GLO-7050 : Apprentissage machine en pratique

Mini-Projet # 3

Remis par

Vincent Gaudreault

111 267 663

Gabriel Rieger Junqueira

111 192 282

Remis le 19 août 2020

Résumé

Dans ce troisième mini-projet, deux modèles de prédiction par analyse d'image sont développés afin de déterminer quel chiffre est présent dans une image MNIST modifiée dans le cadre d'une compétition Kaggle. Chaque image contenant 3 chiffres manuscrits de 0 à 9, l'objectif est d'obtenir le chiffre ayant la valeur numérique la plus élevée. D'une part, l'utilisation d'un VGG-16 modifié a permis d'obtenir les meilleures performances. D'autre part, l'emploi d'un ResNet-50 a donné lieu à des performances légèrement inférieures, mais quand même très intéressantes. Tout de même, la création de données supplémentaires grâce à l'augmentation de données est vraiment avantageuse, puisque le modèle est capable de mieux généraliser. Enfin, l'algorithme proposé permet d'obtenir une précision de 97.1% sur le jeu de données de test.

Introduction

La prédiction par analyse d'images est un problème majeur de la vision par ordinateur sur lequel de nombreux articles ont été écrits [1]. Il existe principalement deux approches radicalement opposées. La première technique consiste à utiliser les outils classiques de vision par ordinateur comme le seuillage d'image, tel que proposé par Gonzalez et Woods, 1977 [2]. L'alternative consiste à simplement prendre avantage des réseaux neuronaux convolutifs. Ainsi, au lieu de chercher à extraire les caractéristiques principales des images, l'emphase est plutôt mise sur le développement d'une architecture permettant d'effectuer cette tâche.

L'approche préconisée dans ce mini-projet consiste évaluer les performances d'un VGG-16 modifié ainsi que d'un ResNet-50 sans effectuer aucun traitement sur les images MNIST [3] modifiées. Ainsi, seule une augmentation des données est effectuée avant d'envoyer le tout aux modèles. Ce faisant, une précision de 97.1% est atteinte par le VGG-16 sur les données de test.

Travaux connexes

L'approche par réseaux de neurones convolutionnels ne date pas d'hier. Introduite par LeCun *et al*., 1998 [4], l'architecture LeNet est capable de reconnaître des caractères manuscrits avec une extrême variabilité et une bonne robustesse aux distorsions. Depuis, de nombreux articles ont été publiés suite à la compétition ILSVRC [5]. Par ailleurs, l'utilisation d'un GPU lors de l'entraînement a permis de créer des modèles avec une profondeur plus élevée, permettant ainsi d'obtenir de meilleurs résultats.

Publié par Krizhevsky *et al*., 2012 [6], AlexNet est un modèle de 5 couches convolutionnelles et 3 couches pleinement connectées permettant de résoudre des problèmes de classification d'images. À la base, il reçoit une image appartenant à une des mille classes, et sa sortie est un vecteur de probabilités. Avec plus de 60 millions de paramètres à entraîner, de l'ajout du *dropout* et de son utilisation de la ReLU, AlexNet est une approche révolutionnaire par rapport à ses concurrents de l'époque.

Développé par Simonyan et Zisserman, 2014 [7], VGGNet est un modèle de 13 couches convolutionnelles[1] et 3 couches pleinement connectées permettant de réduire le nombre

[1] Pour la variante VGG-16. Il existe aussi d'autres configurations comme VGG-11, VGG-13 ou VGG-19.

de paramètres à entraîner par rapport à ses prédécesseurs grâce à son utilisation de noyaux de taille 3x3.

Créé par He *et al*., 2015 [8], ResNet est un réseau neuronal artificiel de 17 couches convolutionnelles[2] et 1 couche pleinement connectée qui tente de trouver des relations plus simples lorsqu'elles existent en introduisant des connexions de raccourci. Quand les réseaux deviennent de plus en plus profonds, les gradients finissent par disparaître et les valeurs des dérivées deviennent insignifiantes lorsqu'elles sont propagées aux couches initiales. C'est d'ailleurs à cette fin que les connexions résiduelles entrent en jeu.

Ensemble de données et configuration

Le jeu de données d'entraînement contient 40 000 images MNIST modifiées où chacune d'entre elles contient 3 chiffres manuscrits de 0 à 9. Un autre fichier contient les étiquettes d'entraînement. L'ensemble de tests est composé de 10 000 images, et aucune étiquette n'est disponible.

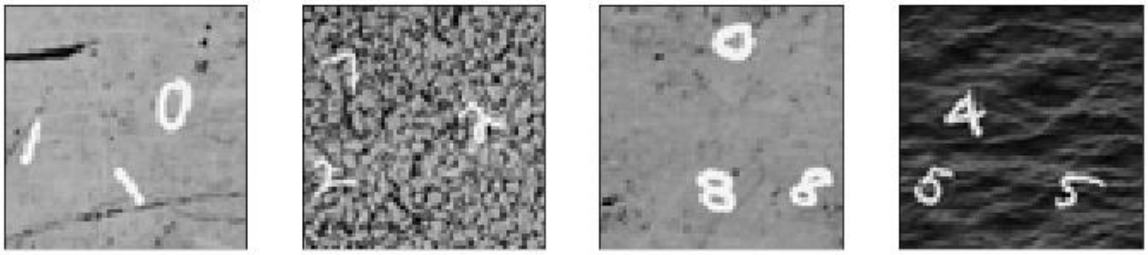

Figure 1 : Exemple d'image d'entraînement

Le fichier d'étiquettes comporte l'identifiant unique de l'image suivi du chiffre ayant la valeur numérique la plus élevée.

0,1 1,7 2,8 3,5

Figure 2 : Exemple d'étiquette d'entraînement

Comme l'approche proposée est un modèle convolutionnel, il n'y a que très peu de prétraitement effectué. Effectivement, les étiquettes d'entraînement sont encodées selon la méthode « one-hot ». Puis, 20% des données du jeu d'entraînement et leurs étiquettes associées sont mises de côté afin qu'elles servent de jeu de validation.

Approche proposée

Afin d'améliorer les performances, une approche d'augmentation des données est effectuée. Ainsi, des rotations allant jusqu'à 30°, des cisaillements et des zooms atteignant 20% de la taille de l'image sont effectués.

Pour le VGG-16, certaines légères modifications ont été apportées selon les recommandations proposées par Qin *et al*. [9]. Ayant un jeu de données semblable au nôtre, leur modèle de prédiction doit être en mesure de fournir le résultat du calcul arithmétique inscrit au sein des images MNIST modifiées. Comme indiqué dans l'article

[2] Pour la variante ResNet-18. Il existe aussi d'autres configurations comme ResNet-34, ResNet-50, ResNet-101 et ResNet-152.

mentionné ci-haut, il est impossible de faire converger le modèle sans ajouter du *dropout* après chaque bloc de convolution. De même, la suppression d'un bloc de 3 couches convolutionnelles se révèle être absolument nécessaire. Ce faisant, le modèle ressemble plutôt à un VGG-13[3].

Pour le ResNet-50, l'architecture est basée sur celle proposée par Priya Dwivedi *et al.* [10]. Initialement, le modèle était conçu pour un jeu de données avec 6 classes et un format d'entrée de 64x64x3. Un changement du nombre de classes en sortie pour 10 a été nécessaire, de même que la modification du format d'entrée par 64x64x1.

Résultats

Basé sur l'expérience acquise avec les travaux pratiques précédents ainsi que la littérature consultée et les notes des cours, il a été décidé d'essayer le SGD [11] pour sa bonne performance reconnue avec des hyperparamètres par défaut. Pour ce qui concerne l'autre choix, Adam [12] est très utilisé lors de l'apprentissage profond. De plus, il est un optimiseur populaire et se présente comme une adaptation améliorée du SGD. Pour avoir le temps de bien exploiter et tester les différentes options, il a été préférable de se concentrer seulement à ces deux optimiseurs.

Les deux meilleurs résultats obtenus ont été 97.1% de précision avec le VGG-16 munit d'un optimiseur SGD et 93.275% avec le ResNet-50 munit d'un optimiseur Adam. Lors de l'entraînement, les modèles ont été entraînés pour 100 itérations d'une taille de lot de 64. Pour éviter le surapprentissage, un processus de « early stopping » avec une patience de 20 époques a été implémenté. Dans certains cas, il a été nécessaire d'incrémenter le nombre d'époques à 200.

En commençant par le VGG-16, nous avons obtenu des améliorations significatives en modifiant l'optimiseur et le taux d'apprentissage, comme le démontre le tableau 1.

Méthode appliquée	**Époque**	**Précision en entraînement**	**Précision en validation**	**Précision en test**
Optimiseur avec Adam	100	0.9478	0.9641	0.96925
Ajout d'une couche convolutionnelle lors du premier bloc	100	0.9370	0.9743	0.96875
Optimiseur avec SGD	100	0.9092	0.9575	-
Modification du taux d'apprentissage à 0.005	137	0.9380	0.9622	-
Ajout du « decay »	80	0.9242	0.9585	-
Modification de la taille de lot à 64	102	0.9272	0.9688	0.97100

[3] Étant donné que ce modèle n'existe pas au sein des variantes de VGGNet, il a été préférable de laisser le nom de notre réseau à VGG-16.

Modification des plages de valeur de l'augmentation des données	160	0.9094	0.9632	0.96968

Tableau 1 : Résultat des différentes modifications apportées au VGG-16

En ce qui concerne le ResNet-50, la stratégie a été différente. Dans un premier moment, nous avons essayé des tailles de lot plus petites, sans succès en raison du temps d'entraînement élevé. Dans un deuxième moment, nous nous sommes concentrés aux hyperparamètres de la fonction de génération de données, soit l'augmentation des données.

Méthode appliquée	**Époque**	**Précision en entraînement**	**Précision en validation**	**Précision en test**
Modification de la taille de lot à 64	100	0.9122	0.9211	0.93275
Modification de la plage de valeurs pour le zoom à 0.4	100	0.9011	0.9113	0.92275
Modification de la taille de lot à 4	100	0.8697	0.8787	0.88025
Modification de la rotation pour 90°	100	0.8881	0.8902	0.90200
Modification de la rotation pour 180°	100	0.7345	0.7444	0.76300

Tableau 2 : Résultat des différentes modifications apportées au ResNet-50

Enfin, l'utilisation d'un processeur graphique est nettement avantageuse au niveau du temps d'entraînement comme le démontre le tableau 3.

Type d'exécution	**Modèle**	**Temps d'exécution (s)**
CPU	VGG-13	720
GPU	VGG-13	30
CPU	Resnet-50	800
GPU	Resnet-50	45

Tableau 3 : Comparaison des temps d'exécution des modèles

Discussion et conclusion

Lors de ce mini-projet, deux modèles ont été implémentés dans le but de classifier des images MNIST modifiées. Le VGG-16 est celui ayant obtenu les meilleures performances lors de la compétition Kaggle, soit 97.1% sur l'ensemble de tests. Le ResNet-50 a tout de même obtenu un score de 93.275% en précision, ce qui n'est pas négligeable. L'augmentation des données par rotation, cisaillement et zoom est la technique qui a permis d'augmenter de façon significative les performances.

Tout de même, il aurait été intéressant d'essayer d'autres techniques telles que la modification de contraste, la symétrie miroir ou le redimensionnement. L'implémentation d'un VGG-5 [13] pourrait aussi permettre d'accroître les performances, considérant un score de 99.72% en précision sur le jeu de données MNIST.

État des contributions

Les deux membres de l'équipe ont contribué à faire les recherches nécessaires et à organiser une séance de remue-méninges afin de décider de la marche à suivre. Vincent s'est occupé de l'implémentation du VGG-16 et Gabriel de l'implémentation du ResNet-50. De ce fait, le travail dans ce rapport est le fruit d'une collaboration équitable de tous les membres de l'équipe.

Bibliographie

[1] « Papers With Code - Image Classification », *Papers With Code*, [En ligne], 2015, https://paperswithcode.com/task/image-classification.

[2] Rafael C. GONZALEZ et Richard E. WOODS, 1977, *Digital Image Processing*, New Jersey, Prentice Hall.

[3] Yann LECUN *et al*., « The MNIST Database of handwritten digits », [En ligne], 1998, http://yann.lecun.com/exdb/mnist/.

[4] Yann LECUN *et al*., « Gradient-Based Learning Applied to Document Recognition », [En ligne], 1998, http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

[5] Stanford Vision Lab, « Large Scale Visual Recognition Challenge », *ImageNet*, [En ligne], 2015, http://www.image-net.org/challenges/LSVRC/.

[6] Alex KRIZHEVSKY *et al*., « ImageNet Classification with Deep Convolutional Neural Networks », *arXiv*, [En ligne], 2012, https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[7] Karen SIMONYAN et Andrew ZISSERMAN, « Very Deep Convolutional Networks for Large-Scale Image Recognition », *arXiv*, [En ligne], 2014, https://arxiv.org/pdf/1409.1556.pdf.

[8] Kaiming HE *et al*., « Deep Residual Learning for Image Recognition », *arXiv*, [En ligne], 2015, https://arxiv.org/pdf/1512.03385.pdf.

[9] Chuan QIN *et al*., « Image Classification: Modified MNIST », *GitHub*, [En ligne], 2017, https://github.com/Bonnie970/Applied_Machine_Learning/blob/master/Assignment3/Machine_Learning_project_3_Report.pdf.

[10] Priya DWIVEDI *et al*. « Residual Network Keras », *GitHub*, [En ligne], 2019, https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Network_Keras.ipynb.

[11] « Optimizers - SGD», *TensorFlow*, [En ligne], 2015, https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD.

[12] « Optimizers - Adam », *TensorFlow*, [En ligne], 2015, https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[13] H M Dipu KABIR *et al*., « SpinalNet: Deep Neural Network with Gradual Input », *arXiv*, [En ligne], 2020, https://arxiv.org/pdf/2007.03347.pdf.