

INHOLLAND

TECHNISCHE INFORMATICA

STAGE A

---

# Het maken van een suggestie voor de dimensies van een reparatie door middel van automatische herkenning van visuele schade

---

*Auteur*

N. OTTENS

612565@student.inholland.nl

*Opdrachtgever*

COMPOSITETENLAB INHOLLAND

---

Stagementor:

M. Wokke

mark.wokke@inholland.nl

---

Stagebegeleider:

Hans Waning

hans.waning@inholland.nl

24 januari 2021

# Inhoudsopgave

<b>1</b>	<b>Samenvatting</b>	<b>4</b>
<b>2</b>	<b>Inleiding</b>	<b>5</b>
<b>3</b>	<b>Vraagstelling</b>	<b>6</b>
<b>4</b>	<b>Schadepatroon definitie</b>	<b>7</b>
4.1	Probleemstelling . . . . .	7
4.2	Methode . . . . .	7
4.3	Resultaten . . . . .	8
4.4	Deelconclusie . . . . .	8
<b>5</b>	<b>Onderzoek naar manieren van automatisch herkennen</b>	<b>9</b>
5.1	Probleemstelling . . . . .	9
5.2	Methode . . . . .	9
5.3	Resultaten . . . . .	10
5.3.1	Neural networks . . . . .	10
5.3.2	Learning rate . . . . .	11
5.3.3	Momentum . . . . .	11
5.3.4	Epochs, itteraties en batch size . . . . .	11
5.3.5	Activatie functies . . . . .	12
5.3.6	Artificial Neural Network (ANN) . . . . .	14
5.3.7	Recurrent Neural Network (RNN) . . . . .	14
5.3.8	Auto Encoder (AE) . . . . .	14
5.3.9	Denoising Auto Encoder (DAE) . . . . .	15
5.3.10	Restricted Boltzmann Machine (RBM) . . . . .	15
5.3.11	Vgg16 . . . . .	17
5.3.12	Convolutioneel neuraal netwerk . . . . .	18
5.4	Deelconclusie . . . . .	20
<b>6</b>	<b>Classificatie systeem</b>	<b>21</b>
6.1	Probleemstelling . . . . .	21
6.2	Methode . . . . .	21
6.3	Resultaten . . . . .	22
6.3.1	Framework . . . . .	22
6.3.2	Tensorflow . . . . .	22
6.3.3	Theano . . . . .	23
6.3.4	Scikit-learn . . . . .	23
6.3.5	Caffe . . . . .	23
6.3.6	Torch . . . . .	23
6.4	Hardware . . . . .	24
6.4.1	Het trainen van een custom dataset . . . . .	25
6.5	Proof of concept . . . . .	26
6.5.1	Ontwerp . . . . .	26
6.5.2	Test . . . . .	27
6.6	Deelconclusie . . . . .	30
<b>7</b>	<b>Maskeren van schade en het custom trainen van een object detectie dataset</b>	<b>31</b>
7.1	Probleemstelling . . . . .	31
7.2	Methode . . . . .	31
7.3	Resultaten . . . . .	32
7.3.1	Mask RCNN . . . . .	33
7.3.2	FastFCN . . . . .	33
7.3.3	Gated-SCNN . . . . .	33
7.3.4	DeepLab . . . . .	33

7.3.5	Unet . . . . .	34
7.4	Proof of concept object detection . . . . .	35
7.4.1	Dataverzameling . . . . .	35
7.4.2	Ontwerp . . . . .	35
7.5	Trainen . . . . .	35
7.5.1	Test . . . . .	36
7.6	Deelconclusie . . . . .	39
<b>8</b>	<b>Metten van schade</b>	<b>40</b>
8.1	Probleemstelling . . . . .	40
8.2	Methode . . . . .	40
8.3	Resultaten . . . . .	40
8.3.1	Camera . . . . .	40
8.4	Intel Realsense familie . . . . .	40
8.4.1	Ontwerp . . . . .	42
8.5	Deelconclusie . . . . .	44
<b>9</b>	<b>Conclusie</b>	<b>45</b>
<b>10</b>	<b>Aanbevelingen</b>	<b>46</b>
<b>11</b>	<b>Bibliografie</b>	<b>47</b>
<b>12</b>	<b>Bijlagen</b>	<b>51</b>
12.1	Tussen tijdse resultaten . . . . .	51
12.2	Code en resultaat . . . . .	51
12.2.1	R1T2 . . . . .	51
12.2.2	R1T3 . . . . .	55
12.2.3	R1T4 . . . . .	59
12.2.4	R1T5 . . . . .	63
12.2.5	R1T6 . . . . .	67
12.2.6	R1T7 . . . . .	71
12.2.7	R1T8 . . . . .	75
12.2.8	R1T9 . . . . .	79
12.2.9	R1T10 . . . . .	84
12.2.10	R1T11 . . . . .	88
12.3	Dependencies . . . . .	95
12.3.1	Classificatie . . . . .	95
12.3.2	Object detectie . . . . .	105

# 1 Samenvatting

In dit open onderzoek bij Inholland Composites is er gekeken naar "hoe er een systeem kan worden opgezet om automatisch visuele schades op composiet te herkennen om vervolgens een suggestie te maken voor de afmetingen van de reparatie van de schade".

Hierbij is er onderzocht welke manieren van automatisch schade herkennen toepasbaar zijn op de context, waarbij schade herkennen van 2mm of meer het meest van belang is.

Bij het onderzoek naar de manieren van het automatisch detecteren is er voortgekomen dat classificatie, object herkenning en segmentatie (per pixel) hier de juiste manieren voor zijn.

Er is vervolgens genoeg data verzameld om een deep learning vorm op alle modellen toe te passen waardoor de accuratesse toeneemt. Vervolgens is er een theoretische opzet gemaakt om de schade te meten en een suggestie te maken voor de reparatie.

Ten slotte is er een plan opzet voor een test implementatie met als uiteindelijke hardware een NVIDIA Jetson AGX Xavier in combinatie met een Intel realsense D415 om hiermee realtime te kunnen detecteren.

## 2 Inleiding

Het project FIXAR (Future Improvements for Composites Sustainable Automated Repair) is in stand gebracht om beschadigde composieten in de luchtvaart- en windenergiesector te repareren. Hierbij zijn de volgende partijen betrokken: Inholland-onderzoekslijn Composiet, Hogeschool van Amsterdam, Hogeschool Saxion, CompositesNL, KVE Compositengroep, PONTIS Engineering, Schatmaker, Colosso, Carbon Racing, Fusion Engineering, Specto Aerospace, Field Lab Zephyros, Fokker, KLM, Koninklijke Luchtmacht, NLR, TU Delft, Energieorgaan SIA. Deze partijen hebben een opdracht aangeleverd aan Inholland Composites, bij dit project is het de bedoeling om verscheidene schades automatisch te herkennen op composieten en vervolgens een suggestie te maken voor de reparatie.

Eerst zal er een probleemomschrijving gemaakt worden om het probleem in kaart te brengen en het probleem af te bakenen. Vervolgens zal het probleem worden opgedeeld in deelvragen, zodat door het beantwoorden en onderzoeken van deze deelvragen uiteindelijk de hoofdvraag wordt beantwoord.

### 3 Vraagstelling

FIXAR heeft een probleem waarbij elke inspectie uitgevoerd moet worden door een specialist gevolgd door een reparatie in het geval van schade. Een project is opgesteld om deze twee processen gedeeltelijk te automatiseren en als gevolg deze dure en intensieve taak te omzeilen.

Het doel is om automatisch schade te herkennen op composiet en vervolgens een suggestie te maken voor de dimensies van de reparatie. Waarbij de schade wordt opgeslagen in combinatie met een scan voor een manuele check door de specialist.

Hierdoor wordt het mogelijk iemand in te zetten met minimale ervaring om de inspectie uit te voeren.

Dit betekent dat er eerst onderzoek gedaan gaat worden naar welke soorten schades herkend moeten worden om te kunnen kijken naar oplossingen die zich hiervoor uitlenen. Hierna zal er een analyse worden opgesteld op basis van voorkeur op vereisten, kostenanalyse, schaalbaarheid en benodigde verwerkingskracht. Ten slotte zal de gekozen oplossing geïmplementeerd worden.

Hieruit leidt de hoofdvraag:

- “Hoe kan er systeem worden opgezet om automatisch visuele schades op composiet te herkennen om vervolgens een suggestie te maken voor de afmetingen van de reparatie van de schade?”

Met de 5 deelvragen:

- “Wanneer moet een onevenheid of patroon als schade herkend worden?”
- “Welke methode is het meeste geschikt om visuele schades te detecteren op composiet?”
- “Hoe kan deze manier van schadedetectie geïmplementeerd worden?”
- “Hoe kan de diameter van de gedetecteerde schade gemeten worden?”
- “Hoe kan deze manier van meten worden geïmplementeerd in het schadedetectiesysteem?”

## 4 Schadepatroon definitie

### 4.1 Probleemstelling

Voordat een schade herkenning systeem gebouwd kan worden, zal er eerst onderzoek gedaan worden naar welke schades er bestaan, welke onderzocht gaan worden en welke niet besproken zullen worden. Vervolgens zal er onderzoek gedaan worden naar het stadium waarvan het de bedoeling is dat deze herkend word, onder de voorwaarde dat deze visueel detecteerbaar is.

De erosie die voorkomt op windturbines door impact van regen en hagel zullen worden onderzocht. Hieruit leidt de vraag:

”Wanneer moet een oneffenheid of patroon als schade herkend worden?”





### 4.2 Methode

Om deze deelvraag te beantwoorden zal er deskresearch gedaan worden om te onderzoeken wanneer een schade in aanmerking komt om gedetecteerd te worden. Hier zal er gekeken worden naar:

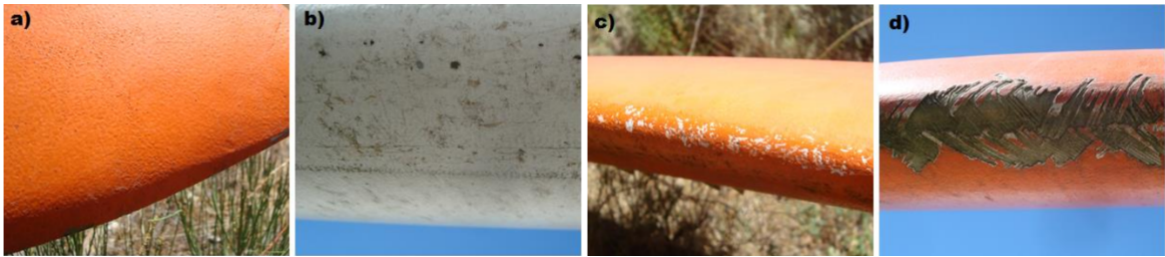
- Type schade
- Dimensies schades/erosie
- Verkleuring

### 4.3 Resultaten

In figuur 1 is te zien dat de schade door erosie onder te verdelen valt vier subcategorieën.

Erosion Level	1	2	3	4
Example Picture				

Figuur 1: levels erosie[44]



Figuur 2: verschillende soorten schaden[32]

Een schade kan in dit geval herkend worden als er een verkleuring te zien is. Dit betekent dat de externe coating van het blad is gaan slijten, in dit geval regen te zien in figuur: 1.

### 4.4 Deelconclusie

Bij een schade van 2mm is deze observeerbaar en de erosie neemt vanaf dit punt snel toe.



## **5 Onderzoek naar manieren van automatisch herkennen**

### **5.1 Probleemstelling**

Er is geconcludeerd dat de doelstelling is om erosie vanaf 2mm of meer te gaan herkennen op composiet. In samenhang met de doelstelling dat dit te automatiseren valt, genereert dit de deelvraag:

”Welke methode is het best geschikt om visuele schades te detecteren op composiet?”

### **5.2 Methode**

Om de best geschikte methode te selecteren om visuele schades op composiet te herkennen zal er exploratief onderzoek toegepast worden. Om zo de manieren te vinden die voor deze deelvraag mogelijk een oplossing bieden.

Deze methode is voornamelijk verkennend, om zo veel informatie te verzamelen omdat de huidige richting onbekend is.

Hier zal er gekeken worden naar de verschillende mogelijke manieren waarmee deze schades gedetecteerd kunnen worden.

## 5.3 Resultaten

Er is onderzoek gedaan naar:

- Neurale netwerken
- Diepe neurale netwerken

### 5.3.1 Neural networks

Neurale netwerken bestaan uit 3 of meer lagen.

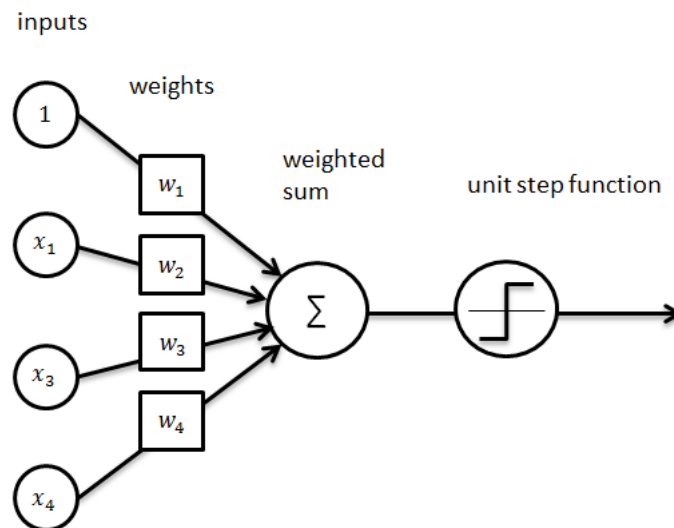
- De input
- De "verborgen laag"
- De output

De hoeveelheid lagen tussen de input en output wordt bepaald door de grootte van de input en de output.

De lagen worden met name gedefinieerd door:

- Het gemiddelde van de input en output lagen
- Minder dan 2x de input nodes
- $2/3$  input nodes + output nodes

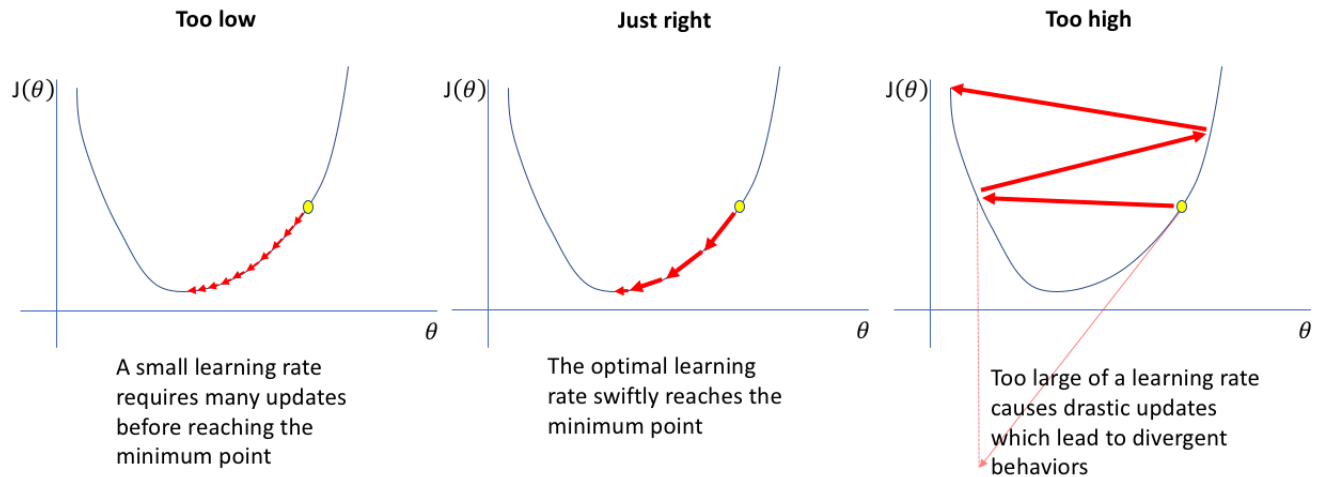
De verbindingen tussen de nodes hebben een gewicht, deze worden aangepast in het proces.



Figuur 3: neuron[18]

### 5.3.2 Learning rate

De learning rate[41] is een mechaniek toegepast in neurale netwerken om een variabele te hebben die direct reflecteerd op hoe agressief het (diepe) neurale netwerk leert. In figuur 4 is te zien hoe deze variabele invloed op heeft.



Figuur 4: learning rate voorbeeld[18]

### 5.3.3 Momentum

Momentum[36] is een term die gebruikt wordt om een afgeleide te geven van de functie die berekent of het model meer of minder accuraat wordt.

### 5.3.4 Epochs, iteraties en batch size

Epochs, iteraties en batch size[17] geven allemaal een orde van grootte aan in een andere vorm.

Één epoch geeft aan dat de gehele dataset één keer heen en weer is geweest door het (diepe) neurale netwerk.

Batch size is de grootte van de groep data die in één keer wordt berekend.

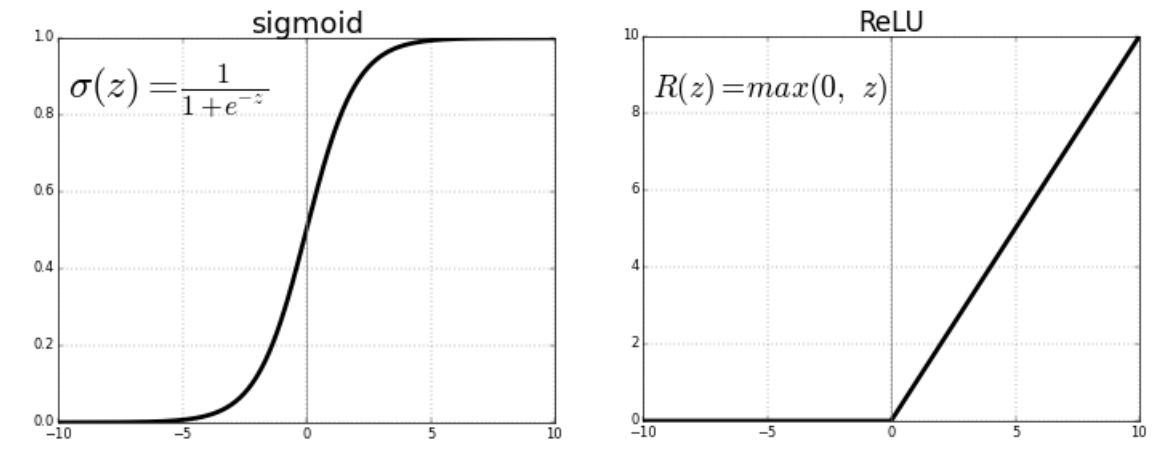
Met één iteratie wordt de grootte aangegeven van batches dat nodig is om één epoch succesvol af te ronden.

Voorbeeld: honderd fotos met een batch size van twintig, neemt vijf iteraties voor één epoch.

### 5.3.5 Activatie functies

Er zijn veel verschillende soorten activatie functies. In dit onderzoek is er alleen gekeken naar de 2 meest verschillende soorten activatie functies:

- Sigmoid
- ReLu



Figuur 5: sigmoid en ReLu functie[1]

Verschillende soorten neurale netwerken:

- Artificial Neural Network (ANN)
- Recurrent Neural Network (RNN)
- Auto Encoder (AE)
- Value Efficiency Analysis / Data Envelopment Analysis (VEA/DEA)
- Denoising Auto Encoder (DAE)
- Restricted Boltzmann Machine (RBM)
- Monte Carlo (MC)
- Hopfield Network (HN)
- Boltzman Machine (BM)
- Deep Boltzmann Machine (DBM)
- Deep Convolutional Neural Network (DCNN)
- Deconvolutional Network (DN)
- Deep Convolutional Inverse Graphics Network (DCIGN)
- Generative Adversarial Network (GANS)
- Liquid State Machine / Extreme Learning Machine (LSM/ELM)
- Echo State Network (ESN)
- Diluted Residual Network (DRN)
- Kohonen Network (KN)
- Neural Turing Machines (NTM)

### 5.3.6 Artificial Neural Network (ANN)

Een artificial neural network [4] is een netwerk dat gebouwd is om neuronen van het brein te simuleren. Deze bestaat dan ook uit honderden tot duizenden neuronen welke worden getraind op data met de gewenste output.

Hierbij wordt ook gebruik gemaakt van terug waardse propogatie, een proces dat de waarden van neuronen aanpast zodat deze de volgende keer mogelijk een meer gewenste output geven. Dit kan gedaan worden tot het moment dat dit neurale netwerk het laagst percentage error geeft.

Artificial neural network wordt bijvoorbeeld gebruikt om te herkennen welke mail spam is en het voorspellen van de beurswaarde van een bedrijf etc.

### 5.3.7 Recurrent Neural Network (RNN)

Een recurrent neural network[33] is een netwerk dat voorgetrainde en huidige acties opslaat. Een RNN kan meerdere vectoren tegelijkertijd als input nemen, om hier vervolgens één of meerdere uitkomsten van te geven.

Een standaard neurale netwerk heeft één grootte aan input vector, welke er op dezelfde manier uitkomt. Bij een recurrent neurale netwerk kan de input grootte en output grootte verschillen, terwijl het netwerk hiervan leert.

### 5.3.8 Auto Encoder (AE)

Een auto encoder network[5] bestaat uit 2 delen: een encoder en een decoder. Dit is een type netwerk dat doormiddel van een slimme compressie dezelfde output krijgt als de input. Deze kan werken met foto's, video's, tekst en spraak. Een auto encoder kan beschreven worden als "feature extraction algorithm", wat betekent dat er specifieke elementen of objecten uit data gehaald kunnen worden.

De nadelen van een auto encoder ligt in het decomprimeren, er gaat dan namelijk data verloren. Hiernaast is dit netwerk enkel toepasbaar op de data waarop deze getraind is.

- Encoder: de encoder encodeert de data
- Decoder: reconstrueert de data

Auto encoders worden gebruikt voor datacompressie.

### **5.3.9 Denoising Auto Encoder (DAE)**

Een denoising auto encoder[14] is een auto encoder waarbij sommige input waarden op 0 worden gezet. Dit wordt gedaan om te voorkomen dat dit specifieke neurale netwerk een null functie ongewild leert. Dit betekent dat de output exact hetzelfde is als de input en de encoder zelf onbruikbaar wordt.

De data wordt gecorumpereerd door 30 tot 50 procent van de input op nul te zetten, zo worden de features beter geleerd.

### **5.3.10 Restricted Boltzmann Machine (RBM)**

Een restricted boltzman machine[34] kan patronen vinden in data door het reconstrueren van de input. Dit kan gebruikt worden om features te herkennen en wordt ook wel een auto encoder genoemd.

Er is niet op diepte ingegaan op de volgende netwerken omdat deze niet toepasbaar zijn in de context:

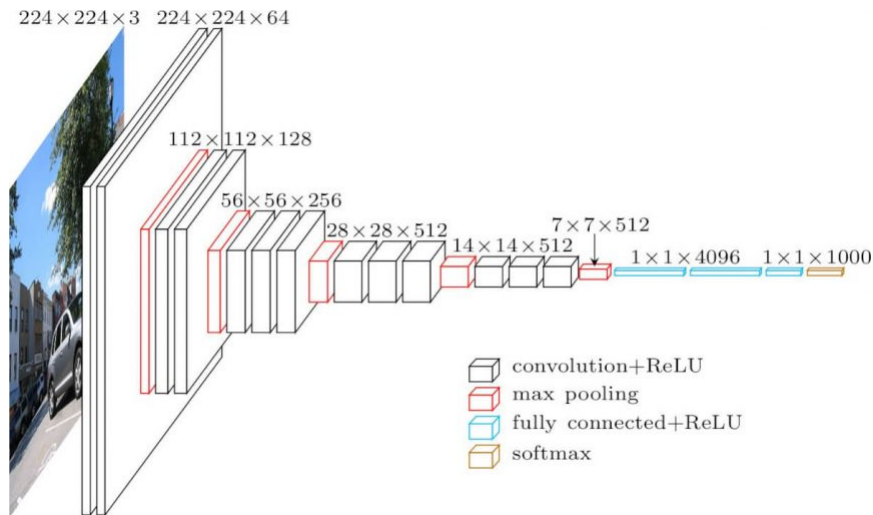
- Monte Carlo [38]
- Hopfield Network[3]
- Boltzmachine Machine[6]
- Deep Boltzmann Machine [10]
- Deep Convolutional Neural Network[12]
- Deconvolutional Network[9]
- Deep Convolutional Inverse Graphics Network [11]
- Generative Adversarial Network[42]
- Liquid State Machine[8]
- Echo State Network[16]
- Diluted Residual Network[15]
- Kohonen Netwerk[27]
- Neural Turing Machine[29]



### 5.3.11 Vgg16

Vgg16[43] is te zien in figuur 6, dit is een convolutioneel neuraal netwerk model dat is voorgetraind op 5000 verschillende objecten.

In figuur 6 is te zien hoe het vgg16 model is opgezet. Hier is er duidelijk te zien dat deze een foto met de dimensies  $224 \times 224$  als input neemt. De laatste 3 staat voor de lagen van de foto, wat in dit geval R, G en B is.



Figuur 6: vgg16

### 5.3.12 Convolutioneel neuraal netwerk

In dit geval zal er alleen gekeken worden naar een convolutioneel neuraal netwerk omdat dit soort netwerk geschikt is om features en patronen uit een foto te halen. Dit zal gecombineerd worden met het voorgetrainde vgg16.

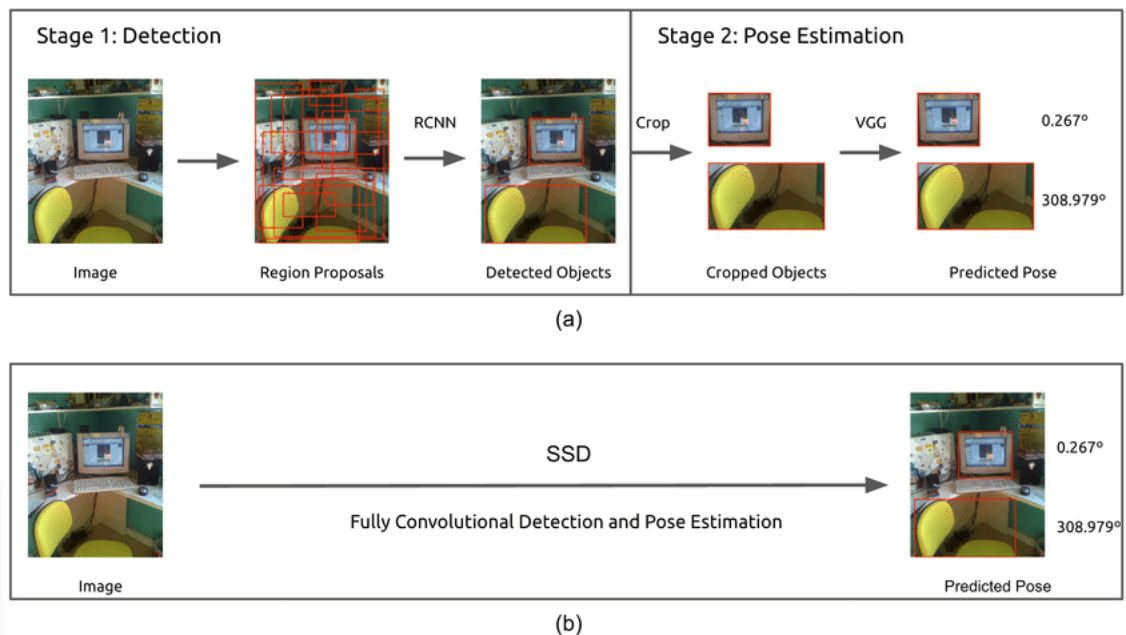
Uit een foto kan geclassificeerd worden of het composiet schade heeft.

Verschillende manieren van Convolutioneel neuraal netwerk:

- 1-stage netwerk(bijv. YOLOV3[47])
- 2-stage netwerk(bijv. R-CNN[30])

Het verschil[31] is dat bij een 1-stage netwerk er direct wordt gezocht naar een object. Bij een 2 stage netwerk worden er eerst “region of interest” uitgelicht, waarna deze uit de foto gesneden worden om vervolgens te worden geclassificeerd.

Een 1-stage netwerk is langzamer maar accurater dan een 2-stage netwerk.



Figuur 7: 2-stappen versus 1-stap[31]

Een 1 stap netwerk kan 10% tot 40% minder accuraat[37] zijn.

Neuraal netwerk[46] bestaat uit

- Input
- Backbone
- Neck
- Dense prediction

Backbone is de naam om te refereren naar een feature extractor network.

Modellen zoals ResNet, DenseNet en VGG worden gebruikt als feature extractors. Deze zijn voorgetraind op foto classificatie datasets zoals ImageNet en vervolgens getuned op de detectie dataset.

De neck zijn extra lagen tussen de backbone en de head. Deze worden gebruikt om verschillende feature maps van verschillende stages uit de backbone te halen. De neck kan bestaan uit bijvoorbeeld: FPN, PANet, Bi-FPN, etc. YOLOv3 gebruikt FPN om features van verschillende schalen van de backbone te halen.

Er moet eerst een op maat gemaakte dataset voorgetraind zullen worden om deze dataset in combinatie met het neurale netwerk te implementeren.

## 5.4 Deelconclusie

Het visueel detecteren is mogelijk door middel van diepe neurale netwerken. Dit is mogelijk omdat de dataset bestaat uit 6131 foto's. Hierdoor kan een classificatie model voorgetraind worden.

Hiervoor zal een Convolutional neural network toegepast worden. Er is hiervoor gekozen omdat deze techniek voornamelijk toegepast wordt om beelden te classificeren. In dit geval zal dit toegepast worden op de windturbine dataset.

## **6 Classificatie systeem**

### **6.1 Probleemstelling**

Er is bekend dat er schade herkent gaat worden en dat er hiervoor een diep neurale netwerk de betere keuze is. Echter zijn er meerdere manieren om een diep neurale netwerk te implementeren.

Hieruit leidt de deelvraag:

”Hoe kan deze manier van schade detectie geïmplementeerd worden?”

### **6.2 Methode**

Voor de derde deelvraag zal er toegepast onderzoek worden uitgevoerd in combinatie met deskresearch om uit te vinden wat de beste methode zal zijn om een proof of concept op te zetten voor deelvraag 2.

De deelvraag ”Hoe kan deze manier van schade detectie geïmplementeerd worden?” zal beantwoordt worden door een vergelijkend onderzoek op te stellen naar de verscheidene software en hardware oplossingen.

## 6.3 Resultaten

### 6.3.1 Framework

De verscheidene frameworks waar deep learning network in geschreven kan worden zijn:

- Tensorflow[2]
- Theano[39]
- Scikit-learn[35]
- Caffe[7]
- Torch[45]

### 6.3.2 Tensorflow

Voordelen:

- Gemakkelijk in gebruik
- Tensorboard
- Community
- Keras implementatie voor neurale netwerken

Nadelen:

- Data input voor tensorflow 2.0 is niet gebruikelijk
- Installeren van TF-GPU

### 6.3.3 Theano

Voordelen:

- Controle over alle variabelen
- Sneller dan Tensorflow

Nadelen:

- Ouder framework
- Relatief meer code tegenover tensorflow voor dezelfde berekeningen
- Niet makkelijk uitbreidbaar naar andere apparaten

### 6.3.4 Scikit-learn

Voordelen:

- Gemakkelijke data manipulatie

Nadelen:

- Abstract

### 6.3.5 Caffe

Voordelen:

- Het nauwkeurig afstellen van modellen
- Modellen maken zonder code te schrijven
- Support tensors

Nadelen:

- Niet goed voor recurrente netwerken
- Geen support/communitie
- Nieuwe GPU lagen moet je zelf produceren
- Gebruikt 1 GPU
- Low level

### 6.3.6 Torch

Voordelen:

- Gemakkelijk eigen lagen ontwerpen
- Modulair
- voorgetrainde modellen

Nadelen:

- Slechte documentatie
- Kan niet gebruikt worden om modellen te trainen

## 6.4 Hardware

Om realtime te kunnen te kunnen classificeren is er rekenkracht nodig. De volgende apparaten zullen met elkaar vergeleken worden om zo een keuze te kunnen maken voor de uiteindelijke hardware implementatie.

- Raspberry pi 4b+
- NVIDIA Jetson AGX Xavier
- NVIDIA Jetson Nano
- NVIDIA Jetson Xavier NX

-	<b>Rapberry pi 4b+</b>	<b>JETSON NANO</b>	<b>XAVIER NX</b>	<b>JETSON AGX</b>
<b>CPU</b>	4-core 64-bit 1.5GHz	4-core ARM	6-core Carmel	8-core 64-bit
<b>GPU</b>	Broadcom VideoCore VI	128-core Maxwell	384-core Volta	512-core Volta
<b>RAM</b>	8GB	4 GB	8 GB	16 GB
<b>TENSORCORES</b>	0	0	48	64
<b>GEWICHT</b>	46g	241g	780g	650g
<b>Power Consumption</b>	3.4 watt	10 wat	10 watt	30 watt
<b>TOPS</b>	13.5 GFLOPS <sup>1</sup>	472 GFLOPs	21 TOPs	32 TOPs



De verschillende talen waarin dit geïmplementeerd kunnen worden:

- Java
- C
- R
- Python
- c++

#### **6.4.1 Het trainen van een custom dataset**

Voor het trainen van een custom dataset is het neurale netwerk getraind met 2500 fotos van windturbine wieken met schade, en 2500 foto van hele windturbine wieken.

Bij het trainen zal er gebruik gemaakt worden van een combinatie van het VGG16 model plus een eigen model.

Het grootste obstakel is hoe snel dit model getraind kan worden op een computer.

## 6.5 Proof of concept

Er is een proof of concept gemaakt met de verzamelde data. Hier zijn vieze/groene vlekken op windturbines bij meegenomen omdat de verkleuring een kenmerk is van erosie en er niet genoeg foto's waren van specifieke plekken met erosie.

### 6.5.1 Ontwerp

Er is uiteindelijk gekozen om de Tensorflow library te gebruiken omdat deze vrij recent is, wordt onderhouden door Google en het .H5 model, wat eruit komt makkelijk over te zetten is naar andere apparaten.

Hier zijn meerdere modellen voor gemaakt welke te vinden zijn in de bijlagen. Uiteindelijk is het onderstaande model eruit gekomen omdat dit na 24 uur trainen het meest accuraat is.

```
def define_model():
    # laad model
    model = VGG16(include_top=False, input_shape=(224, 224, 3))
    # vgg16 lagen niet trainbaar
    for layer in model.layers:
        layer.trainable = False
    # nieuwe classifiers
    flat1 = Flatten()(model.layers[-1].output)
    class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
    output = Dense(1, activation='sigmoid')(class1)
    # nieuw model
    model = Model(inputs=model.inputs, outputs=output)
    # compileer model
    #####
    opt = SGD(lr=0.01, momentum=0.9)
    #####
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

### 6.5.2 Test

Het bovenstaande model is getest met 2 foto's waarvan 1 met schade en 1 zonder schade. Het onderstaande model is de code waarmee er is getest.

```
# make a prediction for a new image.
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import re

# laad foto
def load_image(filename):
    img = load_img(filename, target_size=(224, 224))
    img = img_to_array(img)
    # converteer naar 3 channels
    img = img.reshape(1, 224, 224, 3)
    # center pixel data
    img = img.astype('float32')
    img = img - [123.68, 116.779, 103.939]
    return img

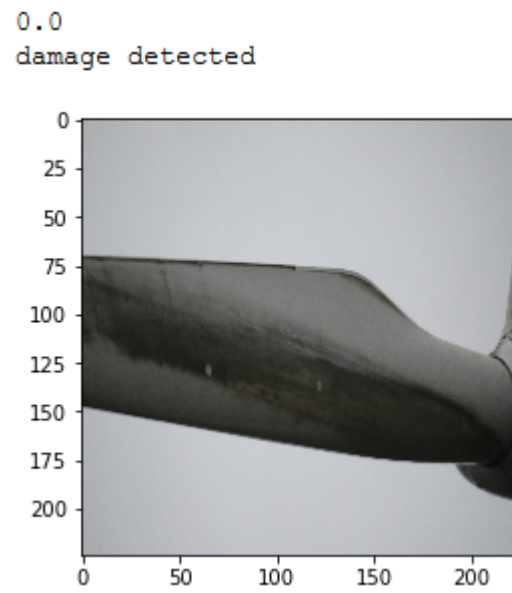
# voorspel image
def run_example():
    # laad image
    img = load_image('/smalltestset/nietschade/2S0A4986.jpg')
    # laad model
    model = load_model('savefiles/final.h5')
    # Predcit
    result = model.predict(img)

    #test = re.search('[+]?'([0-9]*\.[0-9]+|[0-9]+)', result[0])
    list1 = result.tolist()
    value = list1[0][0]
    print(value)

    #print(list1)

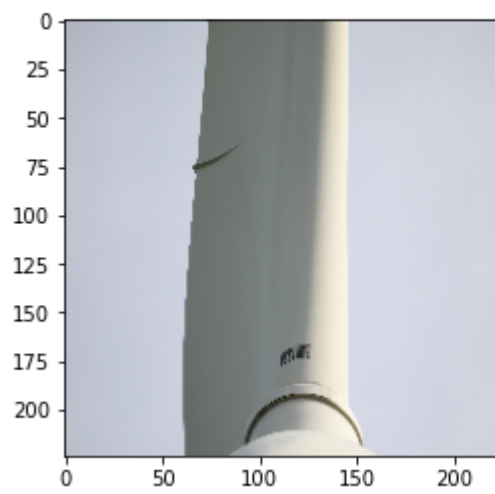
    if(float(value) > 0.5 and float(value) < float(1)):
        print('no damage detected')
        print(list1)
    #elif(result[0] > 1 and result[0] < 2):
    elif(float(value) < 0.5):
        print('damage detected')
    else:
        print('all your base are belong to us')

#run test
run_example()
```

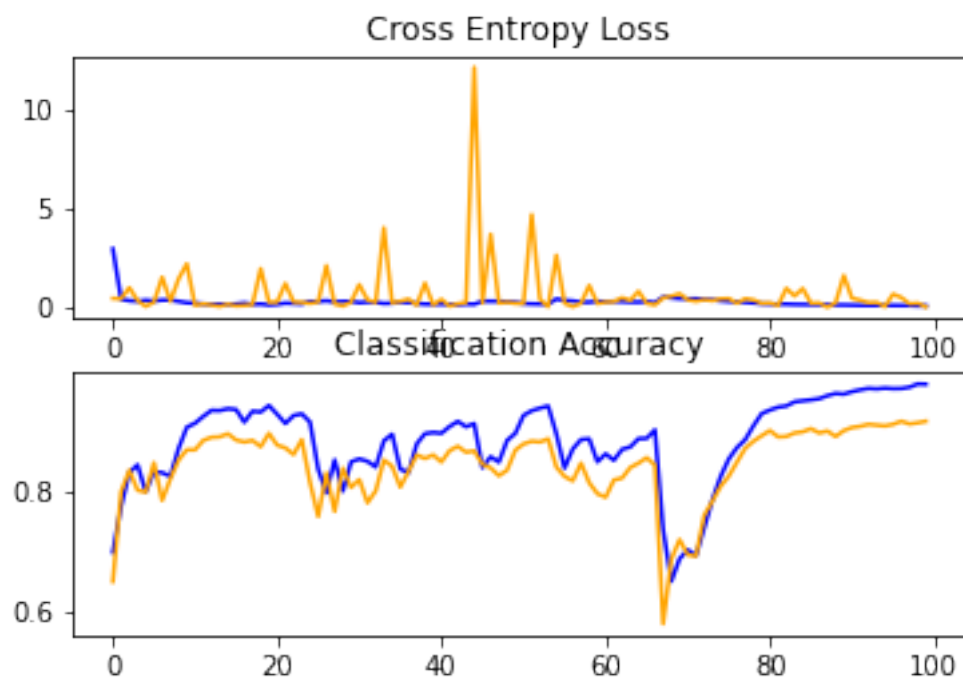


Figuur 8: classificatie correct erosie

```
0.9297806024551392  
no damage detected  
[[0.9297806024551392]]
```



Figuur 9: classificatie correct geen erosie



Figuur 10: accuratesse

## 6.6 Deelconclusie

Uit de test is gekomen dat het model met ongeveer 97 procent zekerheid kan voorspellen op de test data, zie figuur 10.

Hiernaast is voor de NVIDIA Jetson AGX Xavier gekozen omdat deze het meeste rekenkracht heeft. Hiermee kan er op de drone in real-time worden gedetecteerd of de videobeelden kunnen via het nieuwe 5G netwerk worden verzonden om zo op een andere locatie de beelden te verwerken.

Dit is vervolgens getest op een foto met erosie en een foto zonder erosie. Hieruit valt te concluderen dat op de foto met erosie, te zien op figuur 8, dit juist gedetecteerd wordt.

Op de foto zonder erosie te zien op figuur 9, is te zien dat er geen erosie gedetecteerd wordt, dit is tevens ook correct.

## **7 Maskeren van schade en het custom trainen van een object detectie dataset**

### **7.1 Probleemstelling**

Bij het herkennen van de erosie of schade, is het vervolgens de bedoeling dat de schade gemeten gaat worden. Hierbij moet eerst het object herkend worden om vervolgens door middel van segmentatie de schade eruit te halen die vervolgens gemeten kan worden. Hieruit leidt de deelvraag "Hoe kan de diameter van de gedetecteerde schaden gemeten worden?"

### **7.2 Methode**

Voor deelvraag 4 zal er deskresearch worden gedaan om gedetecteerde schaden te kunnen herkennen met een masker. Hierbij zal er onderzoek gedaan worden naar de verschillende soorten segmentatie om vervolgens de schade te kunnen onderscheiden op een pixel niveau. Ten slot zal de de meest reële oplossing in deelvraag 5 geïmplementeerd worden.

## 7.3 Resultaten

De exacte schade herkennen is mogelijk door foto segmentatie. Dit betekent dat er specifieke objecten uit een foto herkend kunnen worden en deze er exact per pixel uit gehaald kunnen worden.

Met deze techniek is het dus mogelijk om in combinatie met een speciale camera de schade te kunnen meten.

Er zijn een grofweg twee verschillende soorten van segmentatie, dit zijn:

- Region-Based Segmentatie
- Edge-Detection Segmentatie

Region-based detectie haalt de objecten uit de foto op basis regio's.

Als het object en de achtergrond een hoog contrast hebben dan ligt de nauwkeurigheid hoog. Daarnaast is Region-based object detection vrij snel.

Edge-detection is beter voor lange doorlopende objecten en gaat niet uit objecten maar haalt randen uit de foto en transformeert deze tot objecten. Als er veel ruis in een foto is of veel door elkaar lopen objecten, neemt de nauwkeurigheid af.

Bij semantische segmentatie worden de objecten er per klasse uitgehaald, in tegenstelling tot instantie segmentatie waar de objecten er per instantie worden uitgehaald.

De modellen die hiervoor het meest geschikt zijn voor omvatten:

- U-net
- FastFCN
- Gated-SCNN
- DeepLab
- Mask R-CNN



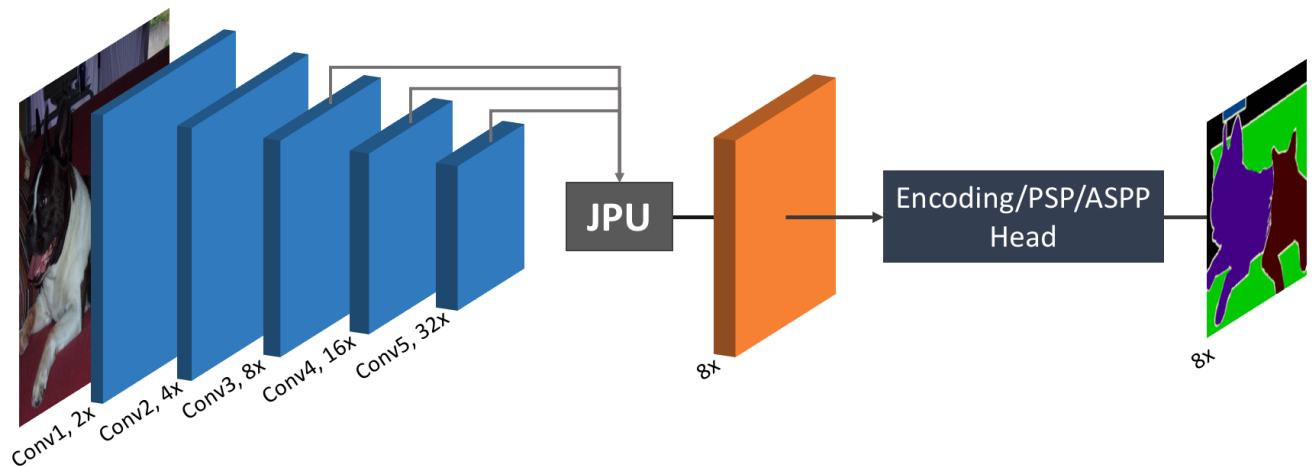
### 7.3.1 Mask RCNN

Mask RCNN[28] is een extensie van faster R-CNN[19].

Faster R-CNN is een diep convolutioneel netwerk dat de klasse genereert en een bounding box aan het object toevoegt. Mask R-CNN doet hetzelfde maar voegt hierna nog een object masker toe.

### 7.3.2 FastFCN

Bij FastFCN[20] wordt er gebruik gemaakt van een techniek die de resolutie verbetert. Het proces waar de resolutie verbeterd wordt heet JPU en dit neemt veel resources in beslag.



Figuur 11: FastFCN architecture

### 7.3.3 Gated-SCNN

Gated-SCNN[21] bestaat uit twee convolutionele netwerken. Hierbij wordt 1 CNN gebruikt voor classificatie en de 2e voor het herkennen van de grenzen tussen objecten.

Bij alleen segmentatie is er veel verlies als de objecten kleiner worden en hier neemt ook de nauwkeurigheid af van de grenzen tussen objecten.

### 7.3.4 DeepLab

In DeepLab[13] worden upsampled filters gebruikt om de resolutie te verbeteren.

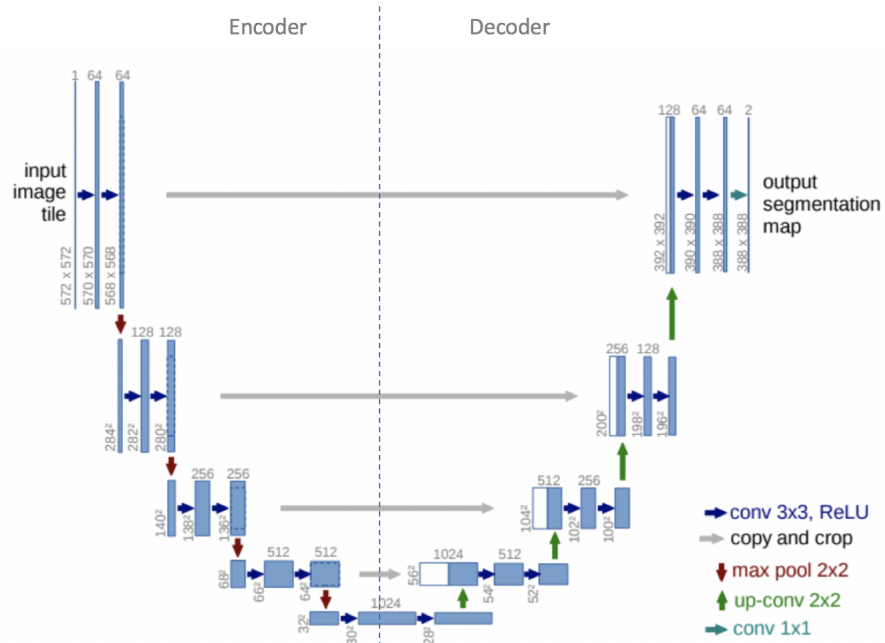
DeepLab bestaat grofweg uit 2 lagen:

- Encoderen: de foto gaat eerst door een CNN om alle features uit de foto te halen en deze te classificeren.
- Decoderen: gefilterde informatie(verzameling features) wordt gereconstrueerd tot de juiste dimensies waarna deze wordt omgezet tot een masker.

### 7.3.5 U-Net

U-Net[40] is een convolutioneel neurale netwerk dat ontwikkeld is voor biomedische beelden. Het model representateert een 'u' in model vorm, te zien in figuur: 12.

Bij U-Net wordt de foto eerst geëncodeerd, dit gebeurt om de ruis eruit te halen. Na dat de ruis eruit is gehaald wordt de foto gedecodeerd en blijven vervolgens de features over.



Figuur 12: U-Net [22]

## **7.4 Proof of concept object detection**

Om een opzet te maken is er eerst gekozen om een object detectie model te maken, om zo bekend te worden met deze techniek in combinatie met de implementatie.

### **7.4.1 Dataverzameling**

Om een object detectie model te trainen is er eerst data nodig. Gezien deze data niet bestond en/of niet publiekelijk beschikbaar was, is er een plan gemaakt en uitgevoerd om een camera te huren en in 1 dag zoveel mogelijk foto's te maken van windturbines. Hier zijn uiteindelijk 6131 foto's van overgebleven.

De windturbines die zijn gefotografeerd bevinden zich in Noord-Holland en Lelystad.

### **7.4.2 Ontwerp**

Er is een ontwerp gemaakt met de `ssd_mobilenet_v1_coco_2018_01_28` in Tensorflow. Dit is een voorgetrainde dataset, om zo de trainingstijd te minimaliseren van de proof of concept.

## **7.5 Trainen**

Er is getraind met 200 foto's, die geschaald zijn naar 300x300 vanwege een buffer overflow. Er is 29551 keer over de gehele data gegaan.

### 7.5.1 Test

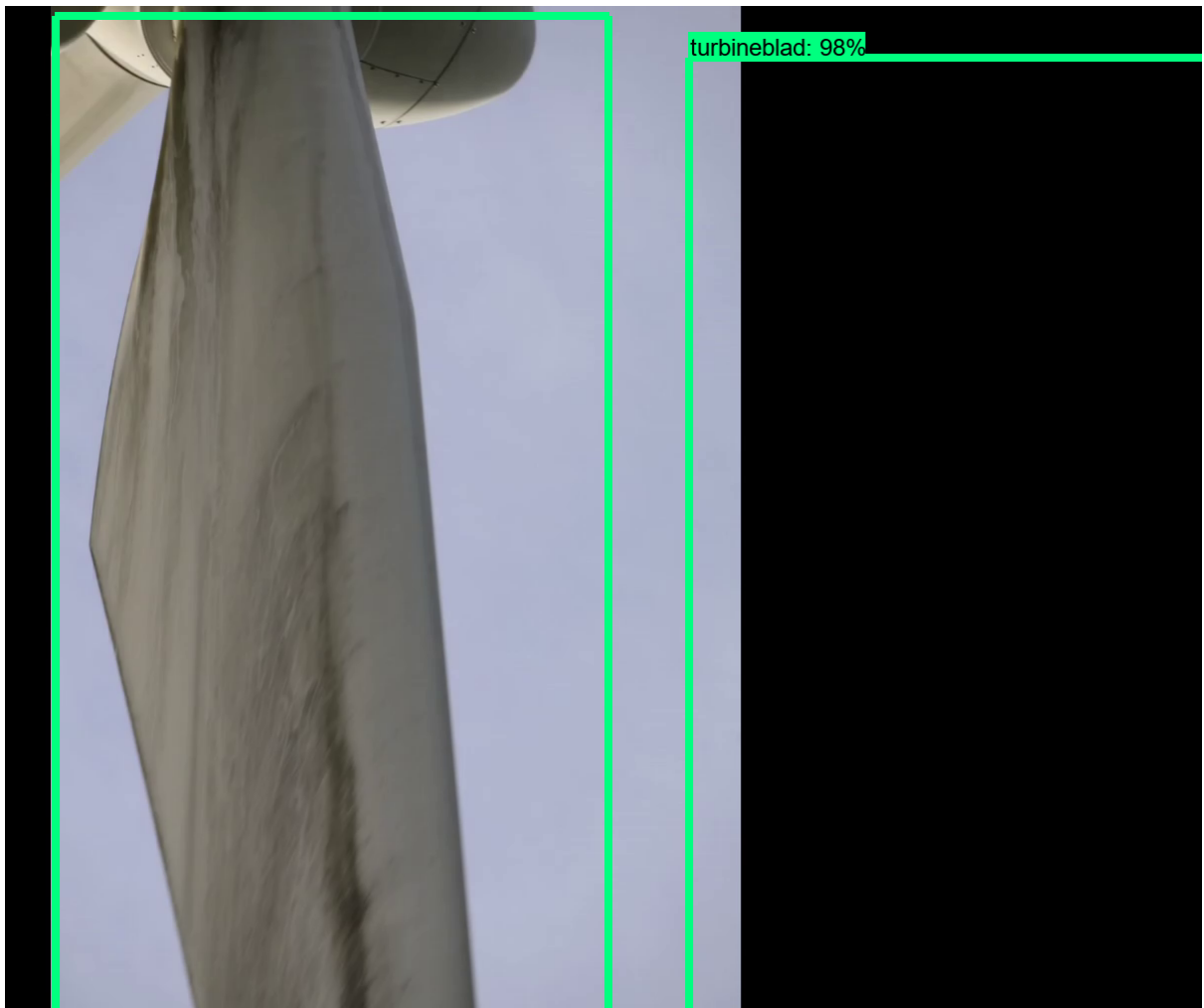
De test is uitgevoerd op een opgenomen video waarin er door foto's wordt gescrolled van de desbetreffende dataset. De resultaten zijn te zien in figuur: 13, 14 en 15. Hierbij is te zien de windturbine bladen worden herkend maar er nog valse positieven in zitten.



Figuur 13: foto 1 object detecteren



Figuur 14: foto 2 object detecteren



Figuur 15: foto 3 object detecteren

## 7.6 Deelconclusie

FastFCN zou de beste keuze zijn als snelheid en nauwkeurigheid allebei even zwaar meewegen. DeepLab en Gated SCNN zouden de preferente keuzes zijn als nauwkeurigheid zwaarder mee weegt.

Uiteindelijk is er gekozen om met een SSD mobilenet, in combinatie met tensorflow, een test opstelling te maken voor het detecteren van objecten. Hierbij is te zien in figuur: 13, 14 en 15 dat deze duidelijk de objecten in realtime herkent.

Het nadeel van de huidige dataset is dat object herkenning werkt door, van te voren objecten aan te geven en hierop door te blijven itereren met het netwerk. Zodanig dat hier een getraint model uitkomt. Echter werkt object detectie met externe ruis in de foto en bij deze dataset zit er geen ruis in de lucht.

Dit zou op te lossen moeten zijn door random objecten te generen en deze random op de foto's plakken. Het gebruiken van alle foto's zou de valse positieven ook moeten verminderen.

## 8 Meten van schade

### 8.1 Probleemstelling

”Hoe kan deze manier van meten worden geïmplementeerd in het schadedetectie-systeem?”

### 8.2 Methode

Voor deelvraag 5 zal er deskresearch worden uitgevoerd om een camera te vinden waarmee vervolgens, in combinatie met deelvraag 4 de schade waarmee automatische de schade mee gemeten kan worden via software.

Hier zal er onderzoek gedaan worden naar verscheidene camera's en welke toepasbaar zijn in deze context om schades te meten die 2mm of groter zijn.

### 8.3 Resultaten

#### 8.3.1 Camera

De camera's zullen op verschillende criteria afgewogen worden, de criteria zijn:

- Gewicht
- Nauwkeurigheid
- Infrarood
- Programmeerbaarheid qua meten (dual camera)

#### 8.4 Intel Realsense familie

Intel Realsense D415[23]

- Gewicht: 72 gram
- Nauwkeurigheid: onder de 2% op 2 m
- Infrarood: Active IR Stereo
- Programmeerbaarheid qua meten (dual camera): librealsense

Intel Realsense D435i[24]

- Gewicht: 72 gram
- Nauwkeurigheid: onder de 2% op 2 m
- Infrarood: Active IR Stereo
- Programmeerbaarheid qua meten (dual camera): librealsense

Intel Realsense D455[25]

- Gewicht: 103 gram
- Nauwkeurigheid: onder de 2% op 4 m
- Infrarood: Active IR Stereo



- Programmeerbaarheid qua meten (dual camera): librealsense

intel realsense L515[26]

- Gewicht: 61 gram
- Nauwkeurigheid: 5 mm tot 14 mm tot 9 m2
- Infrarood: nee
- Programmeerbaarheid qua meten (dual camera): librealsense

#### 8.4.1 Ontwerp

Het ontwerp zorgt ervoor dat de object detectie het element uit volledige resolutie foto's haalt. Deze wordt doorgegeven aan het 2e model waarbij er een masker van de schade wordt gegenereerd.



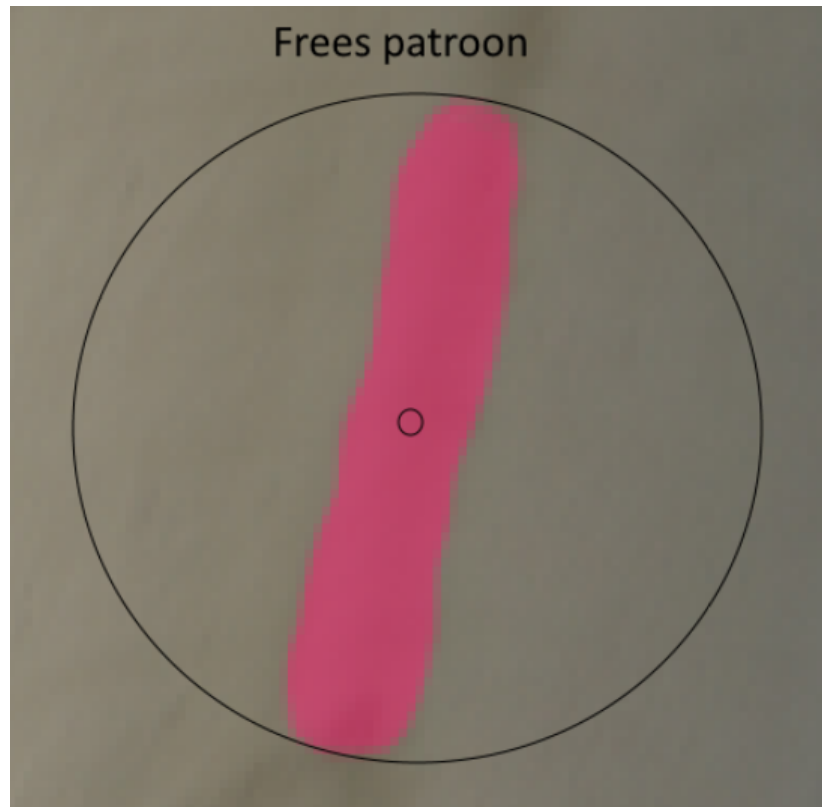
Figuur 16: Gemaskeerde schade

Als dit masker, te zien in figuur: 16, gegenereerd is dan is het de bedoeling dat er van elke buitenste pixel naar elke andere pixel gemeten wordt te zien in figuur: 17. Met deze data kan er een maximale lengte uitgerekend worden, waarbij deze dient als diameter voor de volgende stap.



Figuur 17: Meten van pixels

De diameter kan hierbij helpen in combinatie met het gemiddelde van alle meetpunten om het exacte midden te vinden van deze schaden en dus een suggestie te geven voor de dimensies van de reparatie zoals in figuur 18.



Figuur 18: Voorbeeld freespatroon

## 8.5 Deelconclusie

Er is dus gekozen voor de Intel realsense D415 omdat deze het meest accuraat is samen met de D435. De d435 heeft minder pixels per inch en dus is de d415 scherper/nauwkeuriger in het meten.

Echter met een nauwkeurigheid van onder de 2% op 4 meter betekent dat er een maximale marge is van 8mm. Deze is 4x zo groot als de doelstelling.

De Software implementatie is mogelijk met het gebruik van de librealsense API.

## 9 Conclusie

Er is ondervonden dat een schade van 2mm of meer observeerbaar is en vanaf dit punt de erosie snelheid toeneemt.

Dit kan vervolgens geclassificeerd worden. Belangrijker is dat het mogelijk is dat de schade herkent kan worden door middel van een model waarbij eerst de schade als een object wordt gedetecteerd om vervolgens hier een masker op toe te passen zodat deze pixelgewijs gescheiden is van de rest van het object.

Er is genoeg data verzameld om een deep learning vorm op alle modellen toe te passen waardoor deze accuratesse toeneemt. Vervolgens is het de bedoeling dat deze doorgegeven wordt aan de camera software om deze schade ook exact te meten.

Waarna als laatste het geheel pakket een test implementatie kan doorstaan met als uiteindelijke hardware een NVIDIA Jetson AGX Xavier in combinatie met een Intel realsense D415 om hiermee real time te kunnen detecteren.

Dit geeft de kennis in samenhang met het framework voor een werkend prototype als resultaat. Waarbij er dus schade gedetecteerd wordt, om vervolgens deze schade te kunnen meten en het middelpunt te kunnen genereren.

Uiteindelijk om een suggestie te geven voor de dimensies van de reparatie.

## 10 Aanbevelingen

Uit het onderzoek is gebleken dat object detectie werkt maar veel valse positieven geeft. Hierom is het belangrijk dat het model opnieuw wordt getraind met alle 6131 foto's, met en zonder ruis. Het is ook mogelijk om hier nog een test opzet voor te maken met Gated SCNN om de nauwkeurigheid nog meer te verhogen.

Ook is er gebleken dat nauwkeurigheid van het meten belangrijk is maar dat de camera een maximale afwijking heeft van 8mm wat 4x zo groot is als de originele doelstelling.

Wegens een brede scoop van het onderzoek, uitgebreide theorie en beperkte stageduur is er geen test gemaakt van mask-rcnn om te kunnen te meten.

## 11 Bibliografie

### Referenties

- [1] *Activation Functions in Neural Networks*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.
- [2] *An end-to-end open source machine learning platform*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://www.tensorflow.org/>.
- [3] *Artificial Neural Network - Hopfield Networks*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: [https://www.tutorialspoint.com/artificial\\_neural\\_network/artificial\\_neural\\_network\\_hopfield.htm](https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_hopfield.htm).
- [4] *Artificial Neural Network (ANN)*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp>.
- [5] *Basics of Autoencoders*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f>.
- [6] *Boltzmann machine*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: [http://www.scholarpedia.org/article/Boltzmann\\_machine](http://www.scholarpedia.org/article/Boltzmann_machine).
- [7] *Caffe*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://caffe.berkeleyvision.org/>.
- [8] *D-LSM: Deep Liquid State Machine with unsupervised recurrent reservoir tuning*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://ieeexplore.ieee.org/document/7900035>.
- [9] *Deconvolutional Networks*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://www.matthewzeiler.com/mattzeiler/deconvolutionalnetworks.pdf>.
- [10] *Deep Boltzmann Machines*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <http://proceedings.mlr.press/v5/salakhutdinov09a/salakhutdinov09a.pdf>.
- [11] *Deep Convolutional Inverse Graphics Network*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://arxiv.org/abs/1503.03167>.
- [12] *Deep Convolutional Neural Networks*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://www.sciencedirect.com/topics/computer-science/deep-convolutional-neural-networks>.
- [13] *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <http://liangchiehchen.com/projects/DeepLab.html>.
- [14] *Denoising Autoencoders explained*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2>.
- [15] *Dilated Residual Networks*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: [https://openaccess.thecvf.com/content\\_cvpr\\_2017/papers/Yu\\_Dilated\\_Residual\\_Networks\\_CVPR\\_2017\\_paper.pdf](https://openaccess.thecvf.com/content_cvpr_2017/papers/Yu_Dilated_Residual_Networks_CVPR_2017_paper.pdf).
- [16] *Echo state network*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: [http://www.scholarpedia.org/article/Echo\\_state\\_network](http://www.scholarpedia.org/article/Echo_state_network).
- [17] *Epoch vs Batch Size vs Iterations*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>.

- [18] *Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a>.
- [19] *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://arxiv.org/abs/1506.01497>.
- [20] *FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://arxiv.org/abs/1903.11816>.
- [21] *Gated-SCNN: Gated Shape CNNs for Semantic Segmentation*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://arxiv.org/abs/1907.05740>.
- [22] *How U-net works?* [Online; accessed 6. jan 2021]. Jan 2021. URL: <https://developers.arcgis.com/python/guide/how-unet-works/>.
- [23] *Intel® RealSense™ Depth Camera D415*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://ark.intel.com/content/www/us/en/ark/products/128256/intel-realsense-depth-camera-d415.html>.
- [24] *Intel® RealSense™ Depth Camera D435i*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://ark.intel.com/content/www/us/en/ark/products/190004/intel-realsense-depth-camera-d435i.html>.
- [25] *Intel® RealSense™ Depth Camera D455*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://ark.intel.com/content/www/us/en/ark/products/205847/intel-realsense-depth-camera-d455.html>.
- [26] *Intel® RealSense™ Depth Camera L515*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://ark.intel.com/content/www/us/en/ark/products/201775/intel-realsense-lidar-camera-l515.html>.
- [27] *Kohonen Self-Organizing Maps*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://towardsdatascience.com/kohonen-self-organizing-maps-a29040d688da>.
- [28] *Mask R-CNN*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://arxiv.org/abs/1703.06870>.
- [29] *Neural Turing Machines*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://arxiv.org/abs/1410.5401>.
- [30] *Object Detection for Dummies Part 3: R-CNN Family*. [Online; accessed 28. Dec 2020]. Sep 2020. URL: <https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html>.
- [31] *Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction*. [Online; accessed 28. Dec 2020]. Sep 2020. URL: <https://arxiv.org/abs/1803.08707>.
- [32] *Rain erosion on the leading edge of wind turbines blades*. [Online; accessed 22. Dec 2020]. Sep 2020. URL: <https://www.semanticscholar.org/paper/Rain-erosion-on-the-leading-edge-of-wind-turbines-Marques-Teuwen/6ca1bbdf94bbb6013696ddd321c88bd2>.
- [33] *Recurrent Neural Networks*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce>.
- [34] *Restricted Boltzmann Machines — Simplified*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://towardsdatascience.com/restricted-boltzmann-machines-simplified-eable5878976>.
- [35] *scikit-learn*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://scikit-learn.org/stable/>.



- [36] *Stochastic Gradient Descent with momentum*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d>.
- [37] *Tensorflow Object Detection*. [Online; accessed 28. Dec 2020]. Sep 2020. URL: <https://medium.com/@rlancemartin/tensorflow-object-detection-72715c61e49a>.
- [38] *The Forgotten Algorithm*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://towardsdatascience.com/how-to-design-monte-carlo-simulation-138e9214910a>.
- [39] *Theano*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://github.com/Theano/Theano>.
- [40] *U-Net: Convolutional Networks for Biomedical Image Segmentation*. [Online; accessed 14. jan 2021]. Jan 2021. URL: <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>.
- [41] *Understand the Impact of Learning Rate on Neural Network Performance*. [Online; accessed 24. dec 2020]. Dec 2020. URL: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>.
- [42] *Understanding Generative Adversarial Networks (GANs)*. [Online; accessed 7. Okt 2020]. Okt 2020. URL: <https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29>.
- [43] *VGG16 – Convolutional Network for Classification and Detection*. [Online; accessed 5. jan 2021]. Jan 2021. URL: <https://neurohive.io/en/popular-networks/vgg16/>.
- [44] *Water Droplet Erosion of Wind Turbine Blades:Mechanics, Testing, Modeling andFuture Perspectives*. [Online; accessed 22. Dec 2020]. Sep 2020. URL: [https://www.researchgate.net/publication/320089890\\_On\\_the\\_Material\\_Characterisation\\_of\\_Wind\\_Turbine\\_Blade\\_Coatings\\_The\\_Effect\\_of\\_Interphase\\_Coating-Laminate\\_Adhesion\\_on\\_Rain\\_Erosion\\_Performance/link/59cda211458515cc6aa4bd56/download](https://www.researchgate.net/publication/320089890_On_the_Material_Characterisation_of_Wind_Turbine_Blade_Coatings_The_Effect_of_Interphase_Coating-Laminate_Adhesion_on_Rain_Erosion_Performance/link/59cda211458515cc6aa4bd56/download).
- [45] *What is Torch?* [Online; accessed 5. jan 2021]. Jan 2021. URL: <http://torch.ch/>.
- [46] *YOLO v4: Optimal Speed Accuracy for object detection*. [Online; accessed 28. Dec 2020]. Sep 2020. URL: <https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>.
- [47] *YOLOv3: An Incremental Improvement*. [Online; accessed 28. Dec 2020]. Sep 2020. URL: <https://pjreddie.com/media/files/papers/YOLOv3.pdf>.

## Lijst van figuren

1	levels erosie[44]	8
2	verschillende soorten schaden[32]	8
3	neuron[18]	10
4	learning rate voorbeeld[18]	11
5	sigmoid en ReLu functie[1]	12
6	vgg16	17
7	2-stappen versus 1-stap[31]	18
8	classificatie correct erosie	28
9	classificatie correct geen erosie	28
10	accuratesse	29
11	FastFCN architecture	33
12	Unet [22]	34
13	foto 1 object detecteren	36
14	foto 2 object detecteren	37
15	foto 3 object detecteren	38
16	Gemaskeerde schade	42
17	Meten van pixels	43
18	Voorbeeld freespatroon	43
19	r1t2	55
20	r1t3	59
21	r1t4	63
22	r1t5	67
23	r1t6	71
24	r1t7	75
25	r1t8	79
26	r1t9	84
27	r1t10	88
28	r1t11	93

## 12 Bijlagen

### 12.1 Tussen tijdse resultaten

### 12.2 Code en resultaat

#### 12.2.1 R1T2

```
#install_keras(  
#  method = c("auto", "virtualenv", "conda"),  
#  conda = "auto",  
#  version = "default",  
#  tensorflow = "default",  
#  extra_packages = c("tensorflow-hub")  
#)  
  
#to setup the Keras library and TensorFlow bac  
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE  
  
# -----  
  
##if (!requireNamespace('BiocManager', quietly = TRUE))  
## install.packages('BiocManager')  
##BiocManager::install('EBImage')  
#install.packages("jpeg")  
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")  
  
library(keras)  
library(EBImage)  
library(jpeg)  
library(tensorflow)  
  
#####  
#DATASET EXPLORATION  
#####  
  
getwd()  
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER  
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp  
  
# Reading  
##print(img) # Print  
  
##getFrames(img, type='total') # Split  
  
##display(img)  
# Display image  
#####  
#EN OF DATASET EXPLORATION  
#####
```

```
#####
#####CREATE LIST OF RAT#####
#####

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua"
img.doggo<- sample(dir(currentdir));
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
}

chiuhuahalist<- doggolist

#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir));
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
}

scottishdeerhound<- doggolist

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chiuhuahalist[1:40], scottishdeerhound[1:40])

train<-aperm(combine(train_pool), c(4,1,2,3))
test_pool<-c(chiuhuahalist[41:43], scottishdeerhound[41:43])

test<-aperm(combine(test_pool), c(4,1,2,3))
```

```
#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default
```

```
#####
##### ONE HOT ENCODING #####
#####
```

```
#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Categorical vector for training
#classes
test_lab<- to_categorical(test_y) #Categorical vector for test classes
```

```
#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####
```

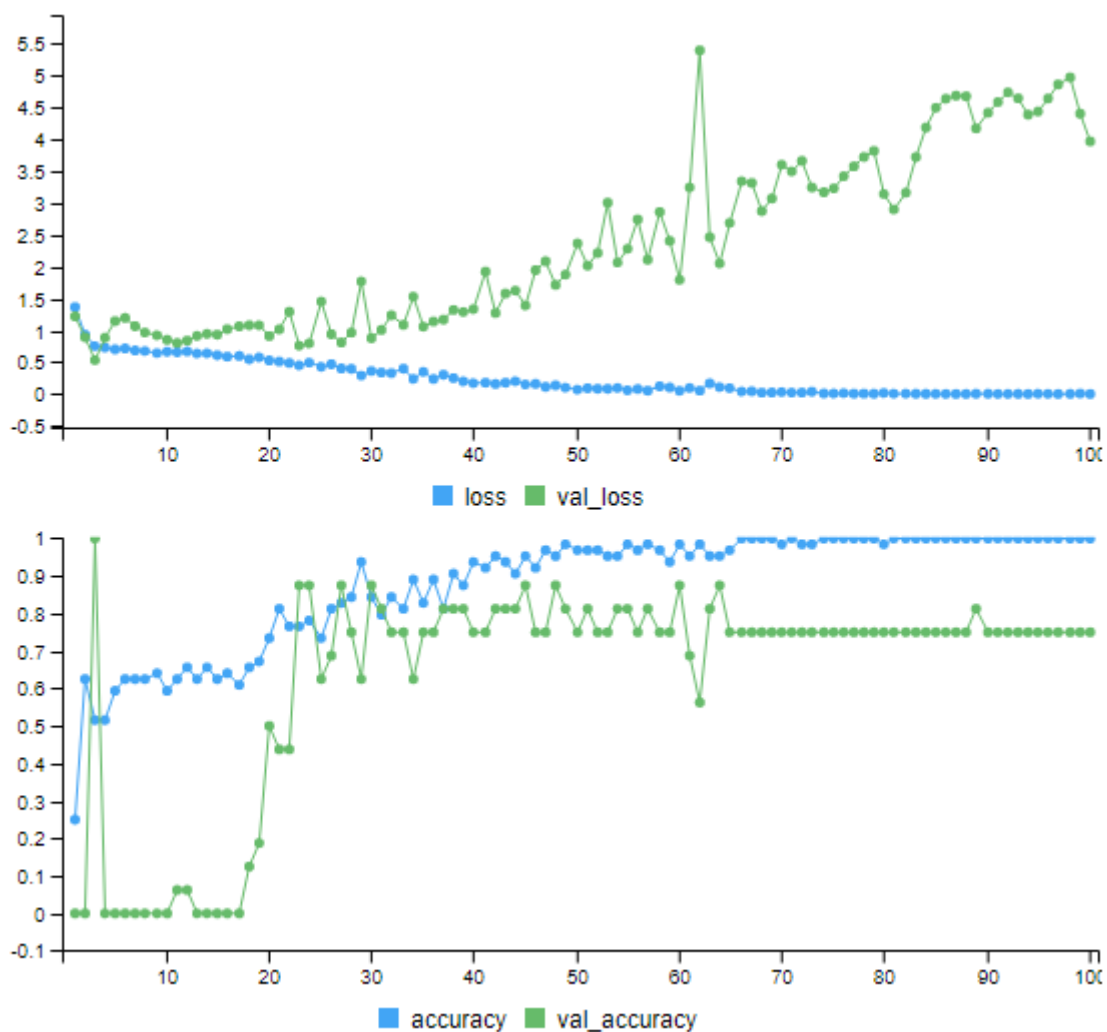
```
# Model Building
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#----linear stack of layers
model.fancydoggos %>% #-----Initiate and connect to #-----
  layer_conv_2d(filters = 40, #-----First convoluted layer
    kernel_size = c(4,4), #---40 Filters with dimension 4x4
    activation = 'relu', #--with a ReLu activation function
    input_shape = c(100,100,3)) %>%
  #------(B)-----#
  layer_conv_2d(filters = 40, #-----Second convoluted layer
    kernel_size = c(4,4), #---40 Filters with dimension 4x4
    activation = 'relu') %>% #--with a ReLu activation function
  #------(C)-----#
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.25) %>% #-----Drop out layer
  #------(D)-----#
  layer_conv_2d(filters = 80, #-----Third convoluted layer
    kernel_size = c(4,4), #---80 Filters with dimension 4x4
    activation = 'relu') %>% #--with a ReLu activation function
  #------(E)-----#
  layer_conv_2d(filters = 80, #-----Fourth convoluted layer
    kernel_size = c(4,4), #---80 Filters with dimension 4x4
    activation = 'relu') %>% #--with a ReLu activation function
  #------(F)-----#
  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
  #-----#
```

```

layer_dropout(rate = 0.35) %>% #-----Drop out layer
#------(G)-----#
layer_flatten() %>% #---Flattening the final stack of feature maps
#------(H)-----#
layer_dense(units = 256, activation = 'relu') %>% #----Hidden layer
#------(I)-----#
layer_dropout(rate = 0.25) %>% #-----Drop-out layer
#-----#
layer_dense(units = 4, activation = "softmax") %>% #----Final Layer
#------(J)-----#
compile(loss = 'categorical_crossentropy',
optimizer = optimizer_adam(),
metrics = c("accuracy")) # Compiling the architecture

#####
##### MODEL FITTING CHART #####
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
      train_lab,
      epochs = 100,
      batch_size = 40,
      validation_split = 0.2
  )

```



Figuur 19: r1t2

### 12.2.2 R1T3

```
#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow back
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE

# -----

##if (!requireNamespace('BiocManager', quietly = TRUE))
##  install.packages('BiocManager')
##BiocManager::install('EBImage')
```

```

#install.packages("jpeg")
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp

# Reading
##print(img) # Print

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

# To acces

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

chihuahahalist<- doggolist # Storin

#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

# To access the images of Clubs suit.

```



```

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100) #resizing
}

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40])

# The fir
# are inc
train<-aperm(combine(train_pool), c(4,1,2,3)) # Combine
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43]) # Vector
# 3 image
# in test
test<-aperm(combine(test_pool), c(4,1,2,3)) # Combine

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODIG #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes

#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####

```

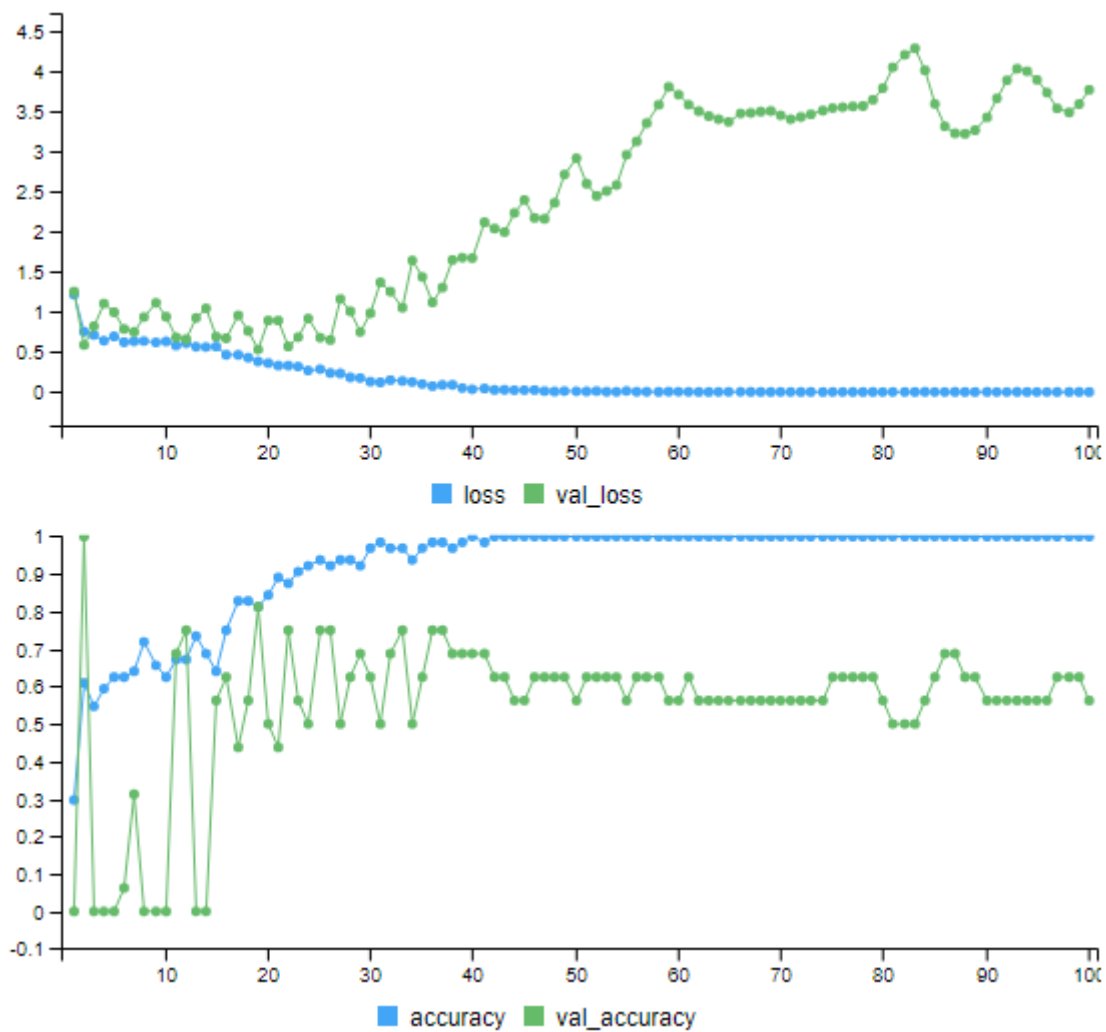
```

# Model Building
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#-----linear stack of layers
model.fancydoggos %>%
  layer_conv_2d(filters = 40, #-----Initiate and connect to #-----
                kernel_size = c(4,4), #-----First convoluted layer
                activation = 'relu', #---40 Filters with dimension 4x4
                input_shape = c(100,100,3)) %>% #--with a ReLu activation function
  #------(B)-----#
  layer_conv_2d(filters = 40, #-----Second convoluted layer
                kernel_size = c(4,4), #---40 Filters with dimension 4x4
                activation = 'relu') %>% #--with a ReLu activation function
  #------(C)-----#
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.25) %>% #-----Drop out layer
  #------(D)-----#
  layer_conv_2d(filters = 80, #-----Third convoluted layer
                kernel_size = c(4,4), #---80 Filters with dimension 4x4
                activation = 'relu') %>% #--with a ReLu activation function
  #------(E)-----#
  #layer_conv_2d(filters = 80, #-----Fourth convoluted layer
  #kernel_size = c(4,4), #---80 Filters with dimension 4x4
  #activation = 'relu') %>% #--with a ReLu activation function
  #------(F)-----#
  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.35) %>% #-----Drop out layer
  #------(G)-----#
  layer_flatten()%>% #---Flattening the final stack of feature maps
  #------(H)-----#
  layer_dense(units = 256, activation = 'relu')%>% #----Hidden layer
  #------(I)-----#
  layer_dropout(rate= 0.25)%>% #-----Drop-out layer
  #-----#
  layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
  #------(J)-----#
  compile(loss = 'categorical_crossentropy',
          optimizer = optimizer_adam(),
          metrics = c("accuracy")) # Compiling the architecture

#####
##### MODEL FITTING CHART #####
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
      train_lab,
      epochs = 100,
      batch_size = 40,
      validation_split = 0.2

```

)



Figuur 20: r1t3

### 12.2.3 R1T4

```
#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

# -----

##if (!requireNamespace('BiocManager', quietly = TRUE))
#  #to setup the Keras library and TensorFlow bac
#  #INSTALL CPU VERSION OF KE
```

```

## install.packages('BiocManager')
##BiocManager::install('EBImage')
install.packages("jpeg")
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jpg")

##print(img) # Reading # Print

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

#####
# To acce

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 300, 300)
} #resizing

chihuahalist<- doggolist # Storin

#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

```

```

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 300, 300)
} #resizing

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40])

# The fir
# are inc
# Combine
train<-aperm(combine(train_pool), c(4,1,2,3))
# Vector
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43])
# 3 image
# in test
test<-aperm(combine(test_pool), c(4,1,2,3))
# Combine

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODIG #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes

#####

```

```

##### BEGIN WITH THE ACTUAL MODEL #####
#####

# Model Building
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#----linear stack of layers
model.fancydoggos %>% #-----Initiate and connect to #-----
  layer_conv_2d(filters = 40, #-----First convoluted layer
    kernel_size = c(4,4), #---40 Filters with dimension 4x4
    activation = 'relu', #--with a ReLu activation function
    input_shape = c(300,300,3)) %>%
  #------(B)-----#
  layer_conv_2d(filters = 40, #-----Second convoluted layer
    kernel_size = c(4,4), #---40 Filters with dimension 4x4
    activation = 'relu') %>% #--with a ReLu activation function
  #------(C)-----#
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.25) %>% #-----Drop out layer
  #------(D)-----#
  layer_conv_2d(filters = 80, #-----Third convoluted layer
    kernel_size = c(4,4), #---80 Filters with dimension 4x4
    activation = 'relu') %>% #--with a ReLu activation function
  #------(E)-----#
  #layer_conv_2d(filters = 80, #-----Fourth convoluted layer
  #kernel_size = c(4,4), #---80 Filters with dimension 4x4
  #activation = 'relu') %>% #--with a ReLu activation function
  #------(F)-----#
  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.35) %>% #-----Drop out layer
  #------(G)-----#
  layer_flatten()%>% #---Flattening the final stack of feature maps
  #------(H)-----#
  layer_dense(units = 256, activation = 'relu')%>% #----Hidden layer
  #------(I)-----#
  layer_dropout(rate= 0.25)%>% #-----Drop-out layer
  #-----#
  layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
  #------(J)-----#
  compile(loss = 'categorical_crossentropy',
    optimizer = optimizer_adam(),
    metrics = c("accuracy")) # Compiling the architecture

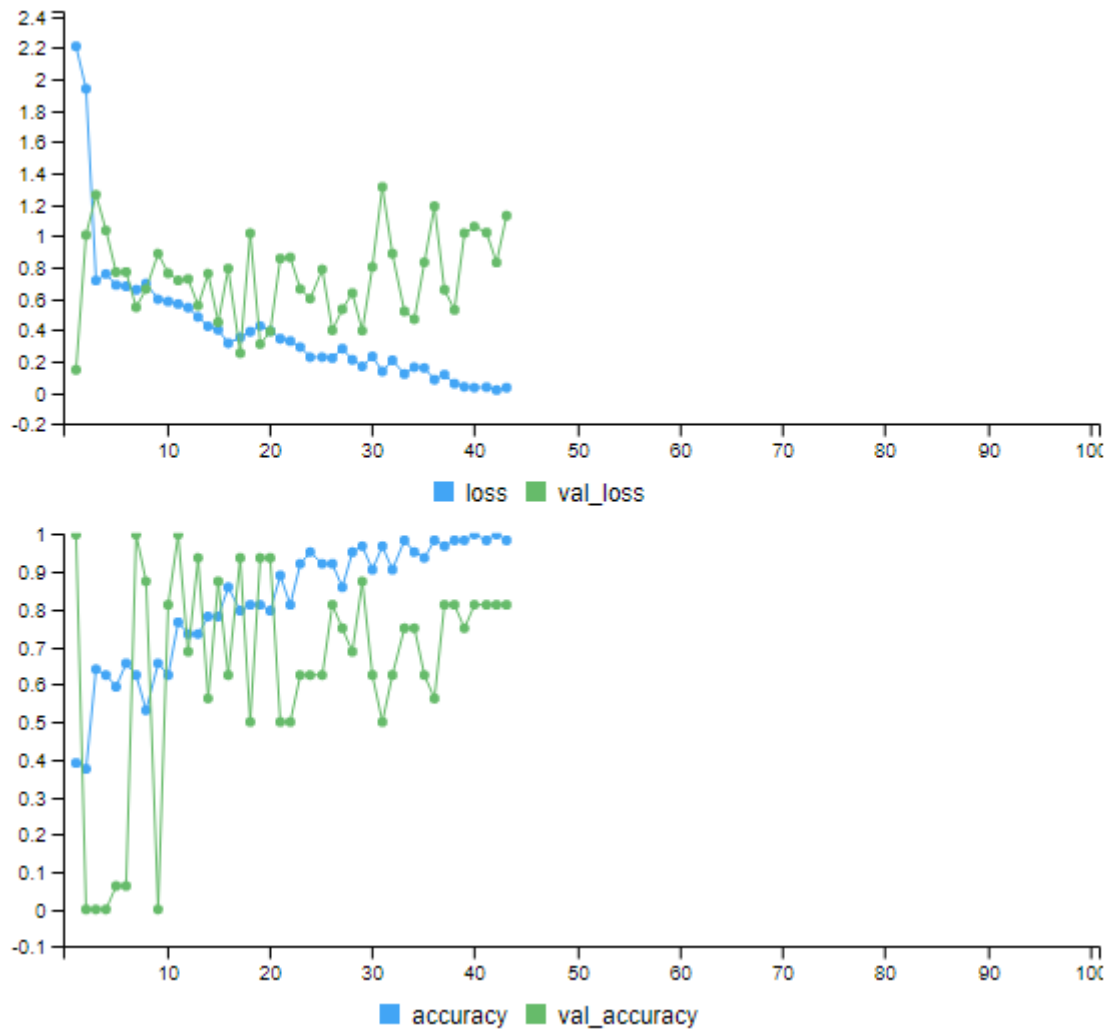
#####
##### MODEL FITTING CHART #####
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
    train_lab,
    epochs = 100,

```

```

    batch_size = 40,
    validation_split = 0.2
)

```



Figuur 21: r1t4

## 12.2.4 R1T5

```

#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow back
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE

# -----

```

```

##if (!requireNamespace('BiocManager', quietly = TRUE))
##  install.packages('BiocManager')
##BiocManager::install('EBImage')
install.packages("jpeg")
#reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp

##print(img) # Reading # Print

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

#####
# To acce

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

chihuahalist<- doggolist # Storin

#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

```



```

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40])

# The fir
# are inc
# Combine
# Vector
# 3 image
# in test
# Combine

train<-aperm(combine(train_pool), c(4,1,2,3))
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43])

test<-aperm(combine(test_pool), c(4,1,2,3))

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODIG #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes

```

```
#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####

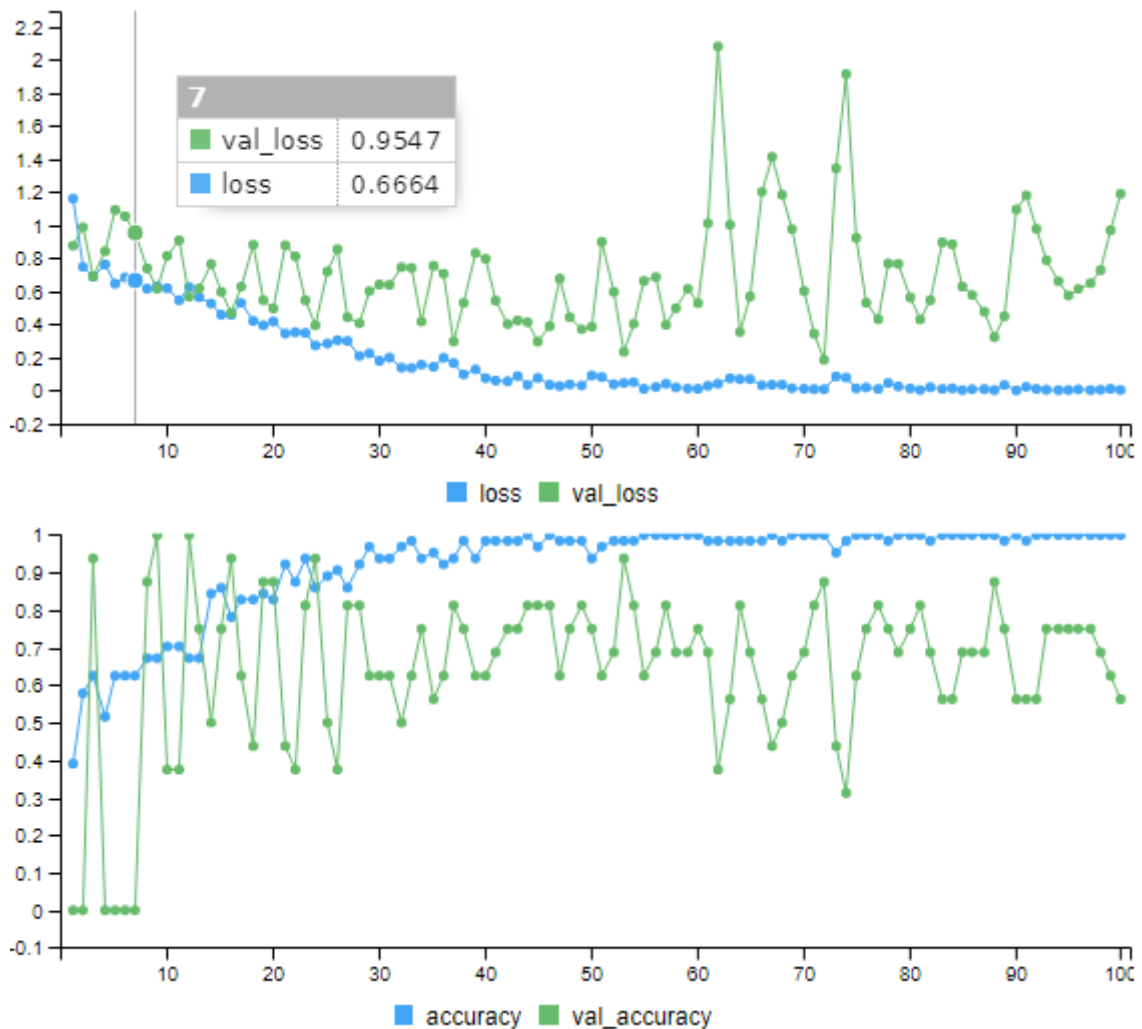
# Model Building
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#----linear stack of layers
model.fancydoggos %>% #-----Initiate and connect to #-----
  layer_conv_2d(filters = 40, #-----First convoluted layer
    kernel_size = c(4,4), #---40 Filters with dimension 4x4
    activation = 'relu', #--with a ReLu activation function
    input_shape = c(100,100,3)) %>%
    #------(B)-----#
    layer_conv_2d(filters = 40, #-----Second convoluted layer
      kernel_size = c(4,4), #---40 Filters with dimension 4x4
      activation = 'relu') %>% #--with a ReLu activation function
    #------(C)-----#
    layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
    #-----#
    layer_dropout(rate = 0.5) %>% #-----Drop out layer
    #------(D)-----#
    layer_conv_2d(filters = 80, #-----Third convoluted layer
      kernel_size = c(4,4), #---80 Filters with dimension 4x4
      activation = 'relu') %>% #--with a ReLu activation function
    #------(E)-----#
    #layer_conv_2d(filters = 80, #-----Fourth convoluted layer
    #kernel_size = c(4,4), #---80 Filters with dimension 4x4
    #activation = 'relu') %>% #--with a ReLu activation function
    #------(F)-----#
    layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
    #-----#
    layer_dropout(rate = 0.35) %>% #-----Drop out layer
    #------(G)-----#
    layer_flatten()%>% #---Flattening the final stack of feature maps
    #------(H)-----#
    layer_dense(units = 256, activation = 'relu')%>% #----Hidden layer
    #------(I)-----#
    layer_dropout(rate= 0.25)%>% #-----Drop-out layer
    #-----#
    layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
    #------(J)-----#
    compile(loss = 'categorical_crossentropy',
      optimizer = optimizer_adam(),
      metrics = c("accuracy")) # Compiling the architecture

#####
##### MODEL FITTING CHART #####
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
    train_lab,
```

```

epochs = 100,
batch_size = 40,
validation_split = 0.2
)

```



Figuur 22: r1t5

### 12.2.5 R1T6

```

#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow bac
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE

# -----

```

```

##if (!requireNamespace('BiocManager', quietly = TRUE))
##  install.packages('BiocManager')
##BiocManager::install('EBImage')
install.packages("jpeg")
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp

# Reading
# Print

##print(img)

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

# To acces

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The path
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

chihuahualist<- doggolist # Storing

#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

```

```

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40])

train<-aperm(combine(train_pool), c(4,1,2,3))
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43])

test<-aperm(combine(test_pool), c(4,1,2,3))

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODIG #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes

```

```
#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####

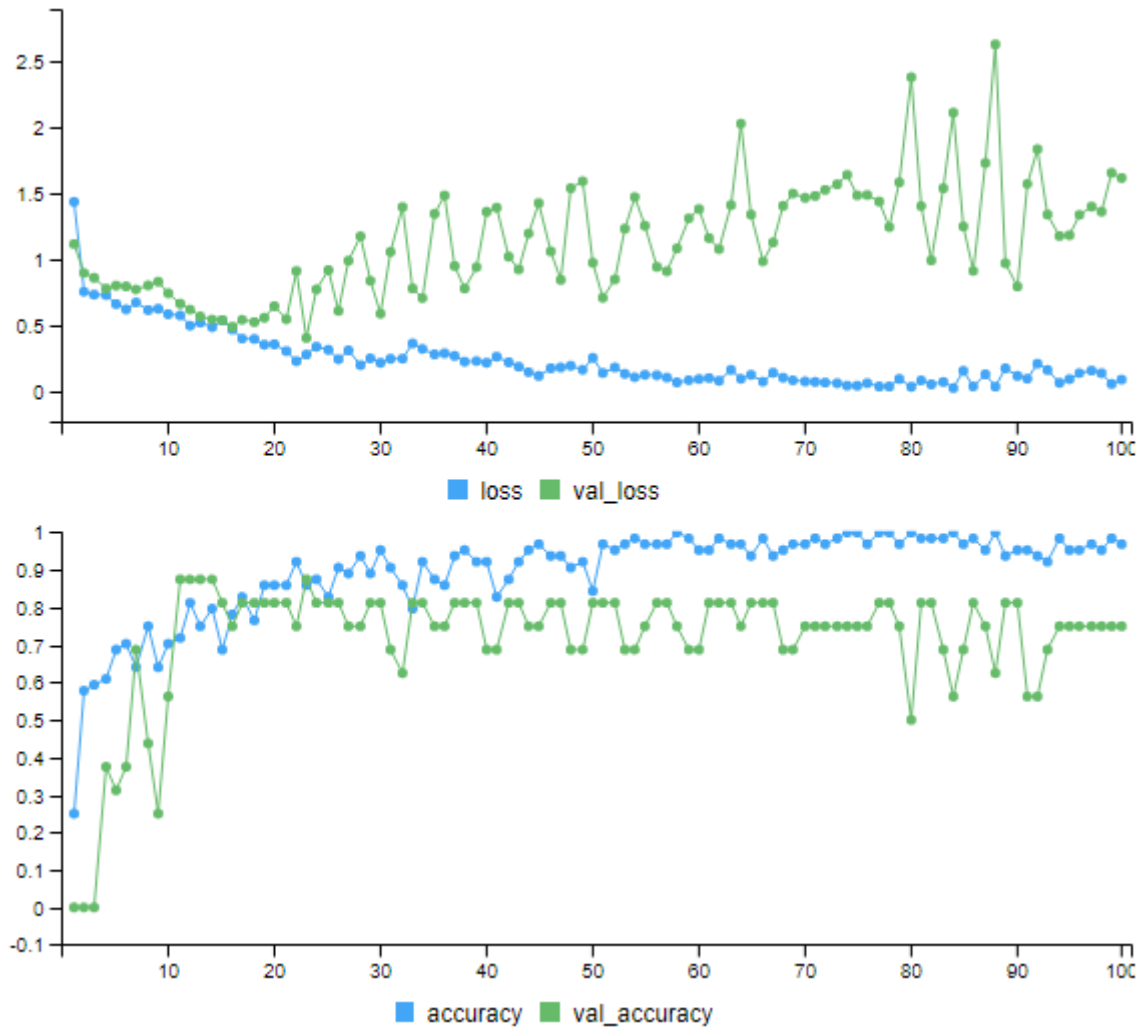
# Model Building
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#----linear stack of layers
model.fancydoggos %>% #-----Initiate and connect to #-----
  layer_conv_2d(filters = 40, #-----First convoluted layer
kernel_size = c(4,4), #---40 Filters with dimension 4x4
activation = 'relu', #--with a ReLu activation function
input_shape = c(100,100,3)) %>%
  #------(B)-----#
  layer_conv_2d(filters = 40, #-----Second convoluted layer
kernel_size = c(4,4), #---40 Filters with dimension 4x4
activation = 'relu') %>% #--with a ReLu activation function
  #------(C)-----#
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.5) %>% #-----Drop out layer
  #------(D)-----#
  layer_conv_2d(filters = 80, #-----Third convoluted layer
kernel_size = c(4,4), #---80 Filters with dimension 4x4
activation = 'relu') %>% #--with a ReLu activation function
  #------(E)-----#
  #layer_conv_2d(filters = 80, #-----Fourth convoluted layer
#kernel_size = c(4,4), #---80 Filters with dimension 4x4
#activation = 'relu') %>% #--with a ReLu activation function
  #------(F)-----#
  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.35) %>% #-----Drop out layer
  #------(G)-----#
  layer_flatten()%>% #---Flattening the final stack of feature maps
  #------(H)-----#
  layer_dense(units = 256, activation = 'relu')%>% #----Hidden layer
  #------(I)-----#
  layer_dropout(rate= 0.5)%>% #-----Drop-out layer
  #-----#
  layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
  #------(J)-----#
  compile(loss = 'categorical_crossentropy',
optimizer = optimizer_adam(),
metrics = c("accuracy")) # Compiling the architecture

#####
##### MODEL FITTING CHART #####
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
```

```

train_lab,
epochs = 100,
batch_size = 40,
validation_split = 0.2
)

```



Figuur 23: r1t6

## 12.2.6 R1T7

```

#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow bac
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE

# -----

```

```

##if (!requireNamespace('BiocManager', quietly = TRUE))
##  install.packages('BiocManager')
##BiocManager::install('EBImage')
#install.packages("jpeg")
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp # Reading
                                                                # Print

##print(img)

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

                                                                # To acce

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

chihuahahalist<- doggolist # Storin

#####

```



```

#####CREATE LIST OF SCOTTISCH DEERHOUND#####
#####

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100) #resizing
}

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40])

train<-aperm(combine(train_pool), c(4,1,2,3))
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43])

test<-aperm(combine(test_pool), c(4,1,2,3))

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODIG #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes

```

```
test_lab<- to_categorical(test_y)#Catagorical vector for test classes
```

```
#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####
```

```
# Model Building
```

```
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
```

```
#----linear stack of layers
```

```
model.fancydoggos %>%
```

```
#-----Initiate and connect to #-----
```

```
  layer_conv_2d(filters = 40,
```

```
#-----First convoluted layer
```

```
  kernel_size = c(4,4),
```

```
#--40 Filters with dimension 4x4
```

```
  activation = 'relu',
```

```
#-with a ReLu activation function
```

```
  input_shape = c(100,100,3)) %>%
```

```
#------(B)-----#
```

```
  layer_conv_2d(filters = 40,
```

```
#-----Second convoluted layer
```

```
  kernel_size = c(4,4),
```

```
#--40 Filters with dimension 4x4
```

```
  activation = 'relu') %>%
```

```
#-with a ReLu activation function
```

```
#------(C)-----#
```

```
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
```

```
#-----#
```

```
  layer_dropout(rate = 0.5) %>% #-----Drop out layer
```

```
#------(D)-----#
```

```
  layer_conv_2d(filters = 80,
```

```
#-----Third convoluted layer
```

```
  kernel_size = c(4,4),
```

```
#--80 Filters with dimension 4x4
```

```
  activation = 'relu') %>%
```

```
#-with a ReLu activation function
```

```
#------(E)-----#
```

```
  #layer_conv_2d(filters = 80,
```

```
#-----Fourth convoluted layer
```

```
  #kernel_size = c(4,4),
```

```
#--80 Filters with dimension 4x4
```

```
  #activation = 'relu') %>%
```

```
#-with a ReLu activation function
```

```
#------(F)-----#
```

```
  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
```

```
#-----#
```

```
  layer_dropout(rate = 0.6) %>% #-----Drop out layer
```

```
#------(G)-----#
```

```
  layer_flatten()%>% #---Flattening the final stack of feature maps
```

```
#------(H)-----#
```

```
  layer_dense(units = 256, activation = 'relu')%>% #----Hidden layer
```

```
#------(I)-----#
```

```
  layer_dropout(rate= 0.25)%>% #-----Drop-out layer
```

```
#-----#
```

```
  layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
```

```
#------(J)-----#
```

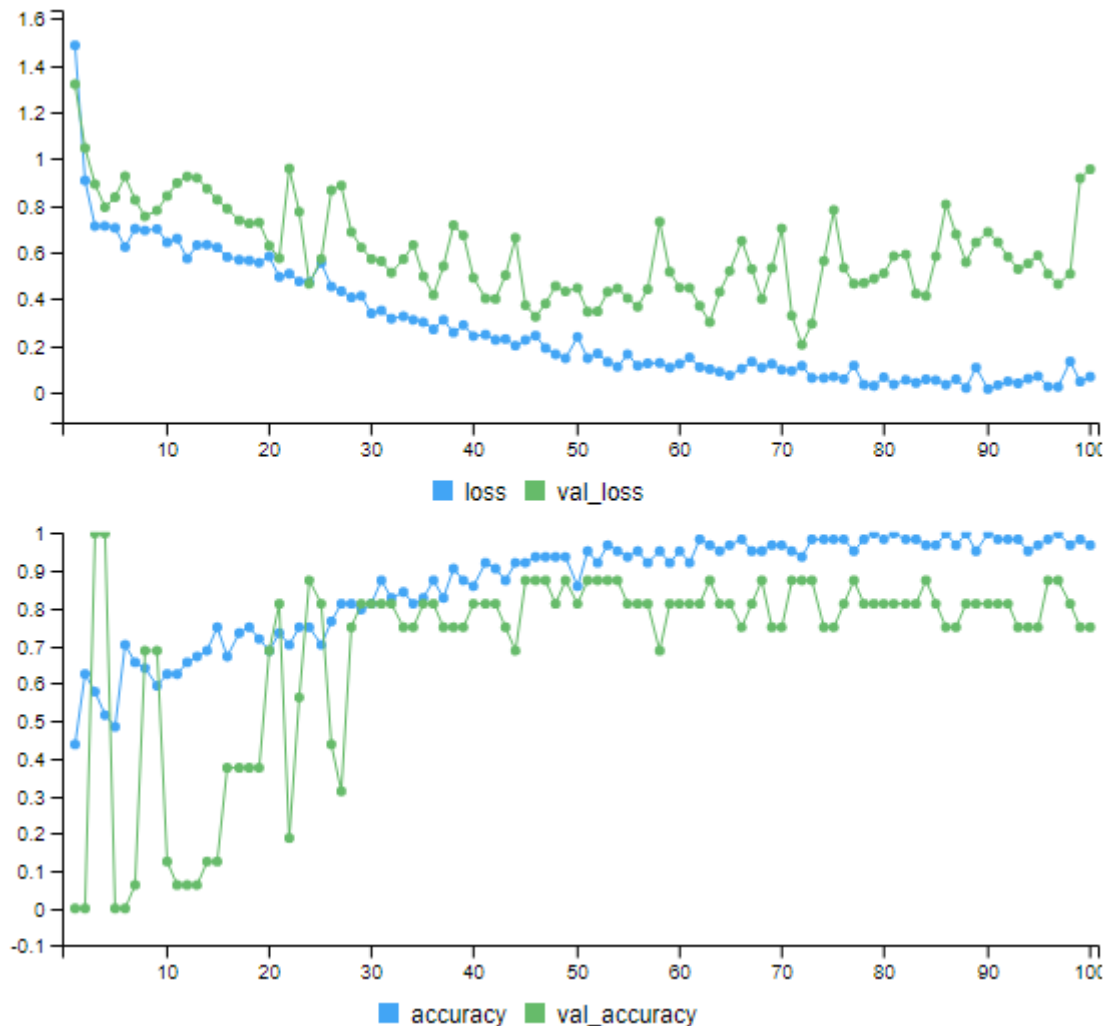
```
  compile(loss = 'categorical_crossentropy',
```

```
  optimizer = optimizer_adam(),
```

```
  metrics = c("accuracy")) # Compiling the architecture
```

```
#####
##### MODEL FITTING CHART #####
#####
#fit model
```

```
history<- model.fancydoggos %>%
  fit(train,
      train_lab,
      epochs = 100,
      batch_size = 40,
      validation_split = 0.2
  )
```



Figuur 24: r1t7

### 12.2.7 R1T8

```
#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow bac
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE
```

```

# -----

##if (!requireNamespace('BiocManager', quietly = TRUE))
## install.packages('BiocManager')
##BiocManager::install('EBImage')
install.packages("jpeg")
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp

##print(img) # Reading
# Print

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

# To acce

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

chihuahahalist<- doggolist # Storin

```

```
#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40])

train<-aperm(combine(train_pool), c(4,1,2,3))
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43])

test<-aperm(combine(test_pool), c(4,1,2,3))

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODING #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
```

```

#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes

#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####

# Model Building
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#----linear stack of layers
model.fancydoggos %>% #-----Initiate and connect to #-----
  layer_conv_2d(filters = 40, #-----First convoluted layer
    kernel_size = c(4,4), #---40 Filters with dimension 4x4
    activation = 'relu', #--with a ReLu activation function
    input_shape = c(100,100,3)) %>%
    #------(B)-----#
    layer_conv_2d(filters = 40, #-----Second convoluted layer
      kernel_size = c(4,4), #---40 Filters with dimension 4x4
      activation = 'relu') %>% #--with a ReLu activation function
      #------(C)-----#
      layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
      #-----#
      layer_dropout(rate = 0.5) %>% #-----Drop out layer
      #------(D)-----#
      layer_conv_2d(filters = 80, #-----Third convoluted layer
        kernel_size = c(4,4), #---80 Filters with dimension 4x4
        activation = 'relu') %>% #--with a ReLu activation function
        #------(E)-----#
        #layer_conv_2d(filters = 80, #-----Fourth convoluted layer
        #kernel_size = c(4,4), #---80 Filters with dimension 4x4
        #activation = 'relu') %>% #--with a ReLu activation function
        #------(F)-----#
        layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
        #-----#
        layer_dropout(rate = 0.6) %>% #-----Drop out layer
        #------(G)-----#
        layer_flatten()%>% #---Flattening the final stack of feature maps
        #------(H)-----#
        layer_dense(units = 256, activation = 'relu')%>% #----Hidden layer
        #------(I)-----#
        layer_dropout(rate= 0.5)%>% #-----Drop-out layer
        #-----#
        layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
        #------(J)-----#
        compile(loss = 'categorical_crossentropy',
          optimizer = optimizer_adam(),
          metrics = c("accuracy")) # Compiling the architecture

#####
##### MODEL FITTING CHART #####
#####

```

```
#fit model
history<- model.fancydoggos %>%
  fit(train,
      train_lab,
      epochs = 100,
      batch_size = 40,
      validation_split = 0.2
  )
```

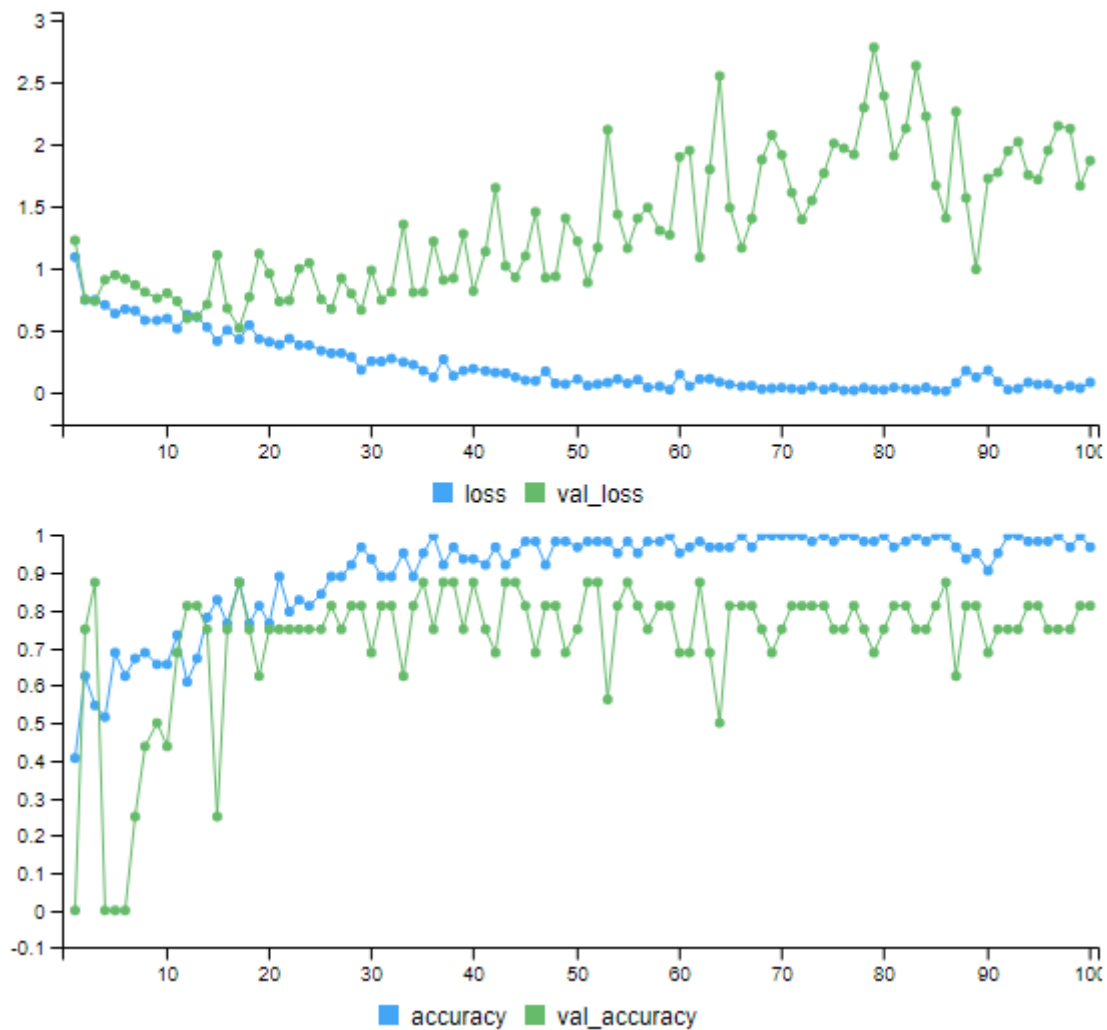


Figure 25: r1t8

### 12.2.8 R1T9

```
#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow back
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE
```

```

# -----

##if (!requireNamespace('BiocManager', quietly = TRUE))
##  install.packages('BiocManager')
##BiocManager::install('EBImage')
#install.packages("jpeg")
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp # Reading
##print(img) # Print

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

# To acce

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
} #resizing

chihuahualist<- doggolist # Storing

```



```
#####
#####CREATE LIST OF SCOTTISCH DEERHOUND#####
#####

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100) #resizing
}

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40]) # The fir
# are inc
train<-aperm(combine(train_pool), c(4,1,2,3)) # Combine
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43]) # Vector
# 3 image
# in test
# Combine

test<-aperm(combine(test_pool), c(4,1,2,3))

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODING #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
```

```
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes
```

```
#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####
```

```
# Model Building
```

```
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#-----linear stack of layers
```

```
model.fancydoggos %>% #-----Initiate and connect to #-----
```

```
  layer_conv_2d(filters = 40, #-----First convoluted layer
```

```
  kernel_size = c(4,4), #---40 Filters with dimension 4x4
```

```
  activation = 'relu', #--with a ReLu activation function
```

```
  input_shape = c(100,100,3)) %>%
```

```
  #------(B)-----#
```

```
  layer_conv_2d(filters = 40, #-----Second convoluted layer
```

```
  kernel_size = c(4,4), #---40 Filters with dimension 4x4
```

```
  activation = 'relu') %>% #--with a ReLu activation function
```

```
  #------(C)-----#
```

```
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
```

```
  #-----#
```

```
  layer_dropout(rate = 0.5) %>% #-----Drop out layer
```

```
  #------(D)-----#
```

```
  layer_conv_2d(filters = 80, #-----Third convoluted layer
```

```
  kernel_size = c(4,4), #---80 Filters with dimension 4x4
```

```
  activation = 'relu') %>% #--with a ReLu activation function
```

```
  #------(E)-----#
```

```
  layer_conv_2d(filters = 80, #-----Fourth convoluted layer
```

```
  kernel_size = c(4,4), #---80 Filters with dimension 4x4
```

```
  activation = 'relu') %>% #--with a ReLu activation function
```

```
  #------(F)-----#
```

```
  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
```

```
  #-----#
```

```
  layer_dropout(rate = 0.6) %>% #-----Drop out layer
```

```
  #------(G)-----#
```

```
  layer_flatten()%>% #---Flattening the final stack of feature maps
```

```
  #------(H)-----#
```

```
  layer_dense(units = 256, activation = 'relu')%>% #----Hidden layer
```

```
  #------(I)-----#
```

```
  layer_dropout(rate= 0.5)%>% #-----Drop-out layer
```

```
  #-----#
```

```
  layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
```

```
  #------(J)-----#
```

```
  compile(loss = 'categorical_crossentropy',
```

```
  optimizer = optimizer_adam(),
```

```
  metrics = c("accuracy")) # Compiling the architecture
```

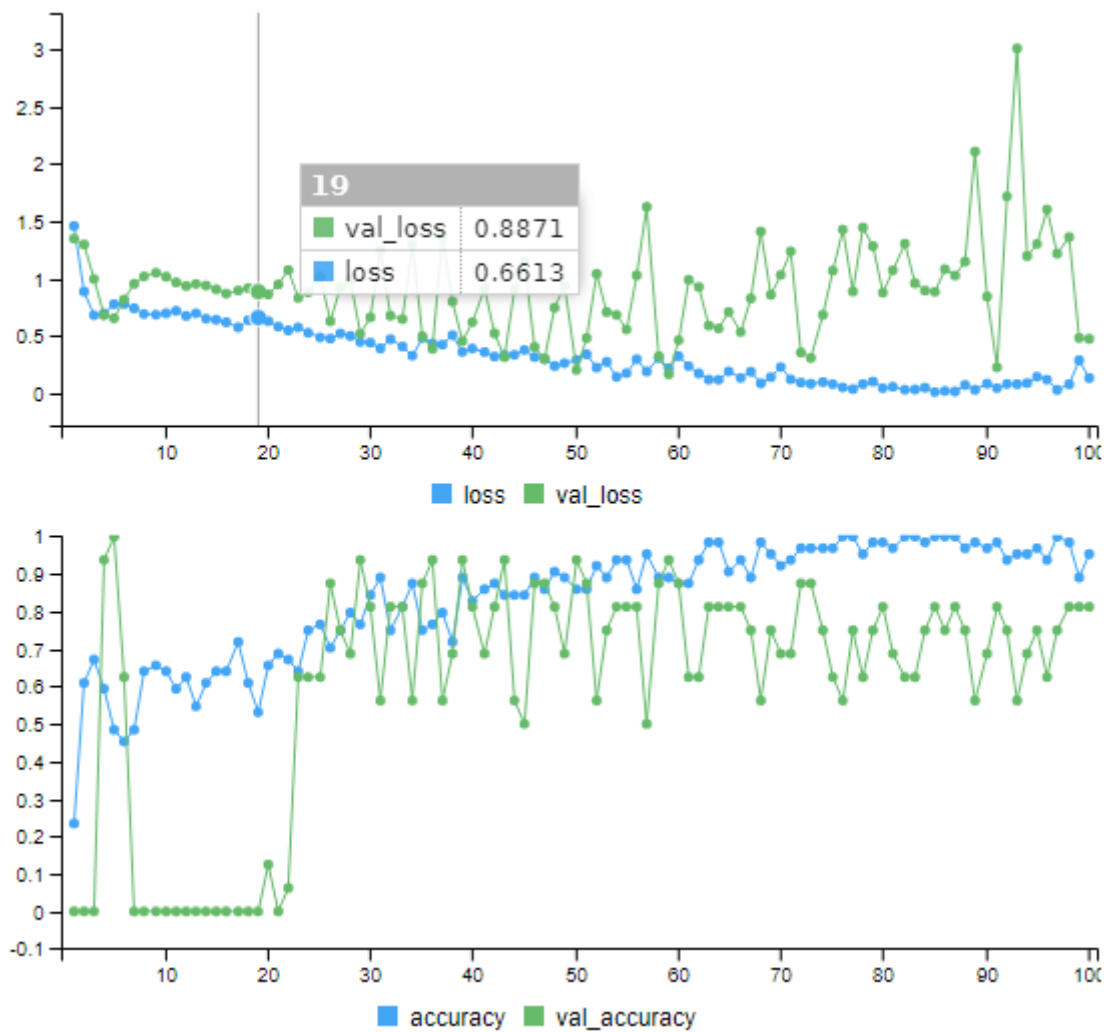
```
#####
##### MODEL FITTING CHART #####
```

```
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
      train_lab,
      epochs = 100,
      batch_size = 40,
      validation_split = 0.2
  )

#Model Evaluation
#model.fancydoggos %>% evaluate(train,train_lab) #Evaluat
#pred<- model.fancydoggos %>% predict_classes(train) #
#Train_Result<-table(Predicted = pred, Actual = train_y) #---Res

#model.fancydoggos %>% evaluate(test, test_lab) #----Ev
#pred1<- model.fancydoggos %>% predict_classes(test) #----Cl
#Test_Result<-table(Predicted = pred1, Actual = test_y) #----Resu

#rownames(Train_Result)<-rownames(Test_Result)<-colnames(Train_Result)<-colnames(Test_Res
#print(Train_Result)
#print(Test_Result)
```



Figuur 26: r1t9

### 12.2.9 R1T10

```
#install_keras(
#  method = c("auto", "virtualenv", "conda"),
#  conda = "auto",
#  version = "default",
#  tensorflow = "default",
#  extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow bac
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE

# -----

##if (!requireNamespace('BiocManager', quietly = TRUE))
##  install.packages('BiocManager')
##BiocManager::install('EBImage')
#install.packages("jpeg")
```

```

##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp # Reading
# Print

##print(img) # Print

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

# To acces

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100) #resizing
}

chihuahalist<- doggolist # Storin

#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"

```

```

setwd(currentdir) # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
  doggolist[[i]]<- readImage(img.doggo[i])
  doggolist[[i]]<- resize(doggolist[[i]], 100, 100) #resizing
}

scottishdeerhound<- doggolist # St

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chihuahahalist[1:40], scottishdeerhound[1:40])

train<-aperm(combine(train_pool), c(4,1,2,3)) # The fir
test_pool<-c(chihuahahalist[41:43], scottishdeerhound[41:43]) # are inc
# Combine
# Vector
# 3 image
# in test
# Combine

test<-aperm(combine(test_pool), c(4,1,2,3))

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
  plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####
##### ONE HOT ENCODIG #####
#####

#one hot encoding
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes

#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####

```

```

# Model Building
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#-----linear stack of layers
model.fancydoggos %>% #-----Initiate and connect to #-----
  layer_conv_2d(filters = 40, #-----First convoluted layer
kernel_size = c(4,4), #---40 Filters with dimension 4x4
activation = 'softsign', #--with a ReLu activation function
input_shape = c(100,100,3)) %>%
  #------(B)-----#
  layer_conv_2d(filters = 40, #-----Second convoluted layer
kernel_size = c(4,4), #---40 Filters with dimension 4x4
activation = 'softsign') %>% #--with a ReLu activation function
  #------(C)-----#
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.5) %>% #-----Drop out layer
  #------(D)-----#
  layer_conv_2d(filters = 80, #-----Third convoluted layer
kernel_size = c(4,4), #---80 Filters with dimension 4x4
activation = 'softsign') %>% #--with a ReLu activation function
  #------(E)-----#
  layer_conv_2d(filters = 80, #-----Fourth convoluted layer
kernel_size = c(4,4), #---80 Filters with dimension 4x4
activation = 'softsign') %>% #--with a ReLu activation function
  #------(F)-----#

  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
  #-----#
  layer_dropout(rate = 0.6) %>% #-----Drop out layer
  #------(G)-----#
  layer_flatten()%>% #---Flattening the final stack of feature maps
  #------(H)-----#
  layer_dense(units = 256, activation = 'softsign')%>% #----Hidden layer
  #------(I)-----#
  layer_dropout(rate= 0.5)%>% #-----Drop-out layer
  #-----#
  layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
  #------(J)-----#

  compile(loss = 'categorical_crossentropy',
optimizer = optimizer_adam(),
metrics = c("accuracy")) # Compiling the architecture

#####
##### MODEL FITTING CHART #####
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
      train_lab,
      epochs = 100,
      batch_size = 40,
      validation_split = 0.2

```

```

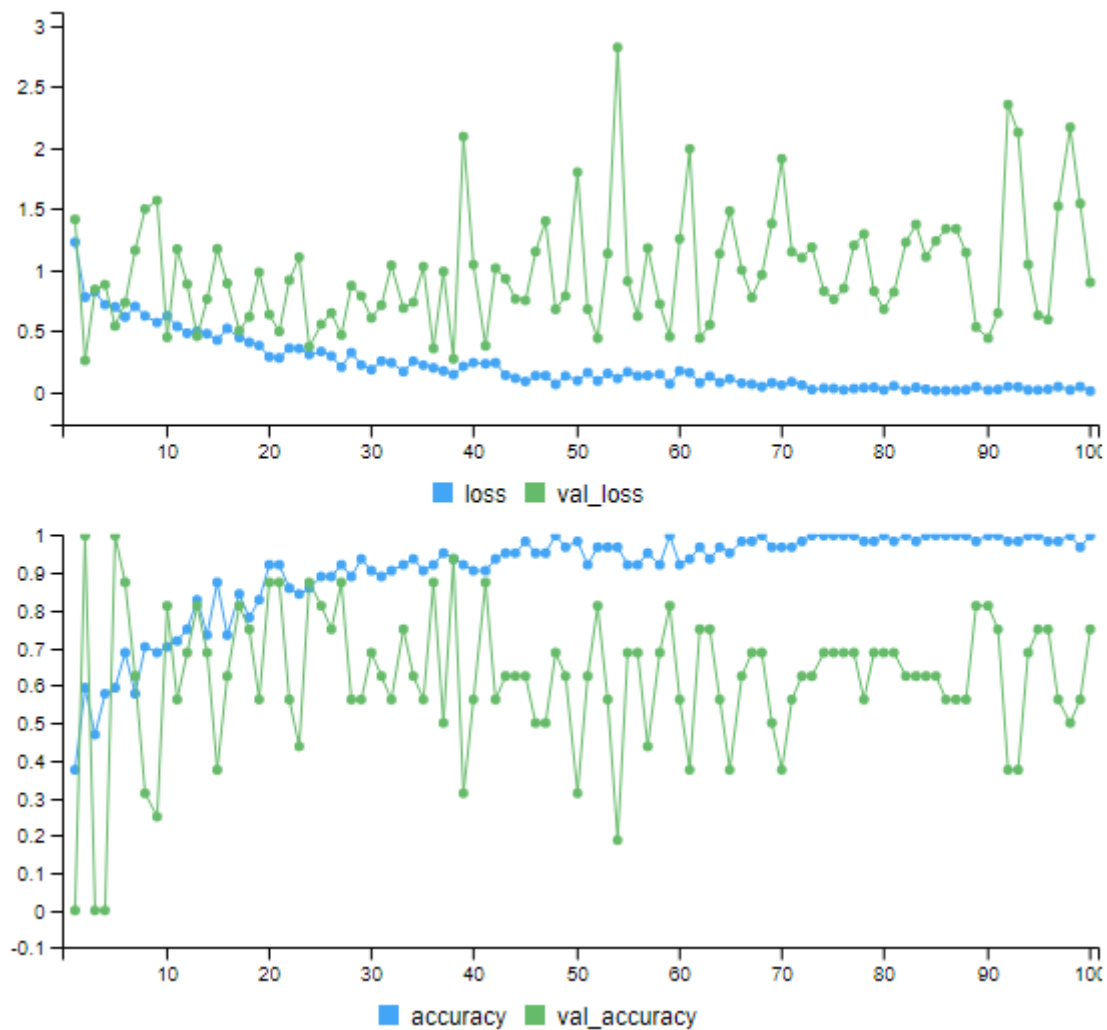
)

#Model Evaluation
model.fancydoggos %>% evaluate(train,train_lab) #Evaluati
pred<-model.fancydoggos %>% predict_classes(train) #-----Clas
Train_Result<-table(Predicted = pred, Actual = train_y) #----Resu

model.fancydoggos %>% evaluate(test, test_lab) #-----Eva
pred1<- model.fancydoggos %>% predict_classes(test) #-----Cla
Test_Result<-table(Predicted = pred1, Actual = test_y) #-----Res

rownames(Train_Result)<-rownames(Test_Result)<-colnames(Train_Result)<-colnames(Test_Resu
print(Train_Result)
print(Test_Result)

```



Figuur 27: r1t10

### 12.2.10 R1T11

```

#install_keras(
# method = c("auto", "virtualenv", "conda"),
# conda = "auto",

```



```

# version = "default",
# tensorflow = "default",
# extra_packages = c("tensorflow-hub")
#)

#to setup the Keras library and TensorFlow bac
#if (!requireNamespace('BiocManager', quietly = TRUE)) #INSTALL CPU VERSION OF KE

# -----

##if (!requireNamespace('BiocManager', quietly = TRUE))
## install.packages('BiocManager')
##BiocManager::install('EBImage')
#install.packages("jpeg")
##reticulate::py_install(envname = "r-tensorflow", packages = "numpy")

library(keras)
library(EBImage)
library(jpeg)
library(tensorflow)

#####
#DATASET EXPLORATION
#####

getwd()
setwd("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua") # FOLDER
##img<- readImage("C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua/n02085620_199.jp

# Reading
##print(img) # Print

##getFrames(img, type='total') # Split

##display(img)
# Display image
#####
#EN OF DATASET EXPLORATION
#####

#####
#####CREATE LIST OF RAT#####
#####

# To acce

currentdir<- "C:/Users/niek/Documents/R/TestCNN/n02085620-Chihuahua" # The pat
img.doggo<- sample(dir(currentdir)); #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{

```

```

    doggolist[[i]]<- readImage(img.doggo[i])
    doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
}                                                                 #resizing

chiuhuahalist<- doggolist                                                                 # Storing

#####
#####CREATE LIST OF SCOTTISH DEERHOUND#####
#####

# To access the images of Clubs suit.

currentdir<- "C:/Users/niek/Documents/testdataset/Images/n02092002-Scottish_deerhound"
setwd(currentdir)                                                                 # FOLDER MET DIE PICCAS
img.doggo<- sample(dir(currentdir));                                                                 #-----s
doggolist<-list(NULL);
for(i in 1:length(img.doggo))
{
    doggolist[[i]]<- readImage(img.doggo[i])
    doggolist[[i]]<- resize(doggolist[[i]], 100, 100)
}                                                                 #resizing

scottishdeerhound<- doggolist                                                                 # Storing

#####
#####CREATE TRAINING POOL#####
#####

train_pool<-c(chiuhuahalist[1:40], scottishdeerhound[1:40])

train<-aperm(combine(train_pool), c(4,1,2,3))
test_pool<-c(chiuhuahalist[41:43], scottishdeerhound[41:43])

test<-aperm(combine(test_pool), c(4,1,2,3))

#####
##### WHICH RAT PICTURES ARE IN TEST SET#####
#####
par(mfrow=c(3,4)) # To contain all images in single frame
for(i in 1:6){
    plot(test_pool[[i]])
}
par(mfrow=c(1,1)) # Reset the default

#####

```

```
##### ONE HOT ENCODING #####
#####
```

```
#one hot encoding
```

```
train_y<-c(rep(0,40),rep(1,40),rep(2,40),rep(3,40))
test_y<-c(rep(0,3),rep(1,3),rep(2,3),rep(3,3))
train_lab<- to_categorical(train_y) #Catagorical vector for training
#classes
test_lab<- to_categorical(test_y)#Catagorical vector for test classes
```

```
#####
##### BEGIN WITH THE ACTUAL MODEL #####
#####
```

```
# Model Building
```

```
model.fancydoggos<- keras_model_sequential() #Keras Model composed of a
#----linear stack of layers
```

```
model.fancydoggos %>% #-----Initiate and connect to #-----
```

```
  layer_conv_2d(filters = 40, #-----First convoluted layer
```

```
  kernel_size = c(4,4), #---40 Filters with dimension 4x4
```

```
  activation = 'selu', #--with a ReLu activation function
```

```
  input_shape = c(100,100,3)) %>%
```

```
  #------(B)-----#
```

```
  layer_conv_2d(filters = 40, #-----Second convoluted layer
```

```
  kernel_size = c(4,4), #---40 Filters with dimension 4x4
```

```
  activation = 'selu') %>% #--with a ReLu activation function
```

```
  #------(C)-----#
```

```
  layer_max_pooling_2d(pool_size = c(4,4) )%>% #-----Max Pooling
```

```
  #-----#
```

```
  layer_dropout(rate = 0.5) %>% #-----Drop out layer
```

```
  #------(D)-----#
```

```
  layer_conv_2d(filters = 80, #-----Third convoluted layer
```

```
  kernel_size = c(4,4), #---80 Filters with dimension 4x4
```

```
  activation = 'selu') %>% #--with a ReLu activation function
```

```
  #------(E)-----#
```

```
  layer_conv_2d(filters = 80, #-----Fourth convoluted layer
```

```
  kernel_size = c(4,4), #---80 Filters with dimension 4x4
```

```
  activation = 'selu') %>% #--with a ReLu activation function
```

```
  #------(F)-----#
```

```
  layer_max_pooling_2d(pool_size = c(4,4)) %>% #-----Max Pooling
```

```
  #-----#
```

```
  layer_dropout(rate = 0.6) %>% #-----Drop out layer
```

```
  #------(G)-----#
```

```
  layer_flatten()%>% #---Flattening the final stack of feature maps
```

```
  #------(H)-----#
```

```
  layer_dense(units = 256, activation = 'selu')%>% #----Hidden layer
```

```
  #------(I)-----#
```

```
  layer_dropout(rate= 0.5)%>% #-----Drop-out layer
```

```
  #-----#
```

```
  layer_dense(units = 4, activation = "softmax")%>% #----Final Layer
```

```
  #------(J)-----#
```

```

compile(loss = 'categorical_crossentropy',
optimizer = optimizer_adam(),
metrics = c("accuracy")) # Compiling the architecture

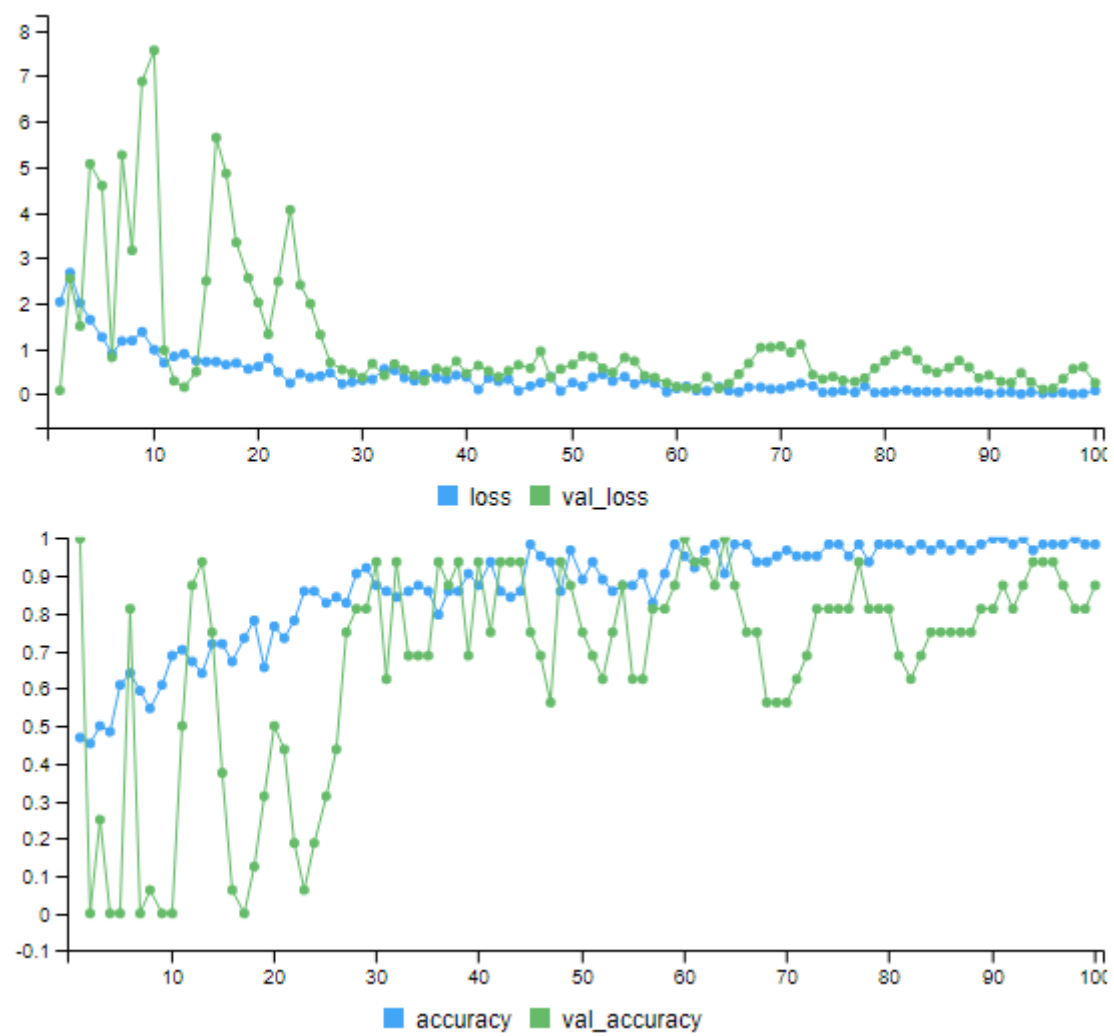
#####
##### MODEL FITTING CHART #####
#####
#fit model
history<- model.fancydoggos %>%
  fit(train,
      train_lab,
      epochs = 100,
      batch_size = 40,
      validation_split = 0.2
  )

#Model Evaluation
model.fancydoggos %>% evaluate(train,train_lab) #Evaluati
pred<-model.fancydoggos %>% predict_classes(train) #----Clas
Train_Result<-table(Predicted = pred, Actual = train_y) #----Resu

model.fancydoggos %>% evaluate(test, test_lab) #----Eva
pred1<- model.fancydoggos %>% predict_classes(test) #----Cla
Test_Result<-table(Predicted = pred1, Actual = test_y) #----Res

rownames(Train_Result)<-rownames(Test_Result)<-colnames(Train_Result)<-colnames(Test_Resu
print(Train_Result)
print(Test_Result)

```



Figuur 28: r1t11



## 12.3 Dependencies

### 12.3.1 Classificatie

package	Name	Version	Build
_ipyw_jlab_nb_ext_conf	0.1.0	py38_0	-
absl-py	0.11.0	pypi_0	pypi
alabaster	0.7.12	py_0	-
anaconda	2020.07	py38_0	-
anaconda-client	1.7.2	py38_0	-
anaconda-navigator	1.9.12	py38_0	-
anaconda-project	0.8.4	py_0	-
argh	0.26.2	py38_0	-
asn1crypto	1.3.0	py38_0	-
astroid	2.4.2	py38_0	-
astropy	4.0.1.post1	py38he774522_1	-
astunparse	1.6.3	pypi_0	pypi
atomicwrites	1.4.0	py_0	-
attrs	19.3.0	py_0	-
autopep8	1.5.3	py_0	-
babel	2.8.0	py_0	-
backcall	0.2.0	py_0	-
backports	1	py_2	-
backports.functools_lru_cache	1.6.1	pyhd3eb1b0_0	-
backports.shutil_get_terminal_size	1.0.0	py38_2	-
backports.tempfile	1	py_1	-
backports.weakref	1.0.post1	py_1	-
bcrypt	3.1.7	py38he774522_1	-
beautifulsoup4	4.9.1	py38_0	-
bitarray	1.4.0	py38he774522_0	-
bkcharts	0.2	py38_0	-
blas	1	mkl	-
bleach	3.1.5	py_0	-
blosc	1.19.0	h7bd577a_0	-
bokeh	2.1.1	py38_0	-
boto	2.49.0	py38_0	-
bottleneck	1.3.2	py38h2a96729_1	-
brotlipy	0.7.0	py38he774522_1000	-
bzip2	1.0.8	he774522_0	-
ca-certificates	2020.6.24	0	anaconda
cachetools	4.1.1	pypi_0	pypi
certifi	2020.6.20	py38_0	anaconda
cffi	1.14.0	py38h7a1dbc1_0	-
chardet	3.0.4	py38_1003	-
click	7.1.2	py_0	-
cloudpickle	1.5.0	py_0	-
clyent	1.2.2	py38_1	-
colorama	0.4.3	py_0	-
comtypes	1.1.7	py38_1001	-
conda	4.9.2	py38haa95532_0	-
conda-build	3.18.11	py38_1	-
conda-env	2.6.0	1	-
conda-package-handling	1.7.2	py38h76e460a_0	-
conda-verify	3.4.2	py_1	-
console_shortcut	0.1.1	4	-

package	Name	Version	Build
contextlib2	0.6.0.post1	py_0	-
cryptography	2.9.2	py38h7a1dbc1_0	-
curl	7.71.1	h2a8f88b_1	-
cycler	0.10.0	py38_0	-
cython	0.29.21	py38ha925a31_0	-
cytoolz	0.10.1	py38he774522_0	-
dask	2.20.0	py_0	-
dask-core	2.20.0	py_0	-
dataclasses	0.6	pypi_0	pypi
decorator	4.4.2	py_0	-
defusedxml	0.6.0	py_0	-
diff-match-patch	20200713	py_0	-
dill	0.3.3	pypi_0	pypi
distributed	2.20.0	py38_0	-
dm-tree	0.1.5	pypi_0	pypi
docutils	0.16	py38_1	-
entrypoints	0.3	py38_0	-
et_xmlfile	1.0.1	py_1001	-
fastcache	1.1.0	py38he774522_0	-
filelock	3.0.12	py_0	-
flake8	3.8.3	py_0	-
flask	1.1.2	py_0	-
freetype	2.10.2	hd328e21_0	-
fsspec	0.7.4	py_0	-
future	0.18.2	py38_1	-
gast	0.3.3	pypi_0	pypi
get_terminal_size	1.0.0	h38e98db_0	-
gevent	20.6.2	py38he774522_0	-
gin-config	0.4.0	pypi_0	pypi
git	2.23.0	h6bb4b03_0	-
glob2	0.7	py_0	-
gmpy2	2.0.8	py38h7edee0f_3	-
google-api-core	1.23.0	pypi_0	pypi
google-api-python-client	1.12.8	pypi_0	pypi
google-auth	1.23.0	pypi_0	pypi
google-auth-httpplib2	0.0.4	pypi_0	pypi
google-auth-oauthlib	0.4.2	pypi_0	pypi
google-cloud-bigquery	2.5.0	pypi_0	pypi
google-cloud-core	1.4.4	pypi_0	pypi
google-crc32c	1.0.0	pypi_0	pypi
google-pasta	0.2.0	pypi_0	pypi
google-resumable-media	1.1.0	pypi_0	pypi
googleapis-common-protos	1.52.0	pypi_0	pypi
greenlet	0.4.16	py38he774522_0	-
grpcio	1.33.2	pypi_0	pypi
h5py	2.10.0	py38h5e291fa_0	-
hdf5	1.10.4	h7ebc959_0	-
heapdict	1.0.1	py_0	-
html5lib	1.1	py_0	-



package	Name	Version	Build
httplib2	0.18.1	pypi_0	pypi
icc_rt	2019.0.0	h0cc432a_1	-
icu	58.2	ha925a31_3	-
idna	2.1	py_0	-
imageio	2.9.0	py_0	-
imagesize	1.2.0	py_0	-
imgaug	0.4.0	pypi_0	pypi
importlib-metadata	1.7.0	py38_0	-
importlib-resources	3.3.0	pypi_0	pypi
importlib_metadata	1.7.0	0	-
intel-openmp	2020.1	216	-
intervaltree	3.0.2	py_1	-
ipykernel	5.3.2	py38h5ca1d4c_0	-
ipyparallel	6.3.0	pypi_0	pypi
ipython	7.16.1	py38h5ca1d4c_0	-
ipython_genutils	0.2.0	py38_0	-
ipywidgets	7.5.1	py_0	-
isort	4.3.21	py38_0	-
itsdangerous	1.1.0	py_0	-
jdcal	1.4.1	py_0	-
jedi	0.17.1	py38_0	-
jinja2	2.11.2	py_0	-
joblib	0.16.0	py_0	-
jpeg	9b	hb83a4c4_2	-
json5	0.9.5	py_0	-
jsonschema	3.2.0	py38_0	-
jupyter	1.0.0	py38_7	-
jupyter_client	6.1.6	py_0	-
jupyter_console	6.1.0	py_0	-
jupyter_core	4.6.3	py38_0	-
jupyterlab	2.1.5	py_0	-
jupyterlab_server	1.2.0	py_0	-
kaggle	1.5.10	pypi_0	pypi
keras	2.4.3	pypi_0	pypi
keras-preprocessing	1.1.2	pypi_0	pypi
keyring	21.2.1	py38_0	-
kiwisolver	1.2.0	py38h74a9793_0	-
krb5	1.18.2	hc04afaa_0	-
lazy-object-proxy	1.4.3	py38he774522_0	-
libarchive	3.4.2	h5e25573_0	-
libcurl	7.71.1	h2a8f88b_1	-
libiconv	1.15	h1df5818_7	-
liblief	0.10.1	ha925a31_0	-
libllvm9	9.0.1	h21ff451_0	-
libpng	1.6.37	h2a8f88b_0	-
libsodium	1.0.18	h62dcd97_0	-

package	Name	Version	Build
libspatialindex	1.9.3	h33f27b4_0	-
libssh2	1.9.0	h7a1dbc1_1	-
libtiff	4.1.0	h56a325e_1	-
libxml2	2.9.10	h464c3ec_1	-
libxslt	1.1.34	he774522_0	-
llvmlite	0.33.0	py38ha925a31_0	-
loket	0.2.0	py38_1	-
lxml	4.5.2	py38h1350720_0	anaconda
lz4-c	1.9.2	h62dcd97_0	-
lzo	2.1	he774522_2	-
m2w64-gcc-libgfortran	5.3.0	6	-
m2w64-gcc-libc	5.3.0	7	-
m2w64-gcc-libc-core	5.3.0	7	-
m2w64-gmp	6.1.0	2	-
m2w64-libwinpthread-git	5.0.0.4634.697f757	2	-
markdown	3.3.3	pypi_0	pypi
markupsafe	1.1.1	py38he774522_0	-
matplotlib	3.2.2	0	-
matplotlib-base	3.2.2	py38h64f37c6_0	-
mccabe	0.6.1	py38_1	-
menuinst	1.4.16	py38he774522_1	-
mistune	0.8.4	py38he774522_1000	-
mkl	2020.1	216	-
mkl-service	2.3.0	py38hb782905_0	-
mkl_fft	1.1.0	py38h45dec08_0	-
mkl_random	1.1.1	py38h47e9c7a_0	-
mock	4.0.2	py_0	-
more-itertools	8.4.0	py_0	-
mpc	1.1.0	h7edee0f_1	-
mpfr	4.0.2	h62dcd97_1	-
mpir	3.0.0	hec2e145_1	-
mpmath	1.1.0	py38_0	-
msgpack-python	1.0.0	py38h74a9793_1	-
msys2-conda-epoch	20160418	1	-
multipledispatch	0.6.0	py38_0	-
navigator-updater	0.2.1	py38_0	-
nbconvert	5.6.1	py38_0	-
nbformat	5.0.7	py_0	-
networkx	2.4	py_1	-
nlTK	3.5	py_0	-
nose	1.3.7	py38_2	-
notebook	6.0.3	py38_0	-
numba	0.50.1	py38h47e9c7a_0	-
numexpr	2.7.1	py38h25d0782_0	-
numpy	1.18.5	py38h6530119_0	-
numpy-base	1.18.5	py38hc3f5095_0	-
numpydoc	1.1.0	py_0	-
oauthlib	3.1.0	pypi_0	pypi
object-detection	0.1	pypi_0	pypi
olefile	0.46	py_0	-

package	Name	Version	Build
opencv-python	4.4.0.46	pypi_0	pypi
opencv-python-headless	4.4.0.46	pypi_0	pypi
openpyxl	3.0.4	py_0	-
openssl	1.1.1g	he774522_0	anaconda
opt-einsum	3.3.0	pypi_0	pypi
packaging	20.4	py_0	-
pandas	1.0.5	py38h47e9c7a_0	-
pandoc	2.1	0	-
pandocfilters	1.4.2	py38_1	-
paramiko	2.7.1	py_0	-
parso	0.7.0	py_0	-
partd	1.1.0	py_0	-
path	13.1.0	py38_0	-
path.py	12.4.0	0	-
pathlib2	2.3.5	py38_0	-
pathtools	0.1.2	py_1	-
patsy	0.5.1	py38_0	-
pep8	1.7.1	py38_0	-
pexpect	4.8.0	py38_0	-
pickleshare	0.7.5	py38_1000	-
pillow	7.2.0	py38hcc1f983_0	-
pip	20.1.1	py38_1	-
pkginfo	1.5.0.1	py38_0	-
pluggy	0.13.1	py38_0	-
ply	3.11	py38_0	-
powershell_shortcut	0.0.1	3	-
prometheus_client	0.8.0	py_0	-
promise	2.3	pypi_0	pypi
prompt-toolkit	3.0.5	py_0	-
prompt_toolkit	3.0.5	0	-
proto-plus	1.11.0	pypi_0	pypi
protobuf	3.14.0	pypi_0	pypi
psutil	5.7.0	py38he774522_0	-

package	Name	Version	Build
py	1.9.0	py_0	-
py-cpuinfo	7.0.0	pypi_0	pypi
py-lief	0.10.1	py38ha925a31_0	-
pyasn1	0.4.8	pypi_0	pypi
pyasn1-modules	0.2.8	pypi_0	pypi
pycocotools-windows	2.0.0.2	pypi_0	pypi
pycodestyle	2.6.0	py_0	-
pycosat	0.6.3	py38he774522_0	-
pycparser	2.2	py_2	-
pycurl	7.43.0.5	py38h7a1dbc1_0	-
pydocstyle	5.0.2	py_0	-
pyflakes	2.2.0	py_0	-
pygments	2.6.1	py_0	-
pylint	2.5.3	py38_0	-
pynacl	1.4.0	py38h62dcd97_1	-
pyodbc	4.0.30	py38ha925a31_0	-
pyopenssl	19.1.0	py_1	-
pyparsing	2.4.7	py_0	-
pyqt	5.9.2	py38ha925a31_4	-
pyreadline	2.1	py38_1	-
pyrsistent	0.16.0	py38he774522_0	-
pysocks	1.7.1	py38_0	-
pytables	3.6.1	py38ha5be198_0	-
pytest	5.4.3	py38_0	-
python	3.8.3	he1778fa_2	-
python-dateutil	2.8.1	py_0	-
python-jsonrpc-server	0.3.4	py_1	-
python-language-server	0.34.1	py38_0	-
python-libarchive-c	2.9	py_0	-
python-slugify	4.0.1	pypi_0	pypi
pytz	2020.1	py_0	-
pywavelets	1.1.1	py38he774522_0	-
pywin32	227	py38he774522_1	-
pywin32-ctypes	0.2.0	py38_1000	-
pywinpty	0.5.7	py38_0	-
pyyaml	5.3.1	py38he774522_1	-
pyzmq	19.0.1	py38ha925a31_1	-
qdarkstyle	2.8.1	py_0	-
qt	5.9.7	vc14h73c81de_0	-
qtawesome	0.7.2	py_0	-
qtconsole	4.7.5	py_0	-
qtpy	1.9.0	py_0	-
regex	2020.6.8	py38he774522_0	-

package	Name	Version	Build
requests	2.24.0	py_0	-
requests-oauthlib	1.3.0	pypi_0	pypi
rope	0.17.0	py_0	-
rsa	4.6	pypi_0	pypi
rtree	0.9.4	py38h21ff451_1	-
ruamel_yaml	0.15.87	py38he774522_1	-
scikit-image	0.16.2	py38h47e9c7a_0	-
scikit-learn	0.23.1	py38h25d0782_0	-
scipy	1.5.0	py38h9439919_0	-
seaborn	0.10.1	py_0	-
send2trash	1.5.0	py38_0	-
sentencepiece	0.1.94	pypi_0	pypi
setuptools	49.2.0	py38_0	-
shapely	1.7.1	pypi_0	pypi
simplegeneric	0.8.1	py38_2	-
singledispatch	3.4.0.3	py38_0	-
sip	4.19.13	py38ha925a31_0	-
six	1.15.0	py_0	-
snappy	1.1.8	h33f27b4_0	-
snowballstemmer	2.0.0	py_0	-
sortedcollections	1.2.1	py_0	-
sortedcontainers	2.2.2	py_0	-
soupsieve	2.0.1	py_0	-
sphinx	3.1.2	py_0	-
sphinxcontrib	1	py38_1	-
sphinxcontrib-applehelp	1.0.2	py_0	-
sphinxcontrib-devhelp	1.0.2	py_0	-
sphinxcontrib-htmlhelp	1.0.3	py_0	-
sphinxcontrib-jsmath	1.0.1	py_0	-
sphinxcontrib-qthelp	1.0.3	py_0	-
sphinxcontrib-serializinghtml	1.1.4	py_0	-
sphinxcontrib-websupport	1.2.3	py_0	-
spyder	4.1.4	py38_0	-
spyder-kernels	1.9.2	py38_0	-
sqlalchemy	1.3.18	py38he774522_0	-
sqlite	3.32.3	h2a8f88b_0	-
statsmodels	0.11.1	py38he774522_0	-
sympy	1.6.1	py38_0	-
tbb	2020	h74a9793_0	-
tblib	1.6.0	py_0	-
tensorboard	2.4.0	pypi_0	pyp

package	Name	Version	Build
tensorboard-plugin-wit	1.7.0	pypi_0	pyp
tensorflow	2.3.1	pypi_0	pyp
tensorflow-addons	0.11.2	pypi_0	pyp
tensorflow-datasets	4.1.0	pypi_0	pyp
tensorflow-estimator	2.3.0	pypi_0	pyp
tensorflow-gpu	2.3.1	pypi_0	pyp
tensorflow-gpu-estimator	2.3.0	pypi_0	pyp
tensorflow-hub	0.10.0	pypi_0	pyp
tensorflow-metadata	0.25.0	pypi_0	pyp
tensorflow-model-optimization	0.5.0	pypi_0	-
termcolor	1.1.0	pypi_0	pyp
terminado	0.8.3	py38_0	
	-		
testpath	0.4.4	py_0	
text-unidecode	1.3	pypi_0	-
tf-models-official	2.3.0	pypi_0	pyp
tf-slim	1.1.0	pypi_0	pyp
threadpoolctl	2.1.0	pyh5ca1d4c_0	
	-		
tk	8.6.10	he774522_0	
	-		
toml	0.10.1	py_0	
	-		
toolz	0.10.0	py_0	
	-		
tornado	6.0.4	py38he774522_1	
	-		
tqdm	4.47.0	py_0	
	-		
traitlets	4.3.3	py38_0	
	-		
typeguard	2.10.0	pypi_0	pyp
typing_extensions	3.7.4.2	py_0	
	-		
ujson	1.35	py38he774522_0	
	-		
unicodedev	0.14.1	py38_0	
	-		
uritemplate	3.0.1	pypi_0	pyp
urllib3	1.25.9	py_0	
	-		
vboxapi	1	pypi_0	pyp
vc	14.1	h0510ff6_4	-
vs2015_runtime	14.16.27012	hf0eaf9b_3	-

package	Name	Version	Build
watchdog	0.10.3	py38_0	-
wcwidth	0.2.5	py_0	-
webencodings	0.5.1	py38_1	-
werkzeug	1.0.1	py_0	-
wheel	0.34.2	py38_0	-
widgetsnbextension	3.5.1	py38_0	-
win_inet_pton	1.1.0	py38_0	-
win_unicode_console	0.5	py38_0	-
wincertstore	0.2	py38_0	-
winpty	0.4.3	4	-
wrapt	1.11.2	py38he774522_0	-
xlrd	1.2.0	py_0	-
xlsxwriter	1.2.9	py_0	-
xlwings	0.19.5	py38_0	-
xlwt	1.3.0	py38_0	-
xmldict	0.12.0	py_0	-
xz	5.2.5	h62dcd97_0	-
yaml	0.2.5	he774522_0	-
yapf	0.30.0	py_0	-
zeromq	4.3.2	ha925a31_2	-
zict	2.0.0	py_0	-
zipp	3.1.0	py_0	-
zlib	1.2.11	h62dcd97_4	-
zope	1	py38_1	-
zope.event	4.4	py38_0	-
zope.interface	4.7.1	py38he774522_0	-
zstd	1.4.5	ha9fde0e_0	-





### 12.3.2 Object detectie

#	Name	Version	Build
_tfflow_select	2.1.0	gpu	
absl-py	0.11.0	py37haa95532_0	
astor	0.8.1	py37_0	
blas	1	mkl	
ca-certificates	2020.10.14	0	anaconda
certifi	2020.6.20	py37_0	anaconda
chardet	3.0.4	pypi_0	pypi
cuda-toolkit	9	1	
cudnn	7.6.5	cuda9.0_0	
cupy	6.0.0	py37hb633afd_0	anaconda
cycler	0.10.0	pypi_0	pypi
cython	3.0a6	pypi_0	pypi
fastlock	0.5	pypi_0	pypi
filelock	3.0.12	pypi_0	pypi
gast	0.4.0	py_0	
grpcio	1.31.0	py37he7da953_0	
h5py	2.10.0	py37h5e291fa_0	
hdf5	1.10.4	h7ebc959_0	
icc_rt	2019.0.0	h0cc432a_1	
idna	2.1	pypi_0	pypi
importlib-metadata	2.0.0	py_1	
intel-openmp	2020.2	254	
keras-applications	1.0.8	py_1	
keras-preprocessing	1.1.0	py_1	
kiwisolver	1.3.1	pypi_0	pypi
libprotobuf	3.13.0.1	h200bbdf_0	
lvis	0.5.3	pypi_0	pypi
markdown	3.3.3	py37haa95532_0	
matplotlib	3.3.3	pypi_0	pypi
mkl	2020.2	256	
mkl-service	2.3.0	py37h2bbff1b_0	
mkl_fft	1.2.0	py37h45dec08_0	
mkl_random	1.1.1	py37h47e9c7a_0	
nets	0.0.3	pypi_0	pypi
nose	1.3.7	pypi_0	pypi
numpy	1.19.2	py37hadc3359_0	
numpy-base	1.19.2	py37ha3acd2a_0	
object-detection	0.1	pypi_0	pypi
opencv-python	4.4.0.46	pypi_0	pypi
openssl	1.1.1h	he774522_0	anaconda
pillow	8.0.1	pypi_0	pypi
pip	20.3	py37haa95532_0	
protobuf	3.13.0.1	py37ha925a31_1	
pycocotools-windows	2.0.0.2	pypi_0	pypi
pyparsing	3.0.0b1	pypi_0	pypi
pyreadline	2.1	py37_1	
python	3.7.9	h60c2a47_0	
python-dateutil	2.8.1	pypi_0	pypi
requests	2.25.0	pypi_0	pypi
scipy	1.5.2	py37h9439919_0	
setuptools	50.3.1	py37haa95532_1	
six	1.15.0	py37haa95532_0	
sqlite	3.33.0	h2a8f88b_0	
tensorboard	1.14.0	py37he3c9ec2_0	
tensorflow	1.14.0	gpu py37h2fabf85_0	

package	Name	Version	Build
tensorflow-gpu	1.14.0	h0d30ee6_0	
termcolor	1.1.0	py37_1	
tf-slim	1.1.0	pypi_0	pypi
urllib3	1.26.2	pypi_0	pypi
vc	14.1	h0510ff6_4	
vs2015_runtime	14.16.27012	hf0eaf9b_3	
werkzeug	1.0.1	py_0	
wheel	0.35.1	pyhd3eb1b0_0	
wincertstore	0.2	py37_0	
wrapt	1.12.1	py37he774522_1	
zipp	3.4.0	pyhd3eb1b0_0	
zlib	1.2.11	h62dcd97_4	