FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

<RENNERHERKENNING VOOR SYNCHRONISATIE VAN VLAAMS WIELERERFGOED MET KOERS OP TV>

<Simon De Smul>
Studentennummer: 01505837

Promotor(en): Prof. dr. “Steven Verstockt”, Prof. dr. “Nico Van de Weghe”

Masterproef voorgelegd voor het behalen van de graad master in de Master of Science in de industriele wetenschappen : informatica

Academiejaar: 2018 – 2019

UNIVERSITEIT
GENT

Vertrouwelijk tot en met 31/12/2028

Belangrijk

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Rennerherkenning voor synchronisatie van Vlaams Wielererfgoed met koers op TV

Simon De Smul
Studentennummer: 01505837

Promotoren: prof. dr. Steven Verstockt, prof. dr. Nico Van de Weghe
Begeleiders: ir. Jelle De Bock, Jarich Braeckevelt

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Academiejaar 2018-2019

UNIVERSITEIT
GENT

Woord vooraf

Eerst en vooral wens ik prof. dr. Steven Verstockt, prof. dr. Nico Van de Weghe, ir. Jelle De Bock en Jarich Braeckevelt te bedanken voor het uitlijnen van een duidelijke planning en structuur tijdens de periode van deze masterproef.

In het bijzonder wens ik Prof. Dr. Steven Verstockt en ir. Jelle De Bock te bedanken. Prof. Dr. Steven Verstockt wens ik te bedanken voor het suggereren van alternatieve technologieën en/of methodieken tijdens cruciale momenten in het project. Ir. Jelle De Bock wens ik te bedanken voor de steun en expertise omtrent het onderwerp van *deep learning*.

Ten laatste wens ik ook de medewerkers van wielermuseum KOERS te Roeselare te bedanken, in het bijzonder Thomas Ameye.

Indien in deze thesis woorden worden ontleend aan de Engelse taal zullen deze de eerste maal cursief worden geschreven.

31/03/2019

Simon De Smul, Master of Science in de industriële wetenschappen : informatica, Universiteit Gent
Abstract behorend tot master thesis, Ingegeven 21 augustus 2019
Rennerherkenning voor synchronisatie van Vlaams Wielererfgoed met koers op TV

Het doel van deze masterproef is het herkennen van wielerploegen op basis van videobeelden. Meer specifiek de uitzending van een wielerwedstrijd, zoals uitgezonden door een bepaalde omroep.

Dit werd gerealiseerd door het ontwerpen en trainen van een *Convolutional Neural Network* (CNN). Een CNN valt onder de noemer van *Artificial Neural Network* (ANN). Een ANN kan best omschreven worden als een verzameling van algoritmes die samenwerken om een hoeveelheid complexe data te verwerken. In het geval van een CNN is deze complexe data een set geannoteerde afbeeldingen, waarvan elke koppel afbeeldingen, annotatie vervolgens geanalyseerd wordt.

In een eerste fase van deze masterproef werd het CNN getraind met als einddoel het classificeren van afbeeldingen, in een tweede fase werd overgegaan naar regressie. Deze overgang wordt verderop in deze paper in meer detail behandeld.

De afbeeldingen werden afgeleid uit de videobeelden door periodiek een frame van deze videobeelden op te slaan als afbeelding. Deze afbeeldingen werd hierna als input gebruikt voor het CNN, waarna het CNN aangaf welke wielerploeg(en) op dat moment zichtbaar was/waren.

Trefwoorden die het onderwerp omschrijven : *artificial intelligence, machine learning, deep learning, image classification*

Inhoudstafel

Lijst van figuren

Lijst van tabellen

Afkortingen

AI = Artificial Intelligence

ANN = Artificial Neural Network

CNN = Convolutional Neural Network

API = Application Programming Interface

ReLu = Rectified Linear Unit

Hoofdstuk 1 : Inleiding

1.1 Kadering masterproef

De wielersport is niet meer wat het ooit geweest is. Tijden waarin vedetten zoals Eddy Merckx, Fausto Coppi en Bernard Hinault op handen werden gedragen zijn gepasseerd. Uiteraard zijn er nog steeds hedendaagse supersterren zoals Tom Boonen of Chris Froome, maar deze lijken hun voorgangers niet te evenaren in roem en bekendheid.

Dit verlies in populariteit heeft verschillende gevolgen. Jongere generaties lijken interesse te hebben verloren in de sport, de renners worden aangemoedigd door minder en minder fans en het wielertoerisme daalt sterk in populariteit.

Een ander, minder opgemerkt, gevolg is het saaie, ouderwetse label dat tegenwoordig wordt toegewezen aan de verschillende wielermusea. Deze wielermusea bezitten vaak een indrukwekkende collectie aan wielermemorabilia zoals trofeeën of handgetekend materieel, maar vinden het moeilijk om het publiek warm te maken voor deze grote verzameling aan wielerhistoriek.

Wat als een wielermuseum zijn indrukwekkende collectie op een modernere manier zou kunnen tentoonstellen? Wielermuseum KOERS te Roeselare had net dit idee. In samenwerking met IDLab Ghent werd een ambitieus project gestart, genaamd ‘Museum in de Living : Etappe 2’. Het doel van dit project was om de kijker tijdens het verloop van de koers relevantie collectiestukken te tonen, gerelateerd aan de positie waarop de renners zich op dat moment bevonden. Het tonen van deze collectiestukken gebeurt via een live *webfeed* die gehost wordt op de website van KOERS zelf. Deze kan dan bekeken worden door de kijker via een smartphone, laptop of tablet.

Het project zelf werd opgesplitst in 4 fasen. De eerste fase bestond uit het voorzien van een *geotag* voor elk collectiestuk, zodat elk collectiestuk gelinkt kon worden aan een bepaalde locatie. In de tweede fase werden deze geotags gebruikt om op basis van de huidige locatie van de kopgroep van een bepaalde wielerwedstrijd een relevant collectiestuk te tonen. De derde fase bestond uit het optimaliseren van de tweede fase, aan de hand van een ‘*recommender engine’*. Deze filterde op basis van het gedrag van de eindgebruiker de collectie aan wielererfgoed zodat deze beter afgestemd was op de interesses van de eindgebruiker. In de vierde fase werd de webfeed zelf ontworpen, om zo de collectiestukken te tonen aan de eindgebruiker.

1.2 Probleemstelling masterproef

Deze masterproef kadert in het project ‘Museum in de Living : Etappe 2’, meer specifiek als een deel van fase 2. Na fase 1 was het reeds mogelijk om aan een bepaalde locatie een collectiestuk te linken. Dit collectiestuk mocht echter alleen getoond worden indien deze locatie de locatie van de kopgroep was. De locatie van de kopgroep is relatief eenvoudig af te leiden uit de kilometerindicator die steeds te zien is tijdens een televisie-uitzending. Dit volstond echter niet, aangezien de kopgroep ook effectief in beeld moest worden gebracht alvorens dit collectiestuk getoond kon worden.

Er moest met andere woorden een methode gevonden worden om te detecteren of de renners die zichtbaar waren, de renners waren die zich in de kopgroep bevonden.

1.3 Doelstelling masterproef

Deze masterproef beschrijft hoe renner herkenning op basis van wielertenue werd geïmplementeerd. Er werd gebruik gemaakt van een *Convolutional Neural Network* (CNN), een vorm van *Artificial Intelligence* (AI). Het gebruik van een CNN maakte het mogelijk om op basis van de videobeelden van een uitzending te herkennen welke wielerploegen op dat ogenblik in beeld waren.

Indien gekend was wat de verdeling van de kopgroep was, met andere woorden hoeveel renners van elke wielerploeg aanwezig waren, kon deze informatie gecombineerd worden om zo de kopgroep te identificeren.

Hieronder volgt een korte inleiding tot AI, waarna het verloop van de masterproef zelf wordt beschreven.

1.4 Inhoud masterproef

In recente tijden is AI waarschijnlijk het meest besproken onderwerp in de wereld van de informatica. Toch wordt het vaak verward met termen zoals *machine learning* of *deep learning*. Een korte introductie is vereist.

1.4.1 Artificiële intelligentie

Een programma valt onder de tak van AI indien het geen vaste regels meekrijgt, maar zelf verwacht wordt te leren wat die regels zijn. Neem bijvoorbeeld een programma dat 2 getallen optelt. Indien dit programma geen regels meekrijgt die het vertellen hoe 2 getallen op te tellen, maar een groot aantal voorbeelden (2+2=4, 5+6=11, …) en hieruit moet afleiden hoe 2 getallen op te tellen, valt het onder de tak van AI.
AI kan dus zeer eenvoudig zijn.

1.4.2 Machine learning

Machine learning is niets meer dan een manier om AI te implementeren, het is een vorm van AI. In het geval van machine learning noemt men het programma ook wel een model. Per definitie krijgt dat model een set voorbeelden en is de doelstelling hier patronen in te herkennen.

Een model wordt altijd getraind, gevalideerd en getest aan de hand van respectievelijk een *training set*, *validation set* en *test set*. Elke set bevat geannoteerde data. Geannoteerde data is data waarvoor elke input kan gelinkt worden aan de gewenste output. Het model zelf weet echter niet altijd wat die verwachte output is, een meer gedetailleerde uitleg volgt hieronder.

De eerste stap in het ontwikkelen van een model is steeds het definiëren van de architectuur van dit model. Er bestaan verschillende soorten modellen, elk gekenmerkt door hun voor- en nadelen. Welk soort model wordt gebruikt hangt sterk af van het soort probleem. Ongeacht de gekozen architectuur van het model of de dataset van de welke de data uitmaakt, wordt de data steeds op dezelfde manier aan het model gepresenteerd.

Een dataset bestaat steeds uit n dataelementen of *samples*. Een dataset wordt vervolgens steeds opgedeeld in een aantal *batches*, gedefinieerd aan de hand van de *batch size*. Hierbij bevat 1 batch minimaal 1 en maximaal n samples.

Neem een dataset die bijvoorbeeld 200 samples bevat. Indien de batch size 10 bedraagt, kan deze dataset opgedeeld worden in 20 batches van 10 samples. Indien de batch size 7 zou bedragen, zou de dataset opgedeeld worden in 29 batches. De eerste 28 batches tellen dan 7 samples, de laatste batch telt 4 samples.

Een dataset wordt batch per batch gepresenteerd aan het model tot de volledige dataset doorlopen is. De mogelijkheid bestaat dat dit proces meerdere malen herhaald wordt, waarbij de dataset meerdere malen als input voor het model wordt gebruikt. Dit wordt gedefinieerd door het aantal epochs. Indien het aantal epochs bijvoorbeeld 5 bedraagt, zal een dataset 5 maal het volledige model doorlopen als input.

Na het vastleggen van de architectuur van een model volgt de *training fase* of *learning fase.* In deze fase wordt de training set als input voor het model gebruikt. Het model kent de verwachte output en heeft als doel het zoeken van een verband tussen deze input en verwachte output. Tijdens het verwerken van de input worden de modelparameters constant aangepast. Op deze manier tracht het model de complexe relatie tussen input en verwachte output na te bootsen in volgende fases.

De eerste van die volgende fases is de *validation fase*. In deze fase wordt de validation set gebruikt als input voor het model. In tegenstelling tot de vorige fase kent het model hier niet de verwachte output. Het model wordt verwacht zelf voorspellingen te maken naar de verwachte output, op basis van wat het geleerd heeft in de training fase. De nauwkeurigheid van deze voorspellingen worden weergegeven door de *validation accuracy*.

Hiernaast wordt de validation fase ook gebruikt om de hyperparameters van het model te configureren. Een hyperparameter is een parameter waarvan de waarde vooraf wordt ingesteld. Dit verschilt dus van de parameters die tijdens de learning fase werden aangepast. Hyperparameters kunnen ook aangepast worden, maar dan gebeurt dit steeds vooraf en indien blijkt dat het model kwalitatief niet goed genoeg is. Hiernaast mag de data ook niet als oorzaak gezien worden van dit tegenvallend resultaat, want dan kan het veranderen van de dataset de kwaliteit van het model verbeteren. Een goed voorbeeld van een hyperparameter is het aantal epochs. Indien opvalt dat de validation accuracy te laag ligt, kan men er voor kiezen om het aantal epochs te verhogen. Op deze manier krijgt het model meer tijd om patronen te ontdekken tijdens de training fase.

De test set ten laatste wordt gebruikt om een objectieve maat te krijgen voor de kwaliteit van het model. Vaak wordt deze gebruikt om een onderscheid te maken tussen verschillende modellen en hieruit de beste te selecteren. Merk op dat elk model de test set maar één maal te zien krijgt. Een model kan echter meerdere malen dezelfde of lichte variaties van de training en validation set doorlopen. Pas als gevalideerd is dat het model goed getraind is, krijgt het model de test set te zien. Deze laatste fase wordt ook wel de *testing fase* genoemd.

1.4.3 Deep learning

Deep learning is een subset van machine learning. In dit geval heeft het model de vorm van een *Artificial Neural Network* (ANN). Een ANN tracht de biologische werking van een brein na te bootsen.

Een ANN bestaat uit *nodes*, die elk een neuron voorstellen. De verbindingen tussen de neuronen in een brein worden vertaald in het doorsturen van data van de ene node naar de andere. Een meer gedetailleerde uitleg over de architectuur en werking van een ANN volgt in volgend hoofdstuk.

Zoals reeds vermeld in de inleiding werd doorheen het verloop van deze masterproef gebruik gemaakt van een CNN. Een CNN is een soort ANN dat geoptimaliseerd is voor het verwerken van afbeeldingen.

Hoofdstuk 2 : Convolutional Neural Network

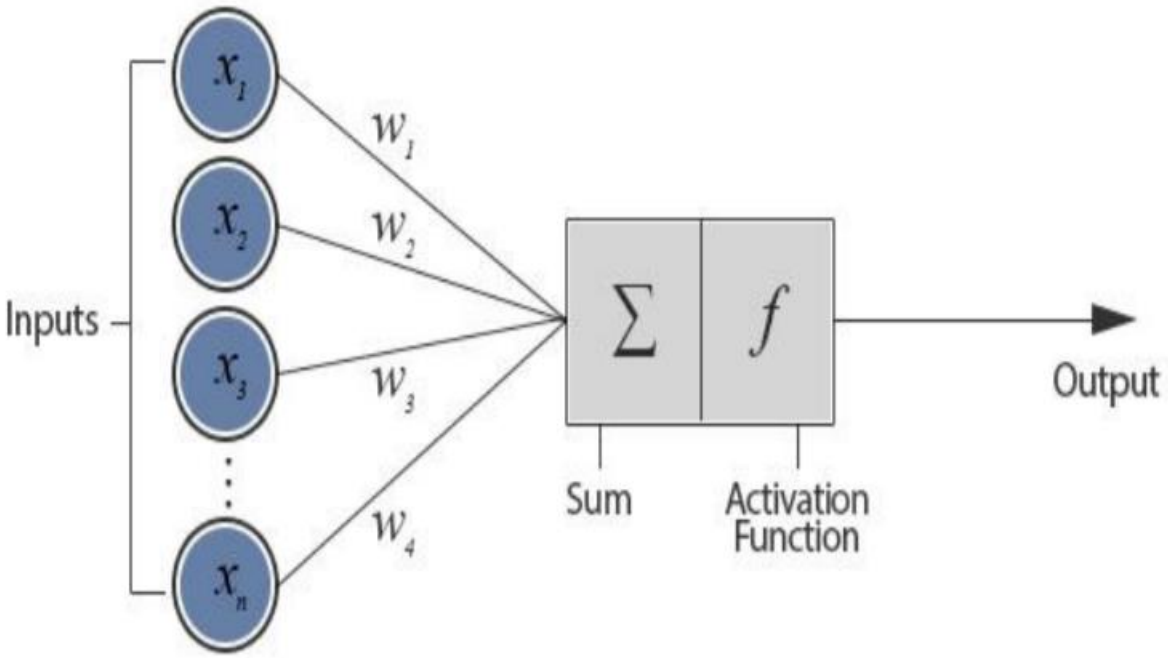
2.1 Algemene architectuur brein

Het menselijke brein is een complexe architectuur van nauw verweven neuronen, onderling verbonden via synapsen. De vele elektrische impulsen worden via deze synapsen van neuron naar neuron gestuurd, om op deze manier door elke neuron verwerkt te worden alvorens eventueel opnieuw doorgestuurd te worden. Een impuls wordt namelijk alleen doorgestuurd indien de neuron geactiveerd werd. Deze activatie, of het gebrek daaraan, is het resultaat van een chemisch proces binnenin de neuron zelf.

2.2 Algemene architectuur ANN

Zoals reeds vermeld bestaat een ANN uit een groot aantal nodes, waarbij elke node de functie van een neuron in het menselijke brein nabootst. Net zoals neuronen in het brein zijn de nodes in een ANN ook zeer nauw verweven. Naar analogie met de neuron die een impuls verwerkt, verwerkt een neuron data [1]. Elke neuron heeft de mogelijkheid om data te ontvangen, verwerken en doorsturen.

Dit proces van ontvangen, verwerken en doorsturen van data wordt hieronder, aan de hand van figuur 1, meer in detail uitgelegd.



Figuur 1 : werking node

De data die een node ontvangt kan gezien worden als een verzameling van n inputs x_n die elk langs een eigen verbinding binnenkomen. Elke input, of in het algemeen elke verbinding, draagt een bepaald gewicht w_n. Dit gewicht toont het belang van deze inkomende verbinding tegenover de andere inkomende verbindingen. Het verwerken van de data, of inputs, die een node ontvangt gebeurt steeds in twee stappen.

In een eerste stap wordt de som genomen over alle inputs, vermenigvuldigd met het gewicht van de verbinding langs de welke de input is binnengekomen in de node. Dit is weergegeven in onderstaande formule.

$$Som = \sum_0^i x_j * w_j$$

In een tweede stap wordt het resultaat van deze som doorgegeven aan een activatiefunctie. Wat is nu het nut van deze activatiefunctie? Zoals al eerder vermeld tracht een ANN de werking van een brein na te bootsen. Dat geldt ook voor de manier waarop informatie doorheen het brein passeert. Indien een neuron in het brein informatie ontvangt, is er nooit een garantie dat deze informatie doorgegeven zal worden. Het is pas wanneer deze informatie als nuttig wordt ervaren dat het neuron deze informatie zal doorgegeven aan een ander neuron. Op dat moment is de neuron geactiveerd.

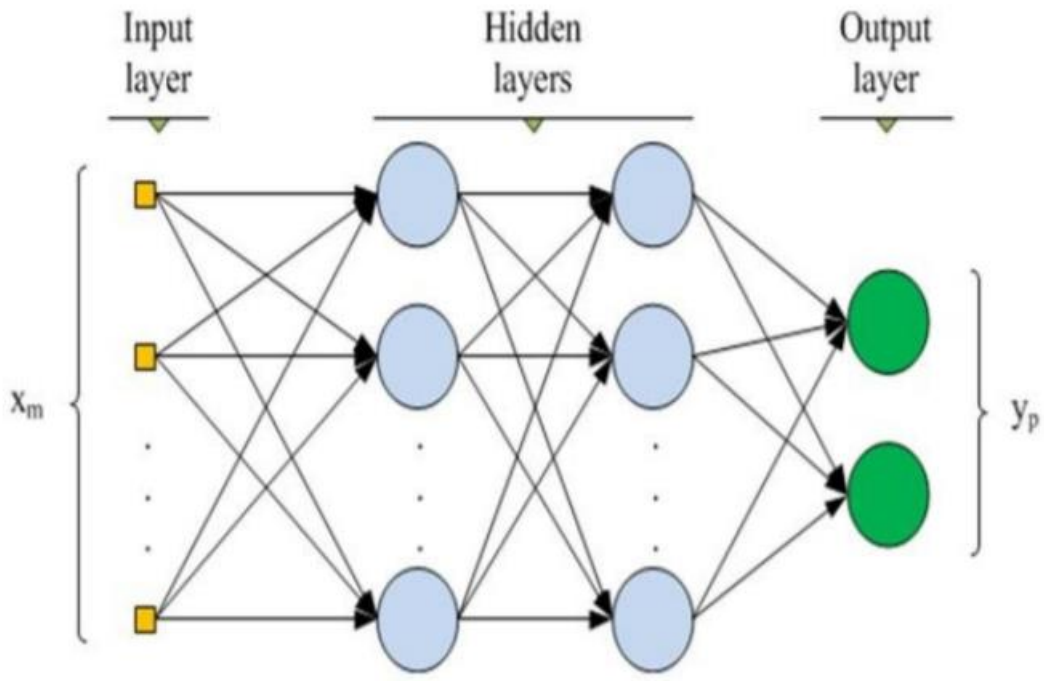
Een node heeft uiteraard geen chemisch proces die beslist of de informatie die het ontvangen heeft nuttig is of niet. Deze functionaliteit wordt voorzien door de activatiefunctie. De activatiefunctie moet op zich niet zeer complex zijn. Één van de eerste activatiefuncties die ooit ontworpen werd, was zeer eenvoudig en werkte als volgt. De activatiefunctie ontvangt één parameter, namelijk de sommatie zoals uitgerekend volgens bovenstaande formule. Indien het resultaat van deze som groter is dan een bepaalde vooraf ingestelde grenswaarde is de node geactiveerd.

Een goed voorbeeld van een activatiefunctie die in de praktijk vaak wordt gebruikt is de ReLu (Rectified Linear Unit) activatiefunctie, de formule wordt hieronder getoond.

$$f(x) = \max(x, 0)$$

Zoals de formule toont is de grenswaarde in dit geval 0, alle positieve waarden zullen een node activeren.

Zoals reeds vermeld bestaat uit het brein uit een nauw verweven architectuur van miljoenen neuronen en tracht een ANN deze structuur na te bootsen via een nauw verweven architectuur van nodes. Een ANN kan echter ook omschreven worden als een opeenvolging van verschillende *layers* of lagen, waarbij aan elke layer nodes worden toegewezen.



Figuur 2 : typische architectuur ANN

In zijn meest eenvoudige vorm bestaat een ANN uit 3 layers of lagen, zoals weergegeven in figuur 2. De *input layer* die de input accepteert, de *hidden layer* die de input verwerkt en de *output layer* die de output presenteert. In de werkelijkheid bestaat deze hidden layer meestal uit verschillende layers, die opeenvolgende transformaties uitvoeren op de ontvangen data. Dit is ook het geval in bovenstaande afbeelding, waarbij twee hidden layers zichtbaar zijn [2].

De nodes hebben een andere functie naargelang welke layer ze toe behoren. *Input nodes* worden gebruikt om informatie van buitenaf door te geven aan het model en manipuleren deze data meestal niet, tenzij een conversie moet gebeuren. *Output nodes* presenteren de informatie aan de buitenwereld. Eventueel gebeuren in deze fase ook nog een klein aantal manipulaties op de data, maar deze zijn dan eerder gerelateerd aan de vorm van die data dan aan de inhoud ervan.

Hidden nodes manipuleren de data wel. De manier waarop deze gemanipuleerd wordt hangt af van de hidden layer waartoe de hidden node behoort. Ook de activatiefunctie die wordt toegepast kan verschillen van hidden layer tot hidden layer. Dit wordt in meer detail behandeld in sectie 2.3 .

2.3 Algemene architectuur CNN

Een CNN is een soort ANN geoptimaliseerd voor het verwerken van afbeeldingen. In wat volgt wordt de structuur van een typische CNN uitgelegd door de functionaliteit van de opeenvolgende layers gedetailleerd uit te leggen. Merk op dat de opeenvolging van layers zoals deze hieronder worden beschreven niet noodzakelijk overeenkomt met de opeenvolging van layers in de CNN die werd toegepast in het kader van deze masterproef. Dit is het gevolg van het feit dat de architectuur van een model een grote impact heeft op de kwaliteit ervan en ook zeer probleem specifiek gebonden is.

De eerste layer, die ook steeds aanwezig moet zijn, is de input layer. Deze layer is verantwoordelijk voor het manipuleren van de ontvangen data op een manier dat deze als input kan worden gebruikt voor de volgende layer. In de context van deze masterproef houdt dat in dat een batch van afbeeldingen wordt omgezet in een array, waarin elk element een afbeelding voorstelt als een 3 dimensionale array. Een voorbeeld wordt hieronder in figuur 3 getoond.

```
[[[178 177 185]   [[176 175 183]    [[175 174 182]
  [176 175 183]    [174 173 181]     [172 171 179]
  [174 173 181]    [172 171 179]     [170 169 177]
  ...              ...               ...
  [182 185 192]    [177 180 187]     [173 173 181]
  [184 187 194]    [179 182 189]     [175 175 183]
  [185 188 195]]   [180 183 190]]    [176 176 184]]
                      ...
[[ 40  26  15]   [[ 31  17   8]    [[ 30  16   7]
 [ 58  40  30]    [ 44  26  16]     [ 36  19  11]
 [ 59  32  21]    [ 41  17   7]     [ 33   8   1]
 ...              ...               ...
 [116  71  66]    [100  57  51]     [ 93  52  46]
 [106  61  56]    [ 97  54  48]     [ 93  52  46]
 [104  59  56]]   [ 96  53  47]]    [ 92  51  45]]]
```

Figuur 3 : afbeelding en bijhorende RGB matrix

De dimensies van deze array zijn steeds breedte afbeelding x hoogte afbeelding x 3. Dit is het gevolg van het feit dat de afbeelding per pixel 3 waarden bevat, de 3 RGB waarden (rood, groen en blauw). De derde waarde, in dit geval 3, wordt ook wel de diepte van de array of matrix genoemd. Een array zoals hierboven getoond wordt ook wel een RGB matrix genoemd.

2.3.1 Convolutional layer

De layer die hierop volgt is de *convolution* of *convolutional layer.* Deze layer vormt de kern van het CNN en voert de meeste berekeningen uit. De convolutional layer voert op verschillende delen van de afbeelding een convolutie uit, wat hieronder in meer detail wordt uitgelegd.

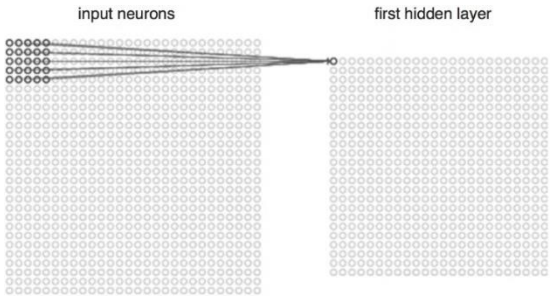
De convolutie is een wiskunde berekening, gegeven door onderstaande formule.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

Eenvoudig verwoord is de convolutie een operatie waarbij het resultaat een functie is die aangeeft hoe één functie een andere functie transformeert. Opdat een convolutie dus kan uitgevoerd worden zijn twee functies nodig, de functie die getransformeerd wordt en de functie die de transformatie uitvoert.

De convolutie-operatie die in een CNN wordt toegepast is gebaseerd op bovenstaande formule, maar niet dezelfde. Zo ontvangt de convolutie-operatie als input niet twee functies, maar twee matrices. De ene matrix is de RGB matrix van een afbeelding, de andere is de filter of kernel. Deze filter of kernel is qua dimensies veel kleiner, maar heeft wel dezelfde diepte. Een realistische filter zou bijvoorbeeld de dimensies 5 x 5 x 3 hebben.

Deze filter wordt vervolgens verplaatst over de afbeelding. In het begin bevindt deze filter zich bijvoorbeeld links bovenaan op de afbeelding. Het overlapt als het ware met de 5 x 5 x 3 deelmatrix die links bovenaan is te vinden in de RGB matrix van de oorspronkelijke afbeelding. Dit wordt ook schematisch weergegeven in figuur 4.



Figuur 4 : werking convolutie

Tijdens deze overlapping wordt een convolutie uitgevoerd. Het resultaat van deze convolutie is de eerste waarde van een resultaatmatrix. Na het uitvoeren van deze convolutie wordt de filter één pixel opgeschoven in een arbitraire richting, dit wordt ook wel een *stride* genoemd. Meestal wordt de filter eerst volledig horizontaal opgeschoven en wordt dit proces herhaald voor de volledige hoogte van de afbeelding, maar in theorie maakt dit niet uit.

Uiteindelijk is de filter over de volledige matrix van de afbeelding verplaatst en is voor elke mogelijke combinatie van filter met deelmatrix van de RGB matrix een convolutie uitgevoerd. Het resultaat van deze convolutie is steeds bijgehouden, waarbij al deze resultaten nu een nieuwe resultaatmatrix vormen [3].

De dimensie van de resultaatmatrix is steeds (breedte matrix – breedte filter + 1) x (hoogte matrix – hoogte filter + 1) x 3. Uiteraard hangt de manier waarop de resultaatmatrix moet geïnterpreteerd worden sterk af van de soort filter die werd gebruikt.

Zoals eerder uitgelegd is een filter een relatief kleine matrix. Deze matrix, alhoewel niet direct afgeleid van een afbeelding, kan uiteraard wel geïnterpreteerd worden als een afbeelding. Een matrix van een filter die alleen op de hoofddiagonaal waarden bevat, zou perfect een schuine lijn kunnen voorstellen.

Stel nu dat een convolutie wordt uitgevoerd met deze laatstgenoemde filter over een bepaalde afbeelding. Indien de resultaatmatrix van deze convolutie op bepaalde plaatsen relatief hoge waarden bevat, is de kans groot dat op die plaatsen een schuine lijn te vinden is in de afbeelding. Op deze manier is het mogelijk om in afbeeldingen bepaalde vormen te detecteren.

Merk op dat tijdens het trainen van een model deze filter initieel geen specifieke vormen zal zoeken, aangezien het model nog geen idee heeft welke patronen voorkomen in de dataset. Initieel zal de filter dus opgevuld worden met willekeurig verspreide waarden. Geleidelijk aan zal het model echte leren welke vormen het vaakst voorkomen in de set van afbeeldingen.

Op basis hiervan zal de filter anders opgevuld worden om het ontdekken van deze vormen of patronen ten goede te komen. Uiteraard heeft de grootte van de filter ook een effect op de snelheid en kwaliteit van deze operatie, wat ook verderop in deze masterproef zal blijken.

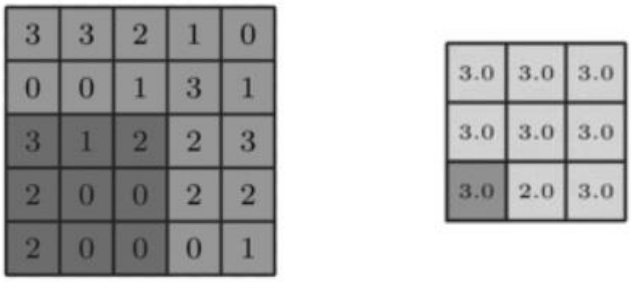
2.3.2 Pooling layer

Na elke convolutional layer volgt steeds een *activation layer*. De meest gebruikte activation layer is de ReLu layer, welke de ReLu activatiefunctie toepast op de ontvangen data.

Hierna volgt een *pooling layer*. Een pooling layer wordt gebruikt om de resultaatmatrix te verkleinen. Dit heeft twee voordelen. Enerzijds vermindert hierdoor de hoeveelheid data die bijgehouden moet worden voor volgende fases. Anderzijds gaat dit *overfitting* tegen.

Overfitting is het fenomeen waarbij een model te veel patronen heeft herkend in een bepaalde dataset en toekomstige voorspellingen hierdoor niet langer accuraat zijn. Door het verminderen van de grootte van de matrix verkleint de kans dat verborgen patronen ontdekt worden, aangezien alleen bepaalde waarden worden bijgehouden.

Welke waarden worden bijgehouden hangt af van het soort pooling layer dat gebruikt wordt, een *average pooling layer* of *max pooling layer*. Een average pooling layer verkleint de resultaatmatrix door een gebied te reduceren tot het gemiddelde van alle waarden in dat gebied. Een max pooling layer verkleint een resultaatmatrix door een gebied te vervangen door de maximale waarde die in dat gebied voorkwam.



Figuur 5 : overzicht pooling

Een voorbeeld van een max pooling layer is hierboven in figuur 5 gegeven. In dit voorbeeld wordt de oorspronkelijke 5x5 matrix via pooling gereduceerd tot een 3x3 matrix. Om dit te bereiken wordt de 5x5 beginmatrix gebied per gebied overlopen, waarbij elk gebied een 3x3 matrix is. In deze 3x3 matrix wordt de hoogste waarde gezocht en in een nieuwe cel geplaatst in de resultaatmatrix. Merk op dat de positie van de maximale waarde in de eindmatrix uiteraard overeenkomt met de relatieve positie van het gebied in de beginmatrix.

In de praktijk wordt de max pooling layer het vaakst gebruikt. Zoals reeds vermeld zullen alleen de hoogste waarden worden behouden indien *max pooling* wordt toegepast. Met andere woorden alleen de dominante *features* van de afbeelding zullen worden behouden. Dit biedt het voordeel dat indien minder dominante features gedetecteerd worden deze geen invloed hebben op het leerproces dat volgt. Indien echter *average pooling* wordt toegepast worden ook deze minder dominante features in rekening gebracht, wat negatieve gevolgen kan hebben op het leerproces.

2.3.3 Dropout layer

Het is mogelijk, maar niet verplicht, dat op een pooling layer een *dropout layer* volgt. Het nut van deze dropout layer is het verder verkleinen van de kans op overfitting. Alhoewel overfitting reeds tegengegaan wordt door het gebruik van een pooling layer bestaat de kans dat overfitting nog steeds plaatsvindt.

Zo zal een model dat een relatief kleine training set tot zijn beschikking heeft hoogstwaarschijnlijk patronen ontdekken in deze training set die eigen zijn aan die specifieke verzameling data zelf en niet aan de grotere verzameling van data in het algemeen.

Bovenstaand probleem van een tekort aan data kan verholpen worden via *data augmentation*. In dit geval wordt elke afbeelding geroteerd, bijgesneden of vervormd om zo een grotere dataset te bekomen. Dit wordt ook toegepast in deze masterproef en wordt verder in deze paper in meer detail vermeld. Data augmentation is echter niet altijd een optie. Zo zal een model dat getraind wordt voor nummerherkenning na data augmentation zeer veel moeite hebben om de cijfers 6 en 9 van elkaar te onderscheiden.

Een dropout layer kan echter steeds toegepast worden. Het principe achter een dropout layer zit verscholen in de naam. Een dropout layer zal een willekeurig aantal nodes van de eigen layer op gegeven momenten uitschakelen. Deze gedeactiveerde nodes zullen met andere woorden op dit moment niet kunnen leren van de data die op dat moment passeert, en dus ook niet kunnen communiceren met de layer die op de dropout layer volgt.

Stel dat een dropout layer tussen twee layers geplaatst wordt, respectievelijk genaamd L1 en L2. L2 is zich niet bewust van de aanwezigheid van de dropout layer en heeft de illusie dat het data ontvangt van L1. Indien de dropout layer bepaalde nodes deactiveert, verandert de perceptie die L2 heeft van L1, aangezien in het opzicht van L2 de output van de dropout layer de output van L1 is. Indien dit proces meerdere malen herhaald wordt, zal L2 steeds opnieuw data ontvangen van een aangepaste vorm van L1. Dit zorgt ervoor dat L2 telkens opnieuw gedwongen wordt de data anders te verwerken, waardoor de kans verkleint dat L2 steeds opnieuw dezelfde patronen ontdekt. Op deze manier wordt overfitting tegengegaan [4].

2.3.4 Fully connected layer

Ten laatste volgt een *fully connected layer* of *dense layer*. Deze layer verwacht de input data steeds ééndimensionaal. De output data van de bovenstaande layers is vaak meerdimensionaal en zeer algemeen. Zo zal de output data van een pooling layer bijvoorbeeld aangeven welke zones van een afbeelding met de hoogste waarschijnlijkheid een schuine lijn bevatten, maar opdat deze informatie kan gecombineerd worden met opeenvolgende layers, moet deze wel gecomprimeerd worden tot de essentie.

Om dit te bereiken wordt tussen een pooling layer en dense layer vaak een *flatten layer* geplaatst. Deze flatten layer transformeert de meerdimensionale matrix die de pooling layer presenteert in een ééndimensionale gegevensstructuur die de essentie van de data bevat.

Hoofdstuk 3 : Opstellen dataset

Het opstellen van de dataset moet steeds gebeuren met één doel voor ogen : een zo hoog mogelijke nauwkeurigheid verkrijgen wanneer het model toegepast wordt in real time situaties.

Zoals reeds vermeld in het abstract is het doel van deze masterproef het trainen van een model dat gebruikt kan worden om afbeeldingen te analyseren en hierin renners te herkennen op basis van de wielerploeg waartoe ze behoren. Dit in gedachten houdende betekent het opstellen van een goede dataset in het kader van deze masterproef het volgende.

De dataset dient afbeeldingen te bevatten van een voldoende kwaliteit opdat het model er in slaagt in een willekeurig frame van een video-uitzending aanwezige renners in te delen volgens wielerploeg.

In een eerste fase van het project werden twee beperkingen opgelegd, om zo sneller tot een werkende versie te komen.

Ten eerste werd besloten om alleen te focussen op het tenue van de renner en niet op het volgnummer. Dit heeft als gevolg dat in de eerste fase van het project alleen maar voorspellingen worden gemaakt over de ploeg waartoe de renner behoort en niet de identiteit van de renner zelf. Indien het volgnummer en de startlijst van een wedstrijd namelijk gekend zijn, is het eenvoudig om het volgnummer te linken aan de identiteit van een bepaalde renner.

Ten tweede werd ook besloten om de dataset op te vullen met foto's van renners die uitkomen voor een team in de UCI World Tour. Het wielrennen kent namelijk twee internationale wielercompetities : de UCI World Tour en de UCI Continental Tour. De UCI World Tour is de meest prestigieuze competitie van de twee en het best gekend bij de alledaagse wielerfan. Deze UCI World Tour kent 18 ploegen (AG2R La Mondiale, Astana Pro Team, Bahrain-Merida, Bora-Hansgrohe, CCC Team, Deceuninck – Quick Step, EF Education First Pro Cycling, Groupama-FDJ, Lotto Soudal, Movistar Team, Mitchelton-Scott, Team Dimension Data, Team Ineos, Team Jumbo-Visma, Team Katusha-Alpecin, Team Sunweb, Trek-Segafredo, UAE Team Emirates). Indien een bepaalde voorspelling wordt gedaan naar de wielerploeg van een renner, zal deze dus steeds één van deze 18 ploegen zijn.

Verschillende technieken werden doorheen het verloop van de masterproef toegepast om afbeeldingen op te halen, dit varieerde van het aanspreken van een API tot het *scrapen* van een HTML pagina. Deze technieken worden in meer detail toegelicht daar waar deze zijn toegepast. Indien nodig zal een codevoorbeeld meer uitleg verschaffen, deze zijn steeds terug te vinden in de appendix.

Hoofdstuk 4 : Classificatie

Zoals reeds aangehaald in de inleiding werd in een eerste fase van deze masterproef besloten om afbeeldingen te classificeren. Het classificeren van deze afbeeldingen gebeurde aan de hand van 18 klassen, namelijk de 18 UCI World Tour ploegen zoals reeds vermeld in hoofdstuk 3.

4.1 Ontwerp basismodel

Om het model op te bouwen werd gebruik gemaakt van Keras. Keras wordt vaak omschreven als een deep learning *library* gescheven in Python, terwijl het eigenlijk een Python *high level interface* is die 4 verschillende deep learning libraries kan aanspreken : Tensorflow, Microsoft Cognitive Toolkit, Theano en PlaidML. Dit wordt schematisch weergegeven in figuur 6.

Figuur 6 : architectuur keras

Tijdens het verloop van deze masterproef werd gekozen om Tensorflow als deep learning library te gebruiken en deze dus steeds aan te spreken via Keras. Het aanspreken van Tensorflow via Keras in tegenstelling tot Tensorflow direct aanspreken biedt het voordeel dat het meer gebruiksvriendelijk is.

Codevoorbeeld 1 in de appendix toont hoe de architectuur van het model werd opgebouwd via Keras. Het model was een sequentieel model. Dit hield in dat het model laag per laag opgebouwd kon worden wat het voordeel bood dat het eenvoudiger aan te passen is. Merk op dat ook de hyperparameters reeds ingesteld werden.

In een eerste fase werden 3 layers toegevoegd aan het model. De eerste layer die werd toegevoegd was een convolutional layer, waarbij ook de 'input_shape' werd meegegeven als argument. Merk op dat het meegeven van dit argument 'input_shape' impliciet wijst op de aanwezigheid van de input layer. Alhoewel deze dus niet expliciet vermeld werd, was deze wel degelijk aanwezig.

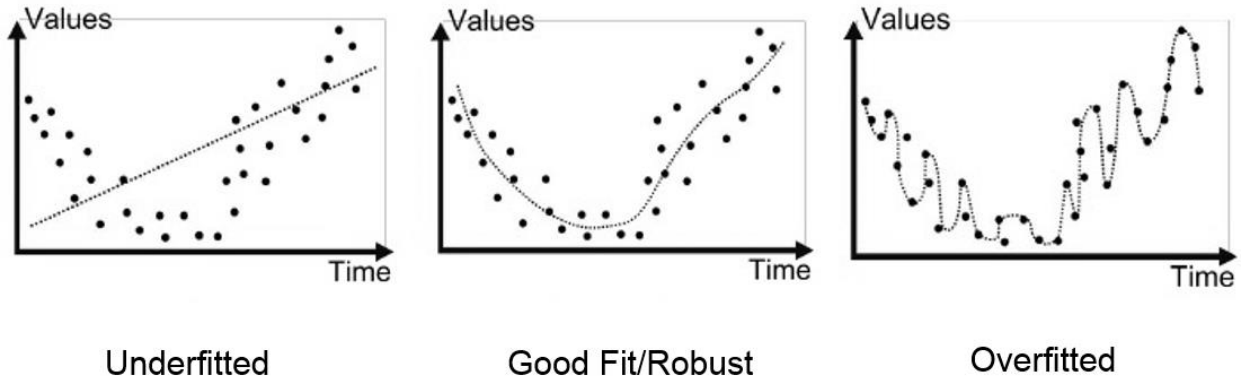
Vervolgens werd een activation layer toegevoegd, meer specifiek voor de ReLu activatiefunctie. Zoals reeds vermeld in de inleiding evalueert de ReLu activatiefunctie de input die het ontvangt en vervangt het elk negatief element met 0.

Hierna werd een pooling layer toegevoegd. Deze pooling layer voerde max pooling uit, waarbij de grootte van het gebied een 2x2 matrix was.

Deze opeenvolging van 3 layers werd nog 2 maal herhaald. Op dit punt zou het model beschreven kunnen worden als een opeenvolging van 3 bijna identieke modules. Niet volledig identiek aangezien in de eerste module de convolutional layer voorafgegaan werd door een input layer.

Om de output te presenteren moet een fully connected of dense layer toegevoegd worden. Zoals reeds besproken dient een dense layer voorafgegaan te worden door een flatten layer om de meerdimensionale matrix te converteren in een ééndimensionale gegevensstructuur . Als gevolg hiervan werd eerst een flatten layer toegevoegd, waarna de dense layer zelf werd toegevoegd. Het argument dat meegegeven werd aan de dense layer, in dit geval 64, dient op het aantal nodes dat de dense layer tot zijn beschikking heeft.

Zoals het model nu geconfigureerd was, voldeed het aan alle vereisten waaraan een CNN moet voldoen, maar toch werd besloten om onder meer nog een dense layer toe te voegen. De reden hiervoor was dezelfde reden waarom meerdere convolutional, activation en pooling layers werden toegevoegd : om *underfitting* te voorkomen. Underfitting komt voor indien een model niet genoeg patronen in de data heeft ontdekt en als gevolg inaccurate voorspelling maakt indien nieuwe data wordt gepresenteerd. Underfitting is als het ware het tegenovergestelde van overfitting, alhoewel beide leiden tot inaccurate voorspellingen. Om verwarring te vermijden toont figuur 7 de relatie tussen under- en overfitting.



Figuur 7 : underfitting , overfitting

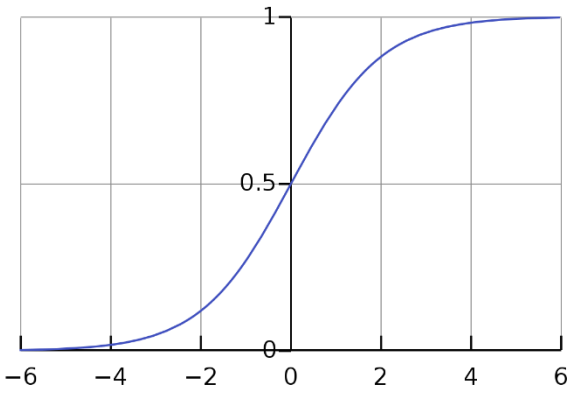
Zoals te zien is in de figuur zal underfitting ervoor zorgen dat het model niet genoeg patronen ontdekt in de dataset. Om dit te voorkomen werd een tweede dense layer toegevoegd.

Merk op dat ditmaal deze dense layer maar één node bevatte en niet voorafgegaan werd door een flatten layer, maar door een activation en dropout layer. Enerzijds was er geen nood om een nieuwe flatten layer toe te voegen aangezien de vorige flatten layer de data reeds had gereduceerd tot een ééndimensionale gegevensstructuur. Anderzijds verzekerde het toevoegen van de activation layer de afwezigheid van negatieve waarden en verzekerde de aanwezigheid van de dropout layer een kleinere kans op overfitting.

Aangezien het model getraind werd om afbeeldingen te classificeren had de laatste dense layer maar nood aan één node. De output van deze node was dan de klasse tot de welke de afbeelding behoorde.

Als laatste volgde een nieuwe activation layer. Deze gebruikt wel een andere activatiefunctie, namelijk de sigmoid functie. De functie zelf en het verloop ervan worden in figuur 8 getoond.

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



Figuur 8 : sigmoid functie en verloop

De reden waarom de sigmoid functie hier werd gebruikt is omdat het bereik van deze functie steeds tussen 0 en 1 ligt. Omwille hiervan is het geschikt om de probabiliteit van een voorspelling voor te stellen [5].

De laatste stap was het compileren van het model. Het compileren van het model werd gebruikt voor de initialisatie van verschillende parameters. Na compileren van het model telde het model 1 input layer, 15 hidden layers en 1 output layer.

4.2 Ophalen data

In een eerste poging om een dataset op te stellen werd gebruik gemaakt van de Application Programming Interface (API) van Flickr, alsook van een script geleverd door ir. Jelle De Bock.

4.2.1 Opstellen training set

In een eerste poging om een training set op te stellen, werd de portretfoto van elke renner van elk team behorend tot de UCI World Tour gebruikt. Een portretfoto is een foto die elke renner éénmaal per jaar laat maken. Op deze foto is de renner (en zijn tenue) goed zichtbaar. Elke renner heeft steeds dezelfde positie, recht tegenover de camera gericht en zichtbaar van taille tot net boven het hoofd. Figuur 9 toont de portretfoto van 4 renners : Greg Van Avermaet, Alejandro Valverde, Jurgen Roelandts en Chris Froome.

Figuur 9 : voorbeeld van enkele portretfoto's

Het ophalen van deze portretfoto's gebeurde in een eerste fase handmatig. Ir. Jelle De Bock voorzag echter een script dat automatisch deze foto's ophaalde van de site https://www.procyclingstats.com/. Dit script werd dan ook in het vervolg gebruikt. Er werden in totaal 501 foto's opgeslagen.

4.2.2 Opstellen validation set

Aangezien de training set automatisch werd opgevuld dankzij ir. Jelle De Bock, was het alleen maar logisch om ook de validation set automatisch op te vullen. Hiervoor werd gebruik gemaakt van de API van Flickr. Flickr is wereldwijd de meest gebruikte applicatie voor het uploaden en delen van foto's, ongeacht of deze afkomstig zijn van hobbyfotografen of professionele fotografen. Een logisch beginpunt voor het verzamelen van foto's van wielrenners.

Elke foto die op Flickr verspreid wordt heeft 3 kenmerken die het uniek maken :

- Titel
- Beschrijving
- één of meerdere tags

Een tag is een soort kernwoord dat bij de foto hoort. Zo zal een foto van een kikker bijvoorbeeld de tags 'frog', 'nature' en 'animal' hebben. Alhoewel een tag niet altijd even betrouwbaar of specifiek is, kan deze wel gebruikt worden bij het filteren van foto's.Om toegang te verkrijgen tot de API van Flickr werd eerst een API sleutel opgevraagd, waarvoor een Flickr account moest aangemaakt worden. Deze API sleutel dient als middel van authenticatie.

De FlickrAPI is een RESTful API waarmee gecommuniceerd kan worden via een aantal *endpoints*. Het endpoint dat gebruikt werd voor deze masterproef is de URL https://api.flickr.com/services.

Aangezien deze API een RESTFUL API is, moet deze in theorie aangesproken worden via *HTTP calls*. Om dit te vermijden werd gebruik gemaakt van een Python library die dit reeds implementeert. Deze library, de Python Flickr API, is te vinden op https://github.com/alexis-mignon/python-flickr-api en voorziet een object georiënteerde manier om de Flickr API aan te spreken. Codevoorbeeld 2 in de appendix toont hoe de Python Flickr API werd aangesproken en hoe de afbeeldingen lokaal werden opgeslagen.

De Python Flickr API is een object georiënteerde API, wat inhoudt dat dus eerst een object wordt aangemaakt die de API voorstelt. Hierna werd de methode 'walk' opgeroepen. Deze methode implementeert een HTTP call naar de Flickr API zelf, meer specifiek voor de REST call flickr.photos.search_. Deze REST call laat toe om foto's op te halen op basis van een aantal zoektermen.

De methode “walk” en de REST call “flickr.photos.search_” delen dezelfde parameters. Van deze parameters worden volgende ingevuld :

- api_key : de API sleutel van de gebruiker, deze parameter is verplicht.
- text : foto’s waarvan de titel, beschrijving of tags deze string bevatten, worden teruggegeven.
- tags : foto’s die één of meerdere van deze tags bevatten worden teruggegeven.
- extras : extra informatie die moet opgehaald worden voor de foto, naast de standaard informatie
- per_page : het maximaal aantal foto’s dat mag worden teruggegeven per uitvoering van de methode.
- sort : de manier waarop de foto’s moeten gesorteerd teruggegeven worden
- min_taken_date : de vroegste datum waarop de foto mag genomen zijn
- max_taken_date : de meest recente datum waarop de foto mag genomen zijn

Eenmaal deze parameters correct werden meegegeven, gaf de methode een set terug van alle foto’s die werden gevonden. Per team werden foto’s gezocht die aan twee vereisten voldoen. De titel of beschrijving van de foto bevatte de naam van het team. De tags van de foto bevatten minstens één van volgende tags : ‘UCI World Tour’, ‘UCI’, ‘2018’, ‘2019’, ‘cycling’, ‘wielrennen’, ‘radsport’, ‘cyclisme’, ‘sport’ of de teamnaam zelf.

De parameter ‘extras’ werd gebruikt om de URL van de foto op te vragen, zodat deze later via de python module URLlib.URLrequest lokaal opgeslagen kon worden. Een voorbeeld van welk soort foto’s deze methode teruggaf is hieronder in figuur 10 weergegeven, hier werd gezocht naar Lotto Soudal.

Figuur 10 : voorbeeld foto's gevonden via Flickr API voor Lotto Soudal

Niet elke foto die opgehaald werd is uiteraard een foto die later kon gebruikt worden. Zie onderstaand voorbeelden.

Figuur 11 : gewenste data

Figuur 12 : niet gewenste data

Figuur 11 toont een afbeelding die gebruikt kon worden, de renner komt duidelijk in beeld en de sponsor is goed zichtbaar. De rechtse foto, gegeven door figuur 12 kon uiteraard niet gebruikt worden. Beide foto's zijn echter wel direct gerelateerd aan Lotto Soudal. De opgehaalde data kon dus opgesplitst worden in twee groepen : gewenste en niet gewenste data. De niet gewenste data moest uit de data gefilterd worden. Deze kon echter verschillende vormen aannemen.

Enerzijds was er de mogelijkheid dat foto's werden gevonden die wel gerelateerd waren aan het team waarvoor gezocht werd, maar geen renners toonden, zoals getoond in figuur 12. Vaak waren dan ploegwagens, fietsen, teambussen of ander materieel zichtbaar.

Anderzijds bestond de mogelijkheid dat een foto werd gevonden voor een bepaald team, die een renner toonde van een ander team. Neem volgende situatie. Een renner van Movistar Team wint een rit in de Tour De France. In elke foto van deze rit wordt in de beschrijving van die foto de naam van deze renner en Movistar Team vermeldt. Indien vervolgens foto's worden gezocht met als zoekterm 'Movistar Team' kan in principe elke foto uit deze rit teruggegeven worden, aangezien voor elke foto de beschrijving de gezochte term bevat. Uiteraard is er geen garantie dat elke foto van deze rit effectief een renner van Movistar Team toont, dit is zelfs zeer onwaarschijnlijk.

Aangezien de niet gewenste data nutteloos was in deze fase van het project werd manueel gefilterd. De niet gewenste data werd verwijderd. Dit liet per team minimaal 2 en maximaal 6 foto's over.

In totaal bleven 79 foto's over. Een algemene vuistregel stelt dat een goede verhouding tussen de grootte van de training set en validation set 80/20 bedraagt. In dit geval bedroeg deze 501/79, wat een goede benadering was van deze vuistregel.

4.3 Eerste resultaten

4.3.1 Resultaten

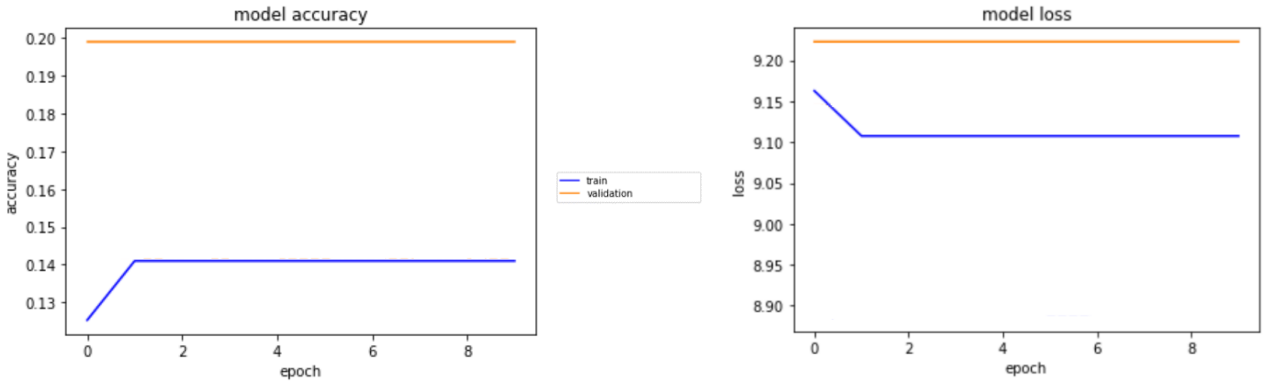
Na het ontwerpen van het model en opvullen van de dataset werd het algoritme voor een eerste maal uitgevoerd. Figuur 13 toont de resultaten.

```
Epoch 1/10
25/25 [==============================] - 21s 833ms/step - loss: -111.0950 - acc: 0.0620 - val_loss: -145.5312 - val_acc: 0.0214
Epoch 2/10
25/25 [==============================] - 16s 647ms/step - loss: -119.3447 - acc: 0.0580 - val_loss: -129.6239 - val_acc: 0.0385
Epoch 3/10
25/25 [==============================] - 16s 638ms/step - loss: -119.3264 - acc: 0.0580 - val_loss: -134.2577 - val_acc: 0.0429
Epoch 4/10
25/25 [==============================] - 16s 635ms/step - loss: -119.2305 - acc: 0.0580 - val_loss: -141.7646 - val_acc: 0.0154
Epoch 5/10
25/25 [==============================] - 18s 733ms/step - loss: -119.3447 - acc: 0.0580 - val_loss: -140.6346 - val_acc: 0.0286
Epoch 6/10
25/25 [==============================] - 16s 632ms/step - loss: -119.3447 - acc: 0.0580 - val_loss: -134.8971 - val_acc: 0.0308
Epoch 7/10
25/25 [==============================] - 16s 621ms/step - loss: -119.3447 - acc: 0.0580 - val_loss: -135.8519 - val_acc: 0.0429
Epoch 8/10
25/25 [==============================] - 16s 655ms/step - loss: -119.3447 - acc: 0.0580 - val_loss: -140.0477 - val_acc: 0.0154
Epoch 9/10
25/25 [==============================] - 17s 684ms/step - loss: -119.3447 - acc: 0.0580 - val_loss: -136.5351 - val_acc: 0.0429
Epoch 10/10
25/25 [==============================] - 16s 621ms/step - loss: -119.3447 - acc: 0.0580 - val_loss: -139.3119 - val_acc: 0.0154
```

Figuur 13 : gedetailleerd overzicht eerste resultaten classificatiemodel

Een korte definitie van bovenstaande statistieken is vereist.

- loss : een waarde die zo laag mogelijk moet gehouden worden. Hoe lager deze waarde, hoe beter de voorspellingen
- acc : een waarde die berekend wordt op basis van een wiskunde functie. Dit is een meer betrouwbare maat voor de correctheid van de voorspellingen.
- val_loss : naar analogie met loss, maar tijdens de validation fase
- val_acc : naar analogie met acc, maar tijdens de validation fase
-



Figuur 14 : eerste resultaten classificatiemodel

4.3.2 Evaluatie en oorzaken tegenvallende resultaten

Figuur 14 toont de evolutie van de statistieken loss en acc voor beide de training en validation fase doorheen de verschillende epochs. Wat onmiddellijk opvalt is dat de waarden van deze statistieken tijdens de training fase bij het begin van de tweede epoch stagneren. Dit is een duidelijk teken van overfitting en volledig te wijten aan de inhoud van de training set, zoals besproken in sectie 4.3.2.1.

De evolutie van de statistieken in de validation fase zijn deels te verklaren door het ondermaatse niveau van training tijdens de training fase, maar ook door de inhoud van de validation set. Een andere reden waarom de val_acc schommelde rond een lage waarde was de keuze van loss functie.

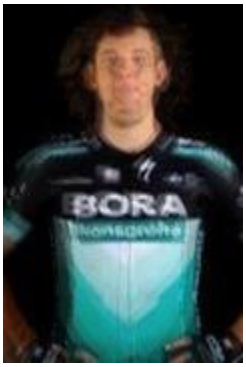
4.3.2.1 Inhoud training set

4.3.2.1.1 Oorzaak overfitting

De inhoud van de training set had een grote invloed op de slechte resultaten. Elke portretfoto had namelijk ongeveer dezelfde vorm, zoals reeds beschreven in sectie 4.1. Deze kleine variatie in wat zichtbaar was op de foto's zorgde ervoor dat het CNN patronen herkende in de data die niet ontdekt moesten worden. Het model leed aan overfitting.

Een CNN is op zich niets meer dan een black box. Een bepaalde groep van afbeeldingen wordt geanalyseerd en hierin worden de meest duidelijke patronen geïdentificeerd. Deze patronen worden later gebruikt om nieuwe afbeeldingen te analyseren en in dit geval te classificeren. Om uit te leggen wat precies was misgelopen, worden figuur 15 en figuur 16 gebruikt.

Figuur 16 : portretofoto's renners Bora-Hansgrohe



Figuur 15 : superimpositie figuur 16

De 4 afbeeldingen in figuur 15 tonen 4 renners van BORA – Hansgrohe, zoals deze ook in de training set te vinden waren. Als persoon is het eenvoudig te zien dat deze 4 afbeeldingen elk een verschillende renner

tonen die elk uitkomen voor Bora-Hansgrohe, het CNN heeft echter geen idee van wat zichtbaar is. Het gaat op zoek naar de meest duidelijke terugkerende patronen.

De eenvoudigste manier om patronen in een groep van afbeeldingen bloot te leggen is het analyseren van de superimpositie van deze afbeeldingen. De superimpositie, zoals getoond in figuur 16, van een groep afbeeldingen wordt verkregen door de afbeeldingen boven elkaar te plaatsen, verzekerende dat elke afbeelding individueel herkenbaar blijft.

De superimpositie van de 4 getoonde afbeeldingen toont volgende terugkerende patronen :

- Zwarte achtergrond
- De tekst Bora en Hansgrohe op de borstkas
- Vorm van het lichaam
- Plaats van gezicht
- Kleur van gezicht
- Kleur van tenue
- Patronen te zien op tenue

Er waren met andere woorden enorm veel patronen te herkennen, waarvan maar enkele effectief herkend dienden te worden. Zo zou een renner van Bora-Hansgrohe herkend kunnen worden op basis van de tekst op het tenue of de patronen die het tenue kenmerken, alsook de kleur van het tenue.

De positie van het lichaam, kleur en plaats van gezicht of de zwarte achtergrond zijn geen patronen die specifiek aan Bora-Hansgrohe toegewezen konden worden. Als volgt was het detecteren van deze laatst opgenoemde patronen nefast voor de performantie van het model, aangezien het model er van uitging dat elke afbeelding die een renner van Bora-Hansgrohe bevatte bovenstaande patronen zou bevatten.

4.3.2.2 Inhoud validation set

De inhoud van de validation set bracht ook problemen met zich mee. Tijdens het wielerseizoen worden namelijk speciale truien gedragen. Zo zal de leider in het algemeen klassement in de Tour De France steeds de gele trui dragen, of zal de leider in het bergklassement de zogenaamde bolletjestrui dragen. Een renner kan ook nationaal kampioen of wereldkampioen zijn, waarbij deze renner dan steeds een variatie van het tenue van zijn wielerploeg draagt. Enkele voorbeelden zijn in figuur 17 gegeven.

Figuur 17 : voorbeelden speciale truien

Alhoewel de training set ook een aantal portretfoto's van nationale kampioenen bevatte, was de impact hiervan verwaarloosbaar op de performantie tijdens de training fase. Aldus herkende het model per team patronen, in de verwachting dat elke renner het standaardtenue van het team droeg. Indien dit niet het geval was, waren de voorspelling niet langer accuraat. Een voorbeeldsituatie wordt geschetst aan de hand van figuur 18.

Figuur 18 : voorbeeld verwarring speciale trui

De middelste afbeelding toont wielrenner Valerio Conti, rijdend voor wielerploeg UAE – Team Emirates. Op het moment waarop de afbeelding werd genomen was Conti leider in het algemeen klassement van de Giro D'Italia, waardoor hij de roze leiderstrui droeg. Deze rode leiderstrui zorgde ervoor dat het herkennen van de wielerploeg van Conti enorm moeilijk werd. Om dit te illustreren tonen de twee uiterste afbeelding de superimpositie van de afbeeldingen in de training set, horend bij respectievelijk UAE – Team Emirates en EF Education First.

Na vergelijken van de middelste afbeelding met beide superimposities wordt duidelijk dat de roze kleur van het tenue de bovenhand haalt op de letters UAE zichtbaar op de borstkas van de renner. Aldus zal de afbeelding van Conti toegewezen worden aan wielerploeg EF Education First, wat een foutieve voorspelling is. Doordat de validation set een aantal afbeeldingen bevatte die een gelijkaardig scenario met zich meebrachten, daalde de validation accuracy.

4.3.2.3 Keuze loss functie

Alvorens dieper in te gaan op de gemaakte keuze van loss functie en de reden waarom deze foutief was, is het beter eerst te definiëren wat een loss functie is. Eenvoudig verwoord geeft de loss functie een beeld van de kwaliteit van de voorspellingen van een model. Indien de voorspelling inaccuraat zijn zal de loss functie een hogere waarde teruggeven, zijn deze voorspellingen accuraat, dan zal de loss functie lagere waarden teruggeven.

De loss functie heeft dus ook een impact op het leerproces zelf. Alhoewel het model zelf bepaalt of een voorspelling correct was of niet, geeft de loss functie een beter beeld over de kwaliteit van voorspellingen over een volledige batch. Hier uit volgt dat het model zijn leerproces aanpast op basis van de waarde die de loss functie teruggeeft. Is deze waarde relatief klein, dan weet het model dat de voorspelling van goede kwaliteit zijn en kan het model verder gaan op de patronen die het reeds ontdekt heeft. Indien deze waarde echter relatief groot is zal het model het leerproces grotendeels opnieuw opstarten, aangezien de patronen die tot dan toe ontdekt werden tot slechte voorspellingen leidden.

Zoals codevoorbeeld 1 in de appendix toont werd als loss functie gekozen voor 'binary_crossentropy' . Deze keuze bleek foutief. Deze lossfunctie wordt vaak gebruikt in het geval van multi label classification, het type classification dat in deze masterproef werd geïmplementeerd was echter multi class classification. Het verschil tussen beide is subtiel.

In beide types classification bestaan meerdere klassen, maar in het geval van multi label classification kan aan een sample meerdere klassen (of labels) worden toegewezen, waar bij multi class classification maar één klasse kan worden toegewezen. In het vervolg van deze masterproef werd als loss functie gekozen voor ‘categorical_crossentropy’. Deze loss functie is geoptimaliseerd voor multi class classification.

4.4 Aanpassen model

Zoals reeds vermeld in de vorige paragraaf werd gekozen voor een andere loss functie, ‘categorical_crossentropy’ . Tevens werd besloten om een convolutional, activation en pooling layer uit het model te verwijderen, in de hoop overfitting zo te beperken.

Hiernaast werden ook het aantal nodes in de laatste dense layer aangepast, deze bevatte nu 18 nodes in plaats van 1 node. Als gevolg hiervan was de uitvoer van het model nu niet langer één waarde, maar een combinatie van 18 waarden, waarbij elke waarde een probabiliteit voorstelde dat de behandelde afbeelding tot de corresponderende klasse behoorde. De 18 klassen komen dan uiteraard overeen met de 18 wielerploegen uit de UCI World Tour.

Merk op dat aangezien het model de afbeeldingen classificeerde, het model zelf besliste aan welke klasse de afbeelding uiteindelijk werd toegewezen, namelijk de klasse met de hoogste probabiliteit.

4.5 Aanpassen dataset

Aangezien de resultaten van de eerste test tegenvallend waren, werd besloten om hierna nieuwe technieken toe te passen om de data op te halen. Ongeacht de oude of nieuwe technieken bleek één probleem steeds terug te keren en niet eenvoudig op te lossen zijn. De opgehaalde afbeeldingen bevatten vaak geen renners, maar andere content gerelateerd aan het team waarnaar gezocht werd.

Als antwoord op dit probleem stelde promotor Prof. dr. Steven Verstockt een aantal technieken voor die automatische object detectie uitvoeren op foto's. Uit deze technieken werden besloten om You Only Look Once (YOLO) te gebruiken.

YOLO is een Python object detection framework dat gebruik maakt van deep learning en een CNN om zo objecten te detecteren. YOLO hoeft echter zelf niet meer getraind te worden. Een goede vergelijking is dat het eindwerk van deze masterproef een kleinschalig alternatief zou kunnen zijn voor YOLO, specifiek getraind voor het herkennen van wielerploegen.

Aangezien YOLO vooraf getraind is, was er geen keuze meer in welke klassen beschikbaar waren. In totaal stelt YOLO 93 klassen beschikbaar, waarvan in het kader van deze masterproef volgende bruikbaar waren : person, bycicle, car, motorbike, bus .

YOLO detecteert objecten in een afbeelding door de bounding box van dat object te bepalen. Een bounding box kan best gedefinieerd worden als een denkbeeldig kader dat rondom het object getekend wordt. Figuur 19 toont hoe YOLO objecten in een afbeelding detecteert.

Figuur 19 : detectie objecten in afbeelding via YOLO

4.5.1 Ophalen data training set

Er werd besloten om niet langer gebruik te maken van de portretfoto's om de dataset op te vullen. Aangezien de training set nu leeg was, werden onderstaande technieken gebruikt om deze op te vullen.

- Flickr API
- Google Custom Search API
- Handmatig verzamelen afbeeldingen

De FlickrAPI werd gebruikt zoals reeds beschreven werd in sectie 4.2.2. Het filteren van de data gebeurde nu echter niet handmatig, maar via YOLO. Het handmatig verzamelen van afbeeldingen diende vooral om een training set op te bouwen die evenwaardig verdeeld was over alle klassen heen.

De Google Custom Search API is een RESTful API die vaak gebruikt wordt door derden om een aangepaste Google zoekfunctie toe te voegen aan een website. In het kader van deze masterproef werd deze API gebruikt om afbeeldingen op te halen van de verschillende wielerploegen.

In analogie met de Flickr API geldt dat de Google Custom Search API in theorie aangesproken moet worden via HTTP calls. Er werd echter opnieuw gebruik gemaakt van een object georiënteerde API die dit reeds implementeert, de google-api-python-client. Deze API is te vinden via volgende URL : https://github.com/googleapis/google-api-python-client .

Codevoorbeeld 3 in de appendix toont hoe de Google Custom Search API werd aangesproken. Aangezien de gebruikte module, google-api-python-client, object georiënteerd is, werd een object aangemaakt waarop de methodes cse en list achtereenvolgens werden opgeroepen. Deze laatste methode verwacht volgende parameters :

- q : de query, zoals deze zou ingetypt worden op de website van Google zelf
- cx : custom search engine id, deze werd op voorhand gegenereerd
- searchType : type data binnen te halen, in dit geval afbeeldingen
- num : aantal foto's binnen te halen per request
- imgType : verplichte parameter in combinatie met num
- fileType : type extensie, png leek de meest logische keuze
- safe : safesearch of niet, hier werd gekozen voor niet

Deze methode gaf een dictionary terug die metadata bevat over de zoekopdracht, gebruikte zoekmachine alsook de zoekopdrachten zelf. De zoekopdrachten werden geraadpleegd door de list op te halen, horend bij key 'items' in de dictionary. Deze list werd vervolgens overlopen waarna in analogie met de Flickr API gebruik werd gemaakt van de urllib.request module om de afbeelding lokaal op te slaan.

Na gebruik van de Google Custom Search API kwamen 2 grote nadelen aan het licht. Ten eerste heeft de API een beperking op het aantal *queries* dat kan uitgevoerd worden, namelijk 100 per dag. Ten tweede waren de afbeeldingen die werden opgehaald niet zeer bruikbaar. Hieronder worden in figuur 20 enkele afbeeldingen getoond die gevonden werden voor trefwoord 'Deceuninck – Quick Step'.

Figuur 20 : voorbeeld foto's gevonden via Google Custom Search API voor Deceuninck - Quick Step

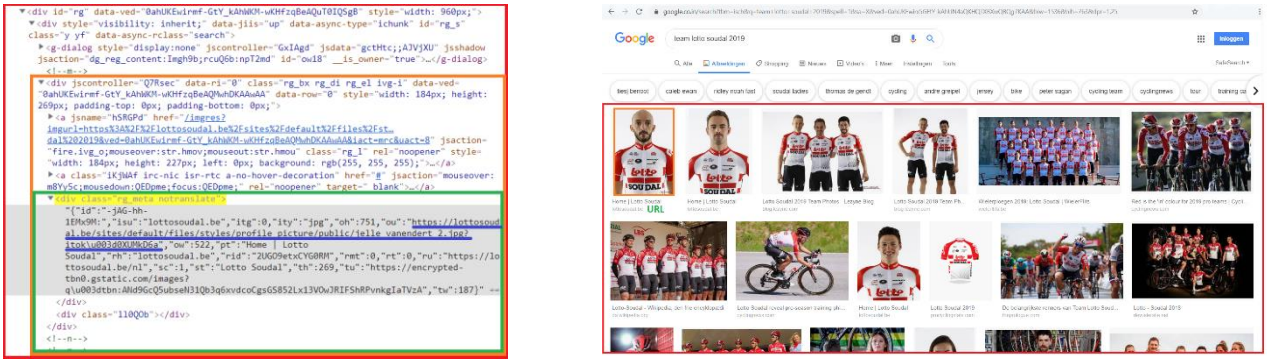
Ondanks deze nadelen bleek de dataset toch genoeg samples te bevatten, dit dankzij het gebruik van de Flickr API. Tevens was deze evenwichtig verdeeld na het handmatig verzamelen van extra afbeeldingen.

4.5.2 Ophalen data validation set

Het opstellen van de validation set gebeurde aan de hand van HTML *scraping*. HTML scraping is een techniek waarbij specifieke data uit een HTML pagina wordt onttrokken. Deze techniek werd in het kader van deze masterproef toegepast om afbeeldingen van renners op te slaan uit de HTML pagina van Google Afbeeldingen. Codevoorbeeld 4 in de appendix toont het script dat hiervoor gebruikt werd.

Om de data uit een HTML pagina te onttrekken werd gebruik gemaakt van BeautifulSoup, een Python library die niet alleen data uit HTML, maar ook uit XML pagina's kan onttrekken. Opdat deze library gebruikt kon worden moest eerst een object aangemaakt worden. De constructor verwachte twee argumenten. Het eerste argument is de URL waarvan de HTML pagina wordt geladen. Het tweede argument is een header waarin het enige verplichte veld, de 'User-Agent', ingevuld dient te worden. Om problemen te vermijden werden de meest gebruikte browsers in dit veld ingevuld.

De volgende stap was het analyseren van de HTML pagina van Google Afbeeldingen, op zoek naar het HTML element van een afbeelding dat de bijhorende URL bevatte.



Figuur 21 : HTML Scraping

Figuur 21 toont de plaats van de URL in de HTML code, alsook de relatie tussen het HTML bestand en de HTML pagina, geladen door een browser.

Om deze URL op te halen werd de methode find_all opgeroepen op het BeautifulSoup object, deze methode verwachte twee paramaters : het soort HTML element en een conditie om te controleren. Aangezien het gezochte HTML element gekend was, werden deze parameters meegegeven om zo alleen deze elementen (op figuur 23 in het groen omkaderd) terug te geven.

Van dit HTML element werd vervolgens 2 attributen opgevraagd, respectievelijk ‘ou’ en ‘ity’ welke de link en het type van de afbeelding teruggaven. Hierna werd opnieuw gebruik gemaakt van de Urllib.request module om de gezochte afbeelding lokaal op te slaan.

4.5.3 Manipuleren data via YOLO

Zoals reeds vermeld was het noodzakelijk om de afbeeldingen in de dataset te manipuleren opdat deze enkel nog renners toonden. In een eerste fase werd YOLO gebruikt om de afbeeldingen in de dataset onder te brengen in 2 grote categorieën : ‘wielrenner’ en ‘niet wielrenner’. Alhoewel dit in principe niet nodig was, gaf dit een beeld over hoeveel opgehaalde data effectief bruikbaar was en of de gebruikte technieken dus wel efficiënt waren.

Eenmaal deze onderverdeling bereikt was, werden de resterende afbeeldingen in de categorie ‘wielrenner’ verder verwerkt. Deze afbeeldingen bevatten namelijk wielrenners, maar de afbeelding was niet van voldoende kwaliteit om deel uit te maken van de dataset, ongeacht training of validation set. Kwaliteit slaat hierbij niet op de zuiverheid van de afbeelding, maar eerder op de inhoud ervan.

Elke afbeelding in de dataset dient exact één renner te tonen, zonder zijn omgeving. De reden hiervoor is eenvoudig. Stel dat de dataset bestaat uit 1000 afbeeldingen, evenwaardig verdeeld over de verschillende klassen, die elk één renner tonen in een bepaalde omgeving.

Een omgeving kan een snelweg zijn waarbij in de achtergrond andere renners zichtbaar zijn of een aardeweg tijdens een beklimming in een Italiaanse klassieker. In elk geval zal de aanwezigheid van deze omgeving, of dus de achtergrond van de afbeelding, een effect hebben op het leerproces van het model.

Het model is, zoals eerder aangehaald, namelijk een black box die elk mogelijk patroon wens te herkennen. Deze herkenbare patronen zijn niet beperkt tot de renner die in de voorgrond van de afbeelding getoond wordt, maar zullen verspreid zijn over de volledige afbeelding aangezien het model zoveel mogelijk patronen wenst te herkennen. Om zo weinig mogelijk nutteloze patronen in de achtergrond te ontdekken of om met andere woorden het model tegen zichzelf te beschermen, is het beter om de achtergrond van een afbeelding niet op te nemen in de uiteindelijke dataset.

Hieruit volgt dat er dus een methode gezocht moest worden om de renner en de achtergrond van de afbeelding van elkaar te scheiden. Hiervoor werd YOLO gebruikt. Codevoorbeeld 5 in de appendix toont het gebruikte algoritme.

YOLO voerde eerst voorbereidend werk uit waarbij de afbeelding werd geanalyseerd en de verschillende objecten geïdentificeerd werden aan de hand van de corresponderende bounding boxes, zoals getoond in figuur 19. De list out_classes bevatte alle gedetecteerde klassen in de afbeelding. Aangezien dit algoritme enkel diende om wielrenners te identificeren was een afbeelding bruikbaar als en alleen als deze een persoon en fiets bevatte.

YOLO vertaalt elke klasse naar een id, waarbij deze van de klassen persoon en fiets respectievelijk 0 en 1 zijn. Aldus werd eerst gecontroleerd of in de afbeelding een persoon en fiets gedetecteerd waren. Na het bijhouden van de index van de gedetecteerde personen en fietsen in de list out_classes, werden deze gebruikt om de coördinaten van de corresponderende bounding boxes op te halen. Op basis hiervan werd de relatieve positie van persoon en fiets gespiegeld aan deze van een wielrenner. Deze relatieve positie van wielrenner tot fiets werd geëvalueerd aan de hand van de volgende 2 predicaten :

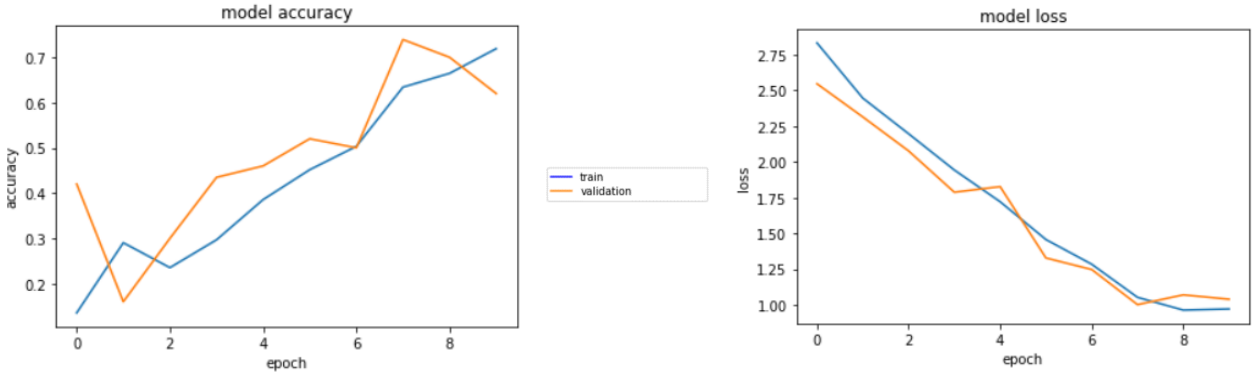
- Het horizontale midden van de bounding box van de wielrenner ligt tussen de horizontale uitersten van de bounding box van de fiets
- De laagste grens van de bounding box van de fiets ligt boven het verticale midden van de bounding box van de wielrenner

Merk op dat de oorsprong van het assenstelsel linksboven ligt, waardoor de y-coördinaten geïnverteerd zijn. Indien een persoon en een fiets voldeden aan deze twee voorwaarden werd de index van de persoon opgeslagen in een nieuwe list, genaamd riderscrop. Deze list bevatte op het einde van het algoritme de indices van alle renners die gedetecteerd werden in een afbeelding.

De voorlaatste stap in dit algoritme was controleren of de gevonden renner de moeite waard was om toe te voegen aan de dataset. Indien de bounding box van een gevonden renner een relatief kleine oppervlakte van de oorspronkelijke afbeelding innam, was de kans groot dat de kwaliteit van dat deel van de afbeelding te slecht zou zijn om het model ten goede te komen. Indien de breedte en hoogte van de bounding box van een renner respectievelijk minimaal 1/20ste en 1/5de van de corresponderende dimensies van de afbeelding bedroegen, werd de renner toegevoegd aan de dataset.

De laatste stap was het bijsnijden van de afbeelding, dit gebeurde op basis van de coördinaten van de gevonden renner.

4.6 Eindresultaten



Figuur 22 : eindresultaten classificatiemodel

Na het aanpassen van de dataset en het model werd een nieuwe test uitgevoerd. Deze resultaten, getoond in figuur 22, toonden een opmerkelijke verbetering tegenover het vorige model. Zo was tijdens de training en validation fase de accuracy gestegen tot respectievelijk 72,25% en 73,91%. Merk op dat deze laatste waarde, de maximale accuracy bereikt tijdens de validation fase, niet voorkwam op het einde van de validation fase. Zoals de bovenstaande grafiek toont werd deze waarde bereikt na de 8ste epoch. Hierna daalde deze waarde in opeenvolgende epochs tot 68,19% en 62,28%.

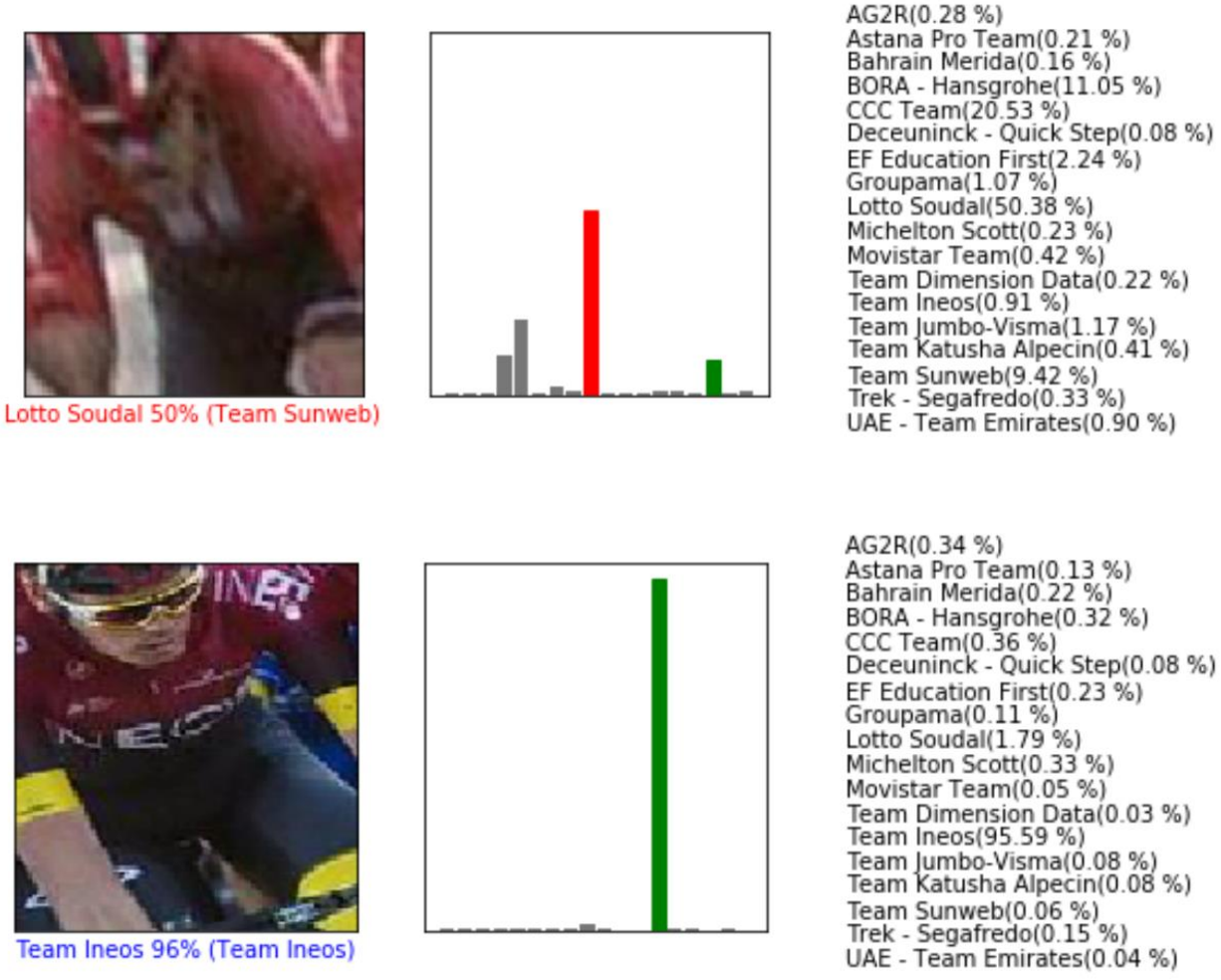
Een verklaring voor deze dalende trend werd gevonden door het verder analyseren van de laatste twee epochs. Zo valt op dat tijdens de laatste twee epochs de training accuracy bleef stijgen, terwijl deze validation accuracy daalde. Als gevolg hiervan werd geconcludeerd dat overfitting de oorzaak was van deze dalende trend. De redenering die hieraan voorafging was de volgende.

Aangezien de training accuracy bleef stijgen, betekende dit dat het model er succesvol in slaagde nieuwe patronen te ontdekken in de dataset welke toelieten de verschillende afbeeldingen met een hoger succespercentage toe te wijzen aan de corresponderende klassen. Ondanks deze nieuw ontdekte patronen daalde de validation accuracy.

De enige mogelijke verklaring hiervoor was dat de ontdekte patronen niet voorkwamen in de validation set, want indien ze daar wel voorkwamen zou ook de validation accuracy gestegen zijn. De ontdekte patronen waren met andere woorden uniek aan de training set of dus te wijten aan toevallig aanwezige patronen in die specifieke verzameling data. Aldus leed het model opnieuw aan overfitting.

Hoofdstuk 5 : Regressie

In overleg met Prof. Dr. Steven Verstockt en ir. Jelle De Bock werd besloten om over te gaan van classificatie naar regressie. Deze overgang heeft als gevolg dat indien een bepaalde afbeelding werd geanalyseerd door het model de voorspelling niet langer een klasse was, maar een waarschijnlijkheidsverdeling over de verschillende klassen. Zie figuur 23 voor enkele voorbeelden.



Figuur 23 : voorbeelden uitvoer regressie

5.1 Ontwerp basismodel

Aangezien overgegaan werd op regressie in plaats van classificatie moest het gebruikte model uiteraard gewijzigd worden. Er werd gekozen om het nieuwe model niet te baseren op het vorige model, maar om volledig opnieuw te beginnen. Codevoorbeeld 6 in de appendix toont hoe het nieuwe model werd opgebouwd.

Dit model was overduidelijk veel eenvoudiger dan het vorige model, maar diende ook maar als basismodel waarop verder werd gebouwd. Dit basismodel bevatte maar 3 lagen.

De eerste laag was een flatten layer die verzekerd dat de input voor de hier op volgende dense layers ééndimensionaal was van vorm. Merk op dat opnieuw impliciet een input layer aanwezig was, aangezien het argument 'input_shape' werd meegegeven met de flatten layer. Na deze flatten layer volgden 2 dense layers. De eerste hiervan beschikte over 128 nodes, de tweede over 18 nodes.

5.2 Opstellen dataset

De beslissing om over te gaan van classificatie naar regressie ging gepaard met de beslissing om de dataset niet langer modulair op te bouwen, maar als een geheel. Waar de training en validation set vooraf apart werden opgevuld, werd op dit punt eerst een hoeveelheid data opgeslagen, waarna deze werd verdeeld over training en validation set. Opnieuw werden de afbeelding verdeeld met een ratio 80/20.

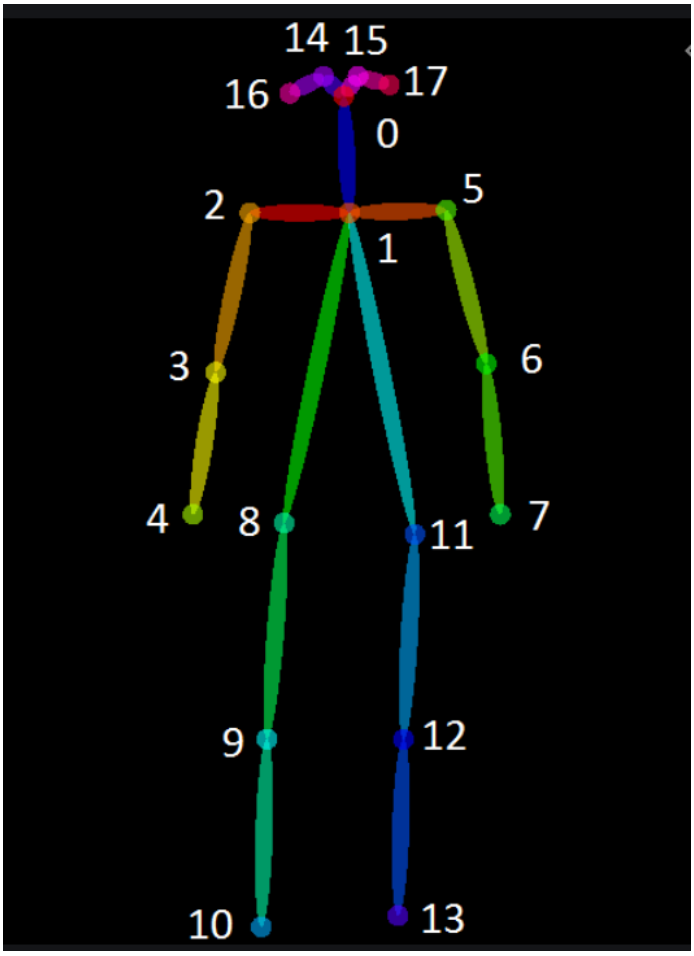
Het opstellen van de dataset gebeurde enerzijds door het aanspreken van de Flickr API en anderzijds door het lokaal opslaan van frames van verschillende video-uitzendingen. Opnieuw werd YOLO gebruikt om in deze afbeeldingen (onafhankelijk van welke databron ze afkomstig waren) de verschillende renners te detecteren. In de hoop de nauwkeurigheid van het model nog verder te verbeteren werd elke afbeelding tevens verder gemanipuleerd via Openpose.

Openpose is een *human pose estimation* library, geschreven in Python. Eenvoudig verwoord detecteert Openpose personen in een afbeelding en probeert het hierna een skelet af te beelden op deze personen.

De afbeeldingen die openpose als input kreeg te verwerken, waren steeds de afbeeldingen die YOLO als output had gepresenteerd. In deze afbeeldingen was de renner met andere woorden al gedetecteerd. Het doel van Openpose was deze afbeelding van de renner te herleiden tot dat deel van de afbeelding dat het model de grootste kans gaf de wielerploeg van de renner te herkennen. Na overleg met Prof. Dr. Steven Verstockt werd besloten elke afbeelding te herleiden tot het bovenlichaam van de renner.

5.2.1 Openpose pose detection

Zoals reeds vermeld tracht Openpose op elke persoon die het kan detecteren een skelet af te beelden. In het beste geval bestaat het skelet uit 18 sleutelpunten. Elk sleutelpunt komt hierbij overeen met een lichaamskenmerk zoals de linkerschouder of borstkas, maar ook minder prominente lichaamskenmerken zoals bijvoorbeeld de ogen of oren. Het skelet en de bijhorende sleutelpunten worden hieronder in figuur 24 getoond.



0 : Neus
1 : Borstkas
2 : Rechterschouder
3 : Rechterelleboog
4 : Rechterpols
5 : Linkerschouder
6 : Linkerelleboog
7 : Linkerpols
8 : Rechterheup
9 : Rechterknie
10 : Rechterenkel
11 : Linkerheup
12 : Linkerknie
13 : Linkerenkel
14 : Rechteroog
15 : Linkeroog
16 : Rechteroor
17 : Linkeroor

Figuur 24 : skelet Openpose

In het slechtste geval detecteert openpose maar enkele sleutelpunten. Deze sleutelpunten zijn dan vaak zeer geconcentreerd, zoals bijvoorbeeld de linkerheup, linkerelleboog en linkerpols. In het gemiddelde geval detecteert openpose tussen de 10 en 18 sleutelpunten, waarbij meestal de sleutelpunten in het bovenlichaam gedetecteerd worden. Figuur 25 toont twee gevallen, respectievelijk het beste en gemiddelde geval.

Figuur 25 : voorbeeld uitvoer Openpose

Niet elke afbeelding die via openpose gedetecteerd wordt is dus later bruikbaar. In principe volstaan de meeste afbeeldingen aangezien het detecteren van het bovenlichaam voldoende is en de heupen, schouders en hoofd meestal gedetecteerd werden.

Codevoorbeeld 7 in de appendix toont hoe Openpose werd geïmplementeerd. Een skelet werd niet sleutelpunt per sleutelpunt, maar per paar sleutelpunten overlopen. Een paar sleutelpunten bestaat dan steeds uit twee sleutelpunten die in het skelet met elkaar verbonden zijn, dit kan bijvoorbeeld de linkerpols en linkerelleboog zijn, maar ook bijvoorbeeld het rechteroog en rechteroor. Deze verzameling van paren sleutelpunten zijn opgeslagen in de list BODY_PARTS_KPT_IDS.

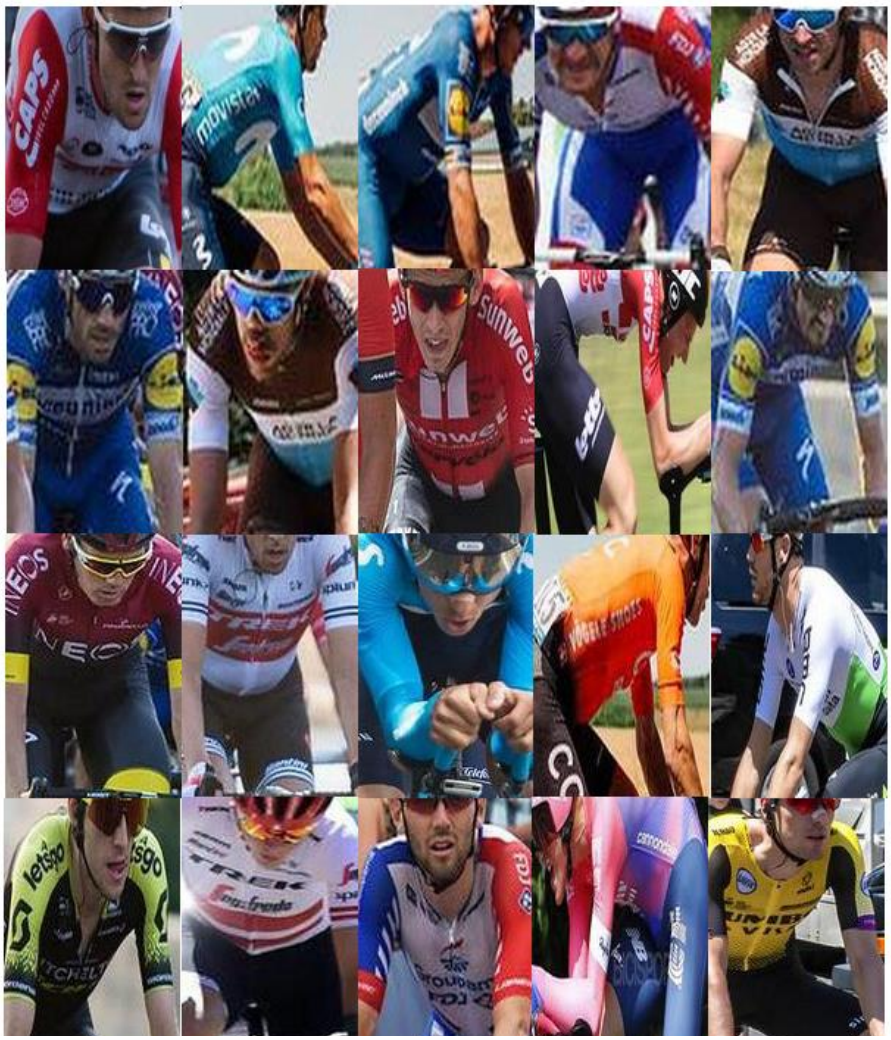
Op basis van de id van een paar sleutelpunten werd vervolgens gecontroleerd of deze sleutelpunten gedetecteerd werden in de afbeelding. Op basis hiervan werd een array all_valid_points opgebouwd. De structuur van deze array was steeds dezelfde. De index gaf aan over welk sleutelpunt het gaat (de array heeft een vaste lengte van 18, naar de 18 sleutelpunten die kunnen ontdekt worden).

Elk element van de array was een nieuwe array met daarin 4 elementen, in volgorde :

- Volgnummer, geeft volgorde in de welke de sleutelpunten zijn gedetecteerd
- X-coördinaat van sleutelpunt
- Y-coördinaat van sleutelpunt
- Accuraatheid van sleutelpunt

Merk op dat indien een sleutelpunt niet gedetecteerd werd in de afbeelding, het 2de en 3de element -1 zijn, het vierde element 0.

Op basis van deze array werd vervolgens bepaald of de schouders, heupen en hoofd gedetecteerd zijn. Indien dit het geval was, werd op basis van de coördinaten van deze sleutelpunten bepaald welk deel van de afbeelding behouden diende te worden, waarna de afbeelding werd bijgesneden. Figuur 26 schetst een beeld van hoe de dataset er op dit moment uitzag.

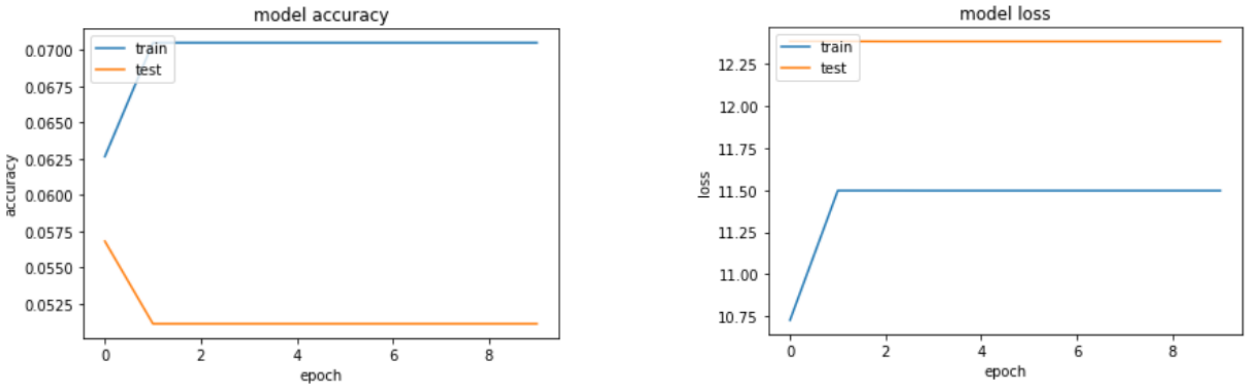


Figuur 26 : overzicht dataset

5.3 Eerste resultaten

5.3.1 Resultaten

De eerste resultaten zijn hieronder gegeven.



Figuur 27 : eerste resulaten regressiemodel

5.3.2 Evaluatie en oorzaken tegenvallende resultaten

Figuren 27 toont de accuracy en loss tijdens de training en validation fase. Deze resultaten zijn zeer gelijkaardig aan de resultaten die werden bekomen bij classificatie, toen het model daar voor de eerste maal werd getraind. De oorzaak van deze tegenvallende resultaten lag toen vooral aan de inhoud van de dataset. Zoals figuur 26 echter toont was de dataset in dit geval goed opgebouwd, waarbij elk team evenwaardig vertegenwoordigd was. Alhoewel de resultaten dus zeer gelijkaardig waren, lag de oorzaak ergens anders.

5.3.2.1 Data augmentation

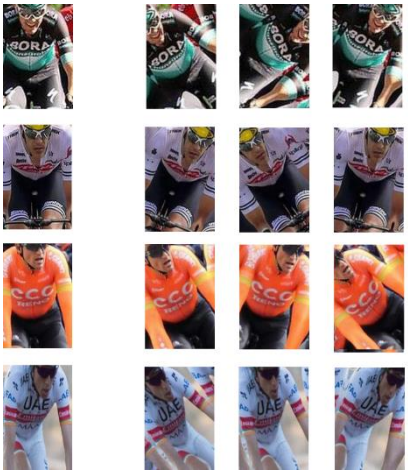
De huidige dataset bevatte 730 afbeeldingen die verdeeld werden over de training set en validation set volgens een ratio 80/20. Alhoewel de dataset dus duidelijk genoeg samples bevatte om uit te leren wordt in verschillende literatuurstudies toch aangeraden om een dataset van minstens enkele duizenden samples op te bouwen.

Om een dataset van noemenswaardige grootte op te stellen bestonden twee mogelijke opties : meer data verzamelen of de reeds opgeslagen data multipliceren. Om tijd te besparen werd gekozen voor deze twee optie, wat werd gerealiseerd aan de hand van data augmentation. Data augmentation is per definitie een techniek waar bij een dataset bestaande uit een relatief klein aantal samples wordt uitgebreid tot een dataset met een groot aantal samples door elke sample apart te manipuleren op verschillende manieren.

Aangezien een sample in de context van deze masterproef een afbeelding was, werd gezocht naar manieren om een afbeelding te vervormen.

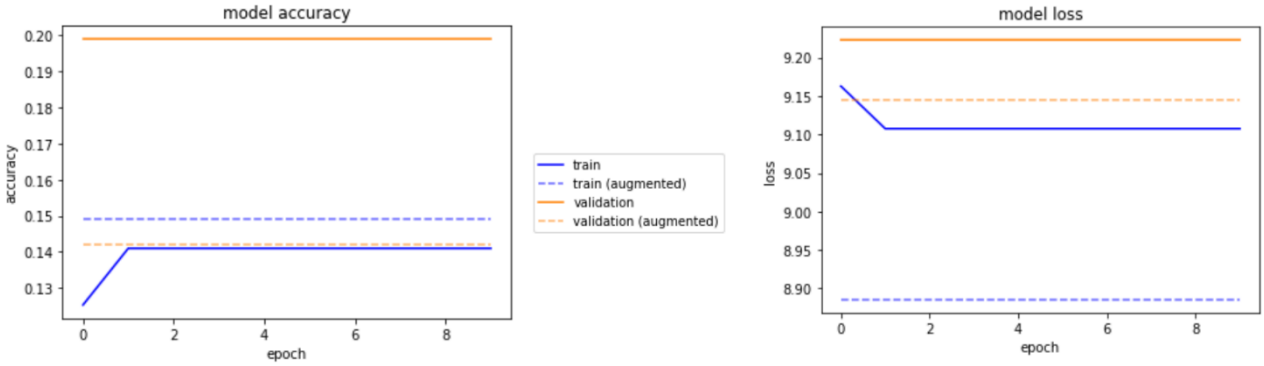
Om de afbeeldingen te vervormen werd gebruik gemaakt van de ImageDataGenerator klasse, die voorzien wordt door Keras zelf. De ImageDataGenerator klasse voorziet een tiental manieren om een afbeelding te manipuleren/vervormen. Codevoorbeeld 8 in de appendix toont hoe de ImageDataGenerator klasse werd toegewend.

Elke afbeelding werd op twee verschillende manieren gemanipuleerd. Enerzijds werd elke afbeelding geroteerd met een hoek van maximaal 40°, anderzijds werd ook elke afbeelding horizontaal gespiegeld. Het argument fill_mode, met waarde ‘reflect’, geeft aan dat indien een afbeelding na rotatie niet langer binnen de oorspronkelijke dimensies paste, het overbodige stuk op het lege stuk werd geplakt. Figuur 28 toont enkele voobeelden.



Figuur 28 : voorbeelden data augmentation

Na het vergroten van de dataset via data augmentation werd het model nogmaals uitgevoerd, om zo het effect van data augmentation in kaart te brengen. Figuur 29 toont dat deze data augmentation voor een lichte verbetering zorgde tijdens de training fase, maar een negatieve impact had tijdens de validation fase.



Figuur 29 : resultaten regressiemodel na data augmentation

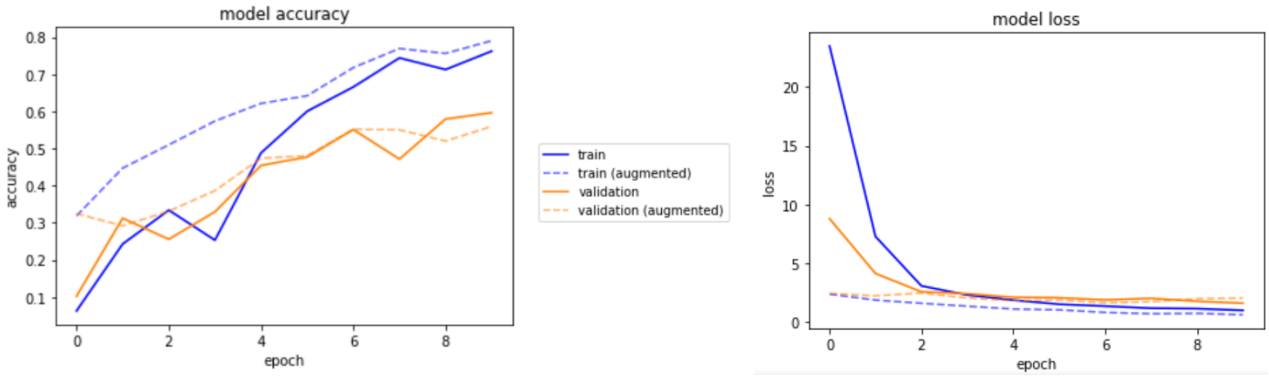
Aangezien de validation fase een beter idee geeft over hoe het model zou presteren in real time situaties was het toepassen van data augmentation hier dus nefast voor het model.

5.3.2.2 Activatiefunctie

Een eerste wijziging in de architectuur van het model was het veranderen van de activatiefunctie van de laatste dense layer. Er werd gekozen om vanaf dit punt gebruik te maken van de softmax activatiefunctie. Deze softmax activatiefunctie is geoptimaliseerd voor multi class classification problemen en dus geschikt in het kader van deze masterproef. Hieronder wordt de softmax activatiefunctie kort aangehaald, waarna de resulaten worden besproken.

Eenvoudig verwoord krijgt de softmax activatiefunctie n inputs, in een interval met onbekende grenzen, waarbij elke input positief of negatief kan zijn. Deze array van inputs wordt vervolgens genormaliseerd zodat de som over deze n inputs steeds 1 is. De verschillende inputs, op dit punt dus steeds positief, worden met andere woorden vertaald naar probabiliteiten, verdeeld over alle mogelijke klassen (aangezien er even veel inputs als klassen zijn).

De resultaten na het veranderen van de activatiefunctie zijn gegeven in figuur 30.

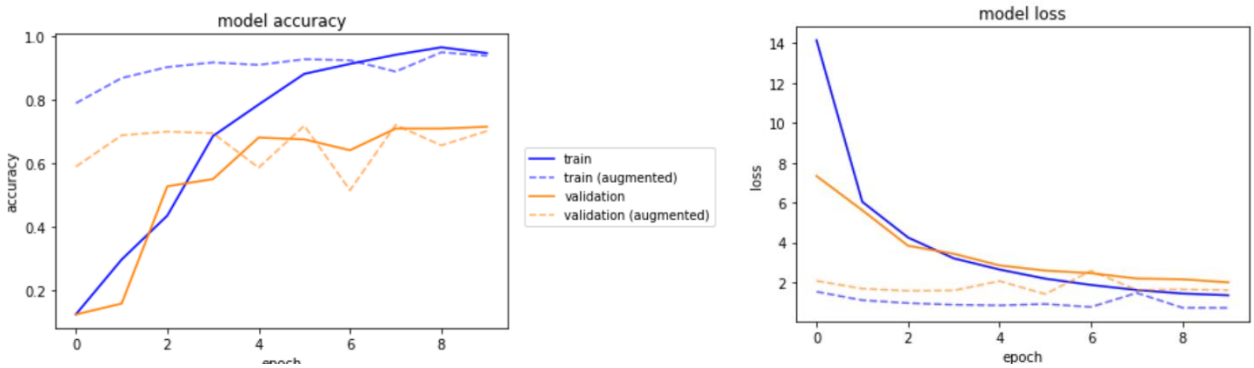


Figuur 30 : resulaten regressiemodel na wijziging activatiefunctie

De keuze om de activatiefunctie in de laatste dense layer te wijzigen had dus duidelijk een positief effect op de performantie van het model. Zonder data augmentation bereikte de training en validation accuracy maximale waarden van respectievelijk 76,24% en 59,66%. Opmerkelijk is dat indien data augmentation wel werd toegepast de training accuracy steeg tot 79,05%, maar de validation accuracy daalde tot 55,97%.

5.3.2.3 Architectuur model

5.3.2.3.1 Toevoegen convolutional en pooling layer



Figuur 31 : resultaten regressiemodel na toevoegen layers

Ondanks de afwezigheid van een convolutional layer slaagde het model er reeds in om net iets meer dan de helft van de voorspelling accuraat uit te voeren. In de hoop dit percentage te verhogen werd in een volgende stap een convolutional, en bijhorende maxpooling layer, toegevoegd aan het model.

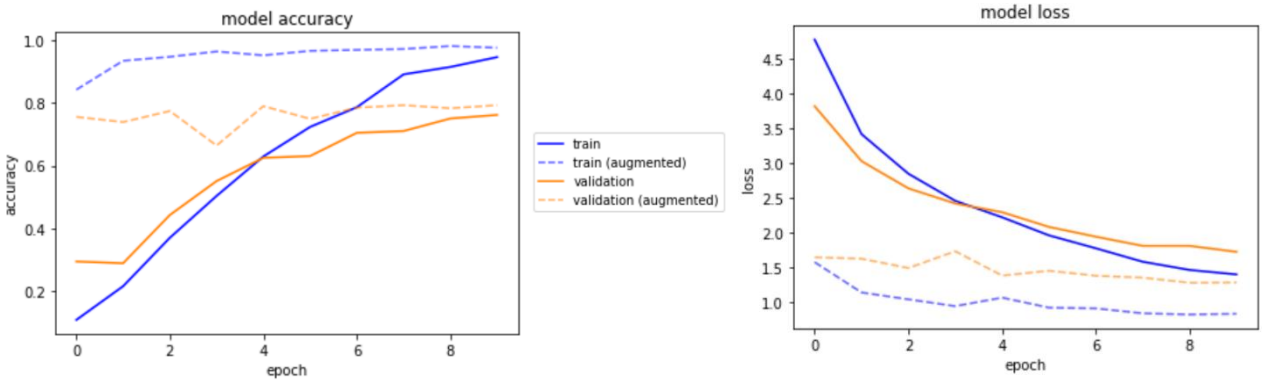
Codevoorbeeld 9 toont de aangepaste architectuur van het model. Merk op dat in beide dense layers een kernel_regularizer toegevoegd werd. Een regularizer voorkomt overfitting door constant de gewichten van de inkomende verbinding van een node te wijzigen. Zoals reeds aangehaald in sectie 2.2 bepaalt het gewicht van een verbinding de impact die een input heeft op de eventuele activatie van de node.

Indien een model lijdt aan overfitting zijn de gewichten van de dominante features vaak zeer hoog in vergelijking met de andere features, waardoor alleen deze dominante features een impact hebben. Het gebruik van een regularizer verzekert dat deze situatie zich niet voordoet.

Indien een feature meermaals ontdekt wordt en als gevolg hiervan een groot gewicht krijgt toegewezen zal de regularizer deze waarde inperken. Merk wel op dat indien de verbindingen gesorteerd zouden worden volgens gewicht, deze lijst dezelfde zou zijn voor en na het toepassen van de regularizer. Alhoewel de gewichten dus gewijzigd worden, blijven de verhoudingen min of meer dezelfde [6].

Figuur 31 toont de resultaten na het trainen en valideren van het model, welke duidelijk tonen dat deze wijziging in architectuur een stijging in training en validation accuracy met zich meebracht. Zo stegen de training en validation accuracy respectievelijk tot 89,23% en 69,87% zonder data augmentation en tot 89,21% en 68,85% met data augmentation. Merk dus ook op dat in dit geval data augmentation een negatief effect had op de performantie van het model.

5.3.2.3.2 Toevoegen meerdere convolutional en pooling layers + dropout layer



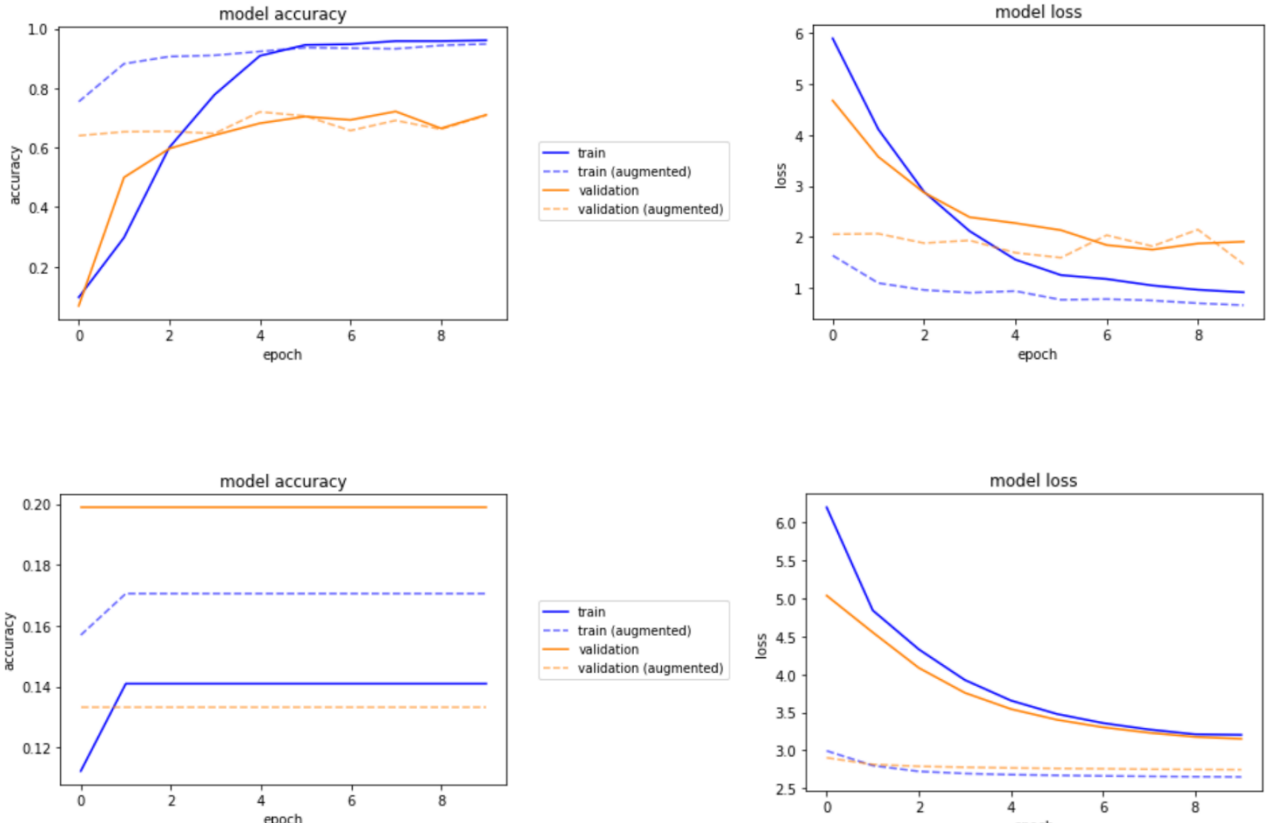
Figuur 32 : resultaten regressiemodel na toevoegen extra layers

In de hoop de trend voort te zetten werden nog enkele convolutional en pooling layers toegevoegd, alsook een dropout layer om overfitting tegen te gaan, zie codevoorbeeld 10 in de appendix. Zoals figuur 32 toont bleek dit een goede zet te zijn aangezien de trainig accuracy en validation accuracy respectievelijk stegen tot 94,52% en 76,14%. Na het toepassen van data augmentation stegen deze waarden nog verder naar 97,50% en 79,23%.

5.3.2.4 Dimensies kernel

In de hoop de validation accuracy te verbeteren werd getest welk effect de grootte van de kernel had op de performantie van het model. Zoals reeds aangehaald in sectie 2.3.1 bepaalt de grootte van deze kernel tevens de grootte van de gebieden van de afbeelding waarin patronen worden gezocht. Hoe kleiner de grootte van deze kernel, hoe meer in detail de afbeelding wordt onderzocht.

De grootte van de kernel was tot nu steeds een 2x3 matrix. Het effect van de grootte van de kernel op de performantie van het model werd getest aan de hand van 6x9 en 12x18 matrix, beide resultaten zijn gegeven in figuur 33 .



Figuur 33 : resultaten regressiemodel na wijziging kerneldimensies

Een kernel met dimensies 6x9 had een klein negatief effect op de performantie van het model. De training en validation accuracy bedroegen 92,06% en 71,02% zonder data augmentation en 94,87% en 70,84% met.

Indien de kernel nog vergroot werd tot dimensies 12x18 werd duidelijk dat het model niet langer in staat was patronen te herkennen die uniek waren per wielerploeg. De training en validation accuracy daalden tot 14,10% en 19,89% zonder data augmentation en 17,06% en 13,31% met.

Uit deze resultaten kon de conclusie getrokken worden dat een kernel met dimensies 2x3 duidelijk de beste resultaten gaf, zoals ook wordt weergegeven in tabel 1.

Tabel 1 : relatie acc, loss tegenover kernel dimensies

Kernel grootte	acc	val_acc	acc (data augmentation)	val_acc (data augmentation)
2x3	94,52%	76,14%	97,50%	79,23%
6x9	92,06%	71,02%	94,87%	70,84%
12x18	14,10%	19,89%	17,06%	13,31%

Als volgt werd besloten om in de toekomst steeds een kernel met dimensies 2x3 te gebruiken.

5.4 Pose afhankelijkheid data

Ongeacht de architectuur van het model, de dimensies van de kernel, gekozen activatiefunctie of aanwezigheid van data augmentation bleek de validation accuracy nooit boven 80% uit te stijgen. Ondanks het feit dat 80% zeker geen slecht resultaat was, verbleekte dit toch tegenover de 97% training accuracy die gehaald werd.

Omwille van het feit dat het model reeds verschillende vormen had aangenomen en dit geen verbeterend effect had op deze validation accuracy, kon de wortel van het probleem alleen nog maar bij de dataset liggen. Aldus werd de dataset in meer detail bekeken.

Prof.dr. Steven Verstockt merkte correct op dat een groot deel van de dataset bestaat uit afbeeldingen die het vooraanzicht van de renner tonen, ongeveer 75%. Figuur 26 in sectie 5.2.1 bevestigt deze vaststelling. Als gevolg hiervan bestond het trainen van het model vooral uit het trainen van het model in herkennen van afbeeldingen waarin het vooraanzicht van de renners werd getoond. Met als gevolg dat indien het model gevraagd werd een renner te herkennen die in zij- of achteraanzicht getoond werd, de voorspelling (hoogstwaarschijnlijk) minder accuraat was.

Opdat de validation accuracy dus omhooggehaald kon worden, diende de indeling van de dataset gewijzigd te worden, rekening houdende met de manier waarop de renner afgebeeld werd op de afbeelding. Er was dus een techniek nodig die toeliet een groep afbeeldingen te sorteren op basis van de manier waarop de renner getoond werd. Opnieuw bleek Openpose het antwoord. Zoals reeds aangehaald tracht Openpose op elke gedetecteerde persoon een skelet af te beelden. Dit skelet bestaat uit maximaal 18 sleutelpunten, waarvan per sleutelpunten de coördinaten gekend zijn.

Op basis van deze sleutelpunten en coördinaten is het mogelijk om te detecteren op welke manier de renner getoond wordt. Zo zal een renner hoogstwaarschijnlijk in vooraanzicht getoond worden indien de relatieve breedte van de heupen tegenover de afbeeldingen hoog is.

Er werd een onderscheid gemaakt tussen 3 verschillende poses :

- front : vooraanzicht van de renner
- back : achteraanzicht van de renner
- rest : zijaanzicht of bovenaanzicht van de renner

Een eerste versie van het algoritme dat werd gebruikt om dit onderscheid te bekomen wordt getoond in codevoorbeeld 11. Merk op dat het detecteren van de 3 verschillende poses niet op basis van 3 verschillende collecties voorwaarden gebeurde, maar maar op basis van 1 groep voorwaarden. Het volstond namelijk om een groep voorwaarden te definiëren die, indien vervuld, verzekerden dat de renner in voor- of achteraanzicht werd getoond.

Het onderscheid tussen voor- en achteraanzicht is namelijk relatief eenvoudig te berekenen op basis van de coördinaten van de sleutelpunten, hier wordt echter nog op teruggekomen. Het definiëren van een groep voorwaarden voor de laatste pose, de 'rest' pose, was niet nodig aangezien de afbeeldingen die niet tot de 'front' of 'back' pose voorwaarden voldeden, wel tot deze categorie moesten behoren.

Het detecteren van de pose van de renner werd met andere woorden gereduceerd tot het detecteren of die renner in voor- of achteraanzicht werd getoond. De set voorwaarden die hiervoor werd gedefinieerd werd gebaseerd op de relatieve positie van de armen en benen van die renner.

Het detecteren van de relatieve positie van de benen en armen werd respectievelijk gebaseerd op de relatieve breedte van de knieën tegenover de afbeelding en de relatieve breedte van de ellebogen tegenover de afbeelding. In detail werkte het algoritme als volgt.

Eerst werd het verschil in x-coördinaten tussen de knieën berekend, waarna de helft van deze waarde werd bijgehouden als een soort marge. Op basis van deze berekende marge en een knie als basispunt werd een interval opgesteld. Dit interval, [x-coördinaat knie – marge : x-coördinaat knie + marge], definieerde met andere woorden twee uiterste grenzen waar binnen de heup en enkel langs hetzelfde been moesten geplaatst kunnen worden. Merk op dat dit proces uiteraard werd herhaald voor beide benen.

Een analoge redenering werd opgesteld voor de armen. Opnieuw werd het verschil tussen x-coördinaten van de ellebogen berekend, waarna een interval, [x-coördinaat elleboog – marge : x-coördinaat elleboog + marge], werd opgesteld. Beide armen, meer specifiek beide schouders en polsen, dienden te voldoen aan de voorwaarde die het interval opstelde.

Indien een skelet voldeed aan reeds vernoemde voorwaarden kon met uitsluitsel bevestigd worden dat de renner of in vooraanzicht of in achteraanzicht op de afbeelding werd getoond. Het uitsluitsel tussen beide mogelijkheden werd bekomen op basis van de marge die eerder berekend werd bij beide de benen en armen. Deze marge werd steeds berekend als het verschil tussen het linker- en rechtergewricht, in die volgorde. Indien een renner in vooraanzicht getoond werd, was dit verschil steeds positief. Indien de renner echter in achteraanzicht getoond werd, was dit verschil negatief. Op basis hiervan kon uitsluitsel gegeven worden over de pose van de renner.

In een eerste test van dit algoritme, werd zeer goed gescoord op het detecteren van de 'front' pose, maar beduidend minder goed op het detecteren van de 'back' pose. Tabel 2 toont de resultaten van deze eerste test.

Tabel 2 : accuraatheid eerste versie openpose

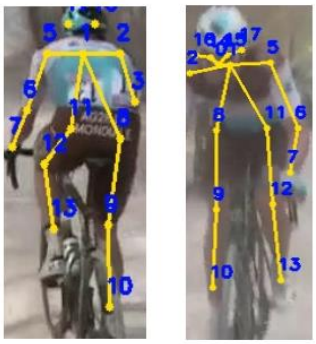
	'Front' pose	**'Back' pose**
Totaal aantal afbeeldingen	38	50
Gedetecteerde afbeeldingen	35	15
Succespercentage	92,11%	30%

Het verschil in succespercentage tussen beide poses was opmerkelijk, aangezien deze volgens dezelfde set voorwaarden werden gedetecteerd. Opdat een beter beeld kon verkregen worden over het probleem werden de foto's die de renner in achteraanzicht tonen in meer detail bekeken. Figuur 34 toont een onderscheid tussen de gedetecteerde en niet gedetecteerde afbeeldingen, de vier meest linkse afbeeldingen werden gedetecteerd, de vier meest rechtse niet.

Figuur 34 : onderscheid gedetecteerde en niet gedetecteerde afbeeldingen openpose

Bovenstaande figuur toont duidelijk dat de afbeeldingen duidelijk renners in achteraanzicht tonen, waarbij geen opmerkelijk verschil bestaat tussen de afbeeldingen uit beide groepen. De reden waarom het succespercentage dus zo sterk verschilde lag dus wel degelijk aan de set voorwaarden.

Vervolgens werd de set voorwaarden in detail overlopen, waarna het probleem relatief snel gevonden werd. De set voorwaarden hield geen rekening met de spiegeling van de sleutelpunten indien een renner in achteraanzicht getoond werd. Figuur 35 verduidelijkt dit.



Figuur 35 : vergelijking skelet openpose front pose en back pose

De voorwaarden moesten dus bijgewerkt worden, codevoorbeeld 12 toont de nieuwe set voorwaarden. Hierin werden twee grote wijzigingen doorgevoerd.

Enerzijds werd de set voorwaarden gedupliceerd, waarbij een licht aangepaste versie werd gebruikt om te detecteren of de renner in achteraanzicht werd getoond. Anderzijds werd ook het interval waartussen de gewrichten dienen te vallen, licht vergroot.

De resultaten van een eerste test van dit aangepaste algoritme, weergegeven in tabel 3,toonde duidelijk dat de ‘back’ pose veel beter gedetecteerd werd, zonder verlies in het detecteren van de ‘front’ pose.

Tabel 3 : accuraatheid tweede versie openpose

	‘Front’ pose	**‘Back’ pose**
Totaal aantal afbeeldingen	38	50
Gedetecteerde afbeeldingen	35	44
Succespercentage	92,11%	88%

Figuur 36 : voorbeelden niet gedetecteerde afbeeldingen openpose

Figuur 36 toont voor beide poses de afbeeldingen die niet gedetecteerd werden, waarbij ook het skelet dat Openpose op deze afbeeldingen afbeeldde, getoond wordt.

De 3 meest linkse afbeeldingen waren de enige afbeeldingen die de renner in vooraanzicht tonen en niet gedetecteerd werden, de reden hiervoor is simpelweg het feit dat niet alle sleutelpunten van armen en benen gedetecteerd werden. Als volgt had het algoritme geen kans om de set voorwaarden te controleren, waardoor de afbeelding niet gedetecteerd werd.

De 6 meest rechtse afbeeldingen tonen de afbeeldingen die de renner in achteraanzicht tonen en niet gedetecteerd werden. De 4 meest rechtse afbeeldingen hiervan werden niet gedetecteerd omwille van dezelfde reden waarom de afbeeldingen die de renner in vooraanzicht tonen niet gedetecteerd werden. Er werden niet genoeg sleutelpunten gedetecteerd om de set voorwaarden te controleren.

De twee meest linkse afbeeldingen van deze groep tonen echter wel genoeg sleutelpunten in de benen om gedetecteerd te worden. De set voorwaarden werd met andere woorden overlopen, maar niet voldaan. De reden hiervoor is simpelweg dat net op dit frame de renner de knie boog, waardoor het sleutelpunt van de knie enerzijds de marge verkleinde en anderzijds het basispunt van het interval verschoof. Hierdoor konden de heupen en enkels niet in het interval geplaatst worden.

Opdat deze twee laatst vernoemde afbeeldingen gedetecteerd zouden kunnen worden zou het algoritme, meer bepaald het interval waarin de sleutelpunten verwacht worden te liggen, aangepast moeten worden. Het interval zou vergroot moeten worden. Het vergroten van dat interval zou uiteraard als direct gevolg hebben dat meer afbeeldingen gedetecteerd worden als een afbeelding die een renner in ‘front’ of ‘back’ pose tonen, waarbij niet elke afbeelding ook effectief gedetecteerd zou mogen worden. Als gevolg hiervan werd besloten het interval niet te vergroten.

De 7 andere afbeeldingen die getoond worden, werden niet gedetecteerd als een direct gevolgd van een tekort aan sleutelpunten. Als gevolg hiervan werd het algoritme nog een laatste maal aangepast, waarbij dit soort situaties, gekenmerkt door een kleiner aantal sleutelpunten, opgevangen werd door een aparte set voorwaarden. Codevoorbeeld 13 toont deze extra set voorwaarden.

In dit codevoorbeeld wordt voor de armen en voor de benen een extra subset voorwaarden gedefinieerd. Het verschil is echter dat in dit geval geen rekening gehouden wordt met de polsen en enkels, aangezien indien maar enkele sleutelpunten gedetecteerd werden de polsen en enkels vaak niet gedetecteerd werden.

Simpelweg verwoord was deze extra set voorwaarden een soort back up voor de reeds besproken set voorwaarden. Aangezien het geen effect had op vooraf gedefinieerde voorwaarden kon het resultaat qua hoeveelheden gedetecteerde afbeeldingen in theorie niet slechter zijn. Er bestond echter wel de mogelijkheid dat bepaalde afbeeldingen gedetecteerd werden, die niet gedetecteerd mochten worden. Tabel 4 toont de resultaten van dit aangepast algoritme.

Tabel 4 : accuraatheid laatste versie openpose

	'Front' pose	**'Back' pose**
Totaal aantal afbeeldingen	38	50
Gedetecteerde afbeeldingen	43	67
Succespercentage	113,16%	134%

Zoals de resultaten tonen werd dit ook werkelijkheid, Figuur 37 toont de afbeeldingen die gedetecteerd werden als afbeeldingen die een renner in 'front' pose tonen.

Figuur 37 : voorbeelden verkeerd gedetecteerde afbeeldingen openpose

Omwille van deze fout werd besloten om deze laatste aanpassing aan het algoritme niet door te voeren, Aldus werd het algoritme zoals getoond in codevoorbeeld 12 gebruikt om de dataset op te splitsen volgens pose. Aangezien deze een benaderd succespercentage van 90% kon voorleggen, was deze ook toepasbaar in de praktijk.

5.5 Eindresultaten

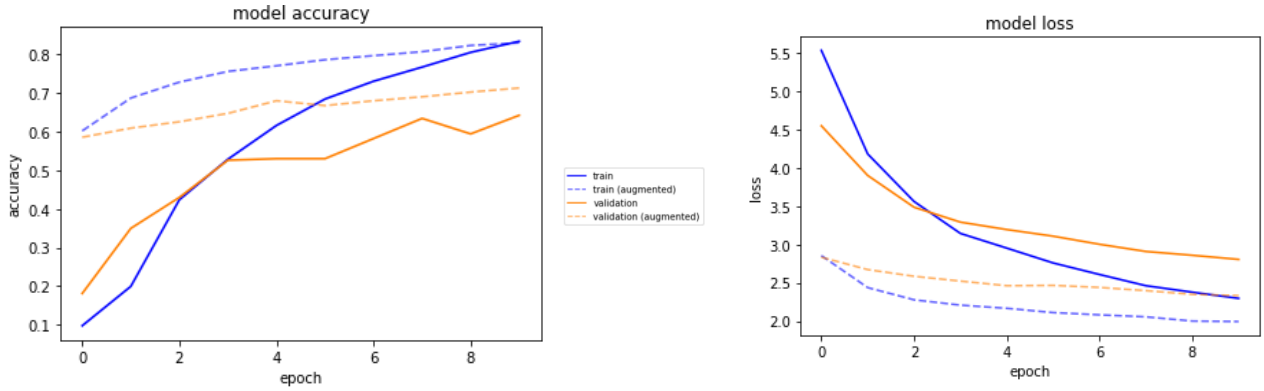
Om het effect van het opsplitsen van de data maximaal te kunnen observeren, werd besloten om de bestaande dataset te transformeren in 4 nieuwe datasets. 3 Van deze 4 datasets bevatten maar een deel van de afbeeldingen van elke wielerploeg. Zo bevatte één dataset alle afbeeldingen die een renner in front pose toonden, een andere alle afbeeldingen die een renner in back pose toonden en ten laatste een dataset die alle afbeeldingen bevatte van renners die in de rest pose getoond werden.

De vierde dataset bevatte 54 klassen. Elke wielerploeg werd niet langer vertaald naar 1 klasse, maar naar 3 klassen. Hierbij bestond per wielerploeg een klasse voor de renners van die wielerploeg die in front, back of rest pose getoond werden. Deze laatste dataset was met andere woorden de combinatie van de 3 kleinere datasets.

Alhoewel deze laatste dataset in theorie volstond, werd dus toch gekozen om 4 aparte datasets aan te maken. De drie eerstgenoemde datasets, elk dus bestaande uit 18 pose specifieke klassen, werden niet aangemaakt met als doel het trainen van een model dat later zou kunnen worden toegepast in real time situaties. Het doel van deze datasets was het dienen als hulpmiddel voor het analyseren van de resultaten van het model, na trainen op de grotere dataset.

Indien het model getraind werd op deze grotere dataset en de resultaten tegenvallend bleken, zouden de resultaten na het trainen van het model op de 3 kleinere datasets een beter beeld kunnen geven van wat precies misliep.

5.5.1 Resultaten



Figuur 38 : resultaten regressiemodel na manipuleren data volgens pose

Figuur 38 toont de resultaten na het trainen van het model op de grotere dataset, bestaande uit 54 klassen. Na het uitvoeren van deze test waren de training en validation accuracy gestegen tot respectievelijk 83,44% en 64,26% zonder data augmentation, en tot 83,04% en 71,34% met data augmentation.

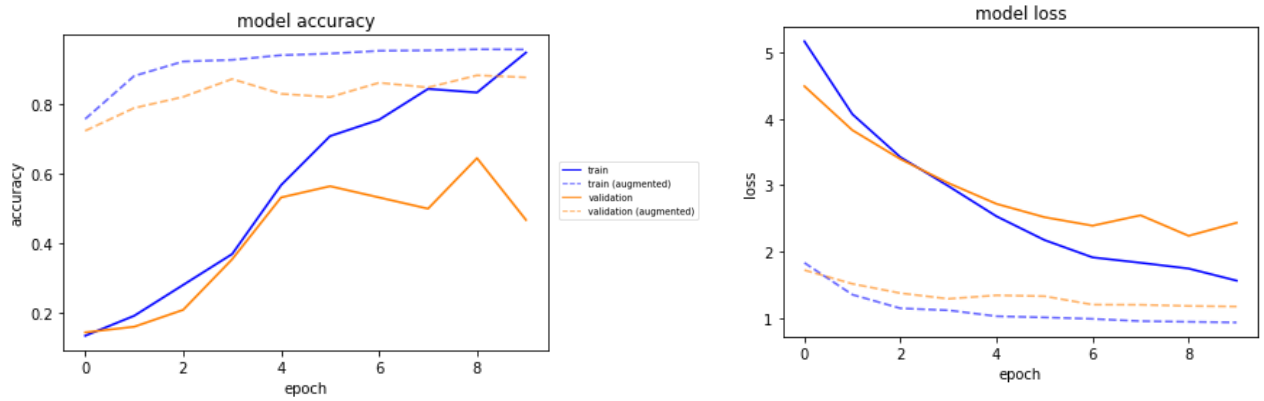
Opmerkelijk aan deze resultaten is dat deze lager liggen dan de behaalde resultaten, toen de dataset nog niet opgesplitst werd op basis van pose (zie sectie 5.3.2.3.2).

5.5.2 Evaluatie en oorzaken tegenvallende resulaten

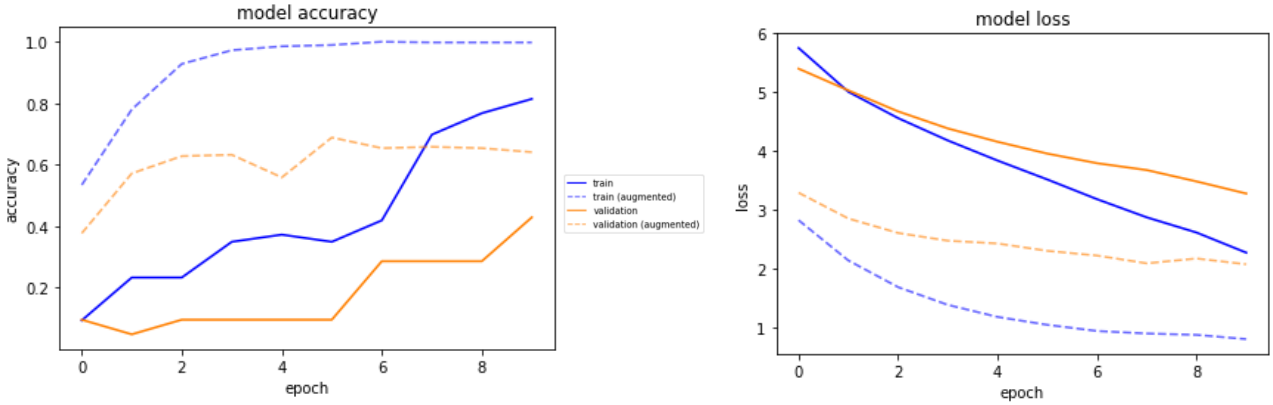
Zoals figuur 38 toont daalde de validation accuracy tijdens het trainen van het model, ongeacht of data augmentation werd toegepast of niet. Enigszins verassend is dat deze dalende trend niet voortgezet werd, maar dat de validation accuracy opnieuw steeg. Aangezien deze dalende trend zich dus niet voortzette kon niet dezelfde conclusie genomen worden als in sectie 4.6.

Het fluctueren van deze validation accuracy duidde erop dat een deel van de samples niet consistent werd toegewezen. Een deel van de samples werd overduidelijk wel correct toegewezen aangezien de validation accuracy minstens 60% bedroeg eenmaal deze begon te fluctueren. Een ander deel werd overduidelijk foutief toegewezen aangezien de validation accuracy niet 100% bedroeg. Een specifiek deel van de samples werd soms correct, soms foutief toegewezen, dit deel van de samples zorgde voor de fluctuatie in validation accuracy.

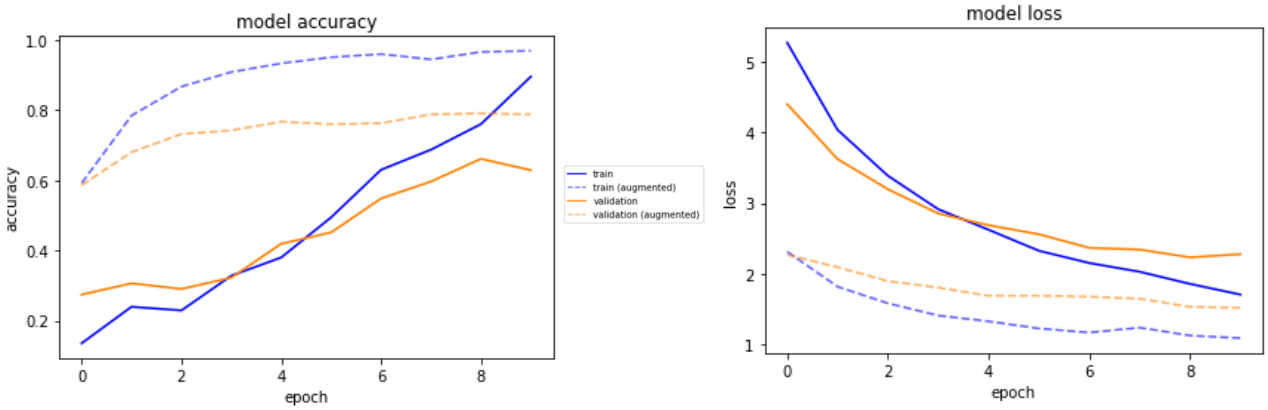
Om een beter idee te krijgen van welke samples zorgden voor deze fluctuatie in validation accuracy, werd het model nogmaals getraind. Ditmaal was de input data de 3 pose specifieke datasets. De resultaten hiervan zijn weergegeven in onderstaande figuur 39 tot en met figuur 41.



Figuur 39 : resultaten regressiemodel voor data front pose



Figuur 40 : resultaten regressiemodel voor data back pose



Figuur 41 : resultaten regressiemodel voor data rest pose

Na inspecteren van bovenstaande grafieken werd duidelijk dat het fluctuerende gedrag van de validation accuracy grotendeels te wijten was aan de afbeeldingen die renners in vooraanzicht toonden, aangezien ook deze resultaten zeer sterk fluctueerden.

Zoals reeds vermeld lag de behaalde training en validation accuracy lager dan verwacht. De verwachting was namelijk dat indien de dataset opgedeeld werd op basis van pose dat de nauwkeurigheid zou verhogen. Deze accuracy daalde echter ongeveer met 15%. Om een overzicht te krijgen van de oorzaak hiervan werden de verschillende statistieken met elkaar vergeleken, zie tabel 5.

Tabel 5 : resultaten regressiemodel pose specifieke datasets

	Training	Validation	Training(aug)	Validation(aug)
Gecombineerd	83,44%	64,26%	83,04%	71,34%
Front	94,79%	46,77%	95,67%	87,86%
Back	81,40%	42,86%	99,79%	64,07%
Rest	89,58%	62,90%	96,95%	78,79%

Wanneer deze verschillende statistieken met elkaar werden vergeleken, werd snel duidelijk dat data augmentation een enorme verbetering met zich meebracht. Daarnaast werd het ook duidelijk dat het front pose gedeelte van de dataset veruit de beste resultaten garandeerde.

Beide het rest pose gedeelte en back pose gedeelte van de dataset toonden echter tegenvallende resultaten, zelfs na het toepassen van data augmentation. Hiervoor waren eenvoudig verklaringen te vinden.

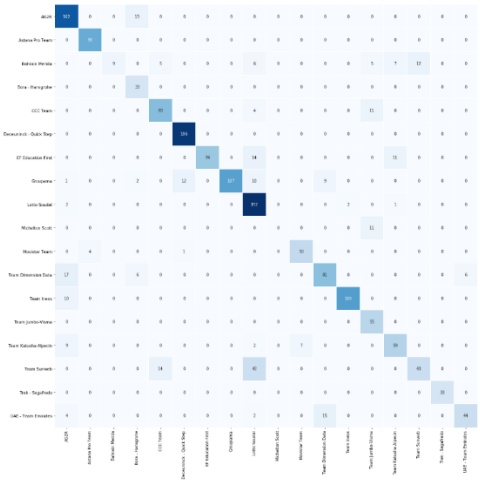
Het tegenvallende resultaat van het back pose gedeelte van de dataset is volledig te wijten aan een tekort aan data. Zo bevatte elke klasse, of dus elk team, voor het toepassen van data augmentation ongeveer 2 à 3 afbeeldingen. Dit tekort aan data zorgde ervoor dat overfitting relatief snel een negatieve impact had op

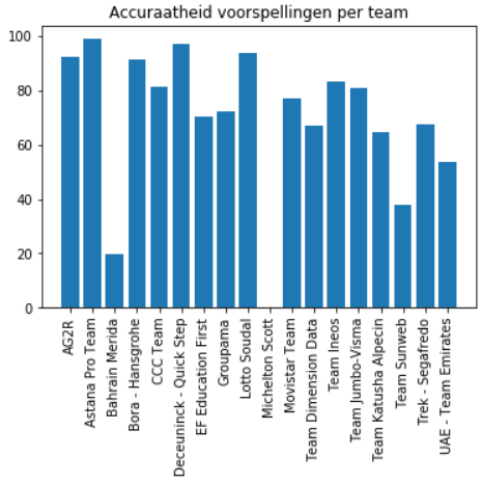
het leerproces. Figuur 40 ondersteunt dit ook door het zichtbaar stagnerende gedrag van de training en validation accuracy.

De verklaring voor het teleurstellende resultaat van het rest pose gedeelte van de dataset lag niet alleen aan de kwantiteit van de data, maar ook aan de kwaliteit. Aangezien dit gedeelte van de dataset elke afbeelding bevatte die niet als front of back pose gedetecteerd werd, bevatte dit gedeelte dus een zeer gevarieerde verzameling afbeeldingen. Zo waren sommige afbeelding duidelijk een zijaanzicht van de renner, een schuin vooraanzicht van de renner of eventueel zelfs een achteraanzicht van de renner dat niet gedetecteerd werd. Aangezien deze data zo gevarieerd was, was het detecteren van patronen allerminst eenvoudig. De combinatie met een tekort aan data zorgde voor een sterke daling in validation accuracy.

5.5.2.1 Confusion matrix

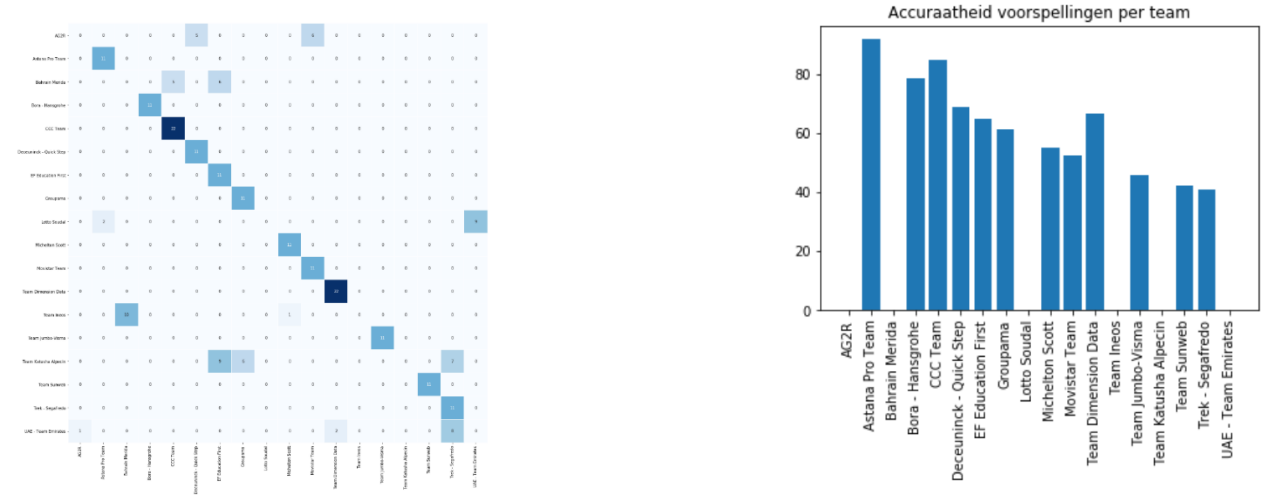
Om een beter beeld te krijgen van welke voorspellingen accuraat waren of niet, werd per dataset die getraind werd door het model een confusion matrix opgesteld. Een confusion matrix, steeds een vierkante matrix met dimensies aantal klassen x aantal klassen, toont vervolgens aan aan welke klassen de verschillende voorspellingen werden toegewezen. Hiernaast werd ook een grafiek geplot die per klasse de accuraatheid van de gemaakte voorspellingen toont.



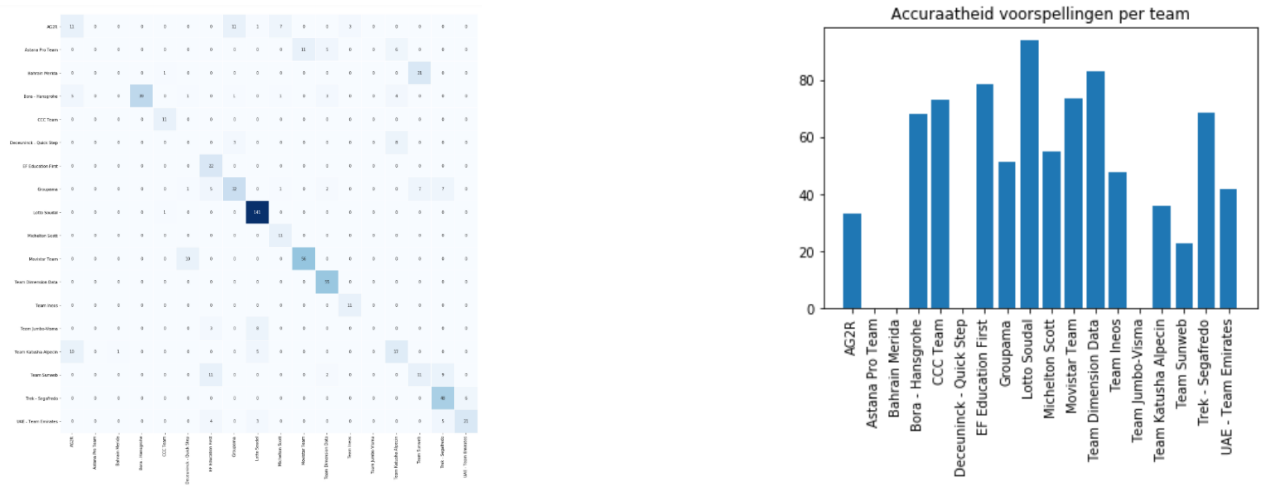


Figuur 422 : confusion matrix front pose

Figuur 42 tot en met 44, tonen de confusion matrix voor de voorspellingen die het model maakte op basis van de afbeeldingen die gedetecteerd werden als respectievelijk front, back en rest pose. De drie confusion matrices tonen elk hetzelfde terugkerende patroon, de hoofddiagonaal bevat de meeste waarden. Uiteraard is dit een positief teken, aangezien een correcte voorspelling van een afbeelding steeds leidt tot verhoging van de waarde van een cel op de hoofddiagonaal.

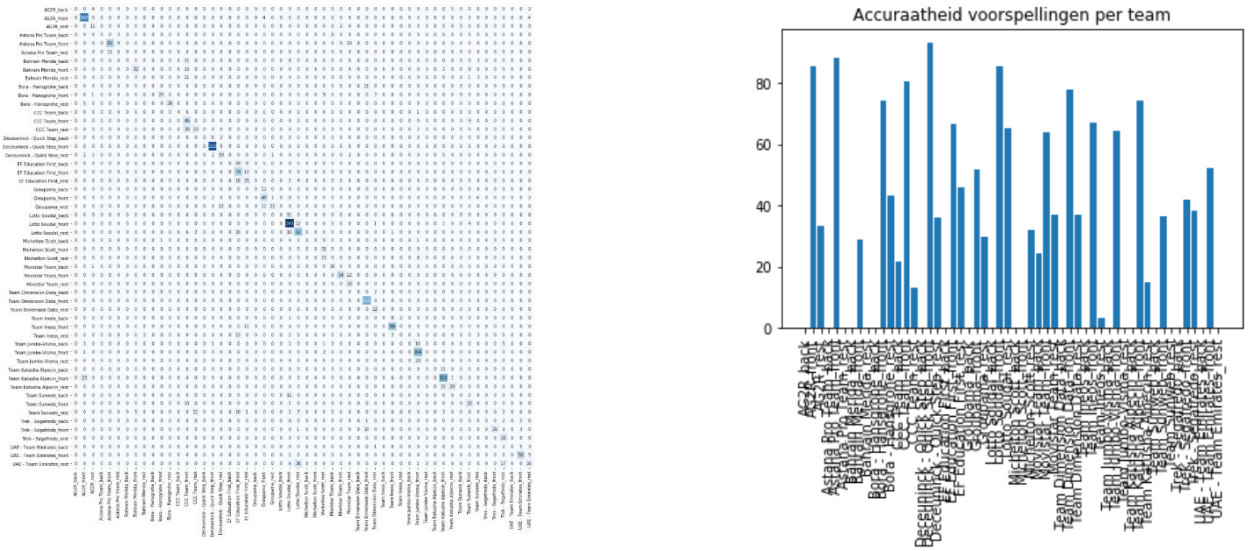


Figuur 43 : confusion matrix back pose



Figuur 444 : confusion matrix rest pose

Omdat een confusion matrix met een relatief hoog aantal klassen, zoals in dit geval, eerder dient om een globaal te verkrijgen, werd ook een grafiek geplot die per klasse de nauwkeurigheid van de voorspellingen toont. Indien deze corresponderende grafieken van dichterbij worden bekeken valt op dat de voorspellingen voor afbeeldingen die gedetecteerd werden als front pose opmerkelijk hoger liggen. Tevens valt op dat enkele klassen een accuraatheid van 0% bereikten. Dit duidt er op dat de afbeeldingen behorend tot deze klassen consequent aan de verkeerde klassen werden toegewezen.



Figuur 45 : confusion matrix overheen alle poses

Figuur 45 ten laatste toont de confusion matrix in het geval waarbij 54 klassen werden aangewend. Opnieuw toont de confusion matrix de aanwezigheid van de hoofddiagonaal. De corresponderende grafiek toont echter dat veel klassen niet gedetecteerd werden.

Hoofdstuk 6 : Conclusie en Reflectie

6.1 Conclusie

Het doel van deze masterproef was het ontwerpen en trainen van een CNN dat toeliet renners te detecteren en onderscheiden op basis van wielertenue. Een renner werd hierbij toegewezen aan één van 18 mogelijke klassen, waarbij elke klasse overeenkwam met een wielerploeg uit de UCI World Tour wielercompetitie.

In een eerste fase van de masterproef werd een classificatiemodel ontworpen, welk toeliet een renner toe te wijzen aan de correcte klasse. Als gevolg van teleurstellende resultaten werden meerdere aanpassingen gemaakt aan het gebruikte classificatiemodel. Deze aanpassingen varieerden van het aantal lagen tot individuele parameters, zoals een gebruikte activatiefunctie. Hiernaast werden ook aanpassingen gemaakt aan de dataset, waarbij elke afbeeldingen die een renner toonde via YOLO werd bijgesneden om zo enkel die renner te tonen. Na deze aanpassingen werd een maximale accuraatheid van 73,91% bereikt. Deze accuraatheid kon echter niet behouden worden.

Vervolgens werd in een tweede fase besloten om over te gaan van classificatie naar regressie. Aangezien dit inhield dat een voorspelling niet langer één klasse was, maar een verzameling probabiliteiten over 18 klassen heen, liet dit toe meer inzicht te krijgen in de accuraatheid van de voorspellingen.

Het regressiemodel was ook onderhevig aan meerdere wijzigingen. Daar waar de wijzigingen aan het classificatiemodel het directe gevolg waren van tegenvallende resultaten, waren de meeste wijzigingen aan het regressiemodel een poging in het verder verbeteren van een reeds voldoende accuraatheid.

Na meerdere aanpassingen aan het regressiemodel en de bijhorende dataset werd een maximale accuraatheid van 79,23% bereikt.

In een laatste fase en poging de accuraatheid verder te verhoging werd de dataset verder opgedeeld naargelang de pose van de renner in de afbeelding. Er werd een onderscheid gemaakt tussen 3 poses. De ‘front’ pose die een renner in vooraanzicht toonde, de ‘back’ pose die een renner in achteraanzicht toonde en de ‘rest’ pose die alle andere gevallen omvatte. Alhoewel deze aanpassing aan de dataset verwacht werd een positief effect te hebben op de accuraatheid, daalde de accuraatheid met ongeveer 15%. De reden hiervoor was een tekort aan data.

Uit deze resultaten kan de conclusie getrokken worden dat alhoewel een maximale accuraatheid van 79,23% bereikt werd, deze hoogstwaarschijnlijk verhoogd kan worden na het opstellen van een grotere, meer representatieve dataset.

6.2 Reflectie

6.2.1 Tekortkomingen

Ongeacht het beschouwde model is elk model onderhevig aan enkele tekortkomingen. Zoals in sectie 3.0 reeds aangehaald werd, werd vooraf beslist om twee beperkingen op te leggen aan de inhoud van de dataset. Deze beperkingen brengen hun tekortkomingen met zich mee.

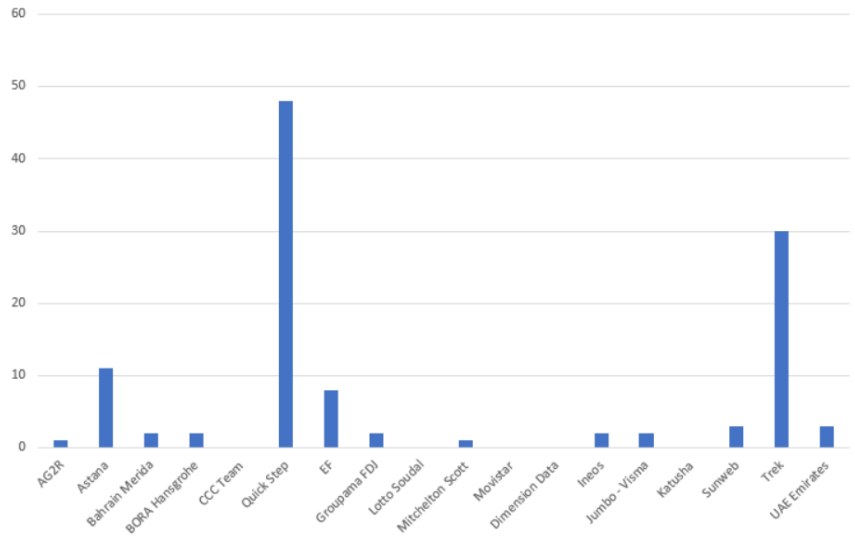
Enerzijds werd besloten de dataset te beperken tot renners die uitkwamen voor een team in de UCI World Tour. Als gevolg hiervan is het onmogelijk om een renner, die uitkomt voor een team dat zich niet in de UCI World Tour bevind, toe te wijzen aan zijn corresponderend team.

Anderzijds werd besloten om het herkennen van de renner te baseren op het tenue van de renner en niet op het volgnummer. Als gevolg hiervan is het onmogelijk om de identiteit van een renner te achterhalen.

Zoals reeds aangehaald in sectie 4.3.2.2 is het mogelijk dat een renner in een wedstrijd een speciale versie van het tenue van zijn wielerteam draagt. De aanwezigheid van speciale versies van truien, zoals een leiderstrui of nationale kampioenetrui, is niet opgevangen in de dataset. Als direct gevolg hiervan zal het voorspellen van een wielerteam van een renner die een speciale versie van het tenue van zijn wielerteam draagt, hoogstwaarschijnlijk inaccuraat zijn.

6.2.2 Toepassingen

Ondanks het feit dat het model in het beste geval slechts een accuraatheid van 80% bereikte, kent het verschillende mogelijke toepassingen.



Figuur 46 : frequentietabel renners laatste 40 km Clásica San Sebastián

Een eerste mogelijke toepassing is het opstellen van een frequentietabel van de gedetecteerde renners per team, zoals weergegeven in figuur 46. Een frequentietabel toont niet alleen interessante informatie, maar kan ook gebruikt worden voor commerciële doeleinden.

Het is bijvoorbeeld mogelijk om de inkomsten van televisierechten van een bepaalde uitzending van een wielerwedstrijd volgens de verhoudingen weergegeven in deze frequentietabel te verdelen. Op deze manier krijgt een team meer geld indien het meer in beeld is geweest, wat uiteraard eerlijker is dan het geld gelijk verdelen over alle teams die deelnamen aan die wedstrijd.

Een andere mogelijke toepassing, specifiek in het kader van de teamsponsors, is het gebruiken van deze data om doelstellingen te stellen. Een teamsponsor zoals Lotto hecht meer waarde aan wielerwedstrijden op eigen bodem, goede voorbeelden zijn de Ronde van Vlaanderen, E3 Harelbeke of Gent Wevelgem. In deze wielerwedstrijden verwacht de teamsponsor toch een bepaald deel van de uitzending in beeld te zijn, bij voorkeur in het latere deel van de uitzending aangezien dan meer mensen de wielerwedstrijd bekijken. Een teamsponsor zoals Lotto zou bijvoorbeeld de voorwaarde kunnen stellen dat het team minstens in de top 3 van de meest gedetecteerde teams moet voorkomen. Op basis van data, zoals getoond in figuur 46, kan deze voorwaarde achteraf gecontroleerd worden, waarna eventueel een financiële beloning of straf volgt.

Referenties

[1] S. Jabeen, S. D. Patil, S. V. Bhosale, M. C. Bharati en P. S. Patil, „A Study On Basics Of Neural Network," Pune, 2017.

[2] K. T. O'Shea, „An Introduction To Convolutional Neural Networks," Aberystwyth University, 2015.

[3] F. S. Panchal en M. Panchal, „Review on Methods of Selecting Number of Hidden Nodes in Artificial Neural Network," 2014.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever en R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting," University of Toronto, 2014.

[5] C. Enyinna Nwankpa, W. Ijomah, A. Gachagan en S. Marshall, „Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," 2018.

[6] K. Yu, W. Xu en Y. Gong, „Deep Learning with Kernel Regularization for Visual Recognition," NEC Laboratories America, Cupertino, CA 95014, USA.

Appendix

Codevoorbeeld 1

```
#Width and height of every image
img_width=120
img_height=180

# Checks the default Keras image data format convention and adapts to it
if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

# Defining a sequential model
model = Sequential()

model.add(Conv2D(32, (2, 2), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

#Compiling the model
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Codevoorbeeld 2

```
…

#Create object representing the flickrAPI
flickr = flickrapi.FlickrAPI(api_key, api_secret, cache=True)

teams=["AG2R", "Astana Pro Team", "Bahrain Merida", "BORA - Hansgrohe", "CCC Team",    "Deceuninck - Quick Step", "EF Education First", "Groupama", "Lotto Soudal","Michelton Scott", "Movistar Team", "Team Dimension Data","Team Jumbo Visma", "Team Katusha Alpecin", "Team Sky", "Team Sunweb", "Trek - Segafredo", "UAE - Team Emirates"]

for team in teams :
   tags = 'UCI World Tour, UCI, 2018, 2019, cycling, wielrennen, radsport, cyclisme, sport '
   tags+=", " + team

   photos = flickr.walk(text=team,                    #Searches description, title of photo for this text.
                        tags=tags,                    #All possible tags, if one is present it suffices
                        extras='URL_c',                  #Extra data to be retrieved, in this case the URL
                        per_page=100,
                        sort='relevance',
                        min_taken_date='2019-01-01',
                        max_taken_date=datetime.datetime.today().strftime('%Y-%m-%d'),
                       )

   for i, photo in enumerate(photos):

      if photo.get('URL_c') is not None :              #Only if URL is present
         URL = photo.get('URL_c')
         URLlib.request.URLretrieve(URL, dir + '/' + team + '/' + team + '' + str(i) + '.jpg')
```

Codevoorbeeld 3

```
…

api_key = 'AIzaSyBllpIFIgo_P7Lm8tJaMin_rk1kPOriiAI' #API key, requested via google
service = build('customsearch', 'v1', developerKey=api_key) #Create object representing googleAPI service

teams=["AG2R La Mondiale", "Astana Pro Team", "Bahrain Merida", "BORA - Hansgrohe", "CCC Team",
"Deceuninck - Quick Step", "EF Education First", "Groupama", "Lotto Soudal","Michelton Scott", "Movistar
Team", "Team Dimension Data", "Team Jumbo Visma", "Team Katusha Alpecin", "Team Sky", "Team
Sunweb", "Trek - Segafredo", "UAE - Team Emirates"]

for team in teams :
    i=0
    res = service.cse().list(
            q=team + ' rider cyclist 2019',
            cx='017666544973388121336:wsbvq9h3euy',
            searchType='image',
            num=10,
            imgType=image,
            fileType='png',
            safe= 'off'
        ).execute()

    if not 'items' in res:
        print ('No result !!\nres is: {}'.format(res))

    else:
        for item in res['items']:
            try :
                i+=1
                URLlib.request.URLretrieve(item['link'], dir + '/' + team + '/' + team + str(i) + '.jpg')

            except Exception as e:
                print ("could not load ( ", e, ") :\n", item['link'])
```

Codevoorbeeld 4

```
#Function returns BeautifulSoup object that scrapes for images
def get_soup(URL,header):
    return
BeautifulSoup(URLlib.request.URLopen(URLlib.request.Request(URL,headers=header)),'html.parser')

teams=["AG2R La Mondiale", "Astana Pro Team", "Bahrain Merida", "BORA - Hansgrohe", "CCC Team",
"Deceuninck - Quick Step", "EF Education First", "Groupama", "Lotto Soudal","Michelton Scott", "Movistar
Team", "Team Dimension Data", "Team Jumbo Visma", "Team Katusha Alpecin", "Team Sky", "Team
Sunweb", "Trek - Segafredo", "UAE - Team Emirates"]

for team in teams :
    query = team + " 2019"

    image_type="ActiOn"

    #Split multiple words and concatenate using + sign (URL requirement)
    query= query.split()
    query='+'.join(query)

    #Form URL
    URL="https://www.google.co.in/search?q="+query+"&source=lnms&tbm=isch"

    #Define header
    header={'User-Agent':"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/43.0.2357.134 Safari/537.36"
    }

    #Get soup object using pre defined function
    soup = get_soup(URL,header)

    #Scrape the HTML for images and preserve the links of those images in the array 'ActualImages'
    ActualImages=[]
    for a in soup.find_all("div",{"class":"rg_meta"}):
        link, Type =json.loads(a.text)["ou"],json.loads(a.text)["ity"]
        ActualImages.append((link,Type))
    print  ("(", team, ") Total of ", len(ActualImages)," images")

    #Save images (scraping itself)
    for i, (URL, type) in enumerate( ActualImages):
        try :
            URLlib.request.URLretrieve(URL, dir + '/' + team + '/' + str(i) + '.jpg')
        except Exception as e:
            print ("could not load ( ", e, ") :\n", URL)
```

Codevoorbeeld 5

```
def check_vicinity_riderbike(box_rider, box_bike):
    ridertop, riderleft, riderbottom, riderright = box_rider
    biketop, bikeleft, bikebottom, bikeright = box_bike

    ridercenter_hor = (riderleft + riderright)/2
    ridercenter_ver = (ridertop + riderbottom)/2

    if bikeleft < ridercenter_hor and bikeright > ridercenter_hor :
        if bikebottom > ridercenter_ver:
            return True

    return False

def rider_visibility( boxcoords, imagesize):
    #top, left, bottom, right
    width, height = imagesize

    width_ratio=(boxcoords[3]-boxcoords[1])/width
    height_ratio=(boxcoords[2]-boxcoords[0])/height
    #print("Width ratio = ", width_ratio, " Height ratio = ", height_ratio)

    if width_ratio > 0.05 and height_ratio > 0.2 :
        return True

def predict(sess, image_file):
    image, image_data = preprocess_image(image_file, model_image_size = (608, 608))
    out_scores, out_boxes, out_classes = sess.run([scores, boxes, classes],
                                                  feed_dict={yolo_model.input: image_data, K.learning_phase(): 0})

    if 0 in out_classes and 1 in out_classes : #Means a person and bike are present

        riders=[]
        bikes=[]

        for i,c in enumerate(out_classes):
            if c==0 :  #It's a rider, we save the corresponding index to the array riders
                riders.append(i)
            if c==1 :  #It's a bike, we save the corresponding index to the array bikes
                bikes.append(i)


        riderscrop=[]

        for riderindex in riders :
            for bikeindex in bikes :

                #Save parameters
```

```
            ridertop, riderleft, riderbottom, riderright = out_boxes[riderindex]
            biketop, bikeleft, bikebottom, bikeright = out_boxes[bikeindex]

            if check_vicinity_riderbike(out_boxes[riderindex],out_boxes[bikeindex]) :
                riderscrop.append(riderindex)

    for riderindex in riderscrop :
        if rider_visibility(out_boxes[riderindex], image.size) :
            cropped = image.crop((out_boxes[riderindex][1],out_boxes[riderindex][0],
                                  out_boxes[riderindex][3],out_boxes[riderindex][2]))



            if not os.path.exists(dir_write):
                os.makedirs(dir_write)

            cropped.save(dir_write + "\\" + str(counter) + ".jpg")

            counter+=1

os.remove(image_file)
```

Codevoorbeeld 6

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(img_width,img_height,3)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(18, activation=tf.nn.relu)
])
```

Codevoorbeeld 7

```
#Voorbereidend werk

net = …
checkpoint = …
load_state(net, checkpoint)

detect_riders(net,dir_read,256,False)

---------- detect_riders(net,dir_read,256,False)  ----------

    …
    frame_number = 0

    for (dirpath, dirnames, filenames) in os.walk(dir_read):
        for dirname in dirnames :  #Loop over all directories in the main directory
            for (dirpath, dirnames, filenames) in os.walk(dir_read + "/" + dirname):
                for filename in filenames:  #Loop over all files in the current directory
                    …

                    #Loop over all connections of joints (part = connection of two joints)
                    all_valid_points = [0]*18

                    for part_id in range(len(BODY_PARTS_PAF_IDS) - 2):
                        #Start joint
                        kpt_a_id = BODY_PARTS_KPT_IDS[part_id][0]
                        global_kpt_a_id = pose_entries[0][kpt_a_id]

                        if global_kpt_a_id != -1:
                            x_a, y_a = all_keypoints[int(global_kpt_a_id), 0:2]
                            cv2.circle(img, (int(x_a), int(y_a)), 3, color, -1)

                            all_valid_points[kpt_a_id]=[
                                                global_kpt_a_id, x_a, y_a,   all_keypoints[int(global_kpt_a_id), 2]]
                        else :
                            all_valid_points[kpt_a_id]=[global_kpt_a_id, -1, -1, 0]

                        #End joint
                        kpt_b_id = BODY_PARTS_KPT_IDS[part_id][1]
                        global_kpt_b_id = pose_entries[0][kpt_b_id]
```

```
                #If the joint has been located
                if global_kpt_b_id != -1:
                    x_b, y_b = all_keypoints[int(global_kpt_b_id), 0:2]

                    cv2.circle(img, (int(x_b), int(y_b)), 3, color, -1)
                    all_valid_points[kpt_b_id]=[global_kpt_b_id, x_b, y_b, all_keypoints[int(global_kpt_b_id),
2]]
                else :
                    all_valid_points[kpt_b_id]=[global_kpt_b_id, -1, -1, 0]

                #Combine both joints to form a part (connection shown on figure)
                if global_kpt_a_id != -1 and global_kpt_b_id != -1:
                    cv2.line(img, (int(x_a), int(y_a)), (int(x_b), int(y_b)), color, 2)
```

Codevoorbeeld 8

```
datagen = ImageDataGenerator(
        rotation_range=40,
        rescale=1./255,
        horizontal_flip=True,
        fill_mode='reflect'
        )
```

Codevoorbeeld 9

```
model = keras.Sequential([
    keras.layers.Conv2D(kernel_size = (2,3), filters = 16,activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None),
    keras.layers.Flatten(input_shape=(img_width,img_height,3)),
    keras.layers.Dense(128, activation=tf.nn.relu, kernel_regularizer=regularizers.l2(0.01)),
    keras.layers.Dense(18, activation=tf.nn.softmax, kernel_regularizer=regularizers.l2(0.01))
])
```

Codevoorbeeld 10

```
model = keras.Sequential([
    keras.layers.Conv2D(kernel_size = (2,3), filters = 16,activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None),
    keras.layers.Conv2D(kernel_size = (2,3), filters = 32,activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None),
    keras.layers.Conv2D(kernel_size = (2,3), filters = 64,activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None),
    keras.layers.Dropout(rate=0.5),
    keras.layers.Flatten(input_shape=(img_width,img_height,3)),
    keras.layers.Dense(128, activation=tf.nn.sigmoid, kernel_regularizer=regularizers.l2(0.01)),
    keras.layers.Dense(18, activation=tf.nn.softmax, kernel_regularizer=regularizers.l2(0.01))
])
```

Codevoorbeeld 11

```
def check_frontback(width,height,all_valid_points):

    #Check x-coordinaten benen
    try :
        #BENEN
        base1=int(all_valid_points[9][1])
        base2=int(all_valid_points[12][1])
        margin=int((base2-base1)/2)
        back=True if margin<0 else False
        print("base1 = ", base1, "\tbase2 = ", base2, "\tmargin = ", margin)

        margin=abs(margin)
        print("margin after abs = ", margin)
        if base1-margin <= all_valid_points[8][1] <= base1+margin and
           base1-margin <= all_valid_points[10][1] <= base1+margin :
            if base2-margin <= all_valid_points[11][1] <= base2+margin and
               base2-margin <= all_valid_points[13][1] <= base2+margin :
                return 'back' if back==True else 'front'

    except Exception as e :
        print(e)

    try :
        #ARMEN
        base1=int(all_valid_points[3][1])
        base2=int(all_valid_points[6][1])
        margin=int((base2-base1)/2)
        back=True if margin<0 else False
        print("base1 = ", base1, "\tbase2 = ", base2, "\tmargin = ", margin)

        margin=abs(margin)
        print("margin after abs = ", margin)

        if base1-margin <= all_valid_points[2][1] <= base1+margin and
           base1-margin <= all_valid_points[4][1] <= base1+margin :
            if base2-margin <= all_valid_points[5][1] <= base2+margin and
               base2-margin <= all_valid_points[7][1] <= base2+margin :
                return 'back' if back==True else 'front'

    except Exception as e :
        print(e)

    return ‘rest’
```

Codevoorbeeld 12

```
def check_frontback(width,height,all_valid_points):
    #Check x-coordinaten benen

    try :
        #BENEN
        base1=int(all_valid_points[9][1])
        base2=int(all_valid_points[12][1])
        margin=int((base2-base1)/2)
        back=True if margin<0 else False

        margin=abs(margin)

        #Toepassen voorwaarden
        if back==True:
            if base2 <= all_valid_points[8][1] <= base1+margin and
               base2 <= all_valid_points[10][1] <= base1+margin :
                if base2-margin <= all_valid_points[11][1] <= base1 and
                   base2-margin <= all_valid_points[13][1] <= base1 :
                    return 'back'
        else :
            if base1-margin <= all_valid_points[8][1] <= base2 and
               base1-margin <= all_valid_points[10][1] <= base2 :
                if base1 <= all_valid_points[11][1] <= base2+margin and
                   base1 <= all_valid_points[13][1] <= base2+margin :
                    return 'front'

    except Exception as e :
        print(e)


    try :
        #ARMEN
        base1=int(all_valid_points[3][1])
        base2=int(all_valid_points[6][1])
        margin=int((base2-base1)/2)
        back=True if margin<0 else False

        margin=abs(margin)

        #Toepassen voorwaarden
        if back==True:
            if base2 <= all_valid_points[2][1] <= base1+margin and
               base2 <= all_valid_points[4][1] <= base1+margin :
                if base2-margin <= all_valid_points[5][1] <= base1 and
                   base2-margin <= all_valid_points[7][1] <= base1 :
                    return 'back'
        else :
            if base1-margin <= all_valid_points[2][1] <= base2 and
```

```
            base1-margin <= all_valid_points[4][1] <= base2 :
             if base1 <= all_valid_points[5][1] <= base2+margin and
                base1 <= all_valid_points[7][1] <= base2+margin :
                 return 'front'

    except Exception as e :
        print(e)

    return None
```

Codevoorbeeld 13

```
try :
        #ARMENBENEN

        base1=int(all_valid_points[3][1])
        base2=int(all_valid_points[6][1])
        margin=int((base2-base1)/2)
        back=True if margin<0 else False

        margin=abs(margin)

        #Toepassen voorwaarden
        if back==True:
            if base2 <= all_valid_points[2][1] <= base1+margin and
               base2-margin <= all_valid_points[5][1] <= base1:
                  return 'back'
        else :
            if base1-margin <= all_valid_points[2][1] <= base2 and
               base1 <= all_valid_points[5][1] <= base2+margin:
                  return 'front'

        base1=int(all_valid_points[9][1])
        base2=int(all_valid_points[12][1])
        margin=int((base2-base1)/2)
        back=True if margin<0 else False

        margin=abs(margin)

        #Toepassen voorwaarden
        if back==True:
            if base2 <= all_valid_points[8][1] <= base1+margin  and
               base2-margin <= all_valid_points[11][1] <= base1:
                   return 'back'
        else :
            if base1-margin <= all_valid_points[8][1] <= base2 and
               base1 <= all_valid_points[11][1] <= base2+margin:
                  return 'front'

    except Exception as e :
        print(e)
```