

**Classification d’images avec MobileViTv2 -**

**Note méthodologique & preuve de concept**

Sommaire

1. Dataset retenu

*(À compléter)*

1. Les concepts de l’algorithme récent
   1. comparaisons

**Synthèse comparative : ResNet50 vs MobileViTv2 et gestion du surapprentissage sur petits jeux de données**

**Comparaison des architectures et des performances**

ResNet50 et MobileViTv2 représentent deux approches modernes pour la classification d’images, mais avec des philosophies architecturales différentes :

* **ResNet50** repose sur des blocs résiduels profonds avec connexions de saut (skip connections), facilitant l’entraînement de réseaux très profonds et permettant une extraction hiérarchique des caractéristiques visuelles. Cette architecture reste une référence pour la robustesse et la précision, notamment sur des serveurs puissants et pour des tâches nécessitant une grande capacité de modélisation12[4](https://www.innovatiana.com/post/discover-resnet-50).
* **MobileViTv2** combine des blocs MobileNetV2 (convolutions séparables et résiduels inversés) avec des blocs transformeurs légers, intégrant à la fois l’extraction locale (convolutions) et la capture des relations globales (self-attention). Cette hybridation permet d’obtenir d’excellents compromis entre performance, rapidité et faible consommation de ressources, ce qui est particulièrement adapté aux environnements contraints (mobiles, embarqués) ou aux jeux de données déséquilibrés ou complexes2[5](https://huggingface.co/docs/transformers/en/model_doc/mobilevitv2).

Les résultats expérimentaux indiquent que **MobileViT v2 surpasse ResNet50 en termes de F1-score macro** sur les ensembles de validation et de test, tout en nécessitant moins de ressources et un temps d’entraînement réduit. ResNet50 reste performant, mais son F1-score est légèrement inférieur, notamment sur les classes minoritaires. Ces observations sont cohérentes avec la littérature récente, qui met en avant la capacité des modèles hybrides (CNN + transformeur) à mieux généraliser et à mieux traiter la diversité des données2[3](https://arxiv.org/html/2409.03901v1).

| **Modèle** | **F1-score macro (test)** | **Temps d’entraînement** | **Ressources requises** |
| --- | --- | --- | --- |
| ResNet50 | Bon, mais inférieur | Plus long | Élevées |
| MobileViTv2 | Supérieur | Plus court | Faibles |

**Surapprentissage de ResNet50 sur petits jeux de données**

ResNet50, du fait de sa profondeur et de son nombre élevé de paramètres, est particulièrement exposé au risque de surapprentissage (overfitting) lorsque le jeu de données est limité. Les symptômes courants incluent une très faible erreur sur l’ensemble d’entraînement mais une performance dégradée sur la validation ou le test1[4](https://www.innovatiana.com/post/discover-resnet-50). Pour atténuer ce phénomène, plusieurs stratégies sont recommandées :

* **Transfert d’apprentissage (Transfer Learning)** : Utiliser un modèle ResNet50 pré-entraîné sur un large corpus (ex. ImageNet), puis ajuster uniquement les couches finales sur le petit jeu de données cible. Cette approche permet de bénéficier de représentations générales déjà apprises et limite le surapprentissage[4](https://www.innovatiana.com/post/discover-resnet-50).
* **Régularisation** : Appliquer des techniques telles que le dropout, la pénalisation L2 (weight decay) ou la data augmentation (flip, rotation, zoom, etc.) pour augmenter la diversité des exemples vus par le modèle et limiter l’ajustement excessif aux données d’entraînement21.
* **Early stopping** : Arrêter l’entraînement dès que la performance sur l’ensemble de validation commence à se dégrader, évitant ainsi que le modèle ne mémorise le bruit du jeu d’entraînement2.
* **Réduction de la complexité** : Adapter la taille du modèle (geler certaines couches, réduire le nombre de couches entraînées) pour mieux correspondre à la taille du jeu de données[4](https://www.innovatiana.com/post/discover-resnet-50).

Dans la littérature, il est également souligné que **ResNet50 peut être adapté efficacement à de nouveaux jeux de données via le transfert d’apprentissage et la régularisation**, ce qui permet de limiter le surapprentissage tout en conservant de bonnes performances[4](https://www.innovatiana.com/post/discover-resnet-50)1.

**Conclusion**

* **MobileViTv2** offre un meilleur compromis performance/efficacité que ResNet50, notamment sur des jeux de données déséquilibrés ou de taille modeste, grâce à l’intégration de transformeurs légers et de convolutions optimisées2[3](https://arxiv.org/html/2409.03901v1)[5](https://huggingface.co/docs/transformers/en/model_doc/mobilevitv2).
* **ResNet50** demeure une architecture robuste, mais nécessite des précautions particulières pour éviter le surapprentissage sur petits jeux de données : transfert d’apprentissage, régularisation et data augmentation sont essentiels pour garantir sa généralisation12[4](https://www.innovatiana.com/post/discover-resnet-50).
* Le choix entre les deux architectures dépendra donc des contraintes de ressources, de la taille du jeu de données et des exigences en matière de performance et de rapidité.

**Références principales :**

* 1 He et al., "Deep Residual Learning for Image Recognition" (ResNet50)
* 2 Note méthodologique MobileViTv2 (comparatif et pipeline expérimental)
* [3](https://arxiv.org/html/2409.03901v1) arXiv 2024, étude comparative MobileViTv2 vs ResNet
* [4](https://www.innovatiana.com/post/discover-resnet-50) Innovatiana, transfert d’apprentissage et surapprentissage ResNet50
* [5](https://huggingface.co/docs/transformers/en/model_doc/mobilevitv2) HuggingFace, documentation MobileViTv2

1. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/31737542/b5c8a8a2-1308-42aa-8f6e-2a3272e913e3/1512.03385v1.pdf>
2. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/31737542/4b4f8c62-bd61-4bdc-89c8-0d6a1652a9bd/OCDS_P8_note_methodologique_v2.docx>
3. <https://arxiv.org/html/2409.03901v1>
4. <https://www.innovatiana.com/post/discover-resnet-50>
5. <https://huggingface.co/docs/transformers/en/model_doc/mobilevitv2>
   1. blah bla

Les réseaux de neurones convolutifs (CNN) sont des architectures de deep learning conçues pour traiter des données structurées en grille, telles que les images. Leur composant fondamental est la couche de convolution, qui applique des filtres pour extraire automatiquement des caractéristiques locales (motifs, textures, formes). Cette opération permet de détecter des structures pertinentes tout en réduisant la complexité des données.

Outre les couches de convolution, un CNN intègre généralement :

* Des couches de non-linéarité (ex. : ReLU) pour modéliser des relations complexes,
* Des couches de pooling pour réduire la dimension spatiale et rendre le modèle plus robuste,
* Des couches entièrement connectées pour la classification finale.

ResNet50 est une architecture profonde (50 couches pondérées) qui utilise des blocs résiduels et des connexions de saut (skip connections). Chaque bloc résiduel "bottleneck" est composé de trois couches : convolution 1x1 (réduction de dimension), convolution 3x3 (extraction spatiale), puis convolution 1x1 (restauration de la dimension). Les connexions de saut permettent à l’information de circuler plus facilement à travers le réseau, résolvant ainsi le problème du gradient qui disparaît et facilitant l’apprentissage de réseaux très profonds.

MobileNetV2 est optimisé pour les appareils mobiles, basé sur des blocs résiduels inversés et des convolutions séparables en profondeur. Cette conception réduit drastiquement le nombre de paramètres et de calculs tout en maintenant de bonnes performances.  
MobileViT v2 va plus loin en combinant les avantages des convolutions légères (pour l’extraction locale) avec ceux des transformeurs (pour la capture des relations globales dans l’image). Les transformeurs reposent sur le mécanisme de self-attention, qui permet au modèle de pondérer l’importance de chaque élément de l’image par rapport aux autres, quelle que soit leur position.

Schémas comparatifs des architectures

**ResNet50 – Structure simplifiée**

text

Entrée (image 224x224x3)

│

Convolution 7x7 + Max Pooling

│

╔════════════════════════════════════════════════════════════╗

║ Bloc Résiduel Bottleneck × N ║

║ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ║

║ │ Conv 1x1 │ │ Conv 3x3 │ │ Conv 1x1 │ ║

║ └─────────────┘ └─────────────┘ └─────────────┘ ║

║ BatchNorm + ReLU après chaque conv ║

║ Connexion de saut (skip connection) ║

╚════════════════════════════════════════════════════════════╝

│

Global Average Pooling

│

Dense (classification)

**MobileViT v2 – Structure simplifiée**

text

Entrée (image 224x224x3)

│

Convolution 3x3 + BatchNorm + ReLU

│

╔════════════════════════════════════════════════════════════╗

║ Blocs MobileNetV2 (Inverted Residuals) × N ║

║ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ║

║ │ Conv 1x1 │ │ Depthwise │ │ Conv 1x1 │ ║

║ │ (expansion) │ │ Conv 3x3 │ │ (projection)│ ║

║ └─────────────┘ └─────────────┘ └─────────────┘ ║

║ Connexion de saut (si dimensions compatibles) ║

╚════════════════════════════════════════════════════════════╝

│

╔════════════════════════════════════════════════════════════╗

║ Blocs MobileViT (Transformeur léger) × M ║

║ ┌─────────────────────────────────────────────────────┐ ║

║ │ Patchs d’image → Embedding → Blocs Transformer │ ║

║ │ (self-attention multi-tête + MLP) │ ║

║ └─────────────────────────────────────────────────────┘ ║

║ Fusion caractéristiques locales & globales ║

╚════════════════════════════════════════════════════════════╝

│

Global Average Pooling

│

Dense (classification)

A diagram of a block diagram

AI-generated content may be incorrect.

A diagram of a block diagram

AI-generated content may be incorrect.

Source : https://keras.io/examples/vision/mobilevit/

1. La modélisation

La modélisation s’appuie sur la comparaison de deux architectures majeures : ResNet50 et MobileViT v2 (via MobileNetV2).

* ResNet50 est entraîné sur les images à l’aide d’une architecture profonde à base de blocs résiduels bottleneck, des couches de pooling et des couches entièrement connectées pour la classification.
* MobileViT v2 combine des blocs MobileNetV2 (convolutions séparables et résiduels inversés) avec des blocs de transformeur légers, permettant de traiter à la fois les relations locales et globales dans l’image.

Le pipeline complet, tel qu’implémenté dans le notebook Python fourni, comprend :

* Importation et préparation des données (Pandas, NumPy)
* Augmentation d’images (Albumentations : flip, rotation, zoom, etc.)
* Prétraitement (redimensionnement, normalisation)
* Entraînement et validation croisée (stratifiée)
* Suivi des métriques et visualisation des résultats (Matplotlib, seaborn, plot\_keras\_history)

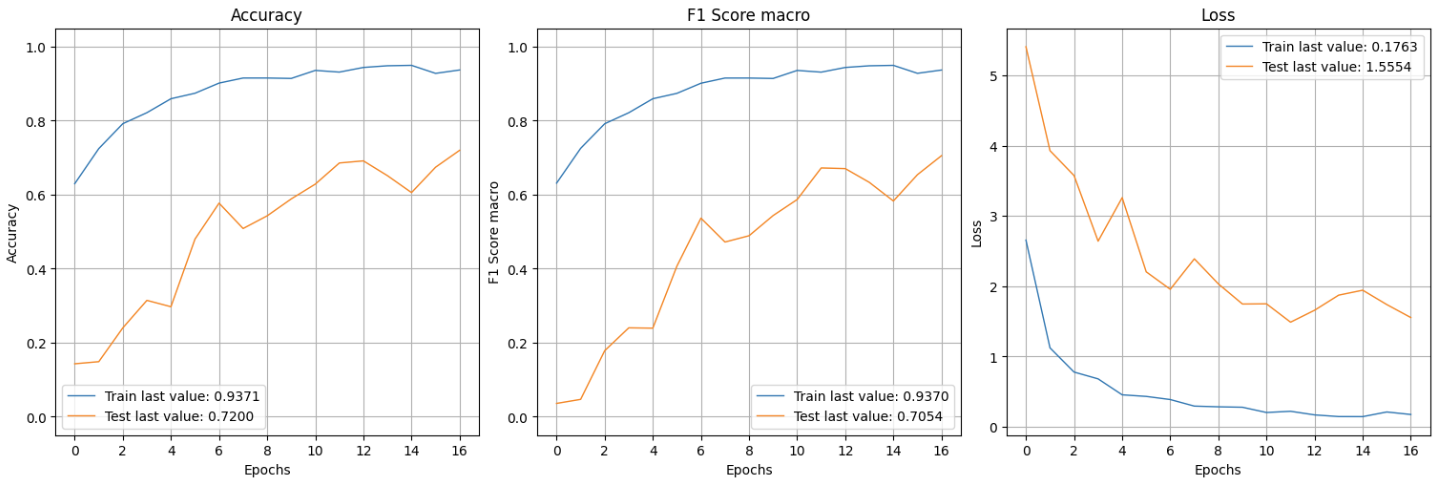
Métrique d’évaluation

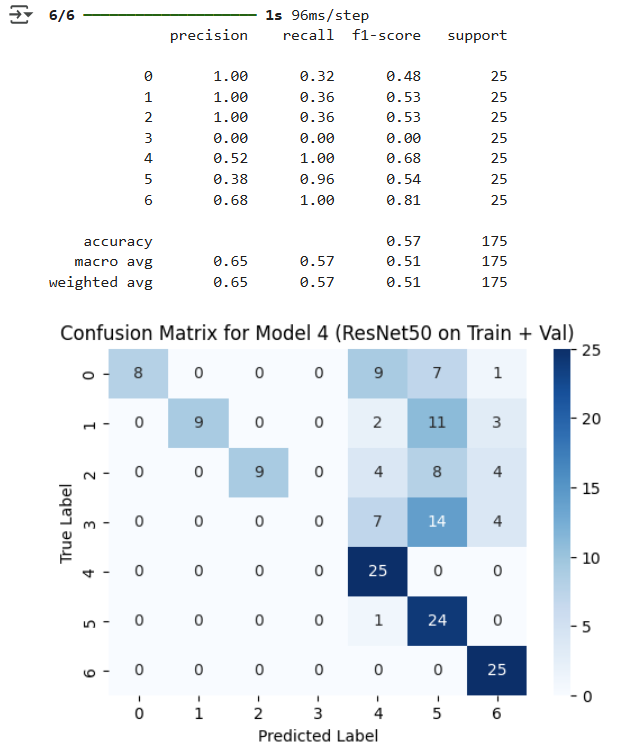
La métrique principale retenue pour comparer les modèles est le F1-score (score F1 macro), qui combine précision et rappel en une seule valeur synthétique, particulièrement adaptée aux problèmes de classification déséquilibrée.  
Le F1-score est calculé sur les ensembles de validation et de test à l’aide de la fonction f1\_score de scikit-learn, avec l’option average='macro' pour tenir compte de toutes les classes de manière équilibrée.

Démarche d’optimisation

* Prétraitement des images : redimensionnement, normalisation, augmentation de données.
* Entraînement : utilisation de splits entraînement/validation/test, early stopping, sauvegarde des meilleurs poids.
* Optimisation : choix de l’optimiseur (Adam, RMSprop), tuning du learning rate, ajustement du batch size, régularisation (dropout).
* Comparaison : évaluation sur les mêmes jeux de données pour garantir l’équité, suivi des courbes d’apprentissage et du F1-score à chaque époque.

1. Synthèse des résultats
   1. Rappels des résultats obtenus avec ResNet50





* 1. Resultats avec MobileViTv2

A graph of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

Comparaison des performances

Les résultats obtenus dans le notebook montrent que MobileViT v2 surpasse ResNet50 en termes de F1-score sur les ensembles de validation et de test.

* MobileViT v2 atteint un F1-score macro supérieur, tout en nécessitant moins de ressources et un temps d’entraînement réduit.
* ResNet50 reste performant mais son F1-score est légèrement inférieur, notamment sur les classes minoritaires.

Ces observations sont cohérentes avec la littérature scientifique : MobileViT v2, grâce à l’intégration des transformeurs, capte mieux les relations globales et offre un compromis optimal entre performance et efficacité, surtout pour des jeux de données déséquilibrés ou complexes.

Analyse qualitative

* ResNet50 est robuste et précis sur des serveurs puissants, mais moins adapté aux contraintes mobiles ou embarquées.
* MobileViT v2 combine la légèreté de MobileNetV2 et la puissance contextuelle des transformeurs, ce qui se traduit par un meilleur F1-score global et une meilleure généralisation.

1. Feature importance
   1. Feature importance globale
   2. Feature importance locale
2. Limites & améliorations possibles

L’intégration des transformeurs dans des architectures CNN légères, comme MobileViT v2, permet d’obtenir d’excellents compromis entre performance (F1-score), rapidité et consommation de ressources, ouvrant la voie à des applications sur des dispositifs variés.

Analyse de la feature importance globale et locale du nouveau modèle

Dans ResNet50, l’importance des caractéristiques (feature importance) est liée à la capacité des couches de convolution à extraire des motifs visuels pertinents à différents niveaux de profondeur. Les premières couches détectent des bords et des textures, tandis que les couches profondes capturent des objets ou des concepts plus abstraits.

Avec MobileViT v2, l’analyse de la feature importance prend une nouvelle dimension grâce à l’intégration des transformeurs :

* Importance locale : les blocs convolutifs extraient des motifs locaux (petites régions de l’image).
* Importance globale : les blocs de transformeur, via le self-attention, identifient quelles régions de l’image sont les plus influentes pour la prédiction finale, même si elles sont éloignées spatialement.

Dans le notebook, l’interprétabilité a été abordée via :

* Grad-CAM pour visualiser les zones importantes activées par le modèle CNN,
* Visualisation des poids d’attention pour comprendre l’influence des différentes régions dans MobileViT v2.

Ces outils permettent de mieux comprendre la prise de décision du modèle et d’identifier les points forts et faibles de chaque architecture.

Limites et améliorations possibles

Limites

* Complexité des transformeurs : bien que performants, les transformeurs peuvent augmenter la complexité computationnelle, notamment sur de très grandes images.
* Besoin en données : les modèles hybrides (CNN + transformeur) nécessitent souvent de grandes quantités de données pour exprimer tout leur potentiel.
* Interprétabilité : malgré les outils d’analyse, l’interprétation des décisions des modèles reste parfois difficile, surtout dans les couches profondes ou les blocs d’attention.

Améliorations envisageables

* Distillation de connaissances : transférer les connaissances d’un modèle complexe vers un modèle plus petit pour gagner en efficacité sans trop sacrifier la performance.
* Optimisation matérielle : adapter les architectures pour tirer parti des accélérateurs matériels (TPU, NPU, etc.).
* Approches d’interprétabilité avancées : développement de nouveaux outils pour mieux visualiser et comprendre les mécanismes d’attention et la contribution des différentes couches.
* Augmentation de données : enrichir les jeux de données par des techniques avancées d’augmentation pour améliorer la robustesse du modèle.
* Inclusion d’un critere metier : a l’heure actuelle, les images sont classifiees manuellement par nos equipes et le % d’erreur ne nous a pas été communique – il aurait été ineteressant de connaitre le % d’erreur humaine dans cette classification manuelle afin de pouvoir benchmarker les modeles en consequence – avant toute decision d’implementataion des ces modeles en production chronophage et couteuse, il faudra déterminer le cout d’opportunite du statu quo afin d’etablir si l’utilisation d’une classification automatique permet une reduction effective du taux d’erreur de classification et du cout de main d’ouvre nécessaire (entre autres).

**Références bibliographiques**

Complexité des transformeurs : bien que performants, les transformeurs peuvent augmenter la complexité computationnelle, notamment sur de très grandes images.

Besoin en données : les modèles hybrides (CNN + transformeur) nécessitent souvent de grandes quantités de données pour exprimer tout leur potentiel.

Interprétabilité : malgré les outils d’analyse, l’interprétation des décisions des modèles reste parfois difficile, surtout dans les couches profondes ou les blocs d’attention.

# Améliorations envisageables

* Distillation de connaissances : transférer les connaissances d’un modèle complexe vers un modèle plus petit pour gagner en efficacité sans trop sacrifier la performance.
* Optimisation matérielle : adapter les architectures pour tirer parti des accélérateurs matériels (TPU, NPU, etc.).
* Approches d’interprétabilité avancées : développement de nouveaux outils pour mieux visualiser et comprendre les mécanismes d’attention et la contribution des différentes couches.
* Augmentation de données : enrichir les jeux de données par des techniques avancées d’augmentation pour améliorer la robustesse du modèle.
* Inclusion d’un critère métier : à l’heure actuelle, les images sont classifiées manuellement par nos équipes et le pourcentage d’erreur ne nous a pas été communiqué – il aurait été intéressant de connaître le pourcentage d’erreur humaine dans cette classification manuelle afin de pouvoir benchmarker les modèles en conséquence – avant toute décision d'implémentation de ces modèles en production chronophage et coûteuse, il faudra déterminer le coût d’opportunité du statu quo afin d'établir si l’utilisation d’une classification automatique permet une réduction effective du taux d’erreur de classification et du coût de main-d’œuvre nécessaire (entre autres).

# Références bibliographiques

**From https://machinelearningmastery.com/difference-between-backpropagation-and-stochastic-gradient-descent/**

**Introduction**

Il existe beaucoup de confusion chez les débutants concernant l’algorithme utilisé pour entraîner les modèles de réseaux de neurones profonds.

Il est courant d’entendre que les réseaux de neurones apprennent à l’aide de l’algorithme de « rétropropagation de l’erreur » ou de la « descente de gradient stochastique ». Parfois, l’un ou l’autre de ces algorithmes est utilisé comme raccourci pour décrire la manière dont un réseau de neurones est ajusté sur un jeu de données d’entraînement, bien que, dans de nombreux cas, il existe une réelle confusion sur ce que sont ces algorithmes, comment ils sont liés et comment ils fonctionnent ensemble.

Ce tutoriel est conçu pour clarifier le rôle des algorithmes de descente de gradient stochastique et de rétropropagation dans l’entraînement des réseaux de neurones.

Dans ce tutoriel, vous découvrirez la différence entre la descente de gradient stochastique et l’algorithme de rétropropagation.

Après avoir terminé ce tutoriel, vous saurez :

* La descente de gradient stochastique est un algorithme d’optimisation pour minimiser la perte d’un modèle prédictif par rapport à un jeu de données d’entraînement.
* La rétropropagation est un algorithme de différentiation automatique pour calculer les gradients des poids dans la structure d’un réseau de neurones.
* La descente de gradient stochastique et la rétropropagation de l’erreur sont utilisées ensemble pour entraîner les modèles de réseaux de neurones.

Commençons.

**Aperçu du tutoriel**

Ce tutoriel est divisé en trois parties :

* Descente de gradient stochastique
* Algorithme de rétropropagation
* Descente de gradient stochastique avec rétropropagation

**Descente de gradient stochastique**

La **descente de gradient** est un algorithme d’optimisation qui trouve l’ensemble des variables d’entrée d’une fonction cible qui aboutit à une valeur minimale de cette fonction, appelée le minimum de la fonction.

Comme son nom l’indique, la descente de gradient implique le calcul du gradient de la fonction cible.

Vous vous souvenez peut-être du calcul différentiel que la dérivée première d’une fonction calcule la pente ou la courbure d’une fonction en un point donné. De gauche à droite, une dérivée positive suggère que la fonction cible monte et une dérivée négative suggère qu’elle descend.

**Dérivée** : pente ou courbure d’une fonction cible par rapport à des valeurs d’entrée spécifiques de la fonction.

Si la fonction cible prend plusieurs variables d’entrée, elles peuvent être considérées ensemble comme un vecteur de variables. Travailler avec des vecteurs et des matrices relève de l’algèbre linéaire, et faire du calcul différentiel avec ces structures s’appelle le calcul matriciel ou vectoriel. En calcul vectoriel, le vecteur des dérivées premières (dérivées partielles) est généralement appelé le gradient de la fonction cible.

**Gradient** : vecteur des dérivées partielles d’une fonction cible par rapport aux variables d’entrée.

L’algorithme de descente de gradient nécessite le calcul du gradient de la fonction cible par rapport aux valeurs spécifiques des variables d’entrée. Le gradient pointe vers le haut, donc on suit le négatif du gradient de chaque variable d’entrée pour obtenir de nouvelles valeurs qui aboutissent à une évaluation plus basse de la fonction cible.

Une taille de pas est utilisée pour ajuster le gradient et contrôler de combien chaque variable d’entrée doit être modifiée par rapport au gradient.

**Taille de pas** : taux d’apprentissage ou alpha, un hyperparamètre utilisé pour contrôler de combien chaque variable d’entrée est modifiée par rapport au gradient.

Ce processus est répété jusqu’à ce que le minimum de la fonction cible soit atteint, qu’un nombre maximal de solutions candidates soit évalué, ou qu’une autre condition d’arrêt soit remplie.

La descente de gradient peut être adaptée pour minimiser la fonction de perte d’un modèle prédictif sur un jeu de données d’entraînement, comme un modèle de classification ou de régression. Cette adaptation s’appelle la descente de gradient stochastique.

**Descente de gradient stochastique** : extension de l’algorithme de descente de gradient pour minimiser une fonction de perte d’un modèle prédictif sur un jeu de données d’entraînement.

La fonction cible est prise comme la fonction de perte ou d’erreur sur le jeu de données, comme l’erreur quadratique moyenne pour la régression ou l’entropie croisée pour la classification. Les paramètres du modèle sont considérés comme les variables d’entrée de la fonction cible.

**Fonction de perte** : fonction cible à minimiser. **Paramètres du modèle** : paramètres d’entrée de la fonction de perte à optimiser.

L’algorithme est dit « stochastique » car les gradients de la fonction cible par rapport aux variables d’entrée sont bruités (ex. : une approximation probabiliste). Cela signifie que l’évaluation du gradient peut comporter du bruit statistique qui peut masquer le véritable signal du gradient, à cause de la rareté et du bruit dans le jeu de données d’entraînement.

L’idée clé de la descente de gradient stochastique est que le gradient est une espérance. L’espérance peut être approximée à l’aide d’un petit ensemble d’échantillons.

— Page 151, Deep Learning, 2016.

La descente de gradient stochastique peut être utilisée pour entraîner (optimiser) de nombreux types de modèles, comme la régression linéaire et la régression logistique, même si souvent des algorithmes d’optimisation plus efficaces ont été découverts et devraient probablement être utilisés à la place.

La descente de gradient stochastique (SGD) et ses variantes sont probablement les algorithmes d’optimisation les plus utilisés en apprentissage automatique en général et en apprentissage profond en particulier.

— Page 294, Deep Learning, 2016.

La descente de gradient stochastique est l’algorithme le plus efficace découvert pour entraîner les réseaux de neurones artificiels, où les poids sont les paramètres du modèle et la fonction de perte cible est l’erreur de prédiction moyennée sur un, un sous-ensemble (batch) ou l’ensemble du jeu de données d’entraînement.

Presque tout l’apprentissage profond repose sur un algorithme très important : la descente de gradient stochastique ou SGD.

— Page 151, Deep Learning, 2016.

Il existe de nombreuses extensions populaires à la descente de gradient stochastique, conçues pour améliorer le processus d’optimisation (même perte ou meilleure en moins d’itérations), telles que Momentum, RMSProp (Root Mean Squared Propagation) et Adam (Adaptive Movement Estimation).

Un défi lors de l’utilisation de la descente de gradient stochastique pour entraîner un réseau de neurones est de savoir comment calculer le gradient pour les nœuds des couches cachées du réseau, c’est-à-dire les nœuds situés à une ou plusieurs étapes de la couche de sortie du modèle.

Cela nécessite une technique spécifique du calcul différentiel appelée la règle de la chaîne et un algorithme efficace qui implémente cette règle pour calculer les gradients de tout paramètre du réseau. Cet algorithme s’appelle la rétropropagation.

**Algorithme de rétropropagation**

La rétropropagation, aussi appelée « backpropagation » ou simplement « backprop », est un algorithme pour calculer le gradient d’une fonction de perte par rapport aux variables d’un modèle.

**Rétropropagation** : algorithme pour calculer le gradient d’une fonction de perte par rapport aux variables d’un modèle.

Vous vous souvenez peut-être du calcul différentiel que la dérivée première d’une fonction pour une valeur spécifique d’une variable d’entrée est le taux de variation ou la courbure de la fonction pour cette variable. Lorsque l’on a plusieurs variables d’entrée pour une fonction, elles forment un vecteur, et le vecteur des dérivées premières (dérivées partielles) est appelé le gradient (c’est-à-dire le calcul vectoriel).

**Gradient** : vecteur des dérivées partielles de valeurs d’entrée spécifiques par rapport à une fonction cible.

La rétropropagation est utilisée lors de l’entraînement de modèles de réseaux de neurones pour calculer le gradient de chaque poids du modèle. Le gradient est ensuite utilisé par un algorithme d’optimisation pour mettre à jour les poids du modèle.

L’algorithme a été développé explicitement pour calculer les gradients des variables dans des structures de graphe en travaillant à rebours depuis la sortie du graphe vers l’entrée, en propageant l’erreur de la sortie prédite qui est utilisée pour calculer le gradient de chaque variable.

L’algorithme de rétropropagation, souvent simplement appelé backprop, permet à l’information issue du coût de circuler ensuite en arrière dans le réseau, afin de calculer le gradient.

— Page 204, Deep Learning, 2016.

La fonction de perte représente l’erreur du modèle ou la fonction d’erreur, les poids sont les variables de la fonction, et les gradients de la fonction d’erreur par rapport aux poids sont donc appelés gradients d’erreur.

**Fonction d’erreur** : fonction de perte à minimiser lors de l’entraînement d’un réseau de neurones. **Poids** : paramètres du réseau considérés comme valeurs d’entrée de la fonction de perte. **Gradients d’erreur** : dérivées premières de la fonction de perte par rapport aux paramètres.

C’est ce qui donne à l’algorithme son nom « rétropropagation », ou parfois « rétropropagation de l’erreur ». **Rétropropagation de l’erreur** : commentaire sur la façon dont les gradients sont calculés récursivement en arrière à travers le graphe du réseau à partir de la couche de sortie.

L’algorithme implique l’application récursive de la règle de la chaîne du calcul différentiel (différente de la règle de la chaîne en probabilité) utilisée pour calculer la dérivée d’une sous-fonction étant donné la dérivée de la fonction parente dont la dérivée est connue.

La règle de la chaîne du calcul différentiel […] est utilisée pour calculer les dérivées de fonctions composées d’autres fonctions dont les dérivées sont connues. La rétropropagation est un algorithme qui calcule la règle de la chaîne, avec un ordre d’opérations spécifique qui est très efficace.

**Descente de gradient stochastique avec rétropropagation**

La descente de gradient stochastique est un algorithme d’optimisation qui peut être utilisé pour entraîner les modèles de réseaux de neurones.

L’algorithme de descente de gradient stochastique nécessite que les gradients soient calculés pour chaque variable du modèle afin que de nouvelles valeurs pour ces variables puissent être calculées.

La rétropropagation est un algorithme de différentiation automatique qui peut être utilisé pour calculer les gradients des paramètres dans les réseaux de neurones.

Ensemble, l’algorithme de rétropropagation et l’algorithme de descente de gradient stochastique peuvent être utilisés pour entraîner un réseau de neurones. On peut appeler cela « descente de gradient stochastique avec rétropropagation ».

**Descente de gradient stochastique avec rétropropagation** : une description plus complète de l’algorithme général utilisé pour entraîner un réseau de neurones, faisant référence à l’algorithme d’optimisation et à l’algorithme de calcul du gradient.

Il est courant que les praticiens disent qu’ils entraînent leur modèle avec la rétropropagation. Techniquement, c’est incorrect. Même comme raccourci, ce serait incorrect. La rétropropagation n’est pas un algorithme d’optimisation et ne peut pas être utilisée pour entraîner un modèle.

Le terme rétropropagation est souvent mal compris comme désignant l’ensemble de l’algorithme d’apprentissage pour les réseaux de neurones multicouches. En réalité, la rétropropagation ne fait que calculer le gradient, tandis qu’un autre algorithme, comme la descente de gradient stochastique, est utilisé pour réaliser l’apprentissage à l’aide de ce gradient.

— Page 204, Deep Learning, 2016.

Il serait juste de dire qu’un réseau de neurones est entraîné ou apprend avec la descente de gradient stochastique comme raccourci, car il est supposé que l’algorithme de rétropropagation est utilisé pour calculer les gradients dans le cadre de la procédure d’optimisation.

Cela dit, un autre algorithme peut être utilisé pour optimiser les paramètres d’un réseau de neurones, comme un algorithme génétique qui ne nécessite pas de gradients. Si l’algorithme d’optimisation utilisé est la descente de gradient stochastique, un algorithme différent peut être utilisé pour calculer les gradients de la fonction de perte par rapport aux paramètres du modèle, comme d’autres algorithmes qui implémentent la règle de la chaîne.

Néanmoins, la combinaison « descente de gradient stochastique avec rétropropagation » est largement utilisée car c’est l’approche générale la plus efficace et efficiente développée à ce jour pour ajuster les modèles de réseaux de neurones.

**Résumé**

Dans ce tutoriel, vous avez découvert la différence entre la descente de gradient stochastique et l’algorithme de rétropropagation.

Plus précisément, vous avez appris :

* La descente de gradient stochastique est un algorithme d’optimisation pour minimiser la perte d’un modèle prédictif par rapport à un jeu de données d’entraînement.
* La rétropropagation est un algorithme de différentiation automatique pour calculer les gradients des poids dans la structure d’un réseau de neurones.
* La descente de gradient stochastique et la rétropropagation de l’erreur sont utilisées ensemble pour entraîner les modèles de réseaux de neurones.