Kennismaking met neuronen in neurale netwerken

deel 3: neuronen in de praktijk

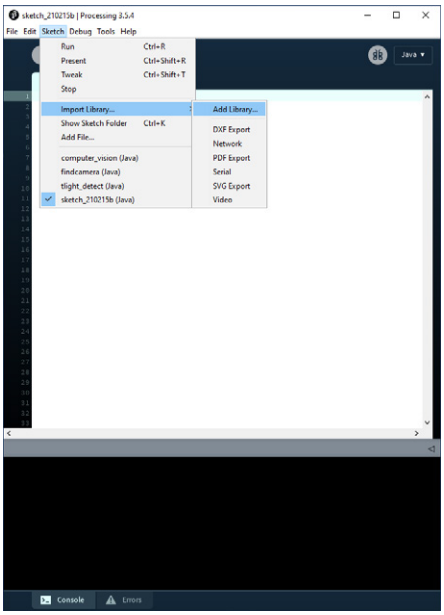
Stuart Cording (Elektor)

In de vorige afleveringen hebben we een neuraal netwerk gemaakt en inzicht gekregen in de werking ervan. We zijn er zelfs in geslaagd het te leren hoe de XOR-functie werkt, wat de classificatie van invoerpatronen vereist. In dit artikel zullen we onderzoeken hoe we een onderdeel van een autonoom rijdend systeem kunnen implementeren: het herkennen van de status van een verkeerslicht.

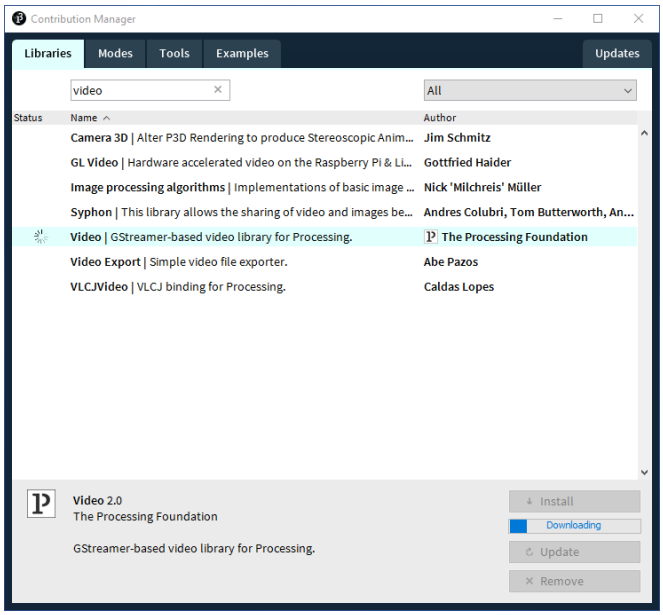
Er zijn al veel voorspellingen gedaan over zelfrijdende auto's [1], waarvan nog geen enkele is uitgekomen. Veel van de complexiteit komt voort uit pogingen om de bedoeling van andere weggebruikers en voetgangers te begrijpen. Veel van het autonoom rijden gaat echter over het classificeren van alles wat de vele sensoren en camera's rond het voertuig 'zien'. Dit varieert van verkeerslichten en verkeersborden tot straat- of wegmarkeringen, voertuigtypes en mensen. Nu we een werkend neuraal netwerk hebben in de vorm van ons meerlaags perceptron, gaan we dat gebruiken om de kleuren van een verkeerslicht te detecteren. Een van de geweldige dingen van de Processing-omgeving die we tot nu toe hebben gebruikt is het gemak waarmee webcams kunnen worden benaderd. Onze voorbeelden hier draaien op vrijwel elke laptop of PC met Windows, Linux of macOS met een geïntegreerde of externe camera.

Zoek de camera

Voordat we aan de slag kunnen, moet er een nieuwe bibliotheek aan Processing worden toegevoegd die toegang geeft tot eventueel gekoppelde webcams. Selecteer in het menu *Sketch -> Import Library... -> Add Library...* (**figuur 1**), waarmee het venster van **figuur 2** wordt geopend. Zorg ervoor dat het tabblad *Libraries* is geselecteerd en voer *"video"* in het zoekvak in. Uit de lijst die verschijnt selecteren we



Figuur 1. Voeg een nieuwe bibliotheek toe aan Processing.



Figuur 2. Zoeken en installeren van de Video-bibliotheek.

Video|GStreamer-based video library for Processing en klikken vervolgens op *Install*. Net als eerder gebruiken we de code uit de GitHub-repository die voor deze serie artikelen is voorbereid [2].

Wanneer de bibliotheek is geïnstalleerd, kunnen we het project */trafficlight/findcamera/findcamera.pde* uitvoeren. Deze code vraagt gewoon de lijst op van de beschikbare camera's die op de computer zijn aangesloten en laat de gebruiker er eentje selecteren door een nummer tussen 0 en 9 in te voeren. Zorg ervoor dat het kleine *findcamera*-venster actief is (klik er in) voordat u een nummer invoert. Als het Processing IDE-venster actief is, typt u een nummer ergens in de broncode...

Zodra een camera is geselecteerd, wordt de uitvoer ervan met een geringe resolutie (320 × 240 pixels) in het venster getoond (**figuur 3**). De tekstconsole bevat ook de broncode die nodig is om de camera te selecteren voor de volgende twee projecten die we zullen bekijken (**figuur 4**).

Tenslotte hebben we een verkeerslicht nodig. Aangezien u er waarschijnlijk geen bij de hand hebt, is er een klaargezet in *trafficlight/resources* in verschillende bestandsformaten. Druk een daarvan af en houd die bij de hand.

Wanneer u Processing-projecten stopt die de camera gebruiken, kan de boodschap "*WARNING: no real random source present!*" verschijnen. Dit lijkt een bug te zijn die samenhangt met het gebruik van de Video-bibliotheek, maar heeft geen invloed op de functionaliteit van de code.

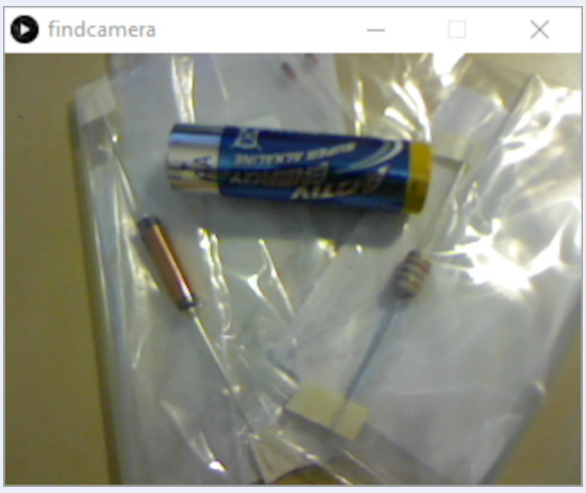
Hoe zien camera's?

Eén van de belangrijkste aspecten van het gebruik van neurale netwerken is inzicht in de wijze waarop een computer de binnenkomende gegevens interpreteert. In het geval van camera-invoer wordt het computerbeeld weergegeven door de drie primaire kleuren rood, groen en blauw te mengen. Dit wordt gewoonlijk het RGB-formaat genoemd. De drie waarden variëren tussen 0 en 255 (of 0x00 en 0xFF in hexadecimaal). Als we de camera op iets blauws richten, zouden we verwachten dat de B-waarde relatief hoog is en de andere waarden vrij laag. Richten we de camera op iets geels, dan zouden de R- en G-waarden hoog moeten zijn, omdat rood en groen samen geel maken. Om een beter gevoel te krijgen voor additieve kleurmenging, kunnen we het project */trafficlights/addititive/additive.pde* aanbevelen (**figuur 5**).

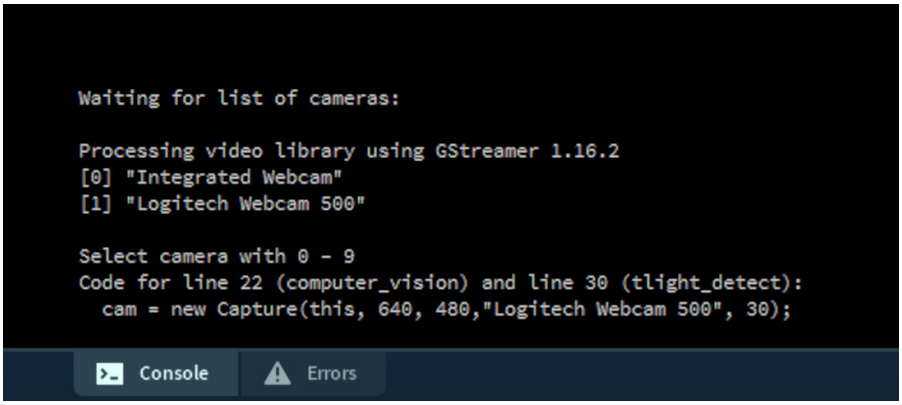
Gewapend met deze kennis gaan we nu onderzoeken hoe onze camera de kleuren van ons verkeerslicht 'ziet' en de RGB-waarden noteren die worden gerapporteerd. Vervolgens kunnen we ons neuraal netwerk de drie kleuren van het verkeerslicht leren, zodat het deze kan classificeren als rood, oranje (geel) en groen.

Bepaling van de RGB-waarden

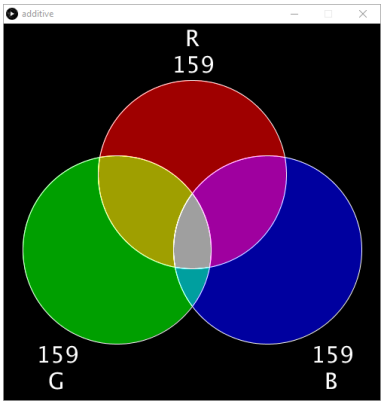
Het project *trafficlight/computer_vision/computer_vision.pde* is het volgende project dat we nodig hebben. Dit toont de uitvoer van de geselecteerde camera naast de gerapporteerde RGB-waarden. Met dit project kunnen we de RGB-waarden opnemen die de camera ziet. Vergeet niet om de regel code die u hebt bepaald in *findcamera.pde* op regel 22 in dit project in te voegen voordat u met de code begint. Dit zorgt ervoor dat



Figuur 3. Voorbeeld van uitvoer van findcamera.pde.



Figuur 4. De console-uitvoer van findcamera.pde levert de initialisatie-code voor de andere projecten die de betreffende camera gebruiken.



Figuur 5. additive.pde demonstreert additieve kleurmenging.

de door u gekozen camera wordt gebruikt. Noteer de kleur die u ziet (bovenaan) terwijl u de camera op het rode verkeerslicht richt (houd het 'licht' ongeveer binnen de gestippelde cirkel) en de RGB-waarden voor deze kleur (**figuur 6**). De RGB-waarden zijn in feite het gemiddelde van alle pixels die de camera vastlegt in het rood omkaderde vierkant. Hoewel het een gemiddelde is, zullen de RGB-waarden snel fluctueren. Om een enkele waarde vast te leggen, drukt u op '*p*' op het toetsenbord; de code pauzeert dan bij een enkele reeks waarden. Zorg ervoor dat u in dit venster klikt voordat u op '*p*' drukt om het te activeren. Door op '*s*' te drukken start u de camera-opname en het genereren van RGB-waarden opnieuw. De hier gebruikte afdruk van het verkeerslicht was gelamineerd, dus er zijn hier en daar wat reflecties. Als u uw afbeelding met een laserprinter hebt afgedrukt, kunt u ook een ietwat reflecterend oppervlak hebben. Het gevolg is dat, hoewel de camera op het rode verkeerslicht is gericht, de 'waargenomen' kleur rozig kan zijn of zelfs bijna wit. Dit is uiteraard niet erg leerzaam voor het neurale netwerk. Het moet de kleur 'verkeerslichtrood' onder optimale omstandigheden kennen, dus verplaats de camera of verander de belichting tot u het gevoel hebt dat de RGB-waarde de kleur optimaal weergeeft.

Tabel 1. RGB-waarden voor de drie kleuren van het verkeerslicht.

Kleur	R	G	B
Rood	220	56	8
Geel	216	130	11
Groen	123	150	128

Figuur 6. computer_vision.pde levert de RGB-waarden die de camera 'ziet' voor elke kleur van het verkeerslicht.

Figuur 7. De respons van tlight_detect.pde voordat het kleuren heeft geleerd.

Dit werpt nog een ander probleem op in verband met de nauwkeurigheid. Bestuurders weten hoe moeilijk het is om te bepalen welk verkeerslicht actief is wanneer de zon direct in onze ogen of op het verkeerslicht schijnt. Als wij mensen al niet kunnen onderscheiden welke lamp brandt, hoe kan een neuraal netwerk dat dan? Het antwoord is: dat kan het niet. Als we een robuuster algoritme nodig hebben, moeten we het trainen met gegevens over 'slecht zicht'. Misschien hebben we ook andere gegevens nodig, zoals waar de zon op dat moment staat, zodat het neurale netwerk weet wanneer er sprake is van 'slecht zicht' en dat het de betreffende set waarden moet gebruiken. Tenslotte zouden we de door de camera geleverde data kunnen verbeteren, misschien door een infraroodbeeld van het verkeerslicht toe te voegen (welke lampen zijn heet) of een andere slimme filtering. We zullen onze trainingsgegevens wat robuuster maken om enige variatie in de gedetecteerde kleur van het verkeerslicht aan te kunnen.

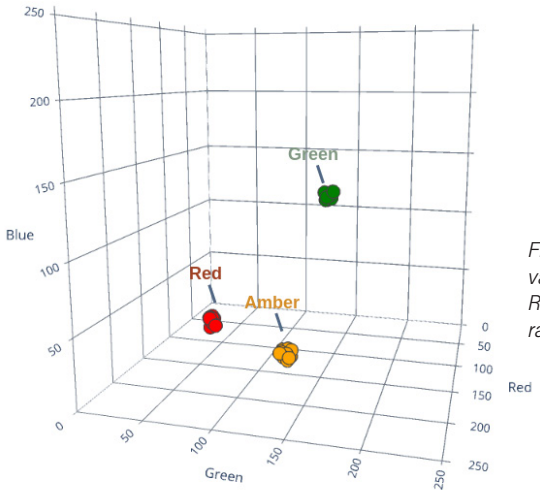
De belangrijkste taak op dit punt is het verzamelen van de RGB-gegevens voor de rode, gele en groene verkeerslichten met behulp van camera onder de gegeven lichtomstandigheden. De gegevens die met de opstelling van de auteur zijn verzameld, zijn te zien in **tabel 1**.

Verkeerslichten detecteren

Gewapend met onze gegevens, kunnen we nu het neurale netwerk de kleuren van ons verkeerslicht trainen. Voor deze laatste fase hebben we het project *trafficlight/tlight_detect/tlight_detect.pde* nodig, geopend in Processing. Als eerste moet u de juiste camera-initialisatiecode in regel 38 invoegen (uit *find_camera.pde*).

Dit project maakt gebruik van een MLP met drie ingangen, zes verborgen nodes en vier uitgangen (3/6/4). De drie ingangen zijn voor de drie kleuren R, G, en B. Drie van de vier uitgangen zijn voor het classificeren van 'rood', 'geel' en 'groen' van het verkeerslicht. De vierde uitgang zal later worden gebruikt. Het gebruik van zes verborgen nodes is willekeurig gekozen omdat dit 'genoeg' is voor de classificatie. We zullen zien dat deze configuratie werkt en de lezer wordt aangemoedigd te experimenteren met meer of minder verborgen nodes.

Als de code wordt uitgevoerd '*as is*', werkt het neurale netwerk zonder te leren. De resultaten (**figuur 7**) laten zien dat elke kleur wordt geclassificeerd als alle kleuren die het netwerk zoals gedefinieerd moet detecteren. Het leren van het neurale netwerk vindt plaats rond regel 51. Haal het commentaar weg bij de eerste drie methoden en voeg de RGB-waarden toe die u eerder hebt verkregen. De drie methoden die worden gebruikt om rood, geel en groen te definiëren zijn:



Figuur 8. De drie kleuren van het verkeerslicht als RGB-waarden in 3D na random-variatie.

```
teachRed(159, 65, 37);
teachAmber(213, 141, 40);
teachGreen(128, 152, 130);
```

Telkens wanneer deze functies worden aangeroepen, wordt het netwerk getraind om die RGB-combinatie te classificeren als de bijbehorende kleur (**figuur 8**). Om enige variatie in verlichting en automatische veranderingen in belichting door de camera mogelijk te maken, wordt een beetje variatie (±4) losgelaten op de RGB-waarden met behulp van een `randomise()`-functie (regel 336). Ook hier kunt u experimenteren met de effectiviteit van deze aanpak en de mate van variatie die wordt toegepast.
Het trainen van het netwerk gaat snel in vergelijking met vorige projecten, omdat we de foutwaarden tijdens het leren niet langer naar een bestand schrijven. Start het project en na een paar seconden zal het neurale netwerk de kleur in het rood omkaderde vierkant van het cameravenster gaan evalueren (**figuur 9**).
Het verkeerslicht in het gezichtsveld van de camera wordt bepaald op regel 156 en verder. De vastgelegde RGB-kleuren worden aangelegd op de ingangen van het MLP en de uitvoer van het netwerk wordt berekend:

```
network.setInputNode(0, (float) r
   / 255.0);
network.setInputNode(1, (float) g
   / 255.0);
network.setInputNode(2, (float) b
   / 255.0);
network.calculateOutput();
```

De beslissing over welke kleur is gezien, wordt genomen vanaf regel 171. De uitvoer van elke uitvoernode wordt geëvalueerd. Indien de classificatiezekerheid boven 90% (0,90) ligt, wordt de gedetecteerde kleur in een cirkel weergegeven, samen met de classificator 'Rood' (red), 'Geel' (amber) of 'Groen' (green).

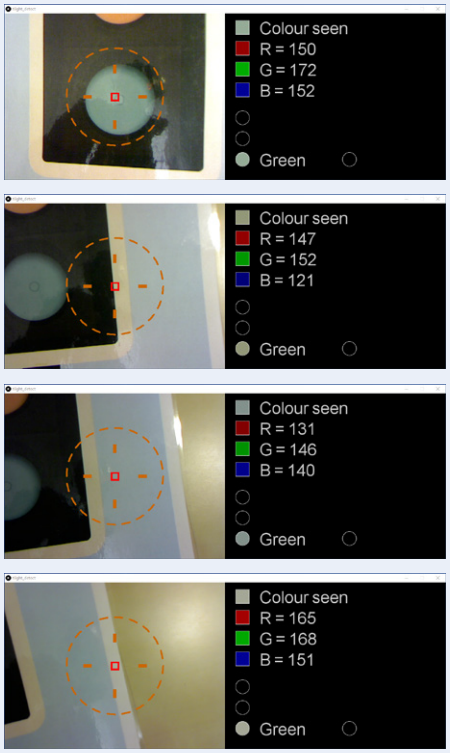
```
// If likelihood of 'Red' > 90%...
if (network.getOutputNode(0) >
   0.90) {
  fill(200, 200, 200);
     // …write "Red"…
  text("Red", 640+(100), 320);
     // … and set color to the
       color seen.
  fill(r, g, b);
} else {
     // Otherwise, set color to
        black
  fill(0, 0, 0);
}
// Now draw the color seen in a
   circle
ellipse(640+(50), 300, 40, 40);
```

U kunt experimenteren met de nauwkeurigheid van het MLP door de camera op de verschillende verkeerslichten te richten onder verschillende hoeken en onder verschillende lichtomstandigheden. Probeer ook de camera te richten op objecten in uw omgeving die naar uw mening de kleuren benaderen die het MLP heeft geleerd.
Het valt u wellicht op dat een breed scala van kleuren foutief wordt geclassificeerd als een bekende kleur. In het voorbeeld van de auteur krijgt ook de omgeving van het verkeerslicht, het frame en de algemene beeldachtergrond de classificatie 'groen' (**figuur 10**).

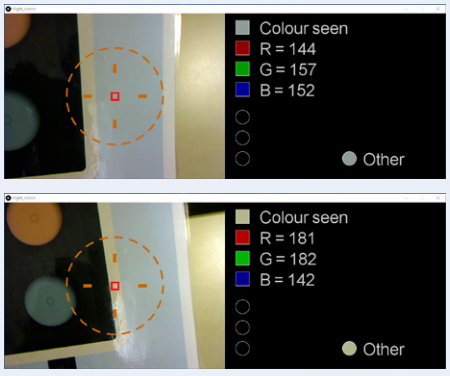
Aanscherping van de classificatie

Uit de RGB-waarden blijkt duidelijk dat de kleuren die aan het MLP worden doorgegeven tamelijk dicht in de buurt liggen van de gewenste classificatie voor 'groen'. Er zijn

Figuur 9. tlight_detect.pde nadat het de kleuren rood, geel en groen heeft geleerd.



Figuur 10. Foutieve classificatie van andere kleuren als 'groen.'



Figuur 11. Classificatie van andere kleuren als 'groen' is opgelost met de 'other'-node.

Tabel 2. RGB-waarden voor de 'ongewenste' kleuren.

Object	R	G	B
Donkere omgeving van het verkeerslicht	76	72	35
Wit kader	175	167	138
Blauwe achtergrond	152	167	161

verschillende manieren om de classificatie aan te scherpen. De eerste zou zijn om de 90%-drempel voor nauwkeurige classificatie te verhogen. Een andere mogelijkheid is het aantal leer-epochs te verhogen. De andere is de implementatie van de classificatie te heroverwegen.

Tot nu toe hebben we ons geconcentreerd op wat we positief willen classificeren. Soms kan het echter helpen om het neurale netwerk te leren wat *niet* tot de patronen behoort die we zoeken. In wezen kunnen we dan zeggen: "Dit zijn drie dingen waarnaar we op zoek zijn, maar hier zijn enkele soortgelijke dingen waarnaar we zeker niet op zoek zijn." Dit is waar onze vierde output-node in het spel komt.

Als we nog eens teruggaan naar het *computer_vision.pde*-project, kunnen we de RGB-waarden vastleggen voor kleuren die we willen classificeren als 'overig'. Als voorbeeld werden de RGB-waarden voor de donkere rand van het verkeerslicht, de witte rand en de blauwe achtergrond verzameld, met de resultaten van in **tabel 2**.

Terug in *tlight_detect.pde* kunnen deze waarden worden aangeleerd als 'overige' ongewenste kleuren met behulp van de `teachOther()`-methode. Verwijder gewoon de commentaartekens uit de code rond regel 60 en voeg uw RGB-waarden in:

```
teachOther(76, 72, 35);
teachOther(175, 167, 138);
teachOther(152, 167, 161);
```

Het opnieuw uitvoeren van het project leidt tot een merkbare verbetering. Het gebied rond de verkeerslichten (rand, kader, achtergrond) wordt nu geclassificeerd als 'overig' in plaats van als 'groen' (**figuur 11**).

In de volgende aflevering...

In dit artikel hebben we gezien hoe een neuraal netwerk een echt classificatieprobleem oplost. We hebben ook geleerd dat het nuttig kan zijn om zowel de gewenste classificatie als de ongewenste classificatiegegevens aan te leren.

Waarom bouwen we niet voort op dit voorbeeld om het volgende te onderzoeken?

- Hoe 'robuust' kan het MLP worden gemaakt voor variaties in camerahoek en belichting? Is het beter om de leerdata meer te variëren of de lat voor classificatie hoger te leggen (> 90%)?
- Verbetert de nauwkeurigheid als het aantal 'overige' output-nodes wordt vergroot en elke node een ongewenste kleur leert?
- Welke invloed heeft het verlagen of verhogen van het aantal verborgen nodes op het systeem?
- Zou een vierde ingang voor 'algemene helderheid' helpen om de herkenningsnauwkeurigheid bij variërende lichtomstandigheden te vergroten?

Het is natuurlijk 'lekker' om neurale netwerken op krachtige laptops en PC's te draaien, maar veel lezers willen dat ongetwijfeld doen op de kleinere processoren van boards zoals Arduino. In het laatste artikel van deze serie zullen we een RGB-sensor en een Arduino gebruiken om een kleur-detecterend neuraal netwerk te implementeren. En misschien vormt dat dan de basis voor uw volgende slimme Arduino-project!

210160-C-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via stuart.cording@elektor.com of naar de redactie van Elektor via redactie@elektor.com.

Een bijdrage van

Idee, tekst en foto's: **Stuart Cording**
Redactie: **Jens Nickel, C.J. Abate**
Illustraties: **Patrick Wielders**
Vertaling: **Jelle Aarnoudse**
Layout: **Harmen Heida**

WEBLINKS

[1] M. Anderson, "Surprise! 2020 Is Not the Year for Self-Driving Cars," IEEE Spectrum, April 2020: http://bit.ly/2ZSwm5f
[2] GitHub repository - simple-neural-network: https://bit.ly/2ZHLv9p
[3] H. Zhang et al., "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks," CVF, December 2017: https://bit.ly/3qZccCs