

## Table of Contents

<b>Inleiding</b> .....	2
<b>Achtergrond</b> .....	2
Methode 1: Het CNN .....	2
Methode 2: de SVM .....	2
<b>Hypothese</b> .....	3
<b>Methodes</b> .....	3
Methode 1 .....	3
Methode 2 .....	3
Testen .....	3
<b>Experimenten</b> .....	4
Experiment 1 .....	4
Test 1 .....	4
Test 2 .....	6
Test3 .....	8
Experiment 2 .....	10
Test1 .....	10
Test2 .....	10
Test3 .....	11
<b>Conclusie en discussie</b> .....	12
<b>Literatuurlijst</b> .....	13
<b>Bijlagen</b> .....	13

## Inleiding

De diagnosestelling van hersentumoren is een belangrijk aspect in de geneeskunde, dit mag niet fout gaan, zowel een vals positief als een vals negatief kan ingrijpende gevolgen hebben voor de patiënten en hun omgeving, daarom is het van belang om een arts bij deze diagnose te ondersteunen. Hiervoor kan een tweede arts zijn mening geven, echter kan ook een classificatie algoritme gebruikt worden om te bepalen of de gemaakte scan tumor vrij is, als dit overeenkomt met de bevindingen van de arts is het een stukje extra geruststelling, zowel voor arts als patiënt. Als dit echter niet overeen komt is het een reden voor de arts om een second opinion van een collega te vragen, of zelf nog een keer de scan te bekijken. Het doel van dit onderzoek is het vergelijken van de classificatie met behulp van een SVM en classificatie met een CNN, hierin is mijn verwachting dat het CNN veel beter gaat zijn dan de SVM.

## Achtergrond

### Methode 1: Het CNN

De eerste methode die in dit onderzoek van toepassing is, is het CNN, hiervoor heb ik gekeken naar het onderzoek van Mesut Toğaçar, Burhan Ergen en Zafer Cömert [\[1\]](#) waarin deze een nieuw CNN voorstelt, dit netwerk heeft uiteindelijk een accuraatheid van 96% gehaald, dit is aanzienlijk meer dan de in het paper aangehaalde 78% van een eerdere SVM-methode, mijn verwachting van het zwak zijn van een SVM voor de taak is deels hierop gebaseerd.

Een CNN (convolutional neural network) is een type ANN (artificial neural network) waarbij de input vaak begint met een of meer convolutie lagen, dit zijn lagen die filters toepassen op de input, en daarbij ook “leren” welke convoluties met welke gewichten het beste zijn, om het beeld voor te bereiden voor het classificatie deel van het netwerk. Tussen de convolutional lagen in kunnen eventueel max-pooling lagen worden toegepast, het voordeel hiervan is dat het verschil tussen kleur regio’s duidelijker wordt, echter is het nadeel dat max-pooling de afbeelding verkleint. Na al de voorbewerking komt dan een flatten laag, deze is bedoeld om een multidimensionaal-array om te zetten naar een 1-dimensionaal array. Dan komen de Dense lagen, deze lagen “leren” de gewenste classificatie te doen, door middel van gewichten die tijdens het trainen de ideale waardes approximeren.

### Methode 2: de SVM

De tweede methode die in dit onderzoek van toepassing is het SVM, hiervoor heb ik gekeken naar het onderzoek is van Sarate, G. G en Nagalakkar, V. J. [\[2\]](#) waarin de door hun gebruikte SVM 91% van de positieve gevallen correct identificeert, voor negatieve gevallen is dit 94%. Dit is aanzienlijk hoger dan de op SVM gebaseerde vergelijkingsmethode van het eerste paper [\[1\]](#).

Een SVM (support vector machine) is een binaire classificatie methode, bij het trainen krijgt deze constant data, van deze maakt het vectoren die geplaatst worden in een grafiek, als alle train data is geplott, trekt het machine een lijn gebaseerd op positie van de vectoren, met hun klassen, de bedoeling is dat er aan beide kanten van de lijn diverse voorbeelden zijn, de lijn representeert de scheiding van klassen, voor nieuwe input wordt op basis van aan welke kant van de lijn de vector valt bepaalt, welke klasse het is.

## Hypothese

Mijn hypothese is dat het CNN beter zal scoren dan het SVM

## Methodes

### Methode 1

Voor mijn eerste methode gebruik ik een CNN deze wordt geïmplementeerd met tensorflow, en een dataset van 3000 afbeeldingen, deze set is exact half om half verdeeld in positief en negatief. Het model begint altijd met een Rescaling laag, dit wordt gedaan om ervoor te zorgen dat alle images wat compacter zijn, zodat het netwerk sneller zijn werk kan doen, vervolgens volgt een convolutionele laag, gevolgd door een max-pooling laag. En nog twee convolutionele lagen. Hierna volgt een flatten laag. Dit is altijd de opzet, met de hoeveelheid dense lagen en hun parameters, in dit geval de activatie wordt geëxperimenteerd. De laatste Dense laag van het model heeft altijd 2 outputs, dit moet namelijk gelijk aan het aantal klassen zijn. Het model wordt gecompileerd met de optimizer "adam" en de loss functie "catagorical crossentropy".

### Methode 2

Voor mijn tweede methode gebruik ik een SVM deze wordt geïmplementeerd met scikit-learn, en een dataset van 3000 afbeeldingen, deze set is exact half om half verdeeld in positief en negatief. Het SVM is de SVC variant van scikit-learn.

### Testen

Voor het testen gebruik ik 200 afbeeldingen die het netwerk nooit eerder gezien heeft, dit staat volledig los van de validatie data die steeds een willekeurige 20% van de training afbeeldingen bedraagt, zowel de test als validatie afbeeldingen worden natuurlijk niet gebruikt als trainingsdata.

## Experimenten

Voor het onderzoek ga ik per methode een aantal test doen, de te meten factor is de accuraatheid en de loss, van deze laatste is het belangrijk dat de waarde zo laag mogelijk is, maar in elk geval lager dan 1.5, anders is het netwerk niet beter dan een muntje opgooien, zelfs al heeft het dan veel goed, komt dat vooral omdat het gokt niet omdat het heeft geleerd.

### Experiment 1

#### Test 1

In onderstaande afbeelding is de opzet van het model in de eerste test te zien, de eerder besproken lagen die gelijk blijven en daarbij de lagen die steeds zullen veranderen, een laag met 128 outputs en softmax activatie, gevolgd door een laag met 2 outputs en de default activatie

```
1 model = tf.keras.Sequential([
2     tf.keras.layers.Rescaling(1./255),
3     tf.keras.layers.Conv2D(32, 3, activation='relu'),
4     tf.keras.layers.MaxPooling2D(2),
5     tf.keras.layers.Conv2D(32, 3, activation='relu'),
6     tf.keras.layers.Conv2D(32, 3, activation='relu'),
7     tf.keras.layers.Flatten(),
8     tf.keras.layers.Dense(128, activation='softmax'),
9     tf.keras.layers.Dense(2)
10 ])
11
12 model.compile(
13     optimizer='adam',
14     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
15     metrics=['accuracy'])
```

[3] ✓ 0.6s Python

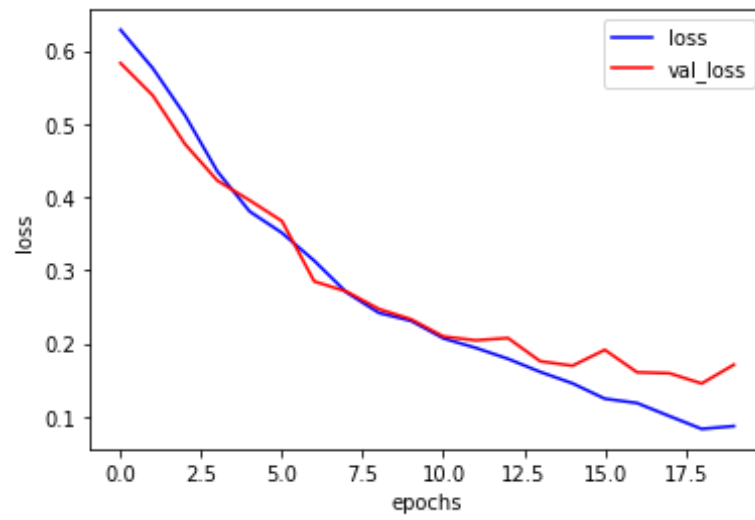
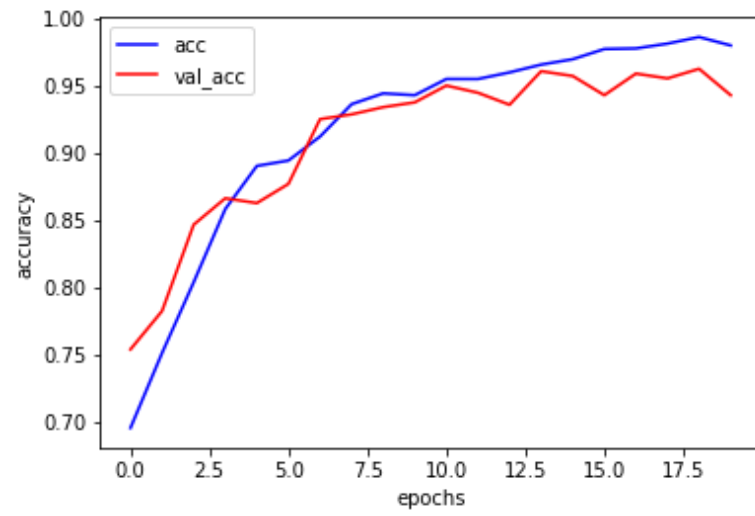
Zoals te zien in onderstaande afbeelding, haalt het model in deze configuratie een accuraatheid van 92,5%, dit resultaat komt na 20 epochs trainen op de test data.

```
1 test_dir = "..\\test"
2 test_ds = tf.keras.utils.image_dataset_from_directory(
3     test_dir,
4     seed=486235864,
5     image_size=(28, 32),
6     batch_size=1)
7
8 test_loss, test_acc = model.evaluate(test_ds, verbose=2)
9
10
```

54] ✓ 0.5s Python

.. Found 200 files belonging to 2 classes.  
200/200 - 0s - loss: 0.2621 - accuracy: 0.9250 - 445ms/epoch - 2ms/step

In onderstaande grafiek is het verloop van de training te zien te zien is dat de val\_acc bij 18 epochs gepiekt heeft, hieruit is te concluderen dat na 18 epochs overfitting voorkomt bij dit model, dit wordt ondersteund door de loss-grafiek die ook bei 18 epochs een toename van de loss toont, voor verder onderzoek zou men kunnen kijken naar wat er gebeurt als er veel meer epochs gedaan worden, het zou kunnen dat er na nog meer trainen nog een stijging voorkomt die de accuraatheid boven het maximum van 95% bij 18 epochs brengt.



## Test 2

Voor de tweede test heb ik bijna hetzelfde gedaan als voor de eerste, alleen heeft de dense laag nu een relu activatie in plaats van softmax

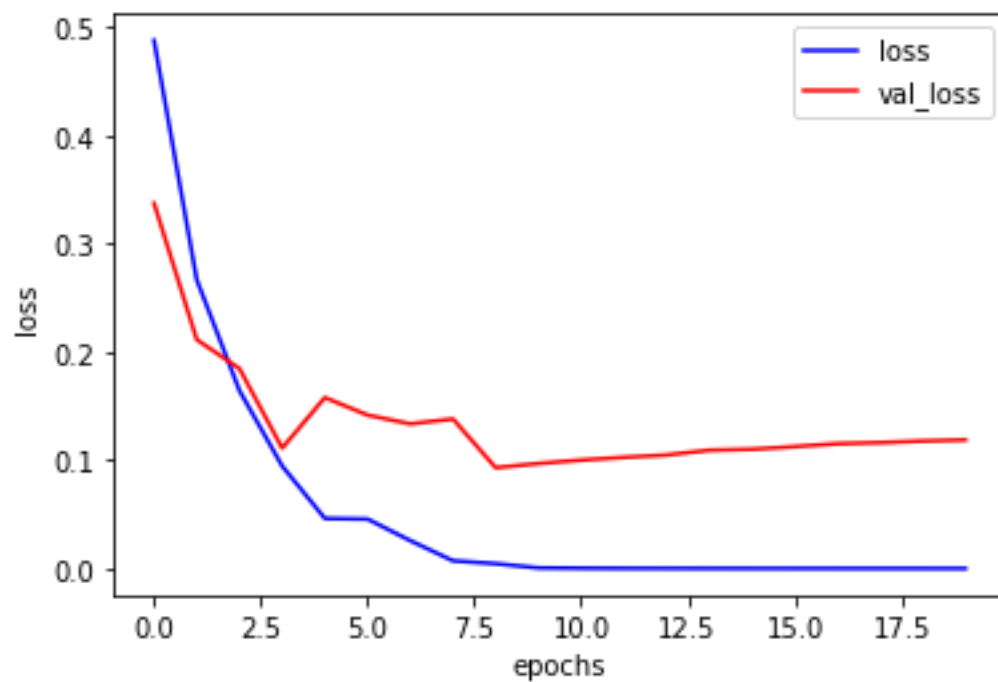
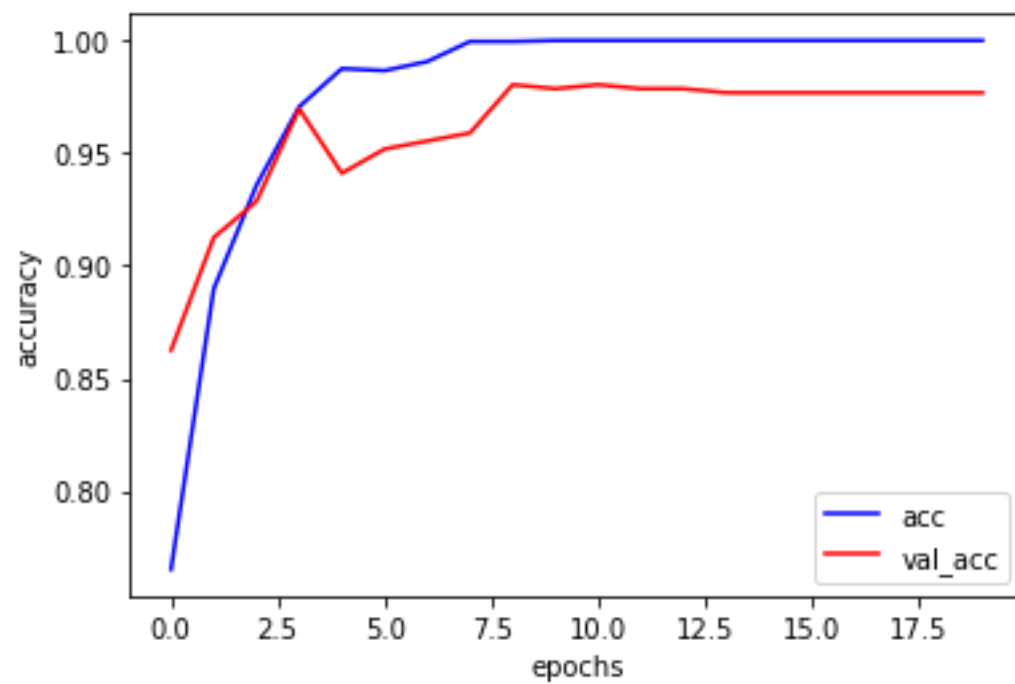
```
1 model = tf.keras.Sequential([
2     tf.keras.layers.Rescaling(1./255),
3     tf.keras.layers.Conv2D(32, 3, activation='relu'),
4     tf.keras.layers.MaxPooling2D(2),
5     tf.keras.layers.Conv2D(32, 3, activation='relu'),
6     tf.keras.layers.Conv2D(32, 3, activation='relu'),
7     tf.keras.layers.Flatten(),
8     tf.keras.layers.Dense(128, activation='relu'),
9     tf.keras.layers.Dense(2)
10 ])
11
12 model.compile(
13     optimizer='adam',
14     loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
15     metrics=['accuracy'])
[163] ✓ 0.1s Python
```

Op deze wijze is er een accuraatheid van 94% bereikt hoger dan de vorige test, met softmax.

```
1 test_dir = "..\\test"
2 test_ds = tf.keras.utils.image_dataset_from_directory(
3     test_dir,
4     seed=486235864,
5     image_size=(28, 32),
6     batch_size=1)
7
8 test_loss, test_acc = model.evaluate(test_ds, verbose=2)
9
10
[174] ✓ 1.4s Python

... Found 200 files belonging to 2 classes.
200/200 - 1s - loss: 0.6688 - accuracy: 0.9400 - 1s/epoch - 5ms/step
```

Ook is in onderstaande grafieken te zien dat de loss weliswaar iets hoger is, maar beide de loss en accuraatheid zijn stabiel tegen het einde van het trainen, ze gaan niet zo erg op en neer als bij de softmax.



### Test3

De derde en laatste test is gedaan met een extra dense laag, met 64 outputs en een relu activatie.

```
1 model = tf.keras.Sequential([
2     tf.keras.layers.Rescaling(1./255),
3     tf.keras.layers.Conv2D(32, 3, activation='relu'),
4     tf.keras.layers.MaxPooling2D(2),
5     tf.keras.layers.Conv2D(32, 3, activation='relu'),
6     tf.keras.layers.Conv2D(32, 3, activation='relu'),
7     tf.keras.layers.Flatten(),
8     tf.keras.layers.Dense(128, activation='relu'),
9     tf.keras.layers.Dense(64, activation='relu'),
10    tf.keras.layers.Dense(2)
11 ])
12
13 model.compile(
14     optimizer='adam',
15     loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
16     metrics=['accuracy'])
```

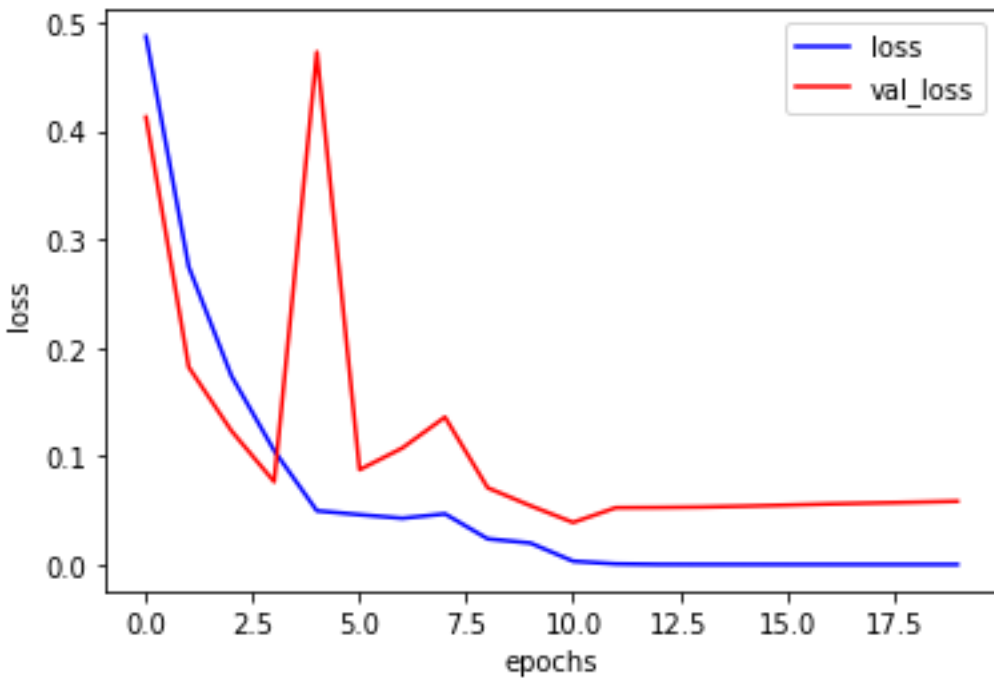
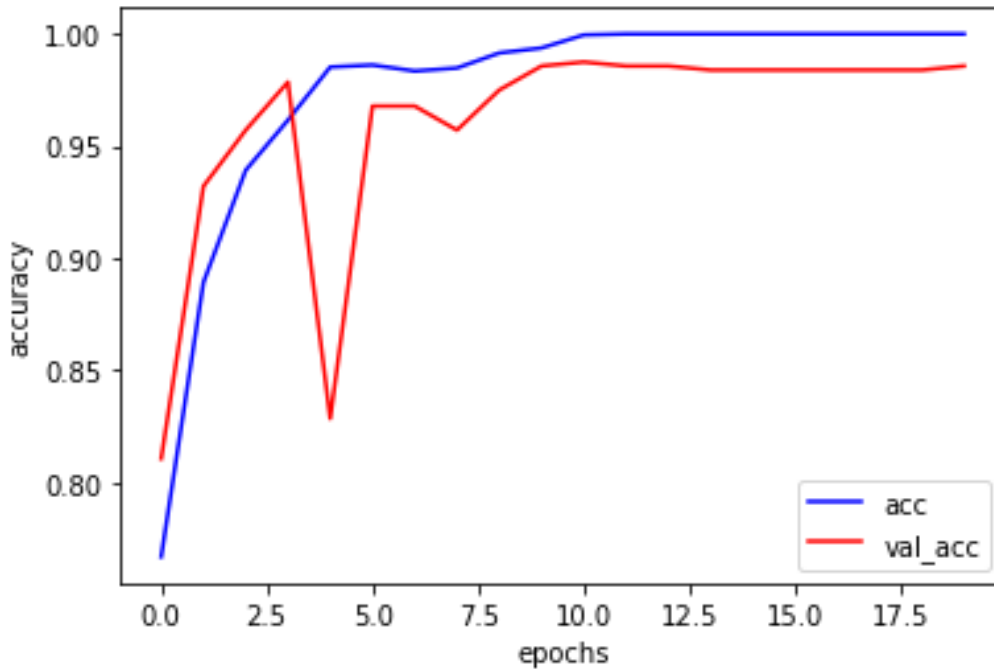
Concrete veranderd er vrij weinig met de toevoeging van een nieuwe laag de resultaten zijn precies hetzelfde.

```
1 test_dir = "..\\test"
2 test_ds = tf.keras.utils.image_dataset_from_directory(
3     test_dir,
4     seed=486235864,
5     image_size=(28, 32),
6     batch_size=1)
7
8 test_loss, test_acc = model.evaluate(test_ds, verbose=2)
9
10
[174] ✓ 1.4s Python

... Found 200 files belonging to 2 classes.
200/200 - 1s - loss: 0.6688 - accuracy: 0.9400 - 1s/epoch - 5ms/step
```

In de grafiek hieronder is af te leiden dat het resultaat weliswaar hetzelfde is maar wel degelijk anders tot stand is gekomen





Uit de bovenstaande resultaten blijkt dat voor het classificeren van hersentumoren een extra laag niet nuttig is en alleen tijd en onnodige gewichten(die weer fout geleerd zouden kunnen worden) toevoegd.

Verder blijkt dat met mijn opzet, maximaal 94% accuraatheid haalbaar is, voor vervolg onderzoek is het interessant om het model verder aan te passen, gedacht kan ook worden aan een niet-sequentieel model.

## Experiment 2

Voor het tweede experiment ga ik kijken naar de SVM, hiervoor is de sci-kit image SVC variant gebruikt, hierin is gespeeld met de waarde van C en gamma, om een zo goed mogelijk resultaat te vinden, de hypothese is dat dit model niet beter zal kunnen voorspellen welke foto's tumoren bevatten dan een CNN

### Test1

Hieronder is de eerste configuratie die ik gebruikt heb te zien, een gamma van 0.001 met een C van 100

```
1 from sklearn import metrics
2 Svm = svm.SVC(gamma= 0.001,C=100)
3 Svm.fit(x,y)
4
5 yespath = '..\\test\\yes'
6 yesfiles = [f for f in listdir(yespath) if isfile(join(yespath, f))]]
7
8 x_test = []
```

Uit dit model komt een accuraatheid van 89%, dit is meteen 3,5% lager dan de laagste score van het CNN.

```
31
32 prediction = Svm.predict(x_test)
33 evaluation = metrics.accuracy_score(y_test,prediction)
34 print(evaluation)
35
36
37
38
39
[34] ✓ 9.3s Python
... 0.89
```

### Test2

Onderstaand de volgende poging nu met een gamma van 0.002

```
1 from sklearn import metrics
2 Svm = svm.SVC(gamma= 0.002,C=100)
3 Svm.fit(x,y)
4
5 yespath = '..\\test\\yes'
6 yesfiles = [f for f in listdir(yespath) if isfile(join(yespath, f))]]
7
```

Duidelijk te zien is een score van 92,5% gelijk met de slechtste score van het SVM

```
32 prediction = Svm.predict(x_test)
33 evaluation = metrics.accuracy_score(y_test,prediction)
34 print(evaluation)
35
36
37
38
39
[34] ✓ 9.9s Python
... 0.925
```

### Test3

Nu met een gamma van 1

```
1 from sklearn import metrics
2 Svm = svm.SVC(gamma= 1 ,C=100)
3 Svm.fit(x,y)
4
5 yespath = '..\\test\\yes'
6 yesfiles = [f for f in listdir(yespath) if isfile(join(yespath, f))]
```

En wow! 95,5% dat is 1,5% hoger dan de CNN

```
32 prediction = Svm.predict(x_test)
33 evaluation = metrics.accuracy_score(y_test,prediction)
34 print(evaluation)
35
36
37
38
39
```

51] ✓ 12.6s

Python

0.955

## Conclusie en discussie

Tot slot, volgend uit voorgaand onderzoek was mijn hypothese dat het CNN beter zou werken, echter is er gebleken dat er het SVM beter werkt dan de CNN, dus mijn hypothese lijkt fout te zijn. Echter is het CNN uit het voorgaande onderzoek een stuk uitgebreider dan mijn CNN, bovendien is dit een non-sequentieel netwerk, dit kan deels het verschil uitleggen. Verder is mijn onderzoek niet volledig sluitend, er zijn meer mogelijke lagen en waardes te proberen, en het CNN kan verder uitgebreid worden, dit raad ik dan ook aan voor vervolg onderzoek.

Mijn onderzoek en de voorgaande studies lijken elkaar dus tegen te spreken, hiervoor is aan te raden bij vervolg onderzoek met een non-sequentieel CNN te werken, ook kan aan de SVM kant gekeken worden naar een uitgebreider of zelfgemaakte SVM

## Literatuurlijst

[1]

Mesut Toğaçar, Burhan Ergen, Zafer Cömert,

BrainMRNet: Brain tumor detection using magnetic resonance images with a novel convolutional neural network model,

Medical Hypotheses,

Volume 134,

2020,

109531,

ISSN 0306-9877,

<https://doi.org/10.1016/j.mehy.2019.109531>.

(<https://www.sciencedirect.com/science/article/pii/S0306987719313416>)

Keywords: Biomedical signal processing; Attention module; Magnetic resonance image; Hypercolumn technique; Brain tumor

[2]

Sarate, G. G., & Nagalakkar, V. J. (2019). Brain Tumor Detection and Identification using

Support Vector Machine. *International research journal of engineering and technology*,

06(12). <https://www.irjet.net/archives/V6/i12/IRJET-V6I12349.pdf>

## Bijlagen

De student kan vision-algoritmes implementeren ten behoeve van een geselecteerde Vision taak.

Zoals te vinden in [github](#) zijn er 2 .ipynb files met daarin twee verschillende gebruikte vision algoritmes

De student kan papers uit de wetenschappelijke literatuur over computer vision lezen, begrijpen, en omzetten naar code

Zoals te vinden in de notebooks is de code van commentaar voorzien, dit verwijst ook terug naar de literatuur, waar van toepassing

De student kan een opdracht analyseren en op basis hiervan het benodigde werk (waaronder het implementeren) verdelen en plannen.

De planning zoals door mij gemaakt omvat de door mij ingeschatte taken, en hun MoScow prioriteit, naast een week planning.

De student kan omgaan met software oplossingen voor versiebeheer (GIT) en kan deze software inzetten voor het ontwikkelen van code.


Github is gebruikt en toegang is verleend aan docent en student-assistent

De student kan een experiment opzetten en uitvoeren om aan te tonen hoe de geïmplementeerde vision-methode werkt, hoe effectief (zowel in kwaliteit als in snelheid) deze methode en implementatie is en waar grenzen liggen van de gekozen methode en implementatie

In beide methodes is een evaluatie functie gebruikt om de werking te achterhalen, in dit document, staan de verschillende geprobeerde waardes en de bijbehorende resultaten

De student kan een rapport schrijven om het gekozen en uitgevoerde experiment te beschrijven, de resultaten te beschrijven en conclusies te trekken

In dit rapport zijn experimenten uitgewerkt, alsmede uit de resultaten, conclusies en aanbevelingen gedaan



De student kan een zelf-opgezet experiment uitvoeren en de resultaten op gestructureerde wijze verzamelen.

De opdracht is zelf gekozen en zelfstanding uitgevoerd, steeds zijn schermopnames van relevante data gemaakt