Algebraic Representation

Geschreven door: Matthijs Koelewijn

Opdrachten

Opdracht A

De opdracht bestaat uit drie delen: A1, A2 en A3

Opdracht A1:

"Voeg ALLEEN Dense en/of Dropout layers toe aan het onderstaande model.

Probeer een zo eenvoudig mogelijk model te vinden(dus met zo min mogelijk parameters) dat een test accurracy oplevert van tenminste 0,95.

Beschrijf daarbij met welke stappen je bent gekomen tot je model, en beargumenteer elk van je stappen."

Het volgende model is geschreven:

```
def buildMyModel(inputShape):
    model = keras.Sequential(
        [
            keras.Input(shape=inputShape),
            layers.Dropout(.1),
            layers.Dense(units=200, activation='sigmoid'),
            layers.Dropout(.1),
            layers.Dense(units=100, activation='sigmoid'),
            layers.Dropout(.1),
            layers.Dense(units=10, activation='sigmoid'),
        ]
    )
    return model
```

Voor opdracht A1 heb ik het model aangepast om een zo eenvoudig mogelijk model te creëren dat een testnauwkeurigheid van minstens 0,95 kan behalen. Ik heb de volgende aanpassingen gemaakt:

1. De lagen worden sequentieel toegevoegd.

2. De eerste laag krijgt de vorm van `inputShape` als parameter. Dit definieert de inputlaag van het model, die overeenkomt met de vorm van de invoergegevens. Deze `inputShape` heeft een aantal input neurons van 28x28x1 = 784, dit is de algemene MNIST shape.

3. Een dropout-laag wordt toegevoegd met een dropout-waarde van 0.1. Dropout is een techniek die helpt overfitting te voorkomen door willekeurig een deel van de neuronen uit te schakelen tijdens

het trainen. Een dropout-waarde van 0.1 betekent dat 10% van de neuronen tijdens het trainen wordt uitgeschakeld. Dit leek een goede waarde te zijn voor een shape van 28x28x1.

4. Een dense-laag wordt toegevoegd met 200 neuronen en een sigmoid-activatiefunctie. Deze laag fungeert als een volledig verbonden laag waarin elke input wordt verbonden met elk neuron in de laag. De sigmoid-activatiefunctie wordt gebruikt om niet-lineaire activatie te introduceren in het model.

5. Opnieuw wordt een dropout-laag toegevoegd met een dropout-waarde van 0.1 om overfitting te helpen voorkomen.

6. Een dense-laag met 100 neuronen en een sigmoid-activatiefunctie wordt toegevoegd.

7. Opnieuw wordt een dropout-laag toegevoegd met een dropout-waarde van 0.1.

8. Ten slotte wordt de laatste dense-laag toegevoegd met 10 neuronen (overeenkomend met het aantal klassen in de MNIST-dataset) en een sigmoid-activatiefunctie.

Met deze stappen heb ik een eenvoudig model gemaakt dat een redelijke test accuracy behaalt van 0.95.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dropout (Dropout)           (None, 784)               0

 dense (Dense)               (None, 200)               157000

 dropout_1 (Dropout)         (None, 200)               0

 dense_1 (Dense)             (None, 100)               20100

 dropout_2 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 178,110
Trainable params: 178,110
Non-trainable params: 0
_________________________________________________________________
```

Opdracht A2:

“Verklaar met kleine berekening het aantal parameters dat bij elke laag staat genoemd in bovenstaande model summary.“

De model summary toont het aantal parameters voor elke laag. We kunnen het aantal parameters voor een Dense-laag berekenen door de formule: (aantal_input_neurons * aantal_output_neurons) + aantal_output_neurons.

MNIST Shape = 28 x 28 x 1, dus aantal input neurons van dense = 28 x 28 x 1 = 784

Hier zijn de berekeningen voor de lagen in het gegeven model:

- Dense: (784 * 200) + 200 = 157.000 parameters

- Dense_1: (200 * 100) + 100 = 20.100 parameters

- Dense_2: (100 * 10) + 10 = 1.010 parameters

In totaal heeft het gegeven model 178.110 parameters. Het aantal parameters in een model bepaalt de capaciteit en complexiteit ervan. Door het aantal parameters te beperken, zoals in dit geval, kunnen we een eenvoudiger model krijgen dat minder gevoelig is voor overfitting en beter generaliseert.

Opdracht A3:
"Leg uit wat de betekenis is van de output images die bovenstaand genenereerd worden met "printFeatureMapsForLayer". "

De outputbeelden gegenereerd door de functie `printFeatureMapsForLayer` laten zien hoe de activaties van de neuronen in de gekozen laag reageren op de invoerbeelden. Elk beeld in de gegenereerde reeks toont de activaties van een specifiek neuron in de laag voor verschillende invoerbeelden. Dit geeft ons inzicht in de representatie van de invoergegevens op dat specifieke niveau.

```
ViewTools_NN.printFeatureMapsForLayer(nLayer, model, x_test_flat, x_test, 0, baseFilenameForSave)
```

In bovenstaande aanroep code geven we **`nLayer` , `model`, `x_test_flat`, `x_test`, 0, `baseFilenameForSave`**

`nLayer` , de index van de gewenste laag

`model`, het model

`x_test_flat`, `x_test`, allebei test gegevens

0, sample index, oftewel de index van het voorbeeld in de testgegevens dat we willen visualiseren.

`baseFilenameForSave` de filenaam waarin we de beelden opslaan.

"Leg uit wat de betekenis zou zijn van wat je ziet als je aan die functie de index van de eerste dense layer meegeeft."

Als we de index van de eerste Dense-laag aan de functie "printFeatureMapsForLayer" zouden geven, zouden we geen betekenisvolle outputbeelden krijgen. De eerste Dense-laag voert een lineaire transformatie uit op de invoer en heeft geen ingebouwde niet-lineariteit.

Hierdoor zouden de activaties van de neuronen in de laag niet veel verschillen tonen en zouden de outputbeelden mogelijk niet informatief zijn.

Door hier naar te kijken, kunnen we een gevoel krijgen over welke representaties en kenmerken het model heeft geleerd te herkennen. Dit gevoel kunnen we gebruiken voor het verbeteren van het model.

"Leg uit wat de betekenis zou zijn van wat je ziet als je aan die functie de index van de tweede dense layer meegeeft."

Als we de index van de tweede Dense-laag aan de functie "printFeatureMapsForLayer" zouden geven, zouden we de activaties van de neuronen in die laag kunnen visualiseren.

Aangezien de tweede Dense-laag non-lineariteit introduceert door de sigmoid-activatiefunctie, zouden de outputbeelden ons laten zien hoe de neuronen in die laag verschillende patronen in de invoergegevens detecteren.

Net als bij de vorige vraag kunnen we hier ook een gevoel krijgen van hoe goed het model classificeert.

Opdracht B

“De opdracht bestaat uit drie delen: B1, B2 en B3”

Opdracht B1

“ Voeg als hidden layers ALLEEN Convolution en/of Dropout layers toe aan het onderstaande model. Probeer een zo eenvoudig mogelijk model te vinden (dus met zo min mogelijk parameters) dat een test accurracy oplevert van tenminste 0,97.

Beschrijf daarbij met welke stappen je bent gekomen tot je model, je ervaringen daarbij en probeer die ervaringen te verklaren.

“

Het volgende model is geschreven:

```
def buildMyModel(inputShape):
    model = keras.Sequential(
        [
            keras.Input(shape=inputShape),

            # Convolutional layers
            layers.Conv2D(32, kernel_size=(3, 3), padding='same',
activation='relu'),
            layers.Dropout(0.25),

            layers.Conv2D(64, kernel_size=(3, 3), padding='same',
activation='relu'),
            layers.Dropout(0.25),

            # MaxPooling layer
            layers.MaxPooling2D(pool_size=(2, 2)),

            # Flatten the output
            layers.Flatten(),

            # Output layer
            layers.Dense(units=num_classes, activation='sigmoid')
        ]
    )
    return model
```

In deze aangepaste model heb ik de MaxPooling2D-laag verplaatst naar na de Conv2D-laag. Ik heb ook het “valid” padding-type vervangen door “same”padding, zodat de ruimtelijke dimensies van de feature maps behouden blijven.

Alleen weet ik niet zeker of de pooling-laag(`layers.MaxPooling2D`) is toegestaan volgens de opdracht.

We kunnen namelijk ook deze laag verwijderen en de convolutielagen op elkaar stapelen.

De stappen die zijn genomen:

1. Invoerlaag (keras.Input) inputshape geeft de vorm van de MNIST-beelden
2. Conv2D-laag voegen we toe met 32 filters en een kernelgrootte van 3x3. We gebruiken ‘same’padding om ervoor te zorgen dat de ruimtelijke dimensies van de feature maps behouden blijven. De activatiefunctie is ‘relu’, die helpt bij het introduceren van niet-lineariteit in het model.
3. Vervolgens voegen we een Dropout-laag toe. Met een dropout percentage van 0.25. Dit helpt om overfitting te voorkomen door willekeurig een deel van de verbindingen tussen neuronen uit te schakelen tijdens het trainen.
4. Daarna voegen we een Conv2D-laag toe met 64 filters en een kernelgrootte van 3x3. We gebruiken weer ‘same’ padding en ‘relu’ als activatiefunctie. Dit helpt bij het extracten van meer complexe kenmerken uit de inputbeelden.
5. We voegen nog een dropout-laag toe van 0.25 om verder te reguleren op het extracten van de kenmerken.
6. We laten de MaxPooling2D-laag staan, die al in het orginele model zat. Deze heeft een poolgrootte van 2x2. Deze laag voert een downsampling uit op de feature maps door de hoogste activatiewaarde in elk 2x2 gebied te selecteren. Dit helpt bij het verminden van de ruimtelijke dimensies.
7. Ook de Flatten-laag laten we staan. Dit is nodig om de feature maps te vormen tot een 1D-vector, zodat het geschikt is voor de Dense laag.
8. Tot slot is de Dense-laag(`layers.Dense`) met num_classes en een sigmoid-activatiefunctie. Deze laag geeft de uiteindelijke classificatie-uitvoer voor elk van de 10 klassen van de MNIST dataset.

Het model kan verder worden geoptimaliseerd door het toevoegen van meer convolutie- of dropout-lagen, maar voor dit voorbeeld heb ik geprobeerd een eenvoudig model te vinden dat nog steeds een goede prestatie levert.

Binnen 6 epochs is de nauwkeurigheid van 0.97 bereikt.

Opdracht B2 en B3

"Kopieer bovenstaande model summary en verklaar bij bovenstaande model summary bij elke laag met kleine berekeningen de Output Shape" en "Verklaar nu bij elke laag met kleine berekeningen het aantal parameters."

Gegeven is de volgende model summary :

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 dropout (Dropout)           (None, 28, 28, 32)        0

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        18496

 dropout_1 (Dropout)         (None, 28, 28, 64)        0

 max_pooling2d (MaxPooling2D  (None, 14, 14, 64)       0
 )

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 10)                125450

=================================================================
Total params: 144,266
Trainable params: 144,266
Non-trainable params: 0
```

Formule aantal parameters conv2d: (Aantal filters x kernelgrootte x inputkanaal) + bias

Formule aantal parameters dense: (Aantal input neuronen x aantal output neuronen) + aantal output neuronen

conv2d:

Input Shape: (None, 28, 28, 1) afkomstig van de invoerlaag

Output Shape: (None, 28, 28, 32)

Berekening: conv2d past 32 filters toe op elke input van 3x3, wat

resulteert in een output feature map van 28x28 met 32 filters.

Het aantal parameters voor deze laag is:

(32 filters x (3x3) kernelgrootte x 1 inputkanaal) + 32 bias = 320 parameters

(32 x 9 x 1) + 32 = 320 parameters

dropout:

De Dropout-laag voert geen wijzigingen toe aan de vorm van de feature maps.

Het schakelt wel de 25% van de elementen uit, wat resulteert in een uitgedunde representatie.

conv2d_1:

Input Shape: (None, 28, 28, 32)

Output Shape: (None, 28, 28, 64)

Berekening: conv2d_1 past 64 filters toe op elke input van 3x3, wat

resulteert in een output feature map van 28x28 met 64 filters.

Het aantal parameters voor deze laag is:

(64 filters x (3x3) kernelgrootte x 32 inputkanaal) + 64 bias = 18.496 parameters

dropout_1:

Wederom voert de Dropout-laag geen wijzigingen toe aan de vorm van de feature maps.

Het schakelt wel de 25% van de elementen uit, wat resulteert in een uitgedunde representatie.

max_pooling2d:

Input Shape: (None, 28, 28, 64)

Output Shape: (None, 14, 14, 64)

Berekening: max_pooling2d verdeelt de ruimtelijke dimensies van de feature maps

door de opgegeven poolgrootte (2x2). Hierdoor worden de dimensies gehalveerd (28/2 = 14)

en blijven de 64 kanalen behouden. Dit leidt tot een halvering van het aantal pixels

en helpt bij het verhogen van de ruimtelijke invariantie.

flatten:

Input Shape: (None, 14, 14, 64)

Output Shape: (None, 12544)

Berekening: De flatten laag herstructureert de input feature maps naar een 1D-vector.

In dit geval wordt de dimensie 14x14x64 omgezet naar een vector van lengte 12.544.

14x14x64 = 12.544

dense:

Input Shape: (None, 12544)

Output Shape: (None, 10)

Berekening: De Dense-laag bevat 10 neuronen, wat overeenkomt met het aantal klassen in de dataset(0-9).

Het aantal parameters voor deze laag is 12544 inputkenmerken x 10 neuronen + 10 bias termen = 125.450

Totale aantal parameters = 320 + 18.496 + 125.450 = 144.266

Opdracht C

“Bij deze opdracht gebruik je geen dense layer meer. De output is nu niet meer een vector van 10, maar een plaatje van 13 pixel groot en 10 lagen diep.

Deze opdracht bestaat uit vier delen: C1, C2, C3 en C4”

Opdracht C1

“Voeg ALLEEN Convolution en/of MaxPooling2D layers toe aan het onderstaande model. (dus GEEN dense layers, ook niet voor de output layer) Probeer een zo eenvoudig mogelijk model te vinden (dus met zo min mogelijk parameters) dat een test accurracy oplevert van tenminste 0.98.

Beschrijf daarbij met welke stappen je bent gekomen tot je model, en beargumenteer elk van je stappen.

”

Het volgende model is geschreven, na 80 epochs behaalt het een accurracy van 0.98:

```
def buildMyModel(inputShape):
    model = keras.Sequential(
        [
            keras.Input(shape=inputShape),
            layers.Conv2D(20, kernel_size=(5, 5)),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(20, kernel_size=(3, 3)),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(num_classes, kernel_size=(5, 5))
        ]
    )
    return model
```

Dit model is een goede keuze omdat het eenvoudig is, met een klein aantal parameters, en toch een redelijk hoge testnauwkeurigheid kan behalen.
Convolutionele lagen: De eerste convolutionele laag met 20 filters en een kernelgrootte van 5x5 kan de basiskenmerken leren, terwijl de tweede convolutielaag met 20 filters en een kleinere kernel grootte van 3x3 verfijndere kenmerken kan leren. Dit helpt om onnodige details te verminderen.

MaxPooling lagen: Dit helpt om de feature maps te verkleinen, waardoor de berekeningslast wordt verminderd. Door de poolgrootte van 2x2 toe te passen na elke convolutielaag, wordt de ruimtelijke dimensie gehalveerd, waardoor het model overzichtelijker wordt.

Aantal filters: Het gebruik van 20 filters in de convolutielagen helpt bij het leren van verschillende, 20 filters waren net genoeg om de gewenste kenmerken te leren zonder te veel complexiteit toe te voegen. Ook heb ik 32 filters geprobeerd, maar dit gaf een grote hoeveelheid parameters, wat niet gewenst is.

Opdracht C2, C3 en C4

“

Kopieer bovenstaande model summary en verklaar bij bovenstaande model summary bij elke laag met kleine berekeningen de Output Shape“

“Verklaar nu bij elke laag met kleine berekeningen het aantal parameters.“

*“Bij elke conv layer hoort een aantal elementaire operaties (+ en *).*

** Geef PER CONV LAYER een berekening van het totaal aantal operaties dat nodig is voor het klassificeren van 1 test-sample.*

** Op welk aantal operaties kom je uit voor alle conv layers samen?”*

Model summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 20)        520

max_pooling2d (MaxPooling2D  (None, 12, 12, 20)       0
)

conv2d_1 (Conv2D)            (None, 10, 10, 20)        3620

max_pooling2d_1 (MaxPooling  (None, 5, 5, 20)         0
2D)

conv2d_2 (Conv2D)            (None, 1, 1, 10)          5010

=================================================================
Total params: 9,150
Trainable params: 9,150
Non-trainable params: 0
```

conv2d: (20 filters, kernel size: 5x5):

- Input shape: (None, 28, 28, 1)
- Output shape: (None, 24, 24, 20)
- Parameters: (20 filters * (5*5) kernel size * 1 inputkanaal) + 20 bias = 520
- Operaties per filter: 5x5x1 = 25
- Operaties per output pixel: 25x20 = 500
- Totaal aantal operaties: 24x24x500 = 288.000

Verklaring: conv2d gebruikt 20 filters met een kernelgrootte van 5x5.

Het aantal parameters wordt bepaald door de grootte van de kernel en het aantal filters.

max_pooling2d: (pool size: 2x2):

- Input shape: (None, 24, 24, 20)
- Output shape: (None, 12, 12, 20)

- Parameters: 0

Verklaring: max_pooling2d verkleint de ruimtelijke dimensies door de maximale waarde binnen elk 2x2-gebied te selecteren.

Er zijn geen parameters omdat het een niet-trainbare laag is. Geen extra operaties uitgevoerd

conv2d_1 (20 filters, kernel size: 3x3):

- Input shape: (None, 12, 12, 20)

- Output shape: (None, 10, 10, 20)

- Parameters: (20 filters * (3x3) kernel_size * 20 inputkanalen) + 20 bias = 3.620

- Operaties per filter: 3x3x20 = 180

- Operaties per output pixel: 180x20 = 3.600

- Totaal aantal operaties: 10x10x3600 = 360.000

Verklaring: conv2d_1 gebruikt opnieuw 20 filters met een kernelgrootte van 3x3.

Het aantal parameters wordt bepaald door de grootte van de kernel en het aantal filters.

max_pooling2d_1 (pool size: 3x3):

- Input shape: (None, 10, 10, 20)

- Output shape: (None, 5, 5, 20)

- Parameters: 0

Verklaring: Opnieuw wordt de ruimtelijke dimensie verkleind door max pooling toe te passen met een poolgrootte van 3x3.

Er zijn geen parameters in deze laag. Geen extra operaties uitgevoerd

conv2d_2 (20 filters, kernel size: 5x5):

- Input shape: (None, 5, 5, 20)

- Output shape: (None, 1, 1, 10)

- Parameters: (10 filters * (5x5) kernel_size * 20 inputkanalen) + 10 bias = 5.010

- Operaties per filter: 5x5x20 = 500

- Operaties per output pixel: 500x10 = 5.000

- Totaal aantal operaties: 1x1x5.000 = 5.000

Verklaring: De laatste conv2d_2 heeft 10 filters met een kernelgrootte van 5x5.

Het aantal parameters wordt bepaald door de grootte van de kernel en het aantal filters.

Totaal aantal parameters: 9,150

Totaal aantal operaties: 288.000 + 0 + 360.000 + 0 + 5.000 = 653.000

Dus het totale aantal elementaire operaties dat nodig is voor het klassificeren van één test-sample in dit model is 653,000.

Opdracht D

“Bij deze opdracht passen we het in opdracht C gevonden model toe om de posities van handgeschreven cijfers in een plaatje te localiseren.”

Opdracht D1

“ Leg uit hoe elk van die plaatjes tot stand komt en verklaar aan de hand van een paar voorbeelden wat er te zien is in die plaatjes.“*

largeTestImage
Dit is de uitvoer van het model voor een testafbeelding. Het toont de classificatie of voorspelling van het model voor die afbeelding.

Featuremaps png’s
Dit is afhankelijk van de specifieke laag en het trainingsproces van het model. Over het algemeen vertegenwoordigen featuremaps de reactie van de filters op bepaalde kenmerken in de inputafbeelding. Door de featuremaps t e visualiseren, kunnen we de geactiveerde kenmerken in de afbeelding zien die het model heeft geleerd te herkennen. In dit geval wordt er gereageerd op randen, curven of textuurpatronen in de cijfers.

*" * Als het goed is zie je dat de pooling layers in grote lijnen bepalen hoe groot de uiteindelijke output bitmaps zijn. Hoe werkt dat?”*

Klopt, de pooling layers bepalen in grote lijnen de grootte van de uiteindelijke output bitmaps. Dit komt doordat pooling layers een downsampling doen op de invoergegevens. Door de downsampling wordt de resolutie van de matrix verlaagd, waarbij de belangrijkste kenmerken behouden blijven.

Het aantal elementen in de uitvoer is kleiner dan in de invoer, waardoor het model sneller kan werken en minder gevoelig is voor kleine verplaatsingen in de invoer. Max-pooling neemt de grootste waarde in elk bijv(2x2) gebied en creëert zo een verkleinde versie van de matrix met behoud van de belangrijkste kenmerken.

Opdracht D2

" Met welke welke stappen zou je uit de featuremap plaatjes lijsten kunnen genereren met de posities van de cijfers?"*

Stap 1: Bepaal een drempelwaarde: Stel een drempelwaarde in die bepaalt wanneer een pixel als geactiveerd wordt beschouwd. Dit kan bijvoorbeeld een bepaalde intensiteitswaarde zijn.

Stap 2: Loop door elke featuremap

Stap 3: Loop door elke pixel in de featuremap plaatje

Stap 4: Controleer of de intensiteitswaarde hoger is dan de drempelwaarde. Als dit het geval is, markeer dan de positie van de pixel.

Stap 5: Voeg de positie toe aan de lijst

```
# Drempelwaarde en detectie van posities
threshold = 0.5
digit_positions = []

for digit in range(10):
    digit_position = []

    # Loop door elke featuremap
    for featuremap_index, featuremap in enumerate(prediction2):
        # Loop door elke pixel in de featuremap
        for row in range(featuremap.shape[0]):
            for col in range(featuremap.shape[1]):
                # Controleer of de intensiteitswaarde hoger is dan de drempelwaarde
                if featuremap[row, col, digit] > threshold:
                    # Voeg de positie toe aan de lisjt
                    digit_position.append((featuremap_index, row, col))

    digit_positions.append(digit_position)
```

Opdracht D3

" Hoeveel elementaire operaties zijn er in totaal nodig voor de convoluties voor onze largeTestImage? (Tip: kijk terug hoe je dat bij opdracht C berekende voor de kleinere input image)"*

Voor het berekenen van het exacte aantal elementaire operaties gebruiken we een formule FLOPs die we voor elke convolutielaag berekenen.

Het model van opdracht C heeft drie convolutielagen:

conv_2d: Conv2D met 20 filters en kernelgrootte (5, 5)

conv_2d_1: Conv2D met 20 filters en kernelgrootte (3, 3)

conv_2d_2: Conv2D met 10 filters en kernelgrootte (5, 5)

Laten we aannemen dat de largeTestImage een grootte heeft van (1024, 1024)

Voor elke convolutielaag kunnen we het aantal FLOPs berekenen met behulp van de formule:
FLOPs = (Ho * Wo * Hi * Wi * K * K * Ci * Co) * B

Ho en Wo: de hoogte en breedte van de uitvoerfeaturemap
Hi en Wi: de hoogte en breedte van de invoerfeaturemap
K: de kernelgrootte
Ci en Co: het aantal kanalen (filters) in de invoer- en uitvoerfeaturemap
B: het aantal beelden (batchgrootte)

Dus voor de convolutie lagen berekenen we:

Conv_2d: Conv2D met 20 filters en kernelgrootte (5, 5):

- Ho = (1024 - 5 + 1) / 1 = 1020
- Wo = (1024 - 5 + 1) / 1 = 1020
- Hi = 1024
- Wi = 1024
- K = 5
- Ci = 1 (grijswaardenafbeelding)
- Co = 20
- B = 1 (aangezien we slechts één largeTestImage hebben)

FLOPs_1 = (1020 * 1020 * 1024 * 1024 * 5 * 5 * 1 * 20) * 1

Conv_2d_1: Conv2D met 20 filters en kernelgrootte (3, 3):

- Ho = (1020 - 3 + 1) / 1 = 1018
- Wo = (1020 - 3 + 1) / 1 = 1018
- Hi = 1020
- Wi = 1020
- K = 3
- Ci = 20
- Co = 20
- B = 1

FLOPs_2 = (1018 * 1018 * 1020 * 1020 * 3 * 3 * 20 * 20) * 1

Conv_2d_2: Conv2D met 10 filters en kernelgrootte (5, 5):

- Ho = (1018 - 5 + 1) / 1 = 1014
- Wo = (1018 - 5 + 1) / 1 = 1014
- Hi = 1018
- Wi = 1018
- K = 5
- Ci = 20
- Co = 10
- B = 1

FLOPs_3 = (1014 * 1014 * 1018 * 1018 * 5 * 5 * 20 * 10) * 1

Het totale aantal FLOPs voor alle convoluties is dan:

Total FLOPs = FLOPs_1 + FLOPs_2 + FLOPs_3

" Hoe snel zou een grafische kaart met 1 Teraflops dat theoretisch kunnen uitrekenen?"*

We willen berekenen hoe lang het zou duren om alle berekening uit te voeren die nodig zijn voor de convoluties van onze grote testafbeelding.

Laten we aannemen dat we in totaal 1 miljard berekeningen moeten doen. In dit geval zou het theoretisch slechts 1 seconde duren voor de grafische kaart om al die berekeningen uit te voeren.

Tijd = (1 miljard FLOPs) / (1 Teraflops) = 1 seconde

Maar het is belangrijk om te weten dat deze berekening erg vereenvoudigd is en geen rekening houdt met andere factoren die de prestaties kunnen beinvloeden.

" Waarom zal het in de praktijk wat langer duren?"*

Dit komt door enkele factoren die de totale berekeningstijd beïnvloeden. Een belangrijke factor is geheugentoegang. Tijdens de berekingen moet de grafische kaart mogelijk gegevens ophalen uit het geheugen, zoals de invoer- en filtergegevens voor de convoluties. Hoewel de kaart zelf ongelooflijk snel kan rekenen, kan het ophalen van gegevens van het geheugen relatief traag zijn. Dit betekent dat er enige vertraging kan optreden terwijl de kaart wacht op de benodigde gegevens.

Daarnaast is er ook geheugenbandbreedte. Dit verwijst naar de snelheid waarmee gegevens van en naar het geheugen kunnen worden overgedragen. Als de geheugenbandbreedte niet voldoende is om aan de eisen van de berekeningen te voldoen, kan dit resulteren in wachttijden en vertragingen.

Ook kan natuurlijk de grafische kaart voor meerdere doeleinden tegelijkertijd gebruikt worden, zoals het weergeven van grafische beelden op een scherm. Dus kunnen er ook meerdere taken tegelijkertijd zijn, waardoor het de beschikbare rekenkracht vermindert.