Experiment handmatig hypertunen Convolutional Neural Networks

Stephan Eikenhorst & Floris Verheijen

Github Repository: *https://github.com/Willem03/ML22_opdracht1*

1. Aanpak

Stap 1: Variëren met het aantal Convolutional Layers
- Creëren modellen met verschillend aantal Convolutional layers (1, 2 of 3);
- Aantal units in de Linear Layers is gerelateerd aan het aantal Convolutional Layers; alle overige parameters houden we constant: *Linear Layers*: 3, *Filters in conv layer:* 32 & 32, *kernel size*: 3; *loss fie*: CrossEntropyLoss, *learning rate (LR)*: 0,01 & *Optimizer*: Adam.
- Trainen van de modellen **M1.3.32.32, M2.3.32.32 & M3.3.32.32** met 5 epochs; modelkeuze op basis van accuracy.

Stap 2: Variëren met het aantal Linear Layers
- Creëren modellen met verschillend aantal Linear layers (2, 3 of 4);
- Aantal units in de Linear Layers is gerelateerd aan het aantal Convolutional & Linear Layers; Alle overige parameters houden we constant. Idem als in stap 1;
- Trainen van modellen met 5 epochs;
- Gebruikte modellen: **M2.2.32.32, M2.4.32.32, M3.2.32.32 & M3.4.32.32**

Stap 3: Variëren met het aantal channels in de Convolutional layers
- Aanpassen van het aantal channels in Convolutional layers in de modellen **M2.2, M2.3, M2.4, M3.2, M3.3 & M3.4**. Per model worden de volgende combinaties channels gebruikt: **.16.32, .32.64 & .64.128 (bv: M2.4.32.64)**;
- Bijbehorend worden ook het aantal units in de Linear layers mee aanpast. Alle overige parameters houden we constant. Idem als in stap 1 & 2;
- Alle modellen trainen met 5 epochs;
- Op basis van de accuracy & loss functie worden de best presterende modellen opnieuw getraind met 30 epochs;

Stap 4: Variëren met de learning rate
- N.a.v. resultaten in Stap 3 enkele modellen 5 epochs trainen met een LR van 0.001;
- Vervolgens de twee beste modellen 50 epochs trainen;
- Van deze twee modellen het beste model 5 epochs trainen met learning rates van 0.005 & 0.0005.

2. Resultaten

Stap 1:

Een model met meer dan 3 Conv Layers en een kernel size van 3x3 is niet mogelijk, omdat de plaatjes dan te klein worden. Daarom is er geen model M4.3.32.32. Verder presteert een model met 1 Conv Layer (**M1.3.32.32**) ongeveer gelijk aan **M2.3.32.32 & M3.3.32.32**, maar 5 epochs trainen kost onevenredig veel meer tijd dan de modellen met 2 of 3 Conv Layers (zie tabel 1). Dit komt door het grote aantal te trainen parameters. Daarom wordt M1.3.32.32 niet verder onderzocht.

Tabel 1

Model	Train tijd	Param.	5 Epochs	
			Max Acc.	Gem. Acc.
M1.3.32.32	24m 59s	24,606,954	89,2%	88,7%
M2.3.32.32	3m 1s	842,762	87,1%	87,8%
M3.3.32.32	2m 29s	29,482	87,8%	86,8%

Stap 2:

Naarmate het aantal lagen toeneemt (Linear & Convolutional), lijkt de prestatie net wat af te nemen. Dus, beter modellen gebruiken met minder lagen.

Tabel 2

	Model.Conv.Lin.FilterIn.FilterOut	Tot. Aant. Parameters	5 Epochs	
			Max Acc.	Gem. Acc.
Stap 1	M1.3.32.32	24,606,954	89,2%	88,7%
Stap 2	M2.2.32.32	679,466	87,1%	86,0%
Stap 1	M2.3.32.32	842,762	87,0%	85,2%
Stap 2	M2.4.32.32	882,938	84,3%	86,9%
Stap 2	M3.2.32.32	27,722	87,8%	86,8%
Stap 1	M3.3.32.32	29,482	87,7%	86,3%
Stap 2	M3.4.32.32	29,482	86,9%	85,3%

Stap 3

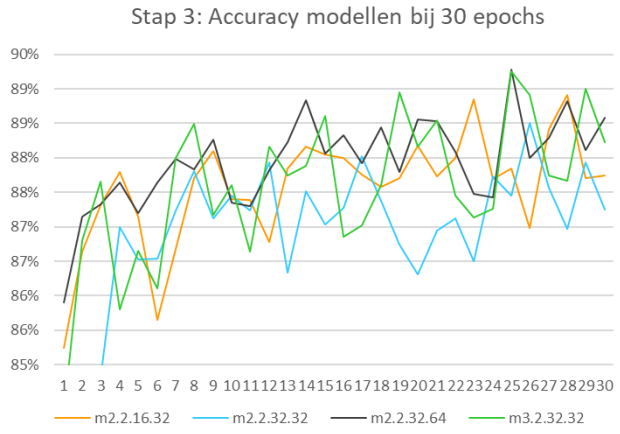
Een aantal zaken valt op in de resultaten van stap 3: Modellen met de filtercombinatie 64-128 vragen 3 tot 4 keer zoveel tijd om te trainen. Variatie met het aantal filters lijkt een patroon uit stap 2 te bevestigen: hoe meer lagen in een model, hoe minder goed de prestatie.

Tabel 3

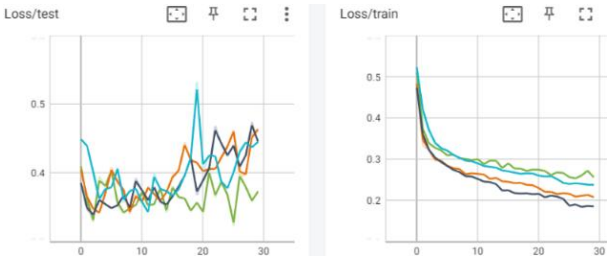
Model.Conv.Lin.FilterIn.FilterOut		5 Epochs			
		Avg. Avg. Acc. Modellen	Max Acc.	Avg. Acc.	Tijd
Stap 3	M2.2.16.32	86,6%	88,9%	87,6%	2m 9s
Stap 1	M2.2.32.32		87,1%	86,0%	2m 58s
Stap 3	M2.2.32.64		85,9%	87,6%	5m 19s
Stap 3	M2.2.64.128		86,0%	85,0%	14m 57s
Stap 3	M2.3.16.32	84,6%	85,7%	85,1%	5m 19s
Stap 1	M2.3.32.32		87,0%	85,2%	3m 13s
Stap 3	M2.3.32.64		85,4%	83,3%	5m 21s
Stap 3	M2.3.64.128		85,7%	84,9%	17m 45s
Stap 3	M2.4.16.32	66,0%	87,6%	86,8%	2m 23s
Stap 1	M2.4.32.32		84,3%	83,6%	3m 18s
Stap 3	M2.4.32.64		85,0%	83,5%	5m 42s
Stap 3	M2.4.64.128		10,5%	10,1%	17m 13s
Stap 3	M3.4.16.32	47,1%	85,3%	83,2%	5m 45s
Stap 1	M.3.4.32.32		86,9%	85,3%	2m 29s
Stap 3	M3.4.32.64		10,3%	10,2%	5m 40s
Stap 3	M3.4.64.128		10,1%	9,7%	17m 8s

Alleen modellen met 2 lineaire lagen en een laag aantal filters zijn verder getraind met 30 epochs (**M2.2.16.32, M2.2.32.32, M2.2.32.64 & M3.2.32.32**). Model **M2.2.32.64** (grijze lijn) presteert het beste.

Grafiek 1: Accuracy per model per epoch



Grafiek 2 & 3 Loss-functies van test & train data



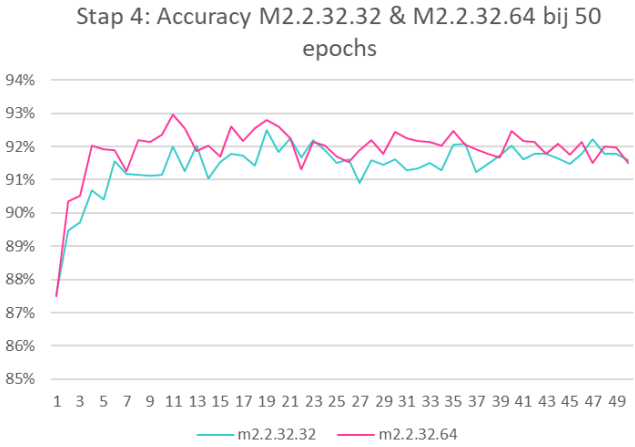
Stap 4

Een LR van 0.001 levert een hogere max accuracy op bij 5 epochs. De controle of de een LR met een factor vijf kleiner of groter (0.005 of 0.0005) levert net wat mindere resultaten op. Dus een LR van 0.001 lijkt het beste voor deze modellen.

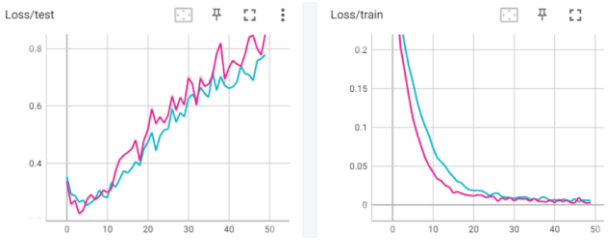
Model.Conv.Lin.FilterIn.FilterOut	LR = 0.01	LR = 0.001	LR = 0.005	LR = 0.0005
	Max Acc.	Max Acc.	Max Acc.	Max Acc.
M2.2.16.32	88,9%	90,5%	-	-
M2.2.32.32	87,1%	91,8%	89,9%	90,4%
M2.2.32.64	85,9%	91,6%	90,4%	91,2%
M3.2.32.32	87,8%	88,1%	-	-

Het vervolgens trainen van **M2.2.32.32 & M2.2.32.64** met 50 epochs laat vergelijkbare goede resultaten zien.

Grafiek 4: Accuracy per model per epoch



Grafiek 5 & 6 Loss-functies van test & train data



Conclusie

De gedachte was dat we eerst met de hyperparameters zouden variëren die de meeste invloed op de prestatie van de modellen zouden hebben. Het is bekend dat de Fashion MNIST data goed te trainen zijn met een LR van 0.001. Echter, tijdens Stap 1 t/m 3 zijn alle modellen getraind met een LR van 0.01. Dat had een LR van 0.001 moeten zijn. Alsnog aanpassen naar 0.001 is niet gebeurd, omdat het weer opnieuw trainen van alle modellen en het verwerken van alle resultaten nog meer en daarmee te veel tijd zou kosten. Het gevolg is dat het effect van keuzes t.a.v. andere hyperparameters minder goed zichtbaar is. Daardoor is de onderbouwing van de gemaakte keuzes minder sterk. Desondanks zijn er twee goede modellen uitgekomen **(M2.2.32.32 & M2.2.32.64)**, is ons goed duidelijk geworden wat het effect is van de keuze van hyperparameters en dat het veel tijd en geduld vraagt.