Algebraic-representation

Wilco Matthijssen
Twan Hoogveld

Uitleg Code

Call Tree

In de huidige code is niet echt een call tree groter dan alleen de main functie.
Dit komt omdat er veel gebruik wordt gemaakt van de keras library. Deze gebruikt vooral objecten. Op deze objecten kunnen wij dan ook functies aanroepen, maar niet zo zeer zelf geschreven functies die zijn toegevoegd. Alle functies die überhaupt zijn aangeroepen staan wel hieronder aangegeven.

main
- keras.datasets.mnist.load_data()
- (np) astype()
- (np) astype()
- np.expand_dims()
- np.expand_dims()
- keras.utils.to_categorial()
- keras.utils.to_categorial()
- summary()
- compile()
- fit()
- evaluate()

Uitleg

je begint met een **keras.datasets.mnist.load_data**, hieruit worden de **train** en **test** sets opgehaald. Op deze manier heb je meteen beide sets in je script. (Gecached voor snellere volgende runs)

Vervolgens wordt alle data hierin genormaliseerd door middel van de numpy functie **astype**.

Na deze stap zorgen we dat alle afbeeldingen in de **train** en **test** set echt 28x28 px groot zijn. Dit is om in de toekomst geen problemen met OutOfBounds te krijgen o.i.d.

keras.utils.to_categorical zet ‘een’ value in een vector om naar een binaire matrix. Alle values worden dus vervangen door een 1 of 0. De **num_classes** van 10 werkte in ons geval het allerbeste. Als je hier geen 10 meegeeft, maar bijvoorbeeld niks, dan wordt de grootste value in de vector + 1 gepakt. En dat gaf een minder goed resultaat.

Nu het meeste voorwerk is gedaan kan er een **model** worden gemaakt met de daarbij horende **layers.** Allereerst geef je een **input** aan je model. dit is bij ons de **input_shape**, deze shape heeft de grootte van het aantal pixels in iedere afbeelding. 28x28 dus.

De eerste **layer** is een **Conv2D**, dit is een spatial convolution. Hierin gebruiken wij 32 als batch size. **kernel_size** van 3x3 & de **activation** op 'relu'. Selu gaf hier een minder goed resultaat wat in de eerste instantie niet verwacht werd.

Na de Conv2D layer komt er een **MaxPooling2D** layer. Dit downsampled de input die de functie krijgt met een windows ter grote van de **pool_size**. Hieruit wordt de grootste value gehaald. De rest van de pixels wordt dan verwijderd. Dit geeft een duidelijker beeld voor de vorm van het cijfer.

De 2 bovengenoemde layers worden nogmaals toegepast in dezelfde volgorde. een krijgt en volgende **Conv2D** een grotere batch size mee van 64.

De **Flatten** layer veranderd de shape van de output. Deze gaat van een matrix naar een Array.

Dropout layer zet random inputs naar 0 met een frequentie van voorkomen van de meegeven parameter. Deze is in ons geval 0.5. Dit zorgt voor een vermindering van overfitting. Hier heb je zonder de **dropout** layer meer last van waardoor je resultaten slechter worden.

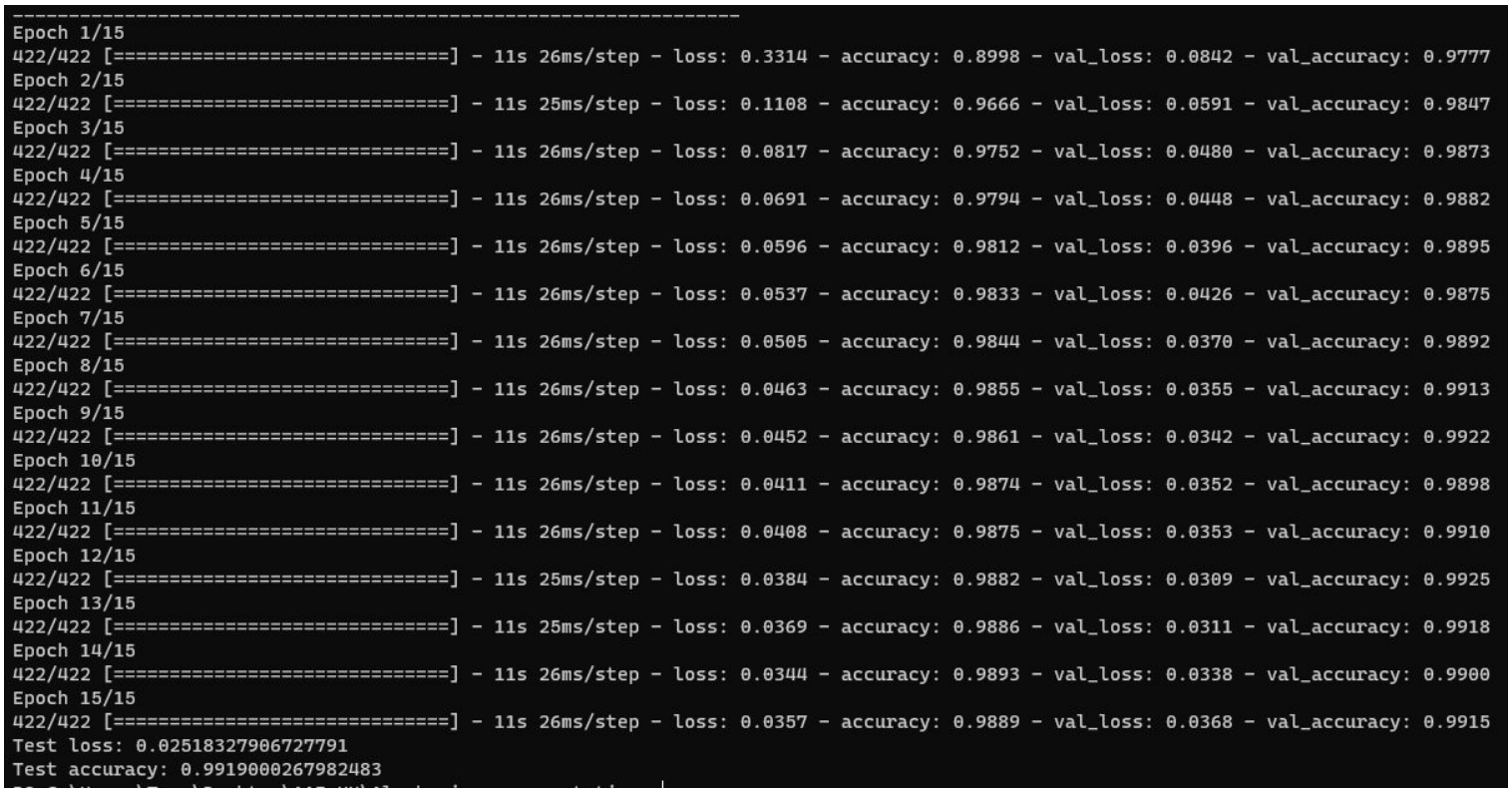
De laatste layer is een **Dense** layer. Deze zorgt voor de output shape van het model en dat alles goed aan elkaar wordt geknoopt. Deze krijgt een **num_classes** mee en zet de output shape dan ook op deze groten.

Op dit **model** kan je een **compile** aanroepen, hierin geef je aan de de metric de accuracy is. Vervolgens kan je het model trainen door de **fit** functie aan te roepen. Deze krijgt de **x_train, y_train** een **batch_size, epocs & validation_split** mee.

De **batch_size** zorgt voor een hoeveelheid waarop wordt getraind. bij 128 pakt deze dus 128 samples en traint de set hierop. Verder spreken de variabelen voor zichzelf.

Resultaten

na meerdere keren testen en runnen komen we vrijwel altijd op de ~99% uit. Het testen met 15 epochs gaat vrij snel. een epoch duurt ongeveer 8/9 seconden op een systeem met 6 cores / 12 threads. Keras maakt volledig gebruik van alle cores in dit geval. Hoewel het aantal epochs na 8 vrijwel niet veel meer uitmaakt en het je nog maar weinig resultaat extra geeft is het natuurlijk te betwijfelen of 15 nodig is, maar voor de zekerheid en de korte runtime is hier toch voor gekozen.



Antwoord op de vraag

De beste variablen zijn in ons geval:

- num_classes = 10
- batch_size = 128
- kernel_size = (3,3)
- pool_size = (2,2)

en 7 layers in het NN.

Er is getest met verschillende kernel- en pool-sizes maar dat maakte een klein verschil. Niet in snelheid maar vooral in de nauwkeurigheid. Zo gaf een kernel van 4x4 bijvoorbeeld een lager resultaat dan 3x3. En een te groot kernel zou weer niet accuraat genoeg zijn omdat de afbeeldingen daar te klein voor zijn.

Vrijwel alle variabelen die nu worden aangepast geven een minder resultaat. Zo ook bijvoorbeeld de batch_size. Als je deze op 512 zet, 4 keer meer dan 128, dan krijg je een minder resultaat van 98.94 %. Dit scheelt weinig, maar het scheelde wel iets natuurlijk. Alleen de doorlooptijd van 512 was sneller met 2 seconden per epoch.