

TC5 : Traitement des Images et du Signal

Etude d'un article scientifique

Eléonore Bartenlian, Adrien Pavão

Décembre 2017

Contents

1	Introduction	2
2	Données	2
2.1	CIFAR-10	2
2.2	Pré-traitements	2
2.3	Ajout de nouvelles données de test	4
3	BinaryConnect	8
3.1	Expériences	9
3.2	Réseau neuronal simple	9
3.2.1	Résultats	9
3.2.2	Analyse des résultats	11
3.3	Une architecture plus complexe	11
3.4	Résultats	12
4	Conclusion	17

1 Introduction

Pour ce projet, nous avons choisi d'implémenter la méthode d'entraînement de réseau neuronal présentée par l'article "*BinaryConnect : Training Deep Neural Networks with binary weights during propagations*". [5]. Le principe est d'entraîner un réseau neuronal avec des poids binaires (+1 ou -1) lors des phases de forward propagation et backward propagation.

Nous avons décidé d'entraîner un modèle basé sur l'article sur la base de données CIFAR-10. Nous l'avons ensuite testé non seulement sur cette même base de données mais également sur des images que nous avons récoltées et pré-traitées nous-même. ¹

2 Données

2.1 CIFAR-10

CIFAR-10 est une base de données d'images très utilisée pour tester et comparer des algorithmes de classification d'images. Elle est composée de 60000 images en format (3, 32, 32) : 3 canaux (RGB), et des dimensions d'image 32 par 32. Les couleurs sont représentées par des entiers entre 0 et 255 exclu. Elles sont étiquetées en 10 classes représentant des objets ou animaux du monde réel : airplane, automobile, bird, cat, deer, dog, frog, horse, ship et truck.



Figure 1: Exemples d'images de CIFAR-10

2.2 Pré-traitements

Global Contrast Normalization

Ce traitement consiste à normaliser le contraste d'une image en soustrayant chaque pixel par la moyenne des pixels, puis en divisant par l'écart-type. On re-normalise ensuite les pixels pour que leur valeur soit entre 0 et 254. Cela permet d'améliorer considérablement les résultats[1].


¹ : <https://github.com/Didayolo/binaryconnect>



Figure 2: Exemples d'images après application de la GCN

ZCA whitening

Le ZCA whitening[7] est un prétraitement qui transforme les données de telle sorte que leur matrice de covariance Σ soit la matrice identité. Cela permet de décorréler les features.

Avec N points dans \mathbb{R}^n , alors la matrice de covariance $\Sigma \in \mathbb{R}^{n \times n}$ est estimée de la manière suivante :

$$\hat{\Sigma}_{jk} = \frac{1}{N-1} \sum_{i=1}^N (x_{ij} - \bar{x}_j) \cdot (x_{ik} - \bar{x}_k)$$

où \bar{x}_j est le j ième composant de la moyenne estimée des échantillons x . On dit de toute matrice $W \in \mathbb{R}^{n \times n}$ qui satisfait la condition

$$W^W = C^{-1}T$$

qu'elle *blanchit* les données. Dans le cas du ZCA whitening, on prend $X = M^{-\frac{1}{2}}$.

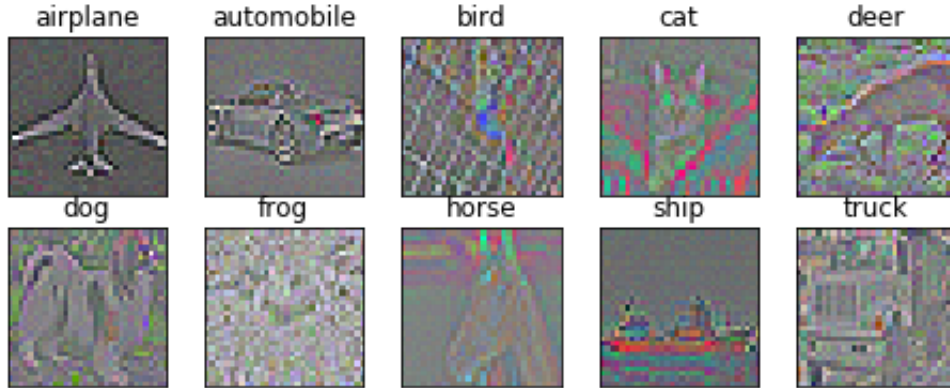


Figure 3: Exemples d'images après application du ZCA whitening, $\epsilon = 1e-6$

Cependant, les résultats n'étaient pas ceux escomptés. Les images semblaient correctement whitenées mais nous n'avons pas réussi à obtenir des résultats corrects.

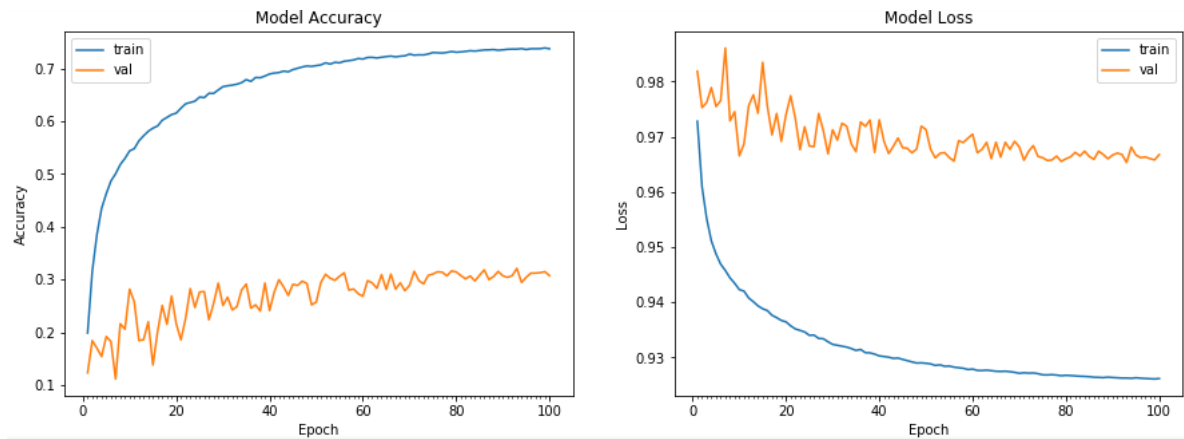


Figure 4: Résultats après 200 époques d'un réseau simple sur des données blanchies

On constate un grand écart entre la précision des données d'entraînement et celles des données de test : les résultats de validation se stabilisent à seulement 30% alors que ceux d'entraînement montent à plus de 70%. Nous n'avons pas réussi à corriger cela.

Pour les longs pré-processings, nous avons implémenté la possibilité d'enregistrer les images au format pickle afin de ne pas avoir à les recalculer à chaque fois.

2.3 Ajout de nouvelles données de test

Nous avons rajouté des images de Google Images ². Nous avons pris les 300 premiers résultats environ de chaque recherche dans Google Images, et nous avons ensuite vérifié chaque image à la main pour enlever celles qui ne correspondaient pas à la recherche. Nous avons enlevé toutes les images avec du texte ou des logos, celles qui étaient des dessins, celles qui n'avaient pas un fond naturel, ou qui ne correspondait pas à la recherche. Nous les avons ensuite renommées.

Nous avons rajouté les images suivantes :

- Des images représentant les **mêmes classes** que celles de la base de données CIFAR-10.



Figure 5: Exemples d'images que nous avons ajoutées

- Des images contenant **plusieurs objets différents** de la base de données : des chats avec des chiens et des chiens avec des oiseaux.

²Plugin firefox : <https://addons.mozilla.org/en-US/firefox/addon/google-images-downloader/>

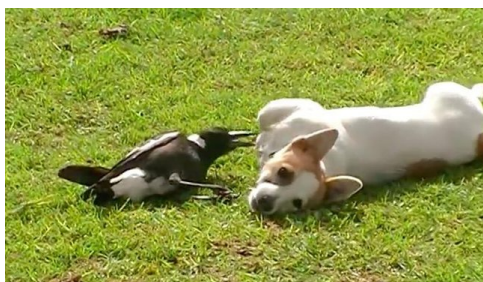


Figure 6: Un oiseau avec un chien



Figure 7: Un chat avec un chien

- En faisant cette dernière recherche, nous avons notamment trouvé un nombre surprenamment élevé de montages avec des **oiseaux à tête de chiens**. Nous avons donc décidé de les tester également afin de voir si la formé générale de l'oiseau l'emporterait sur la tête du chien, et quel probabilité l'algorithme donnerait aux différentes classes.

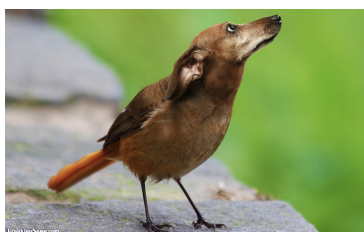


Figure 8: Images d'oiseaux à tête de chien.

- Des objets qui **n'appartiennent pas aux classes du dataset**, avec une vingtaine d'images par classe : des poissons, des fleurs, des arbres, des motos, des nuages, des lions, des serpents, des hélicoptères, des trains la mer et des images constituées uniquement de bruit.

Transformations

Afin d'effectuer des tests, nous avons appliqué les transformations suivantes aux images :

Flou gaussien

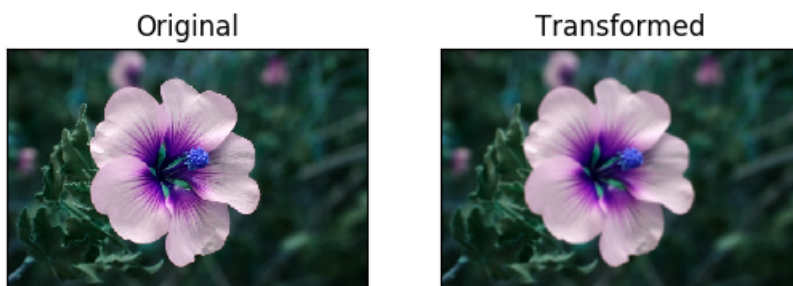


Figure 9: Image avant et après flou gaussien.

Bruit gaussien

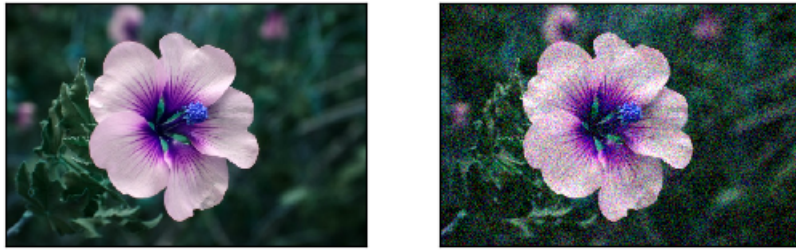


Figure 10: Image avant et après bruitage.

Flou et bruit gaussien

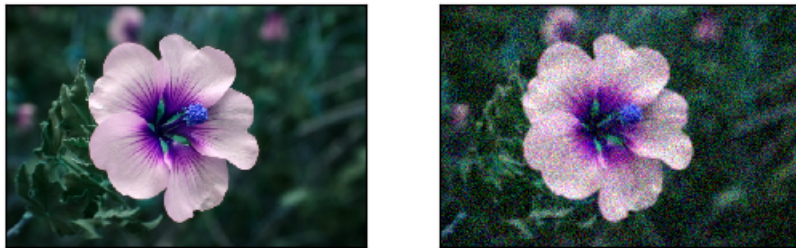


Figure 11: Image avant et après floutage et bruitage.

Rotation

Pour la rotation, nous avons d'abord tenté l'expérience avec un angle aléatoire. Cependant, l'image devant être redimensionnée, cela donnait des bandes noires qui risquaient de trop affecter l'apprentissage. Nous avons donc choisi de tourner l'image que de 90° , 180° ou 270° afin de ne pas avoir de bandes noires.

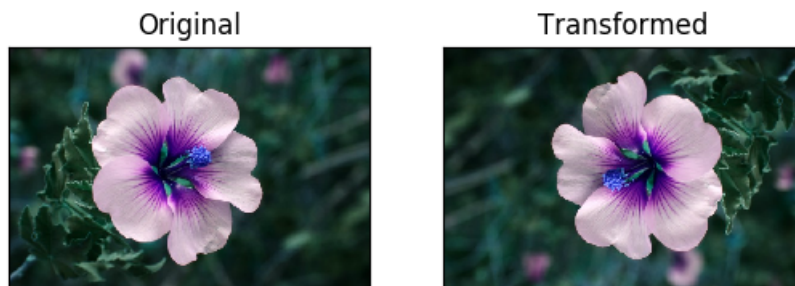


Figure 12: Image tournée selon un angle de 180°

Rotation et bruit

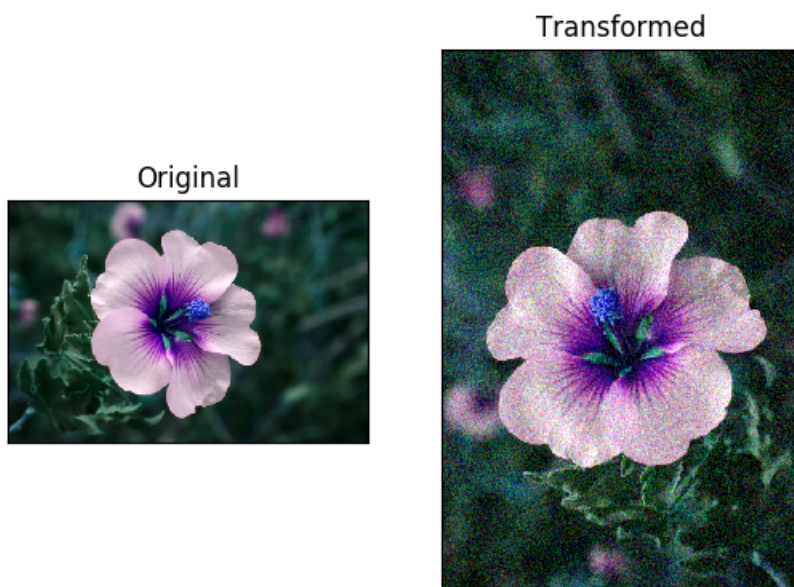


Figure 13: Image tournée selon un angle aléatoirement et bruitée.

Changement de contraste

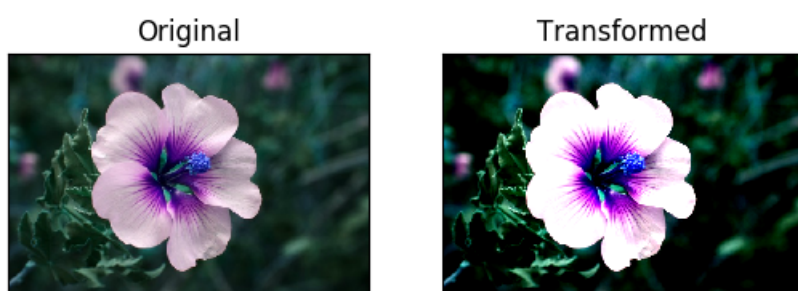


Figure 14: Augmentation sévère du contraste.



Figure 15: légère augmentation du contraste.



Figure 16: Diminution du contraste.

Inversion des couleurs

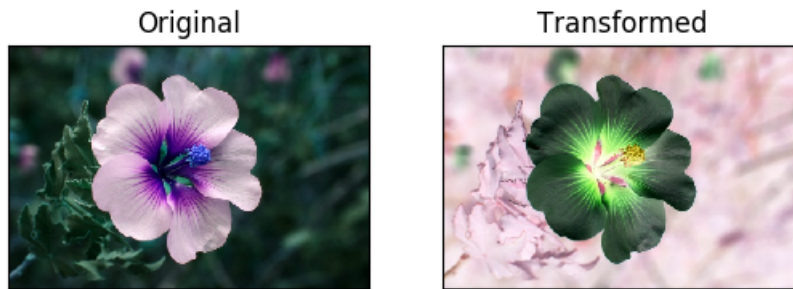


Figure 17: Inversion des couleurs.

Redimensionnement

Après avoir fait ces transformations, nous avons dû redimensionner les images : en effet, les tailles des images de CIFAR étant 32×32 , la première couche de notre réseau neuronal fait également cette taille. Nous avons pour cela utilisé la fonction `imresize` de `spicy` qui fonctionne avec une interpolation bilinéaire.

3 BinaryConnect

Les réseaux neuronaux profonds ont atteint des bonnes performances au prix de temps d'entraînement élevés, notamment en traitement d'images. Durant les phases de forward propagation et backward propagation, l'algorithme fait beaucoup de calculs de multiplication et d'addition sur des réels. Il faut en effet pour chaque neurone faire la somme de toutes ses entrées pondérées par les poids du neurone. Toutes ces valeurs sont des réels.

La méthode présentée dans l'article, *Binary Connect* est une tentative de réduire ces temps d'entraînement sans influencer négativement les performances.

Cela consiste à binariser les poids, c'est-à-dire leur faire prendre seulement 2 valeurs possibles, durant les phases de **forward propagation** et de **backward propagation**. La première correspond au calcul de la sortie du modèle à partir de son entrée, et la seconde correspond à la rétropropagation du gradient dépendant de la distance entre la sortie du modèle et la sortie attendue. On ne modifie pas les poids durant la phase de mise-à-jour. Ainsi, beaucoup d'opérations de multiplications peuvent être remplacées par de simples additions et soustractions, ce qui simplifie considérablement les calculs[3].

Les deux méthodes de **binarisation** proposées sont les suivantes :

- **Déterministe :**

$$w = \begin{cases} +1 & \text{si } w \geq 0 \\ -1 & \text{sinon} \end{cases}$$

- **Stochastique :**

$$w = \begin{cases} +1 & \text{avec une probabilité } p = \sigma(w) \\ -1 & \text{avec une probabilité } 1 - p \end{cases}$$

Avec σ la fonction *hard sigmoid*.

3.1 Expériences

Optimizer et fonction de perte

Comme dans l'article, nous utilisons l'optimiseur ADAM[4] et la fonction de perte *square hinge*. Pour un résultat attendu $t = \pm 1$ et un score de classification y , la *hinge loss* de la prédiction y est définie comme :

$$l(y) = \max(0, 1 - t.y)$$

Batch normalization

L'idée est de normaliser les entrées de chaque couche de telle sorte qu'elles aient une activation de sortie moyenne de 0 et une déviation standard de 1. Cela aide non seulement le modèle à apprendre plus vite mais également à obtenir de meilleurs résultats[2].

3.2 Réseau neuronal simple

Nous avons commencé par tester l'influence de la binarisation sur la performance et le temps de calcul avec un réseau neuronal feed-forward simple, sans convolution et sans prétraitement sur les données.

Les tests ont été faits sur CIFAR-10 avec 50000 images d'entraînement et 10000 de test. Les paramètres étaient les suivants pour chaque test :

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation	Taille de batch
[100, 100, 100]	20	1e-3 à 1e-4	tanh	128

3.2.1 Résultats

Sans binarisation

Le modèle a pris **61.43 secondes** à s'entraîner. La précision sur l'ensemble de test est de **42.10%**.

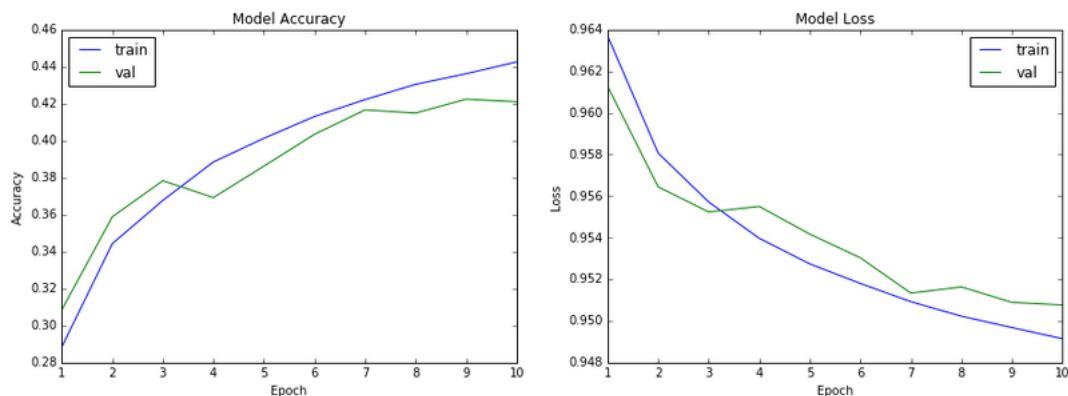


Figure 18: Évolution de la précision et de la perte en fonction de l'époque, sans binarisation

Les courbes sont classiques et conformes à ce que l'on pouvait attendre. La précision est assez faible, mais c'est tout à fait normal pour ce genre de problème pour un réseau de neurone sans couche de convolution.

En ajoutant une normalisation des batches à chaque couche, de paramètres :

- $\epsilon = 1e-6$
- $\text{momentum} = 0.9$

On passe à **70.16** secondes d'entraînement, et une précision de **46.43%**. Nous appliquerons des batch normalizations dans tous les exemples suivants.

Avec binarisation déterministe

Le modèle a pris **118.48 secondes** à s'entraîner. La précision sur l'ensemble de test est de **26.79%**.

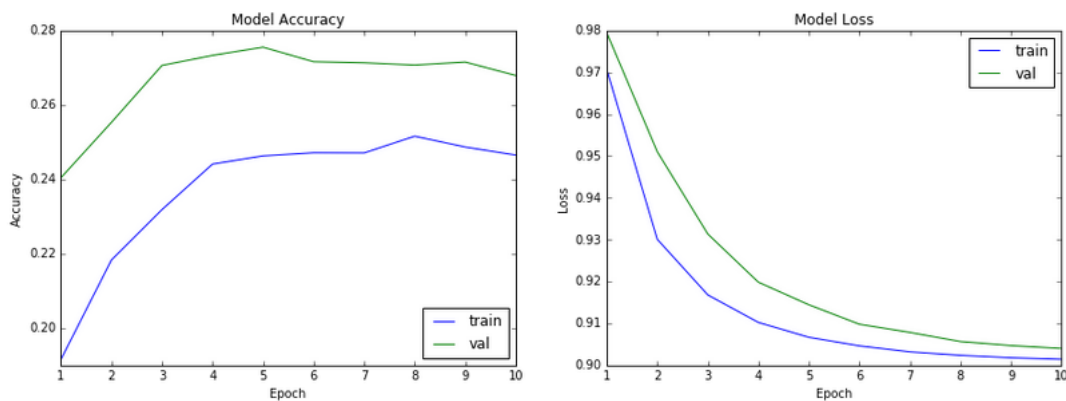


Figure 19: Évolution de la précision et de la perte en fonction de l'époque, avec binarisation déterministe

On remarque que la précision sur l'ensemble de test est supérieure à celle sur l'ensemble d'apprentissage.

Avec binarisation stochastique

Le modèle a pris **120.62 secondes** à s'entraîner. La précision sur l'ensemble de test est de **26.24%**.

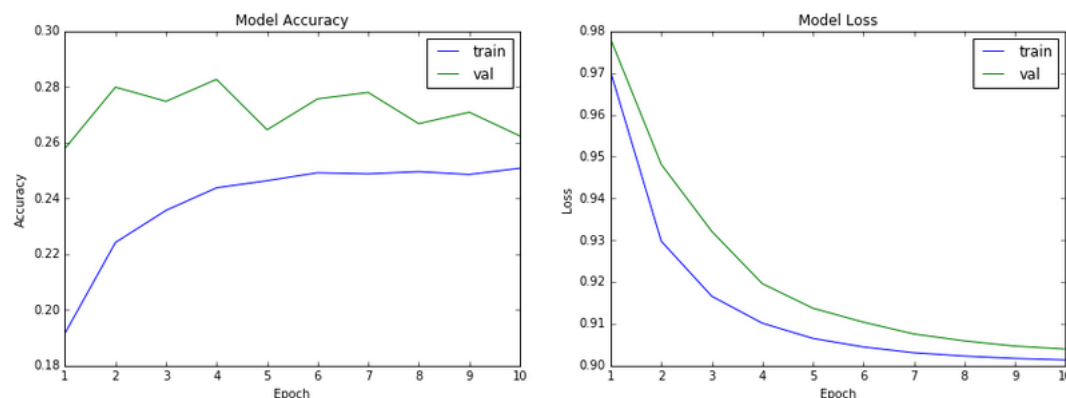


Figure 20: Évolution de la précision et de la perte en fonction de l'époque, avec binarisation stochastique

On retrouve des résultats assez similaires à ceux de la binarisation déterministe. Cette fois ci, on a une grande variance dans la généralisation, c'est-à-dire la précision sur les données de test. Cela est sans doute dû au caractère aléatoire du processus de binarisation des poids lors de la propagation du gradient, et donc de la mise-à-jour de ces poids.

3.2.2 Analyse des résultats

Avec la binarisation,

- **La précision** passe d'environ 42% à environ 26%. Il va falloir créer un réseau neuronal plus complexe, avec plus de couches, plus de neurones et différents types de couches. On l'entraînera ensuite sur un grand nombre d'époques.
- **Le temps de calcul** double presque. On peut supposer qu'il n'y a pas un nombre assez grand de neurone dans nos couches, et donc pas un nombre suffisamment grand d'opérations d'additions et de multiplications pour que les bénéfices de la binarisation soient visibles.

Les résultats sont donc très mauvais, on ne fait que dégrader notre réseau neuronal. Voyons comment faire mieux.

3.3 Une architecture plus complexe

Nous avons ensuite implémenté l'architecture décrite dans l'article elle-même inspiré de [6].

Grâce à la fonction Keras *model.summary()*, on peut obtenir une visualisation du réseau neuronal :

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 128, 32, 32)	3584
conv2d_14 (Conv2D)	(None, 128, 30, 30)	147584
max_pooling2d_7 (MaxPooling2D)	(None, 128, 15, 15)	0
conv2d_15 (Conv2D)	(None, 256, 15, 15)	295168
conv2d_16 (Conv2D)	(None, 256, 13, 13)	590080
max_pooling2d_8 (MaxPooling2D)	(None, 256, 6, 6)	0
conv2d_17 (Conv2D)	(None, 512, 6, 6)	1180160
conv2d_18 (Conv2D)	(None, 512, 4, 4)	2359808
max_pooling2d_9 (MaxPooling2D)	(None, 512, 2, 2)	0
flatten_10 (Flatten)	(None, 2048)	0
dense_34 (Dense)	(None, 1024)	2098176
dense_35 (Dense)	(None, 1024)	1049600
dense_36 (Dense)	(None, 10)	10250
Total params: 7,734,410		
Trainable params: 7,734,410		
Non-trainable params: 0		

Figure 21: Architecture du réseau

A chaque couche, la fonction d'activation est la fonction de rectification linéaire (relu). La dernière couche possède une régularisation de norme L2 et une fonction d'activation linéaire. La taille de batch utilisée est 50, et l'on effectue des batch normalizations.

Au cours de l'entraînement, nous sauvegardions le modèle entraîné à chaque époque. Malheureusement, nous ne disposons pas de la puissance de calcul nécessaire pour pouvoir laisser le modèle tourner suffisamment longtemps pour obtenir des résultats corrects. Nous avons donc arrêté le calcul au bout de 15 époques et 3 jours de calcul pour faire un nouveau réseau plus simple.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
dropout_1 (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 30, 30)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 32, 15, 15)	0
flatten_1 (Flatten)	(None, 7200)	0
dense_1 (Dense)	(None, 512)	3686912
dropout_2 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130
Total params: 3,702,186		
Trainable params: 3,702,186		
Non-trainable params: 0		

Figure 22: Architecture du nouveau réseau

C'est celui-ci qui a été utilisé pour les tests.

3.4 Résultats

Le modèle utilisé est le réseau neuronal présenté plus haut. La taille des mini-batch est de 32, le nombre d'époque est 25 et la binarisation utilisée était la binarisation stochastique. L'entraînement a été réalisé uniquement sur CIFAR-10. Nous allons à présent tester le modèle sur les données présentées dans la section 2.

CIFAR-10

Le batch de test de CIFAR-10, sans ajout ni modification, contenant 10000 images.

Classe	Précision	Recall
airplane	0.599	0.291
automobile	0.712	0.430
bird	0.361	0.245
cat	0.249	0.305
deer	0.509	0.057
dog	0.253	0.659
frog	0.598	0.204
horse	0.369	0.496
ship	0.466	0.672
truck	0.473	0.549

Figure 23: Résultats sur CIFAR-10

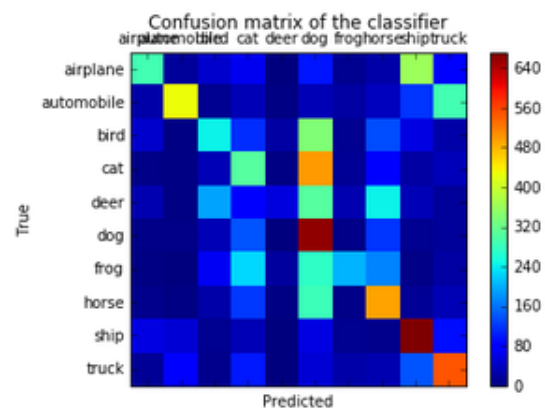


Figure 24: Matrice de confusion

La précision moyenne est de **46%**. Le modèle confond les animaux en général avec les chiens. Il est bon avec les automobiles et les avions. Les avions et les bateaux sont souvent confondus.

Images ajoutées et transformations

Nos images sont les images prises sur internet ayant les mêmes classes que CIFAR-10. Tous les effets et les transformations ont été appliqués sur ces images. Pour les effets de baisse et d'augmentation du contraste, la Global Contrast Normalization est désactivée lors du pré-traitement des données.

- **Nos images** : Notre ensemble de test avec les mêmes classes que CIFAR-10, le même format et des images prises sur Google Images. Cet ensemble est constitué de 1370 images.

Classe	Précision	Recall
airplane	0.773	0.446
automobile	0.935	0.391
bird	0.338	0.249
cat	0.319	0.209
deer	0.263	0.038
dog	0.171	0.533
frog	0.841	0.248
horse	0.287	0.500
ship	0.465	0.915
truck	0.504	0.547

Figure 25: Résultats sur nos images

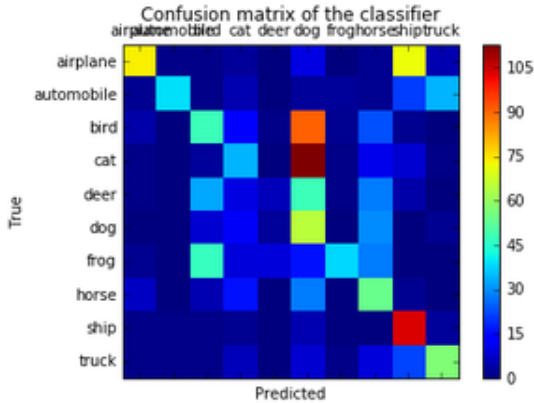


Figure 26: Matrice de confusion

La précision moyenne est de **49%** et les confusions sont globalement les mêmes. La classe *chien* est encore plus mal reconnue, mais les résultats sont très similaires. Nos données sont proches de celles de CIFAR-10.

- **Noisy** : Notre ensemble de test après avoir bruitée les images.
Les résultats sont les mêmes. Cela est sans doute dû au fait que nous réduisons la taille des images après les avoir bruitées, annulant ainsi cet effet.
- **Rotated** : Images tournées à 90°, 180° ou 270° aléatoirement.



Figure 27: Exemples d'images après rotation

Classe	Précision	Recall
airplane	0.507	0.446
automobile	0.419	0.164
bird	0.241	0.189
cat	0.184	0.198
deer	0.333	0.038
dog	0.073	0.308
frog	0.582	0.215
horse	0.125	0.188
ship	0.143	0.085
truck	0.065	0.019

Figure 28: Résultats sur nos images après rotation

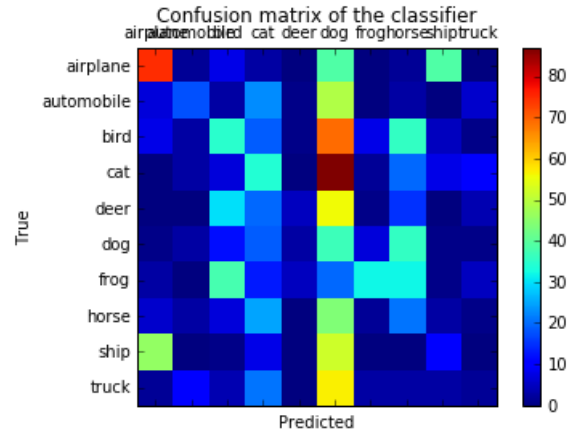


Figure 29: Matrice de confusion

Après de simples rotations, les résultats sont très mauvais. La précision tombe à **28%** et le recall à 20% en moyenne. La précision reste correcte sur les avions (51%) et les grenouilles (58%), mais les recalls sont faibles. La précision sur les chiens est de 7%.

- **Blurred** : Images floutées.

Tout comme pour les images bruitées, les résultats sont identiques avec ou sans le flou. On peut une fois de plus supposer que c'est la réduction de la taille des images après le flou qui rend son impact négligeable.

- **Couleurs inversées** : Images avec les couleurs inversées.



Figure 30: Exemples d'images après inversion des couleurs

Classe	Précision	Recall
airplane	0.090	0.143
automobile	0.218	0.200
bird	0.152	0.054
cat	0.122	0.116
deer	0.000	0.000
dog	0.068	0.250
frog	0.000	0.000
horse	0.000	0.000
ship	0.028	0.060
truck	0.062	0.028

Figure 31: Résultats sur nos images après inversion des couleurs

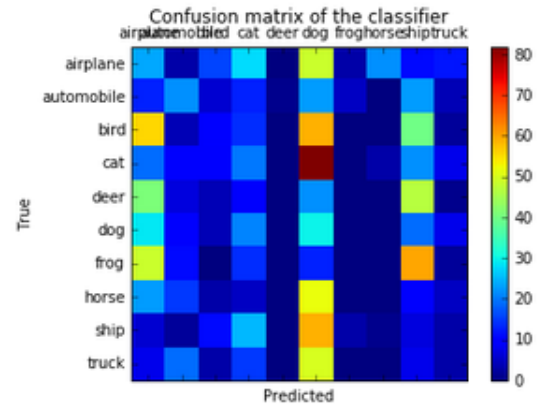


Figure 32: Matrice de confusion

L'inversion des couleurs provoque des résultats désastreux. La précision moyenne est de **7.7%**. On fait moins bien que le hasard pour 7 classes sur les 10. De façon plutôt prévisible, les couleurs sont vraiment importantes pour le classifieur.

- **Baisse du contraste :**

Pas de différences notables. C'est probablement dû au fait que nous appliquons la global contrast normalization en prétraitement. Les résultats semblent un peu meilleur, mais cela peut-être du au hasard.

- **Forte augmentation contraste :**

Pas de grandes différences non plus. Les résultats sont un peu moins bons qu'avec la baisse de contraste.

- **Faible augmentation du contraste :**

Les résultats sont bons. Comme on pouvait s'y attendre, cet effet ne change pas grand chose.

- **NoisyBlur :** Images bruitées puis floutées.

Nous avons vu que le bruit et le flou n'influençaient pas les résultats. Il n'est donc pas surprenant que les deux ensembles n'influencent pas non plus les résultats.

- **NoisyRotated :** Images bruitées puis tournée à 90°, 180° ou 270°.

Les résultats chutent de la même façon qu'avec seulement les rotations appliquées aux images. Une fois de plus, le bruit n'influence pas les résultats.

Images hors classes

Nous allons maintenant tester les images n'appartenant à aucune classe. Nous ne nous intéressons donc pas à la précision, mais dans quelles classes les images ont été classifiées par notre modèle. Il est important de garder à l'esprit que les tests ont été menés avec peu d'exemples par classe.

- **Oiseau-chien :** Montages photo d'oiseaux à tête de chiens. 49 images.

Classe	Moyenne des probabilités prédites
airplane	0.009
automobile	0.004
bird	0.111
cat	0.186
deer	0.016
dog	0.388
frog	0.020
horse	0.131
ship	0.049
truck	0.086

Figure 33: Résultats pour "birdogs"

Voit surtout des chiens, et peu d'oiseau (chat et cheval passent avant)

- **Chat et chien :** Un chien et un chat sur la même photo. 132 images.

Classe	Moyenne des probabilités prédites
airplane	0.000
automobile	0.005
bird	0.007
cat	0.385
deer	0.013
dog	0.379
frog	0.024
horse	0.095
ship	0.002
truck	0.089

Figure 34: Résultats pour “catsdogs”

Voit les deux pareil. 76% des images vues (38 et 38) sont des chats et des chiens.

- **Oiseaux et chiens** : Un oiseau et un chien sur le même photo. 29 images.

Classe	Moyenne des probabilités prédites
airplane	0.016
automobile	0.000
bird	0.080
cat	0.394
deer	0.004
dog	0.332
frog	0.030
horse	0.082
ship	0.000
truck	0.061

Figure 35: Résultats pour “birdsdogs”

Le classifieur voit des chiens et des chats, mais voit mal les oiseaux. C’est probablement parce que les oiseaux sont en général plus petits que les chiens donc il ne les repère pas sur l’image.

- **Nuage** : 20 images.

Les nuages ont été majoritairement reconnus en tant que bateaux (0.34), puis en chiens (0.28). On pourrait penser que les nuages rappelle la fourrure de certains chiens. La classe chien est la plus souvent proposée et confondue par notre modèle, c’est pourquoi on la retrouve régulièrement en cas de doute. La classe avion a eu une moyenne de probabilité de prédiction de 0.11 ; c’est moins que ce à quoi on aurait pu s’attendre.

- **Poisson** : 20 images.

Les poissons sont principalement pris pour des bateaux (0.32). Une fois de plus, la classe chien est assez représentée (0.20).

- **Fleur** : 20 images.

Pour les fleurs, c’est la classe chien qui l’emporte (0.30). Cela montre que le classifieur est perdu face à ces nouveaux exemples n’appartenant pas aux classes habituelles. On remarque que les probabilités sont plus grandes pour les animaux que les objets dans cet exemple ci : ce sont ensuite les classes cheval et chat qui ont le score le plus élevés. Cela peut s’expliquer par le fait que les fleurs et les animaux sont souvent photographiés dans des paysages naturels, que le réseau neuronal identifie bien (notamment la couleur verte).

- **Hélicoptère** : 20 images.

Les hélicoptères sont pris pour des avions avec une probabilité de prédiction moyenne de 0.50, et ensuite pour des bateaux avec 0.36. Ce résultat semble tout à fait cohérent.

- **Lion** : 20 images.

Classe	Moyenne des probabilités prédites
airplane	0.003
automobile	0.000
bird	0.063
cat	0.172
deer	0.001
dog	0.324
frog	0.001
horse	0.260
ship	0.050
truck	0.127

Figure 36: Résultats pour lions

Les lions sont vus par le modèle comme des chiens (0.32), des chevaux (0.26) et des chats (0.17). Ces résultats sont cohérents puisque ce sont des animaux assez ressemblants.

- **Moto** : 19 images.

La classe ayant le meilleur score est camion (0.35). Chien et chat ont des scores plutôt élevés également (0.19 et 0.20), ce qui montre que le classifieur n'est pas efficace sur ces images. La classe automobile a un score faible (0.10).

- **Bruit** : Des images constituées uniquement de bruit. 20 images.

Sur ces images ne représentant rien, juste du bruit de toutes les couleurs, l'algorithme prédit principalement des grenouilles, avec un score élevé de 0.53. Les images de grenouille sont souvent pleines de couleurs (vert, bleu, jaune), cependant ce résultat est inattendu.

- **Mer** : 20 images.

Le classifieur voit beaucoup de bateaux pour un score de 0.74. Il reconnaît plus de bateaux dans ces images de mer seule que dans les images de bateaux. La mer est sans doute un point clé pour reconnaître les bateaux.

- **Serpent** : 20 images.

Sur les images de serpents, l'algorithme prédit principalement des chiens (0.28). Il s'agit de la confusion qui survient chaque fois.

- **Train** : 20 images.

Les photos de trains ont été vues comme des camions (0.30) et des bateaux (0.22). C'est ce à quoi on pouvait s'attendre. On remarque toutefois que la classe voiture obtient un score très faible (0.002).

- **Arbre** : 20 images.

Le programme voit en grande partie des oiseaux (0.28). Cela vient probablement du fait que les oiseaux sont souvent dans des arbres sur les images de CIFAR-10.

4 Conclusion

Les résultats sur nos données sont pour la plupart ceux à quoi on s'attendait et semblent cohérents. En revanche, nous n'avons pas réussi à obtenir un taux de précision supérieur à 45% sur les données en général avec binarisation des poids. C'est probablement dû au fait que notre réseau neuronal n'était pas assez grand.

Nous avons vu certains aspects moins plaisants du Machine Learning : une grande partie du travail a consisté à récolter les données, et nous étions très limités par les temps de calculs et la puissance de nos machines.

Cependant, ce projet s'est avéré très enrichissant. En effet, il nous aura permis à la fois de renforcer et d'appliquer nos connaissances en traitement d'images, en Deep Learning, d'apprendre à lire et implémenter un article scientifique, et de découvrir des outils Python.

References

- [1] H. Lee A. Coates and A. Ng. “An Analysis of Single-Layer Networks in Unsupervised Feature Learning”. In: *AISTATS 14, 2011* (2011). URL: <http://proceedings.mlr.press/v15/coates11a/coates11a.pdf>.
- [2] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *ICML 2015* (2015). URL: <http://proceedings.mlr.press/v37/ioffe15.pdf>.
- [3] Kassem Kalach Jean Pierre David and Nicolas Tittley. “Hardware Complexity of Modular Multiplication and Exponentiation”. In: *IEEE Transactions on Computers* (2007). URL: https://www.researchgate.net/publication/3049587_Hardware_Complexity_of_Modular_Multiplication_and_Exponentiation.
- [4] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference for Learning Representations* (2015). URL: <https://arxiv.org/pdf/1412.6980.pdf>.
- [5] Yoshua Bengio Matthieu Courbariaux and Jean-Pierre David. “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”. In: *NIPS 2015* (2015). URL: <http://papers.nips.cc/paper/5647-binaryconnect-training-deep-neural-networks-with-binary-weights-during-propagations.pdf>.
- [6] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *ICLR 2015* (2015). URL: <https://arxiv.org/pdf/1409.1556.pdf>.
- [7] Yangyu Fan Zuhe LiEmail and Weihua Liu. “The effect of whitening transformation on pooling operations in convolutional autoencoders”. In: *EURASIP Journal on Advances in Signal Processing* (2015). URL: <https://link.springer.com/article/10.1186/s13634-015-0222-1>.