FACULTÉ DES SCIENCES INFORMATIQUES
Calcul Scientifique

Minimisation d'erreurs cas d'invariance des transformations géométriques dans la reconnaissance des plaques d'immatriculation

---

Travail de fin d'études présenté et défendu en vue de l'obtention du grade de licencié en Calcul Scientifique.

Auteur : **TSHELEKA KAJILA Hassan**
Directeur : **Prof. MASAKUNA Jordan**

12 juillet 2022

RÉSUMÉ

TABLE DES MATIÈRES

LISTE DES ACRONYMES

IA Intelligence Artificielle

AI Artificial Intelligence

SA Supervised Learning

ML Machine Learning

CV Computer Vision

OCR Optical Character Recognition

ANPR Automatic Number-Plate Recognition

ALPR Automatic License Plate Recognition

GD Gradient Descent

SGD Stochastic Gradient Descent

CNN Convolutional Neural Network

ANN Artificial Neural Network

VGG Visual Geometry Group

ILSVRC ImageNet Large Scale Visual Recognition Challenge

NAG Nesterov Accelerated Gradient

Adam Adaptive Moment Estimation

Nadam Nesterov-accelerated Adaptive Moment Estimation

ReLu Rectified Linear Unit

MSE Mean Squared Error

MLP MultiLayer perceptron

SLNN Single-Layer Neural Network

$\mathbb{N}$	Ensemble des entiers naturels
$\mathbb{R}^n$	Ensemble des réels ou Espace euclidien de dimension n
$\mathbb{B}^n = \{0,1\}^n$	Espace booléen de dimension n
$\mathcal{O}(\cdot)$ ou $\Omega(\cdot)$	L'ordre de grandeur maximal de complexité d'un algorithme
$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$	Un vecteur
$x = (x_1, \vdots, x_n)^T$	Un vecteur
$x^T = (x_1, \vdots, x_n)^T$	Un vecteur transposé
$\langle xy \rangle = x^T y$	Le produit vectoriel
$\parallel x \parallel$	La norme du vecteur
M^{-1}	La matrice inverse d'une matrice M
M^T	La matrice transposée
$\frac{\partial}{\partial x} f(x,y)$	La dérivée partielle par rapport à x de al fonction f des deux variable x et y
$\nabla_A J(A,B)$	Le vecteur dérivé par rapport au vecteur A de la fonctionnelle J des deux vecteurs A et B

Les éléments en apprentissage

$\mathcal{S}$	L'échantillon d'apprentissage (un ensemble ou une suite d'exemple)
$\hat{y}$	La valeurs prédite après l'entrainement d'un modèle d'apprentissage automatique
$\hat{y}_i \in \mathcal{Y}$	La prédiction, ou sortie désirée, d'un exemple
$\mathcal{H}$	Espace des hypothèses d'apprentissages
$h \in \mathcal{H}$	Une hypothèses produite par un algorithme d'apprentissage
$y = h(x) \in \mathcal{Y}$	La prédiction faite par l'hypothèse h sur la description s d'un exemple
$\ell(f(x), h(x))$	La fonction perte ou fonction coût entre la fonction cible et une hypothèse sur x d'un exemple
$\mathcal{C}$	L'ensemble des classes
C	Le nombre de classes
$c_i \in \mathcal{C}$	Une sous classe de $\mathcal{C}$

0 RÉSULTATS ET DISCUSSION

0.1 EXPÉRIMENTATION

0.1.1 *Outils, Choix des technologies*

A *Language & technologie*

Le langage de programmation et technologie utilisée

Python : est le language majeur utilisé dans ce projet, c'est un langage de programmation interprété de haut niveau. Sa philosophie de conception met l'accent sur la lisibilité du code avec l'utilisation d'une indentation significative. Python est typé dynamiquement et ramassé.

TensorFlow : c'est une bibliothèque de logiciels gratuite et open source pour l'apprentissage automatique et l'intelligence artificielle. TensorFlow fournit un ensemble de workflows pour développer et former des modèles à l'aide de Python ou JavaScript. Il peut être utilisé dans une gamme de tâches, mais se concentre particulièrement sur la formation et l'inférence des réseaux de neurones profonds.

OpenCV : est une bibliothèque de fonctions de programmation principalement destinées à la vision par ordinateur en temps réel. OpenCV fournit une bibliothèque, des outils et du matériel de vision par ordinateur optimisés en temps réel. Il prend également en charge l'exécution de modèles pour Machine Learning.

LabelImg : est un outil gratuit et open source pour l'étiquetage graphique, un outil graphique d'annotation d'images.
Il est écrit en Python et utilise Qt pour son interface graphique. Les annotations sont enregistrées sous forme de fichiers XML au format PASCAL VOC, le format utilisé par ImageNet. Il est écrit en Python et utilise QT pour son interface graphique. C'est un moyen facile et gratuit d'étiqueter quelques centaines d'images pour notre projet de détection d'objets. En outre, il prend également en charge les formats YOLO et CreateML.

B *choix du modèle de CNN*

Le VGGNet (Visual Geometry Group), Le modèle VGG16 du VGGNet atteint près de 92,7 % de précision dans le top 5 des tests dans ImageNet. le VGGNet-16 prend en charge 16 couches (tout comme le VGG19 prend en charge 19 couches.) et peut classer les images en 1000 catégories d'objets,de plusieure catégorie de plus, le modèle a une taille d'entrée d'image de 224 par 224 (*cf.* le chapitre **??**, section **??**, point **??**).

VGG16 surpasse largement les versions précédentes des modèles des compétitions ILSVRC-2012 et ILSVRC-2013. De plus, le résultat VGG16 est en compétition pour le

vainqueur de la tâche de classification (GoogLeNet avec une erreur de 6,7%) et surpasse considérablement la soumission gagnante ILSVRC-2013 Clarifai. Il a obtenu 11,2% avec des données de formation externes et environ 11,7% sans elles. En termes de performances à réseau unique, le modèle VGGNet-16 obtient le meilleur résultat avec environ 7,0% d'erreur de test, dépassant ainsi un seul GoogLeNet d'environ 0,9% [29]

	Training accuracy	**Validation accuracy**
Réseau neuronal convolutif de base	`98.20%`	`72.40%`
Réglage fin de CNN avec augmentation d'image	`81.30%`	`79.20%`
Réglage fin de CNN avec modèle VGG-16 pré-entraîné et augmentation d'image	`86.50%`	`95.40%`

Le tableau ci-dessus montre la précision de la formation et de la validation (training accuracy & validation accuracy) pour différents modèles de réseaux neuronaux, résultats comparatifs de [30].

Nous avons aussi fait un test comparatif entre VGG-19 et VGG-16 pour savoir quel modèle est plus adapté à notre cas de reconnaissance des plaques d'immatriculation. Le résultats comparatifs dans le tableau ci-dessous, qui nous montre le nombre total de paramètres, le nombre de paramètres entrainables, le nombre de paramètres non-entrainables, la perte, la précision (respectivement : total params, trainable params, non-trainable params, loss, accuracy).

Message de sortie du Modèle VGG19	
Total params : 24,784,644	loss : 4.0037e-04
Trainable params : 4,760,260	accuracy : 0.9636
Non-trainable params : 20,024,384	validation loss : 0.0109
	validation accuracy : 0.7727
Message de sortie du Modèle VGG16	
Total params : 19,474,948	loss : 3.4066e-04
Trainable params : 4,760,260	accuracy : 0.9570
Non-trainable params : 14,714,688	validation loss : 0.0102
	validation accuracy : 0.8182

Peut importe le nombre de couches, 19 ou 16, le deux modèle génère le même nombre de paramètres entrainable (donc minimisable) du coup il y a rien d'intéressant à prendre le VGG-19 vu que sa précision de validation est faible comparé au modèle VGG-16 et qu'il n'y a pas un grand écart entre leurs précision et d'entraînement.

0.1.2 *Élaboration du dataset*

C'est une tâche, fastidieuse dans le l'apprentissage automatique, qui consiste à montrer à la machine où se trouve la plaque d'immatriculation (la zone précise) sur l'image. Ce

processus aussi nommé étiquetage est très important dans l'étape de l'apprentissage supervisé. Et il faut faire ça pour toute 430 images qui constituent notre Dataset.

Dataset est construit avec l'outil **LabelImg** est un outil graphique d'annotation d'images. Le Dataset est un répertoire composé des sous répertoire qui contient :

- Les annotations sont des fichiers xml, qui contient les coordonnées des régions intérêt sur les images (les étiquettes).
- Les differentes images des plaques d'immatriculations.

Nous avons 433 images des plaques d'immatriculations pour 433 fichiers xml d'annotations. Le dataset est fractionné en trois sous ensemble proportionnellement : 70% pour l'entrainement, 10% pour la validation, 20% pour le teste.

FIGURE 1 : Exemple des quelques images de notre dataset sans annotations

FIGURE 2 : Exemple des images (figure 1) de notre dataset sans annotations

0.1.3 *Entraînement du CNN (VGG)*

A *Élaboration du modèle*

```
import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Dropout
from keras.applications.vgg19 import VGG19
from keras.applications.vgg16 import VGG16

IMAGE_SIZE = 200

model = Sequential()

model.add(VGG16(
        weights="imagenet",
        include_top=False,
        input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
)
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(256, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(64,  activation="relu"))
model.add(Dense(4,   activation="sigmoid"))
model.layers[-7].trainable = False

model.summary()
```

Ci-dessus se trouve le code pour construire un modèle pré-entraîné VGG-16. Nous devons inclure $weights = \text{"}imagenet\text{"}$ pour récupérer le modèle VGG-16 qui est formé sur l'ensemble de données imagenet. Il est important de définir $include\_top = False$ pour éviter de télécharger les couches entièrement connectées du modèle pré-entraîné [30]. Nous devons ajouter notre propre classificateur car le classificateur de modèle pré-entraîné a plus de 1 classe alors que notre objectif est de classer l'image en 1 classe (image de plaques). Une fois que les couches convolutionnelles du modèle pré-entraîné ont extrait les caractéristiques d'image de bas niveau telles que les bords, les lignes et les blobs, la couche entièrement connectée les classe ensuite en 1 catégories.

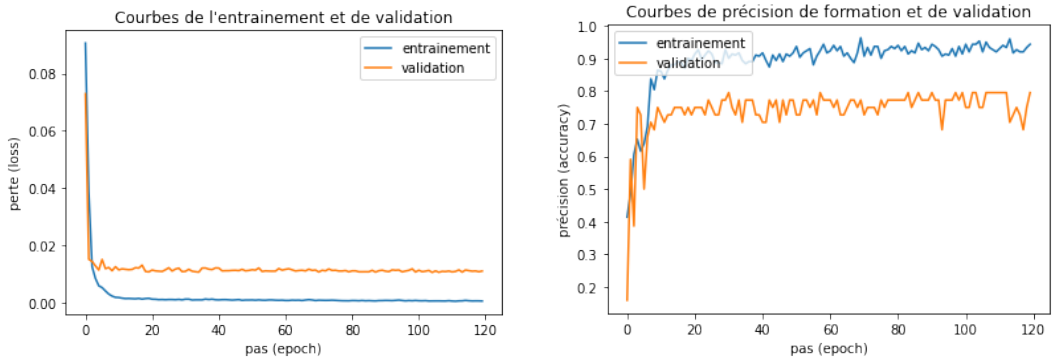
Nous avons crée un modèle **"Séquentiel"** et ajoutons **VGG16** comme première couche. Le modèle prendra en entrée des tableaux de taille $(200 \times 200 \times 3)$ qui correspondent à la taille d'une image du dataset. La sortie du VGG16 passe dans **Flatten** une couche qui a aplatis le tableau $(200 \times 200 \times 3)$ en un tableau de taille (18432×1). Enfin nous ajoutons 4 couches **Dense** qui feront office d'un réseau complètement connecté avec comme sortie un tableau de taille (4×1) qui correspond au 4 points qui nous permettront d'obtenir ue rectangle contenant la plaque d'immatriculation sur l'image.

Ci-dessous le message de sortie, les informations sur notre reseaux de neurones construit.

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 6, 6, 512)	14714688
flatten (Flatten)	(None, 18432)	0
dropout (Dropout)	(None, 18432)	0
dense (Dense)	(None, 256)	4718848
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 4)	260

Total params: 19,474,948
Trainable params: 4,760,260
Non-trainable params: 14,714,688

Il est difficile de trouver un critère général pour arrêter cet algorithme. Le problème est que l'erreur tend à diminuer lentement et à ne jamais se stabiliser complètement, ce qui mène à un surapprentissage. La meilleure manière d'éviter ce phénomène est d'utiliser un ensemble de validation.



(a) loss : perte (b) accuracy : précision

FIGURE 3 : Graphe de précision (accuracy) et perte (loss) du modèle VGG

Le script ci-dessous est une partie du programme que nous avons élaboré pour l'entraînement du CNN. Ici nous entraînons avec l'optimiseur SGD et un taux d'apprentissage de $1e-4 = 1.10^{-4}$. Dans la section 0.2.1 nous montrons les résultats avec différents optimiseurs.

```
import keras.optimizers as optimizers
import keras.losses as losses
optimizer = optimizers.SGD(learning_rate=1e-4)
loss = losses.MeanSquaredError(reduction="auto", name="mean_squared_error")
model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])
model.fit(X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=200,
        batch_size=32,
        verbose=1
)
```

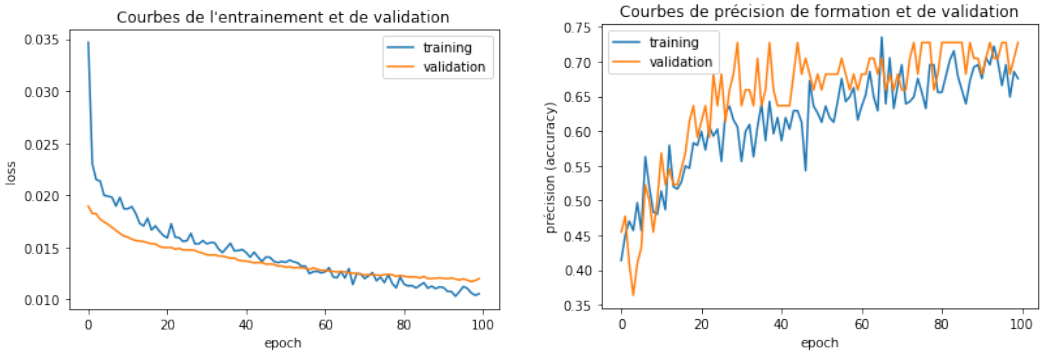
0.2 ÉVALUATION DU RÉSULTATS D'EXPÉRIMENTATION

0.2.1 *Résultat comparatif des différent optimiseurs*

Pour le même modèle élaboré pour la reconnaissance de plaque d'immatriculation. Nous allons tester son efficacité après entraînement en fonction des différents optimiseurs. Nous évaluons les quelques optimiseurs candidats pour cette études (*cf.* chapitre **??**, section **??**). Voici une liste non exhaustive des optimiseurs étudiés.

- *SGD*

Training	**Validation**	**Test**
`loss: 0.0124`	`val loss: 0.0156`	`test loss: 0.0092`
`accuracy: 0.6291`	`val accuracy: 0.6591`	`test accuracy: 0.5946`



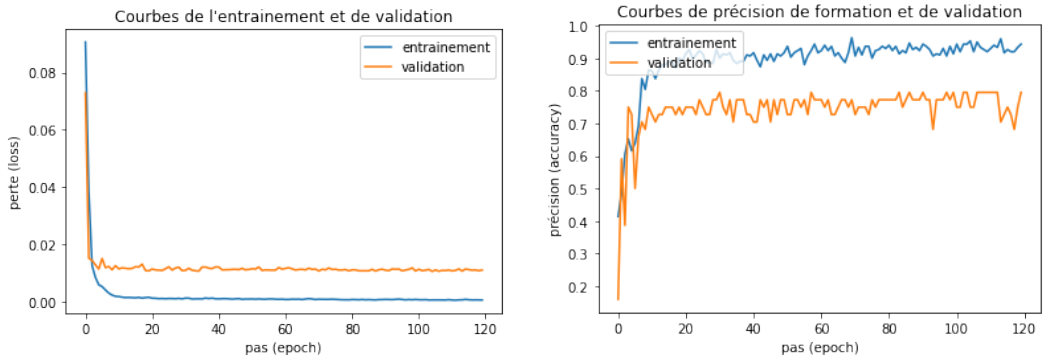
(a) loss : perte

(b) accuracy : précision

FIGURE 4 : Graphe de précision (accuracy) et perte (loss) pour RMSprop

• *RMSprop*

Training	Validation	Test
`loss: 7.0379e-04`	`val loss: 0.0109`	`test loss : 0.005158`
`accuracy: 0.9470`	`val accuracy: 0.7727`	`test accuracy : 0.86206`

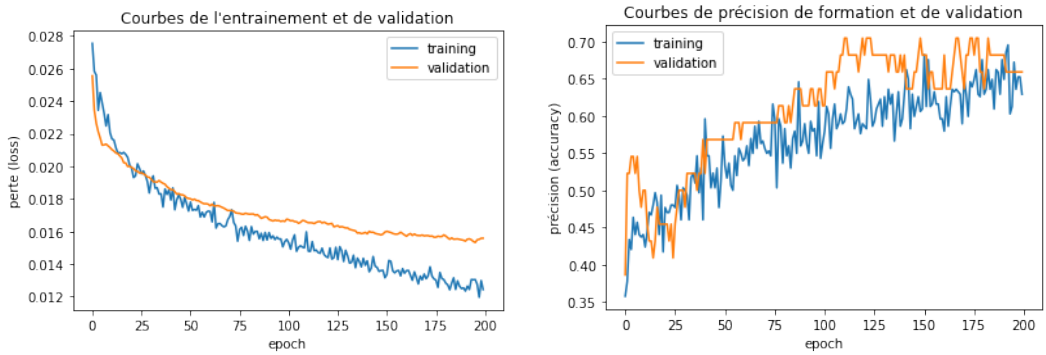


(a) loss : perte (b) accuracy : précision

FIGURE 5 : Graphe de précision (accuracy) et perte (loss) pour RMSprop

• *Adagrad*

Training	Validation	Test
`loss: 0.0124`	`val loss: 0.0156`	`test loss: 0.0092`
`accuracy: 0.6291`	`val accuracy: 0.6591`	`test accuracy: 0.5946`

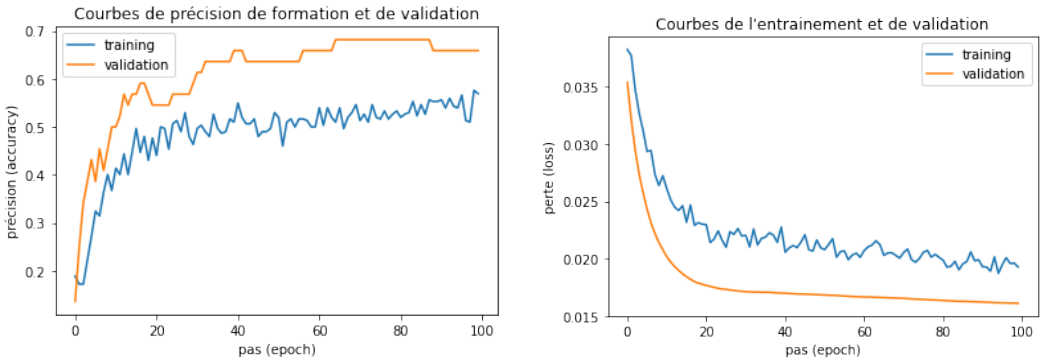


(a) loss : perte (b) accuracy : précision

FIGURE 6 : Graphe de précision (accuracy) et perte (loss) pour Adagrad

• *Adadelta*

Training	Validation	Test
`loss: 0.0130`	`val loss: 0.0156`	`test loss : 0.01374378`
`accuracy: 0.6523`	`val accuracy: 0.6591`	`test accuracy : 0.54022`

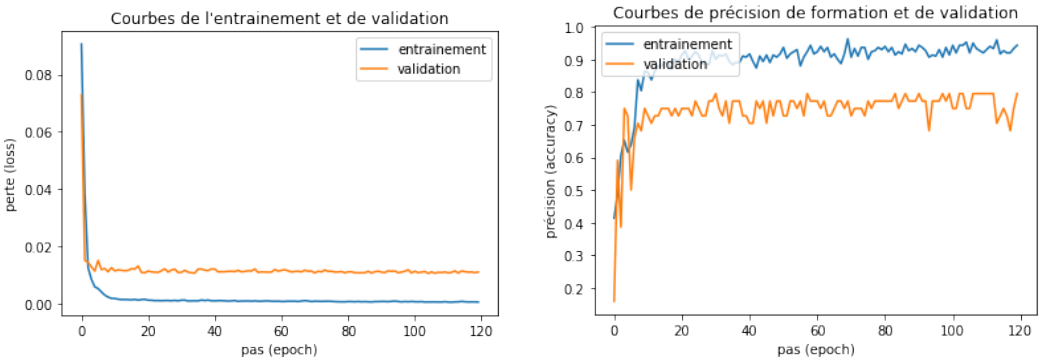


(a) loss : perte (b) accuracy : précision

FIGURE 7 : Graphe de précision (accuracy) et perte (loss) pour Adadelta

• *Adam*

Training	Validation	Test
`loss: 4.0037e-04`	`val loss: 0.0109`	`test loss : 0.00407`
`accuracy: 0.9636`	`val accuracy: 0.7727`	`test accuracy : 0.88505`

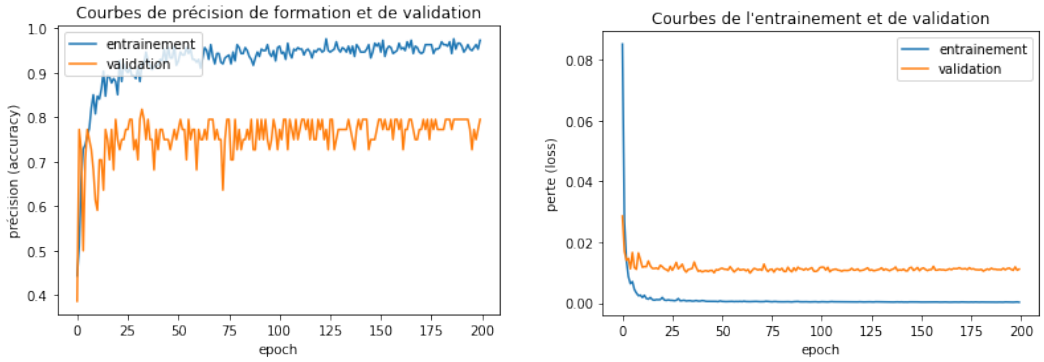


(a) loss : perte (b) accuracy : précision

FIGURE 8 : Graphe de précision (accuracy) et perte (loss) pour Adam

- *Nadam*

Training	Validation	Test
`loss: 3.0496e-04`	`val loss: 0.0111`	`test loss : 0.004073061`
`accuracy: 0.9536`	`val accuracy: 0.7955`	`test accuracy : 0.87356`



(a) loss : perte (b) accuracy : précision

FIGURE 9 : Graphe de précision (accuracy) et perte (loss) pour Nadam

Nous remarquons RMSprop minimise le mieux les erreurs par rapport aux autres optimiseurs mais il n'est pas le plus précis non plus. Ci-dessous une liste des optimiseurs qui minimise le mieux dans le cas de notre problème reconnaissance de plaque d'immatriculation.

N°	Training	Validation	Test
1. RMSprop	`7.0379e-04`	`0.0109`	`0.005158`
2. Nadam	`3.0496e-04`	`0.0111`	`0.004073`
3. Adam	`4.0037e-04`	`0.0109`	`0.00407`
4. Adagrad	`0.0124`	`0.0156`	`0.0092`
5. Adadelta	`0.0130`	`0.0156`	`0.013743`

0.2.2 *Invariance des transformation géométrique*

Nous avons dit en amont que le modèle entraîné doit reconnaître les objet même après transformation géométrique dans l'image. Nous allons examiner l'efficacité par rapport au 3 meilleur optimiseurs c.a.d les 3 modèle qui ont on eu un bon score d'entraînement, de validation et de test.

	Training	Validation	Test
Nadam	`95.36%`	`79.55%`	`87.35%`
Adam	`96.36%`	`77.27%`	`88.50%`
RMSprop	`94.70%`	`77.27%`	`86.20%`

??? (Suite)

BIBLIOGRAPHIE

[1] Yaovi Ahadjitse. "Reconnaissance d'objets en mouvement dans la vidéo par description géométrique et apprentissage supervisé". Thèse de doct. Université du Québec en Outaouais, 2013.

[2] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2010. isbn : 9780262012430.

[3] Vincent Barra Antoine Cornuéjols Laurent Michet. *Apprentissage artificiel : Deep leaning, concepts et algorithmes*. 3rd. Eyrolles, 2018, p. 239-263.

[4] P. Benner, M. Bollhöfer, D. Kressner, C. Mehl et T. Stykel. *Numerical Algebra, Matrix Theory, Differential-Algebraic Equations and Control Theory : Festschrift in Honor of Volker Mehrmann*. Springer International Publishing, 2015. isbn : 9783319152608. url : https://books.google.cd/books?id=MilACQAAQBAJ.

[5] M. Bierlaire. *Introduction à l'optimisation différentiable*. Enseignement des mathématiques. Presses polytechniques et universitaires romandes, 2006. isbn : 9782880746698. url : https://books.google.cd/books?id=HK55T5x2bygC.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. First. Springer-Verlag New York, 2006, p. 179-195.

[7] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In : *Proceedings of COMPSTAT'2010*. Springer, 2010, p. 177-186.

[8] Léon Bottou. "Stochastic gradient descent tricks". In : *Neural networks : Tricks of the trade*. Springer, 2012, p. 421-436.

[9] Léon Bottou, Frank E Curtis et Jorge Nocedal. "Optimization methods for large-scale machine learning". In : *Siam Review* 60.2 (2018), p. 223-311.

[10] F. Coulombeau, G. Debeaumarché, B. David, F. Dorra, S. Dupont et M. Hochart. *Mathématiques MPSI-PCSI : Programme 2013 avec algorithmique en Scilab*. Cap Prépa. Pearson, 2013. isbn : 9782744076527.

[11] Pádraig Cunningham, Matthieu Cord et Sarah Jane Delany. "Supervised learning". In : *Machine learning techniques for multimedia*. Springer, 2008, p. 21-49.

[12] R.B. Darlington et A.F. Hayes. *Regression Analysis and Linear Models : Concepts, Applications, and Implementation*. Methodology in the Social Sciences. Guilford Publications, 2016. isbn : 9781462521135.

[13] Natarajan Deepa, B Prabadevi, Praveen Kumar Maddikunta, Thippa Reddy Gadekallu, Thar Baker, M Ajmal Khan et Usman Tariq. "An AI-based intelligent system for healthcare analysis using Ridge-Adaline Stochastic Gradient Descent Classifier". In : *The Journal of Supercomputing* 77 (2021), p. 1998-2017.

[14] Kary Främling. "Scaled Gradient Descent Learning Rate". In : *Reinforcement Learning With Light-Seeking Robot, Proceedings of ICINCO* (2004), p. 1-8.

[15] Yoav Freund et Robert E Schapire. "Large margin classification using the perceptron algorithm". In : *Machine learning* 37.3 (1999), p. 277-296.

[16] A. Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. ISBN : 9781491962299.

[17] I. Goodfellow, Y. Bengio et A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN : 9780262035613.

[18] F.E. Harrell et F.E.H. jrl. *Regression Modeling Strategies : With Applications to Linear Models, Logistic Regression, and Survival Analysis*. Graduate Texts in Mathematics. Springer, 2001. ISBN : 9780387952321. URL : https://books.google.cd/books?id=kfHrF-bVcvQC.

[19] Daniel Kirsch Judith Hurwitz. *Machine Learning For Dummies*. IBM Limited Edition. John Wiley et Sons, Inc., 2018.

[20] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In : *Advances in neural information processing systems* 25 (2012).

[21] N. Matloff. *Statistical Regression and Classification : From Linear Models to Machine Learning*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2017. ISBN : 9781351645898.

[22] Praneeth Netrapalli. "Stochastic gradient descent and its variants in machine learning". In : *Journal of the Indian Institute of Science* 99.2 (2019), p. 201-213.

[23] Jorge Nocedal et Stephen J Wright. *Numerical optimization*. T. 35. 1999.

[24] Arnu Pretorius, Elan Van Biljon, Steve Kroon et Herman Kamper. "Critical initialisation for deep signal propagation in noisy rectifier neural networks". In : *Advances in Neural Information Processing Systems* 31 (2018).

[25] D. Sarkar, R. Bali et T. Sharma. *Practical Machine Learning with Python : A Problem-Solver's Guide to Building Real-World Intelligent Systems*. Apress, 2017. ISBN : 9781484232071.

[26] Carl-Erik Särndal, Bengt Swensson et Jan Wretman. *Model assisted survey sampling*. Springer Science & Business Media, 2003.

[27] Vahid Mirjalili Sebastien Raschka. *Python Machine Learning and Deep Learning, with sckit-learn and Tensorflow*. 2nd. Packt, 2017, p. 17-139.

[28] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura et Ronald M Summers. "Deep convolutional neural networks for computer-aided detection : CNN architectures, dataset characteristics and transfer learning". In : *IEEE transactions on medical imaging* 35.5 (2016), p. 1285-1298.

[29] Karen Simonyan et Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In : *arXiv preprint arXiv :1409.1556* (2014).

[30] Srikanth Tammina. "Transfer learning using VGG-16 with deep convolutional neural network for classifying images". In : *International Journal of Scientific and Research Publications (IJSRP)* 9.10 (2019), p. 143-150.

[31] Rob GJ Wijnhoven et PHN de With. "Fast training of object detection using stochastic gradient descent". In : *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, p. 424-427.

[32] Wei Yu, Kuiyuan Yang, Yalong Bai, Tianjun Xiao, Hongxun Yao et Yong Rui. "Visualizing and comparing AlexNet and VGG using deconvolutional layers". In : *Proceedings of the 33 rd International Conference on Machine Learning*. 2016.