Study of the best stochastic gradient descent optimizers for automatic license plate recognition

Hassan T. Kajila[1], Pascal Sungu, Godwill Ilunga[1], August Muhau[2], and Jordan Masakuna[3]

[1] Université Nouveaux Horizons, Faculté des Sciences Informatiques, Lubumbashi, DR of Congo
[2] Université Nouveaux Horizons, Département des Sciences de base, Lubumbashi, DR of Congo
home page: `https://www.unhorizons.org/`
[3] Université de Kinshasa, Département des Sciences Informatiques, Kinshasa, DR of Congo
home page: `https://www.unikin.ac.cd/`

Abstract. Nowadays, a lot of cameras are deployed exclusively for surveillance. This makes content tracking and analysis so expensive, not to mention the mistakes that human fatigue and carelessness can introduce. In this work we propose a method, based on artificial intelligence, which allows computer systems to derive meaningful information from digital images. As part of this work, the aim is to perform automatic license plate recognition using optimizers based on the *stochastic gradient descent*. Therefore, the main objective of this present study is to examine the different optimizers and determine the best able to recognize license plates, in spite of possible geometric transformations, in order to reduce the constraints in the detection of license plates as to the position of the camera. Experiments on several license plates made it possible to validate the performance obtained with this approach.

Key words: Supervised learning, Stochastic optimization methods, Stochastic gradient descent, Visual Geometry Group (VGG), Automatic License Plate Recognition, Geometric transformations.

1 Introduction

The first untrained model built in supervised learning to perform a certain task usually does not yield an accurate result compared to the output or task it was supposed to do [Goodfellow et al., 2016, Géron, 2017]. The loss function (also called cost function) is a way to measure whether the model better schematizes the system it's supposed to describe. This is necessary to determine the deviation, or learning error, between the current output of model and its expected output [Antoine Cornuéjols, 2018]. For example for classification tasks, a learning error for a binary SVM (Support Vector Machine) classifier might be reported, although the loss function used to train the SVM only very loosely models the number of errors on the training set, and similarly neural net training uses convex costs, such as mean square error or cross-entropy [Burges et al., 2006]. Thus often in machine learning tasks, there are actually two cost functions: the desired cost, and the one used in the optimization process.

Minimizing errors or optimizing weight parameters for a neural network is an important task in the learning process. The training dataset for object detection problems are typically very large and the capabilities of machine learning methods are limited by computation time rather than sample size. For deep convolutional networks trained in a supervised setting, the training objective is typically the minimization of learning error (e. g. classification error see [Sebastien Raschka, 2017, chap. 3]) at the top network layer.
Many neural network training algorithms explicitly minimize a loss function. For instance, the back-propagation technique uses the gradient descent algorithm to minimize the mean squared error (MSE see [Bosman et al., 2020]) loss function. Deep convolutional neural networks trained

using back-propagation are thus achieving record performance in a variety of large-scale computer vision tasks [Krizhevsky et al., 2012, Simonyan and Zisserman, 2014, Huang et al., 2017].

This paper deals with the use of numerical optimization algorithms, precisely the minimization of errors. They will be applied to machine learning that will enable computer systems to automatically recognize vehicle license plates. This work will be done using optimizers focused on stochastic gradient descent to optimize the parameters of the convolutional neural network (CNN). Therefore for each algorithm or optimizer, their efficiency will be examined and compared with each other, and classified according to the score (accuracy, loss) obtained.

We discuss the stochastic gradient descent (SGD) method in section 2. We describe the proposed CNN dataset and model built with the VGG architecture in section 3 and quantitatively evaluate the performance of different SGD-based optimizers. And also we present the results of the application in automatic license plate recognition in section 4. We present our findings and an in-depth discussion on the performance of the best optimizers in section 5.

2 Minimization by the stochastic descent gradient

The loss function, as mentioned in the introduction, is a way to measure whether the algorithm has done a good job. Let the following e.g (from [Bottou et al., 1991]), Consider a function, $J(z, w)$, that measures the cost incurred by a system defined by some parameter w processing an observation z. In the case of connectionist learning, this function is equal to $(y - f(x, w))^2$ (with $f(x, w) = w^T x$) and measures how well the output y for pattern x is approached. Learning consists of finding the parameter $w*$ that minimizes the following cost:

$$\begin{aligned} Q(w) &= \langle (y - w^T x)^2 \rangle = \langle (y - f(x, w))^2 \rangle \\ Q(w) &= \ell(J(z, w)) = \int J(z, w) dP(z) \end{aligned} \tag{1}$$

dP is an unknown probability distribution which characterizes the problem to learn, and J, the loss function, describes the learning system itself.

Gradient descent refers to a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. It performs a minimization optimization by following the negative of the downward gradient of the loss function to locate the minimum thereof. This algorithm requires a starting point in the problem, such as a randomly selected point in the input space [Bottou, 2012]. The gradient descent will find the minimum of the loss function $\ell \in \mathbb{R}$ starting from the random parameters w_ia which will be the initial coordinates.

The optimizers based on gradient stochastic are more used in neural network weights optimization [Ruder, 2016]. Gradient descent takes the loss function $\ell = Q(w)$ parameterized by the parameters of the model $w \in \mathbb{R}^d$ and minimizes it by updating the parameters in the opposite direction to the gradient of the loss function. It's a way to convert add $\nabla_w Q(w)$ to the parameters. The learning rate η determines the step size required to reach the local minimum. In other words, it follows the direction of the slope of the surface produced by the downward loss function until it reaches a minimum [Bottou et al., 1991, Ruder, 2016]. Minimizing the cost, see equation 1, apparently is not a difficult problem. If the training set is finite, the distribution $dP(z)$ is discrete, and we can explicitly apply the gradient descent algorithm. Each iteration involves a sum over the N examples z_i. Let the following equation:

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla_w Q(w_t) \\ &= w_t - \eta_t \int \nabla_w J(z, w_t) dP(z) \\ &= w_t - \eta_t \frac{1}{N} \sum_{i=1}^{N} \nabla_w J(z_i, w_t) \end{aligned} \tag{2}$$

A stochastic process $w_t; t = 1, ...N$ depends on the randomly selected sample at each iteration. The stochastic gradient descent directly optimizes the expected risk, since the examples are randomly drawn from the ground truth distribution [Bottou, 2010]. However, stochastic gradient descent algorithm have proven to be faster, more reliable, and less likely to hit bad local minima than standard gradient descent methods [Wijnhoven and de With, 2010]. The algorithm updates the weights according to the gradient of the loss function after each example is presented [Bottou et al., 1991].

SGD has difficulty navigating through cost function valleys, i.e. areas where the surface curves much more steeply in one dimension than another, which are common around local optima [Antoine Cornuéjols, 2018]. In these scenarios, SGD oscillates on the slopes of the valley while making only a hesitant progression along the bottom towards the local optimum [Ruder, 2016, Bottou et al., 1991]. It's about to solve this problem of oscillation that we resort to the various optimizers, always focused on the SGD, which proposes an improvement of the traditional SGD by playing the various parameters and hyperparameters of this one [Lin et al., 2019, Ruder, 2016].

In the following, for the comparative study, we will select the fastest SGD optimizers variants which are widely used in Deep Learning, according to [Géron, 2017, Bottou, 2012], to deal with the above-mentioned challenges [Ruder, 2016]. optimizer: Adaptive Gradient or AdaGrad [Lydia and Francis, 2019], Root Mean Squared Propagation or RMSprop [Géron, 2017], Adaptive Moment Estimation or Adam [Kingma and Ba, 2014] and Nesterov-accelerated Adaptive Moment Estimation or Nadam [Lin et al., 2019, Dozat, 2016].

3 Dataset & Training Model

3.1 Data Preprocessing

The training set was built with a dataset of 450 license plate images and an xml annotation file. Image labeling is done with LabelImg, which is a graphical image annotation tool. The dataset is divided into three subsets: 70% for training, 10% for validation, and 20% for testing.

3.2 Build the CNN model with VGGNet architecture

VGG stand for Visual Geometry Group [Antoine Cornuéjols, 2018, chap. 10], it is a standard deep Convolutional Neural Network architecture with multiple layers. VGG competed in ILSVRC 2014, got good classification rate. Today VGGNet is a family of deep neural networks (from A to E). The VGG is the basis of revolutionary object recognition models [Tammina, 2019].

First a **"Sequential"** model is created in which we add **VGG-16** (i. e. with 16 layers, see [Antoine Cornuéjols, 2018, Géron, 2017]) as the first layer. The VGG-16 will take as input arrays of size $(200 \times 200 \times 3)$ which correspond to the reduced size of the images of the dataset (reduced in the preprocessing stage). The VGG-16 output is flattened with the **Flatten** layer. Finally, 4 layers **Dense**, are added later, and will act as completely connected layers with as output an array of size (4×1) which corresponds to the 4 points which will allow the model to draw the rectangle that will frame the license plate in the image.

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		

Table 1: The information the convolutional neural network builds with VGG-16.

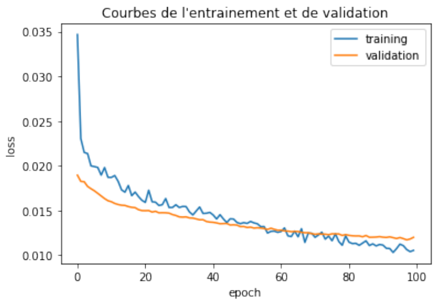
In this section, we will first explore and evaluate an architecture of convolutional neural networks, the Visual Geometry Group. The developed model, VGG-16, contains `19,474,948` parameters (fewer parameters compared to VGG-19) and about `4.7` million parameters are exploitable (optimizable).

4 Result & discussion

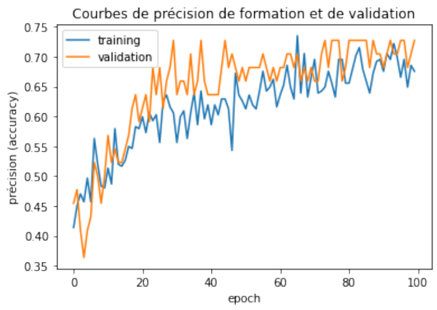
4.1 Optimizer evaluation

- **SGD** : The SGD minimizes the model parameter with a score of 67.5% accuracy and about 1% error. And on the validation data, the accuracy is 72.7% and an error of 1.2%. The test result reveals an error of 0.73% and an accuracy of 79.3%.

Training	Validation	Test
loss: 0.0105	loss: 0.0120	loss: 0.00732
accuracy: 0.6755	accuracy: 0.7273	accuracy: 0.7931



(a) SGD loss



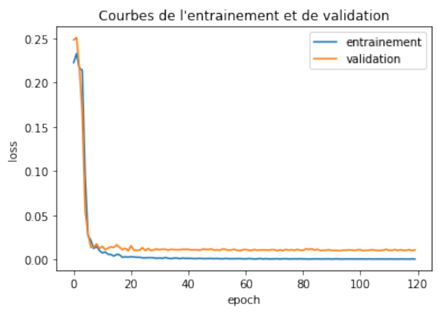
(b) SGD accuracy

Fig. 1: Accuracy and loss graph for SGD.

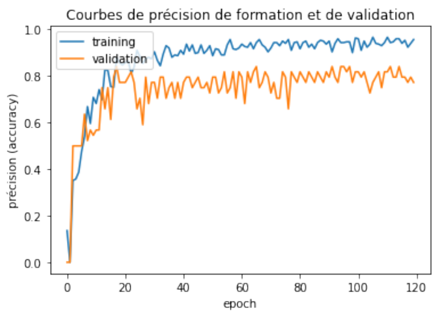
- **RMSprop** : The model minimized with the RMSprop gives a score of 95.7% accuracy, displays a deviation of about 28.2% compared to the SGD, and 0.044%. The error and accuracy

rate of RMSpro in validation is 1% and 77.2% respectively. The RMSprop optimizer also shows a lower error percentage than the SGD.
The convergence of RMSprop is rather accentuated because the minimization stabilizes after 80 steps and reaches the minimum value at the 110th step.

Training	**Validation**	**Test**
`loss: 4.4475e-04`	`loss: 0.0109`	`loss : 0.005589`
`accuracy: 0.9570`	`accuracy: 0.7727`	`accuracy : 0.8505`



(a) RMSprop loss

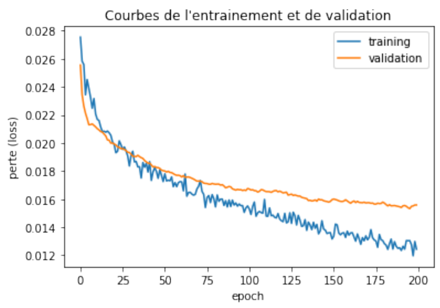


(b) RMSprop accuracy

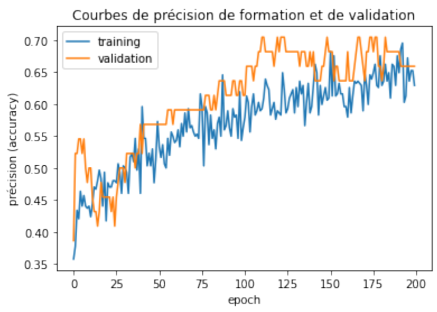
Fig. 2: Accuracy and loss graph for RMSprop.

- **AdaGrad** : AdaGrad scored poorly compared to the previous optimizer, RMSprop. An error rate of 1.2% and accuracy of 62.9% is the score of the AdaGrad optimizer. After 120 steps, which is the maximum step set for the tests, the minimization still does not stabilize. The minimum values are reached at the level of the 80*th* step.

Training	**Validation**	**Test**
`loss: 0.0124`	`loss: 0.0156`	`loss: 0.0094`
`accuracy: 0.6291`	`accuracy: 0.6591`	`accuracy: 0.5747`



(a) AdaGrad loss

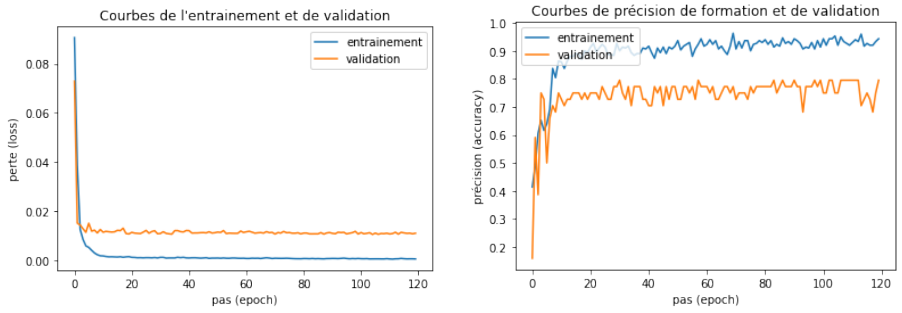


(b) AdaGrad accuracy

Fig. 3: Accuracy and loss graph for Adagrad.

• **Adam** : The Adam forms a model with 92.3% accuracy and an error of 0.086%, a bit higher compared to RMSprop. The validation score is 81.8% accuracy and 1.1% error. After the tests, 83.9% accuracy and 0.05% error is the observed score. RMSprop minimizes the cost better and shows a nice score compared to Adam.

Training	**Validation**	**Test**
`loss: 8.6163e-04`	`loss: 0.0110`	`loss : 0.00526`
`accuracy: 0.9238`	`accuracy: 0.8182`	`accuracy : 0.8390`

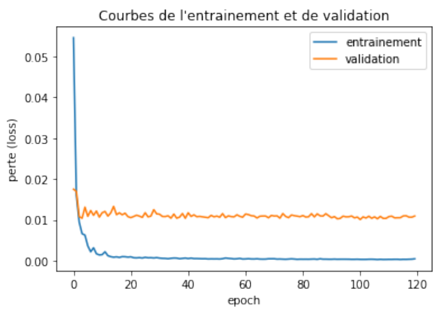


(a) Adam loss (b) Adam accuracy

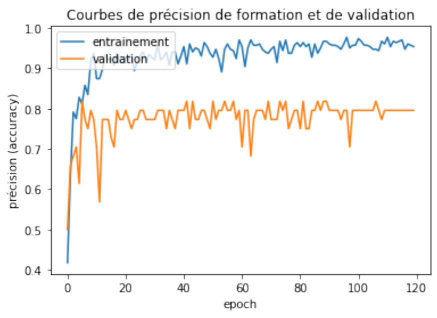
Fig. 4: Accuracy and loss graph for Adam.

• **Nadam** : Nadam, (Nesterov-accelerated Adaptive Moment Estimation), is an extension of the Adam optimizer to which we add the Nesterov Accelerated Gradient (NAG) [Ruder, 2016]. Nadam shows an attractive performance (from training, validation and testing point of view) compared to other optimizers studied in this work. It gives, as result, an accuracy of 95.3% and an error of 0.04%. Nadam accelerates the speed of convergence in a drastic way, it stabilizes at 30 steps and reaches the minimum value in only 40 steps.

Training	**Validation**	**Test**
`loss: 4.6549e-04`	`loss: 0.0109`	`loss : 0.00479`
`accuracy: 0.9536`	`accuracy: 0.7955`	`accuracy : 0.9080`



(a) Nadam loss



(b) Nadam accuracy

Fig. 5: Accuracy and loss graph for Nadam

RMSprop gives very low error in training compared to other optimizers, in testing phase Adam reveals lower error than RMSProp. The optimizer which has a good average, in the 3 phase, is Nadam with very good precision.

4.2 Optimizer classify according to ALPR

This work admits the Nadam optimizer as an algorithm that best minimizes errors and obtains a good score, in the context of ALPR, compared to the other optimizers studied. Below is a list of the best minimizing optimizers (table 1), ranked in descending order, for the case of the automatic license plate recognition problem.

N°	**Training**	**Validation**	**Test**
1. RMSprop	0.0004447	0.0109	0.005597
2. Nadam	0.00046549	0.0109	0.00479
3. Adam	0.00086163	0.0110	0.00526
4. Adagrad	0.0124	0.0156	0.0094
5. SGD	0.0105	0.0120	0.00732

Table 1: Table showing error rate per optimizer, listed in descending order.

The top 3 optimizers (from table 2) served as tests in license plate detection despite the geometric transformation, i.e. the 3 models that had a good training, validation and test score.

	Training	**Validation**	**Test**
Nadam	95.36%	79.55%	90.80%
Adam	92.58%	81.82%	83.90%
RMSprop	95.70%	77.27%	85.05%

Table 2: Table illustrating the resulting accuracy by the 3 best optimizer.

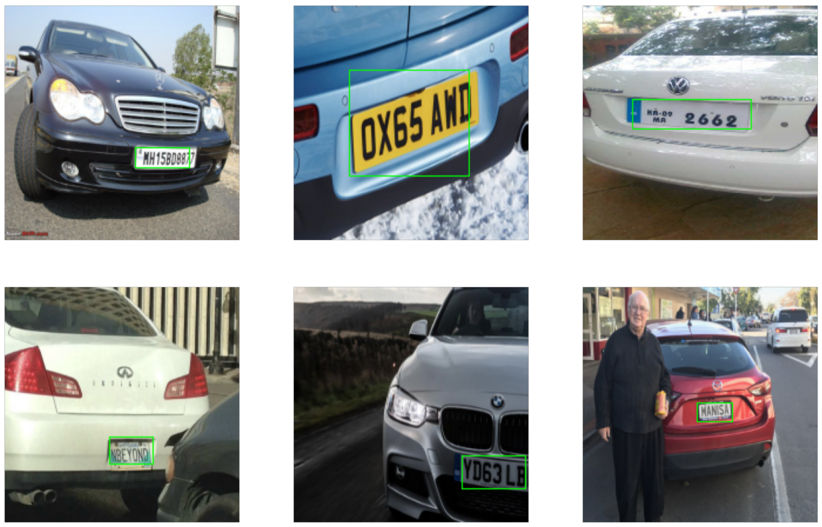
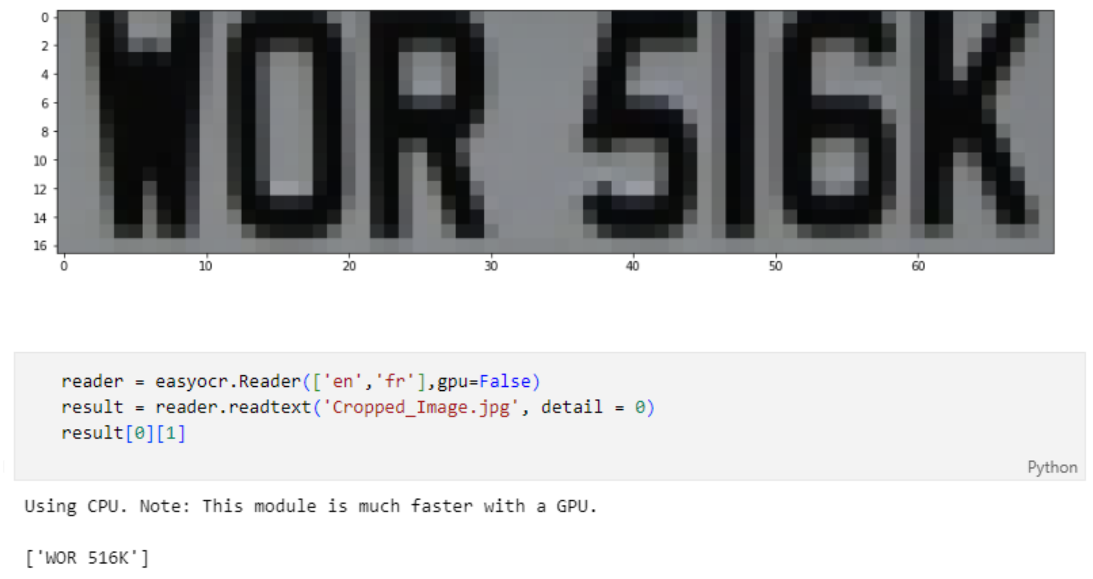


Fig. 6: The plates recognized despite the slight geometric transformation.

4.3 Optical Character Recognition

After recognizing the license plate, with the OpenCV Python library properties, it will be easy to extract the region of interest on the image which is the detected plate in the green rectangle. OCR processing is performed on the area, as shown in figure 7.

The result returned by EasyOCR Reader is an array containing a tuple of size 3, and the value we are interested in is at position 2 which corresponds to index 1 (`result[0][1]`).
`result = [([[0, 0], [70, 0], [70, 17], [0, 17]], 'Wor 5i6k', 0.49005824012267324)]`
The result in the `[Wor 5i6k]` terminal corresponds to the registered license plate number.



```
reader = easyocr.Reader(['en','fr'],gpu=False)
result = reader.readtext('Cropped_Image.jpg', detail = 0)
result[0][1]
```

```
Using CPU. Note: This module is much faster with a GPU.

['WOR 516K']
```

Fig. 7: The result of character recognition.

5 Conclusion

This paper provides an answer to our problem. It was a question of explaining when the minimization of errors in learning occurs. In order to justify the comparative study made, which made it possible to determine which optimizer is suitable for the problem of license plate detection in an image.

The conclusive results of the experiment, presented in section 4, show the performance of different optimizers treated in this work. The 3 best optimizers identified were used in license plate recognition. The RMSprop optimizer showed its feat in minimizing the cost function well compared to Nadam or Adam but from the precision point of view the Nadam optimizer came out first for this problem of the APLR.

References

V. B. Antoine Cornuéjols, Laurent Michet. *Apprentissage artificiel : Deep leaning, concepts et algorithmes.* Eyrolles, 3rd edition, 2018.

A. S. Bosman, A. Engelbrecht, and M. Helbig. Visualising basins of attraction for the cross-entropy and the squared error neural network loss functions. *Neurocomputing*, 400:113–136, 2020.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

L. Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

L. Bottou et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

C. Burges, R. Ragno, and Q. Le. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems*, 19, 2006.

T. Dozat. Incorporating nesterov momentum into adam. 2016.

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.

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

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

J. Lin, C. Song, K. He, L. Wang, and J. E. Hopcroft. Nesterov accelerated gradient and scale invariance for adversarial attacks. *arXiv preprint arXiv:1908.06281*, 2019.

A. Lydia and S. Francis. Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci*, 6(5):566–568, 2019.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

V. M. Sebastien Raschka. *Python Machine Learning and Deep Learning, with sckit-learn and Tensorflow.* Packt, 2nd edition, 2017.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

S. Tammina. Transfer learning using vgg-16 with deep convolutional neural network for classifying images. *International Journal of Scientific and Research Publications (IJSRP)*, 9(10): 143–150, 2019.

R. G. Wijnhoven and P. de With. Fast training of object detection using stochastic gradient descent. In *2010 20th International Conference on Pattern Recognition*, pages 424–427. IEEE, 2010.