Université Sultan Moulay Slimane

Faculté Polydisciplinaire **Khouribga**

Systèmes d'Information et Intelligence Artificielle

Deep Learning

Chapitre 2 : Réseau de Neurones à Convolution

Pr. Ibtissam Bakkouri

i.bakkouri@usms.ma

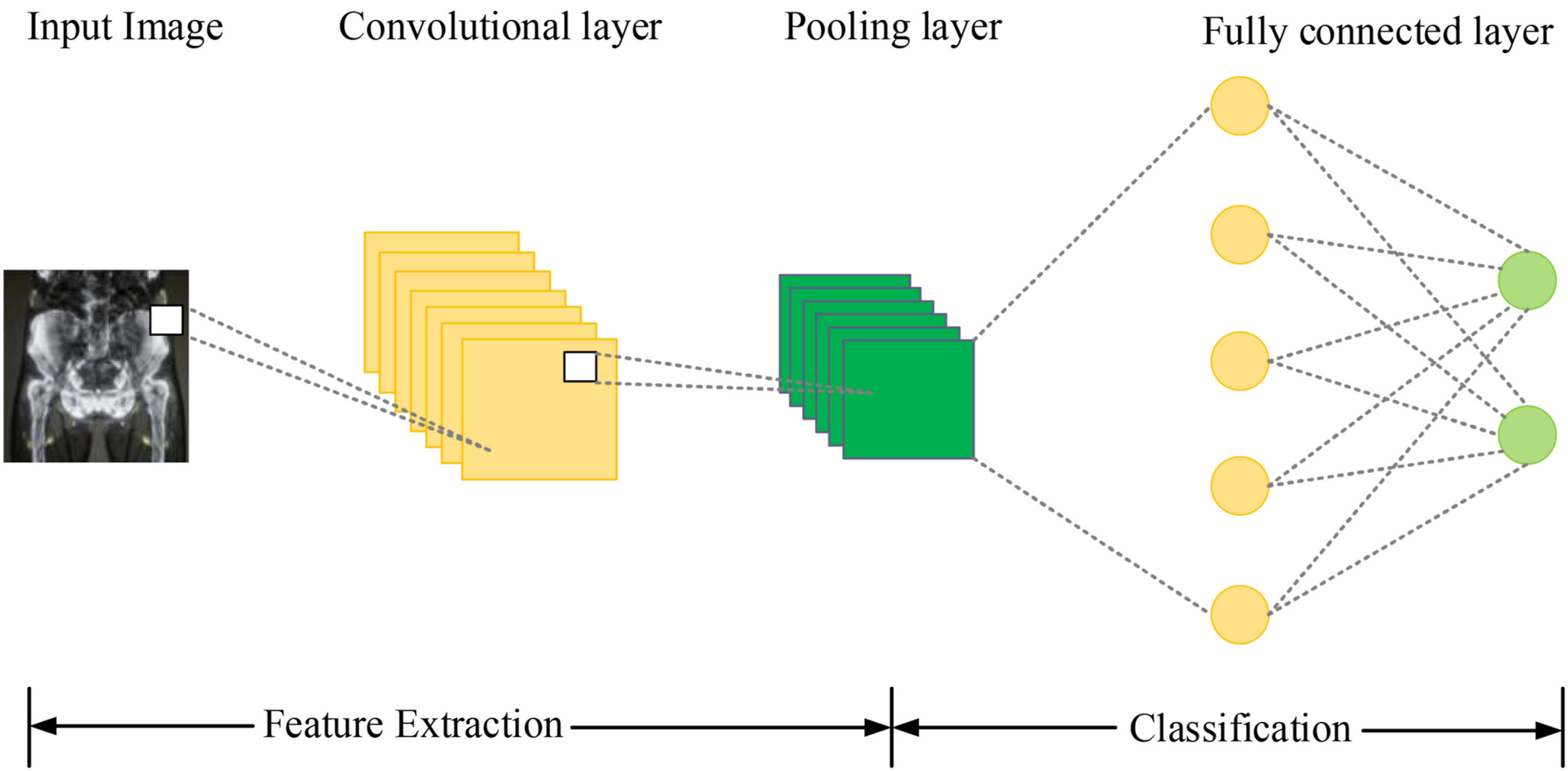
Année Universitaire : **2024/2025**

Plan

Réseau de Neurones à Convolution

- Les réseaux de neurones à convolution (CNN) sont une classe de modèles puissants pour le traitement des images et des signaux.
- Ils ont révolutionné des domaines tels que la vision par ordinateur, la reconnaissance vocale et le traitement du langage naturel.
- Un CNN est composé de plusieurs couches interconnectées, dont les couches de convolution, de normalisation, de sous-échantillonnage et entièrement connectées.

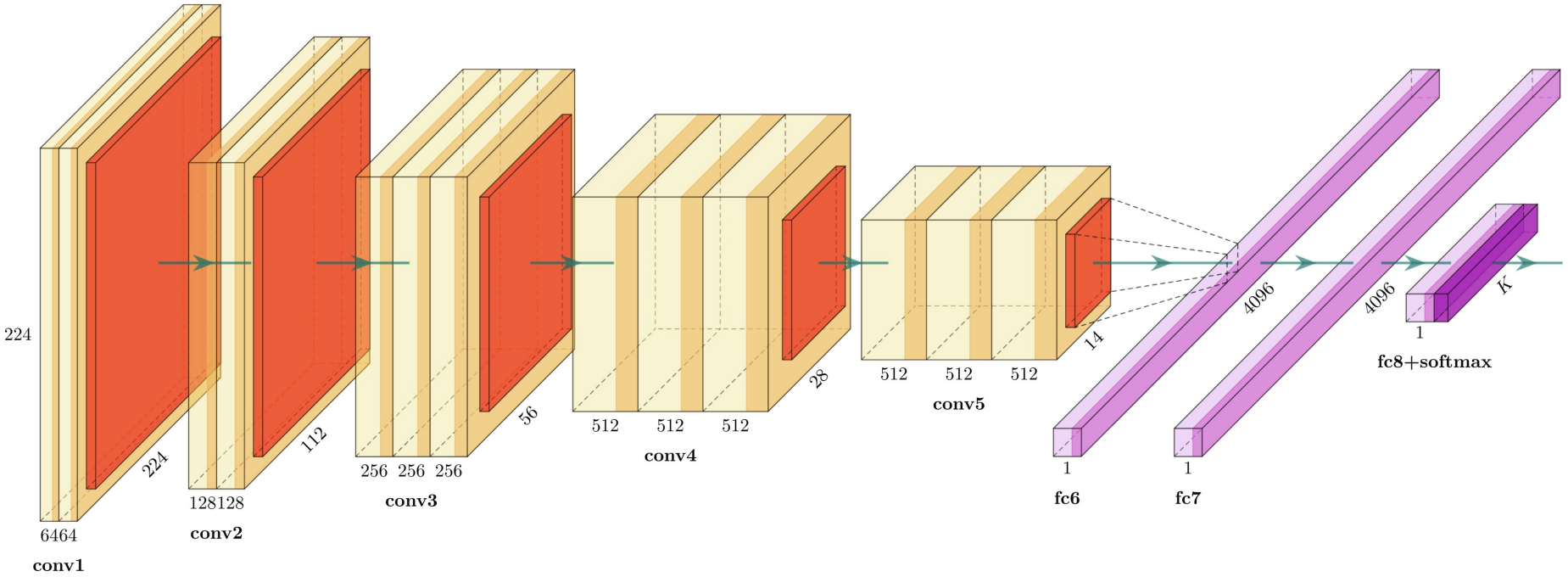
Réseau de Neurones à Convolution



Objectifs des Réseaux de Neurones à Convolution

- Extraire des caractéristiques hiérarchiques des données, en particulier des images.
- Réduire la complexité computationnelle tout en maintenant des performances élevées.
- Assurer une invariance à la translation des objets dans les images.
- Apprendre automatiquement des filtres (kernels) pour détecter les motifs locaux à différents niveaux de granularité.

Réseau de Neurones à Convolution



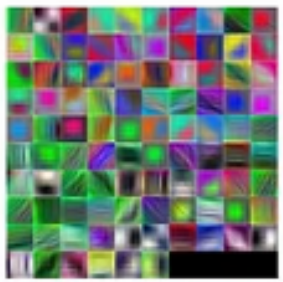
Importance des Convolutions dans les CNN

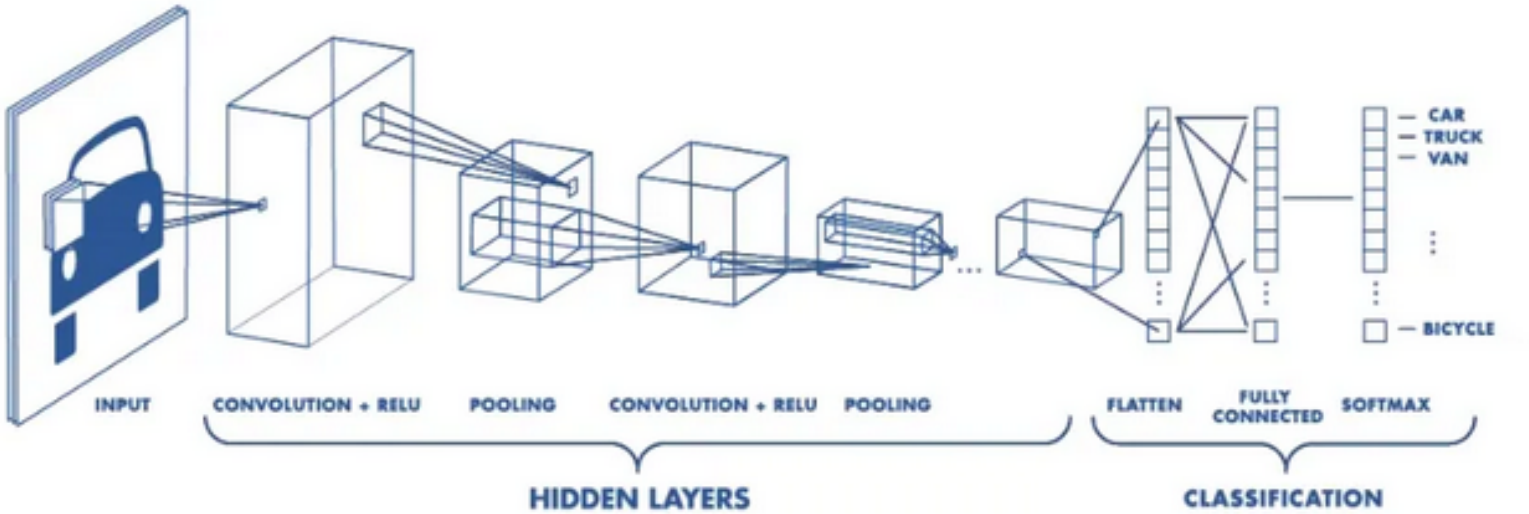
- La convolution est l'opération clé des CNN, permettant d'extraire des caractéristiques locales dans les images.
- Chaque filtre de convolution est appris pour détecter un motif particulier, comme les bords, les textures ou les objets.
- Ces caractéristiques sont ensuite combinées dans des couches plus profondes pour apprendre des représentations de plus en plus complexes.
- L'un des grands avantages des CNN est leur capacité à apprendre ces filtres directement à partir des données, sans intervention manuelle.

Couches de Convolution

- **Fonction principale** : Extraire les caractéristiques importantes des données d'entrée.
- **Opérations clés** :
  - Application de filtres (kernels) pour détecter des motifs spécifiques.
  - Capture des bords, textures, ou formes dans les images.

Couches de Convolution

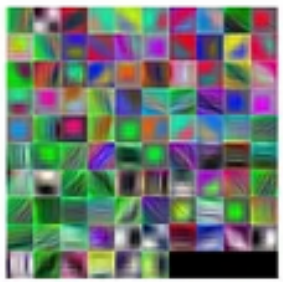


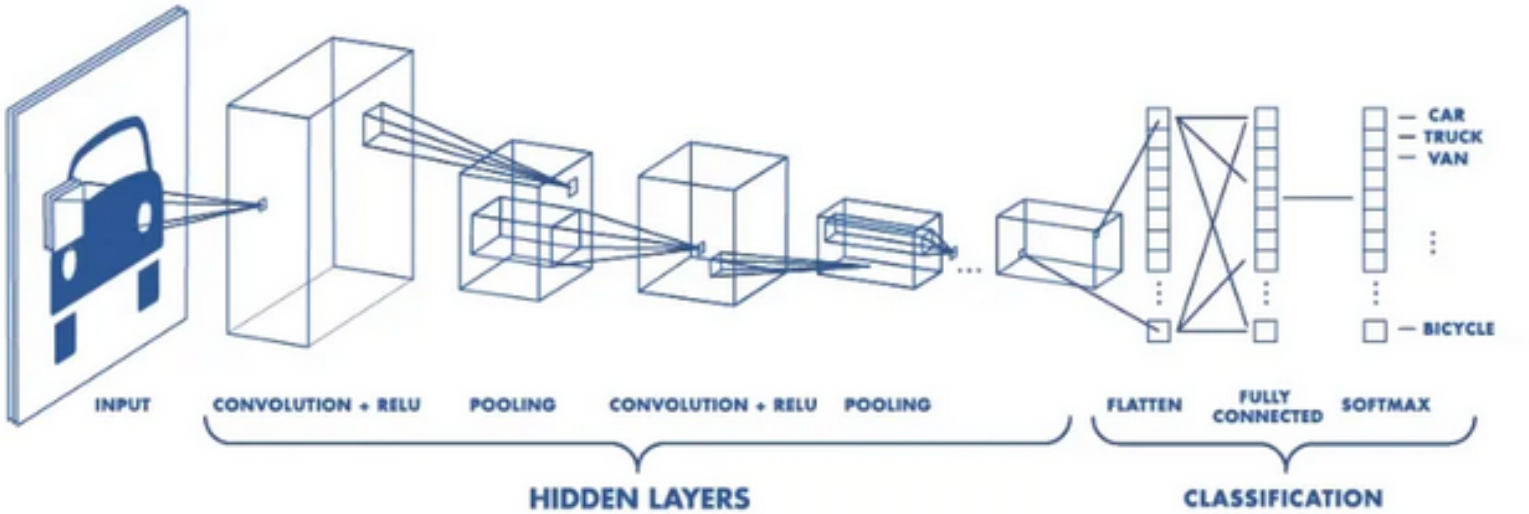


Couches de Convolution

- **Composants essentiels** :
  - **Filtres** : Matrices de poids apprises.
  - **Strides** : Nombre de pixels de déplacement du filtre.
  - **Padding** : Ajout de pixels pour préserver les dimensions.
- **Résultat** : Carte de caractéristiques des motifs détectés.

Couches de Convolution





Couches de Convolution

Equation générale :

$$C_{i+1}(m,n) = C_i(m,n) \times W_i + b_i$$

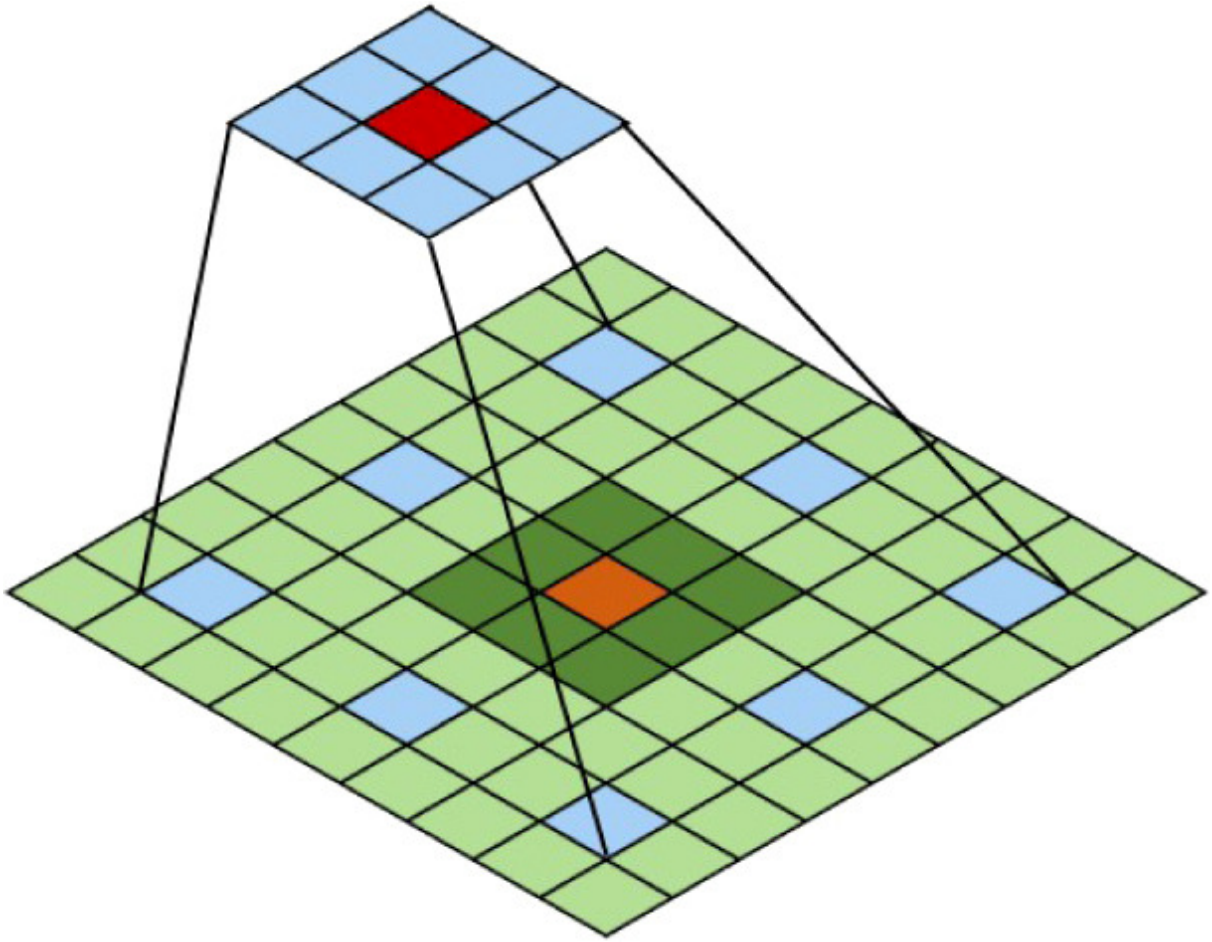
- La couche de convolution est utilisée pour extraire des caractéristiques importantes d'une entrée (images, signaux, etc.).
- $C_i(m,n)$: Représente l'entrée de la couche, qui peut être :
  - Une image brute (première couche).
  - Une carte de caractéristiques des couches précédentes.

Filtre et Convolution

Eléments clés de la convolution :

- W_i **(matrice de poids apprise)** :
  - Représente un filtre ou kernel détectant des motifs spécifiques (bords, textures, etc.).
  - Appris durant l'entraînement à l'aide de la rétropropagation.
- **Produit de convolution ($\times$)** :
  - Multiplie les valeurs de l'entrée avec celles du filtre de façon élément par élément.
  - La somme des produits est utilisée pour créer un point dans la carte de caractéristiques en sortie.
- La convolution est appliquée de manière glissante sur toute l'entrée.

Filtre et Convolution



Biais et Résultat de la Convolution

- b_i **(biais)** :
  - Une valeur scalaire ajoutée après la convolution.
  - Permet un ajustement plus précis des activations de sortie.
- **Résultat** $(C_{i+1}(m, n))$:
  - Représente la carte de caractéristiques produite en sortie.
  - Indique la présence des motifs détectés par le filtre aux différentes positions de l'entrée.
  - Chaque carte de caractéristiques est spécifique à un filtre.
- **Applications principales :**
  - Détection d'objets.
  - Reconnaissance d'images.
  - Analyse de signaux.

Calcul de la taille de la fenêtre de convolution

La taille de la fenêtre de convolution (Wout) peut être calculée avec la formule :

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - K + 2P}{S} \right\rfloor + 1$$

Où :

- W_{out} : Taille de la sortie (output)
- W_{in} : Taille de l'entrée (input)
- K : Taille du noyau (kernel)
- P : Padding (nombre de pixels ajoutés autour de l'entrée)
- S : Stride (pas de déplacement du noyau)

Exemple d'application de la formule

Soit les paramètres suivants :

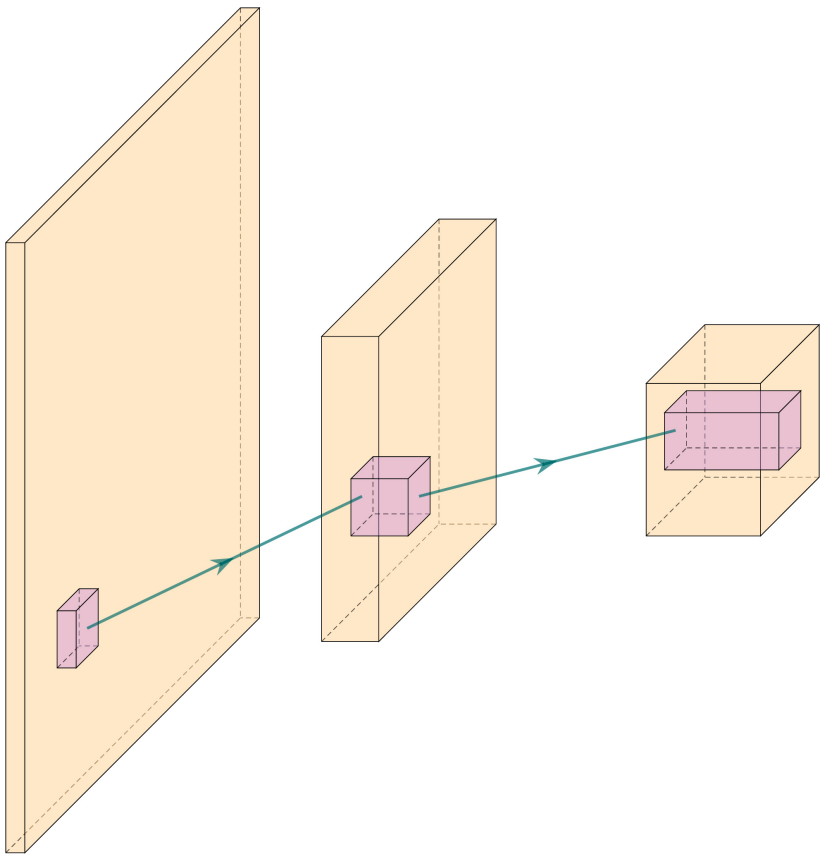
- Taille de l'entrée $W_{\text{in}} = 32$
- Taille du noyau $K = 3$
- Padding $P = 1$
- Stride $S = 2$

En appliquant la formule :

$$W_{\text{out}} = \left\lfloor \frac{32 - 3 + 2(1)}{2} \right\rfloor + 1 = \left\lfloor \frac{30}{2} \right\rfloor + 1 = 15 + 1 = 16$$

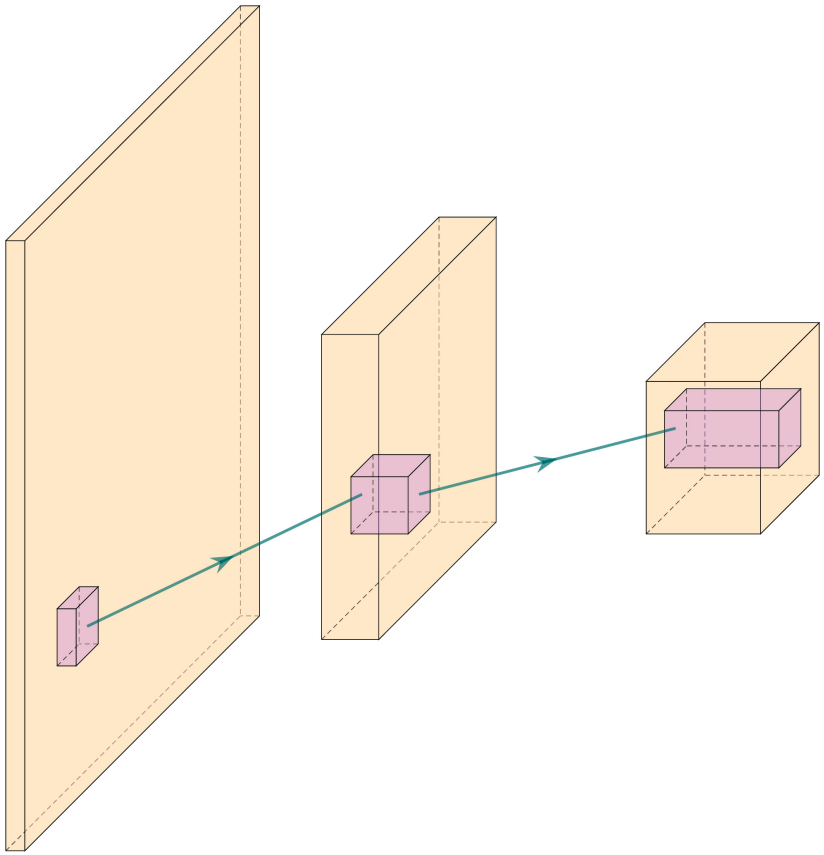
La taille de la sortie est donc $W_{\text{out}} = 16$.

Calcul de la taille de la fenêtre de convolution



Lorsque l'on applique des convolutions avec **padding = 0**, aucune valeur n'est ajoutée autour de l'entrée, ce qui réduit la taille de la fenêtre de sortie par rapport à l'entrée. Avec **stride = 1**, la fenêtre de convolution se déplace d'un pixel à la fois, ce qui peut entraîner une réduction supplémentaire de la taille de la sortie.

Calcul de la taille de la fenêtre de convolution



En combinant **padding = 0** et **stride = 1**, la taille de la sortie sera plus petite que celle de l'entrée, en fonction de la taille de la fenêtre de convolution.

Couches d'Activation

Les couches d'activation sont cruciales dans les réseaux de neurones pour introduire de la non-linéarité, ce qui permet au modèle d'apprendre des relations complexes.

- Sans activation, un réseau de neurones serait simplement une combinaison linéaire de ses entrées.
- Les fonctions d'activation permettent au réseau d'apprendre des patterns plus complexes et non-linéaires.

Fonction ReLU (Rectified Linear Unit)

La fonction ReLU est l'une des plus couramment utilisées pour les couches d'activation dans les réseaux de neurones :

$$h(a) = \max(0, a)$$

- Si $a \geq 0$, $h(a) = a$.
- Si $a < 0$, $h(a) = 0$.

Avantages de ReLU :

- Simple et efficace.
- Résout le problème du gradient qui disparaît, contrairement à des fonctions comme la sigmoïde.

Autres Fonctions d'Activation

- **Sigmoïde** :

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  - Utilisée dans les réseaux de neurones pour des sorties entre 0 et 1.
  - Problème de vanishing gradient pour des valeurs extrêmes.
- **Tanh (Tangente Hyperbolique)** :

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

  - Sortie entre -1 et 1.
  - Moins sensible que la sigmoïde aux valeurs extrêmes.

Non-linéarité et Amélioration de l'Apprentissage

Après une opération de convolution, la fonction d'activation introduit de la **non-linéarité** dans le réseau. Cela permet au réseau de :

- **Apprendre des relations complexes** entre les données, ce qui serait impossible avec une combinaison linéaire seule.
- **Modéliser des fonctions non linéaires**, ce qui améliore la capacité à résoudre des tâches complexes comme la classification d'images.

Gestion du Problème du Gradient qui Disparaît

Certaines fonctions d'activation comme **ReLU** sont particulièrement efficaces pour éviter le **problème du gradient qui disparaît**, qui est fréquent avec des fonctions comme la sigmoïde :

- **ReLU** permet d'obtenir un gradient constant pour les entrées positives, ce qui accélère l'apprentissage.
- Contrairement à la sigmoïde, qui a un gradient très faible pour les valeurs extrêmes, ReLU maintient un gradient suffisant pour l'entraînement.

Sparsité de la Représentation et Efficacité

Les couches d'activation comme **ReLU** entraînent une **sparsité** dans les activations :

- Une grande partie des activations deviennent nulles (pour $a < 0$), ce qui rend la représentation plus **compacte**.
- Cette sparsité permet de **réduire la complexité** du modèle et d'augmenter son **efficacité**, en réduisant les ressources nécessaires pour le calcul.

Sous-échantillonnage

Le sous-échantillonnage (ou **Pooling**) est une opération qui permet de réduire la taille des matrices tout en conservant les caractéristiques essentielles des données.

- Réduit la complexité computationnelle.
- Aide à prévenir le sur-apprentissage (overfitting).
- Facilite l'invariance par rapport à la translation.

Sous-échantillonnage

Le **Sous-échantillonnage** est souvent appliqué aux cartes de caractéristiques (feature maps) générées par la convolution dans les réseaux de neurones convolutionnels (CNN).

- Les opérations les plus courantes : **Max-Pooling**, **Min-Pooling**, et **Average-Pooling**.
- Chaque opération prend une fenêtre de taille $k \times k$ et applique une fonction de réduction sur cette fenêtre.

Max-Pooling

Le **Max-Pooling** consiste à prendre la valeur maximale dans chaque sous-région (fenêtre) de taille $k \times k$ dans la carte de caractéristiques.

$$MP = \max(X)$$

Où X est un ensemble de valeurs dans une fenêtre $k \times k$.

- Cela aide à extraire les caractéristiques les plus saillantes.
- Couramment utilisé dans les réseaux de neurones pour ses avantages en termes de robustesse et de simplicité.

Exemple de Max-Pooling

Considérons une fenêtre de taille 2×2 appliquée sur une matrice d'entrée :

$$\begin{bmatrix} 5 & 2 & 9 & 3 \\ 8 & 4 & 7 & 6 \\ 1 & 6 & 2 & 7 \\ 3 & 5 & 8 & 4 \end{bmatrix}$$

Le **Max-Pooling** sur une fenêtre 2×2 extrait les valeurs maximales :

$$\begin{bmatrix} 8 & 9 \\ 6 & 8 \end{bmatrix}$$

Ici, chaque valeur est le maximum d'une sous-région 2×2.

Min-Pooling

Le **Min-Pooling** est une variation du **Pooling** où, au lieu de prendre la valeur maximale ou moyenne dans une fenêtre $k \times k$, on prend la valeur **minimale**.

- Min-Pooling aide à extraire les valeurs les plus faibles dans chaque région locale.
- Cette approche peut être utile pour des tâches où la minimisation ou la détection des faibles valeurs est importante.
- Fonctionnement similaire au Max-Pooling, mais au lieu de maximiser, on minimise les valeurs dans chaque fenêtre.

Exemple de Min-Pooling

Considérons une matrice d'entrée de taille 4×4 et une fenêtre de 2×2 :

$$\begin{bmatrix} 5 & 2 & 9 & 3 \\ 8 & 4 & 7 & 6 \\ 1 & 6 & 2 & 7 \\ 3 & 5 & 8 & 4 \end{bmatrix}$$

Appliquons un **Min-Pooling** avec une fenêtre 2×2 :

$$\begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix}$$

Ici, chaque élément de la sortie est la valeur minimale dans une sous-région 2×2.

Average Pooling

L'**Average Pooling** consiste à calculer la moyenne des valeurs dans chaque sous-région (fenêtre) de taille $k \times k$ d'une carte de caractéristiques :

$$AP = \frac{1}{k^2} \sum_{i=1}^{k} \sum_{j=1}^{k} x_{i,j}$$

où $x_{i,j}$ représente les éléments de la fenêtre $k \times k$.

- Cette opération réduit la dimensionnalité tout en préservant une représentation lissée des caractéristiques.
- Elle est moins agressive que le **Max Pooling** et permet de conserver davantage d'informations globales.

Exemple d'Average Pooling

Considérons une matrice d'entrée 4×4 :

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 4 & 6 & 5 & 8 \\ 7 & 8 & 9 & 10 \\ 2 & 4 & 6 & 8 \end{bmatrix}$$

L'**Average Pooling** avec une fenêtre 2×2 et un pas de 2 calcule la moyenne des valeurs dans chaque sous-région :

$$\begin{bmatrix} \frac{1+3+4+6}{4} & \frac{2+4+5+8}{4} \\ \frac{7+8+2+4}{4} & \frac{9+10+6+8}{4} \end{bmatrix} = \begin{bmatrix} 3.5 & 4.75 \\ 5.25 & 8.25 \end{bmatrix}$$

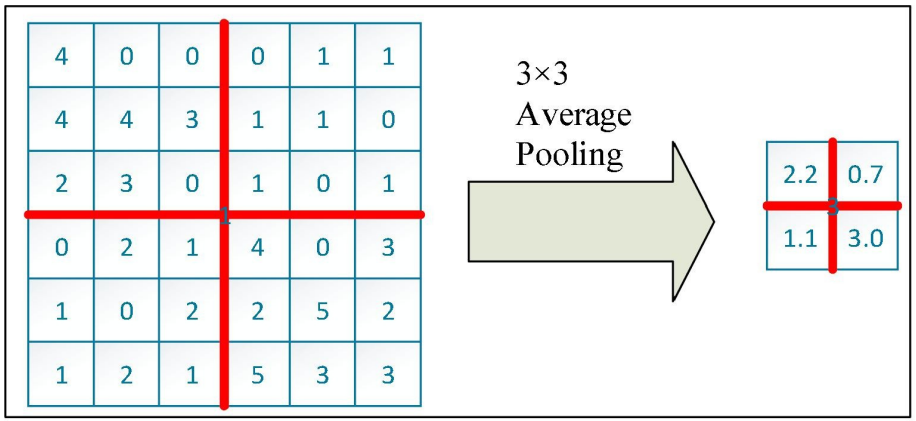
Ici, chaque valeur est la moyenne des éléments d'une sous-région 2×2.

Avantages du Pooling

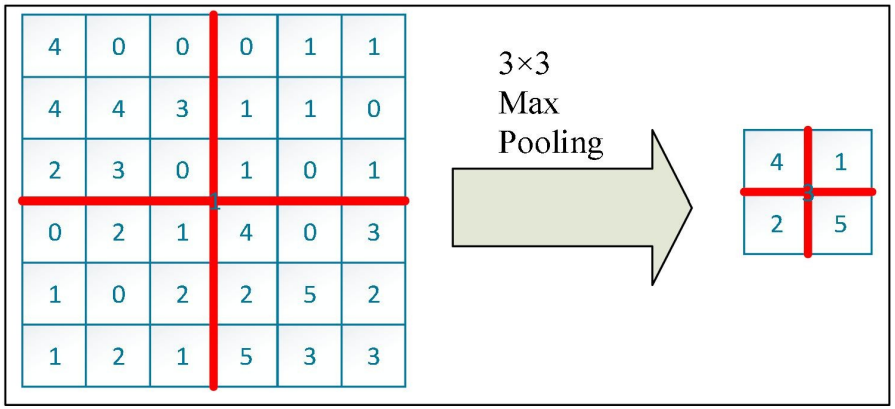
Le pooling présente plusieurs avantages dans les réseaux de neurones :

- **Réduction de la taille des données** : Diminue la complexité du modèle en réduisant la dimension des cartes de caractéristiques.
- **Invariance de translation** : Le modèle devient plus robuste aux déplacements des objets dans l'image.
- **Prévention du sur-apprentissage** : En réduisant la dimensionnalité, le pooling aide à éviter l'overfitting.

Couche de sous-échantillonnage



(a) Average pooling



(b) Maximum pooling

Normalisation

Objectif de la Normalisation :

- Améliorer la stabilité et la convergence de l'entraînement des réseaux de neurones.
- Réduire la dépendance des valeurs des entrées à leurs échelles.
- Permettre un apprentissage plus rapide grâce à une distribution uniforme des activations.

Les couches de normalisation ajustent les activations des neurones à l'aide d'opérations mathématiques spécifiques.

Batch Normalization (BN)

La **Batch Normalization** normalise les activations d'un lot (batch) de données :

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

où :

- x_i : activation avant normalisation.
- μ_B : moyenne des activations dans le batch.
- σ_B^2 : variance des activations dans le batch.
- ϵ : petite constante pour éviter la division par zéro.

Batch Normalization (BN)

Les activations normalisées sont ensuite transformées avec des paramètres apprenables :

$$y_i = \gamma \hat{x}_i + \beta$$

où γ et β sont les paramètres d'échelle et de décalage.

Layer Normalization (LN)

La **Layer Normalization** normalise les activations d'une couche entière au lieu d'un batch :

$$\hat{x}_i = \frac{x_i - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}}$$

où :

- μ_L : moyenne des activations dans une couche.
- σ_L^2 : variance des activations dans une couche.

Utilisée principalement dans les architectures séquentielles (par exemple, RNNs).

Instance Normalization (IN)

L'**Instance Normalization** est utilisée pour normaliser les activations d'une seule image (ou instance) au lieu d'un batch :

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_{I,j}}{\sqrt{\sigma^2_{I,j} + \epsilon}}$$

où :

- $\mu_{I,j}$: moyenne des activations pour le j-ème canal d'une image.
- $\sigma^2_{I,j}$: variance des activations pour le j-ème canal d'une image.

Utilisée principalement pour les tâches de style transfert et segmentation.

Group Normalization (GN)

La **Group Normalization** divise les activations en groupes avant la normalisation :

$$\hat{x}_{i,g} = \frac{x_{i,g} - \mu_G}{\sqrt{\sigma_G^2 + \epsilon}}$$

où :

- μ_G : moyenne des activations dans un groupe.
- σ_G^2 : variance des activations dans un groupe.

Avantage : fonctionne bien même avec de petits lots (*mini-batch sizes*).

Comparaison des Types de Normalisation

Différences clés entre les types de normalisation :

Type	**Avantages**
Batch Normalization (BN)	Accélère la convergence
Layer Normalization (LN)	Indépendant de la taille du batch
Instance Normalization (IN)	Adapté aux petites images
Group Normalization (GN)	Flexible, même avec petits lots

Résumé :

- Chaque méthode a ses propres avantages et est adaptée à des scénarios spécifiques.
- Le choix dépend de la taille du batch et des besoins de l'application.

Couche de Fusion

- La **Couche de Fusion** combine les caractéristiques extraites par différentes couches d'un réseau.
- Elle peut être utilisée pour :
  - Fusionner plusieurs cartes de caractéristiques.
  - Intégrer des informations provenant de multiples résolutions.
- **Objectif principal :** Améliorer la richesse des représentations pour une meilleure performance du modèle.

Types de Fusion

- **Fusion par addition** :

$$F_{\text{fusion}} = F_1 + F_2$$

Combine les caractéristiques élément par élément en additionnant leurs valeurs.

- **Fusion par concaténation** :

$$F_{\text{fusion}} = \text{Concat}(F_1, F_2)$$

Regroupe les caractéristiques en ajoutant une dimension supplémentaire.

Types de Fusion

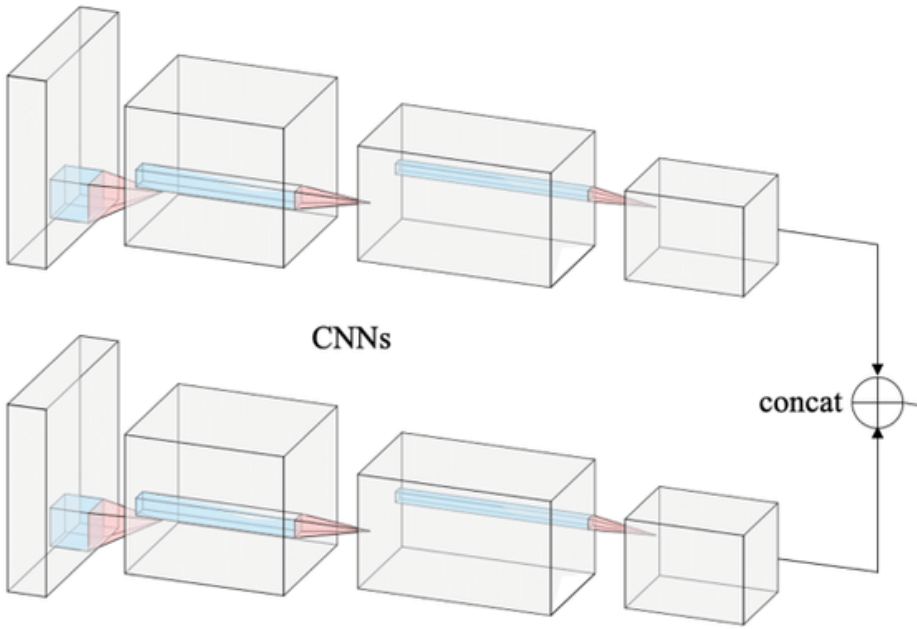
- **Fusion pondérée** :

$$F_{\text{fusion}} = \alpha \cdot F_1 + (1 - \alpha) \cdot F_2$$

Utilise des poids pour équilibrer les contributions.

Illustration de la Fusion

- **Addition** : Combine les intensités des pixels.
- **Concaténation** : Étend les dimensions pour inclure tous les canaux.
- **Fusion pondérée** : Ajuste les contributions selon les poids.



Applications de la Couche de Fusion

- **Vision par ordinateur** :
  - Fusion de cartes de caractéristiques multi-échelles pour la segmentation.
  - Utilisée dans les modèles de détection d'objets (par ex., FPN).
- **Traitement du langage naturel (NLP)** :
  - Fusion de représentations de mots provenant de différents niveaux.
  - Utilisée dans les modèles de traduction automatique.
- **Multimodalité** :
  - Combinaison des données texte et image pour des tâches comme la description d'image.

Couche Entièrement Connectée

- Une **Couche entièrement connectée (Fully Connected Layer)** connecte chaque neurone d'une couche à tous les neurones de la couche suivante.
- Elle est souvent utilisée en fin de réseau pour effectuer des prédictions ou des classifications.
- Chaque connexion est pondérée et contribue au calcul de l'activation :

$$y_i = \sigma\left(\sum_j w_{ij} x_j + b_i\right)$$

Introduction à la Couche Entièrement Connectée

- Chaque connexion est pondérée et contribue au calcul de l'activation :

$$y_i = \sigma\left(\sum_j w_{ij} x_j + b_i\right)$$

où :

- w_{ij} : poids entre le j-ème neurone de l'entrée et le i-ème neurone de sortie.
- b_i : biais associé au i-ème neurone.

Rôle de la Couche Entièrement Connectée

- **Fonction principale** : Extraire des relations globales entre les caractéristiques extraites par les couches précédentes.
- Les poids et biais sont optimisés pendant l'entraînement pour minimiser une fonction de perte :

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \hat{y}_i)$$

où :

  - y_i : valeur réelle.
  - $\hat{y}_i$: prédiction du réseau.

- Utilisée pour transformer les caractéristiques en probabilités dans des tâches de classification.

Fonction de Perte $\ell(y_i, \hat{y}_i)$

- La fonction de perte $\ell(y_i, \hat{y}_i)$ mesure l'écart entre la sortie prédite $(\hat{y}_i)$ et la sortie réelle (y_i).
- Exemples de fonctions de perte courantes :
  - **Erreur quadratique moyenne (MSE)** :

$$\ell(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$$

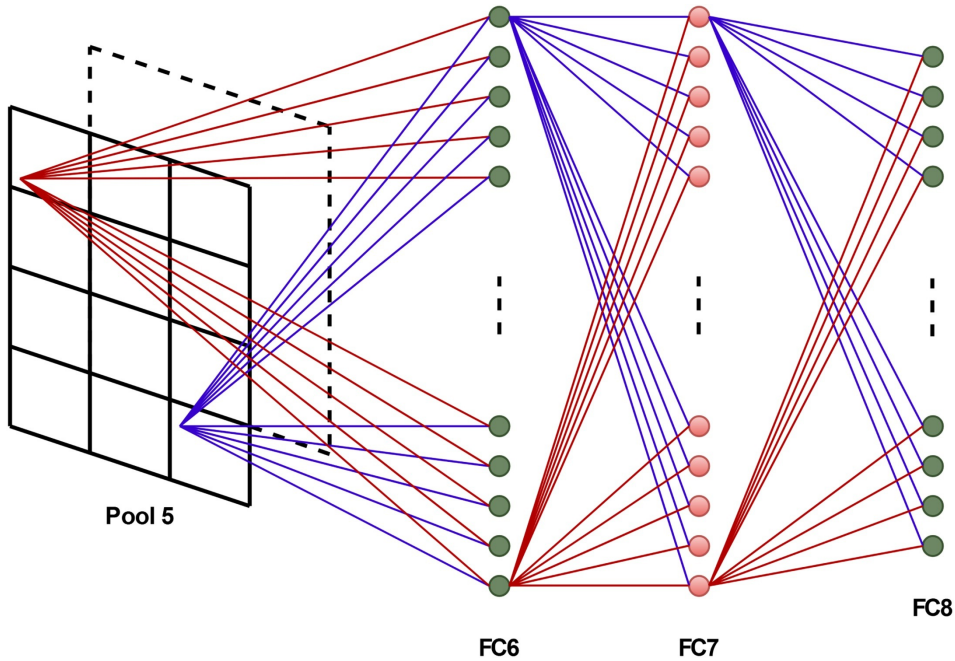
    Utilisée pour des tâches de régression.
  - **Entropie croisée (Cross-Entropy)** :

$$\ell(y_i, \hat{y}_i) = -\sum_k y_i^{(k)} \log(\hat{y}_i^{(k)})$$

    Utilisée pour des tâches de classification.

Représentation Visuelle d'une Couche Entièrement Connectée

- Chaque neurone de la couche d'entrée est connecté à tous les neurones de la couche suivante.
- La propagation se fait à travers des calculs pondérés.



Points Clés à Retenir

- Les réseaux de neurones à convolution **(CNN)** sont la pierre angulaire des architectures modernes de vision par ordinateur.
- Chaque couche (convolution, activation, sous-échantillonnage, normalisation, fusion, entièrement connectée) joue un rôle **crucial** dans l'extraction, la transformation et la prise de décision basée sur les données.
- L'interaction entre **ces couches** permet de traiter efficacement les images et d'autres types de données complexes.