjøring/ Testsett	1	2	3	$\overline{X}^*$	1	2	3	$\overline{X}^*$	1	2	3	$\overline{X}^*$
001	85,54	90,57	84,80	**85,17**	73,25	66,02	67,90	**69,06**	77,29	76,92	82,40	**78,87**
002	62,26	65,56	64,27	**64,03**	78,17	71,15	70,97	**73,43**	88,52	89,07	88,72	**88,71**
003	34,34	37,43	34,70	**35,49**	97,63	98,50	99,01	**98,38**	81,47	90,76	96,96	**89,73**
004	70,81	70,67	72,13	**71,20**	90,21	90,22	90,18	**90,20**	89,29	89,79	89,72	**89,60**
005	83,04	77,48	76,39	**78,97**	99,31	98,30	99,19	**98,93**	98,86	98,53	99,05	**98,81**
006	66,90	67,17	64,85	**66,31**	98,21	98,46	98,46	**98,38**	92,21	91,99	96,62	**93,61**
007	69,57	66,60	67,75	**67,97**	84,62	83,00	85,16	**84,26**	84,14	85,02	85,43	**84,86**
008	88,08	88,36	93,02	**89,82**	59,96	50,93	52,98	**54,62**	21,79	35,85	32,87	**30,17**
009	84,16	90,20	90,41	**88,26**	84,08	83,86	84,10	**84,01**	91,27	91,90	92,63	**91,93**
010	44,59	44,21	44,78	**44,53**	48,22	48,84	52,90	**49,99**	74,78	72,98	80,44	**76,07**
$\overline{X}^*$	**68,93**	**67,52**	**69,31**	**68,59**	**81,37**	**78,93**	**80,09**	**80,13**	**79,96**	**82,28**	**84,48**	**82,24**

Tabell 4.3: Resultater for ANN-kjøringer
*Gjennomsnitt for kjøringene

Tre ANN-modeller ble implementert (én for hver sensormengde) og deretter kjørt tre ganger hver, i likhet med de andre nevrale nettverkene. Resultatet etter disse kjøringene, samt gjennomsnittene, er gitt i Tabell 4.3. Hver av de nevrale nettverkenes lagstruktur og hyperparametere er beskrevet i Vedlegg B.

Ved å observere resultatene fra Tabell 4.3 er det logisk å mistenke at modellene lider av overtrening - spesielt ved bruk av 1 sensor. Den store variansen fra testsett til testsett viser at modellen ikke generaliserer spesielt godt, og ser ut til å favorisere spesifikke mønster (spesielt testsettene 001, 008 og 009). Her er det verdt å merke seg at testsett 008 er fra en usett testperson. På kjøringen av to sensorer ser det imidlertid ut til at modellen klarer å generaliserer litt bedre, selv om den fortsatt ser ut til å favorisere noen over andre. Ved data fra tre sensorer ser man at modellen generaliserer bedre enn ved både én og to sensorer.

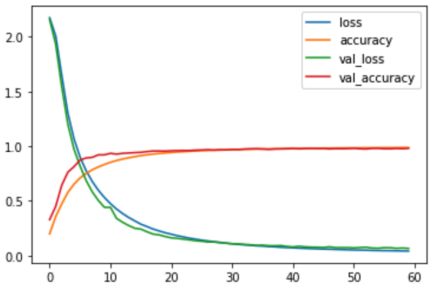
Det er verdt å merke seg resultater for testsett 008 ved de ulike kjøringene. Med én sensor hadde modellen høy nøyaktighet ved predikering av dette settet, men presterer dårligere ved to og tre sensorer. Dette blir videre diskutert i neste kapittel.

Ved hver kjøring ble testsett 004 brukt som valideringssett etter å ha vist seg å være det beste settet for justering av vektene underveis. Det er også en god visualisering å se hvordan valideringssettets treffsikkerhet og loss er i forhold til modellens. Vedlagt Figur C.1 viser at oppnådd nøyaktighet under trening (*accuracy*), og oppnådd nøyaktighet under validering (*val_accuracy*) konvergerer. Samme gjelder for treningens og valideringens tap (*loss* og *val_loss*). Videre viser Figur C.5 en forvirrings-matrise og linjediagrammer som viser modellens klassifiseringer ved siste kjøring av ANN-modellen med 3 sensorer.

4.3.4 CNN

I likhet med de andre nevrale nettverkene er hver modell trent tre ganger per sensormengde, på grunn av vektenes tilfeldighet ved initialisering. Resultatet for hvert av de ti testsettene, samt gjennomsnittet for hver modell, er gitt i Tabell 4.4.

Det første en kan observere i tabellen er at den gjennomsnittlige presisjonen med økende sensorantall øker med omtrent 10%. Dette er logisk å diskutere videre i diskusjonskapittelet, og vil bli tatt opp igjen der da det angår flere av de andre modellene også. Videre kan en observere at modellene er noe bedre enn ANN til å generalisere. Med tre sensorer kan en argumentere for at den generaliserer svært godt, ettersom kun to av settene er under 80%, det ene settet såvidt, og sett 008 vil bli sett nærmere på i videre diskusjon. Videre kan en se at flere sett er på over 90%, med det usette testsettet 010 på hele $99,64\%$. Som en kan se på Figur 4.4 skal det godt gjøres å finne hyperparametere som gir en bedre modell, med en så godt som perfekt presisjons- og tapskurve. Denne modellen er dog den som må trene absolutt lengst, og benytter seg av flere konvolusjonelle lag basert på forskning gjort av Ronao & Cho [31]. Avslutningsvis er det også verdt å merke seg at modellen for tre sensorer gjør det svært godt også på usette datasett som 007 og 010. Grafer for hver modelltrening kan finnes i Vedlegg C.



Figur 4.4: Diagram for læringsrate og presisjon for kjøring 2 av CNN

Antall sensorer	**1**				**2**				**3**			
Kjøring/ Testsett	1	2	3	$\overline{X}^*$	1	2	3	$\overline{X}^*$	1	2	3	$\overline{X}^*$
001	86,62	82,23	83,52	**84,12**	79,26	87,15	78,63	**81,68**	89,54	88,74	88,83	**89,04**
002	55,44	52,47	50,66	**52,86**	68,29	74,26	68,81	**70,45**	80,06	80,82	78,55	**79,81**
003	34,54	34,40	34,78	**34,57**	95,85	97,90	94,56	**96,10**	99,33	94,15	99,29	**97,59**
005	66,58	71,64	68,93	**69,05**	86,85	80,30	78,90	**82,02**	95,84	98,69	97,63	**97,39**
006	68,81	67,05	66,82	**67,56**	89,98	89,96	95,96	**91,97**	93,42	91,57	86,23	**90,41**
007	68,22	69,91	69,50	**69,21**	77,26	78,00	80,97	**78,74**	85,16	85,63	84,62	**85,14**
008	85,75	83,89	91,25	**86,96**	51,58	57,08	58,85	**55,84**	44,51	44,41	40,88	**43,27**
009	71,48	69,55	72,69	**71,24**	67,79	67,69	65,70	**67,06**	85,68	92,78	84,10	**87,52**
010	45,98	46,76	49,34	**47,36**	76,27	72,57	69,61	**72,82**	99,58	99,74	99,60	**99,64**
$\overline{X}^*$	**64,82**	**64,21**	**65,28**	**64,77**	**77,01**	**78,32**	**76,89**	**77,41**	**85,90**	**86,28**	**84,41**	**85,53**

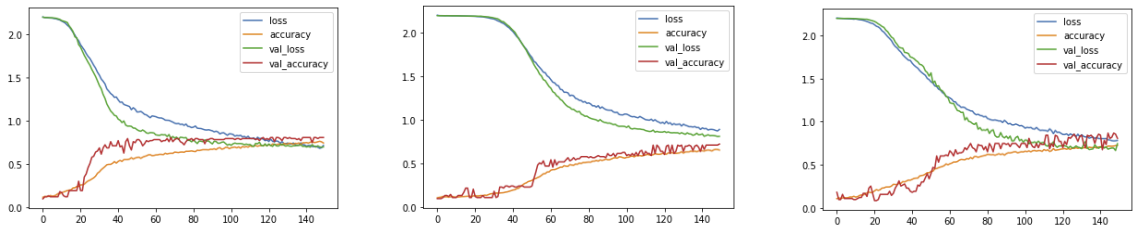
Tabell 4.4: Resultater for CNN-kjøringer
*Gjennomsnitt for kjøringene

4.3.5 LSTM

Tabell 4.5 viser resultatene fra kjøringen av LSTM-algoritmer, kjørt tre ganger med en, to og tre sensorer på samme måte som ANN og CNN tidligere. Ved å betrakte disse er det klart at LSTM-modellene med nåværende data ikke er ideell med tanke på generaliering. Det er heller ikke oppnådd bedre resultater ved justering av parametre. Videre demonstrerer resultatene i Tabell 4.5 og at det er noe større variasjon fra kjøring til kjøring ved LSTM enn ved de andre nettverkene. Grafene i Figur 4.5 illustrerer nettopp dette. Diskusjon rundt dette vil bli foretatt i kapittel 5 - Diskujon.

Antall sensorer	**1**				**2**				**3**			
Kjøring/ Testsett	1	2	3	$\overline{X}$*	1	2	3	$\overline{X}$*	1	2	3	$\overline{X}$*
001	76,83	54,88	73,17	**68,29**	67,07	65,85	75,61	**69,51**	81,71	84,15	81,71	**82,52**
002	79,49	69,23	79,49	**76,07**	69,23	70,51	65,38	**68,37**	91,03	92,31	88,46	**90,60**
003	45,00	44,00	55,00	**48,00**	80,00	76,00	84,00	**80,00**	82,00	82,00	91,00	**85,00**
004	80,72	72,29	80,72	**77,91**	90,36	89,16	89,16	**89,56**	89,16	89,16	90,36	**89,56**
005	84,44	64,44	74,44	**74,44**	96,67	90,00	97,78	**94,82**	96,67	96,67	98,56	**97,30**
006	67,12	78,08	64,38	**69,86**	95,89	97,26	97,26	**96,80**	83,56	89,04	86,30	**86,30**
007	64,29	64,29	71,43	**66,67**	78,57	78,57	85,71	**80,95**	85,27	78,57	71,43	**78,42**
008	80,00	70,00	70,00	**73,33**	50,00	60,00	60,00	**56,67**	20,00	20,00	20,00	**20,00**
009	92,50	80,00	91,25	**87,92**	82,59	58,75	67,50	**69,61**	87,50	91,25	91,25	**90,00**
010	25,30	34,94	43,37	**34,54**	44,58	44,58	55,42	**48,19**	61,45	66,27	66,27	**64,66**
$\overline{X}$*	**69,57**	**63,22**	**70,33**	**67,70**	**75,50**	**73,07**	**77,78**	**75,45**	**77,84**	78,94	**78,53**	**78,44**

Tabell 4.5: Resultater for LSTM-kjøringer
*Gjennomsnitt for kjøringene



(a) Kjøring 1 (b) Kjøring 2 (c) Kjøring 3

Figur 4.5: Taps- og nøyaktighetsgrafer for LSTM med 1 sensor

I likhet med de andre nevrale nettverkene presterer modellen bedre desto flere sensorer som blir brukt, i tillegg til at generaliseringen ser ut til å være bedre ved tre sensorer enn ved to og en. Det er også verdt å merke seg at testsett 008 oppfører seg likt ved LSTM som ved de andre nettverkene i og med at treffsikkerheten for dette settet minker desto flere sensorer man har. På samme tid blir modellen bedre til å predikere 010-settet, som er testdata fra en person modellen ikke har blitt trent på.

5 Diskusjon

Sentrale spørsmål som gjenstår å besvare er hvorvidt nøyaktigheten påvirkes mest av hvilken maskinlæringsalgoritme som tas i bruk, datasettet trent og testet på eller hvor mange sensorer en klassifiserer med. I tillegg er det verdt å diskutere hvilke feilkilder og potensielle svakheter en står ovenfor, i tillegg til det profesjonsetiske ansvaret man har når man utvikler noe som dette. Med utgangspunkt i tidligere kapitler og resultatene som er fått vil en i dette kapittelet prøve besvare disse spørsmålene med utgangspunkt i problemstillingene: *Hvilken nøyaktighet kan man oppnå ved klassifisering av sittestillinger basert på maskinlæringsalgoritmer og posisjonsdata fra IMU-sensorer? Og i hvor stor grad påvirkes klassifiseringsnøyaktigheten av antall sensorer?*

5.1 Datagrunnlag

Når man jobber med maskinlæring er det viktig å ha nok data til å la modellen som trenes bli så generell som mulig. Det er nesten alltid tilfellet at mer data vil forbedre en modell ettersom dette vil føre til mindre overtilpasning til tilfeldige mønstre i mindre datasett. Ettersom målet for prosjektet var å utforske hvor nøyaktig man kunne få en maskinlæringsmodell til å bli på egeninnspilt data, ble det tidlig bestemt at vi skulle ha et tilstrekkelig stort datasett å jobbe med.

Gjennom prosjektet har det blitt samlet inn data fra ti ulike personer, og ved noen tilfeller, flere ganger fra samme person. Utvalget av ulike personer ble gjort for å bygge et mer generelt og større datasett enn det som ville vært mulig om datainnsamlingen hadde blitt holdt innad i gruppa. Selv om en kan tenke at datasettet spilt inn i dette prosjektet er av en god størrelse (nærmere to millioner rader), hadde det absolutt vært ønskelig å ha mer data. Dette har med at det som ble sett på som mye data av gruppens medlemmer ikke var like mye i et maskinlæringsperspektiv. Prosjektet startet med en kortvarig datainnsamlingsperiode hvor et knippe testpersoner ble kontaktet og innsamling så ble gjennomført. Etter dette ble det et mer sporadisk preg over innsamlingen, og effektiviteten på rekvirering av nye data falt drastisk. Det en burde gjort her kunne vært å utvide tidsrommet satt av til datainnsamling og sendt ut melding til de det skulle gjelde mye tidligere i innsamlingsfasen. Dette ville ikke bare effektivisert rekvireringen av data, men også mest sannsynligvis ført med seg data fra flere testpersoner som igjen kunne bidratt til å generalisere datasettet enda mer.

En kan også stille spørsmål rundt den overordnede kvaliteten på innhentet data. Innhentingen ble gjort med manuell tidtaking og over bluetooth-forbindelser som hadde en tendens til å bryte opp ved høyere innspillingsfrekvenser. Tross gruppens beste innsatser for å holde en så god kvalitet på dataene som mulig, er det alltid rom for feil. Om en husker tilbake til resultatene for modellene kommer man tilbake til testsett 008 som skilte seg ut. Dette testsettet viste seg å være svært vanskelig i forhold til de andre. Nøyaktigheten til modellene var jevnlig rundt 40% på dette settet. Man kunne sett for seg at dette enkelt kunne forklares med at modellene som var trent ikke var blitt generelle nok, men

om en ser på de andre testsettene blir disse klassifisert med over 80% treffrate. Det er altså usannsynlig at dette er årsaken. En mer realistisk årsak kan være feil i datainnsamlingen. Om en inspiserer forvirringsmatrisen til modellene på testsett 008, kan man ane et mønster rundt sittestilling 3-7 hvor det virker som disse hyppig blandes sammen av modellene. Ved nærmere undersøkelse av opptakene gjort av testperson 008 kan man se at vedkommende sitter veldig rett, selv om personen egentlig skal lene seg til for eksempel høyre. Alle de andre testpersonene har lent seg godt til siden da de ble bedt om å innta disse stillingene. Dette er en tydelig glipp fra gruppa sin side, som skulle ha sagt ifra til vedkommende slik at kvaliteten på testsettet hadde blitt opprettholdt. Tross dette mindre gode datasettet, observeres som sagt allikevel en høy treffrate på resterende datasett, så det kan argumenteres for at de testsett som ikke inneholdt data fra testperson 008 holder en jevnt over god kvalitet.

Når en først diskuterer testsett 008 kan en tenke at personen som det ble testet på bare satt slik man ville sittet naturlig i gitt posisjon, og dette er mest sannsynlig tilfellet. Man ser derfor noe som kan være en annen svakhet ved valgt framgangsmåte. Har en kanskje for få sittestillinger å velge mellom? Det kan fort være at en person sitter litt imellom en stilling som kan klassifiseres som enten strak i ryggen eller fremoverlent, men sittestillingen vil bli klassifisert som en av de to forannevnte. Beslutningen om å begrense oss til ni positurer ble tatt i samsvar med råd mottatt fra veileder, men en kan allikevel sette spørsmålstegn ved denne avgjørelsen. For tilfellet med testsett 008 viser tydelig at det kanskje ikke holder med ni stillinger, og at en heller skulle innført ulik grad av lening på kroppen i de forskjellige retningene. Et problem med den løsningen, kunne vært at modellen ville fått en vanskelig oppgave i å skille to svært nære sittestillinger. Dette bringer oss til temaet diskutert i delkapittelets andre avsnitt, datamengde. Problemet kunne muligens vært løst med større datamengde fra testpersoner som sitter likt, eller tilnærmet likt som testperson 008. Den økte datamengden av slik posisjonsdata, kunne vært med å løse dette problemet, men også introdusert et nytt i form av at data for sittestillingene 3-7 hadde blitt for variert, og at modellene da hadde feilklassifisert andre stillinger som resultat av dette. Dog er dette kun hypoteser som vi ikke har fått testet ut med datamengden vi har å råde over og det er ikke noe som kan tas som fakta for øyeblikket. Det leder oss tilbake til hovedpoenget i kapittelet, gruppen hadde gjerne sett at vi hadde klart å anskaffe mer data.

5.2 Maskinlæringsmodeller

Som presentert i resultatkapittelet har det blitt utviklet fire modeller som hver har blitt trent på et datasett bestående av data fra syv testpersoner, for så å bli testet på ti testsett. I denne seksjonen vil disse resultatene bli tolket og diskutert på bakgrunn av relevant maskinlæringsteori, i tillegg til å bli satt opp imot hverandre.

5.2.1 Nevrale nettverk

Det standard nevrale nettverket ser ut til å prestere svært godt med både to og tre sensorer. Dette kan være en indikator på at selv et veldig enkelt nevralt nettverk vil kunne prestere

godt med få lag, noe som kanskje sier noe om dataen en har spilt inn. Den virker å være god til å finne hvilke features den skal legge vekt på, og som en kan se av Figur C.1 er grafene for både to og tre sensorer relativt like. Det som virker å skille tre sensorer fra to er at modellen klarer å generalisere enda bedre. Dette kan ha med at det med tre sensorer er enda flere trekk som skiller sittestillingene fra hverandre. Dersom en hadde sett bort fra testsett 008 som beskrevet i forrige delkapittel, hadde en nærmet seg en nøyaktighet på 88%, noe som kan ansees som svært godt. Dette er altså en god pekepinn på hvor bra nevrale nettverk kan klare seg uten å ha for mange komplekse lag.

Om en ser videre på CNN kan en se at denne presterer svært likt som ANN. For én og to sensorer presterer den litt dårligere, som kan skyldes at det konvolusjonelle nettverket sliter med å finne gode mønstre. Dette nettverket er av mye høyere kompleksitet, som gjør den mer generell i jakten på de riktige trekkene, men kan på den andre siden gjøre den mer sårbar for å ikke kunne skille mellom to stillinger, ettersom den sammenlikner flere mønstre som kan være motstridende. Her kan det argumenteres for at det finnes bedre modeller for både én og to sensorer, som ikke er funnet enda. For lagstrukturen som er blitt tatt i bruk er det om en ser på Figur C.2 ganske gode parametere i bruk, men en kunne kanskje ha lagt til noen flere lag og fått litt bedre resultater her. Det er litt av fallgruven med maskinlæring, ettersom en alltids kan finne bedre modeller enn de som er presentert. For tre sensorer er dette dog den beste modellen presentert i prosjektet. Om en ser bort ifra testsett 008 ville en hatt en nøyaktighet på nærmere 92%. Kurven presentert i Figur C.2 viser så godt som ideelle kurver for både modell- og valideringsnøyaktighet og tap. Dette støtter opp under arbeidet til Ronao & Cho [31] som beskriver et multilags CNN som en god modell for tidsseriedata til aktivitetsgjenkjenning.

Når det kommer til LSTM kan denne også vise til gode resultater. Her er det dog slik at testsettene fra usette personer presterer generelt dårligere enn de fra sette testpersoner. Dette tyder på dårlig generalisering eller overtrening. Dette kan henge sammen med at et LSTM må ha tredimensjonal input-data på formen (# samples, # timestamps, # input-felter). I vår LSTM modell består hver sample av 100 timestamps (som med 50 Hz utgjør et bilde på to sekunder). For hver sample får man en klassifisering, det vil si at man trenger 100 sensormålinger for hver klassifisering. Man får altså 100 ganger så få samples å predikere ved LSTM enn ved ANN og CNN, noe som kan forklare dens tendens til dårlig generalisering - spesielt ved en og to sensorer.

Dette kan også være noe av grunnen til at oppnådd nøyaktighet mellom kjøringene varierer så mye som de gjør. Testsettene som testes er i utgangspunktet relativt små; ca i underkant av 10 000 rader etter sammenslåing på timestamp. Ved å dele disse opp i samples bestående av 100 rader, ender man opp med testsett som bare består av i underkant av 100 samples. Ved færre samples vil eventuelle feilklassifiseringer være mer utslagsgivende - noe som vil føre til at det skal færre feilklassifiseringer til før gjennomsnittsprosenten påvirkes av dette.

Til tross for dette ser det ut til at LSTM presterer relativt godt, og presterer i likhet med de andre nevrale nettverkene bedre med tre sensorer enn med én eller to. En kan også her observere gode resultater selv på testsett fra personer som modellene ikke har blitt

introdusert for tidligere. En annen fordel en ser er at til tross for noe lavere nøyaktighet, klassifiserer den raskere enn CNN på grunn av lavere kompleksitet. I likhet med de andre nevrale nettverkene ser det ut til at modellen klarer generalisere best ved tre sensorer.

5.2.2 RFC

RFC viste seg å passe svært godt til vårt formål. Som vi ser av resultatene på testsettene er den ganske nøyaktig, med riktig klassifisering ca. 80% av gangene. Dette er ikke den mest nøyaktige modellen, men den er svært rask i forhold til de andre modellene vi har testet, noe som gjør at en kan gi brukeren live data som ikke henger etter. Dette er grunnet enkelheten av å regne ut klassifikasjonene når treet først er trent, det er bare å gå gjennom en serie med boolske operasjoner. Den lille unøyaktigheten på 20% gjør en så godt som opp for ved at den mest gjettede posituren av flere individuelle klassifikasjoner blir valgt. Antallet individuelle klassifikasjoner vil avhenge av frekvensen sensorene sender data med, som er justerbart, og som har blitt satt til 5Hz. Med denne frekvensen, brukes også fem ulike klassifikasjoner og den mest gjettede blant dem blir funnet. Dette minker påvirkningskraften til det alleredede lite sannsynlige tilfellet hvor modellen tar feil.

I resultatkapittelet kan man se en graf over RFC sin OOB-feil. Denne grafen ser svært imponerende ut, med feilprosenter på under 0,001%. I praksis derimot, ser man et markant skille mellom faktiske feilprosenter ved bruk av testsett og OOB-feilprosenten. Ifølge resultatene, ligger vår beste RFC-modell på 80% nøyaktighet, noe som gir en mye høyere feilprosent enn den estimerte OOB-feilen. Vanligvis skal OOB-feilestimering være en god pekepinn på en RFC-modell sin presisjon, men i dette tilfellet var det ikke slik. En forklaring på denne uoverensstemmelsen kan være hvordan dataen vår er strukturert. Når en person sitter i samme stilling over lengre tid, vil mange rader med data være tilnærmet like. Dette kan føre til at selv om et datapunkt er usett for modellen, så har den allerede trent på flere tilnærmet like datapunkter fra før, og kan med stor sikkerhet klassifisere også dette 'ukjente' datapunktet. Dermed er ikke OOB-feilestimering like nøyaktig lenger.

Videre feilreduksjon ble gjort ved å beholde alle features i modellen mens den trente. Det er ofte man finner et mål på viktigheten av hver feature og plukker ut de viktigste. Dette øker hastigheten til utregninger med modellen, men kommer på bekostning av nøyaktighet. Siden modellen allerede oppfylte våre krav til hastighet, så vi ingen grunn til å minke mengden features, og vi bruker derfor alle sammen. Man kan se for seg en situasjon hvor det hadde vært ønskelig med en raskere utregning. For eksempel kjøring av programmet og modellen på en mindre kraftig datamaskin, eller samtidig med mange andre programmer. For øyeblikket er dette overlatt som et ansvar til brukeren, men om det viser seg at en senere forbedring på dette området skulle være nødvendig er ikke en fiks for dette spesielt krevende, og kunne blitt utarbeidet på kort tid.

En kan argumentere for at et bedre valg ville vært å bruke en modell som CNN, med lavere frekvens grunnet høyere kompleksitet, men denne er såpass mye tregere at det ville blitt en lite responsiv opplevelse for brukerne og vi gikk dermed bort fra dette. Vi gikk derfor for den litt mindre nøyaktige modellen RFC, som til gjengjeld er mye raskere.

5.2.3 Ulik prestasjon

Noe som kan være spennende å ta opp er hva som er årsaken til at de ulike modellene presterer ulikt. En kan tenke seg at de nevrale nettverkene kunne prestert bedre ved å ha tilgang på mer data. For å støtte oppunder denne påstanden kan en se på andre kjente maskinlæringsprosjekter som har vist at store datasett er veien å gå. Facebooks *DeepFace*, som beskrevet i Taigman *et al.* [36], hadde et treningssett på fire millioner bilder med mer enn fire tusen identiteter og oppnådde en nøyaktighet på 97,35% på sitt nevrale nettverk. Direktøren for AI hos Tesla, A. Karpathy, beskrev under sitt doktorgradsstudie ved Stanford [16] hvordan nevrale nettverk kan identifisere flere gjenstander i et bilde ved bruk av 94,000 treningsbilder og i overkant av fire millioner bildetekster på ulike språk. Karpathy har også hevdet at deres motto er å holde datamengden stor, algoritmene enkle og klassene svake (eng: weak) [25]. Allikevel virker det som at tilstrekkelig data er innhentet i prosjektet for at de nevrale nettverkene skal oppnå tilfredsstillende nøyaktighet. Det er også her gjort nyere forskning på en variant av *DeepFace* på små datamengder som hevder å få relativt like resultater om den er trent på 10 000 bilder som 500 000 [14], noe som kan tyde på at det i noen tilfeller ikke nødvendigvis trenger å være et behov for enorme datamengder. Det skal dog sies at en trygt kan anta at en større datamengde ville hatt en positiv innvirkning på modellenes nøyaktighet, men her må det tas en avveiing mellom ytelse og presisjon.

Det er ikke nødvendigvis bare datamengden som kan påvirke nøyaktigheten til modellene. I teorien burde LSTM modellen som anses som svært god for tidsseriedata prestere bedre enn den gjør for øyeblikket. En grunn til at den ikke presterer så godt som den burde kan være avviket fra treningsdata og testdata. Som kjent står LSTM for *Long Short-Term Memory* og hinter til modellens hukommelsesliknende mekanisme. Modellen kan muligens ha lært seg at når en person først sitter i en stilling, så sitter de der over lengre tid, og dermed vil den lene mot denne beslutningen i møte med ny data. Hvis personen som testes, nå ikke sitter i nærheten av så lenge som det ble gjort i treningssettene, så kan dette by på problemer.

5.3 Oppnåelig versus akseptabel nøyaktighet

Det er flere faktorer en må ta høyde for når en ser på hvor mange sensorer det er ønskelig å ta i bruk i klassifiseringen. En må ha en avveining mellom blant annet brukervennlighet, hastighet og nøyaktighet. Det som har vært hovedfokus for denne oppgaven, som gjenspeiles i problemstillingene valgt, er hvor stor effekt antallet sensorer har på nøyaktigheten. I dette underkapittelet vil dette bli sett nærmere på og diskutert.

Før en tar for seg forskjellen i nøyaktighet basert på sensorantall, er det greit å ha diskutert hva som defineres som *god nok* nøyaktighet. Dette er et spørsmål som er vanskelig å ta stilling til, for hvem er det som skal bestemme hva som er godt nok? Et godt sted å starte er å se på tidligere relevant arbeid. Gjennomsnittet for de liknende produktene som ble sett på under skrivingen av visjonsdokumentet la seg på omtrent 90% nøyaktighet, men da med færre posisjoner eller andre former for sensorer. Den høyeste treffraten vi har

observert i andre relevante forskningsartikler er på 95%, beskrevet i Bhat *et al.* [5], selv om denne har et litt annerledes formål ved å istedenfor klassifisere aktivitetsformer, og er trent på mer data (22 personer). Det er ikke godt å si hva som bør være laveste terskel for hva som er bra nok, ettersom en helst vil komme så nærme 100% som mulig, men det er en relativt liten sannsynlighet for å få laget en modell som klarer det på all data den blir presentert for.

Samtidig er det slik at en prøver å se på variasjon her. Det er ikke en bestemt fasit på god eller dårlig, så dersom det viser seg at det ikke er mulig oppnå for eksempel 90% stabilt og dette egentlig foretrekkes som terskel, så kan en til forskning være bevvist på at det er visse feilkilder og at variasjonen kan avvike litt i forhold til det grafene eventuelt skulle vise. Det kan også nevnes igjen at det har blitt gjort tiltak for å begrense muligheten for feilklassifiseringer ytterligere ved å ta typetallet av flere klassifiseringer, som er med på å øke nøyaktigheten i praksis. Dette diskuteres nærmere i neste delkapittel. Det er altså vanskelig å bestemme hva som skal være krav når det kommer til nøyaktighet, men en kan som en pekepinn anse nøyaktigheter på minst 80% og helst opp mot 90-95% som reelle målsetninger.

Som resultatene viser, påvirkes nøyaktigheten av antall sensorer modellene tar i bruk. En grunn til dette kan være at flere sensorer vil dekke en større andel av kroppen, og dermed gi data som danner et bedre helhetsbilde over kroppens posisjon. Om en bruker alle tre sensorene gruppa har benyttet seg av, vil de plasseres på tre ulike posisjoner: nedre nakke, korsryggen og høyre skulderblad. Med disse plasseringene vil en samle inn data for kroppens horisontale og vertikale rotasjon, samt orientering av brukerens hode. Som fysioterapeut Børke vektla på møtet avholdt tidlig i prosjektetfasen (Vedlegg D), gir målinger på bekkenet en god indikator på kroppens orientering, og det er tiltenkt at en får med deler av bekkenbevegelsen ved å plassere en sensor på korsryggen. Utifra dette kan en også se for seg at flere sensorer ville hatt en positiv påvirkning på modellenes nøyaktighet. Man burde allikevel øke dette antallet forsiktig, ettersom et for høyt antall kan gjøre dataene vanskelige å jobbe med, og dermed nødvendigvis vil føre til en økt kompleksitet for modellene. Om presisjon er målet, vil ikke dette være et stort problem, men om en vil tolke hvordan modellen kom frem til resultatene, kan dette være en ulempe.

Avslutningsvis er det også greit å bemerke seg at også brukervennlighet og hastighet er noe en må ta hensyn til. Det mest brukervennlige ville vært å putte én sensor på personen som skal observeres. Dette ville vært mye mer lettvint og man trenger ikke flytte unna klær og annet som kan være i veien for å feste de andre sensorene på kroppen. Allikevel ansees det å sette på flere sensorer som en såpass liten hindring at en ikke skal måtte gå vekk fra beste sensorantall i forhold til nøyaktighet på grunn av dette. En har også hastighetsaspektet, som blir relevant når det kommer til utviklingen av selve applikasjonen. Antallet sensorer påvirker hastigheten ved å styre hvor mye data som mates til modellen, som igjen påvirker utregningshastigheten av klassifiseringer. Det har ikke blitt observert noen særlige endringer i hastighet med færre sensorer, men tenker allikevel at det var et relevant poeng å ta i betraktning ettersom det kan ha innvirkning på tregere maskiner. Det har altså heller ikke blitt begrenset antall sensorer på grunn av hastighet. Vi ser det fortsatt som hensiktsmessig å gå for antallet som vil gi høyest mulig nøyaktighet.

5.4 Klassifisering i sanntid

Gjennom hele oppgaven har det blitt strevet etter å lage den beste mulige modellen til de tilgjengelige datasettene. Det har blitt brukt flere ulike maskinlæringsalgoritmer som alle har vært ganske gode, men som nevnt i forrige delkapittel er CNN den beste modellen på data fra tre sensorer. Allikevel blir det valgt å bruke den mindre nøyaktige RFC-modellen i applikasjonen utviklet i prosjektet. Dette er grunnet en avveining en måtte gjøre i forhold til tid og nøyaktighet. Det konvolusjonelle nevrale nettverket er som sagt den mest nøyaktige, men den lider av å være svært treg i forhold til RFC. Grunnen til dette er kompleksiteten til hver av modellene. I en RFC modell traverserer en gjennom binærtrær for så å finne den vanligste av klassifiseringene. Nevrale nettverk derimot, er mye mer komplekse strukturer, som gjør det tyngre for maskinen å jobbe seg fram til en klassifisering. Forskjellene som ble observert i tidsbruk var opptil fem sekunder forskjell mellom RFC-modellen og CNN-modellen. I en sammenheng hvor brukeren skal få data i sanntid vil den tregere modellen føre til forsinkelser og gjøre at hele systemet henger etter. Brukeren vil da få opplyst helt feil klassifisering i forhold til deres posisjon i et tidsrom etter de bytter stilling.

På bakgrunn av dette ble det dermed valgt å gå for RFC-modellen. Denne har en treffrate på ~81% utifra våre forsøk. Det lille tapet i treffsikkerhet gjøres opp for ved høyere hastigheter enn nevrale nettverk. Samtidig gjør en opp for mer av unøyaktigheten til modellen ved å bruke gjennomsnittet av et gitt antall klassifiseringer som den endelige klassifiseringen sin. For øyeblikket er antallet klassifiseringer man plukker fra satt til å være lineært avhengig av innspillingsfrekvensen slik at modellen skal klare å holde følge med dataene som blir innsendt. Det kan virke som det blir for mye ved at en legger på enda en runde bagging etter at RFC allerede gjør det, men det er erfart av gruppen at dette gir mer stabile og nøyaktige resultater.

Den endelige avgjørelsen som falt på RFC var altså en avgjørelse gjort på våre observasjoner om at de mest nøyaktige modellene ikke klarte å holde følge med den ønskede hastigheten for applikasjonen som ble utviklet. Alle modellene laget i prosjektet ligger allikevel som vedlegg slik at man kan undersøke disse ved interesse.

5.5 Prosjektarbeid, profesjonsetikk og systemperspektiv

Prosjektet har bydd på varierte arbeidsoppgaver innen flere områder man kan jobbe med som dataingeniør. Det å skulle jobbe med både utvikling og forskning har vært en spennende kombinasjon, der en kunne sette seg inn i teori og prosessen med å lage maskinlæringsmodeller, samtidig som en måtte forholde seg til hvordan en kunne utvikle et brukergrensesnitt som gjør det enkelt å ta de i bruk. Teamet består av individer med bred kompetanse og ulike preferanser for hva en liker best å jobbe med, noe denne oppgaven ga spillerom for og som derav har utfylt hverandre godt under prosjektet.

Det er flere årsaker en kan trekke fram som har bidratt til at gruppen har fungert som den har gjort. Teamet har i en tid preget med usikkerhet om en kan oppholde seg på skolen

eller ikke, vært heldige som har hatt tilgang på et rom på skolen til enhver tid. Det har også vært viktig å ha en kjernetid med felles oppmøtetidspunkt og å sitte samlet å jobbe under utvikling og forskning, noe som ble tidlig avtalt og etablert. Dette sammen med en veileder som kun var en Teams-melding unna har ført til at en ikke har støtt på problemer som gruppen selv ikke kunne løse. Teamet står igjen etter endt prosjektperiode med et produkt en kan være stolt av, og som er en god oppsummering på hva en sitter igjen med av kunnskap etter endt treårig studieløp.

Når det kommer til produktet utviklet er det også viktig å kunne se oppgaven i et større systemperspektiv. Maskinlæring er et fagfelt med delte meninger, nettopp fordi tanken om en ny utfordrer til vår intelligens for mange er skremmende. Verden er stadig i endring, og teknologien har utviklet seg med stormskritt det siste århundret. Som Y. Harari skriver i sin *Homo Deus* [11] om fremtidens utfordringer, presenteres muligheten for at kunstig intelligens overgår menneskelig intelligens. *"People are usually afraid of change because they fear the unknown. But the single greatest constant of history is that everything changes"*. Som sitatet så godt beskriver er mennesket redd for endring og det en ikke har kjennskap til, men det har seg slik at endring er det som driver samfunnet videre. Mennesket vil fortsette å utforske og utvikle ny og bedre teknologi, nettopp fordi vi kan.

Samtidig er det også vårt ansvar som utviklere å ta gode etiske avgjørelser. Ettersom mennesket har kontroll over hva som utvikles og hvordan det gjøres, er det som beskrevet av Harris [12] også vårt ansvar å kontrollere hva som blir laget. Harris presenterer hvordan mennesket står ovenfor to muligheter. Den ene muligheten er å stanse utviklingen av teknologien, nemlig fordi det kan føre til at teknologien utrydder menneskerasen. Det andre alternativet er å fortsette å utvikle teknologien, men under kontrollerte grenser. Som utviklere av et prosjekt som kan tas i bruk i videre forskning er det viktig å være klare over hvilket profesjonsetisk ansvar en har. For eksempel er spørsmålet om hva som beregnes som god nok nøyaktighet fra kapittel 5.3 også sentralt her. I et scenario der løsningen vår blir videreutviklet eller tatt i bruk i større forskning og skala, kan en være med på å bestemme folks oppfattelse om egen rygghelse. Dersom en gjennom denne programvaren skulle fått et feilaktig inntrykk av at en har dårlig rygghelse, kan det føre til at mennesker opplever stress, prøver utbedre dette selv eller potensielt oppsøker unødvendig helsehjelp, noe som blant annet kan påvirke ens økonomi. Videre er også personvern sentralt i samråd med computeretikkens prinsipp 1.6 [1]. Her har det vært viktig å få skrevet under et samtykkeskjema levert av Norsk senter for forskningsdata (NSD), for hver person det har blitt spilt inn data fra. Annoteringsvideoer er også trygt slettet etter bruk, og hver person har blitt anonymisert ved hjelp av tallkoder. Det er vårt ansvar som utviklere å kontrollere det som blir utviklet av ny teknologi, samtidig som at det er viktig å være bevisst på elementer som personvern og hvilke konsekvenser det som utvikles kan ha for andre.

Avslutningsvis er det verdt å prøve å sette systemet i en økonomisk sammenheng. For det første har en som nevnt i forrige avsnitt muligheten til å påvirke menneskers personlige økonomi. Dersom dette programmet blir kommersielt tatt i bruk, og at det blir brukt av enkeltpersoner - kan det få faktiske konsekvenser for disse. Helserelaterte problemer kan bli dyre, og en ønsker ikke at mennesker skal prøve justere egen sittestilling selv på

feilaktig grunnlag og gjøre det verre. Dersom det forekommer store avvik i applikasjonen kan dette altså være med på å skape økonomiske byrder for de som rammes. En annen side er dersom applikasjonen skulle blitt solgt av kommersielle årsaker - og noen skulle tjent penger på den. Kan man tilgjengeliggjøre det for noen, men ikke andre? Selge produktet med lovnad om god rygghelse? Dette er spørsmål en står ovenfor når produktet skal tas med videre av andre enn gruppen som startet med utviklingen. Det er et ansvar man har, og som er greit å være bevisst over.

5.6 Feilkilder og potensielle svakheter

5.6.1 Datagrunnlag/innspilling

Til tross for all vår innsats for å redusere feilkilder under datainnsamling vil det alltid komme noen feil igjennom. Feilkilder som kan identifiseres er: feil plassering av sensor, unøyaktig tidtaking, sensorer som kobler ut og at testpersonen sitter feil. Disse feilkildene kan ha påvirket både trenings- og testdata og ført til at modellene våre blir mindre nøyaktige. Det vil nesten alltid oppstå menneskelig feil, så en kan aldri være helt sikre på at datasettene våre er feilfrie.

En mulig løsning på problemet kunne vært å kjøpe data fra profesjonelle aktører. Dette kunne minket feilmarginen noe, men heller ikke her kunne man ha vært helt sikre på fravær av menneskelig feil. Her må det også være tilgjengelig ønsket data, som kan være vanskelig å oppdrive og potensielt koste store summer. Helsedata er også sjelden ønskelig å dele eller selge av blant annet hensyn til personvern. Dette gjør sannsynligheten for at en finner et allerede eksisterende og passende datasett enda mindre.

Samtidig er det som nevnt tidligere en mangel på data for å kunne si at modellene som er laget virkelig er generelle. Under inspeksjon av tidligere forskning på feltet, har det vært opp mot 40 ulike testpersoner som har blitt rekruttert. Dessverre har ikke gruppen hatt en like stor mulighet til å hente inn kandidater grunnet de vanskelige situasjonene corona har ført til, samt at gruppen er noe mindre enn i studier gjort tidligere. Om en skulle fortsatt med videreutvikling av prosjektet ville dermed et godt sted å starte være å fått samlet inn mer data.

5.6.2 Maskinlæringsmodeller

Når man implementerer maskinlæringsmodeller er det flere parametre som kan justeres for å få en modell som er best mulig tilpasset dataen som skal brukes. Eksempler på slike parametre er nevrale nettverks dybde (hvor mange lag i nettverket) og bredde (hvor mange noder i hvert lag), batch-size, læringsrate og antall epoker for å nevne noen. Dessuten er det ikke slik at det finnes en fast fremgangsmåte når man jobber med slik type modellimplementering; alle datasett er ulike, og dersom man må lage sitt eget er man i prinsippet den første som prøver å implementere en modell som skal prøve å etterligne den funksjonen som er best tilpasset akkurat den dataen.

Med andre ord er det en stor sannsynlighet for at det finnes parametere der ute som er bedre tilpasset den dataen vi har produsert enn de vi selv har klart å finne. Dette er en risiko som mer eller mindre alltid vil være til stede, men dette er bare en del av maskinlæring som prosess, en må prøve og feile.

5.6.3 Trening og validering

Det er allikevel mulig å komme frem til meget gode hyperparametere, og disse vil finnes under valideringsfasen av maskinlæringen. Her benyttet gruppen seg av valideringssett som valideringsmetode. Dette kan være en potensiell svakhet, siden bruken av enkle valideringssett kan gjøre modellen sårbar for overtilpasning til valideringssettet selv. Til gjengjeld ble alle datasettene testet for å se hvilket som ville gi høyest presisjon som valideringssett. Dette viste seg å være testsett 004. Til tross for valideringssett sin tendens til overtilpasning, fungerte dette godt og ga en jevnt over god treffrate.

En vanlig metode for validering er kryssvalidering og en kan spørre seg hvorfor ikke dette er brukt til vår valideringsprosess. I vårt tilfelle var det et spørsmål om effektivitet og faktisk gevinst. Kryssvalidering vil i praksis fungere som flere gjennomganger med ulike testsett i motsetning til gjennomgangene med samme valideringssett som vi benyttet oss av. Dette er ment å redusere overtilpasning, noe som ikke viste seg å være tilfellet for vår data når forsøkt med RFC. Siden vi fikk like gode resultater med ett enkelt valideringssett, var det hensiktsmessig å beholde denne mindre komplekse metoden for å spare tid.

6 Konklusjon og videre arbeid

6.1 Konklusjon

Resultatene fra kapittel 4 og slutningene trekt i etterfølgende kapittel 5 gir et godt grunnlag for å kunne svare på begge de aktuelle problemstillingene; *Hvilken nøyaktighet kan man oppnå ved klassifisering av sittestillingerbasert på maskinlæringsalgoritmer og posisjonsdata fra IMU-sensorer? Og i hvor stor grad påvirkes klassifiseringsnøyaktigheten av antall sensorer?*

Det er tydelig at maskinlæringsalgoritmene som er implementert i denne oppgaven kan benyttes for menneskelig aktivitetsgjenkjenning. Med gjennomsnittlige treffrater på ca. 80% inkludert upresis data, og opp mot 90% uten denne, kan man konkludere med at svaret på det første spørsmålet er *minst 80%*. Med mer data er det sannsynlig at denne prosenten vil stige. Hvorvidt dette kan klassifiseres som god nok nøyaktighet eller ikke er vanskelig å svare på. Når det er sagt, kan en ved å ta i bruk typetallet på flere klassifiseringer, slik som gjort i dette prosjektet, få en enda høyere nøyaktighet i praksis. En må være klar over hvilke konsekvenser det kan ha om en bestemmer en nedre terskel for hva som er godt nok som burde vært høyere, dersom systemet skulle blitt tatt i bruk. En bør alltid streve etter å bedre nøyaktigheten til modellene, i tillegg til å se systemet i et større systemperspektiv.

Dette er en del av det profesjonsetiske ansvaret en har som utviklere.

Det er også mulig å trekke en konklusjon hva angår den andre delen av problemstillingen. Resultatene viser at klassifiseringsnøyaktigheten påvirkes i relativt stor grad av antall sensorer. Samtlige modeller generaliserer bedre, og oppnår høyere gjennomsnittlig klassifiseringsnøyaktighet enn ved færre sensorer. Denne økningen av sensorer fører til at en får dekket et større område av brukerens kropp, og danner dermed et bedre helhetsbilde over personens sittestilling. Det har dermed blitt vist at flere sensorer gir bedre nøyaktighet på klassifiseringen av sittestillinger.

6.2 Videre arbeid

Oppgaven viser lovende potensiale, og er i aller høyeste grad et prosjekt som kan arbeides videre med. Ved videre arbeid hadde det vært interessant å se hvordan klassifiseringsnøyaktigheten hadde blitt påvirket av større mengder treningsdata fra flere personer. Videre kan nok også flere typer maskinlæringsmodeller testes, og en kan muligens også gå i dybden på flere kompliserte, sammensatte nettverk som f.eks. CNN-LSTM nettverk. Andre alternativer som kunne vært spennende å utforske er Support Vector Machine (SVM) eller å implementere RFC ved bruk av XGBoost [39] biblioteket.

Som nevnt tidligere ble det forsøkt å ta i bruk kryssvalidering på RFC. Dette viste seg å ikke ha noen merkbar økning i nøyaktighet, men for videre arbeid kunne det ha vært spennende å utforske effekten av dette på de nevrale nettverkene. Antall sittestillinger kan også utforskes. Hvor mange forskjellige sittestillinger modellene hadde vært i stand til å skille mellom hadde vært interessant å finne ut av.

En annen kombinasjon av sensordata kan også være spennende å utforske. Som nevnt i kapittel 3.2.2.2 benyttet vi oss av angulær og lineær akselerometerdata, gyroskopdata og quaternions som datagrunnlag. De ulike datafeltene kan kombineres og settes sammen på mange ulike måter, noe en gjerne skulle utforsket mer dersom en hadde hatt mer tid. Det kan for eksempel hende en kombinasjon av blant annet eulervinkler og andre felter uten quaternions kan gi gode resultater, til tross for våre erfaringer om at quaternions gir bedre nøyaktighet.

Med andre ord kan dette prosjektet fungere som et godt fundament for videre arbeid innen dette feltet, og viser, etter vår mening, lovende potensiale.

Referanser

1. *ACM Code of Ethics and Professional Conduct* https://www.acm.org/code-of-ethics (18. mai 2021).
2. *Anvil videoannoteringsverktøy* https://www.anvil-software.org/ (10. mai 2021).
3. *Back Pain Facts and Statistics* https://www.acatoday.org/Patients/What-is-Chiropractic/Back-Pain-Facts-and-Statistics (27. apr. 2021).
4. Bannach, D. *et al.* Self-Adaptation of Activity Recognition Systems to New Sensors. https://arxiv.org/pdf/1701.08528.pdf (9. mai 2021) (2017).
5. Bhat, G., Tran, N., Shill, H. & Ogras, U. Y. w-HAR: An Activity Recognition Dataset and Framework Using Low-Power Wearable Devices. *Sensors* **20.** ISSN: 1424-8220. https://www.mdpi.com/1424-8220/20/18/5356 (2020).
6. Breiman, L. Random Forests. *Statistics Department, University of California,* 4. https://www.stat.berkeley.edu/users/breiman/randomforest2001.pdf (2001).
7. Fridman, L. *MIT 6.S094: Recurrent Neural Networks for Steering Through Time* https://www.youtube.com/watch?v=nFTQ7kHQWtc (11. mai 2021).
8. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* http://www.deeplearningbook.org (MIT Press, 2016).
9. Gupta, R. *et al.* A Wearable Multisensor Posture Detection System. *Amity University Uttar Pradesh, India.* https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9121082 (3. feb. 2021) (2020).
10. Hammerla, N. Y., Halloran, S. & Plötz, T. Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables. https://arxiv.org/pdf/1604.08880.pdf (10. mai 2021) (2016).
11. Harari, Y. N. *Homo Deus: A Brief History of Tomorrow* ISBN: 978-191-070-187-4 (Harvill Secker, 2016).
12. Harris, S. *Can we build AI without losing control over it?* https://www.ted.com/talks/sam_harris_can_we_build_ai_without_losing_control_over_it (18. mai 2021).
13. Hasan, R., Latif, A. B. & Uz, S. A cost-effective Smart Posture Trainer - PosturePal. *Department of Electrical and Computer Engineering, North South University, Bangladesh.* https://www.researchgate.net/publication/346241363_A_cost-effective_Smart_Posture_Trainer-PosturePal (3. feb. 2021) (2019).
14. Hu, G., Peng, X., Yang, Y., Hospedales, T. M. & Verbeek, J. Frankenstein: Learning Deep Face Representations using Small Data. https://arxiv.org/pdf/1603.06470.pdf (19. mai 2021) (2017).
15. James, G., Witten, D., Hastie, T. & Tibshirani, R. *An Introduction to Statistical Learning* 316–321 (Springer Science+Business Media, 2015).
16. Karpathy, A., Johnson, J. & Fei-Fei, L. DenseCap: Fully Convolutional Localization Networks for Dense Captioning. *Department of Computer Science, Stanford University.* https://cs.stanford.edu/people/karpathy/densecap.pdf (19. mai 2021) (2015).

17. LaValle, S. *Guest lecture about quaternions at NPTEL* https://www.youtube.com/watch?v=5YJwszR246I (30. apr. 2021).

18. Loukas, S. *What is Machine Learning: Supervised, Unsupervised, Semi-Supervised and Reinforcement learning methods* https://towardsdatascience.com/what-is-machine-learning-a-short-note-on-supervised-unsupervised-semi-supervised-and-aed1573ae9bb (5. mai 2021).

19. *LpmsB2-UserGuide-V1.0* Life Performance Research (2016). http://www.alubi.cn/wp-content/uploads/2016/08/LpmsB2-UserGuide-V1.0.pdf.

20. Minsky, M. & Papert, S. *Perceptrons. An Introduction to Computational Geometry.* (Cambridge, Massachusetts, 1969).

21. Mnih, V. *et al.* Playing Atari with Deep reinforcement Learning. *Google DeepMind.* https://arxiv.org/pdf/1312.5602v1.pdf (6. mai 2021) (2013).

22. Murad, A. & Pyun, J. Deep Recurrent Neural Networks for Human Activity Recognition. https://www.researchgate.net/publication/320886290_Deep_Recurrent_Neural_Networks_for_Human_Activity_Recognition (5. mai 2021) (2017).

23. *Nettsiden til LP-RESEARCH Inc.* https://lp-research.com/about-us/ (8. mai 2021).

24. *Neural Networks* https://www.ibm.com/cloud/learn/neural-networks (19. apr. 2021).

25. *Next Generation Machine Learning - Training Deep Learning Models in a Browser: Andrej Karpathy Interview* https://www.datascienceweekly.org/data-scientist-interviews/training-deep-learning-models-browser-andrej-karpathy-interview (19. mai 2021).

26. Panwar, M. *et al.* CNN based approach for activity recognition using a wrist-worn accelerometer, 2438–2441 (2017).

27. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830. https://scikit-learn.org/stable/modules/tree.html (7. mai 2021) (2011).

28. Pienar, S. W. & Malekian, R. Human Activity Recognition Using LSTM-RNN Deep Neural Network Architecture. https://arxiv.org/ftp/arxiv/papers/1905/1905.00599.pdf?fbclid=IwAR35xacuJGniOHaoDUZ1mgArD0zPuT8p6-hAUI3lOajdO-ArFVpGz2mzyP0 (5. mai 2021) (2019).

29. Ramamurthy, S. R. & Roy, N. Recent trends in machine learning for human activity recognition—A survey. *WIREs Data Mining and Knowledge Discovery.* https://onlinelibrary.wiley.com/doi/full/10.1002/widm.1254 (8. mai 2021) (2018).

30. Rashid, T. *Make Your Own Neural Network* 73–76. ISBN: 978-1530826605 (CreateSpace Independent Publishing Platform, 2016).

31. Ronao, C. A. & Cho, S.-B. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications* **59,** 235–244. ISSN: 0957-4174. https://www.sciencedirect.com/science/article/pii/S0957417416302056 (2016).

32. Silver, D. *et al.* A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Journal of Computational Physics.* https://deepmind.com/research/publications/general-reinforcement-learning-algorithm-masters-chess-shogi-and-go-through-self-play (6. mai 2021) (2018).

33. Silver, N. *The Signal and the Noise: Why So Many Predictions Fail—But Some Don't* ISBN: 978-1-59-420411-1 (Penguin group, 2012).

34. *Skisseringsverktøyet Figma* https://figma.com (3. mar. 2021).

35. Su, X. Data analytics techniques. *NTNU IINI3012 Big data* (2020).

36. Taigman, Y., Yang, M., Ranzato, M. & Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. *Conference on Computer Vision and Pattern Recognition (CVPR).* https://research.fb.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/ (19. mai 2021) (2014).

37. *The Agile Manifesto* https://agilemanifesto.org/ (5. mai 2021).

38. Wang, J., Chen, Y., Hao, S., Peng, X. & Hu, L. Deep learning for sensor-based activity recognition: A survey. https://arxiv.org/pdf/1707.03502.pdf (10. mai 2021) (2017).

39. *XGBoost Documentation* https://xgboost.readthedocs.io/en/latest/.

40. *Økt aktivitet ikke alltid beste råd mot nakke- og ryggsmerter* https://gemini.no/2020/04/okt-aktivitet-ikke-alltid-losningen-mot-nakke-og-ryggsmerter/ (5. mai 2021).

Vedlegg

A Liste over vedlagte dokumenter i prosjektmappe

- Visjonsdokument
- Kravdokumentasjon
- Systemdokumentasjon
- Prosessdokumentasjon (Prosjekthåndbok)
- Datainnsamlingsdokument
- Gantt-diagram

B Modeller for de dype nevrale nettverkene

B.1 ANN

Layer	Input	Activation / Dropout	Output
Dense	64	ReLu	32
Dropout	—	0.3	—
Dense	32	ReLu	12
Dropout	—	0.3	—
Dense	12	ReLu	9
Dropout	—	0.3	—
Dense	9	Softmax	9

Tabell B.1: Lagdeling for ANN-modell med 1 sensor

Layer	Input	Activation / Dropout	Output
Dense	64	ReLu	24
Dropout	—	0.2	—
Dense	24	ReLu	9
Dropout	—	0.2	—
Dense	9	ReLu	9

Tabell B.2: Lagdeling for ANN-modell med 2 sensorer

Layer	Input	Activation / Dropout	Output
Dense	64	ReLu	48
Dropout	—	0.2	—
Dense	48	ReLu	24
Dropout	—	0.2	—
Dense	24	ReLu	12
Dropout	—	0.2	—
Dense	12	ReLu	9
Dropout	—	0.2	—
Dense	9	Softmax	9

Tabell B.3: Lagdeling for ANN-modell med 3 sensorer

Hyperparameter	**Verdi**
Epoch	ca. 1000
Batch_size	8192
Lr	0.00001
Optimizer	Adam
Callbacks	Early stopping

Tabell B.4: Hyperparametere for ANN-modell med 1 sensor

Hyperparameter	**Verdi**
Epoch	ca. 700
Batch_size	8192
Lr	0.00001
Optimizer	Adam
Callbacks	Early stopping

Tabell B.5: Hyperparametere for ANN-modell med 2 sensorer

Hyperparameter	**Verdi**
Epoch	ca. 700
Batch_size	8192
Lr	0.000001
Optimizer	Adam
Callbacks	Early stopping

Tabell B.6: Hyperparametere for ANN-modell med 3 sensorer

B.2 CNN

Layer	Input	Activation	Output
Batch_normalization	39xX	None	39x1
Conv1d	39x1	ReLU	39x128
Conv1d	39x128	ReLU	39x64
Conv1d	39x64	ReLU	39x32
Dropout	—	None	—
Max_pooling1d	39x32	None	19x32
Flatten	19x32	None	608
Fully-connected	608	ReLU	64
Dropout	—	None	—
Fully-connected	64	Softmax	9

Tabell B.7: Lagdeling for CNN-modell
Første input har en variabel X som representerer antall rader, som er sensorfrekvens*sekunder med innspilt data. Siste output er antall sittestillinger vi har valgt ta i bruk, som er 9.

Hyperparameter	Verdi
Epoch	60
Batch_size	192
Lr	0.000005
Optimizer	Adam

Tabell B.8: Hyperparametere for CNN-modell

B.3 LSTM

Layer	Input	Activation / Dropout	Output
LSTM (64 celler)	100xX	Sigmoid	64
Dense	64	ReLu	32
Dropout	—	0.2	—
Dense	32	ReLu	12
Dropout	—	0.2	—
Dense	12	ReLu	9
Dropout	—	0.2	—
Dense	9	Softmax	9

Tabell B.9: Lagdeling for LSTM-modell med 1 sensor

Layer	Input	Activation / Dropout	Output
LSTM (128 celler)	100xX	Sigmoid	128
Dense	128	ReLu	64
Dropout	—	0.2	—
Dense	64	ReLu	10
Dropout	—	0.2	—
Dense	10	ReLu	9
Dropout	—	0.2	—
Dense	9	Softmax	9

Tabell B.10: Lagdeling for LSTM-modell med 2 sensorer

Layer	Input	Activation / Dropout	Output
LSTM (128 celler)	100xX	Sigmoid	128
Dense	128	ReLu	64
Dropout	—	0.2	—
Dense	64	ReLu	10
Dropout	—	0.2	—
Dense	10	ReLu	9
Dropout	—	0.2	—
Dense	9	Softmax	9

Tabell B.11: Lagdeling for LSTM-modell med 3 sensorer

For samtlige modeller gjelder det at første input har en variabel X som representerer antall rader, som er sensorfrekvens*sekunder med innspilt data.

Hyperparameter	**Verdi**
Epoch	150
Batch_size	200
Lr	0.0001
Optimizer	Adam
# Timestamps	100

Tabell B.12: Hyperparametere for LSTM-modell med 1 sensor

Hyperparameter	**Verdi**
Epoch	120
Batch_size	200
Lr	0.0001
Optimizer	Adam
# Timestamps	100

Tabell B.13: Hyperparametere for LSTM-modell med 2 sensorer

Hyperparameter	**Verdi**
Epoch	100
Batch_size	200
Lr	0.0001
Optimizer	Adam
# Timestamps	100

Tabell B.14: Hyperparametere for LSTM-modell med 3 sensorer

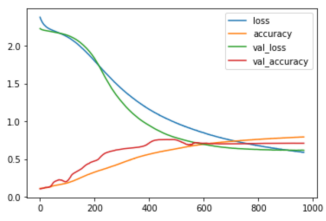
C Større visualiseringer

C.1 Trening av nevrale nettverk

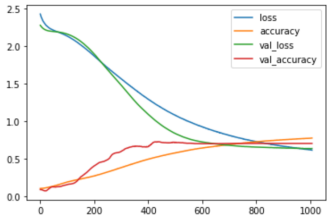
C.1.1 ANN

Figur C.1: Diagrammer for trening av ANN-modeller

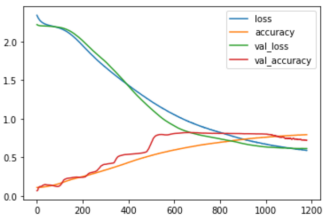
Trening av modell med 1 sensor



(a) Kjøring 1

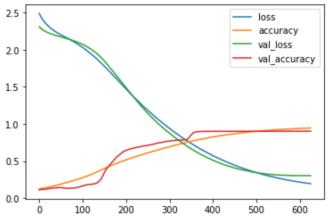


(b) Kjøring 2

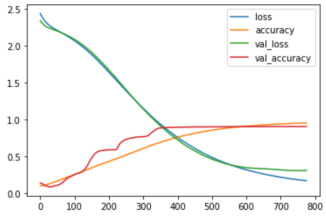


(c) Kjøring 3

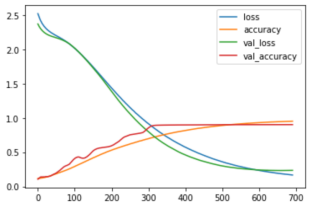
Trening av modell med 2 sensorer



(d) Kjøring 1

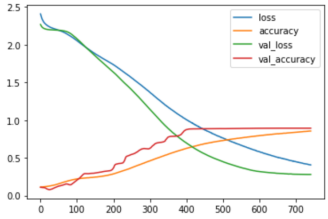


(e) Kjøring 2

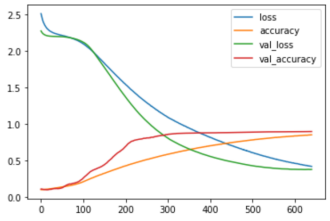


(f) Kjøring 3

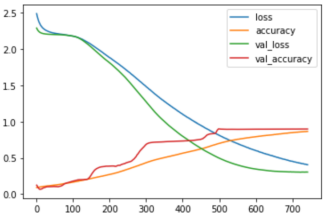
Trening av modell med 3 sensorer



(g) Kjøring 1



(h) Kjøring 2

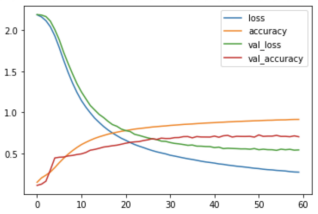


(i) Kjøring 3

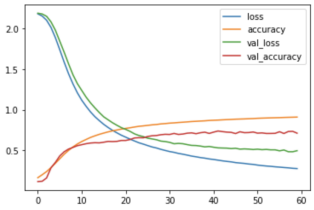
C.1.2 CNN

Figur C.2: Diagrammer for trening av CNN-modeller

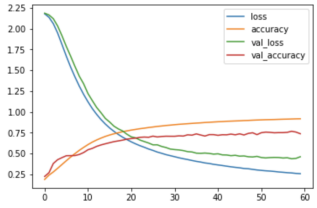
Trening av modell med 1 sensor



(a) Kjøring 1

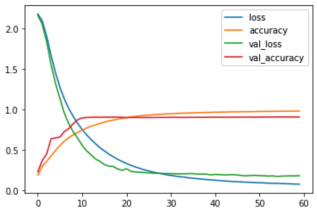


(b) Kjøring 2

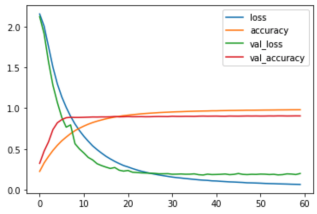


(c) Kjøring 3

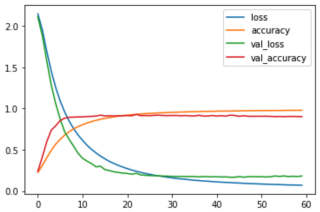
Trening av modell med 2 sensorer



(d) Kjøring 1

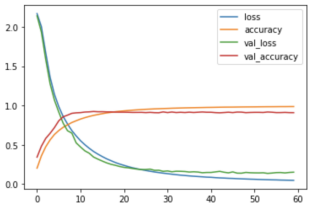


(e) Kjøring 2

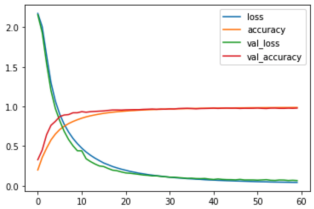


(f) Kjøring 3

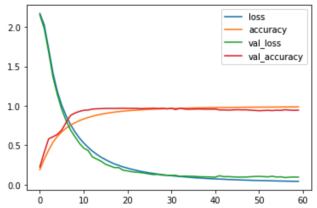
Trening av modell med 3 sensorer



(g) Kjøring 1



(h) Kjøring 2

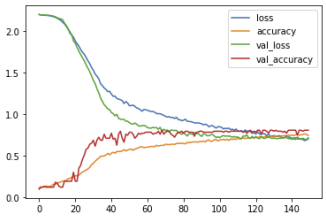


(i) Kjøring 3

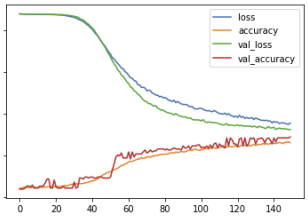
C.1.3 LSTM

Figur C.3: Diagrammer for trening av LSTM-modeller

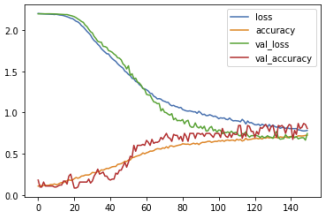
Trening av modell med 1 sensor



(a) Kjøring 1

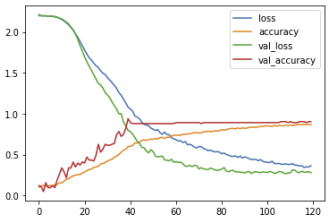


(b) Kjøring 2

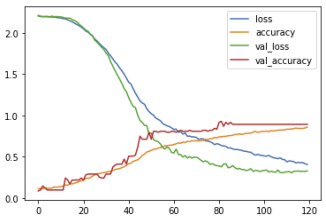


(c) Kjøring 3

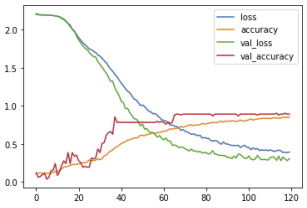
Trening av modell med 2 sensorer



(d) Kjøring 1

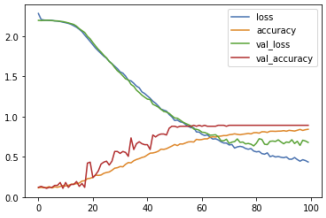


(e) Kjøring 2

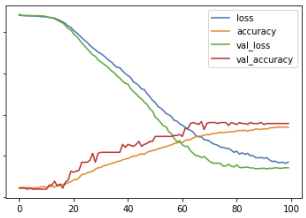


(f) Kjøring 3

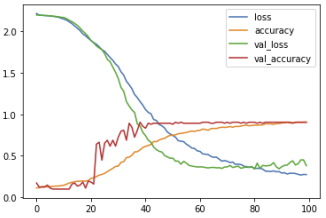
Trening av modell med 3 sensorer



(g) Kjøring 1



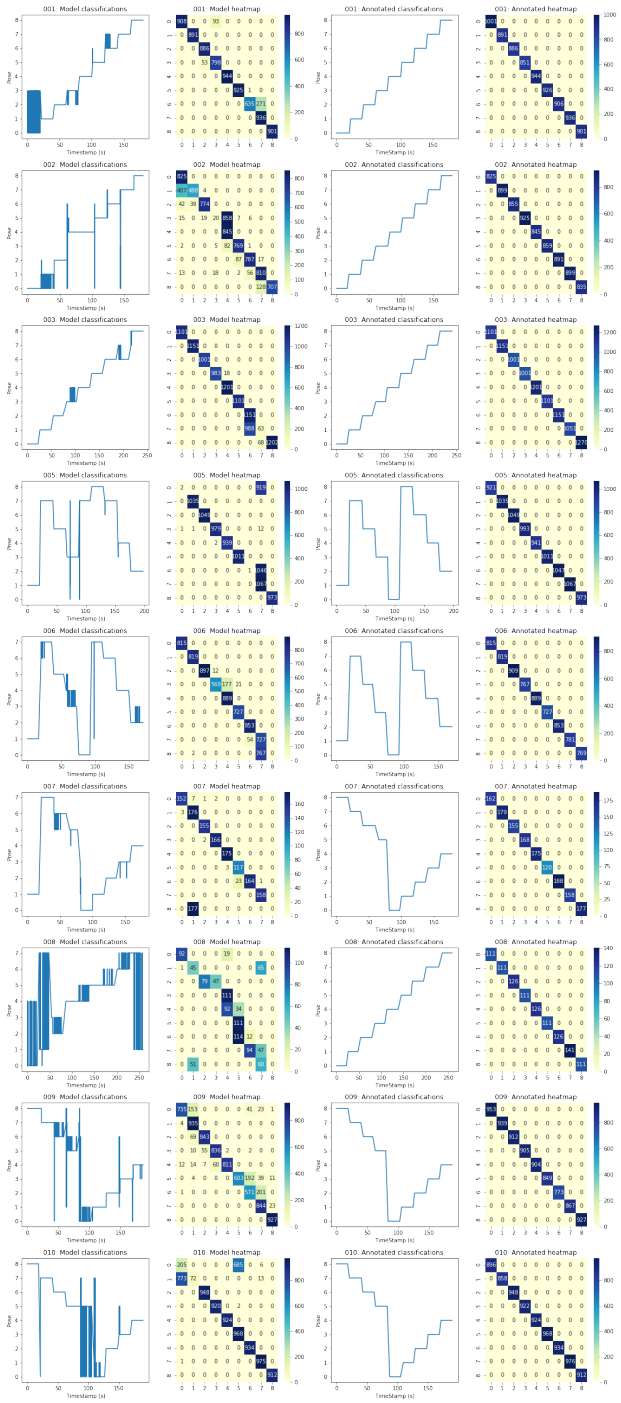
(h) Kjøring 2



(i) Kjøring 3

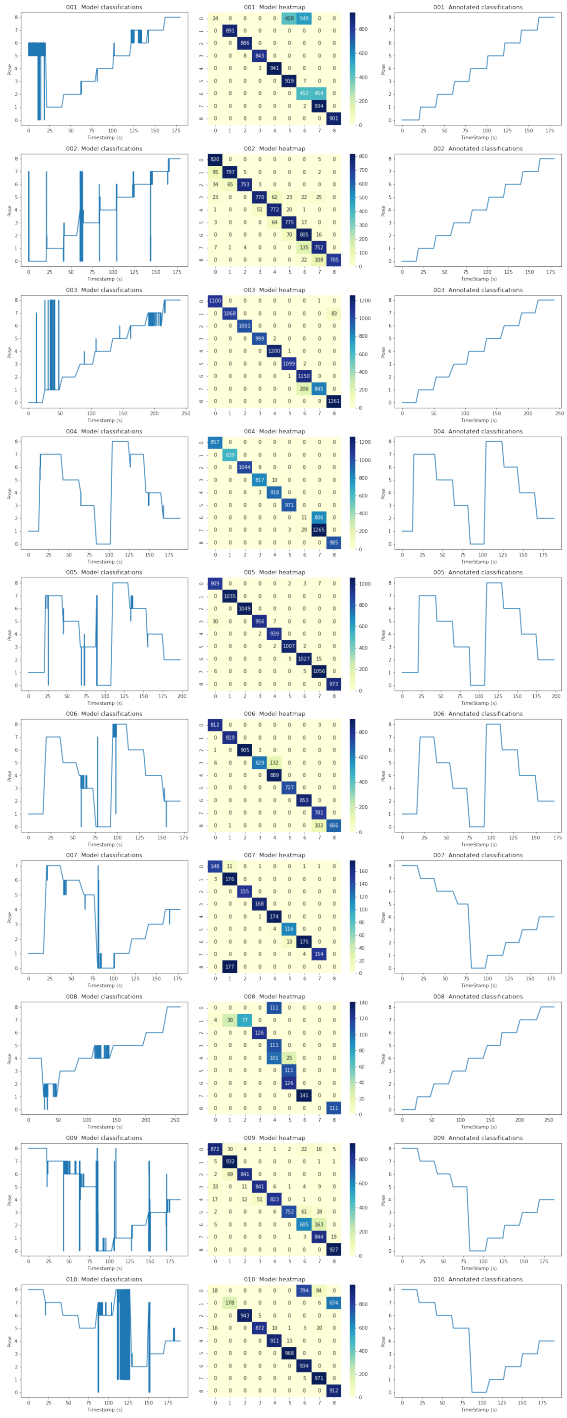
C.2 Trening av nevrale nettverk

C.2.1 RFC



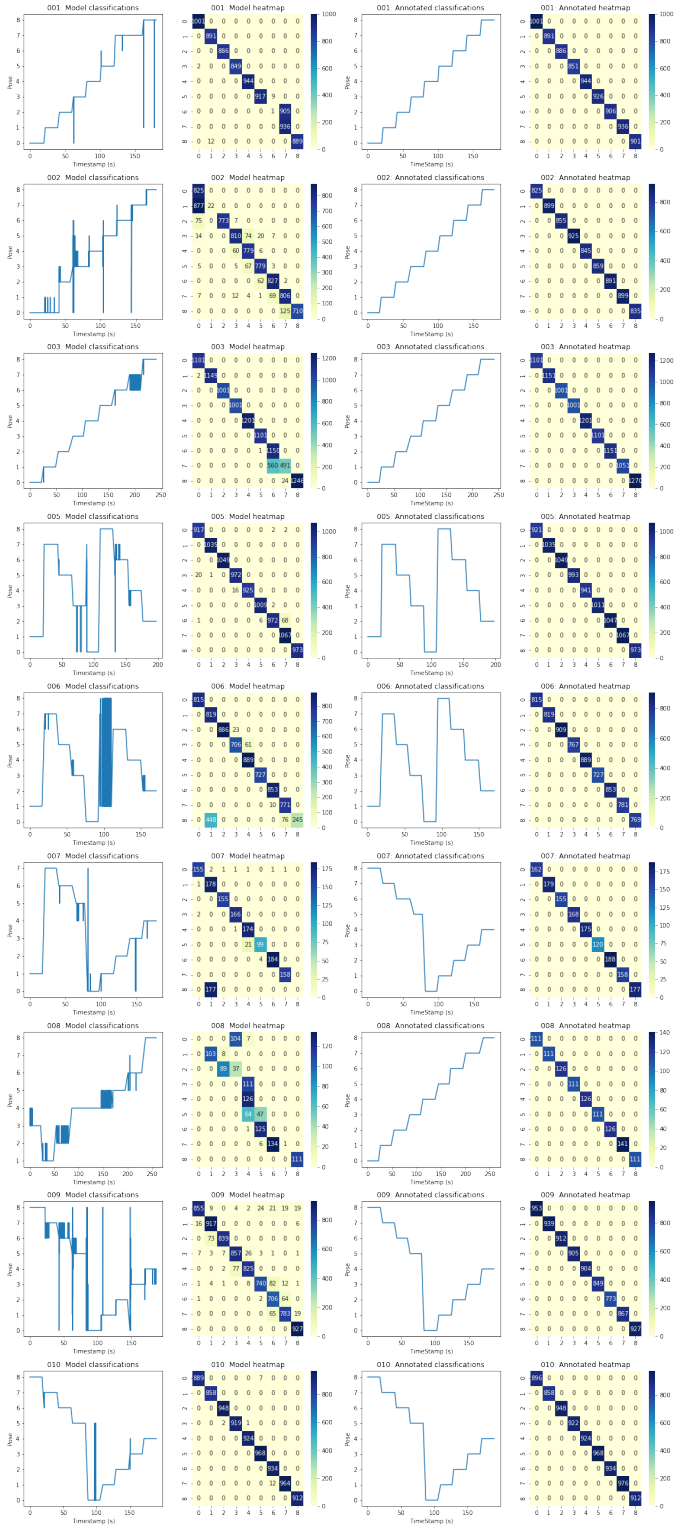
Figur C.4: Grafer og forvirringsmatriser for annotert testsett vs. modellens egen klassifisering ved testing av RFC

C.2.2 ANN



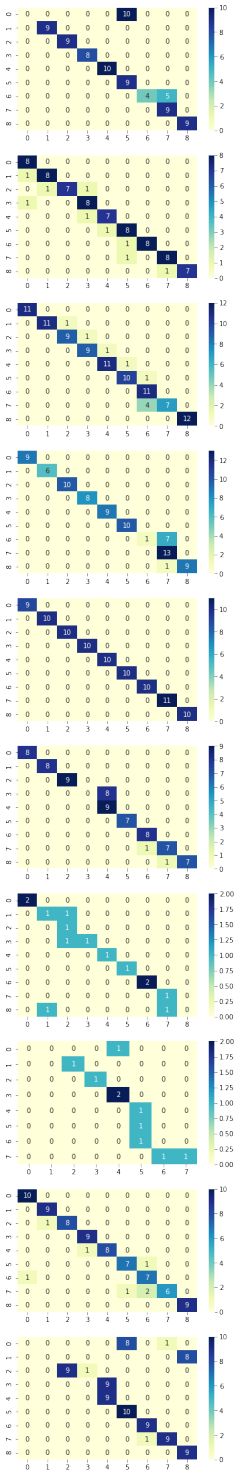
Figur C.5: Grafer og forvirringsmatriser for annotert testsett vs. modellens egen klassifisering ved testing av ANN

C.2.3 CNN



Figur C.6: Grafer og forvirringsmatriser for annotert testsett vs. modellens egen klassifisering ved testing av CNN

C.2.4 LSTM



Figur C.7: Heatmap ved testing av LSTM

På grunn av LSTMs tredimensjonale datastruktur er det vanskelig å fremstille et reelt linjediagram. En forvirringsmatrise, som viser det samme, blir brukt i stedet.

D Møtereferat fra møte med fysioterapaut Janne-Birgitte B. B. Børke ved St. Olavs Hospital

Man har i mange år trodd at det var holdning som var avgjørende for god rygghelse. Man ser at det er noe i det med holdning i forhold til pust osv, men ifølge dokumentasjon har man ikke klart å dokumentere at sittestillinger har noe som kan bidra til at det ødelegger rygghelse.

Side for muskel- og skjelettlidelser.

Forskningsmiljø har vært på kollisjonskurs med ryggforeninger og de som har ryggplager. Forskninga sier at man har ikke klart å påvise at spesielle stillinger gjør det verre enn andre stillinger. Sitter man i en stilling må man skifte. Muskelaturen liker ikke å være i en bestemt stilling hele tiden. "Den neste stillingen er den beste".

Stoler har ikke noe å si, alle har forskjellige gener. Alle er forskjellige, og har forskjellige gener og vil føle ting annerledes.

Når bør man skifte stilling? Varierer veldig fra person til person. Det kan derfor være lurt å være bevisst på sittestillingen og variere den litt mer.

De som sitter mye fremoverlent som en potetsekk kan kjenne det i nakken etter hvert. "Man skal ikke sitte på underlivet, man skal sitte på rumpa". Man skal altså ikke sitte og spenne muskulatur.

Hypotesen er at det er smart å også styrke kjernemuskulaturen for å støtte opp, men den er kanskje viktigere for fysisk aktivitet.

Variering i ryggen er key.

Mer interessant: Hvem får flere plager ut fra de sittestillingene man allerede har?

God fysisk form og bedre muskulatur bidrar selvsagt. Mange får ryggplager fra kort muskulatur.

Hvor på ryggen burde man plassere sensorer?

- Mer opptatt av bekkenbevegelsen
- Bekkenbevegelsen er avhengig av muskulaturen av setet og bein-muskulaturen
- Måle gjesleddet og C7?

Forskning på folk på hjemmekontor? Noe trening må man ha. Lite forskning på sittestillinger siste tiåret.

Snakke med han Kjartan Fersum ved UiB? Tror han er rette mannen å snakke med, og har skrevet artikler om temaet.