Onderzoek: Detectie van ASL-alfabet uit afbeelding door middel van Tensorflow en Keras
Beeldverwerking en computer vision
Emma Raijmakers
09-04-2023

Detectie van ASL-alfabet uit afbeelding door middel van Tensorflow en Keras

Introductie

American sign language (ASL) is een taal die voornamelijk gesproken wordt door doven in Noord-Amerika. De taal wordt op een visuele manier gecommuniceerd door handgebaren en andere visuele uitdrukkingen[1].

In dit onderzoek wordt er gefocust op het ASL-alfabet, dat bestaat uit 26 handgebaren. Ieder handgebaar representeert een letter. Er wordt gekeken of het mogelijk is om ASL-letters te detecteren uit een afbeelding en hoe dit op een efficiënte en correcte manier kan.

Background

Voor dit onderzoek is opgezocht hoe het algoritme YOLO (You Only Look Once) werkt. YOLO werkt niet met classifiers, maar splitst een image in vakjes op en geeft een class probability en mogelijke bounding boxes op ieder vakje. Op basis van hoe confident de class probability is, wordt het object gedetecteerd en de bounding box getekend. YOLO kan gebruikt worden in dit onderzoek door bounding boxes te tekenen om de handen in de images, om zo alleen de informatie van de hand te krijgen. Hierdoor wordt trainen makkelijker en heeft de achtergrond minder invloed. Door tijdgebrek is YOLO niet toegevoegd.

Om dit onderzoek te begrijpen is voorkennis van Python programmeren en werken met de Python library Tensorflow vereist. Deze onderdelen zijn geen deel van het onderzoek en worden dus niet verder uitgelegd.

Methode

Ten eerste, is er een methode nodig om afbeeldingen te verzamelen en om te zetten naar de juiste letter. De afbeeldingen komen van een ASL-dataset van Kaggle[2]. De dataset bestaat uit 87.000 afbeeldingen van het formaat 200x200 en ze zijn opgedeeld in 29 classes. 26 ASL-letters (A-Z) en classes voor SPACE, DELETE and NOTHING. De testdata van de ASL-dataset is erg klein (1 afbeelding per letter). Om de testdataset te vergroten wordt de traindataset gerandomized en gesplitst in 10.000 testdata en de rest (87.000 – 10.000 = 77.000) traindata.

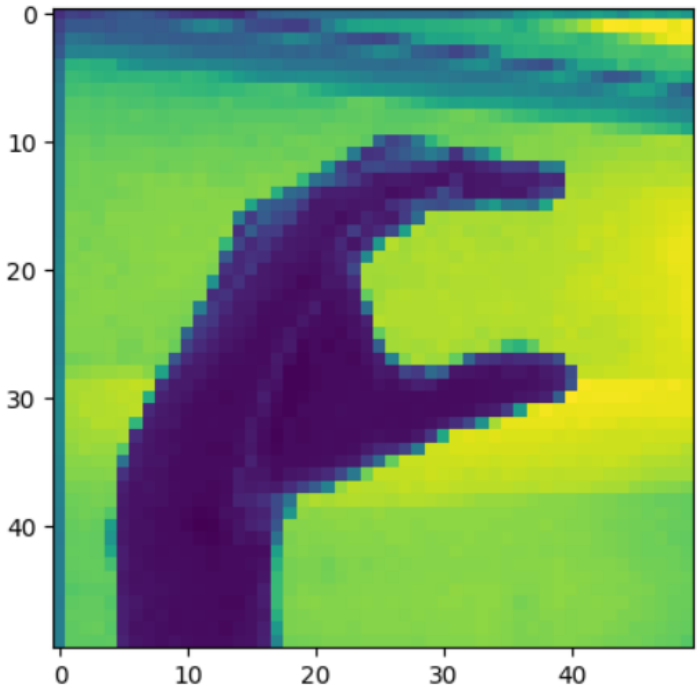
Voor de letterdetectie is er een Convolutional Neural Network (CNN) gemaakt met Tensorflow. Er wordt een CNN gebruikt, want bij een CNN maakt het niet uit waar op de afbeelding het subject staat en welke rotatie het heeft[3]. Tensorflow en Keras zijn goede manieren om objecten te

---

[1] Wikipedia contributors. (2023, april 1). American Sign Language. *Wikipedia*. Geraadpleegd op 9 april 2023, van https://en.wikipedia.org/wiki/American_Sign_Language
[2] AKASH. (2018, 22 april). ASL Alphabet. *Kaggle*. Geraadpleegd op 9 april 2023, van https://www.kaggle.com/datasets/grassknoted/asl-alphabet
[3] Yamamoto-Roijers, D. Video's module 6. *Canvas.* Geraadpleegd op 9 april 2023, van https://canvas.hu.nl/courses/32732/pages/videos-module-6?module_item_id=851333

classificeren, want de library[4] heeft goede documentatie en er is veel te customizen. Hierdoor is het makkelijk aan te passen naar ieder probleem.

Voordat de afbeeldingen aan het CNN gegeven worden, worden ze eerst voorbewerkt. De voorbewerkingen die gedaan worden zijn het zwart-wit maken van de image en het kompressen van de image naar het formaat 50x50. Deze voorbewerkingen worden gedaan om het trainen en testen sneller te laten verlopen. Een zwart-wit image is een 2D image en een gekleurde image een 3D image, want zwart-wit heeft 1 kleur value per pixel en kleur heeft er 3 (RGB). Deze voorbewerkingen zouden geen invloed moeten hebben op de accuracy van het CNN, omdat de ASL-letters nog duidelijk te herkennen zijn op de afbeeldingen (zie de afbeelding hieronder). De voorbewerkte afbeeldingen worden gebruikt als input voor het CNN bij trainen en testen.



Figuur 1: De letter C, zwart-wit en 50x50

Verder wordt er gekeken naar de invloed van verschillende voorbewerkingen[5]. De voorbewerkingen die gebruikt worden voor het onderzoek zijn edge detection en een Gaussian filter. Voor meer informatie zie het kopje Preprocessing.

Om te testen of het CNN de juiste output geeft, wordt er een applicatie gemaakt om een afbeelding van een webcam te halen en die afbeelding wordt gebruikt als input voor het CNN om de juiste ASL-letter te predicten. De library die gebruikt wordt om een afbeelding van de webcam te krijgen is de python versie van OpenCV[6], omdat het een grote library is die veel kan. Dit wordt verder uitgelegd in het kopje Applicatie

---

[4] Keras. Keras documentation: Keras API reference. *Keras.* Geraadpleegd op 9 april 2023, van https://keras.io/api/

[5] T. B. (2022, 17 februari). Massive Tutorial on Image Processing And Preparation For Deep Learning in Python, #1. *Towards Data Science.* Geraadpleegd op 9 april 2023, van https://towardsdatascience.com/massive-tutorial-on-image-processing-and-preparation-for-deep-learning-in-python-1-e534ee42f122

[6] OpenCV. OpenCV-Python Tutorials. *OpenCV.* Geraadpleegd op 9 april 2023, van https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

Experimenten

In dit onderzoek wordt er gekeken of het mogelijk is om ASL-letters te detecteren uit een afbeelding en hoe dit op een efficiënte en correcte manier kan. Voor de efficiëntie wordt er gekeken naar de runtime. Hoelang duurt het om de afbeeldingen te preprocessen en het model te trainen en testen? Voor de correctheid wordt er gekeken naar de accuracy van het model en of de afbeelding genomen vanaf de webcam in de applicatie de juiste letter teruggeeft. Ook wordt er gekeken of het toevoegen van edge detectie of een Gaussian filter voor een hogere accuracy zorgt.

Preprocessing

Voordat de ASL-images als input voor het CNN gebruikt worden, worden ze eerst gepreprocessed. Er zijn verschillende preprocessen die toegevoegd kunnen worden. Dit zijn onder andere:

- De images zwart-wit maken. Deze voorbewerking wordt toegevoegd, want het zorgt ervoor dat een image van een 3D naar een 2D grootte gaat. De RGB-values worden omgezet naar 1 zwart-wit value. Dit maakt het trainen veel sneller. Aangezien het model de juiste uitkomst moet geven voor alle soorten huidskleuren, is de kleur dus ook niet belangrijk.
- Rescalen. Deze voorbewerking wordt ook toegevoegd, omdat hierdoor het trainen wordt versneld. De images worden van 200x200 pixels verkleind naar 50x50. Hierdoor is de image kleiner, maar is de letter wel nog zichtbaar.
- Flippen. Flippen wordt niet toegevoegd, want dit kan problemen geven met het input geven vanaf een camera. Dan moet verzekerd zijn dat de input vanaf de camera ook geflipt wordt. Een letter captured van de camera die niet geflipt is, zal niet herkend worden als het model getraind is op een geflipte image.
- Edge detection. Bij edge detection worden de edges van een image aangegeven. De invloed van edge detection op het model wordt in het experiment onderzocht.
- Smoothing met een Gaussian filter. Een Gaussian filter geeft een blur aan een image. De invloed van een Gaussian filter op het model wordt ook onderzocht.

De dataset is vrij groot (87.000, 3.000 images per letter) dus het uitbreiden van de dataset met random translation, rotation, contrast en brightness lijkt mij niet nodig.

Model

Om de efficiëntie en correctheid van het model te testen, wordt er gekeken naar de runtime en de accuracy berekend door het model te testen. Welke layers er worden gebruikt en de parameters van de layers hebben ook invloed en hierop wordt ook getest. Het model bestaat uit een Convolutional, eventueel een Relu en een Pooling layer. Deze layers kunnen een paar keer herhaald worden. Aan het einde van het model is er een Flatten en Dense layer[7]. Wat goede layers zijn voor een CNN is nog in onderzoek, dus is het nog niet bekend. In mijn onderzoek ga ik verschillende layers en layer variables proberen om voor mijn situatie een werkend model te krijgen.

Het model wordt geprogrammeerd in een Jupyter notebookbestand zodat het inladen van de dataset, het preprocessen en het training apart en niet steeds opnieuw gerund hoeven te worden. De images worden ingeladen, de preprocessing wordt toegepast en de images worden toegevoegd aan een list. Deze list wordt gerandomized en de eerste 10.000 images worden gebruikt voor de test images en labels. De rest (87.000 – 10.000 = 77.000) wordt gebruikt voor de train images en labels.

[7] Yamamoto-Roijers, D. Opdrachten Neurale Netwerken. *Canvas.* Geraadpleegd op 9 april 2023, van https://canvas.hu.nl/courses/32732/assignments/234890?module_item_id=851331

Voor deze code is een tutorial[8] als hulpmiddel gebruikt. De testdataset die de Kaggle dataset levert, bestaat uit 1 afbeelding per letter. Dit is niet geschikt voor testing, daarom wordt er een eigen testdataset gemaakt. De Kaggle testdataset is wel goed voor predicten. Het model wordt gefit met de traindata en vervolgens ge-evaluate met de testdata. Als laatste worden nieuwe images gebruikt om te predicten. Voor meer uitleg, zie de code[9].

Invloed van layers en variables aanpassen in model

De eerste test bestaat uit de volgende code (in de fit functie moet er in plaats van validation data een validation split zijn, dus val_accuracy en val_loss zijn hetzelfde als de evaluate loss en accuracy, maar later in het onderzoek wordt dit opgelost):

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 10 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense"),
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10e epoch resultaten zijn weergegeven (zie de bijlage 1 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	29	0.2760	0.9252	0.3366	0.8981
Test 2	31	0.2475	0.9366	0.2998	0.9091
Test 3	28	0.3532	0.9016	0.4113	0.8776
Test 4	28	0.2798	0.9269	0.3278	0.9030
Test 5	25	0.3105	0.9134	0.3632	0.8915
Average (rounded)	28.5	0.2934	0.9207	0.3477	0.8959

Tabel 1: Resultaten van 1e model, fit functie

[8] Sentdex. (2018, 19 augustus). Convolutional Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.3. *YouTube.* Geraadpleegd op 9 april 2023, van https://www.youtube.com/watch?v=WvoLTXIjBYU

[9] Raijmakers, E. (2023, 9 april). VISN. *Github.* Geraadpleegd op 9 april 2023, van https://github.com/EmmaRaijmakers/VISN/tree/main/Eindopdracht

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.3366	0.8981
Test 2	2	0.2998	0.9091
Test 3	1	0.4113	0.8776
Test 4	2	0.3278	0.9030
Test 5	2	0.3632	0.8915
Average (rounded)	1.8	0.3477	0.8959

Tabel 2: Resultaten van 1e model, evaluate functie

Voor de volgende test is er een Relu layer toegevoegd tussen de convolutional en pooling layers:

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 10 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    Activation("relu"),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense"),
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10^{e} epoch resultaten zijn weergegeven (zie de bijlage 2 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	34	0.5228	0.8350	0.6135	0.8039
Test 2	31	0.3105	0.9020	0.3521	0.8913
Test 3	28	0.3064	0.9106	0.3444	0.8984
Test 4	34	0.4196	0.8713	0.4179	0.8708
Test 5	29	0.3506	0.8879	0.3759	0.8788
Average (rounded)	31.2	0.3820	0.8814	0.4208	0.8686

Tabel 3: Resultaten toevoeging Relu, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	3	0.6135	0.8039
Test 2	2	0.3521	0.8913

Test 3	2	0.3444	0.8984
Test 4	2	0.4179	0.8708
Test 5	2	0.3759	0.8788
Average (rounded)	2.2	0.4208	0.8686

Tabel 4: Resultaten toevoeging Relu, evaluate functie

Het toevoegen van een Relu-layer zorgt voor een hogere loss en lagere accuracy, maar het verschilt niet veel met het model zonder de Relu-layer. Voor de volgende onderzoeken wordt het weggehaald.

Bij dit code block wordt er een extra convolutional en pooling layer toegevoegd.

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 10 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Conv2D(num_filters, filter_size),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense"),
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10^{e} epoch resultaten zijn weergegeven (zie de bijlage 3 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	42	0.7242	0.7732	0.7574	0.7693
Test 2	41	0.6509	0.7980	0.7092	0.7807
Test 3	41	0.6635	0.7871	0.6833	0.7833
Test 4	40	0.7527	0.7636	0.7577	0.7572
Test 5	41	0.6626	0.7941	0.6826	0.7883
Average (rounded)	41	0.6908	0.7832	0.7181	0.7758

Tabel 5: Resultaten 2x convolutional en pooling, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	3	0.7574	0.7693

Test 2	3	0.7092	0.7807
Test 3	2	0.6833	0.7833
Test 4	2	0.7577	0.7572
Test 5	3	0.6826	0.7883
Average (rounded)	2.6	0.7181	0.7758

Tabel 6: Resultaten 2x convolutional en pooling, evaluate functie

2x de convolutional en pooling layer geeft een veel hogere loss en een lagere accuracy in vergelijking met het eerste model, dus dit model is niet beter. Ook duurt het langer om het model te trainen.

In het volgende model wordt een softmax activation gebruikt bij de Dense layer in plaats van een sigmoid activation:

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 10 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Conv2D(num_filters, filter_size),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="softmax", name="dense"),
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10^{e} epoch resultaten zijn weergegeven (zie de bijlage 4 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	38	0.7011	0.7800	0.7271	0.7783
Test 2	38	0.6979	0.7863	0.7218	0.7846
Average (rounded)	38	0.6995	0.7832	0.7245	0.7815

Tabel 7: Resultaten softmax ipv sigmoid, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.7271	0.7783
Test 2	3	0.7218	0.7846
Average (rounded)	2.5	0.7245	0.7815

Tabel 8: Resultaten softmax ipv sigmoid, evaluate functie

Het vervangen van sigmoid naar softmax in de Dense layer, heeft geen grote invloed en de waardes blijven ongeveer gelijk. Bij de softmax is de loss iets hoger, maar ook de accuracy is iets hoger.

Bij dit model zijn er meer filters gebruikt bij de convolutional layer:

```
num_filters = 6
filter_size = 3
pool_size = 2
num_epochs = 10 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10e epoch resultaten zijn weergegeven (zie de bijlage 5 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	29	0.1462	0.9625	0.2192	0.9349
Test 2	30	0.1436	0.9632	0.1912	0.9453
Test 3	30	0.1296	0.9686	0.1863	0.9436
Average (rounded)	29.7	0.1398	0.9648	0.1989	0.9413

Tabel 9: Resultaten meer filters, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.2192	0.9349
Test 2	2	0.1912	0.9453
Test 3	2	0.1863	0.9436
Average (rounded)	2	0.1989	0.9413

Tabel 10: Resultaten meer filters, evaluate functie

Het toevoegen van meer filters kost meer tijd om te verwerken, maar het geeft een lagere loss en een hogere accuracy in vergelijking met het eerste model.

Bij dit model is de filter size vergroot:

```
num_filters = 3
filter_size = 6
pool_size = 2
num_epochs = 10 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10e epoch resultaten zijn weergegeven (zie de bijlage 6 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	43	0.4653	0.8606	0.4920	0.8491
Test 2	44	0.3583	0.8961	0.4246	0.8713
Test 3	43	0.4328	0.8699	0.4868	0.8527
Average (rounded)	43.3	0.4188	0.8755	0.4678	0.8577

Tabel 11: Resultaten grotere filter size, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.4920	0.8491
Test 2	2	0.4246	0.8713
Test 3	2	0.4868	0.8527
Average (rounded)	2	0.4678	0.8577

Tabel 12: Resultaten grotere filter size, evaluate functie

Het vergroten van de filter size geeft een hogere loss en lagere accuracy in vergelijking met het eerste model.

Bij het volgende model wordt de pool size vergroot:

```
num_filters = 3
filter_size = 3
pool_size = 4
num_epochs = 10 #TODO change these vars
```

```
model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10e epoch resultaten zijn weergegeven (zie de bijlage 7 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	28	0.7480	0.7775	0.8317	0.7569
Test 2	27	0.9550	0.7167	0.9783	0.7131
Test 3	33	0.8308	0.7552	0.8532	0.7492
Average (rounded)	29.3	0.8446	0.7498	0.8877	0.7397

Tabel 13: Resultaten grotere pool size, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.8317	0.7569
Test 2	2	0.9783	0.7131
Test 3	2	0.8532	0.7492
Average (rounded)	2	0.8877	0.7397

Tabel 14: Resultaten grotere pool size, evaluate functie

Het vergroten van de pool size geeft een veel grotere loss en een lagere accuracy dan bij het eerste model. De tijd voor 1 epoch is hoger dan bij het eerste model.

Bij dit model is het nummer van epochs verhoogd:

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 20 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])
```

```
model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 20 epochs, de 20e epoch resultaten zijn weergegeven (zie de bijlage 8 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	29	0.1696	0.9514	0.2622	0.9177
Test 2	34	0.1627	0.9542	0.2413	0.9303
Test 3	31	0.1364	0.9635	0.2272	0.9345
Average (rounded)	31.3	0.1562	0.9564	0.2436	0.9275

Tabel 15: Resultaten meer epochs, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.2622	0.9177
Test 2	2	0.2413	0.9303
Test 3	2	0.2272	0.9345
Average (rounded)	2	0.2436	0.9275

Tabel 16: Resultaten meer epochs, evaluate functie

Door meer epochs te runnen, wordt de loss lager en de accuracy hoger in vergelijking met het eerste model. Echter meer epochs runnen kost ook meer tijd.

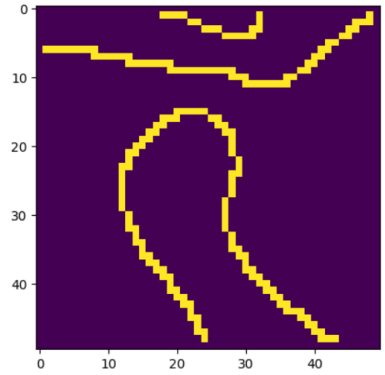
Inloed van toevoegen van preprocessing op model

In het kopje preprocessing zijn twee vormen van preprocessing genoemd: edge detection en een Gaussian filter. In dit deel wordt de invloed van deze preprocessen op het model onderzocht. De preprocessing wordt toegevoegd na het inladen van de images. Edge detection returnt true en false values in de image. True voor een edge en false voor geen edge. Hierdoor is er andere normalisatie nodig. Zie de code[10] voor meer informatie over die normalisatie. De Gaussian filter return values in dezelfde range als de normale images, dus daarvoor is geen andere normalisatie nodig.

[10] Raijmakers, E. (2023, 9 april). VISN. *Github.* Geraadpleegd op 9 april 2023, van https://github.com/EmmaRaijmakers/VISN/tree/main/Eindopdracht

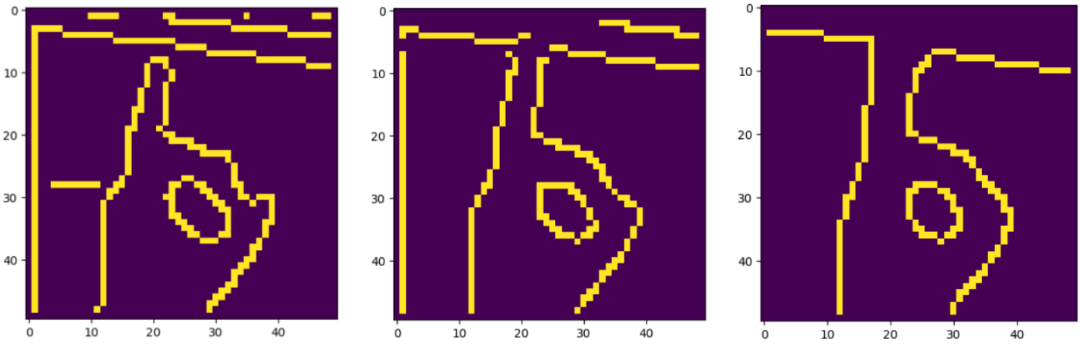
Edge detection

Een image na edge detection ziet er als volgt uit:



Figuur 2: De letter A met edge detection

De sigma die gebruikt is bij de Canny edge detection filter is 2. Deze waarde is gevonden door verschillende sigma's te proberen en een waarde te kiezen waarbij de randen van de hand duidelijk zichtbaar zijn en er zo min mogelijk lijnen in de achtergrond zichtbaar zijn.



Figuur 3: sigma 1, 2 en 3

De volgende code is gebruikt om de edge detection filter aan de preprocessing toe te voegen:

```
#edge detection
canny_filter = feature.canny(image_gray, sigma=2) #TODO change sigma
```

Het voordeel van een edge detection filter is dat achtergronddata weggewerkt wordt, waardoor het trainen gefocust wordt op de hand, maar het toevoegen van de filter aan iedere image kost tijd. Het preprocessen duurt hierdoor langer.

Deze code geeft de volgende resultaten na 10 epochs, de 10e epoch resultaten zijn weergegeven (zie de bijlage 9 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	29	0.2641	0.9113	0.4480	0.8481
Test 2	36	0.3051	0.8975	0.4490	0.8501
Test 3	28	0.2909	0.9015	0.4388	0.8602
Average (rounded)	31	0.2867	0.9034	0.4453	0.8528

Tabel 17: Resultaten edge detection, fit functie

De resultaten van de evaluation van dit model:

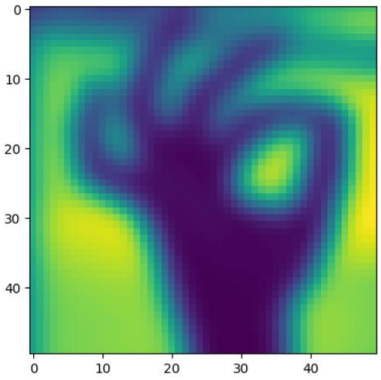
	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.4480	0.8481
Test 2	3	0.4490	0.8501
Test 3	2	0.4388	0.8602
Average (rounded)	2.3	0.4453	0.8528

Tabel 18: Resultaten edge detection, evaluate functie

De loss en accuracy van het model waarbij edge detection in de preprocessing is toegevoegd, is ongeveer gelijk aan de loss en accuracy van het eerste model. Echter, de validation loss is hoger en de validation accuracy is lager. Het toevoegen van een edge detection filter zorgt dus niet voor een beter model.

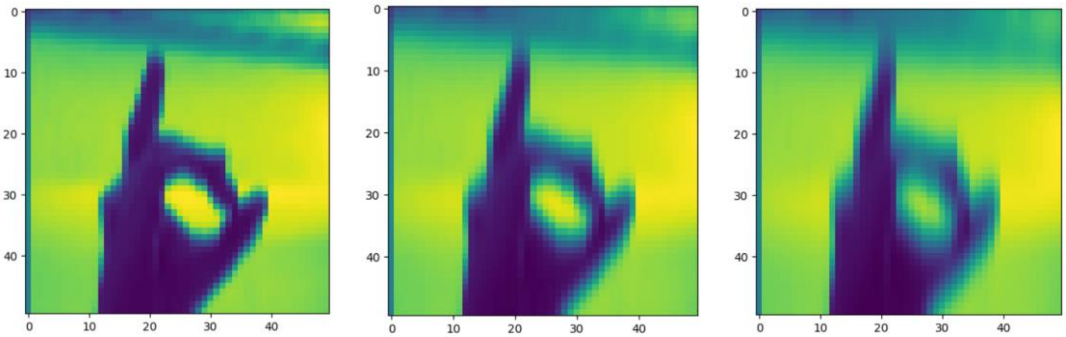
Gaussian filter

Een image na het toevoegen van een Gaussian filter ziet er als volgt uit:



Figuur 4: De letter F met Gaussian filter

De sigma die gebruikt is bij de Gaussian filter is 2. Deze waarde is gevonden door verschillende sigma's te proberen en een waarde te kiezen waarbij de hand geblurd is, maar nog wel te zien is.



Figuur 5: sigma 1, 2 en 3

De volgende code is gebruikt om de edge detection filter aan de preprocessing toe te voegen:

```
#gaussian filter                    #for rgb image
gaussian_filter = gaussian(image, multichannel=True, sigma=2)
```

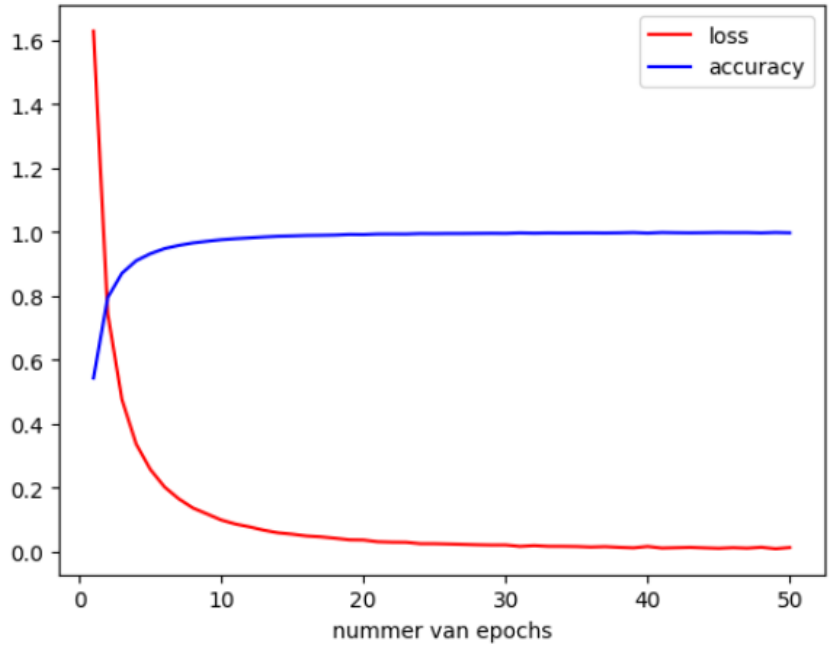
Het toevoegen van de filter aan alle images duurt lang en de tijd om het model 1 epoch te trainen is 25 minuten. Het trainen van dit model is hierdoor vroegtijdig gestopt. De loss na 1 epoch is hoger en de accuracy is lager in vergelijking met de 1^{e} epoch van het 1^{e} model. Het toevoegen van een Gaussian filter is dus niet rendabel en zal niet bij een volgend model toegevoegd worden.

Uiteindelijke model

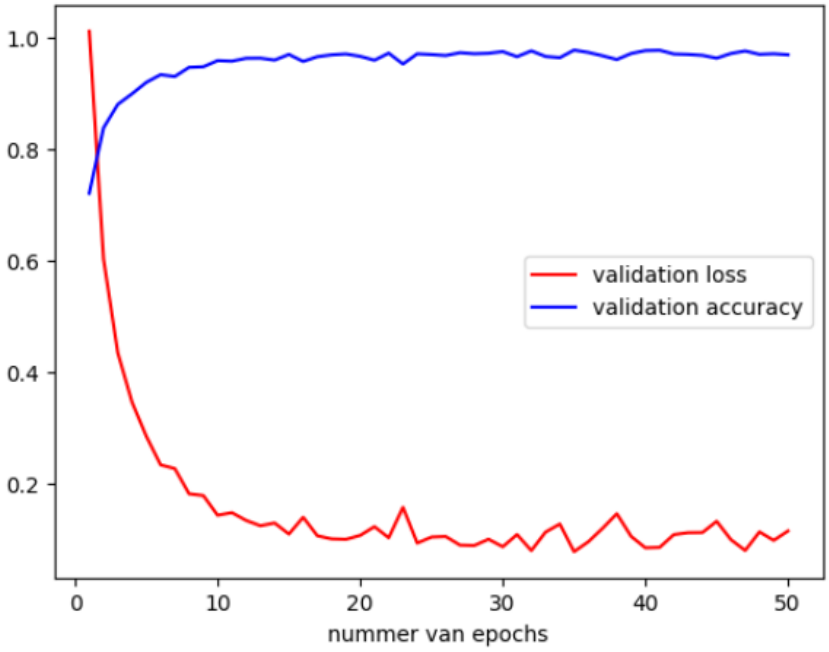
Voor het uiteindelijke model is er gekeken naar wat allemaal voor een lagere loss en hogere accuracy zorgt. Dit is samengevoegd tot de volgende code:

```
num_filters = 10
filter_size = 3
pool_size = 2
num_epochs = 50 #TODO change these vars

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Dit model heeft de volgende resultaten (zie de bijlage 10 voor de volledige resultaten):



Figuur 6: Loss en accuracy van uiteindelijke model na 50 epochs



Figuur 7: validation loss en validation accuracy van uiteindelijke model na 50 epochs

Met de volgende code wordt de letter A uit de testdataset, die niet is gebruikt voor training, gepredict met het model:

```
#open the test image version of the letter
    path =
"C:/Users/emmar/Documents/GitHub/VISN/Eindopdracht/dataset/asl_alphabet_test/a
sl_alphabet_test/A_test.jpg"
    image_to_predict = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

    #preprocess the image the same as images used for training
    compressed_image_to_predict = cv2.resize(image_to_predict, (image_size,
image_size))
    reshaped_image_to_predict = compressed_image_to_predict.reshape(-1,
image_size, image_size, 1)
    normalized_image_to_predict = (reshaped_image_to_predict / 255) - 0.5

    #print the prediction
    prediction = model.predict(normalized_image_to_predict)
    print(prediction)
```

Deze functie is uitgevoerd met verschillende letters uit de testdataset. Het resultaat van de letters A en Z zijn hieronder weergegeven:

A: [[1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]]
Z: [[1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 1.]]

Zoals te zien is in de resultaten, worden er meerdere letters gepredict. Het model zou maar 1 letter moeten predicten, dus is het model overfitted.

Een niet overfitted model

Om ervoor te zorgen dat het model niet overfitted wordt, is er terug gegaan naar een eenvoudig model en de activation in de Dense layer veranderd van sigmoid naar softmax:

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 5

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),
    Flatten(),
    Dense(29, activation="softmax", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

De volgende code is gebruikt om de letters te predicten. In deze code staat een bug, waardoor het alleen mogelijk is om de letter D te predicten, waardoor alleen van deze letter resultaten beschikbaar zijn. Dit wordt later opgelost.

```
for letter in letters:
    path =
"C:/Users/emmar/Documents/GitHub/VISN/Eindopdracht/dataset/asl_alphabet_test/a
sl_alphabet_test/"+letter+"_test.jpg"
    image_to_predict =
cv2.imread("C:/Users/emmar/Documents/GitHub/VISN/Eindopdracht/dataset/asl_alph
abet_test/asl_alphabet_test/D_test.jpg", cv2.IMREAD_GRAYSCALE)
    compressed_image_to_predict = cv2.resize(image_to_predict, (image_size,
image_size))
    reshaped_image_to_predict = compressed_image_to_predict.reshape(-1,
image_size, image_size, 1)

    prediction = model.predict(reshaped_image_to_predict)
    print(prediction)
```

Dit zorgt voor betere resultaten, want er wordt voor iedere prediction maar 1 letter voorspeld. Het resultaat van de letter D is:
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Echter, deze voorspelling geeft de index 2 terug, dit komt overeen met de 3e letter van het alfabet. Dit is de letter C. De voorspelling klopt dus niet.

Vervolgens zijn er een convolutional en pooling layer toegevoegd:

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 5

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),

    Conv2D(num_filters, filter_size),
    MaxPooling2D(pool_size=pool_size),

    Flatten(),
    Dense(29, activation="softmax", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Het resultaat van de prediction van de letter D van de testdataset bij dit model is:
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]

Bij dit resultaat is het percentage voor de letter L het hoogst. Dit is ook niet de goede letter.
Daarna is Relu activation toegevoegd aan de convolutional layers:

```
num_filters = 3
filter_size = 3
pool_size = 2
num_epochs = 5

model = Sequential([
    Conv2D(num_filters, filter_size, activation='relu',
input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),

    Conv2D(num_filters, filter_size, activation='relu'),
    MaxPooling2D(pool_size=pool_size),
```

```
    Flatten(),
    Dense(29, activation="softmax", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_data=(test_images, to_categorical(test_labels)))

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Het resultaat van de prediction van de letter D van de testdataset bij dit model is:
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]

Bij dit resultaat is het percentage voor de letter V het hoogst. Dit is ook niet de goede letter.

Een werkend prediction model

Na feedback gekregen te hebben om ten miste 20 filters te gebruiken, meer layers toe te voegen, een validation split te gebruiken bij de fit functie, de testdataset te normaliseren en de bug opgelost te hebben is dit de code voor het nieuwe model:

```
num_filters = 20
filter_size = 3
pool_size = 2
num_epochs = 10

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),

    # Conv2D(num_filters, filter_size, activation='relu'),
    # MaxPooling2D(pool_size=pool_size),

    # Conv2D(num_filters, filter_size, activation='relu'),
    # MaxPooling2D(pool_size=pool_size),

    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_split=0.15)

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
```

```
print(test_acc)
```

En de code om de predict functie uit te voeren voor alle letters in de testdataset:

```
#for each letter
for i in range(len(letters) - 3):
    #open the test image version of the letter
    path =
"C:/Users/emmar/Documents/GitHub/VISN/Eindopdracht/dataset/asl_alphabet_test/a
sl_alphabet_test/"+letters[i]+"_test.jpg"
    image_to_predict = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

    #preprocess the image the same as images used for training
    compressed_image_to_predict = cv2.resize(image_to_predict, (image_size,
image_size))
    reshaped_image_to_predict = compressed_image_to_predict.reshape(-1,
image_size, image_size, 1)
    normalized_image_to_predict = (reshaped_image_to_predict / 255) - 0.5

    #predict the letter
    prediction = model.predict(normalized_image_to_predict)

    #print the actual letter instead of percentages
    predicted_letter = list(prediction[0]).index(max(prediction[0]))
    # print(prediction)
    print(letters[predicted_letter])
```

Deze code geeft de volgende resultaten na 10 epochs, de 10e epoch resultaten zijn weergegeven (zie de bijlage 14 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	43	0.0686	0.9833	0.1380	0.9577
Test 2	41	0.0726	0.9820	0.1333	0.9609

Tabel 19: Resultaten prediction model, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	2	0.1417	0.9567
Test 2	2	0.1427	0.9597

Tabel 20: Resultaten prediction model, evaluate functie

De letters van de images van de testdataset en de predictions die gemaakt worden voor die letter:

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Test 1	A	A	C	D	A	F	G	H	I	H	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Test 2	A	A	C	D	E	F	G	G	I	J	I	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Tabel 21: Resultaten prediction model, predictions

De predictions zijn nog niet volledig correct dus wordt er een extra convolutional en pooling layer toegevoegd:

```
num_filters = 20
filter_size = 3
pool_size = 2
num_epochs = 10

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),

    Conv2D(num_filters, filter_size, activation='relu'),
    MaxPooling2D(pool_size=pool_size),

    # Conv2D(num_filters, filter_size, activation='relu'),
    # MaxPooling2D(pool_size=pool_size),

    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_split=0.15)

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10e epoch resultaten zijn weergegeven (zie de bijlage 15 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	70	0.0526	0.9851	0.0847	0.9721
Test 2	68	0.0610	0.9823	0.0739	0.9790

Tabel 22: Resultaten 2x convolutional en pooling, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	3	0.0855	0.9728
Test 2	4	0.0789	0.9765

Tabel 23: Resultaten 2x convolutional en pooling, evaluate functie

De letters van de images van de testdataset en de predictions die gemaakt worden voor die letter:

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Test 1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Test 2	A	B	C	D	E	F	G	G	I	J	I	L	M	N	O	P	Q	R	S	T	U	V	V	X	Y	Z

Tabel 24: Resultaten 2x convolutional en pooling, predictions

Er zitten nog fouten in de predictions, dus er wordt nog een concolutional en pooling layer toegevoegd:

```
num_filters = 20
filter_size = 3
pool_size = 2
num_epochs = 10

model = Sequential([
    Conv2D(num_filters, filter_size, input_shape=train_images[0].shape),
    MaxPooling2D(pool_size=pool_size),

    Conv2D(num_filters, filter_size, activation='relu'),
    MaxPooling2D(pool_size=pool_size),

    Conv2D(num_filters, filter_size, activation='relu'),
    MaxPooling2D(pool_size=pool_size),

    Flatten(),
    Dense(29, activation="sigmoid", name="dense")
])

model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_images, to_categorical(train_labels), epochs=num_epochs,
validation_split=0.15)

test_loss, test_acc =
model.evaluate(test_images,  to_categorical(test_labels), verbose=2)
print(test_acc)
```

Deze code geeft de volgende resultaten na 10 epochs, de 10^{e} epoch resultaten zijn weergegeven (zie de bijlage 16 voor de volledige resultaten):

	Tijd voor 1 epoch (s)	Loss	Accuracy	Validation loss	Validation accuracy
Test 1	74	0.0888	0.9697	0.1118	0.9643
Test 2	73	0.0970	0.9674	0.0994	0.9656

Tabel 25: Resultaten 3x convolutional en pooling, fit functie

De resultaten van de evaluation van dit model:

	Tijd voor evaluate (s)	Loss	Accuracy
Test 1	3	0.1118	0.9651
Test 2	5	0.0949	0.9696

Tabel 26: Resultaten 3x convolutional en pooling, evaluate functie

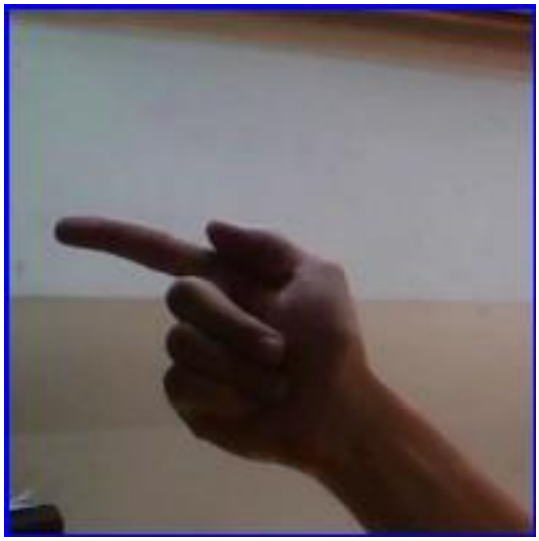
De letters van de images van de testdataset en de predictions die gemaakt worden voor die letter:

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Test 1	A	B	C	D	E	F	G	G	I	G	K	L	M	N	O	G	C	R	S	T	U	V	V	X	Y	Z
Test 2	A	B	C	D	E	F	G	G	I	G	K	L	M	N	O	P	P	R	T	T	U	K	K	X	Y	Z

Tabel 27: Resultaten 3x convolutional en pooling, predictions

De predictions van dit model kloppen ook niet helemaal.

De predictions van het model met 2 paar convolutional en pooling layers, lijkt de beste predictions te geven en wordt dus gebruikt voor het uiteindelijke model. De letter G wordt vaak als verkeerde predictie gegeven. Er zijn meerdere letters waarbij 1 of 2 vingers zijn uitgestoken en de rest ingetrokken. Dit zou een rede kunnen zijn dat de G voor meerdere letters gepredict wordt.



Figuur 8: De letter G uit de testdataset

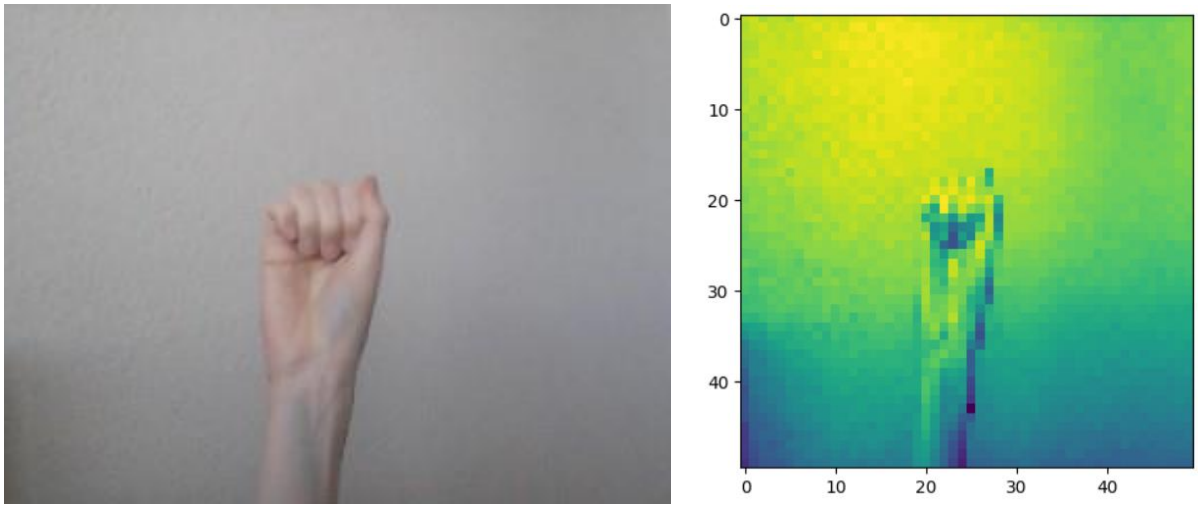
Applicatie

Om zelf input te geven aan het model, wordt er een applicatie gemaakt waarbij de capture van een webcam gebruikt kan worden als input van de predict functie van het model. Door middel van OpenCV wordt er een capture genomen van de webcam van de gebruiker. Hiermee kan de gebruiker zelf een ASL-letter uitbeelden en de bijbehorende letter terugkrijgen. Om ervoor te zorgen dat de goede letter gepredict wordt, krijgt de capture dezelfde preprocessing als de trainingsdata. Echter, iedere webcam heeft andere afmetingen waardoor de images vervormd kunnen worden als ze in de preprocessing omgezet worden naar een 50x50 image. Zodat het model niet iedere keer opnieuw getraind moet worden bij het opstarten, wordt het model opgeslagen en ingeladen.

Captures als input van model

Om de werking van de applicatie makkelijk te testen, is er een testdataset gemaakt met voor iedere ASL-letter een capture. De volledige dataset is te vinden op GitHub[11]. Dit is een voorbeeld van de letter A en de letter A na de preprocessing:

[11] Raijmakers, E. (2023, 9 april). VISN. *Github.* Geraadpleegd op 9 april 2023, van https://github.com/EmmaRaijmakers/VISN/tree/main/Eindopdracht/dataset/asl_alphabet_test/als_alphabet_test_captures



Figuur 9: Een capture van A met bijbehorende preprocessed A

De output van de capture testdataset als input van het model (zie de bijlage 17 voor de volledige resultaten):

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Capture	Z	W	Q	I	D	Z	G	space	W	W	W	Z	W	W	D	Q	Q	W	B	D	W	W	W	W	W	D

Tabel 28: Resultaten predictions capture testdataset

De predictions van het model voor de captured images zijn grotendeels fout. Er zijn 3 letters goed voorspeld. Voor de totale discussie hierover zie het kopje Discussie. Een rede voor de slechte predictions zou kunnen zijn dat de captured images niet genoeg lijken op de trainimages. Dit is te verbeteren door random bewerkingen[12] aan de trainimages toe te voegen. Daarvoor is de volgende code toegevoegd aan het begin van het model:

```
RandomContrast(0.2),
RandomBrightness(0.2, value_range=(-0.5, 0.5)),
```

Deze code zorgt ervoor dat er voor 20% van de images een random contrast en voor 20% een random brightness wordt toegevoegd. Hierdoor wordt het model getraind op meer verschillende waardes en zou het nieuwe images makkelijker moeten herkennen.

Na het trainen van het model met de nieuwe random bewerkingen, geeft de evaluate functie de volgende uitkomst (zie de bijlages 19 en 20 voor de volledige uitkomst):

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Testset	A	B	C	D	E	F	P	H	D	P	K	L	M	N	O	P	Q	L	L	L	U	W	W	L	Y	Z
Capture	A	W	D	F	D	F	G	Y	W	G	W	D	W	W	D	D	G	W	B	D	W	W	W	W	F	F

Tabel 29: Resultaten na random bewerkingen

[12] Keras. Preprocessing layers. *Keras.* Geraadpleegd op 9 april 2023, van https://keras.io/api/layers/preprocessing_layers/

De random bewerkingen hebben voor bij de testdataset niet voor verbeteringen gezorgd. Bij het model met 2x convolutional en pooling layers waren alle letters goed voorspeld bij test 1 en 2 fout voorspeld bij test 2. In dit model zijn er 8 verkeerd voorspeld. In de capture testdataset is er wel vooruitgang. In het vorige model waren 3 letters goed voorspeld en in dit model 4. Om hierover een goede conclusie te trekken zouden de modellen vaker getest moeten worden.

Discussie

Uiteindelijk had het model niet de gewenste uitkomsten. Detectie van ASL-letters uit images door middel van Tensorflow is mogelijk, maar daarvoor zouden aanpassingen gemaakt moeten worden.

Ten eerste zou er een dataset moeten komen met meer verschillende soorten images. De images uit de huidige dataset zijn gemaakt met vrijwel dezelfde achtergrond, positie van de camera en maar van 3 personen. Hierdoor zijn nieuwe images met andere achtergronden, afstand tot de camera en huidskleuren moeilijker te herkennen. Dit probleem zou ook opgelost kunnen worden door de nieuwe images voorbewerkingen te geven zodat ze op de images uit de traindata lijken. Ook kan zelf een dataset gemaakt worden of de huidige dataset kan uitgebreid worden met nieuwe images.

Een probleem met het krijgen van images vanaf een camera bij het predicten met captured images, is dat de gebruikte webcam een resolutie heeft van 320x240. De images uit de dataset zijn 200x200, wat vierkante images zijn. Om de images van de webcam hetzelfde te maken als de images uit de dataset, moest de image vervormd worden. Hierdoor is het lastiger om de image te herkennen. Dit zou opgelost kunnen worden door een grotere dataset te maken, of de image niet te vervormen, maar een 200x200 stuk eruit te knippen. Ook zijn de images van de captured dataset ver van de webcam af genomen en zijn ze dus lastiger om te herkennen. Dit probleem is ook op te lossen met een grotere dataset, of de captures dichter bij de camera te nemen.

Als laatste, de laptop die gebruikt is voor het onderzoek is niet erg snel. Het runnen van de fit functie kost ongeveer 40 seconden per epoch, dit is voor 10 epochs al meer dan 6 minuten. Hierdoor zijn testen maar een paar keer (1, 2, 3 of 5 keer) gerund. Om waardes te krijgen die correcter zijn, moeten testen vaak gerund worden. Dit was door tijdsgebrek niet mogelijk.

Literatuur lijst

AKASH. (2018, 22 april). ASL Alphabet. Kaggle. Geraadpleegd op 9 april 2023, van https://www.kaggle.com/datasets/grassknoted/asl-alphabet

Keras. Keras documentation: Keras API reference. Keras. Geraadpleegd op 9 april 2023, van https://keras.io/api/

Keras. Preprocessing layers. *Keras.* Geraadpleegd op 9 april 2023, van https://keras.io/api/layers/preprocessing_layers/

OpenCV. OpenCV-Python Tutorials. OpenCV. Geraadpleegd op 9 april 2023, van https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

Raijmakers, E. (2023, 9 april). VISN. Github. Geraadpleegd op 9 april 2023, van https://github.com/EmmaRaijmakers/VISN/tree/main/Eindopdracht

Raijmakers, E. (2023, 9 april). VISN. *Github.* Geraadpleegd op 9 april 2023, van https://github.com/EmmaRaijmakers/VISN/tree/main/Eindopdracht/dataset/asl_alphabet_test/als_alphabet_test_captures

Sentdex. (2018, 19 augustus). Convolutional Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.3. YouTube. Geraadpleegd op 9 april 2023, van https://www.youtube.com/watch?v=WvoLTXIjBYU

T. B. (2022, 17 februari). Massive Tutorial on Image Processing And Preparation For Deep Learning in Python, #1. Towards Data Science. Geraadpleegd op 9 april 2023, van https://towardsdatascience.com/massive-tutorial-on-image-processing-and-preparation-for-deep-learning-in-python-1-e534ee42f122

Wikipedia contributors. (2023, april 1). American Sign Language. Wikipedia. Geraadpleegd op 9 april 2023, van https://en.wikipedia.org/wiki/American_Sign_Language

Yamamoto-Roijers, D. Opdrachten Neurale Netwerken. Canvas. Geraadpleegd op 9 april 2023, van https://canvas.hu.nl/courses/32732/assignments/234890?module_item_id=851331

Yamamoto-Roijers, D. Video's module 6. Canvas. Geraadpleegd op 9 april 2023, van https://canvas.hu.nl/courses/32732/pages/videos-module-6?module_item_id=851333

Bijlage

1. Resultaten eerste model:

```
2407/2407 [==============================] - 29s 12ms/step - loss: 0.2760 -
accuracy: 0.9252 - val_loss: 0.3366 - val_accuracy: 0.8981
313/313 - 2s - loss: 0.3366 - accuracy: 0.8981 - 2s/epoch - 5ms/step
0.8981000185012817

2407/2407 [==============================] - 31s 13ms/step - loss: 0.2475 -
accuracy: 0.9366 - val_loss: 0.2998 - val_accuracy: 0.9091
313/313 - 2s - loss: 0.2998 - accuracy: 0.9091 - 2s/epoch - 5ms/step
0.9090999960899353

2407/2407 [==============================] - 28s 12ms/step - loss: 0.3532 -
accuracy: 0.9016 - val_loss: 0.4113 - val_accuracy: 0.8776
313/313 - 1s - loss: 0.4113 - accuracy: 0.8776 - 1s/epoch - 5ms/step
0.8776000142097473

2407/2407 [==============================] - 28s 12ms/step - loss: 0.2798 -
accuracy: 0.9269 - val_loss: 0.3278 - val_accuracy: 0.9030
313/313 - 2s - loss: 0.3278 - accuracy: 0.9030 - 2s/epoch - 5ms/step
0.902999997138977

2407/2407 [==============================] - 25s 11ms/step - loss: 0.3105 -
accuracy: 0.9134 - val_loss: 0.3632 - val_accuracy: 0.8915
313/313 - 2s - loss: 0.3632 - accuracy: 0.8915 - 2s/epoch - 5ms/step
0.8914999961853027
```

2. Resultaten toevoeging Relu layer

```
2407/2407 [==============================] - 34s 14ms/step - loss: 0.5228 -
accuracy: 0.8350 - val_loss: 0.6135 - val_accuracy: 0.8039
313/313 - 3s - loss: 0.6135 - accuracy: 0.8039 - 3s/epoch - 10ms/step
0.8039000034332275

2407/2407 [==============================] - 31s 13ms/step - loss: 0.3105 -
accuracy: 0.9020 - val_loss: 0.3521 - val_accuracy: 0.8913
313/313 - 2s - loss: 0.3521 - accuracy: 0.8913 - 2s/epoch - 6ms/step
0.8913000226020813

2407/2407 [==============================] - 28s 12ms/step - loss: 0.3064 -
accuracy: 0.9106 - val_loss: 0.3444 - val_accuracy: 0.8984
313/313 - 2s - loss: 0.3444 - accuracy: 0.8984 - 2s/epoch - 5ms/step
0.8984000086784363

2407/2407 [==============================] - 34s 14ms/step - loss: 0.4196 -
accuracy: 0.8713 - val_loss: 0.4179 - val_accuracy: 0.8708
313/313 - 2s - loss: 0.4179 - accuracy: 0.8708 - 2s/epoch - 6ms/step
0.8708000183105469

2407/2407 [==============================] - 29s 12ms/step - loss: 0.3506 -
accuracy: 0.8879 - val_loss: 0.3759 - val_accuracy: 0.8788
313/313 - 2s - loss: 0.3759 - accuracy: 0.8788 - 2s/epoch - 5ms/step
0.8787999749183655
```

3. Resultaten 2x convolutional en pooling

```
2407/2407 [==============================] - 42s 18ms/step - loss: 0.7242 -
accuracy: 0.7732 - val_loss: 0.7574 - val_accuracy: 0.7693
313/313 - 3s - loss: 0.7574 - accuracy: 0.7693 - 3s/epoch - 10ms/step
0.7692999839782715

2407/2407 [==============================] - 41s 17ms/step - loss: 0.6509 -
accuracy: 0.7980 - val_loss: 0.7092 - val_accuracy: 0.7807
313/313 - 3s - loss: 0.7092 - accuracy: 0.7807 - 3s/epoch - 9ms/step
0.7807000279426575

2407/2407 [==============================] - 41s 17ms/step - loss: 0.6635 -
accuracy: 0.7871 - val_loss: 0.6833 - val_accuracy: 0.7833
313/313 - 2s - loss: 0.6833 - accuracy: 0.7833 - 2s/epoch - 8ms/step
0.78329998254776

2407/2407 [==============================] - 40s 17ms/step - loss: 0.7527 -
accuracy: 0.7636 - val_loss: 0.7577 - val_accuracy: 0.7572
313/313 - 2s - loss: 0.7577 - accuracy: 0.7572 - 2s/epoch - 7ms/step
0.7572000026702881

2407/2407 [==============================] - 41s 17ms/step - loss: 0.6626 -
accuracy: 0.7941 - val_loss: 0.6826 - val_accuracy: 0.7883
313/313 - 3s - loss: 0.6826 - accuracy: 0.7883 - 3s/epoch - 10ms/step
0.7882999777793884
```

4. Resultaten softmax ipv sigmoid

```
2407/2407 [==============================] - 38s 16ms/step - loss: 0.7011 -
accuracy: 0.7800 - val_loss: 0.7271 - val_accuracy: 0.7783
```

```
313/313 - 2s - loss: 0.7271 - accuracy: 0.7783 - 2s/epoch - 7ms/step
0.7782999873161316

2407/2407 [==============================] - 38s 16ms/step - loss: 0.6979 -
accuracy: 0.7863 - val_loss: 0.7218 - val_accuracy: 0.7846
313/313 - 3s - loss: 0.7218 - accuracy: 0.7846 - 3s/epoch - 11ms/step
0.784600019454956
```

5. Resultaten meer filters

```
2407/2407 [==============================] - 29s 12ms/step - loss: 0.1462 -
accuracy: 0.9625 - val_loss: 0.2192 - val_accuracy: 0.9349
313/313 - 2s - loss: 0.2192 - accuracy: 0.9349 - 2s/epoch - 5ms/step
0.9348999857902527

2407/2407 [==============================] - 30s 13ms/step - loss: 0.1436 -
accuracy: 0.9632 - val_loss: 0.1912 - val_accuracy: 0.9453
313/313 - 2s - loss: 0.1912 - accuracy: 0.9453 - 2s/epoch - 6ms/step
0.9452999830245972

2407/2407 [==============================] - 30s 12ms/step - loss: 0.1296 -
accuracy: 0.9686 - val_loss: 0.1863 - val_accuracy: 0.9436
313/313 - 2s - loss: 0.1863 - accuracy: 0.9436 - 2s/epoch - 5ms/step
0.9435999989509583
```

6. Resultaten grotere filter size

```
2407/2407 [==============================] - 43s 18ms/step - loss: 0.4653 -
accuracy: 0.8606 - val_loss: 0.4920 - val_accuracy: 0.8491
313/313 - 2s - loss: 0.4920 - accuracy: 0.8491 - 2s/epoch - 6ms/step
0.8490999937057495

2407/2407 [==============================] - 44s 18ms/step - loss: 0.3583 -
accuracy: 0.8961 - val_loss: 0.4246 - val_accuracy: 0.8713
313/313 - 2s - loss: 0.4246 - accuracy: 0.8713 - 2s/epoch - 6ms/step
0.8712999820709229

2407/2407 [==============================] - 43s 18ms/step - loss: 0.4328 -
accuracy: 0.8699 - val_loss: 0.4868 - val_accuracy: 0.8527
313/313 - 2s - loss: 0.4868 - accuracy: 0.8527 - 2s/epoch - 6ms/step
0.8526999950408936
```

7. Resultaten grotere pool size

```
2407/2407 [==============================] - 28s 12ms/step - loss: 0.7480 -
accuracy: 0.7775 - val_loss: 0.8317 - val_accuracy: 0.7569
313/313 - 2s - loss: 0.8317 - accuracy: 0.7569 - 2s/epoch - 5ms/step
0.7569000124931335

2407/2407 [==============================] - 27s 11ms/step - loss: 0.9550 -
accuracy: 0.7167 - val_loss: 0.9783 - val_accuracy: 0.7131
313/313 - 2s - loss: 0.9783 - accuracy: 0.7131 - 2s/epoch - 5ms/step
0.713100016117096

2407/2407 [==============================] - 33s 14ms/step - loss: 0.8308 -
accuracy: 0.7552 - val_loss: 0.8532 - val_accuracy: 0.7492
```

```
313/313 - 2s - loss: 0.8532 - accuracy: 0.7492 - 2s/epoch - 5ms/step
0.7491999864578247
```

8. Resultaten meer epochs

```
2407/2407 [==============================] - 29s 12ms/step - loss: 0.1696 -
accuracy: 0.9514 - val_loss: 0.2622 - val_accuracy: 0.9177
313/313 - 2s - loss: 0.2622 - accuracy: 0.9177 - 2s/epoch - 5ms/step
0.9176999926567078

2407/2407 [==============================] - 34s 14ms/step - loss: 0.1627 -
accuracy: 0.9542 - val_loss: 0.2413 - val_accuracy: 0.9303
313/313 - 2s - loss: 0.2413 - accuracy: 0.9303 - 2s/epoch - 6ms/step
0.9302999973297119

2407/2407 [==============================] - 31s 13ms/step - loss: 0.1364 -
accuracy: 0.9635 - val_loss: 0.2272 - val_accuracy: 0.9345
313/313 - 2s - loss: 0.2272 - accuracy: 0.9345 - 2s/epoch - 5ms/step
0.934499979019165
```

9. Resultaten edge detection

```
2407/2407 [==============================] - 29s 12ms/step - loss: 0.2641 -
accuracy: 0.9113 - val_loss: 0.4480 - val_accuracy: 0.8481
313/313 - 2s - loss: 0.4480 - accuracy: 0.8481 - 2s/epoch - 5ms/step
0.8481000065803528

2407/2407 [==============================] - 36s 15ms/step - loss: 0.3051 -
accuracy: 0.8975 - val_loss: 0.4490 - val_accuracy: 0.8501
313/313 - 3s - loss: 0.4490 - accuracy: 0.8501 - 3s/epoch - 9ms/step
0.8500999808311462

2407/2407 [==============================] - 28s 12ms/step - loss: 0.2909 -
accuracy: 0.9015 - val_loss: 0.4388 - val_accuracy: 0.8602
313/313 - 2s - loss: 0.4388 - accuracy: 0.8602 - 2s/epoch - 5ms/step
0.8601999878883362
```

10. Resultaten uiteindelijke model

```
Epoch 1/50
2407/2407 [==============================] - 32s 13ms/step - loss: 1.6279 - accuracy: 0.5426 -
val_loss: 1.0107 - val_accuracy: 0.7203
Epoch 2/50
2407/2407 [==============================] - 31s 13ms/step - loss: 0.7440 - accuracy: 0.7945 -
val_loss: 0.6022 - val_accuracy: 0.8376
Epoch 3/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.4757 - accuracy: 0.8704 -
val_loss: 0.4343 - val_accuracy: 0.8797
Epoch 4/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.3361 - accuracy: 0.9096 -
val_loss: 0.3455 - val_accuracy: 0.8987
Epoch 5/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.2567 - accuracy: 0.9315 -
val_loss: 0.2846 - val_accuracy: 0.9192
```

```
Epoch 6/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.2017 - accuracy: 0.9476 -
val_loss: 0.2333 - val_accuracy: 0.9329
Epoch 7/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.1647 - accuracy: 0.9575 -
val_loss: 0.2266 - val_accuracy: 0.9297
Epoch 8/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.1359 - accuracy: 0.9652 -
val_loss: 0.1813 - val_accuracy: 0.9461
Epoch 9/50
2407/2407 [==============================] - 40s 16ms/step - loss: 0.1172 - accuracy: 0.9705 -
val_loss: 0.1782 - val_accuracy: 0.9471
Epoch 10/50
2407/2407 [==============================] - 36s 15ms/step - loss: 0.0983 - accuracy: 0.9754 -
val_loss: 0.1429 - val_accuracy: 0.9581
Epoch 11/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0857 - accuracy: 0.9785 -
val_loss: 0.1476 - val_accuracy: 0.9571
Epoch 12/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0768 - accuracy: 0.9811 -
val_loss: 0.1336 - val_accuracy: 0.9623
Epoch 13/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0663 - accuracy: 0.9839 -
val_loss: 0.1241 - val_accuracy: 0.9625
Epoch 14/50
2407/2407 [==============================] - 35s 14ms/step - loss: 0.0583 - accuracy: 0.9862 -
val_loss: 0.1290 - val_accuracy: 0.9591
Epoch 15/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0540 - accuracy: 0.9872 -
val_loss: 0.1092 - val_accuracy: 0.9692
Epoch 16/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0484 - accuracy: 0.9886 -
val_loss: 0.1393 - val_accuracy: 0.9566
Epoch 17/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0457 - accuracy: 0.9891 -
val_loss: 0.1060 - val_accuracy: 0.9652
Epoch 18/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0414 - accuracy: 0.9899 -
val_loss: 0.1005 - val_accuracy: 0.9685
Epoch 19/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0364 - accuracy: 0.9918 -
val_loss: 0.0997 - val_accuracy: 0.9700
Epoch 20/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0358 - accuracy: 0.9914 -
val_loss: 0.1068 - val_accuracy: 0.9658
Epoch 21/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0304 - accuracy: 0.9931 -
val_loss: 0.1225 - val_accuracy: 0.9588
Epoch 22/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0291 - accuracy: 0.9932 -
val_loss: 0.1025 - val_accuracy: 0.9715
```

```
Epoch 23/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0287 - accuracy: 0.9931 -
val_loss: 0.1571 - val_accuracy: 0.9520
Epoch 24/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0244 - accuracy: 0.9945 -
val_loss: 0.0930 - val_accuracy: 0.9703
Epoch 25/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0243 - accuracy: 0.9943 -
val_loss: 0.1038 - val_accuracy: 0.9690
Epoch 26/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0233 - accuracy: 0.9947 -
val_loss: 0.1048 - val_accuracy: 0.9673
Epoch 27/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0222 - accuracy: 0.9947 -
val_loss: 0.0893 - val_accuracy: 0.9723
Epoch 28/50
2407/2407 [==============================] - 36s 15ms/step - loss: 0.0210 - accuracy: 0.9951 -
val_loss: 0.0886 - val_accuracy: 0.9705
Epoch 29/50
2407/2407 [==============================] - 34s 14ms/step - loss: 0.0202 - accuracy: 0.9954 -
val_loss: 0.0999 - val_accuracy: 0.9713
Epoch 30/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0203 - accuracy: 0.9951 -
val_loss: 0.0864 - val_accuracy: 0.9743
Epoch 31/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0159 - accuracy: 0.9965 -
val_loss: 0.1084 - val_accuracy: 0.9652
Epoch 32/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0183 - accuracy: 0.9957 -
val_loss: 0.0795 - val_accuracy: 0.9756
Epoch 33/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0161 - accuracy: 0.9963 -
val_loss: 0.1127 - val_accuracy: 0.9656
Epoch 34/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0160 - accuracy: 0.9961 -
val_loss: 0.1272 - val_accuracy: 0.9634
Epoch 35/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0155 - accuracy: 0.9963 -
val_loss: 0.0776 - val_accuracy: 0.9770
Epoch 36/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0139 - accuracy: 0.9965 -
val_loss: 0.0954 - val_accuracy: 0.9728
Epoch 37/50
2407/2407 [==============================] - 38s 16ms/step - loss: 0.0151 - accuracy: 0.9962 -
val_loss: 0.1195 - val_accuracy: 0.9668
Epoch 38/50
2407/2407 [==============================] - 36s 15ms/step - loss: 0.0130 - accuracy: 0.9967 -
val_loss: 0.1457 - val_accuracy: 0.9601
Epoch 39/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0114 - accuracy: 0.9976 -
val_loss: 0.1050 - val_accuracy: 0.9709
```

```
Epoch 40/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0157 - accuracy: 0.9959 -
val_loss: 0.0845 - val_accuracy: 0.9761
Epoch 41/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0107 - accuracy: 0.9978 -
val_loss: 0.0853 - val_accuracy: 0.9767
Epoch 42/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0118 - accuracy: 0.9971 -
val_loss: 0.1083 - val_accuracy: 0.9700
Epoch 43/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0129 - accuracy: 0.9967 -
val_loss: 0.1116 - val_accuracy: 0.9692
Epoch 44/50
2407/2407 [==============================] - 33s 14ms/step - loss: 0.0113 - accuracy: 0.9971 -
val_loss: 0.1119 - val_accuracy: 0.9676
Epoch 45/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0099 - accuracy: 0.9977 -
val_loss: 0.1321 - val_accuracy: 0.9625
Epoch 46/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0119 - accuracy: 0.9974 -
val_loss: 0.0991 - val_accuracy: 0.9709
Epoch 47/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0103 - accuracy: 0.9975 -
val_loss: 0.0797 - val_accuracy: 0.9754
Epoch 48/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0133 - accuracy: 0.9965 -
val_loss: 0.1131 - val_accuracy: 0.9694
Epoch 49/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0085 - accuracy: 0.9979 -
val_loss: 0.0978 - val_accuracy: 0.9704
Epoch 50/50
2407/2407 [==============================] - 32s 13ms/step - loss: 0.0124 - accuracy: 0.9969 -
val_loss: 0.1147 - val_accuracy: 0.9686
313/313 - 2s - loss: 0.1147 - accuracy: 0.9686 - 2s/epoch - 5ms/step
0.9685999751091003
```

11. Resultaten niet overfitted, softmax ipv sigmoid

```
1/1 [==============================] - 0s 36ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 38ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 42ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 35ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 36ms/step
```

```
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 30ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 43ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 45ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 34ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 69ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 32ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 33ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 35ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 23ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 27ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 22ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 26ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 26ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 25ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 29ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 31ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 29ms/step
```

```
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 20ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 29ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 24ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 26ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 27ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 34ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 24ms/step
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0.]]
```

12. Resultaten niet overfitted, 2x convolutional en pooling + softmax

```
1/1 [==============================] - 0s 149ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 37ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 41ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 43ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
```

```
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 41ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 34ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 43ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 50ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 51ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 47ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 48ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
```

```
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 69ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 34ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 36ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 33ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 33ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 44ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 41ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 32ms/step
```

```
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 36ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 35ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 35ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 38ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 31ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 36ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 40ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
```

```
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 33ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 34ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
1/1 [==============================] - 0s 30ms/step
[[0.000000e+00 0.000000e+00 3.185494e-30 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]]
```

13. Resultaten niet overfitted, 2x convolutional en pooling + softmax + relu activation

```
1/1 [==============================] - 0s 33ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 32ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 30ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 34ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 27ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 35ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 34ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 31ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
```

0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 28ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 34ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 28ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 55ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 43ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 33ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 33ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 36ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 37ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 29ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 32ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 31ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 29ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 31ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 29ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 76ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 29ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.

```
 0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 24ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 36ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 52ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
1/1 [==============================] - 0s 44ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
  0. 0. 0. 0. 0.]]
```

14. Resultaten prediction toegevoegd

```
2046/2046 [==============================] - 43s 21ms/step - loss: 0.0686 -
accuracy: 0.9833 - val_loss: 0.1380 - val_accuracy: 0.9577
313/313 - 2s - loss: 0.1417 - accuracy: 0.9567 - 2s/epoch - 7ms/step
0.9567000269889832

1/1 [==============================] - 0s 48ms/step
A
1/1 [==============================] - 0s 26ms/step
A
1/1 [==============================] - 0s 28ms/step
C
1/1 [==============================] - 0s 26ms/step
D
1/1 [==============================] - 0s 21ms/step
A
1/1 [==============================] - 0s 21ms/step
F
1/1 [==============================] - 0s 33ms/step
G
1/1 [==============================] - 0s 28ms/step
H
1/1 [==============================] - 0s 26ms/step
I
1/1 [==============================] - 0s 29ms/step
H
1/1 [==============================] - 0s 27ms/step
K
1/1 [==============================] - 0s 28ms/step
L
1/1 [==============================] - 0s 20ms/step
M
1/1 [==============================] - 0s 24ms/step
N
1/1 [==============================] - 0s 22ms/step
O
1/1 [==============================] - 0s 22ms/step
P
1/1 [==============================] - 0s 32ms/step
Q
```

```
1/1 [==============================] - 0s 41ms/step
R
1/1 [==============================] - 0s 25ms/step
S
1/1 [==============================] - 0s 22ms/step
T
1/1 [==============================] - 0s 20ms/step
U
1/1 [==============================] - 0s 22ms/step
V
1/1 [==============================] - 0s 24ms/step
W
1/1 [==============================] - 0s 20ms/step
X
1/1 [==============================] - 0s 23ms/step
Y
1/1 [==============================] - 0s 24ms/step
Z

2046/2046 [==============================] - 41s 20ms/step - loss: 0.0726 -
accuracy: 0.9820 - val_loss: 0.1333 - val_accuracy: 0.9609
313/313 - 2s - loss: 0.1427 - accuracy: 0.9597 - 2s/epoch - 8ms/step
0.9596999883651733

1/1 [==============================] - 0s 54ms/step
A
1/1 [==============================] - 0s 25ms/step
A
1/1 [==============================] - 0s 25ms/step
C
1/1 [==============================] - 0s 27ms/step
D
1/1 [==============================] - 0s 24ms/step
E
1/1 [==============================] - 0s 25ms/step
F
1/1 [==============================] - 0s 24ms/step
G
1/1 [==============================] - 0s 78ms/step
G
1/1 [==============================] - 0s 26ms/step
I
1/1 [==============================] - 0s 22ms/step
J
1/1 [==============================] - 0s 23ms/step
I
1/1 [==============================] - 0s 45ms/step
L
1/1 [==============================] - 0s 32ms/step
M
1/1 [==============================] - 0s 33ms/step
N
1/1 [==============================] - 0s 35ms/step
O
1/1 [==============================] - 0s 23ms/step
P
```

```
1/1 [==============================] - 0s 19ms/step
Q
1/1 [==============================] - 0s 33ms/step
R
1/1 [==============================] - 0s 21ms/step
S
1/1 [==============================] - 0s 25ms/step
T
1/1 [==============================] - 0s 24ms/step
U
1/1 [==============================] - 0s 29ms/step
V
1/1 [==============================] - 0s 25ms/step
W
1/1 [==============================] - 0s 29ms/step
X
1/1 [==============================] - 0s 29ms/step
Y
1/1 [==============================] - 0s 23ms/step
Z
```

15. Resultaten prediction toegevoegd, 2x convolutional en pooling

```
2046/2046 [==============================] - 70s 34ms/step - loss: 0.0526 -
accuracy: 0.9851 - val_loss: 0.0847 - val_accuracy: 0.9721
313/313 - 3s - loss: 0.0855 - accuracy: 0.9728 - 3s/epoch - 10ms/step
0.9728000164031982

1/1 [==============================] - 0s 160ms/step
A
1/1 [==============================] - 0s 38ms/step
B
1/1 [==============================] - 0s 25ms/step
C
1/1 [==============================] - 0s 28ms/step
D
1/1 [==============================] - 0s 30ms/step
E
1/1 [==============================] - 0s 45ms/step
F
1/1 [==============================] - 0s 38ms/step
G
1/1 [==============================] - 0s 36ms/step
H
1/1 [==============================] - 0s 29ms/step
I
1/1 [==============================] - 0s 37ms/step
J
1/1 [==============================] - 0s 48ms/step
K
1/1 [==============================] - 0s 44ms/step
L
1/1 [==============================] - 0s 40ms/step
M
```

1/1 [==============================] - 0s 37ms/step
N
1/1 [==============================] - 0s 34ms/step
O
1/1 [==============================] - 0s 35ms/step
P
1/1 [==============================] - 0s 54ms/step
Q
1/1 [==============================] - 0s 53ms/step
R
1/1 [==============================] - 0s 55ms/step
S
1/1 [==============================] - 0s 52ms/step
T
1/1 [==============================] - 0s 34ms/step
U
1/1 [==============================] - 0s 28ms/step
V
1/1 [==============================] - 0s 30ms/step
W
1/1 [==============================] - 0s 30ms/step
X
1/1 [==============================] - 0s 30ms/step
Y
1/1 [==============================] - 0s 33ms/step
Z

2046/2046 [==============================] - 68s 33ms/step - loss: 0.0610 - accuracy: 0.9823 - val_loss: 0.0739 - val_accuracy: 0.9790
313/313 - 4s - loss: 0.0789 - accuracy: 0.9765 - 4s/epoch - 11ms/step
0.9764999747276306

1/1 [==============================] - 0s 189ms/step
A
1/1 [==============================] - 0s 60ms/step
B
1/1 [==============================] - 0s 52ms/step
C
1/1 [==============================] - 0s 52ms/step
D
1/1 [==============================] - 0s 49ms/step
E
1/1 [==============================] - 0s 61ms/step
F
1/1 [==============================] - 0s 62ms/step
G
1/1 [==============================] - 0s 54ms/step
G
1/1 [==============================] - 0s 53ms/step
I
1/1 [==============================] - 0s 57ms/step
J

1/1 [==============================] - 0s 63ms/step
K
1/1 [==============================] - 0s 50ms/step
L
1/1 [==============================] - 0s 53ms/step
M
1/1 [==============================] - 0s 58ms/step
N
1/1 [==============================] - 0s 56ms/step
O
1/1 [==============================] - 0s 52ms/step
P
1/1 [==============================] - 0s 35ms/step
Q
1/1 [==============================] - 0s 34ms/step
R
1/1 [==============================] - 0s 33ms/step
S
1/1 [==============================] - 0s 31ms/step
T
1/1 [==============================] - 0s 27ms/step
U
1/1 [==============================] - 0s 38ms/step
V
1/1 [==============================] - 0s 50ms/step
V
1/1 [==============================] - 0s 43ms/step
X
1/1 [==============================] - 0s 32ms/step
Y
1/1 [==============================] - 0s 38ms/step
Z

16. Resultaten prediction toegevoegd, 3x convolutional en pooling

2046/2046 [==============================] - 74s 36ms/step - loss: 0.0888 - accuracy: 0.9697 - val_loss: 0.1118 - val_accuracy: 0.9643
313/313 - 3s - loss: 0.1118 - accuracy: 0.9651 - 3s/epoch - 10ms/step
0.9650999903678894

1/1 [==============================] - 0s 172ms/step
A
1/1 [==============================] - 0s 58ms/step
B
1/1 [==============================] - 0s 60ms/step
C
1/1 [==============================] - 0s 63ms/step
D
1/1 [==============================] - 0s 55ms/step
E
1/1 [==============================] - 0s 71ms/step
F

```
1/1 [==============================] - 0s 52ms/step
G
1/1 [==============================] - 0s 51ms/step
G
1/1 [==============================] - 0s 45ms/step
I
1/1 [==============================] - 0s 47ms/step
G
1/1 [==============================] - 0s 48ms/step
K
1/1 [==============================] - 0s 45ms/step
L
1/1 [==============================] - 0s 37ms/step
M
1/1 [==============================] - 0s 37ms/step
N
1/1 [==============================] - 0s 35ms/step
O
1/1 [==============================] - 0s 33ms/step
G
1/1 [==============================] - 0s 38ms/step
C
1/1 [==============================] - 0s 38ms/step
R
1/1 [==============================] - 0s 49ms/step
S
1/1 [==============================] - 0s 52ms/step
T
1/1 [==============================] - 0s 46ms/step
U
1/1 [==============================] - 0s 38ms/step
V
1/1 [==============================] - 0s 35ms/step
V
1/1 [==============================] - 0s 39ms/step
X
1/1 [==============================] - 0s 31ms/step
Y
1/1 [==============================] - 0s 36ms/step
Z

2046/2046 [==============================] - 73s 36ms/step - loss: 0.0970 -
accuracy: 0.9674 - val_loss: 0.0994 - val_accuracy: 0.9656
313/313 - 5s - loss: 0.0949 - accuracy: 0.9696 - 5s/epoch - 16ms/step
0.9696000218391418

1/1 [==============================] - 0s 169ms/step
A
1/1 [==============================] - 0s 62ms/step
B
1/1 [==============================] - 0s 62ms/step
C
```

1/1 [==============================] - 0s 54ms/step
D
1/1 [==============================] - 0s 69ms/step
E
1/1 [==============================] - 0s 56ms/step
F
1/1 [==============================] - 0s 54ms/step
G
1/1 [==============================] - 0s 62ms/step
G
1/1 [==============================] - 0s 50ms/step
I
1/1 [==============================] - 0s 53ms/step
G
1/1 [==============================] - 0s 58ms/step
K
1/1 [==============================] - 0s 54ms/step
L
1/1 [==============================] - 0s 51ms/step
M
1/1 [==============================] - 0s 70ms/step
N
1/1 [==============================] - 0s 36ms/step
O
1/1 [==============================] - 0s 38ms/step
P
1/1 [==============================] - 0s 33ms/step
P
1/1 [==============================] - 0s 34ms/step
R
1/1 [==============================] - 0s 61ms/step
T
1/1 [==============================] - 0s 24ms/step
T
1/1 [==============================] - 0s 31ms/step
U
1/1 [==============================] - 0s 33ms/step
K
1/1 [==============================] - 0s 32ms/step
K
1/1 [==============================] - 0s 30ms/step
X
1/1 [==============================] - 0s 30ms/step
Y
1/1 [==============================] - 0s 32ms/step
Z

17. Resultaten captured testdataset

1/1 [==============================] - 0s 145ms/step
Z

1/1 [==============================] - 0s 23ms/step
W
1/1 [==============================] - 0s 20ms/step
Q
1/1 [==============================] - 0s 20ms/step
I
1/1 [==============================] - 0s 20ms/step
D
1/1 [==============================] - 0s 21ms/step
Z
1/1 [==============================] - 0s 20ms/step
G
1/1 [==============================] - 0s 20ms/step
space
1/1 [==============================] - 0s 19ms/step
W
1/1 [==============================] - 0s 21ms/step
W
1/1 [==============================] - 0s 19ms/step
W
1/1 [==============================] - 0s 19ms/step
Z
1/1 [==============================] - 0s 19ms/step
W
1/1 [==============================] - 0s 23ms/step
W
1/1 [==============================] - 0s 20ms/step
D
1/1 [==============================] - 0s 19ms/step
Q
1/1 [==============================] - 0s 20ms/step
Q
1/1 [==============================] - 0s 20ms/step
W
1/1 [==============================] - 0s 20ms/step
B
1/1 [==============================] - 0s 19ms/step
D
1/1 [==============================] - 0s 20ms/step
W
1/1 [==============================] - 0s 19ms/step
W
1/1 [==============================] - 0s 20ms/step
W
1/1 [==============================] - 0s 22ms/step
W
1/1 [==============================] - 0s 52ms/step
W
1/1 [==============================] - 0s 31ms/step
D

18. Resultaten random bewerking

```
Epoch 1/20
2046/2046 [==============================] - 110s 52ms/step - loss: 1.4121 - accuracy: 0.5914
- val_loss: 7.3418 - val_accuracy: 0.3454
Epoch 2/20
2046/2046 [==============================] - 111s 54ms/step - loss: 0.5041 - accuracy: 0.8467
- val_loss: 9.2069 - val_accuracy: 0.3860
Epoch 3/20
2046/2046 [==============================] - 108s 53ms/step - loss: 0.3297 - accuracy: 0.8987
- val_loss: 10.4730 - val_accuracy: 0.3906
Epoch 4/20
2046/2046 [==============================] - 103s 50ms/step - loss: 0.2521 - accuracy: 0.9214
- val_loss: 10.6197 - val_accuracy: 0.4281
Epoch 5/20
2046/2046 [==============================] - 106s 52ms/step - loss: 0.2011 - accuracy: 0.9387
- val_loss: 12.7482 - val_accuracy: 0.4051
Epoch 6/20
2046/2046 [==============================] - 104s 51ms/step - loss: 0.1720 - accuracy: 0.9472
- val_loss: 12.8432 - val_accuracy: 0.4227
Epoch 7/20
2046/2046 [==============================] - 103s 50ms/step - loss: 0.1471 - accuracy: 0.9549
- val_loss: 13.4327 - val_accuracy: 0.4328
Epoch 8/20
2046/2046 [==============================] - 103s 50ms/step - loss: 0.1254 - accuracy: 0.9617
- val_loss: 12.8294 - val_accuracy: 0.4584
Epoch 9/20
2046/2046 [==============================] - 108s 53ms/step - loss: 0.1123 - accuracy: 0.9658
- val_loss: 14.0288 - val_accuracy: 0.4565
Epoch 10/20
2046/2046 [==============================] - 104s 51ms/step - loss: 0.1054 - accuracy: 0.9683
- val_loss: 12.4561 - val_accuracy: 0.4718
Epoch 11/20
2046/2046 [==============================] - 109s 53ms/step - loss: 0.0852 - accuracy: 0.9743
- val_loss: 12.7679 - val_accuracy: 0.4886
Epoch 12/20
2046/2046 [==============================] - 107s 53ms/step - loss: 0.0851 - accuracy: 0.9743
- val_loss: 14.1375 - val_accuracy: 0.4638
Epoch 13/20
2046/2046 [==============================] - 110s 54ms/step - loss: 0.0759 - accuracy: 0.9779
- val_loss: 16.7215 - val_accuracy: 0.4490
Epoch 14/20
2046/2046 [==============================] - 110s 54ms/step - loss: 0.0716 - accuracy: 0.9781
- val_loss: 15.2282 - val_accuracy: 0.4667
Epoch 15/20
2046/2046 [==============================] - 111s 54ms/step - loss: 0.0649 - accuracy: 0.9811
- val_loss: 13.9529 - val_accuracy: 0.4863
Epoch 16/20
2046/2046 [==============================] - 113s 55ms/step - loss: 0.0670 - accuracy: 0.9803
- val_loss: 13.4846 - val_accuracy: 0.4849
Epoch 17/20
```

```
2046/2046 [==============================] - 109s 53ms/step - loss: 0.0532 - accuracy: 0.9843
- val_loss: 14.3203 - val_accuracy: 0.4958
Epoch 18/20
2046/2046 [==============================] - 105s 51ms/step - loss: 0.0566 - accuracy: 0.9829
- val_loss: 15.4290 - val_accuracy: 0.4906
Epoch 19/20
2046/2046 [==============================] - 111s 54ms/step - loss: 0.0479 - accuracy: 0.9859
- val_loss: 16.4972 - val_accuracy: 0.4726
Epoch 20/20
2046/2046 [==============================] - 121s 59ms/step - loss: 0.0499 - accuracy: 0.9854
- val_loss: 15.4557 - val_accuracy: 0.4888
313/313 - 4s - loss: 15.2958 - accuracy: 0.4774 - 4s/epoch - 12ms/step
0.477400004863739
```

19. Resultaten random bewerking, testdataset

```
1/1 [==============================] - 0s 25ms/step
A
1/1 [==============================] - 0s 29ms/step
B
1/1 [==============================] - 0s 24ms/step
C
1/1 [==============================] - 0s 28ms/step
D
1/1 [==============================] - 0s 30ms/step
E
1/1 [==============================] - 0s 24ms/step
F
1/1 [==============================] - 0s 27ms/step
P
1/1 [==============================] - 0s 28ms/step
H
1/1 [==============================] - 0s 23ms/step
D
1/1 [==============================] - 0s 23ms/step
P
1/1 [==============================] - 0s 30ms/step
K
1/1 [==============================] - 0s 29ms/step
L
1/1 [==============================] - 0s 25ms/step
M
1/1 [==============================] - 0s 24ms/step
N
1/1 [==============================] - 0s 24ms/step
O
1/1 [==============================] - 0s 30ms/step
P
1/1 [==============================] - 0s 22ms/step
Q
1/1 [==============================] - 0s 24ms/step
```

L
1/1 [==============================] - 0s 21ms/step
L
1/1 [==============================] - 0s 22ms/step
L
1/1 [==============================] - 0s 21ms/step
U
1/1 [==============================] - 0s 22ms/step
W
1/1 [==============================] - 0s 21ms/step
W
1/1 [==============================] - 0s 21ms/step
L
1/1 [==============================] - 0s 20ms/step
Y
1/1 [==============================] - 0s 25ms/step
Z

20. Resultaten random bewerking, captured dataset

1/1 [==============================] - 0s 36ms/step
A
1/1 [==============================] - 0s 25ms/step
W
1/1 [==============================] - 0s 23ms/step
D
1/1 [==============================] - 0s 30ms/step
F
1/1 [==============================] - 0s 26ms/step
D
1/1 [==============================] - 0s 23ms/step
F
1/1 [==============================] - 0s 25ms/step
G
1/1 [==============================] - 0s 25ms/step
Y
1/1 [==============================] - 0s 28ms/step
W
1/1 [==============================] - 0s 34ms/step
G
1/1 [==============================] - 0s 49ms/step
W
1/1 [==============================] - 0s 24ms/step
D
1/1 [==============================] - 0s 22ms/step
W
1/1 [==============================] - 0s 31ms/step
W
1/1 [==============================] - 0s 27ms/step
D
1/1 [==============================] - 0s 25ms/step

D
1/1 [==============================] - 0s 21ms/step
G
1/1 [==============================] - 0s 20ms/step
W
1/1 [==============================] - 0s 20ms/step
B
1/1 [==============================] - 0s 22ms/step
D
1/1 [==============================] - 0s 23ms/step
W
1/1 [==============================] - 0s 24ms/step
W
1/1 [==============================] - 0s 21ms/step
W
1/1 [==============================] - 0s 26ms/step
W
1/1 [==============================] - 0s 31ms/step
F
1/1 [==============================] - 0s 29ms/step
F