Pré-traitement

Lorsque les variables ont des plages de valeurs très différentes (par exemple VAR1 $\in [-2, 4]$ et VAR2 $\in [0.7, 1.3]$), la normalisation devient cruciale pour éviter qu'une variable ne domine excessivement l'analyse, en rééquilibrant leur influence relative.

Normalisation [0;1]

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Normalisation [−1;1]

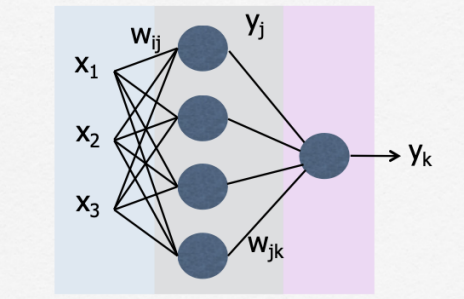
$$x' = 2\frac{x - x_{\min}}{(x_{\max} - x_{\min})} - 1$$

Cependant MinMax peut rencontrer des problèmes pour normaliser s'il y a des valeurs aberrantes. nous devons donc en premier lieu filtrer les valeurs abérantes avec la formule de normalisation z (z-score) est :

$$x' = \frac{x - \mu}{\sigma}$$

x est la valeur originale, **μ** (mu) est la moyenne des données, **σ** (sigma) est l'écart-type

Perceptron



- **Gauche**: nous avons les 3 **entrées** x_1, x_2 et x_3.
- **Milieu**: nous avons la **couche cachée** avec ses neurones chaqu'un ayant des entrées avec un poid w ainsi qu'un biais b.
- **Droite**: nous avons la **sortie** y qui est le résultat de la somme des entrées multipliées par leurs poids respectifs plus le biais b.

Fonction du MLP

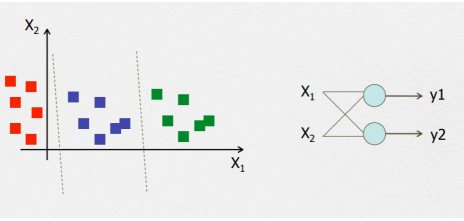
$$Y_k = f\sum_i (X_i \cdot W_{ik} + b_k)$$

Total des poids

On somme:
- Poids d'entrée: nb entrée $*$ nb neurones
- Poids de sortie: nb sortie $*$ nb neurones
- **Ajouter le biais**

Strucutre lors de plusieurs classes

Le nombre d'entrées reflète la complexité du problème (2D x,y = 2 entrées). Le nombre de neurones cachés s'adapte à la séparabilité linéaire, nécessitant plus de neurones ou de couches pour les problèmes non linéairement séparables. Le nombre de sorties correspond aux classes à prédire.



Formule d'un neurone

$$y = f(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

Concept de Momentum

Le momentum est une propriété physique qui permet à un objet ayant une masse de continuer sa trajectoire même lorsqu'une force externe opposée est appliquée.

Dans le contexte des réseaux de neurones, l'idée derrière ce "truc" est d'ajouter un terme de momentum à l'adaptation des poids. Cela permet d'accélérer la convergence du modèle et d'éviter les oscillations lorsque le gradient fluctue de manière importante d'une itération à l'autre.

Backpropagation

1. Initialiser aléatoirement les poids.
- Générer des valeurs aléatoires pour les matrices de poids
- Plage typique : intervalle $[-0.5, 0.5]$

2. Calculer les sorties yk **pour un vecteur d'entrée donné** X**.**

$$y_k = f\left(\sum_j w_{jk} y_j\right)$$

$$y_j = f\left(\sum_i w_{ij} x_i\right)$$

$$f(s) = \frac{1}{1 + e^{-s}}$$

3. Pour chaque neurone de sortie, calculer :

$$\delta_k = (y_k - t_k) f'(y_j)$$

$$f'(y_j) = y_{k(1-y_k)}$$

est la dérivée de la fonction sigmoïde, et t_k est la sortie désirée du neurone de sortie k

4. Pour chaque neurone de la couche cachée, calculer :

$$\delta_j = \sum_k w_{jk} f'(x_i)\delta_k$$

$$f'(x_i) = y_{j(1-y_j)}$$

5. Mettre à jour les poids du réseau comme suit :

$$w_{jk}(t+1) = w_{jk}(t) - \eta\delta_k y_j$$

$$w_{ij}(t+1) = w_{ij}(t) - \eta\delta_j x_i$$

où η est le taux d'apprentissage, $0 < \eta < 1$

6. Répéter les étapes 2 à 5 pendant un nombre donné d'itérations ou jusqu'à ce que l'erreur soit inférieure à un seuil donné.

Fonction d'activation

Sigmoïde

La fonction sigmoide est bornée entre $[0; 1]$

Tanh

La fonction tangeante hyperbolique est bornée entre $[-1; 1]$

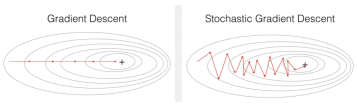
ReLU

La fonction ReLU est bornée entre $[0; +\infty]$

> **Note**
> Si plus de deux classes en sortie alors il faut utiliser la fonction **softmax**!

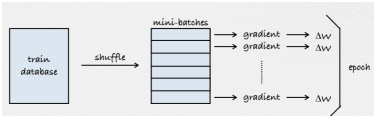
Variantes du calcul du gradient

1. **Descente de gradient par lot** : Le gradient est calculé à partir de l'ensemble de données complet, offrant des mises à jour plus précises mais plus lentes.
2. **Descente de gradient stochastique** : Le gradient est calculé à partir d'un seul échantillon, ce qui accélère les mises à jour mais peut rendre la convergence moins stable.



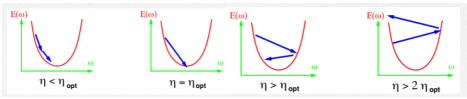
Descente de gradient par mini-batch

La descente de gradient par mini-batch utilise un sous-ensemble aléatoire des données (mini-batch) pour calculer une approximation stochastique du gradient exact. Cela réduit la charge computationnelle et accélère les itérations, mais avec un taux de convergence plus faible comparé à la descente de gradient par batch.



Méthodes de descente de gradient

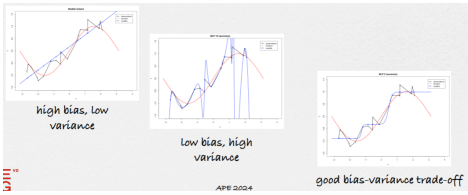
1. **SGD** : Mise à jour des poids après chaque échantillon ou mini-lot, accélère l'apprentissage mais avec une convergence moins stable.
2. **RMSprop** : Ajuste le taux d'apprentissage en fonction des gradients passés, stabilise les mises à jour pour de meilleures performances.
3. **Adam** : Combine SGD et moment, adapte les taux d'apprentissage pour chaque paramètre, accélère la convergence et améliore l'efficacité.
4. Learning Rate :



Compromis biais-variance

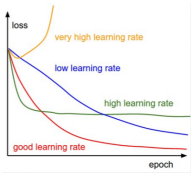
- **Biais** : Erreur systématique du modèle, souvent due à un modèle trop simple.
- **Variance** : Sensibilité du modèle aux variations des données d'entraînement, souvent liée à un modèle trop complexe.

L'objectif est de trouver un équilibre entre biais et variance pour minimiser l'erreur globale et améliorer la généralisation.



Effet du taux d'apprentissage

- Faible taux d'apprentissage (ligne bleue) : Les améliorations semblent linéaires, avec des mises à jour plus petites à chaque itération. Converger lentement, mais avec plus de précision, réduisant le risque de sauter par-dessus un minimum optimal.



- Haut taux d'apprentissage (ligne verte) : Un taux d'apprentissage élevé permet de réduire rapidement la perte, mais il peut entraîner une convergence trop rapide et "sauter" des minima locaux, ce qui peut conduire à des résultats sous-optimaux. Les mises à jour importantes peuvent faire osciller la perte et l'empêcher d'atteindre un minimum global.

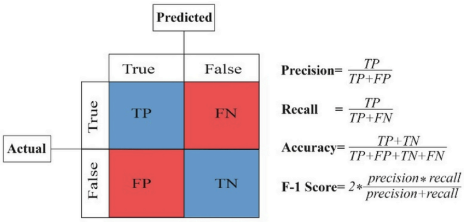
Cross-Validation

Données et Configuration
- Total de 1 000 observations
- 20% réservés pour l'ensemble de test (200 observations)
- 800 observations pour entraînement et validation
- Validation croisée 10-fold

Caractéristiques Principales
- Division de la base d'entraînement/validation 10 fois
- Taille des batchs constante
- Batch de 80 observations
- Un batch pour validation, reste pour entraînement
- Séparation typique : 80/720 observations

Calculs de Mise à Jour des Poids
- Mise à jour des poids : 720 fois
- Avec un batch de 10 observations : 720/10 = 72 mises à jour

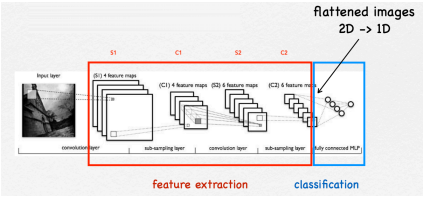
Matrice de confusion



CNN

Les **convolutional neural networds** sont des réseaux de neurones utilisant des opérations de convolution pour extraire des caractéristiques (features) des données d'entrées, en particulier des iamges.

Couches



Convolutionnelles (S1, S2)

Les couches dites **convolutionnelles** appliquent des filtres (kernel) sur les données d'entrée pour produire des cartes de caractéristiques (feature maps). Ces filtres permettent de détecter des motifs type: **bordure**, **textures**, etc..

Sous-échantillonnage (C1, C2)

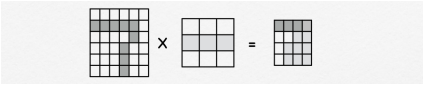
Les couches de sous-échantillonage réduisent les dimensions des cartes de caractéristiques en appliquant des opérations comme le **max pooling** ou l'**average pooling**. Cela permet de réduire le nombre de paramètres et de contrôler le sur-ajustement

Entièrement connectées (classification)

À la fin des couches convolutionnelles et de pooling, les caractéristiques extraites sont aplaties et passées à travers des couches entièrement connectées (comme dans un MLP) pour effectuer la classification ou la régression.

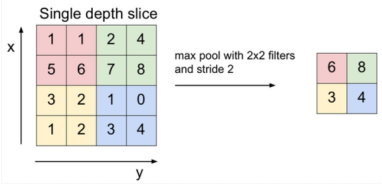
Convolution

Permet de détecter des motifs sur toutes les images d'entrées



Maxpooling

Statistiques récapitulatives de la détection des motifs



Attention
Dans le cas d'un **Global average pooling**, on calcule la moyenne de chaque carte de caractéristiques au lieu de prendre le maximum.

CNN shallow

- **Structure** : 1 à 3 couches convolutionnelles et de pooling.
- **Utilisation** : Pour des tâches simples avec des jeux de données de petite taille ou des motifs simples.
- **Avantages** : Moins de paramètres, réduisant le risque de sur-ajustement. Entraînement rapide et moins exigeant en termes de ressources.
- **Inconvénients** : Capacité limitée à capturer des motifs complexes. Moins performant pour des tâches complexes de vision par ordinateur.

CNN deep

- **Structure** : Plus de 10 couches convolutionnelles et de pooling, parfois des centaines (e.g., ResNet, VGG).
- **Utilisation** : Pour des tâches complexes avec des jeux de données volumineux ou des motifs complexes.
- **Avantages** : Capacité supérieure à capturer des motifs complexes et des relations hiérarchiques. Excellente performance sur des tâches complexes de vision par ordinateur.
- **Inconvénients** : Entraînement long et exigeant en termes de ressources. Plus de paramètres, augmentant le risque de sur-ajustement et nécessitant des techniques de régularisation.

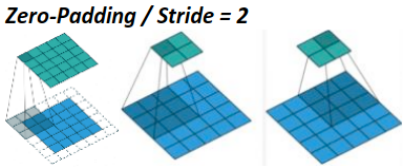
Fonction d'activation

ReLU	Leaky ReLU
$g(z) = \max(0, z)$	$g(z) = \max(0.01 \times z, z)$

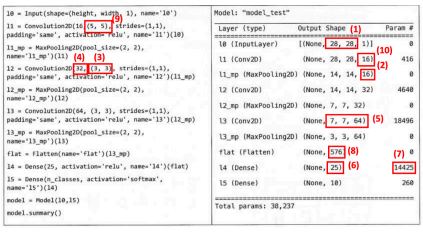
Note
Dès 2006, l'utilisation de la fonction **Leaky ReLU** à été popularisée pour les CNN, car elle permet de résoudre le problème de la disparition du gradient (Vanishing gradient) rencontré avec les fonctions d'activation sigmoïde et tanh.

Stride et padding

- **Stride**: déplacement du filtre lors de la convolution. Un stride de 1 signifie que le filtre se déplace d'un pixel à la fois, tandis qu'un stride de 2 signifie qu'il se déplace de deux pixels.
- **Padding**: ajout de pixels autour de l'image pour contrôler la taille de la sortie.



CNN Keras



1. Taille de l'image traitée : 28x28 pixels
2. Nb image sortie L1 vers L2 = 16
3. Taille image L2 = 3x3 pixels
4. Nb filtres L2 = 32 -> chaque filtreà 3x3 coefficients donc 9
5. Dernière couche L3, il y a 64 images de 7x7 pixels
6. Nb neurones cachés en L4 *Dense* = 25
7. Nb poids reliant L4 à L5 = 14425
9. Taille du filtre en L1 = 5x5 pixels

Nb paramètres en L1
- il y a un filtre de 5x5 pixels (9), avec 16 images d'entrée (10) + 16 donc $5 * 5 * 16 + 16 = 416$

Nb paramètres en L4
- il y a 576 data entrant (8) qui vont dans 25 neurones (6) et chaque neurone a un biais donc $576 * 25 + 25 = 14425$

CNN architecture

LeNet-5
- **Quoi** : Une des premières architectures CNN, conçue pour la reconnaissance de chiffres manuscrits
- **Comment** : Utilisé pour des tâches de reconnaissance de motifs simples comme les chiffres
- **Contexte** : Tâches de reconnaissance de caractères manuscrits (ex. codes postaux)
- **Pourquoi** : Simplicité et efficacité pour les tâches simples avec des données limitées

AlexNet (2012)
- **Quoi** : Un CNN plus profond qui a gagné le défi ILSVRC 2012
- **Comment** : Utilisé pour des tâches de classification d'images à grande échelle
- **Contexte** : Concours de vision par ordinateur, applications nécessitant une bonne précision
- **Pourquoi** : Bonne performance pour les tâches complexes grâce à l'utilisation de ReLU et du dropout

ZF Net (2013)
- **Quoi** : Une amélioration d'AlexNet, avec des filtres plus petits et plus de couches
- **Comment** : Classification d'images et détection d'objets
- **Contexte** : Tâches nécessitant une analyse fine des images
- **Pourquoi** : Meilleure performance que AlexNet sur les tâches complexes

VGG Net (2014)
- **Quoi** : Un CNN avec une architecture simple mais profonde utilisant des filtres 3x3
- **Comment** : Classification d'images, segmentation d'images
- **Contexte** : Recherche en vision par ordinateur, applications nécessitant une haute précision
- **Pourquoi** : Simplicité de conception et bonne performance sur des tâches complexes grâce à sa profondeur

GoogLeNet (Inception, 2015)
- **Quoi** : Un CNN avec des modules d'Inception qui permettent de capturer des informations à différentes échelles
- **Comment** : Classification d'images, reconnaissance d'objets
- **Contexte** : Applications nécessitant une haute précision et une efficacité computationnelle
- **Pourquoi** : Efficacité et capacité à capturer des motifs à différentes échelles
- **Inception** : Utilisation de plusieurs tailles de filtres en parallèle pour capturer des motifs à différentes échelles

ResNet (2015)
- **Quoi** : Un CNN très profond avec des connexions résiduelles pour éviter la dégradation des performances
- **Comment** : Classification d'images, détection d'objets, segmentation d'images
- **Contexte** : Tâches très complexes nécessitant des réseaux profonds
- **Pourquoi** : Permet l'entraînement de réseaux très profonds sans dégradation des performances grâce aux connexions résiduelles

DenseNet (2017)
- **Quoi** : Un CNN où chaque couche est connectée à toutes les couches précédentes
- **Comment** : Classification d'images, détection d'objets, segmentation d'images
- **Contexte** : Applications nécessitant une propagation efficace des caractéristiques
- **Pourquoi** : Réduction du nombre de paramètres et meilleure propagation des caractéristiques

EfficientNet (2020)
- **Quoi** : Une famille de CNN optimisés pour l'efficacité computationnelle en équilibrant profondeur, largeur et résolution
- **Comment** : Classification d'images, détection d'objets, segmentation d'images
- **Contexte** : Applications nécessitant une haute précision et une efficacité computationnelle

- **Pourquoi** : Optimisation des ressources tout en maintenant une haute performance

Transfert learning

Principe du Transfer Learning

- Réutiliser un modèle pré-entraîné (ex: sur ImageNet) pour une nouvelle tâche.
- Motivation : Un modèle entraîné sur une grande base de données a appris des caractéristiques générales utiles pour d'autres tâches.

Étapes du Transfer Learning

1. Chargement du modèle pré-entraîné

- Utiliser un modèle entraîné sur une grande base (ImageNet : 1000 objets, 1000 - images/catégorie)
- Garder l'architecture des couches convolutionnelles

2. Freeze des couches

- Geler les premières couches (caractéristiques générales)
- Les poids ne sont pas mis à jour pendant l'entraînement
- Garde les détecteurs de caractéristiques de bas niveau

3. Modification de la sortie

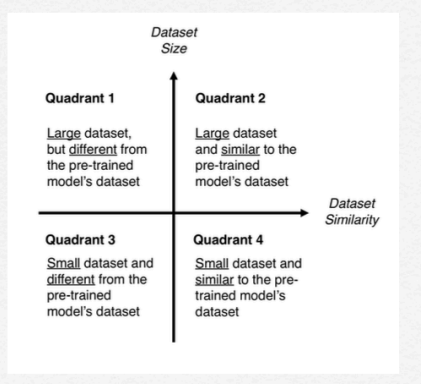
- Remplacer la dernière couche dense
- Adapter au nombre de classes de la nouvelle tâche
- Exemple : ImageNet (1000 classes) → COVID detection (2 classes)

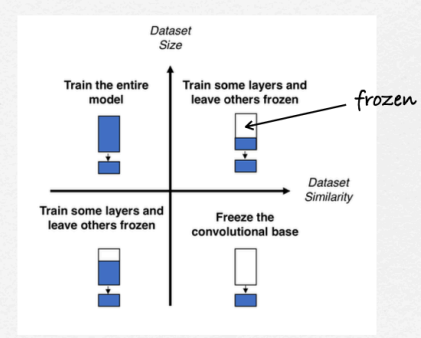
4. Fine-tuning

- Option A : Entraîner seulement les dernières couches
- Option B : Dégeler quelques couches et entraîner avec un petit learning rate
- Stratégie : Plus on a de données, plus on peut dégeler de couches

Avantages du Transfer Learning

- Moins de données nécessaires : Peut fonctionner avec des milliers au lieu de millions d'images
- Convergence plus rapide : Initialisation intelligente des poids
- Meilleures performances sur petits datasets
- Économie de ressources : Pas besoin de réentraîner depuis zéro



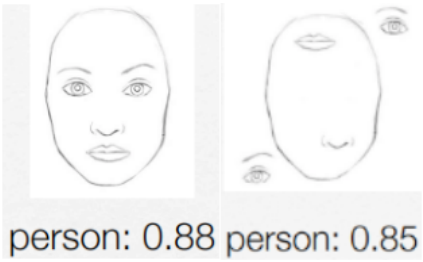


- Si le dataset est **petit** et **similaire** : on freeze la base convolutionnelle et on ne réentraîne que les dernières couches.
- Si le dataset est **plus grand** ou **moins similaire** : on peut débloquer (fine-tuner) certaines couches intermédiaires, voire tout réentraîner si le dataset est très différent et volumineux.

Deep troubles

Avantages des CNN

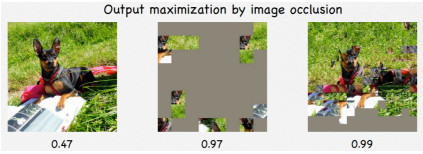
Offre une invariance à la translation spatiale. Il peut donc reconnaître un objet avec des tailles différentes même avec des positions « décalées ».



Corrélations "brute force"

Les CNN ne "comprennent" pas les données comme les humains, mais détectent des caractéristiques spécifiques pour prendre des décisions. Cela peut parfois les tromper.

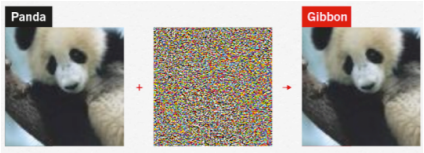
Les CNN peuvent être trompés par des caractéristiques anormales ou bruitées. Ils détectent des motifs sans comprendre le contexte global, ce qui les rend vulnérables à des manipulations subtiles.



Attaques Adversariales

Les CNN peuvent être trompés par des images modifiées de manière imperceptible pour les humains mais classées avec une grande confiance par le modèle.

Les attaques adversariales exploitent les faiblesses des CNN en ajoutant des perturbations spécifiques pour changer la classification. Cela pose des problèmes de sécurité et de fiabilité.



Techniques Contre le Surapprentissage

1. Réduction du Nombre de Paramètres

- Principe : Modèle plus simple → moins de risque de surapprentissage
- Méthodes :
  - Réduire le nombre de couches
  - Réduire le nombre de filtres par couche
  - Utiliser des filtres plus petits
  - Architectures plus efficaces (EfficientNet)

2. Data Augmentation

- Transformations géométriques :
  - Rotation : $[x', y'] = [x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta]$
  - Translation : $[x', y'] = [x + \Delta x, y + \Delta y]$
  - Zoom/Recadrage et retournement
- Transformations colorimétriques :
  - Modification luminosité/contraste
  - Perturbation des canaux RGB
  - Ajout de bruit

Effet : Augmente artificiellement la taille du dataset d'entraînement

3. Dropout

- Principe : Désactiver aléatoirement des neurones pendant l'entraînement
- Implémentation : Avec probabilité p (ex: 0.5), mettre le neurone à 0
- Effet : Force le réseau à ne pas dépendre d'un seul neurone, améliore la généralisation
- Usage : Principalement dans les couches denses, moins dans les couches convolutionnelles

4. Early Stopping

- Principe : Arrêter l'entraînement quand la performance sur validation cesse de s'améliorer
- Surveillance : Suivre l'erreur de validation à chaque époque

Implémentation : Patience (nombre d'époques sans amélioration avant arrêt)

5. Régularisation L1/L2

- L1 : Loss = Loss_original + $\lambda \sum |w_i|$
- L2 : Loss = Loss_original + $\lambda \sum w_i^2$
- Effet : Pénalise les poids trop importants, favorise la simplicité

Application deep learning

Types de Problèmes

Classification / Reconnaissance / Identification

Exemples : Détection de cancer, reconnaissance de chiffres, filtrage de spam, identification de visages et de commandes vocales.

But : Attribuer une classe ou une étiquette spécifique à chaque entrée de données.

Régression

Exemples : Prédiction de variables continues comme l'âge, les scores de plongée, le rendement agricole.

But : Prédire une valeur continue à partir des données d'entrée.

Réseaux de Neurones Entièrement Connectés vs. Convolutifs

Réseaux de Neurones Entièrement Connectés (Fully Connected Networks)

Utilisation : Traiter des données où chaque caractéristique est indépendante.

Exemples : Données tabulaires, régression, filtrage collaboratif pour recommandations.

Réseaux de Neurones Convolutifs (CNN)

Utilisation : Traiter des données avec des relations spatiales.

Exemples : Images, séries temporelles (sons, données de capteurs), vidéos.

Types de Convolutions

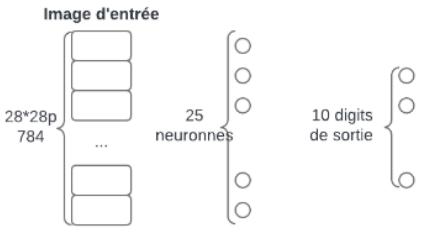
1D Convolutions : Utilisées pour les séries temporelles.

3D Convolutions : Utilisées pour les vidéos.

Helpful ressources

Calculer le nombre de paramètres d'un MLP

Dans le cas ou nous souhaitons calculer le nombre de paramètres d'un MLP pour traiter des images de 10 digits, 28x28 pixels, niveaux de gris. Le MLP est composé d'une seule couche cachée de 25 neurones:



$$\text{Total} = 28 * 28 * 25 + 25(\text{biais}) \\ +25 * 10 + 10(\text{biais}) = 19885$$