



## Research stage

Objectieve puntentelling in sport

IT factory

Wout Mergaerts

Academiejaar 2019-2020

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

# INHOUDSTAFEL

|  |           |
|--|-----------|
| <b>INHOUDSTAFEL .....</b>                    | <b>3</b>  |
| <b>1 INLEIDING .....</b>                     | <b>4</b>  |
| <b>2 POSE ESTIMATION .....</b>               | <b>5</b>  |
| <b>2.1 Research .....</b>                    | <b>6</b>  |
| 2.1.1 DeepLabCut .....                       | 6         |
| 2.1.2 LEAP .....                             | 7         |
| 2.1.3 Conclusie .....                        | 8         |
| <b>3 WEBAPPLICATIE .....</b>                 | <b>9</b>  |
| <b>3.1 Flask .....</b>                       | <b>9</b>  |
| <b>3.2 Tensorflow Lite .....</b>             | <b>10</b> |
| <b>3.3 TensorRT.....</b>                     | <b>10</b> |
| <b>4 VERBETERING POSE ESTIMATION.....</b>    | <b>11</b> |
| <b>4.1 DeepLabCut functies .....</b>         | <b>11</b> |
| 4.1.1 Filterpredictions .....                | 11        |
| 4.1.2 Extract_outlier_frames .....           | 11        |
| <b>4.2 Post-processing .....</b>             | <b>12</b> |
| 4.2.1 Prophet.....                           | 12        |
| 4.2.2 ML modellen.....                       | 12        |
| 4.2.3 Rolling median.....                    | 13        |
| 4.2.3.1 <i>Rolling median + std</i> .....    | 14        |
| <b>5 CLASSIFICATIE VAN OEFENINGEN .....</b>  | <b>15</b> |
| <b>6 SCOREN OP OEFENINGEN.....</b>           | <b>16</b> |
| <b>6.1 Groepen .....</b>                     | <b>16</b> |
| <b>6.2 Data .....</b>                        | <b>16</b> |
| 6.2.1 Hoeken .....                           | 16        |
| 6.2.2 Schalen .....                          | 16        |
| <b>6.3 SMOTER-GN .....</b>                   | <b>17</b> |
| <b>6.4 Tabel.....</b>                        | <b>17</b> |
| 6.4.1 Kolommen .....                         | 18        |
| <b>21</b>                                    |           |
| <b>6.5 Conclusie .....</b>                   | <b>22</b> |
| <b>7 BIBLIOGRAFIE .....</b>                  | <b>23</b> |
| <b>7.1 Pose Estimation .....</b>             | <b>23</b> |
| <b>7.2 Webapplicatie .....</b>               | <b>23</b> |
| <b>7.3 Verbetering Pose Estimation .....</b> | <b>23</b> |
| <b>7.4 Classificatie van oefeningen.....</b> | <b>24</b> |
| <b>7.5 Scoren op oefeningen .....</b>        | <b>24</b> |

# **1        INLEIDING**

Mijn stage gaat over hoe men een objectieve jury kan maken voor verschillende sporten door middel van Pose Estimation.

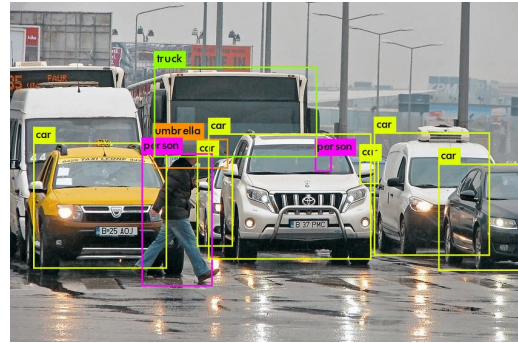
Aangezien deze methode nooit eerder gedaan is en Pose Estimation ook een vrij nieuwe techniek is, heb ik tijdens mijn stage ook veel research gedaan rond de verschillende modellen die hiervoor gebruikt kunnen worden en welke technieken gebruikt zouden kunnen worden om deze te verbeteren. Verder heb ik ook research gedaan naar de verschillende backends die gebruikt zouden kunnen worden voor een webapplicatie met een AI. Ten slotte heb ik ook even research gedaan naar de classificatie van verschillende oefeningen.

Een groot deel van deze research heb ik origineel in het Engels geschreven met de gedachte dat de klant, die niet Nederlandstalig is, deze eventueel zou lezen. Deze heb ik vervolgens opnieuw vertaald naar het Nederlands voor dit bewijsstuk.

## 2 POSE ESTIMATION

Pose Estimation is het lokaliseren van de verschillende onderdelen, ook wel keypoints, van mensen of dieren in een afbeelding of video. Het is een uitdaging die de computerwereld al enkele decennia bezighoudt en waar de laatste paar jaren zeer grote vooruitgang in is geboekt door onder andere de ook zeer grote vooruitgang in Deep Learning algoritmes.

Pose Estimation is bijvoorbeeld een veel grotere uitdaging dan objectdetectie. Bij objectdetectie wordt er op een veel lager niveau gekeken, er wordt "gewoon" een vakje rond het gedetecteerde object gezet waar het zich in de afbeelding/video bevindt. Daartegenover wordt er bij Pose Estimation gekeken naar elke keypoint en waar deze zich bevindt in de afbeelding/video. Een ander, nog moeilijker, aspect van Pose Estimation ten opzichte van objectdetectie is dat mensen/dieren veel "flexibeler" zijn dan objecten. Omdat wij mensen en dieren veel bewegen, zullen de eerdergenoemde keypoints zich dus niet altijd op dezelfde positie of hoek ten opzichte van elkaar bevinden.



*Objectdetectie*

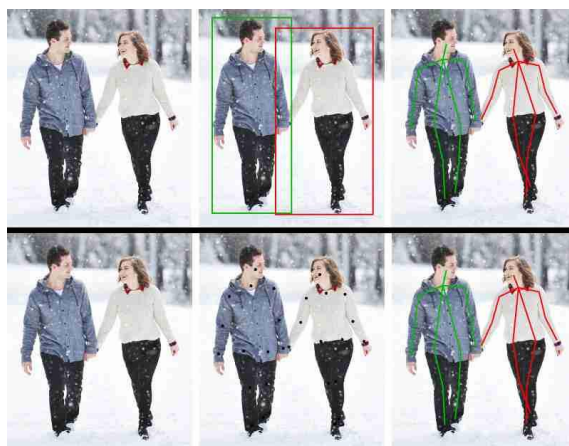


*Pose Estimation*

Voor Pose Estimation heb je veel verschillende mogelijkheden hoe of waarop je deze wilt uitvoeren. Ten eerste is dit de input, waarbij er wordt gezien of je RGB-afbeeldingen, Time of Flight-afbeeldingen of infrarood afbeeldingen hebt. Ook zal je moeten beslissen of je op afbeeldingen of op video's wil werken. Natuurlijk is deze video ook een reeks afbeeldingen, maar sommige modellen maken bijvoorbeeld gebruik van de vorige afbeelding uit de reeks om te helpen bij het voorspellen waar de keypoints deze keer zullen zijn. Een derde, zeer belangrijk, punt om te beslissen is of je 2D of 3D Pose Estimation wil doen. Waar bij 2D gewoon naar de X- en Y-coördinaten wordt gekeken van de verschillende keypoints in de video, zal er bij 3D nog een extra Z-coördinaat bijkomen. Veel 3D Pose Estimation modellen maken echter gebruik van verschillende 2D voorspellingen van verschillende aspecten om deze dan te combineren tot een 3D voorspelling. Ten slotte is er nog het lichaam zelf dat geanalyseerd moet worden. Hierbij kan men bijvoorbeeld gebruik maken van een skeletmodel dat bestaat uit de eerder genoemde keypoints. Ook zijn er sommige modellen die met een mesh werken, waarbij het lichaam in een soort pak van puntjes en lijntjes wordt verwerkt.

Voor het voorspellen van de pose zelf zijn er 2 grote categorieën over hoe je dit kan benaderen.

- **Top-down:** Zoals de naam al zegt, begint deze methode van bovenaf. Eerst wordt er door middel van objectdetectie gekeken waar er ongeveer mensen/dieren zijn. Hierna wordt er dan in deze kleine regio voorspeld waar de verschillende keypoints kunnen zijn.
- **Bottom-up:** Net zoals de naam, is ook de methode van deze benadering omgekeerd van de vorige. Hier wordt eerst recht-streeks gekeken naar waar de keypoints voor kunnen komen om deze vervolgens onderling te associëren met elkaar om een skeletmodel te kunnen vervolledigen.



*Top-down (boven) vs. Bottom-up (onder)*

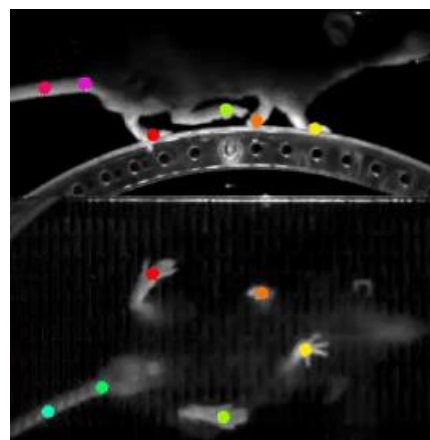
Geen model is echter absoluut perfect. Na deze voorspellingen moet je daardoor de voorspelde data een beetje bewerken en de rare voorspellingen eruit halen. Op die manier kan je ook het model verbeteren. Dit onderdeel zal besproken worden in deel 3 van deze research.

## 2.1 Research

Er zijn verschillende algoritmes die gebruikt kunnen worden voor Pose Estimation op mensen. Enkele voorbeelden hiervan zijn OpenPose, Deep(er)Cut, AlphaPose en nog veel meer. Deze algoritmes zijn echter gespecialiseerd in het vinden van het menselijk lichaam en de keypoints hierop en zijn daarom minder geschikt voor dit project. Enkele researchteams hadden hetzelfde probleem en hebben hun eigen algoritme geschreven, gebaseerd op bestaande modellen. Voor dit project heb ik twee kandidaten gevonden, namelijk DeepLabCut en LEAP

### 2.1.1 DeepLabCut

DeepLabCut is een deep-learning model dat gebruikt kan worden voor het voorspellen van de positie van zowel mensen als dieren. In tegenstelling tot traditionele manieren om dieren te tracken, gebeurt het met dit model "markerless". Bij vorige modellen werden markers aangebracht op de dieren tijdens dat ze gefilmd werden. Met dit model worden de markers digitaal op het dier gezet zodat elke video gebruikt kan worden. Hierna zal het model proberen om de markers, ofwel keypoints zoals ze eerder genoemd werden, op dezelfde plaats te annoteren, zoals in de afbeelding rechts te zien is.



*DeepLabCut*

DeepLabCut maakt gebruik van een voorgetraind model om deze markers te registreren dat gebaseerd is op het eerdergenoemde DeeperCut om de nodige hoeveelheid data drastisch te verminderen. De methode om zo'n voorgetraind model te herbruiken wordt transfer learning genoemd.

Door deze methode heb je "maar" 200 tot 300 afbeeldingen/frames van een video nodig om tot menselijke nauwkeurigheid de keypoints te kunnen laten aanbrengen door het model. Natuurlijk zal het model veel robuuster en nauwkeuriger worden als je er meer data aan geeft. Deze hoeveelheid data is echter zeer laag in vergelijking met traditionele Machine Learning modellen, waar je duizenden of tienduizenden afbeeldingen nodig zou hebben om deze nauwkeurigheid te kunnen verkrijgen.

Een nadeel hiervan is echter dat het model zeer lang getraind moet worden. Volgens de documentatie heb je voor deze nauwkeurigheid een training van 200 000 epochs nodig. Hiermee bedoelt men dat het model 200 000 keer opnieuw zal trainen op de data.

Ook heb je een goede grafische kaart nodig om deze lange training in een redelijke tijdsperiode te kunnen volbrengen. Voor het voorbeeld van 200 000 epochs zal het ongeveer 4 uur duren met een NVIDIA GeForce GTX 1080Ti. Door de opbouw van het model zal deze tijd ook mee schalen met de kwaliteit van de video's. Zo zullen 4K afbeeldingen veel langer nodig hebben om op getraind te worden dan 720p afbeeldingen.

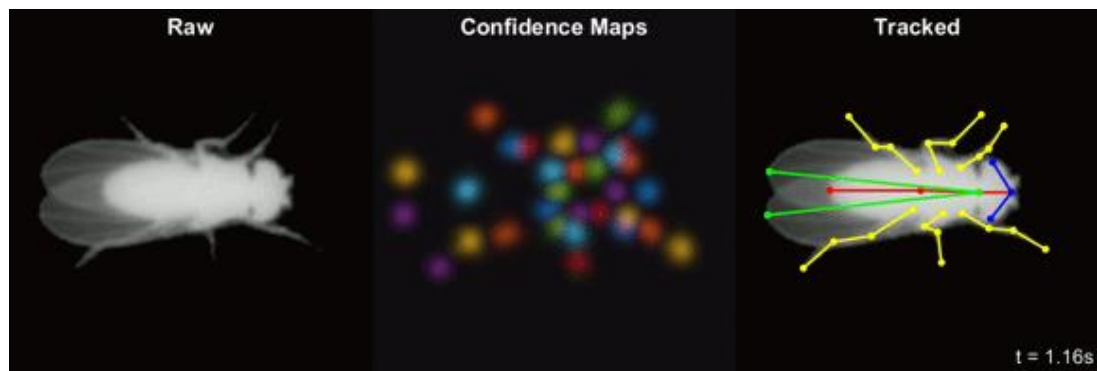
DeepLabCut kan ook gebruikt worden voor 3D Pose Estimation. Hiervoor kan je bijvoorbeeld een paar verschillende 2D modellen trainen van telkens een ander perspectief van het dier, of hier de sporter, en dan een extra model maken die deze verschillende perspectieven kan samenvoegen tot een driedimensionale ruimte. Deze toepassing kan in de toekomst misschien gebruikt worden voor een meer geavanceerde versie van het project.

### **2.1.2 LEAP**

LEAP, ofwel LEAP Estimates Animal Pose, is een ander deep-learning model dat gebruikt zou kunnen worden voor de Pose Estimation op dieren, net zoals DeepLabCut. Het is daarbij ook een "markerless" model waarbij je zelf digitaal de markers kan toevoegen. Deze kan je zien op de afbeelding op de volgende pagina.

LEAP, in tegenstelling tot DeepLabCut, blinkt uit door de zeer snelle trainingstijd. Dit is mogelijk doordat het model een neurale network is met maar 15 lagen in vergelijking met de 50 of 101 lagen van het DeepLabCut model. Deze snelheid komt echter met het nadeel dat het model niet zo robuust is met nieuwe data zoals bijvoorbeeld andere achtergronden of bewegingen. Om dit tegen te gaan zou je veel meer data nodig hebben dan DeepLabCut, die op dit moment niet beschikbaar is.

Een andere tegenstelling met het DeepLabCut model is dat LEAP niet gebruik maakt van een voorgetraind model. Dit is zowel een voor- als nadeel in dat je compleet van nul moet beginnen maar dat je toch wel een volledige vrijheid hebt over hoe je model werkt.



*LEAP*

### **2.1.3 Conclusie**

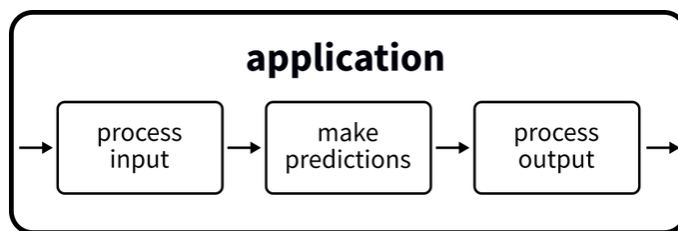
DeebLabCut lijkt een betere keuze voor dit project doordat het meer robuust is en tegelijk minder data nodig heeft. Zij hebben ook meer ondersteuning met zeer veel documentatie en een eigen actief forum. Brainjar is bij eerder onderzoek ook tot deze conclusie gekomen aangezien zij al een model met DeepLabCut getraind hebben.

### 3 WEBAPPLICATIE

Tijdens en na de stage zal de klant toegang willen hebben tot het model om dit te gebruiken voor eigen analyses en toepassingen. Een Jupyter notebook of Google Colab notebook is hiervoor niet echt gebruiksvriendelijk, aangezien dit vooral een ontwikkelingsomgeving is voor Python code. Er moet dus een andere manier zijn om het model op een gebruiksvriendelijke manier tot bij hen te kunnen brengen. Deze methode zal ook makkelijk bruikbaar en aanpasbaar moeten zijn. Hiervoor hebben we gekozen om een kleine webapplicatie te maken waar de klant een video kan uploaden en de applicatie een geannoteerde video terugstuurt die dan gedownload kan worden. In dit onderdeel zal ik verschillende technologieën bespreken die kunnen gebruikt worden als backend voor deze webapplicatie en welke de beste keuze is voor de doeleinden van dit project.



Communicating with the machine learning model



The application from a high level

*Schema van een webapplicatie met AI*

#### 3.1 Flask

Flask is een micro web framework. Dit betekent dat je de backend van functionele webapplicaties kan maken met relatief weinig code, geschreven in Python, zoals u op de afbeelding rechts kan zien. Dit framework is vooral bedoeld in ontwikkelingsomgevingen, maar voor een klein project als dit kan dit wel gebruikt worden als volwaardige backend. Als het project later echter veel groter wordt, is het beter om uit te breiden naar een andere backend, zoals bijvoorbeeld TensorRT.

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     return 'Hey, we have Flask in a Docker container!'
7
8
9 if __name__ == '__main__':
10     app.run(debug=True, host='0.0.0.0')
```

*Flask code*

Een ander voordeel van dit framework is dat het een van de meest populaire frameworks, als het niet de populairste is, om webapplicaties in Python te maken. Hierdoor is er zéér veel documentatie en handleidingen om hiermee te werken.

Voor deze webapplicatie zal ik deze methode gebruiken.



### 3.2 Tensorflow Lite

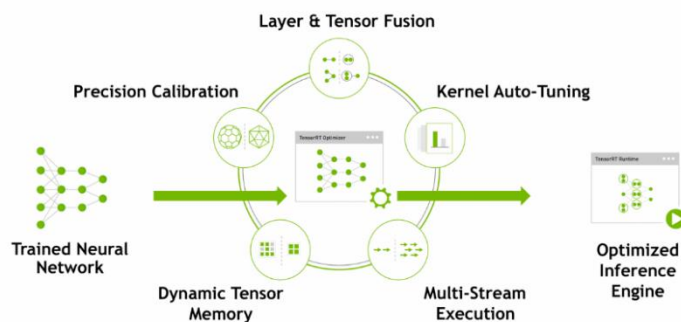
Tensorflow Lite is een lightweight versie van Google's Tensorflow framework. In tegenstelling tot de volwaardige versie, die geschreven is in Python, werkt de Lite versie in Javascript. Hierdoor kan men AI modellen op zo goed als elk apparaat uitvoeren en kan men verschillende dingen zoals latency, privacy, connectiviteit en stroomverbruik verbeteren.

Dit framework wordt ook gebruikt op Firebase, de hosting service van Google voor webapplicaties. Om een model te kunnen gebruiken hierop moet je het wel eerst converteren van Tensorflow naar Tensorflow Lite, wat niet altijd mogelijk is. Een ander, veel groter nadeel, is dat Tensorflow Lite het onmogelijk maakt om het model te laten samenwerken met een grafische kaart. DeepLabCut heeft dit echter wel nodig om annotaties in een behoorlijke tijdspanne te kunnen doen, dus is het onmogelijk om dit te gebruiken.

### 3.3 TensorRT

TensorRT is een SDK die gemaakt is door NVIDIA, de makers van grafische kaarten, zelf. Het is bedoelt voor webapplicaties op zeer grote schaal waar men verschillende servers met meerdere grafische kaarten ter beschikking heeft. TensorRT helpt hierbij om de webapplicatie te optimaliseren door bijvoorbeeld de voorspellingen automatisch te balanceren over de verschillende grafische kaarten die op dat moment beschikbaar zijn.

Als het project ooit op zeer grote schaal gemaakt wordt zou deze SDK misschien gebruikt kunnen worden, maar voor een kleine webapplicatie om het model te kunnen tonen is dit een beetje te veel.



*TensorRT*

## 4 VERBETERING POSE ESTIMATION

Een goede manier om een model te verbeteren is door het meer data te leveren. Spijtig genoeg is deze data niet altijd onmiddellijk, of zelfs helemaal niet, beschikbaar. Gelukkig zijn er nog enkele andere manieren om het model of de data die het genereert te verbeteren. Eerst heb ik onderzoek gedaan naar de functies die al ingebouwd zijn in het DeepLabCut model. Vervolgens heb ik onderzoek gedaan naar andere methodes om deze data te kunnen verbeteren.

### 4.1 DeepLabCut functies

#### 4.1.1 Filterpredictions

Deze functie zal over de annotaties van elke frame gaan en proberen om de fouten er uit te halen door ofwel gebruik te maken van een filter met de mediaan of een ARIMA (Autoregressive Integrated Moving Average) model. Deze nieuwe, gefilterde, annotaties kunnen dan gebruikt worden om geannoteerde video's te maken (door `filtered=True` te zetten op de functie om deze video's te maken). Denk er echter aan dat deze functie niet "magisch" alle fouten uit de data zal kunnen halen en deze op de juiste plaats zet.

Tijdens het testen van deze twee filters ben ik tot de conclusie gekomen dat de mediaan filter veel beter werkt om de fouten uit de data te halen en dit ook op een veel grotere snelheid gedaan te hebben dan de ARIMA filter.

#### 4.1.2 Extract\_outlier\_frames

Een andere manier om de annotaties te verbeteren is om over al de frames te gaan en manueel alle fouten er uit te halen en op de juiste plaats te zetten met de `extract_outlier_frames` functie. Deze aangepaste frames kunnen dan gebruikt worden als extra data om het model verder te kunnen trainen.

De functie heeft verschillende methodes om zulke frames met fouten te detecteren:

- Fitting: De functie zal eerst een ARIMA model trainen op elke keypoint van de data die gemaakt is door het model. Daarna zal het model op elke frame proberen te voorspellen waar deze data zou kunnen zijn. Als het dan te veel afwijkt of het model te onzeker was dat de keypoint op deze plaat zou kunnen zijn, wordt de frame gezien als een "outlier frame".
- Jump: De functie selecteert frames waarbij de keypoints opeens heel snel van plaats versprongen zijn tegenover het vorige frame.
- Uncertain: De functie selecteert frames waar het model niet zeker is of de keypoints juist staan.
- Manual: Je kan manueel frames selecteren

Een andere, zeer belangrijke, parameter van deze functie is de "numframes2extract" parameter in het config.yaml bestand. Dit zal de functie limiteren op hoeveel frames het kan detecteren in de dataset. Van alle gedetecteerde frames zal het deze limiteren tot dit aantal met de gekozen "extractionalgorithm" parameter (uniform of k-means).

Zoals waarschijnlijk al duidelijk was, kan deze methode enkel gebruikt worden op reeds geannoteerde video's. De geselecteerde frames worden daarna opgeslagen in een specifieke map. Hierna kan je zelf de locatie van de keypoints/markers verbeteren met de `refine_labels` functie.

Voorlopig zal ik deze functie niet gebruiken omdat ik zelf geen keypoints geplaatst heb en dus ook niet weet op welke exacte plaats zet moeten komen

## 4.2 Post-processing

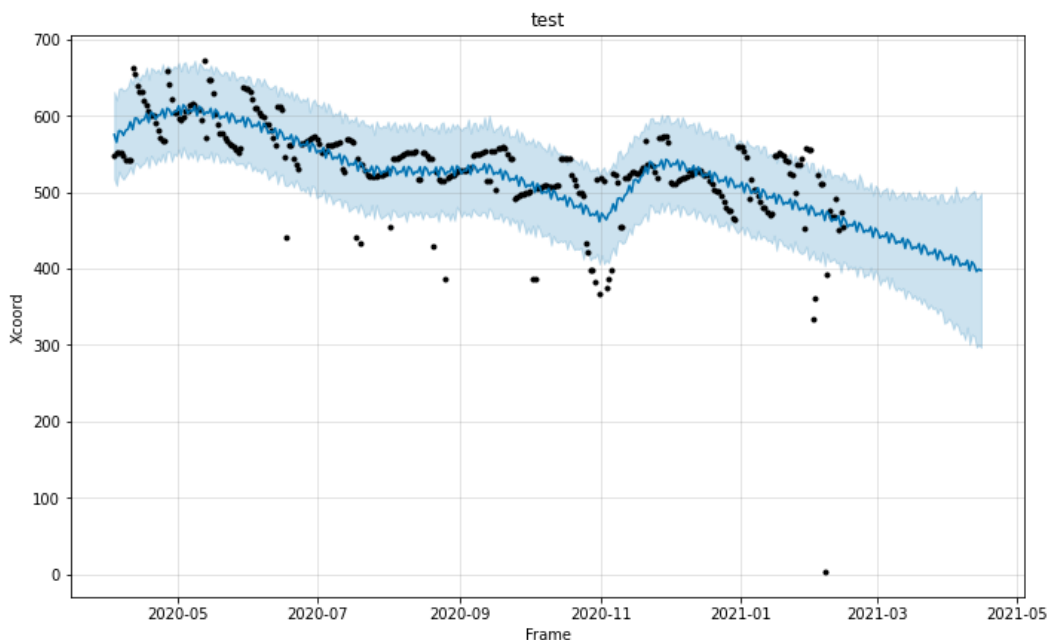
### 4.2.1 Prophet

Prophet is een methode die gemaakt is door Facebook om voorspellingen te maken op time series data. Time series zijn data die over een bepaalde periode gemaakt zijn. Voorbeelden hiervan zijn sensordata, data over de economie en natuurlijk annotaties doorheen een video.

Prophet werkt het beste op time series die een repetitief effect hebben. Het is ook zeer robuust tegen ontbrekende data in deze periode, als bijvoorbeeld een sensor even is uitgevallen, en tegen plotse veranderingen in trends in de data.

Hoewel de data die het DeepLabCut model genereert ook een time series is (omdat het gebaseerd is op coördinaten doorheen al de frames van de video's), zal het niet onmiddellijk kunnen werken met Prophet. Aangezien Prophet vooral gebruikt wordt voor financiële data, is de index gebaseerd op datums. Hierdoor heb ik de indexen moeten omvormen naar dit formaat.

Na enkele tests gedaan te hebben met Prophet op de data die gegenereerd is door DeepLabCut, blijkt het iets te robuust te zijn tegen de plotse veranderingen in trends. In de onderstaande afbeelding kan je zien dat de blauwe lijn, wat de voorspelling is van Prophet, heel veel van de zwarte puntjes, wat de data van DeepLabCut is, overslaat. Deze methode lijkt daardoor niet echt bruikbaar om de data te verbeteren

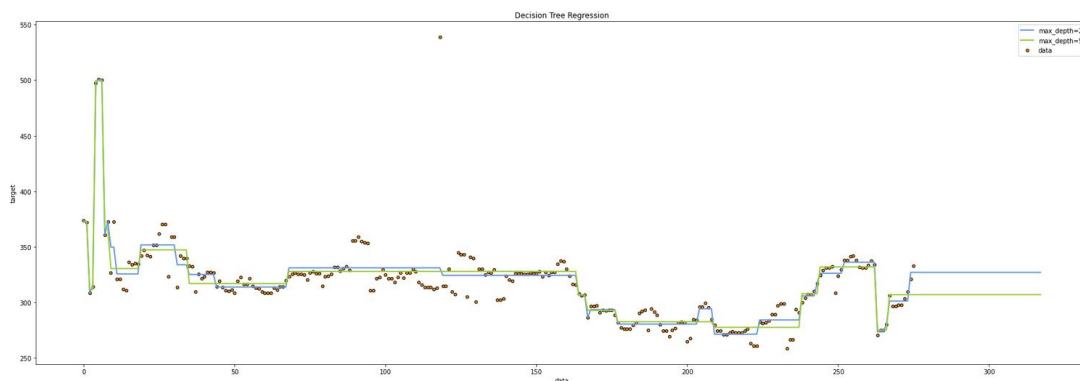


*Post-processing van 1 label met Prophet*

### 4.2.2 ML modellen

Ook heb ik geprobeerd om enkele Machine Learning modellen op de data te trainen en deze op elke frame te laten voorspellen om te kijken of zij ook de fouten er uit zouden kunnen halen.

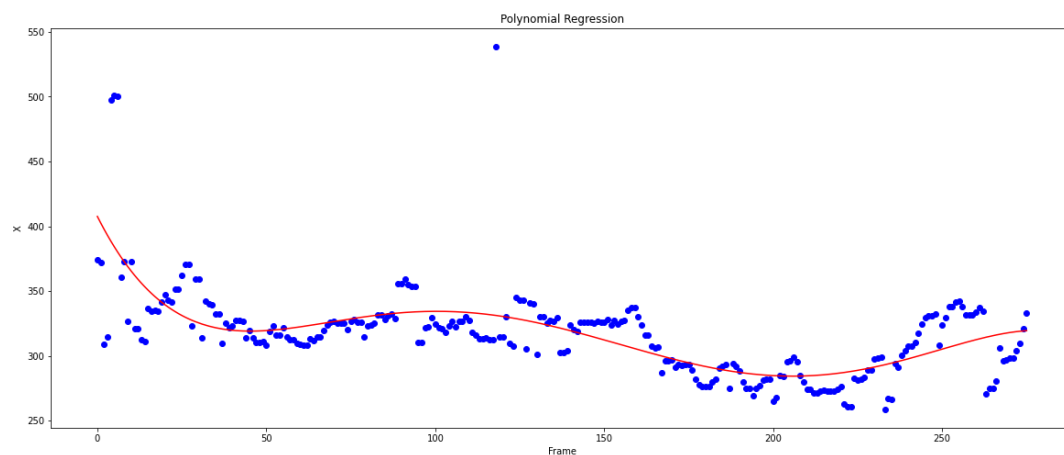
Het eerste model dat ik probeerde te trainen is een DecisionTree Regressor model. Door de max\_depth parameter aan te passen was het mogelijk om de "foutendetectie" van het model aan te passen. Zoals in de afbeelding te zien is zou het soms goed kunnen werken, maar over het algemeen werkt het helemaal niet.



*Post-processing van 1 label met verschillende DecisionTree modellen*

Hetzelfde heb ik geprobeerd met een Polynomial Regression model. Hiervoor heb ik eerst een Linear Regression model op de data getraind. Daarna heb ik de data getransformeerd met een Polynomial Features model en op deze nieuwe data een nieuw Linear Regression model getraind en laten voorspellen.

Zoals te zien is op de afbeelding, lijkt dit iets beter te werken dan het DecisionTree model, maar is het nog altijd niet goed genoeg om de data te kunnen verbeteren. Net zoals bij Prophet is de rode lijn, de voorspelling van het Polynomial Regression model, de blauwe puntjes helemaal niet goed aan het volgen.



*Post-processing van 1 label met een Polynomial Regression model*

### 4.2.3 Rolling median

Een rolling median is een mediaan die berekend wordt op telkens verschillende subsets, of windows, van de volledige dataset. Deze subset wordt telkens verplaatst in de dataset om telkens opnieuw een nieuwe mediaan te berekenen. Dit wordt doorheen heel de dataset herhaald en kan daarvoor gebruikt worden om de plotse fouten uit de data te halen.

Een belangrijk aspect van deze rolling median is de grootte van deze subset. Als deze te groot is, kan de mediaan beïnvloed worden door de fouten. Maar als deze te klein is, is de mediaan bijna letterlijk dezelfde waarde als de data.

Na wat testen op meerdere datasets lijkt deze methode vrij goed te werken met een specifieke grootte van de subset. Een nadeel is echter dat het alle data aanpast, waar bij als er geen fouten zijn de data amper aanpast, maar toch nog steeds aanpast. Dit kan misschien een grote invloed hebben aangezien deze data specifieke coördinaten zijn van de keypoints.

#### *4.2.3.1 Rolling median + std*

Een methode die dit probleem echter kan verhelpen is om de rolling median te gebruiken in combinatie met de standard deviation, of standaardafwijking in het Nederlands. De standaardafwijking is een waarde over hoeveel variatie er in de dataset zit, of in andere woorden hoeveel de data van mekaar verschilt.

Deze afwijking heeft, net zoals de mediaan, ook een rolling variant waarbij er op een kleine subset kan gekeken worden hoeveel de data van mekaar afwijkt. Door deze waarde kan specifiek gekeken worden of de data een fout is en enkel deze waarde kan dan aangepast worden naar de mediaan van die subset, zonder alle andere waardes aan te passen.

## 5 CLASSIFICATIE VAN OEFENINGEN

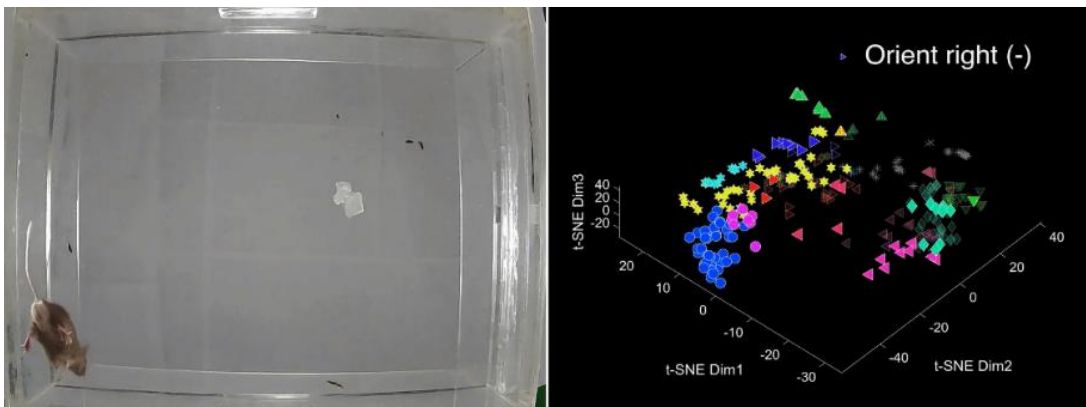
In elke sport zijn er zeer veel verschillende oefeningen die beoordeeld moeten worden. Om beter scores te kunnen berekenen is het daarom zeer handig om te weten welke oefening er bezig is om daarbij de juiste features uit de data te kunnen halen. Deze oefeningen zou je telkens manueel kunnen categoriseren of aanduiden, maar het zou veel beter zijn als dit automatisch gedaan zou kunnen worden door bijvoorbeeld een classificatiemodel. Een voorbeeld van zo'n model dat ook met dieren kan werken is B-SOiD.

B-SOiD, of Behavioral Segmentation Of Open Field in DeepLabCut, is een unsupervised model dat in combinatie met DeepLabCut gebruikt kan worden voor het categoriseren van verschillende gedragen van dieren.

Het model kan de verschillende gedragen groeperen door gebruik te maken van een t-SNE, ofwel T-distributed Stochastic Neighbor Embedding, model op de locatie van de verschillend keypoints die DeepLabCut heeft geannoteerd. Een voorbeeld hiervan dat door het research team gemaakt is is op een paar muizen. Het model kan hierbij herkennen of de muis aan het wandelen, zitten of andere dingen aan het doen is. De benaming van deze categorieën kan gedaan worden met een SVM, of Support Vector Machine, model.

Het B-SOiD model heeft echter wel enkele nadelen. Eerst en vooral is het gemaakt in MATLAB, wat een betaalde software is die niet tot mijn beschikking staat. Ten tweede is het model getraind op data met een vaste camera op eenzelfde omgeving, wat niet het geval zal zijn bij sporten waarbij er constante beweging is en er in verschillende omgevingen aan sport gedaan word. Ten slotte, aangezien dit een traditioneel Machine Learning model is heeft het zeer veel data nodig, wat ook niet beschikbaar is op dit moment.

Aangezien er eerst ook op maar één soort oefening gescoord zal moeten worden, zal dit onderdeel van mijn research waarschijnlijk niet gebruikt worden tijdens mijn stage.



*Classificatie van het "orient right" gedrag door B-SOiD*

## **6 SCOREN OP OEFENINGEN**

Een objectieve jury die geen of inaccurate punten geeft is ook niet nuttig. Om scores te geven aan de oefeningen maak ik gebruik van een paper waar een medestagiair al research naar heeft gedaan, namelijk "A Deep Learning Framework for Assessing Physical Rehabilitation Exercises" van A. Vakanski, Y. Liao en M. Xian.

Dit model werd door hen, en door de medestagiair, gebruikt om scores te kunnen geven op revalidatieoefeningen waardoor het niet helemaal zeker is of dit goed zal werken om sportoefeningen te scoren. Hiervoor zal ik onderzoek doen op de verkregen data van 1 oefening. Voor dit onderzoek heb ik gebruik gemaakt van 2 modellen uit de paper, het Spatio-Temporal Neural Network en het Convolutional Neural Network.

### **6.1 Groepen**

De exacte werking van de modellen kan u vinden in de paper, maar om het heel simpel uit te leggen wordt de data bij het Spatio-Temporal Model eerst in verschillende groepen verdeeld. Dit wordt niet gedaan bij het andere model.

Deze groepen worden eerst elk apart geanalyseerd, waarna deze analyses terug worden samengevoegd om een score te kunnen berekenen. Bij het originele model zijn deze groepen bijvoorbeeld linkerarm, rechterarm, lichaam, linkerbeen en rechterbeen.

Helaas zijn deze groepen niet mogelijk bij de sport waarvoor er scores moeten berekend worden. In de plaats daarvan heb ik 2 verschillende groeperingen uitgetest, elk ook bestaande uit 5 groepen zoals het originele model. Wegens de Non-Disclosure Agreement zijn deze groepen aangeduid als verschillend gekleurde vakken.

### **6.2 Data**

Voor het testen van dit model heb ik 75 video's van 1 oefening beschikbaar met de behaalde score van elke video en de tijd waarin de oefening werd uitgevoerd in de video. Deze video's zijn allemaal geanalyseerd door het DeepLabCut model en hebben elk een CSV-bestand met de posities van alle keypoints doorheen de video.

Deze data werd voor het model gecombineerd in één grote dataset, waarbij alle video's tot dezelfde lengte verlengd werden. Ook werden er door middel van de verbeteringen van hoofdstuk 4 nog enkele fouten uit gehaald.

#### **6.2.1 Hoeken**

De modellen werken origineel met hoeken tussen de verschillende keypoints. Hiervoor heb ik de data van posities van alle keypoints omgevormd naar hoeken, telkens tussen 2 keypoints die bij elkaar kunnen horen.

#### **6.2.2 Schalen**

De modellen werken beide met geschaalde data, waarbij de waardes tussen 0 en 1 liggen. Om de beschikbare data te schalen heb ik 3 manieren gebruikt:

- *Elke dataset apart schalen met een StandardScaler:* Deze methode geeft misschien een mooie dataset tussen 0 en 1, maar het is een zeer slechte manier om de data te schalen. Elke dataset wordt anders geschaald, waardoor dezelfde originele waarden een verschillende waarde zullen krijgen bij de geschaalde data van elke video.
- *Schalen door een vaste waarde:* Door alle data te schalen door een vaste waarde krijg je dezelfde verhoudingen tussen de waarden bij de geschaalde data. Voor de hoeken is deze waarde 360, voor de posities is dit 1920.
- *Schalen met 1 StandardScaler:* Een laatste manier om te schalen die ik probeer is om een StandardScaler te formatteren op de eerste dataset en deze dan te hergebruiken voor alle datasets. Hierdoor zullen de geschaalde verhoudingen ook telkens hetzelfde blijven.

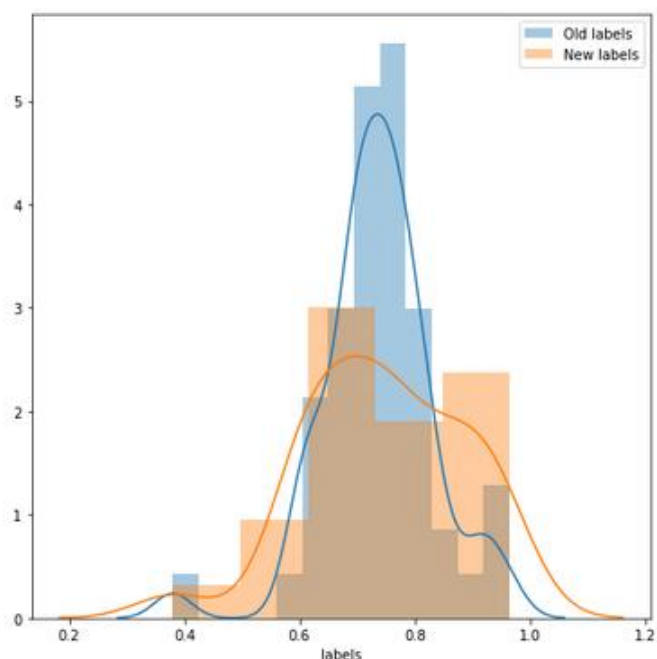
### 6.3 SMOTER-GN

De data die ik gekregen heb was maar zeer weinig voor een AI model, ook voor deze modellen. Bij de originele data die men gebruikte om de modellen te trainen had men 90 oefeningen met telkens 117 hoeken. Ook is deze data veel meer gevarieerd. Dit kan er voor zorgen dat het model geen goede voorspellingen, of hier scores, zal kunnen geven

Bij classificatie kan deze variatie makkelijk opgelost worden door bijvoorbeeld een groter gewicht te geven aan de klassen die minder voorkomen. Hierdoor zullen deze weinige klassen belangrijker worden voor het model om mee te trainen. Voor regressie is dit iets moeilijker, maar zijn er gelukkig enkele python bibliotheken die dit kunnen oplossen.

Een van deze bibliotheken is smogn. Deze maakt gebruik van een techniek om de data beter te variëren die SMOTER-GN, ofwel Synthetic Minority Over-Sampling

Technique for Regression with Gaussian Noise, genoemd wordt. In de figuur rechts kan u zien hoeveel van elke score er beschikbaar is in de data. In de originele data, de blauwe kolommen, waren er bijvoorbeeld zeer veel scores tussen 0,6 en 0,8. Hierdoor gaf het model bij de voorspellingen dan ook vooral scores tussen 0,6 en 0,8 omdat het vooral voorbeelden hiervan heeft gekregen. Met SMOTER-GN werden deze scores, en de bijbehorende data, beter verdeeld, zoals te zien is aan de oranje kolommen.



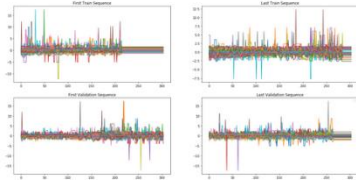

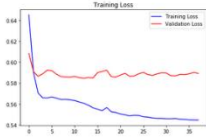
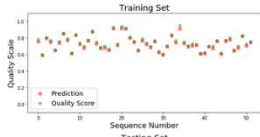
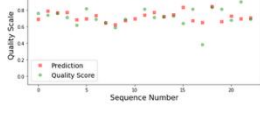
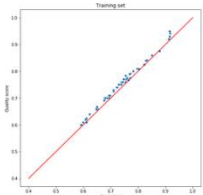
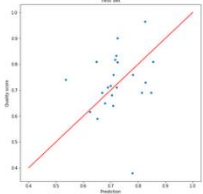
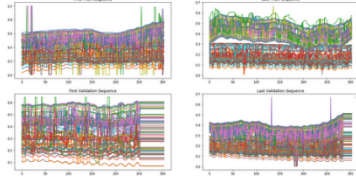
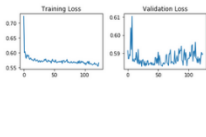
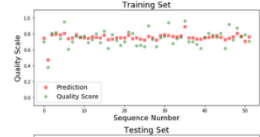
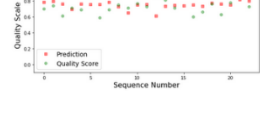
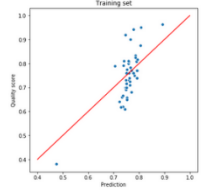
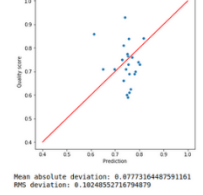
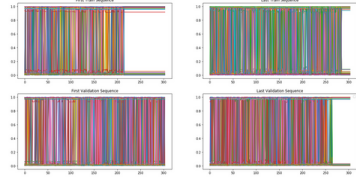
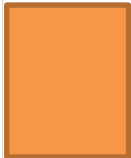
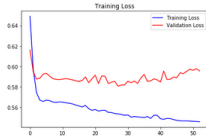
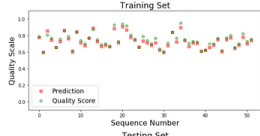
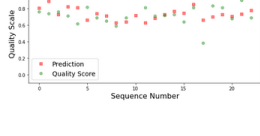
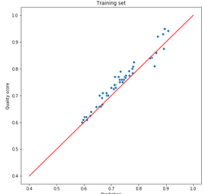
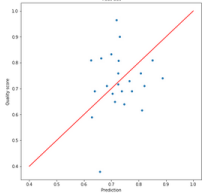

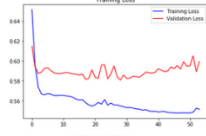
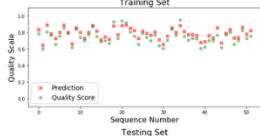
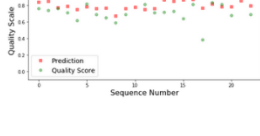
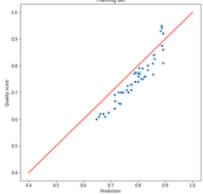
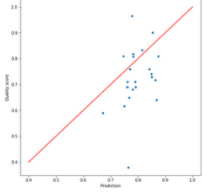
### 6.4 Tabel

Al de resultaten van de testen die ik gedaan heb met beide modellen heb ik in een tabel gegoten. De tabel is zeer groot door de vele testen, waardoor inzoomen soms nodig zal zijn om de grafieken en testen te kunnen bekijken.

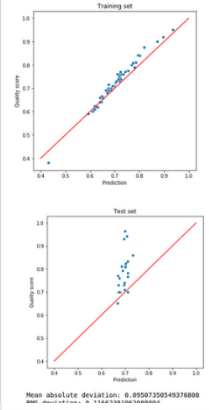
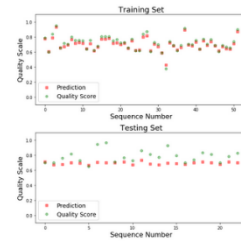
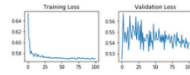


### 6.4.1 Kolommen

- *Data:* Informatie over de data set en wat ik ermee gedaan heb. Ook is er een grafiek van de eerste en laatste oefening van zowel de train als validatie set
- *Model:* Het model dat ik bij de test heb gebruikt
- *Grouping:* De groepen die ik bij het Spatio-Temporal model heb gebruikt bij de test
- *Loss:* De loss tijdens doorheen de trainingen van het model van zowel de training als validatieset
- *Prediction vs Actual score I:* De score die het model geeft aan de oefeningen in vergelijking met de eigenlijke score van die oefening. De groene vakjes zijn de eigenlijke scores en de rode cirkels zijn de scores die het model voorspelde
- *Prediction vs Actual Score II:* Een vergelijking tussen de score die het model geeft en de eigenlijke score om te zien of het model geen willekeurige scores geeft
  - o X: De voorspelde score van het model
  - o Y: De eigenlijke score
  - o Rode lijn: Als het model een perfecte voorspelling zou geven

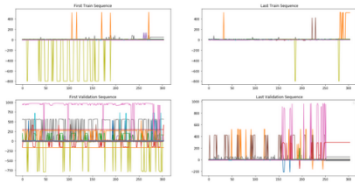
| Data   | Model             | Grouping  | Loss   | Prediction vs Actual score I  | Prediction vs Actual score II   |
|--|-------------------|---|--|---|---|
| <b>Scaled angles (bad way)</b> <ul style="list-style-type: none"> <li>Scaled each dataset separately</li> <li>45 features</li> <li>75 sequences</li> <li>304 frames</li> </ul>  | SpatioTemporal NN |    |  <p>Training loss 0.544585672788081<br/>Validation loss 0.584522994135818</p>     |       |     |
| <b>Scaled positions</b> <ul style="list-style-type: none"> <li>Divided by 1920</li> <li>110 features</li> <li>75 sequences</li> <li>304 frames</li> </ul>                       | CNN_Vicon         |   |   |       |   <p>Mean absolute deviation: 0.0773364487591361<br/>RMS deviation: 0.1824853276798879</p> |
| <b>Scaled angles</b> <ul style="list-style-type: none"> <li>Divided by 360</li> <li>45 features</li> <li>75 sequences</li> <li>304 frames</li> </ul>                          | SpatioTemporal NN |  |  <p>Training loss 0.5459843727471278<br/>Validation loss 0.5807269474734431</p> |   |     |
|  |                   |  |  <p>Training loss 0.5475794898732286<br/>Validation loss 0.58078853897924</p>   |   |     |

CNN Vicon

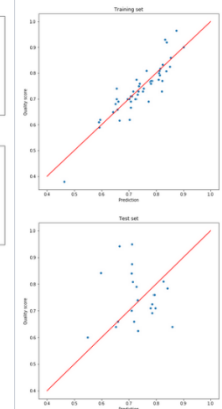
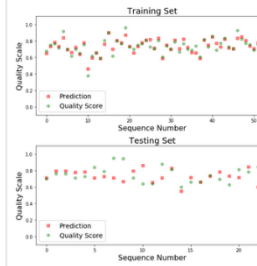
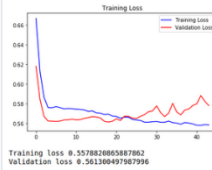
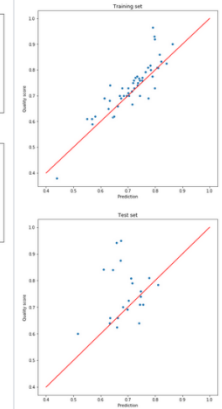
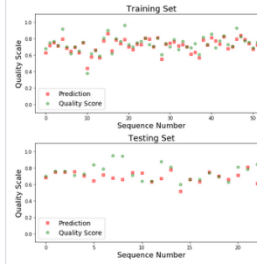
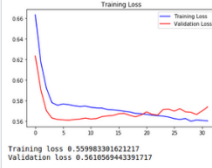
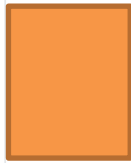


### Scaled angles

- Fit StandardScaler on first dataset and used to transform all datasets
- 45 features
- 75 sequences
- 304 frames

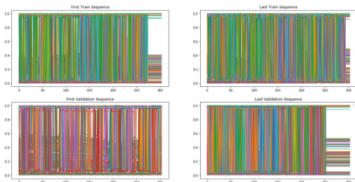


SpatioTemporal NN

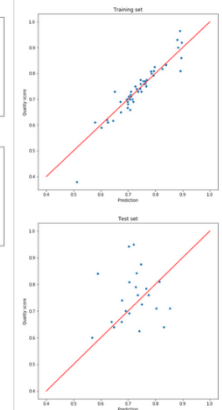
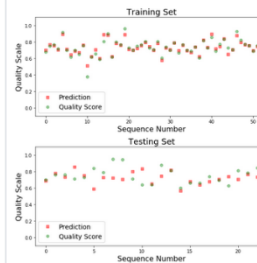
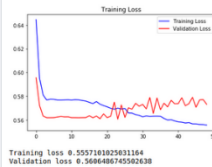


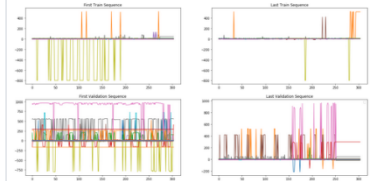
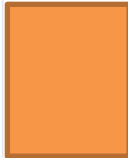
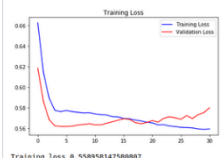
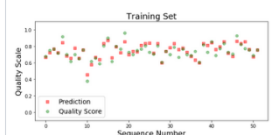
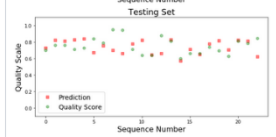
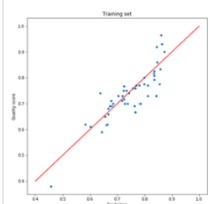
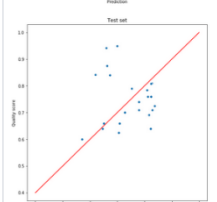
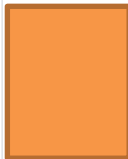
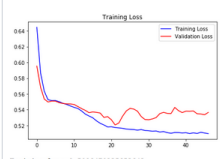
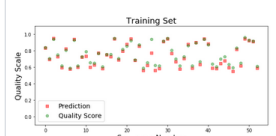
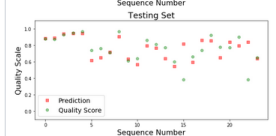
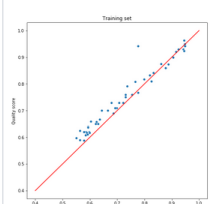
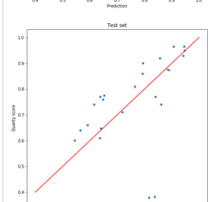
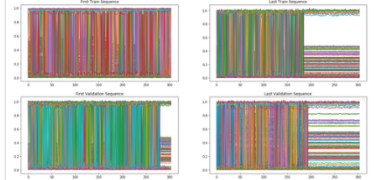
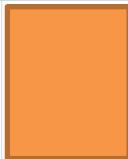
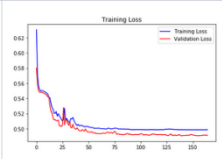
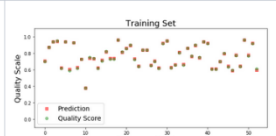
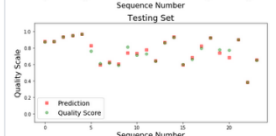
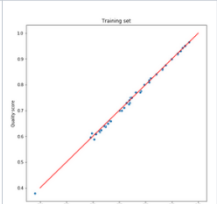
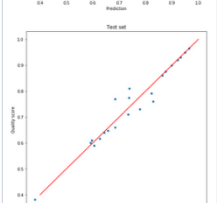

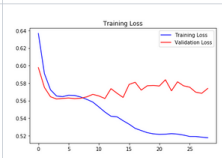

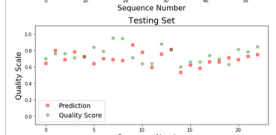
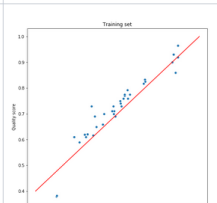
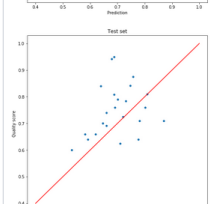
### Scaled angles & positions

- Angles divided by 360, positions divided by 1920
- 155 features
- 75 sequences
- 304 frames



SpatioTemporal NN



|   |   |  |   |   |
|---|---|--|---|---|
| <p><u>Scaled angles &amp; positions</u></p> <ul style="list-style-type: none"> <li>1 <u>StandardScaler</u> for angles, fit on the first dataset and used to transform all datasets. 1 <u>StandardScaler</u> for positions, fit on the first dataset and used to transform all datasets</li> <li>155 features</li> <li>75 sequences</li> <li>304 frames</li> </ul>  | <p><u>SpatioTemporal NN</u></p>    |  <p>Training loss 0.55895014758887<br/>Validation loss 0.5616889124531621</p>     |       |       |
| <p><u>Scaled angles &amp; positions + SMOTER</u></p> <ul style="list-style-type: none"> <li>1 <u>StandardScaler</u> for angles, fit on the first dataset and used to transform all datasets. 1 <u>StandardScaler</u> for positions, fit on the first dataset and used to transform all datasets. <u>SMOTER</u> applied to data for better distribution</li> <li>155 features</li> <li>75 sequences</li> <li>304 frames</li> </ul>                   | <p><u>SpatioTemporal NN</u></p>    |  <p>Training loss 0.5896478235653843<br/>Validation loss 0.5298016481236485</p>   |       |      |
| <p><u>Scaled angles &amp; positions + SMOTER</u></p> <ul style="list-style-type: none"> <li>Angles divided by 360, positions divided by 1920. <u>SMOTER</u> applied to data for better distribution.</li> <li>155 features</li> <li>77 sequences (2 extra created by SMOTER)</li> <li>304 frames</li> </ul>    | <p><u>SpatioTemporal NN</u></p>  |  <p>Training loss 0.4984916499208857<br/>Validation loss 0.4985258844688487</p> |   |   |
| <p><u>Scaled angles &amp; positions + SMOTER (only on training set)</u></p> <ul style="list-style-type: none"> <li>Angles divided by 360, positions divided by 1920. <u>SMOTER</u> applied to data for better distribution, only on the training set.</li> <li>155 features</li> <li>77 sequences (2 extra created by SMOTER)</li> <li>304 frames</li> </ul>  | <p><u>SpatioTemporal NN</u></p>  |  <p>Training loss 0.5176488249398213<br/>Validation loss 0.5618542899958888</p> |   |   |

## 6.5 Conclusie

Uit deze tabel kunnen we meerdere dingen concluderen:

- De Concurrent Neural Network geeft veel slechtere resultaten dan het Spatio-Temporal Neural Network. Dit was ook te verwachten aangezien dit laatste model het voorgestelde model is van de paper.
- De verschillende manieren van schalen gaven maar zeer lichte verschillen in hoe accuraat het model scores geeft, maar het schalen met een vaste waarde geeft iets betere resultaten.
- Het gebruik van SMOTER-GN op alle data geeft bijna perfecte voorspellingen. Dit kan zijn door het herverdelen van de dataset dat er dezelfde data in zowel de test als training set zit.
- Het beste resultaat, de bijna perfecte scores van het vorige punt niet meegerekend, kwam van zowel de posities als de hoeken te gebruiken waarbij beide door een vaste waarde geschaald werden en SMOTER-GN werd toegepast op de training set. Dit beste resultaat is echter niet uitzonderlijk veel beter dan de andere testen.

Het geven van de scores door dit model is nog lang niet perfect, maar door de grafieken in de laatste kolom is wel te zien dat het toch gebruikt zou kunnen worden, mits wat verbeteringen:

- *Verbeteringen van het DeepLabCut model:* In de data die ik gekregen heb zaten nog redelijk veel fouten die geannoteerd werden door het DeepLabCut model. Door deze fouten versprongen de hoeken en posities vaak, wat ook te zien is in de grafieken in de eerste kolom.
- *Meer en meer gevarieerde data:* Dit model heeft veel meer data nodig, zoals eerder al vermeld was. Naar mijn mening is er minstens dubbel zo veel data nodig dan wat ik nu beschikbaar heb gekregen. Ook moet deze data veel meer gevarieerde scores bevatten. Zelfs met het toepassen van SMOTER-GN had het model soms moeite om sommige scores accuraat te kunne voorspellen.

## 7 BIBLIOGRAFIE

### 7.1 Pose Estimation

Aldarondo, D. E. (20 December 2018). *Fast animal pose estimation using deep neural networks*. Opgehaald van Nature: <https://www.nature.com/articles/s41592-018-0234-5>

AlexEMG. (sd). *DeepLabCut*. Opgehaald van Github: <https://github.com/AlexEMG/DeepLabCut>

*DeepLabCut: A software package for animal pose estimation*. (sd). Opgehaald van Mouse Motor Lab: <https://www.mousemotorlab.org/deeplabcut/>

Mathis, A. (20 Augustus 2018). *DeepLabCut: markerless pose estimation of user-defined body parts with deep learning*. Opgehaald van Nature: <https://www.nature.com/articles/s41593-018-0209-y>

Mathis, A. (21 June 2019). *Using DeepLabCut for 3D markerless pose estimation across species and behaviors*. Opgehaald van Nature: <https://www.nature.com/articles/s41596-019-0176-0>

talmo. (sd). *LEAP*. Opgehaald van Github: <https://github.com/talmo/leap>

### 7.2 Webapplicatie

(sd). Opgehaald van Firebase: <https://firebase.google.com/>

*Deploy machine learning models on mobile and IoT devices*. (sd). Opgehaald van TensorFlow: <https://www.tensorflow.org/lite>

*Flask*. (sd). Opgehaald van The Pallets Projects: <https://palletsprojects.com/p/flask/>

*NVIDIA TensorRT*. (sd). Opgehaald van NVIDIA Developer: <https://developer.nvidia.com/tensorrt>

### 7.3 Verbetering Pose Estimation

AlexEMG. (sd). *DeepLabCut function details*. Opgehaald van Github: <https://github.com/AlexEMG/DeepLabCut/blob/f1f3aac06ca88ad477b3c1e18e64c3cd7381e759/docs/functionDetails.md>

Facebook. (sd). *Prophet*. Opgehaald van Github: <https://github.com/facebook/prophet>

Khandelwal, R. (15 November 2019). *Time series prediction using Prophet in Python*. Opgehaald van Towards Data Science: <https://towardsdatascience.com/time-series-prediction-using-prophet-in-python-35d65f626236>

## 7.4 Classificatie van oefeningen

Hsu, A. I. (16 September 2019). *B-SOiD: An Open Source Unsupervised Algorithm for Discovery of Spontaneous Behaviours*. Opgehaald van bioRxiv: <https://www.biorxiv.org/content/10.1101/770271v1>

Yttrilab. (sd). *B-SOID*. Opgehaald van Github: <https://github.com/YttriLab/B-SOiD>

## 7.5 Scoren op oefeningen

Kunz, N. (sd). *Synthetic Minority Over-Sampling Technique for Regression with Gaussian Noise*. Opgehaald van Github: <https://github.com/nickkunz/smogn>

Vakanski, A. (23 Januari 2020). *A Deep Learning Framework for Assessing Physical Rehabilitation Exercises*. Opgehaald van Arxiv: <https://arxiv.org/abs/1901.10435>

Vakanski, A. (sd). *A-Deep-Learning-Framework-for-Assessing-Physical-Rehabilitation-Exercises*. Opgehaald van Github: <https://github.com/avakanski/A-Deep-Learning-Framework-for-Assessing-Physical-Rehabilitation-Exercises>