voyage dans les réseaux neuronaux (3e partie)

Les neurones pratiques

Stuart Cording (Elektor)

Jusqu'à présent, nous avons créé un réseau neuronal et compris comment il fonctionne. Nous avons même pu lui apprendre les mécanismes de la fonction OU exclusif (XOR), qui nécessite la classification de modèles d'entrée. Dans cet article, nous examinerons comment mettre en œuvre une partie d'un système de conduite autonome : reconnaître l'état de feux de circulation.

De nombreuses prédictions ont été faites concernant les voitures à conduite autonome [1], mais aucune ne s'est concrétisée. Une grande partie de la complexité provient de la tentative de comprendre l'intention des autres usagers de la route et des piétons. Cependant, le système de conduite autonome consiste surtout à classer tout ce que les nombreux capteurs et caméras « voient » autour du véhicule. Il peut s'agir de feux ou de panneaux de signalisation, mais aussi de marquage des rues ou des routes, ou encore de types de véhicules et de personnes. Maintenant que nous disposons d'un réseau neuronal fonctionnel sous la forme de notre perceptron multicouche (MLP), utilisons-le pour détecter les couleurs d'un feu de signalisation.

L'un des avantages de l'environnement Processing utilisé jusqu'à présent est la facilité avec laquelle il peut accéder aux webcams. Les exemples décrits ici fonctionneront sur à peu près n'importe quel ordinateur portable ou PC sous Windows, Linux ou macOS possédant une caméra intégrée ou externe.

Identifier votre caméra

Avant de commencer, il faut ajouter à Processing une nouvelle bibliothèque pour accéder à toutes les webcams connectées. Dans le menu, sélectionnez *Sketch -> Import Library... -> Add Library...* (**fig. 1**), ce qui ouvre la fenêtre illustrée par la **figure 2**. En vous assurant que l'onglet *Libraries* est sélectionné, entrez « *Video* » dans le champ de recherche. Dans la liste qui s'affiche, nous sélectionnons *Video | GStreamer-based video library for Processing*, puis nous cliquons sur *Install*. Comme précédemment, nous utilisons le code du dépôt GitHub préparé pour cette série d'articles [2].

Une fois la bibliothèque installée, nous pouvons exécuter le projet */trafficlight/findcamera/findcamera.pde*. Ce code demande simplement la liste des caméras disponibles connectées à l'ordinateur et permet à l'utilisateur d'en sélectionner une en saisissant un nombre entre 0

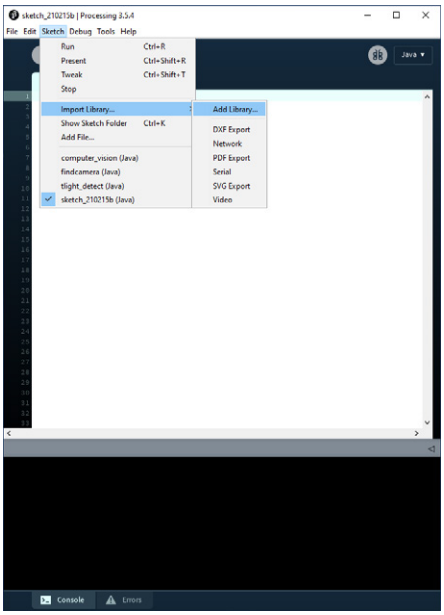

Figure 1. Ajout d'une nouvelle bibliothèque dans Processing.

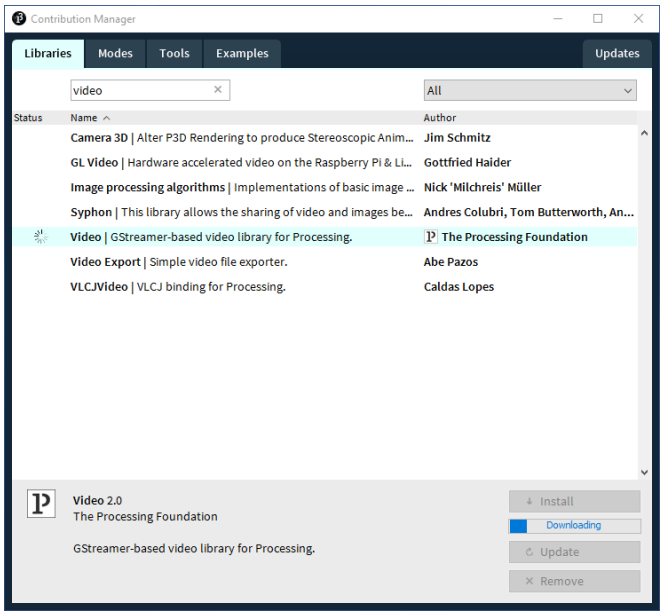

Figure 2. Recherche et installation de la bibliothèque Video.

et 9. Assurez-vous que la petite fenêtre *findcamera* soit active (cliquez à l'intérieur de la fenêtre) avant d'appuyer sur un nombre. Si la fenêtre de l'EDI Processing est active, vous taperez simplement un nombre quelque part dans le code source.

Une fois qu'une source de caméra a été sélectionnée, sa sortie est affichée en basse résolution (320×240 pixels) dans la fenêtre (**fig. 3**). La console texte fournit également le code source nécessaire à la sélection de votre caméra pour les deux prochains projets (**fig. 4**).

Enfin, nous avons besoin d'un feu de signalisation. Comme il est peu probable que vous en ayez un sous la main, il en a été préparé un dans *trafficlight/resources* avec différents formats de fichiers. Il suffit d'en imprimer un et de le garder à portée de main.

Lorsque vous arrêtez des projets Processing qui utilisent la caméra, vous pouvez voir le message « *WARNING: no real random source present !* » (avertissement : aucune source aléatoire réelle n'est présente). Il semble qu'il s'agisse d'un bogue lié à l'utilisation de la bibliothèque *Video* mais sans aucun impact sur la fonctionnalité du code.

Comment les caméras voient-elles ?

L'un des aspects les plus importants de l'utilisation des réseaux neuronaux consiste peut-être à comprendre comment un ordinateur peut interpréter les données entrantes. Dans le cas d'une entrée de type caméra, l'image informatique s'affiche en mélangeant les trois couleurs primaires, rouge, vert et bleu. C'est ce que l'on appelle communément le format RVB. Ces trois valeurs varient entre 0 et 255 (ou 0x00 et 0xFF en hexadécimal). Si nous dirigeons la caméra vers quelque chose de bleu, nous nous attendons à ce que la valeur B soit relativement élevée et les autres valeurs assez faibles. Si nous la dirigeons vers quelque chose de jaune, les valeurs R et V devraient être élevées, car, ensemble, le rouge et le vert forment le jaune. Pour avoir une meilleure idée du mélange additif des couleurs, le projet */trafficlights/additive/additive.pde* peut être intéressant (**fig. 5**).

Sachant cela, il semble logique de déterminer comment la caméra « voit » les couleurs de notre feu de signalisation et de noter les valeurs RVB qu'elle indique. Nous pouvons ensuite apprendre à notre réseau neuronal les trois couleurs des feux de signalisation afin qu'il puisse les classer en rouge, orange et vert.

Détermination des valeurs RVB

Découvrons le projet *trafficlight/computer_vision/computer_vision.pde*. Il affiche la sortie de la caméra sélectionnée à côté des valeurs RVB qu'elle indique. Avec ce projet, nous pouvons enregistrer les valeurs RVB que la caméra perçoit. Avant de commencer le code, n'oubliez pas de coller la ligne de code que vous avez déterminée dans *findcamera.pd* à la ligne 22 de ce projet. Cela garantit l'utilisation de la caméra que vous avez choisie.

En pointant la caméra vers le feu rouge (le feu doit se trouver approximativement dans le cercle en pointillés), notez la couleur vue (en haut) et les valeurs RVB qui la définissent (**fig. 6**). Les valeurs RVB sont en fait la moyenne de tous les pixels capturés par la caméra dans le carré rouge du centre. Bien

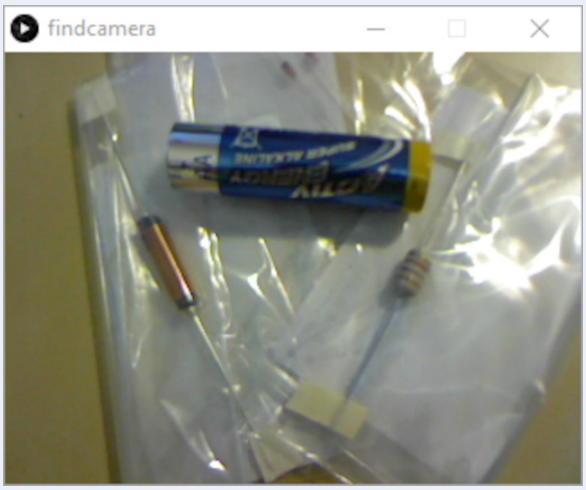

Figure 3. Exemple de sortie de findcamera.pde.

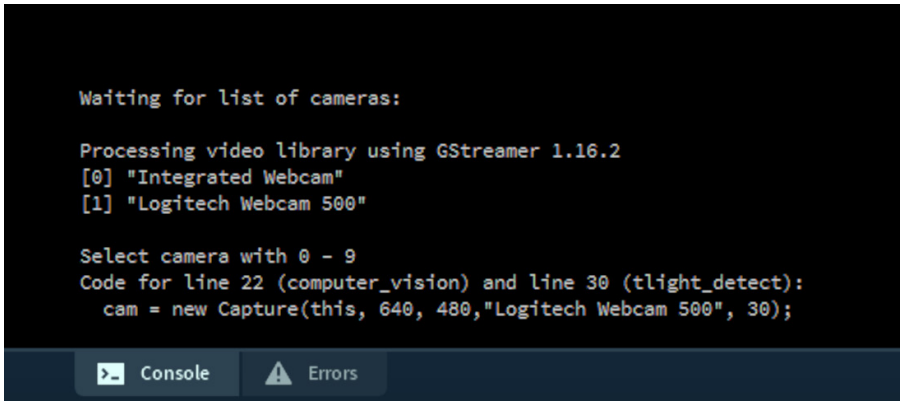

Figure 4. La sortie console de findcamera.pde fournit le code d'initialisation pour que les autres projets utilisent la caméra souhaitée.

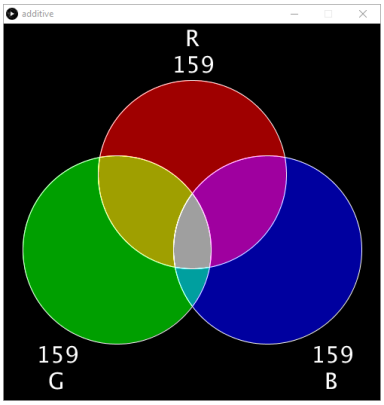

Figure 5. additive.pde montre le mélange additif de couleurs.

qu'il s'agisse d'une moyenne, les valeurs RVB varient rapidement vers le haut et vers le bas. Pour capturer une seule valeur, appuyez sur la lettre 'p' de votre clavier. Le code s'arrête alors sur un seul ensemble de valeurs. Veillez à cliquer dans cette fenêtre avant d'appuyer sur la lettre 'p' pour l'activer. En appuyant sur 's', vous redémarrez la capture de la caméra et la génération RVB.

L'impression des feux de signalisation utilisée ici a été plastifiée et présente donc quelques reflets par endroits. Si vous avez imprimé l'image des feux à l'aide d'une imprimante laser, il se peut que la surface soit légèrement réfléchissante. Dans ce cas, bien que la caméra soit dirigée vers le feu rouge, la « couleur perçue » peut être rosâtre, voire même proche du blanc. Ce qui n'est évidemment pas très utile pour le réseau neuronal. Il doit connaître la couleur « feu rouge » dans des conditions optimales. Pour cela, déplacez la caméra ou modifiez l'éclairage jusqu'à obtenir une valeur RVB représentant la couleur de manière optimale.

Cela soulève également une autre question concernant la précision. Les conducteurs savent à quel point il est difficile de discerner le feu de signalisation actif lorsque le soleil nous éblouit ou se réfléchit dans les lampes. Si nous, les humains, ne pouvons pas distinguer quelle lampe est allumée, comment un réseau neuronal peut-il le faire ? La réponse est simple : il ne peut pas le faire. Si nous avons besoin d'un algorithme plus robuste, nous devons l'entraîner avec des données de « mauvaise visibilité ». Il peut également être nécessaire de fournir une autre entrée, notamment l'endroit d'où provient la lumière du soleil, afin que le réseau neuronal sache quand la visibilité est mauvaise et qu'il utilise alors l'ensemble de données de mauvaise visibilité. Enfin, nous pourrions améliorer les données entrantes de la caméra, par exemple en ajoutant une image infrarouge du feu de circulation (indiquant les lampes chaudes) ou tout autre filtrage intelligent. Mais ne désespérez pas ! Nous allons ajouter une certaine robustesse à nos données d'entraînement pour gérer les variations de couleur des feux de circulation.

À ce stade, l'essentiel est de collecter les données RVB pour les feux rouge, orange et vert en utilisant votre caméra et vos conditions d'éclairage. Les données recueillies à l'aide de l'installation de l'auteur sont présentées dans le **tableau 1**.

Tableau 1. Valeurs RVB capturées pour les trois couleurs d'un feu de signalisation.

Couleur du feu	R	V	B
rouge	220	56	8
ambre	216	130	11
vert	123	150	128

Figure 6. computer_vision.pde crée les valeurs RVB « vues » par la caméra pour chaque couleur du feu de signalisation.

Figure 7. Réponse de tlight_detect.pde avant d'apprendre des couleurs.

Détection des feux de circulation

Armés de nos données, nous pouvons maintenant entraîner le réseau neuronal à repérer les couleurs de notre feu de circulation. Pour cette étape finale, nous aurons besoin du projet *trafficlight/tlight_detect/tlight_detect.pde* ouvert dans Processing. Commencez par vous assurer que le code d'initialisation correct de la caméra est collé dans la ligne 38 (depuis *find_camera.pde*).

Ce projet utilise un MLP à trois nœuds d'entrée, six nœuds cachés et quatre nœuds de sortie (3/6/4). Les trois entrées correspondent aux trois couleurs R, V et B. Trois des quatre sorties servent à classer les couleurs « rouge », « ambre » et « vert » du feu de signalisation. La quatrième sera utilisée ultérieurement. L'utilisation de six nœuds cachés a été choisie arbitrairement comme étant « suffisante » pour gérer la tâche de classification. Nous verrons que cette configuration fonctionne et, comme précédemment, les lecteurs sont encouragés à expérimenter avec un nombre plus ou moins grand de nœuds cachés.

Si vous exécutez le code « tel quel », le réseau neuronal fonctionne sans aucun apprentissage. Les résultats (**fig. 7**) montrent que toute couleur est classée en fonction de celles définies pour leur détection par le réseau.

L'apprentissage du réseau neuronal a lieu autour de la ligne 51. Supprimez les signes 'commentaire' des trois premières méthodes et ajoutez les valeurs RVB acquises précédemment. Les trois méthodes utilisées pour définir le rouge, l'ambre et le vert sont les suivantes :

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

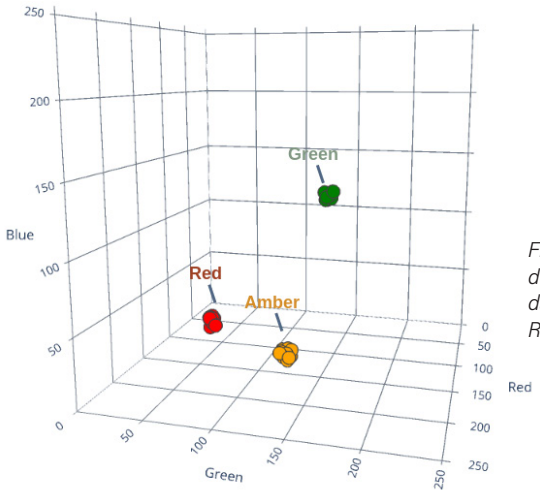

Figure 8. Les trois couleurs des feux de signalisation dans l'espace de couleurs RVB après randomisation.

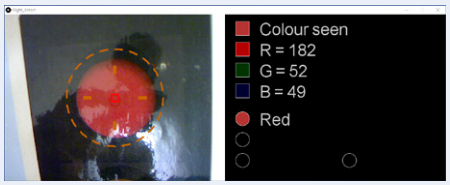

Figure 9. tlight_detect.pde après apprentissage des couleurs rouge, ambre et vert.

Chaque fois que ces fonctions sont appelées, le réseau est entraîné à classer cette combinaison RVB comme la couleur associée (**fig. 8**). Pour tenir compte de la variation de l'éclairage et des modifications automatiques du réglage de l'exposition par la caméra, une petite variation (±4) des valeurs RVB est appliquée à l'aide de la fonction `randomise()` (ligne 336). Là encore, vous pouvez expérimenter l'efficacité de cette approche et la quantité de randomisation appliquée.

L'entraînement du réseau est rapide par rapport aux projets précédents, car nous n'écrivons plus les valeurs d'erreur dans un fichier pendant l'apprentissage. Il suffit de lancer le projet. Quelques secondes après, le réseau neuronal commence à évaluer la couleur dans le carré rouge de la fenêtre de la caméra (**fig. 9**).

Le feu de circulation regardé par la caméra est déterminé à partir de la ligne 156. Les couleurs RVB capturées sont appliquées aux entrées du MLP et la sortie du réseau est calculée :

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

La décision sur la couleur vue est ensuite prise à partir de la ligne 171. La sortie de chaque nœud de sortie est évaluée. Si la certitude de classification est supérieure à 90 % (0,90), la couleur vue est affichée dans un cercle avec le classificateur « Rouge », « Ambre » ou « Vert ».

```
// Si la probabilité de la couleur
   « Red » > 90 %...
if (network.getOutputNode(0) >
    0.90) {
  fill(200, 200, 200);
  // ...écrire « Red »…
  text(«Red», 640+(100), 320);
  // ...et fixer la couleur à
   celle perçue.
  fill(r, g, b);
} else {
  // Sinon, fixer la couleur noire
  fill(0, 0, 0);
}
// Appliquer la couleur perçue
   dans un cercle
ellipse(640+(50), 300, 40, 40);
```

Vous pouvez expérimenter la précision du MLP en pointant la caméra sur chaque feu de signalisation, sous différents angles et dans des conditions d'éclairage diverses. Essayez également de pointer la caméra sur des objets de votre voisinage qui, selon vous, se rapprochent des couleurs que le MLP a apprises.

Vous remarquerez peut-être la classification erronée d'une large gamme de couleurs comme couleur connue. Dans l'exemple de l'auteur, la classification « vert » est également donnée pour l'environnement du feu de signalisation, le cadre et l'arrière-plan de l'image (**fig. 10**).

Amélioration de la classification des réseaux neuronaux

D'après les valeurs RVB affichées, il est clair que les couleurs transmises au MLP sont raisonnablement proches de la classification souhaitée pour le « vert ». Il existe plusieurs façons de resserrer la classification. La première serait de relever la barre des 90 % pour une classification précise, avec une valeur plus élevée. Une autre approche pourrait être d'augmenter le nombre d'époques d'apprentissage. La dernière consiste à repenser la mise en œuvre de la classification.

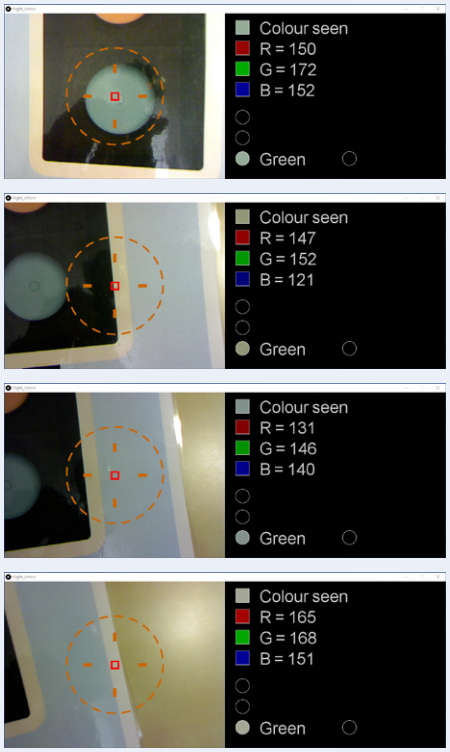

Figure 10. Classification erronée d'autres couleurs dans la catégorie « vert ».

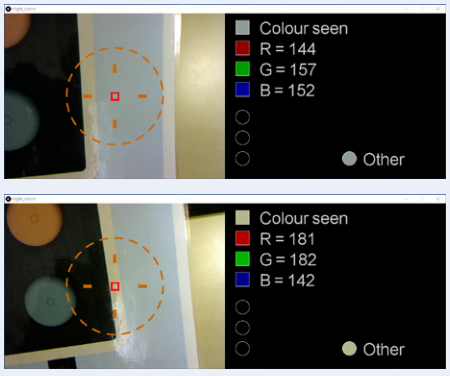

Figure 11. Classification erronée d'autres couleurs dans la catégorie « vert » résolue à l'aide du nœud de sortie « autre ».

Tableau 2. Valeurs RVB capturées pour les couleurs « indésirables ».

Objet	R	G	B
Entourage sombre du feu de signalisation	76	72	35
Bordure blanche	175	167	138
Fond bleu	152	167	161

Jusqu'ici, nous nous sommes concentrés sur ce que nous voulons classer positivement. Cependant, il peut parfois être utile d'apprendre au réseau neuronal ce qui n'appartient pas aux modèles que nous recherchons. Pour l'essentiel, nous pouvons dire : « Voici les trois éléments que nous recherchons, mais voici des éléments similaires que nous ne recherchons absolument pas ». C'est là que notre quatrième nœud de sortie entre en jeu.

En revenant une fois de plus au projet *computer_vision.pde*, nous pouvons capturer les valeurs RVB des couleurs que nous voulons classer comme « autres ». Par exemple, les valeurs RVB de l'entourage sombre du feu de circulation, de la bordure blanche et du fond bleu ont été collectées, avec les résultats présentés dans le **tableau 2**.

Dans le fichier *tlight_detect.pde*, ces valeurs peuvent être apprises comme d'autres couleurs indésirables à l'aide de la méthode `teachOther()`. Il suffit de décommenter le code autour de la ligne 60 comme suit et d'ajouter vos valeurs RVB :

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

Le fait de relancer le projet entraîne une amélioration notable. La zone autour des feux de signalisation (entourage, cadre, arrière-plan) est classée dans la catégorie « autre » (*other*) au lieu de « vert » (**fig. 11**).

Et après ?

Dans cet article, nous avons vu un réseau neuronal résoudre un problème de classification du monde réel. Nous avons également appris qu'il peut être utile d'apprendre à la fois la classification souhaitée et les données de classification non souhaitées.

Pourquoi ne pas essayer d'exploiter cet exemple pour explorer les propositions suivantes ?

- Dans quelle mesure le MLP peut-il être suffisamment « robuste » pour prendre en compte l'angle de la caméra et les changements d'exposition ? Est-il préférable de randomiser davantage les données d'apprentissage ou d'élever le niveau de classification (> 90 %) ?
- La précision s'améliore-t-elle si l'on augmente le nombre de nœuds de sortie de type « autre » et que l'on attribue à chacun une couleur indésirable ?
- Quel est l'impact sur le système de la réduction ou de l'augmentation du nombre de nœuds cachés ?
- Une quatrième entrée pour la « luminosité générale » permettrait-elle d'améliorer la précision de la reconnaissance dans des conditions d'éclairage variables ?

S'il est formidable de faire fonctionner des réseaux neuronaux sur des ordinateurs portables et des PC puissants, beaucoup d'entre nous souhaiteraient disposer d'une telle capacité sur les processeurs plus petits comme ceux d'une carte Arduino. Dans le dernier article de cette série, nous utiliserons un capteur RVB et un Arduino pour mettre en œuvre un réseau neuronal de détection des couleurs. Peut-être sera-t-il la base de votre prochain projet Arduino !

210160-C-04

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (stuart.cording@elektor.com).

Contributeurs

Idée, texte et illustrations : **Stuart Cording**
Rédaction : **Jens Nickel, C. J. Abate**
Illustrations : **Patrick Wielders**
Mise en page : **Harmen Heida**
Traduction : **Pascal Godart**

LIENS

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