SEI - SoC - CNN

S. Mancini

GRENOBLE INP UGA Phelma

Plan

- Projet d'intégration des SoC
- Réseau de neurones CNN
- Détails d'organisation
- CNN cible : CIFAR10

Objectifs

Objectif:
Réaliser un système de traitement HW et SW, et arriver jusqu'à une implémentation avec caméra et affichage

⇨Mettre en oeuvre par vous même toutes les méthodes et outils dont vous avez besoin.

Votre rôle

Vous avez tous le même algorithme, et vous partez d'une page blanche.

Les étapes de votre mission:

- Spécifier la solution
- Définir les techniques de résolution
- Mettre en oeuvre la solution
- Evaluer vos résultats, selon les critères usuels

⇨ Votre attitude est déterminante et vous devez chercher des solutions par vous même.

Notre rôle

- Nous vous donnons des indications méthodologiques
- Nous vous dépannons sur les outils ...
- ... lorsque vous avez exploré par vous même

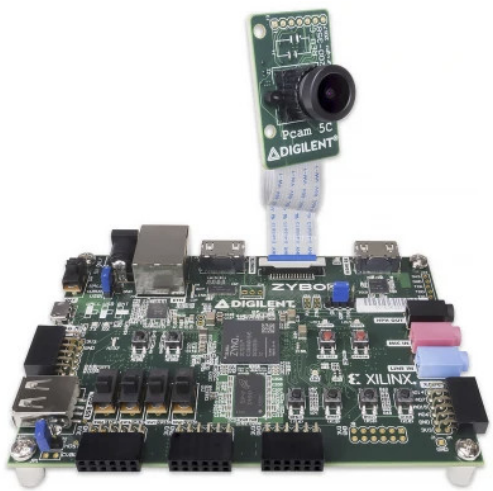
Les connaissances dont vous avez besoin

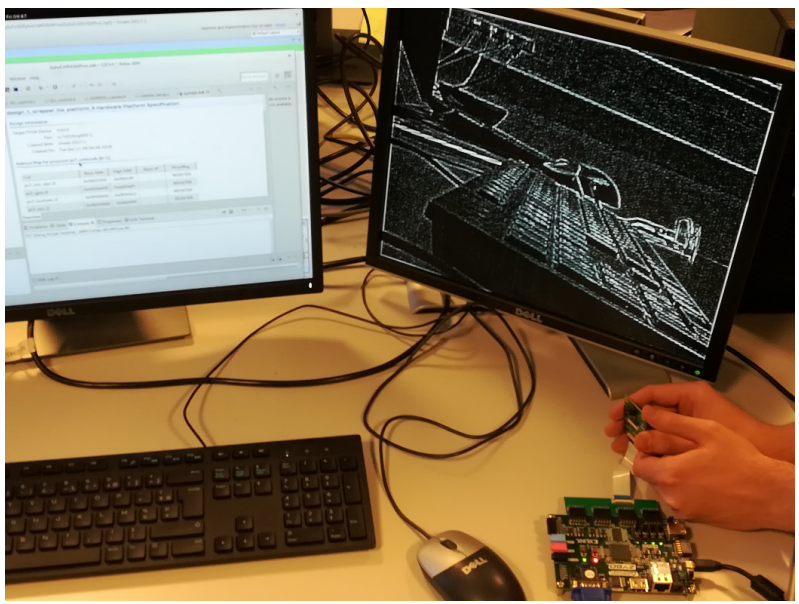
... et que vous avez!

- Passer d'une spécification algorithmique à une architecture de traitement HW ou SW
- Modélisation RTL (VHDL) et HLS (CatapultC)
- Informatique embarqué (C)
- Conception FPGA Xilinx
- Outils informatiques usuels (Linux)
- Python vous sera d'une grande aide

Séances

- Implémenter la référence algorithmique d'un CNN en Python
  3 séances
- Implémenter le CNN en virgule fixe pour la HLS, HLS et vérification
  3 séances
- Tests et émulation sur carte FPGA, avec caméra et affichage
  3 séances
- Evaluation
  Rapport + interview lors de la dernière séance

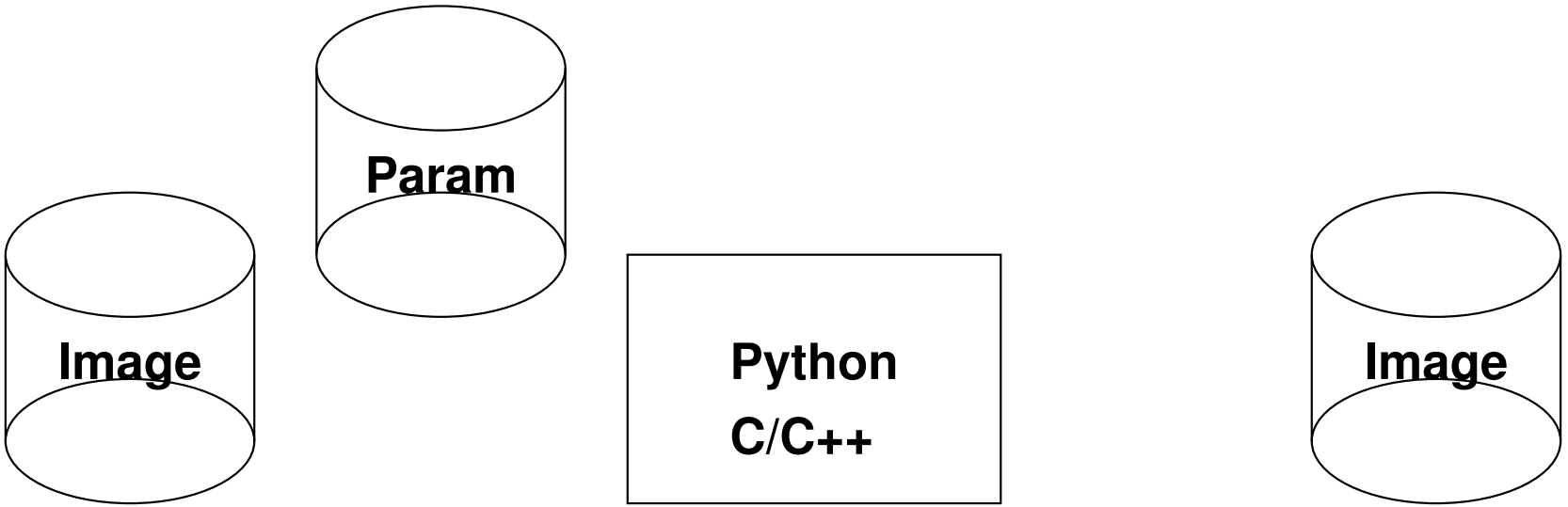




Méthodologie descendante

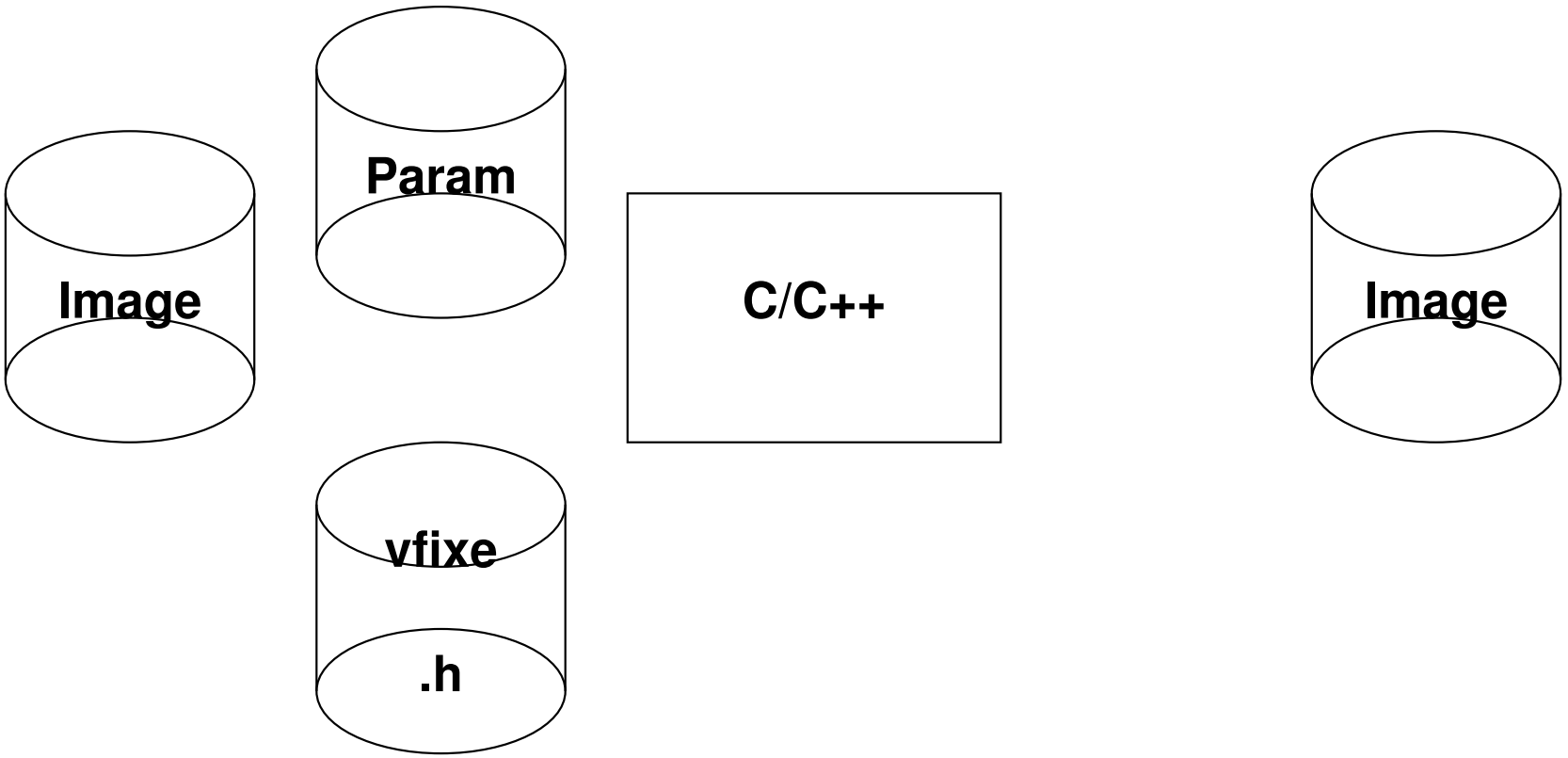
- Objectif:
  - Garantir le succès de l'implémentation sur carte
  - Supprimer le risque de bug impossibles à identifier sur FPGA
- Méthodologie:
  - Procéder par étapes successives, en ajoutant de nouveaux modèles petit à petit
  - Au cours des étapes, vérifier systématiquement les résultats intermédiaires par rapport à une référence
  - Anticiper la suite des étapes pour éviter les erreurs aux conséquences tardives

Etapes de la méthodologie HW



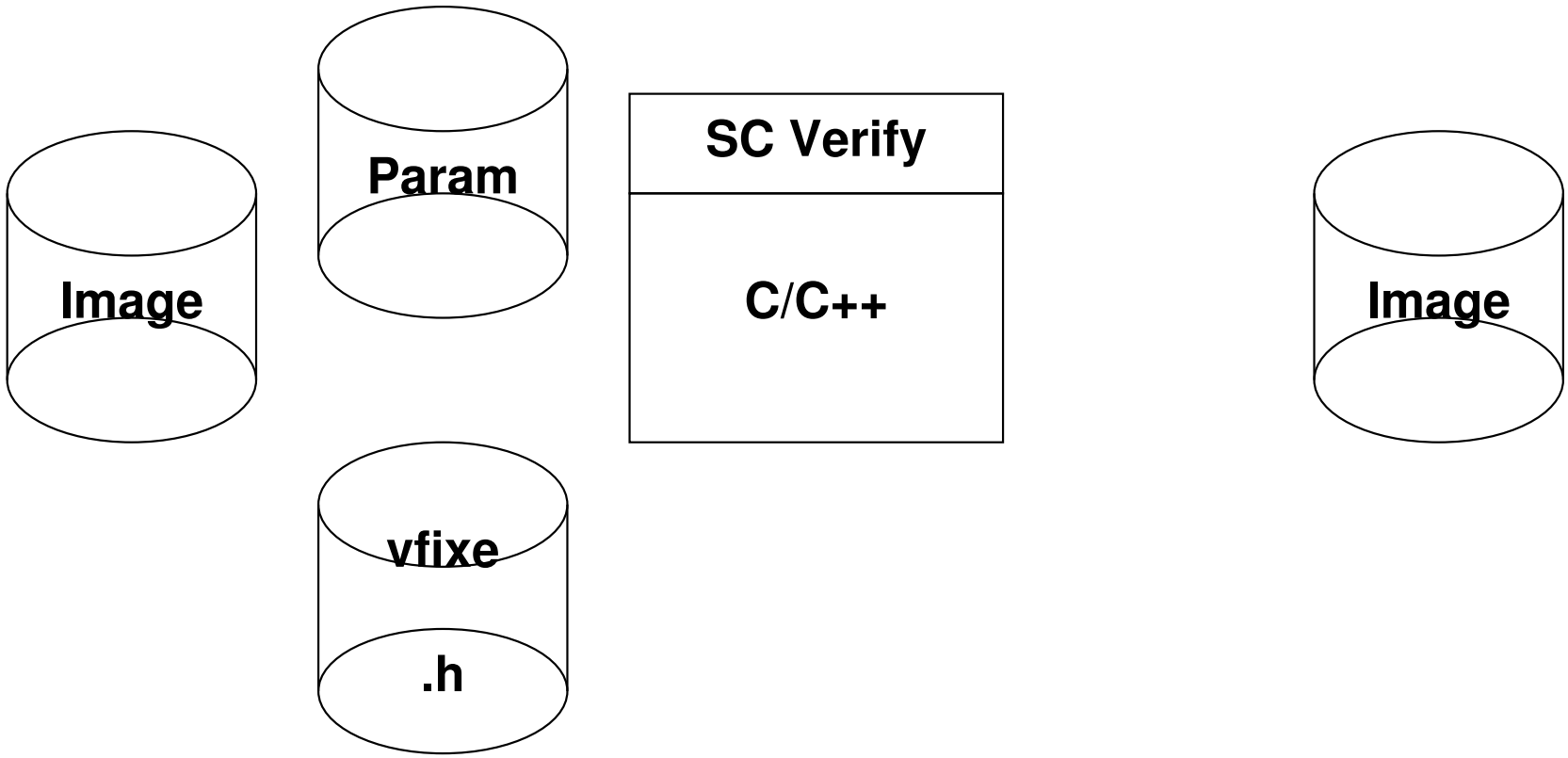
Référence Algorithmique

Etapes de la méthodologie HW



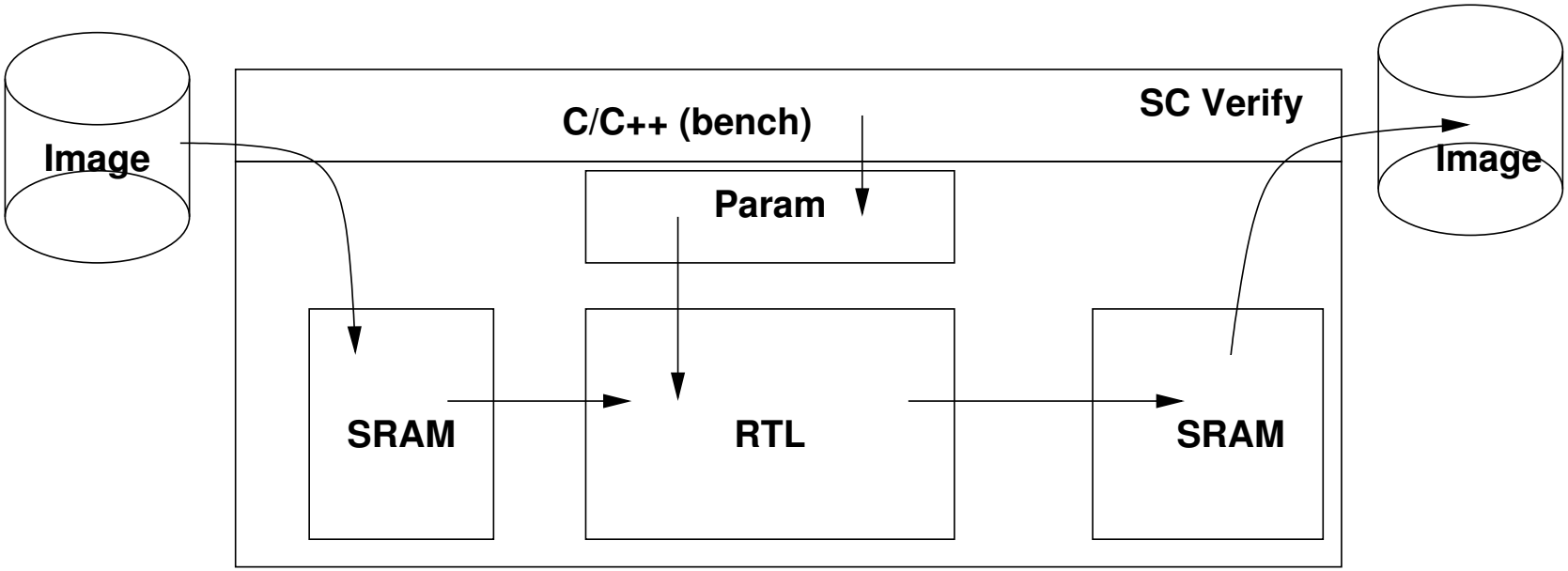
Algorithme en virgule fixe

Etapes de la méthodologie HW



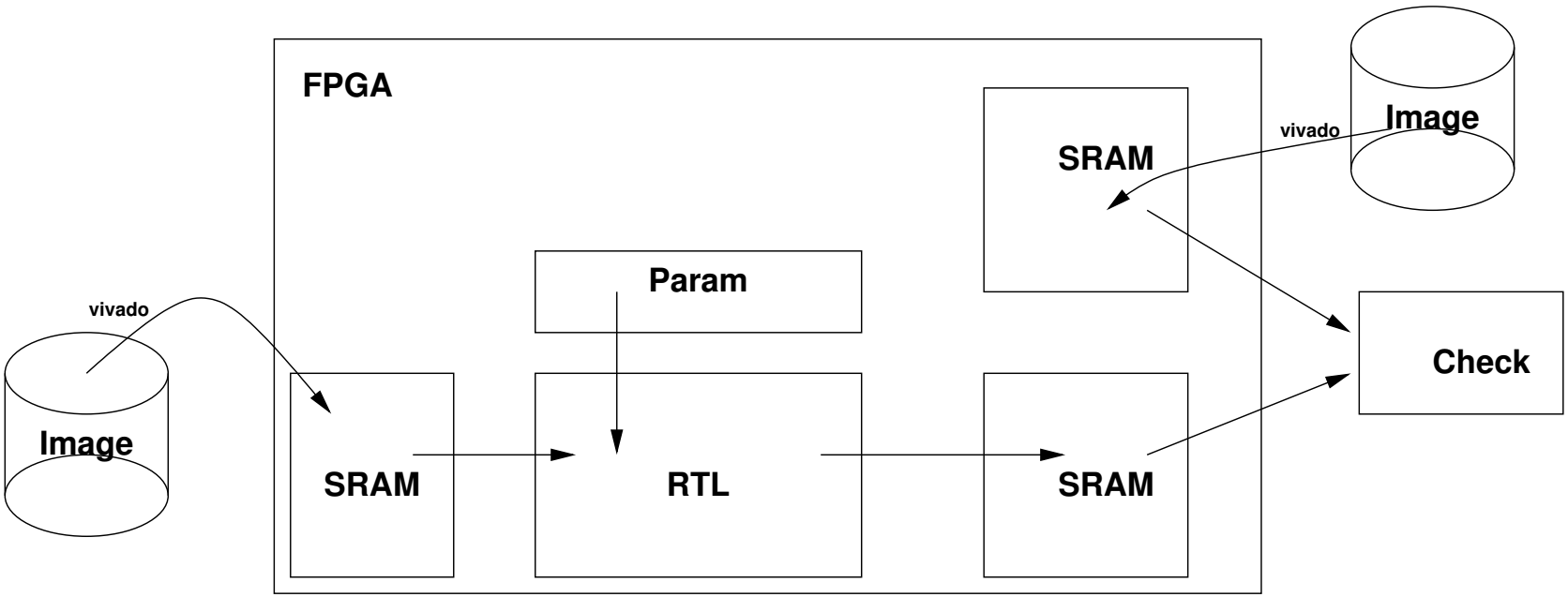
Virgule fixe HLS synthétisable

Etapes de la méthodologie HW



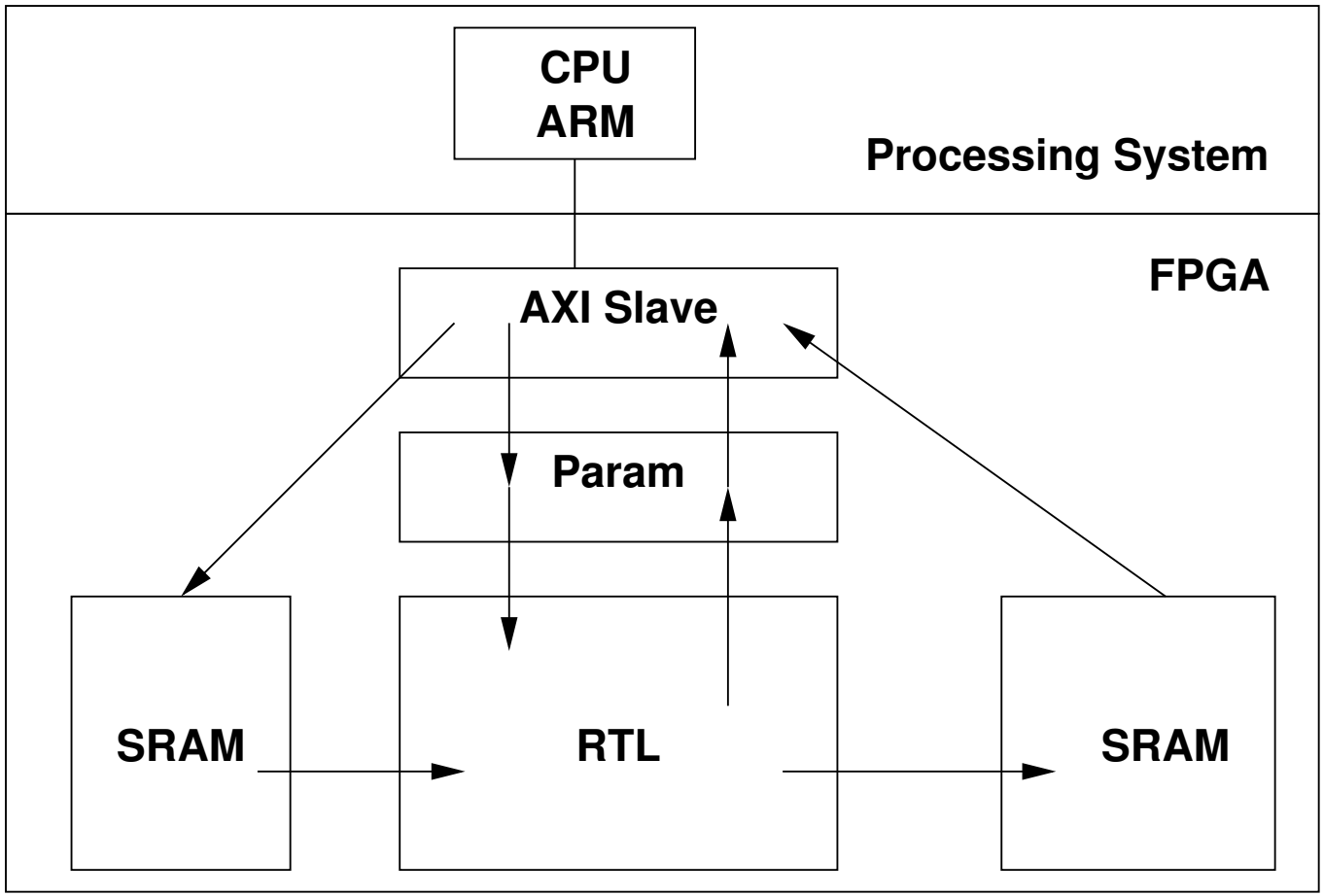
Vérification par rapport à des fichiers de référence

Etapes de la méthodologie HW



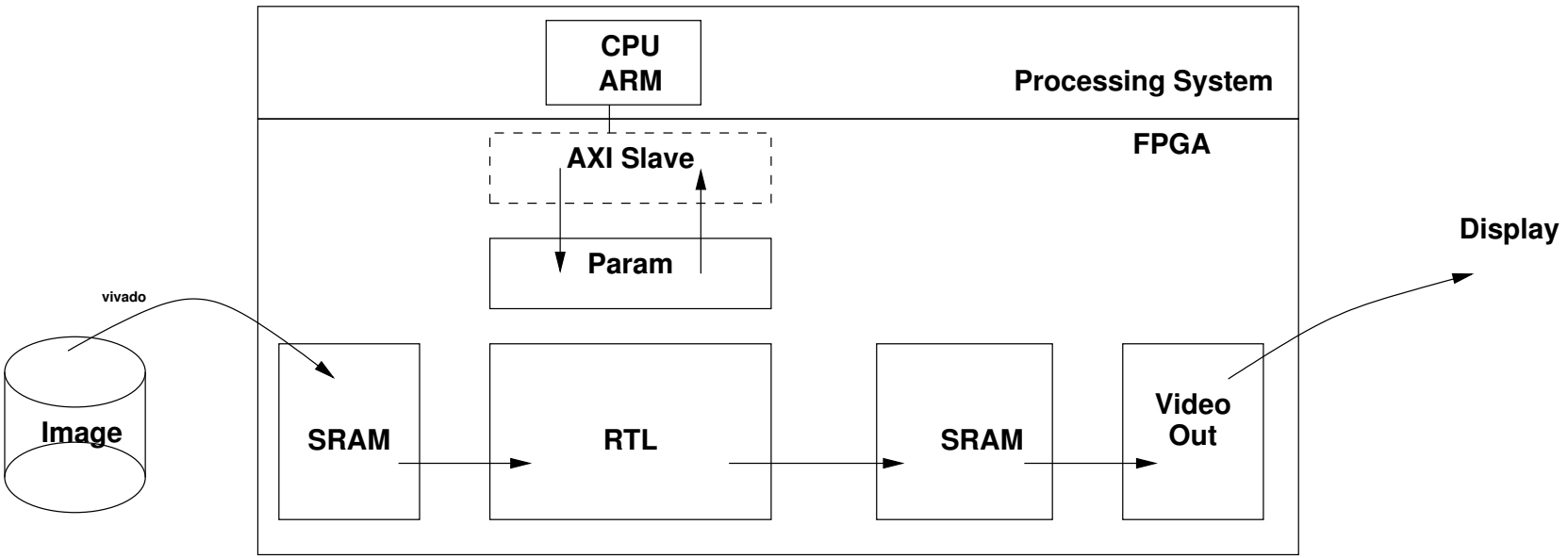
Emulation avec entrées de référence

Etapes de la méthodologie HW



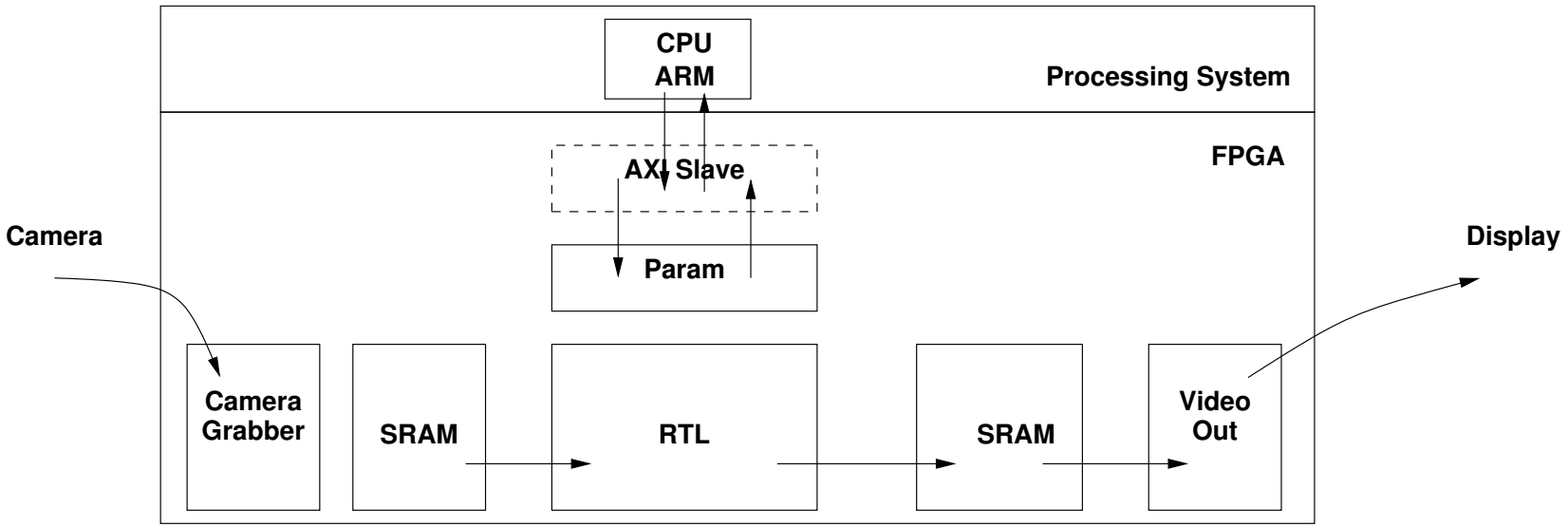
Paramétrage du HW par SW

Etapes de la méthodologie HW



Validation sur émulateur

Etapes de la méthodologie HW



Système avec entrée/sortie vidéo

Plan

- ❑ Projet d'intégration des SoC
- ✖ Réseau de neurones CNN
  - ✓ Réseaux de neurone et apprentissage
  - ❑ Convolutional Neural Network (CNN)
- ❑ Détails d'organisation
- ❑ CNN cible : CIFAR10

Principe généraux de l'IA & réseaux de neurones

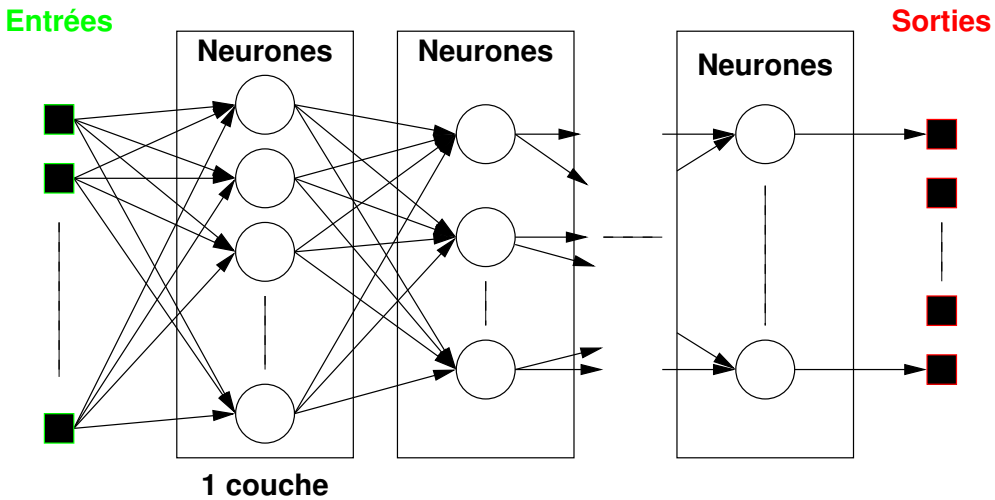
L'Intelligence Artificielle (IA) par Réseau de Neurone Artificiel (**RNA**) à appliquer sur des données un traitement lui-même paramétré par un apprentissage préalable d'après une base de données.
Les tâches d'IA par RNA sont essentiellement:

- Détection Probabilité de présence ou l'absence d'un motif
- Classification Probabilités d'appartenance à chaque classe
- Régression Estimation de la valeur d'un paramètre (taille d'un objet, . . .)
- Synthèse Production d'images réalistes, . . .

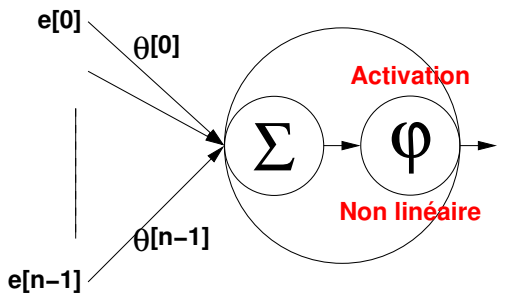
Les **Convolutional Neural Network** (CNN) sont des RNA spécifiques au traitement d'image, qui utilisent des convolutions sur des tableaux de données.

Réseaux de Neurones & CNN

Un RNA est censé reproduire des **couches de neurones**. Chaque couche est constituée de **sommes de produits** des entrées par des **coefficients**, suivis d'une fonction d'activation.



Organisation en couches



Détail d'un neurone

L'**apprentissage** consiste à calculer les coefficients θ à partir d'une base de donnée.

Un **CNN** remplace les sommes de produit par des convolutions.

Apprentissage

L'apprentissage est une tâche complexe, non traitée dans ce cours, et il n'est pas possible d'optimiser automatiquement certains paramètres du RNA:

- L'**architecture** du réseau : nombre de couches, nature des couche, tailles des couches, etc. . .
- Les **hyperparamètres** d'apprentissage: vitesse d'apprentissage, taille des lots, . . .

Pour ne pas **sur-entraîner** le réseau et le rendre trop spécifique (**généralisation**), l'apprentissage se fait en réalité sur une partition de la BDD en deux groupes de données:

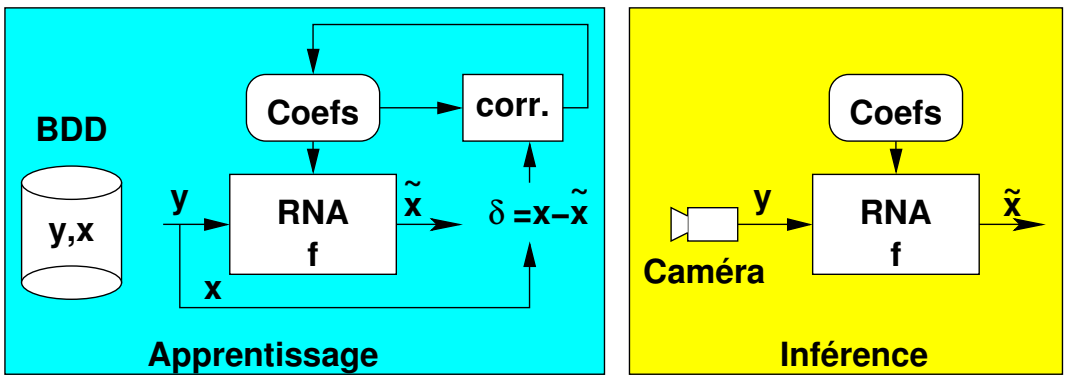
- Des **données d'apprentissage**
- Des **données de test** pour mesurer sa qualité

Fonctions génériques 1/2

Une tâche d'IA consiste à déterminer les données 'génératrices' x qui ont produit une 'observation' y (mesure).

Par ex., la donnée génératrice 'chat' génère toutes les images de chats !

Hypothèse : il existe une fonction f, inconnue à-priori, qui permet de passer d'une 'observation' y à ses données 'génératrices' $\tilde{x} = f(y)$



Le **"deep-learning"** consiste à chercher f à partir d'un ensemble d'observations préalables, la base de données d'apprentissage **étiquetée**: on connaît des données $\mathbf{y}$ et $\mathbf{x}$ associées.

Fonctions génériques 2/2

On construit f à partir de fonctions génériques paramétrables de façon systématique:

$$f(\theta, y) = f_{n-1}(\theta_{n-1}, f_{n-2}(\theta_{n-2}, \dots f_0(\theta_0, y)))$$

Les fonctions f_n sont les **“couches”** du réseau, $\theta = \{\theta_0, \dots \theta_{n-1}\}$ sont les coefficients du réseau, dont les valeurs proviennent d’un apprentissage.

Les couches d’un réseau de neurones sont typiquement:

- Fonctions **linéaires**
  - Couches ’totalement’ connectées : le perceptron
  - Convolution
- Fonctions d’**activation** non-linéaires (RELU, sigmoïd, . . .)
- **Sous-échantillonnage** (maxpool, . . .)

L'apprentisage d'un réseau de neurones

L'apprentissage consiste à déteminer θ) pour une base de données étiquetée $(x, \mathbf{y})$, de façon à minimiser l'erreur entre l'étiquette résultat de $f(\theta, y)$ sur les données étiquetées et l'étiquette attendue x.

- Un objectif : la mission du réseau de neurones
- Une base de données d'apprentissage

  Des données connues et étiquetées du résultat attendu, pour l'apprentissage des paramètres θ et le test du réseau
- Une 'architecture' de réseau

  Le nombre et la nature des couches.
- Une méthode d'apprentissage
  - Fonction de coût et régularisation
    Pour forcer les paramètres à respecter certaines caractéristiques.
  - Méthode d'optimisation
    Le plus souvent descente de gradient stochastique

Plan

- Projet d'intégration des SoC
- Réseau de neurones CNN
  - Réseaux de neurone et apprentissage
  - ✓ Convolutional Neural Network (CNN)
- Détails d'organisation
- CNN cible : CIFAR10

CNN

Les **C**onvolutional **N**eural **N**etwork, sont une technique de Deep Learning pour de nombreuses tâches de **traitement d'image**: classification, détection, identification, amélioration de qualité, flot optique. . .

Un CNN est constitué seulement des couches de

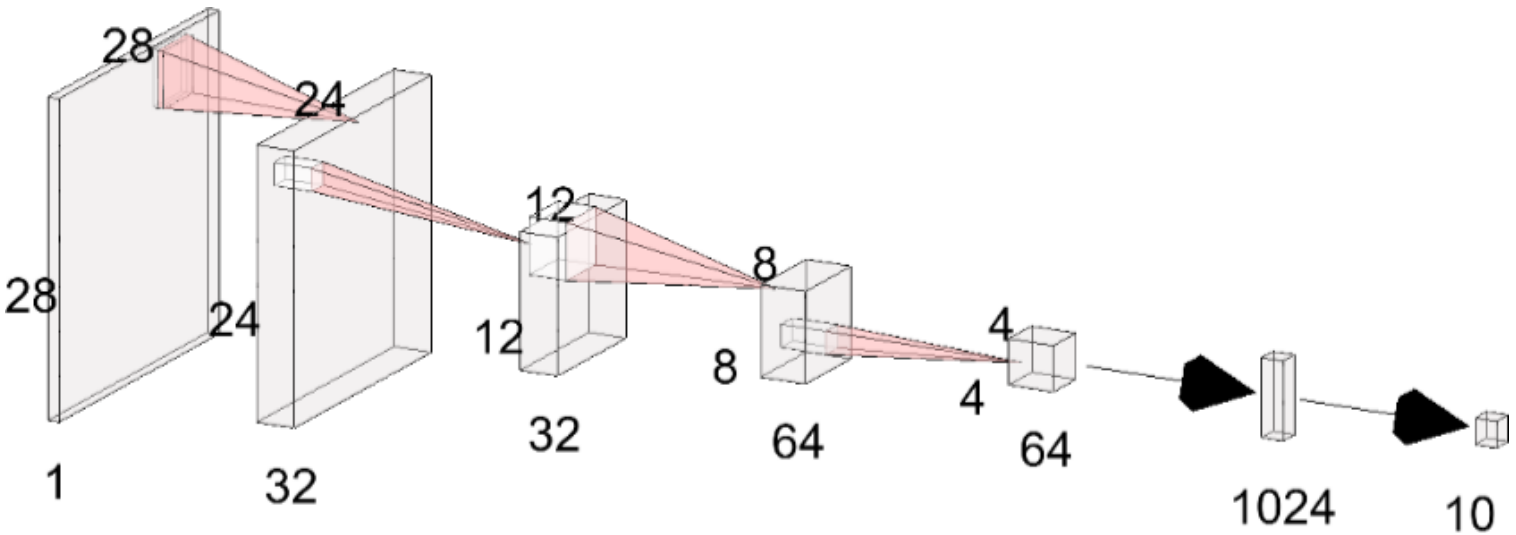
- Convolution,
- RELU
- Sous-échantillonnage par "pooling",
- Complètement connecté (perceptron)

Chaque couche prend en entrée une **feature map** et produit une feature map.

Structures de données d'un CNN

Chaque couche d'un CNN prend un entrée une **feature map** et produit une feature map.

Une fmap est un tableau 3D: $(\texttt{Canaux} * \texttt{Largeur} * \texttt{Hauteur})$



La nature des données représentées varie selon la couche:

- En entrée, l'image traitée (image RGB 3 canaux, ou monochromatique 1 canal)
- Au début, des caractéristiques 'bas niveau' (probabilité de présence d'une couleur, d'un coin, . . .)
- A la fin, plus abstrait, la probabilité de présence d'un motif complexe

Convolution

Un **kernel** K de **convolution** 2D:

$$S_{i,j} = (K \times I)(i,j) = \sum_m \sum_n I_{i+m,j+n} K_{m,n}$$

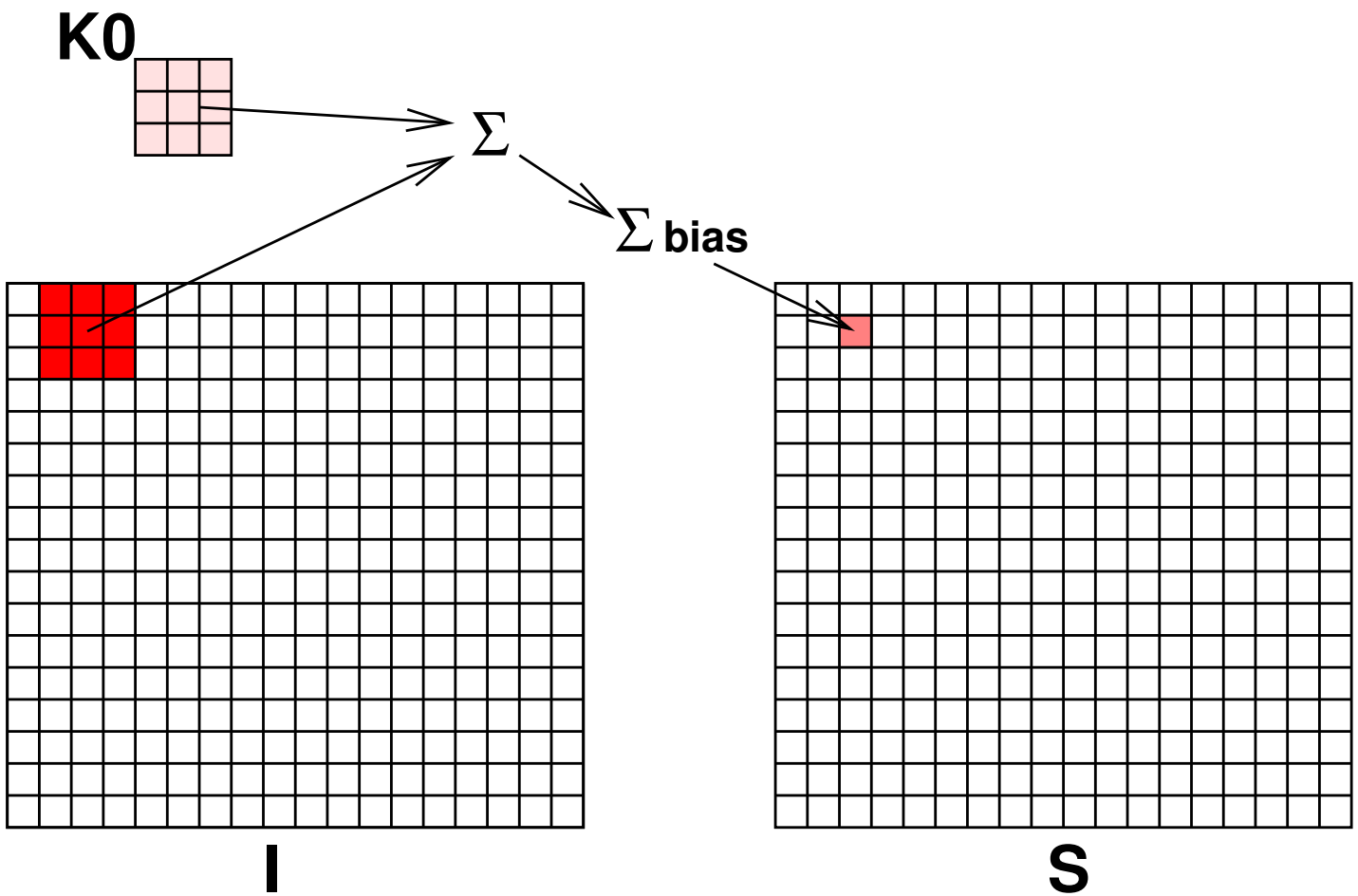
Dans un CNN, une 'feature map' est multi-canal:

$$S_{c,i,j} = \sum_{l,m,n} I_{l,i+m,j+n} K_{c,l,m,n} + \text{bias}_c$$

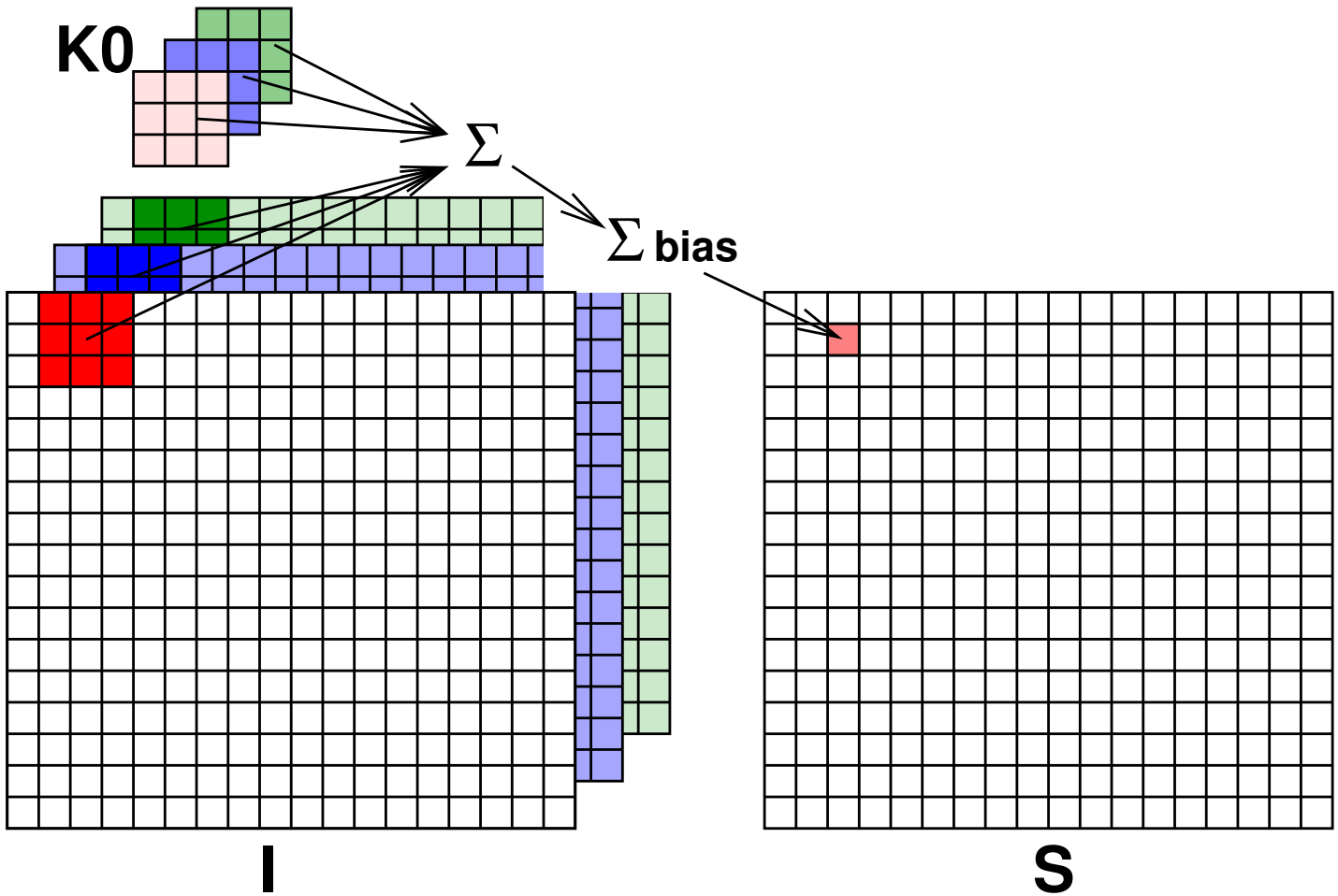
Un canal de sortie c est produit à partir de tous les canaux l de la 'feature map' d'entrée.

L'apprentissage fixe toutes les valeurs de $K_{c,l,m,n}$ pour toutes les couches.

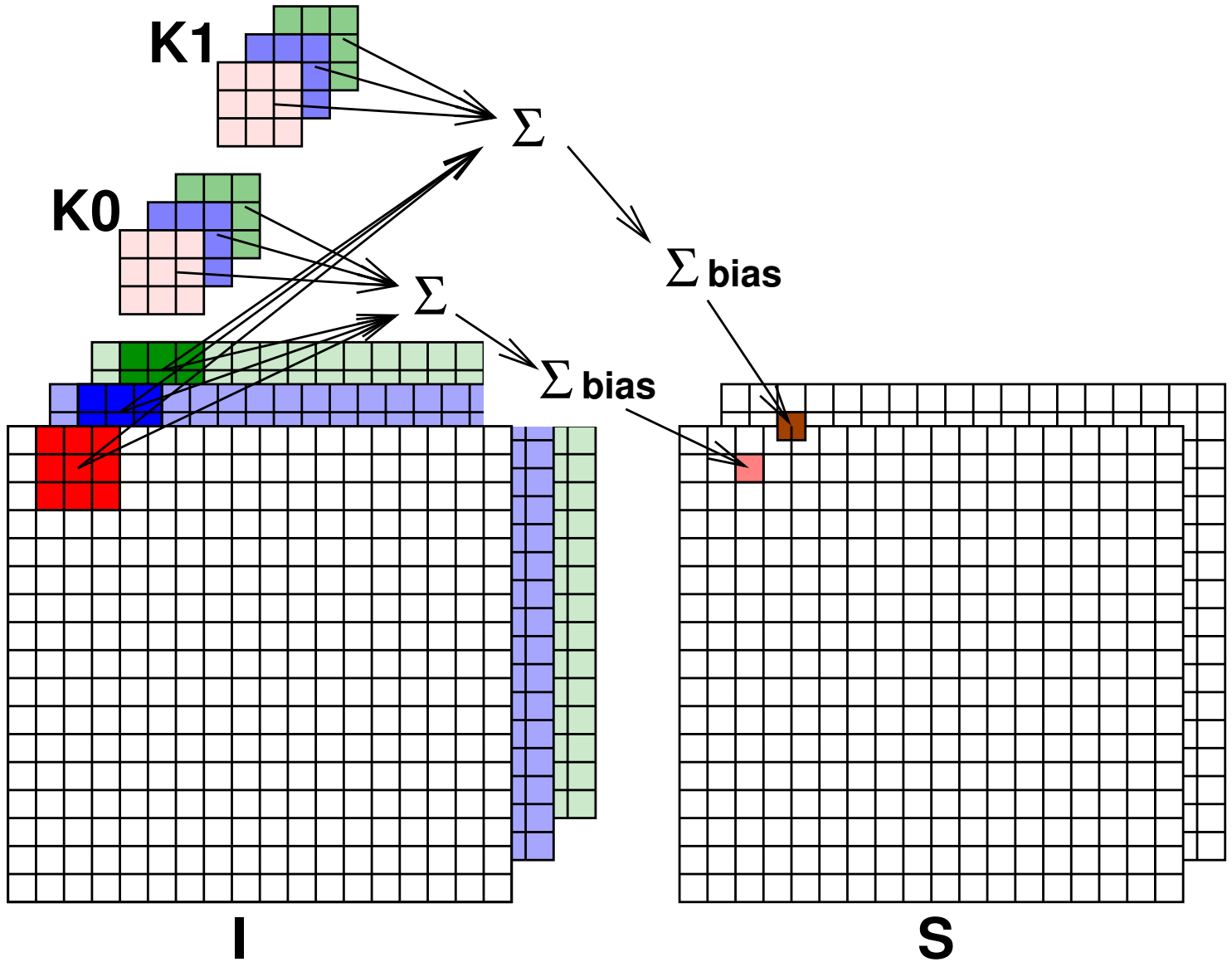
Convolution (bis)



Convolution (bis)



Convolution (bis)



RELU

RELU pour *REctified Linear Unit* est la fonction d'activation non-linéaire des CNN:

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

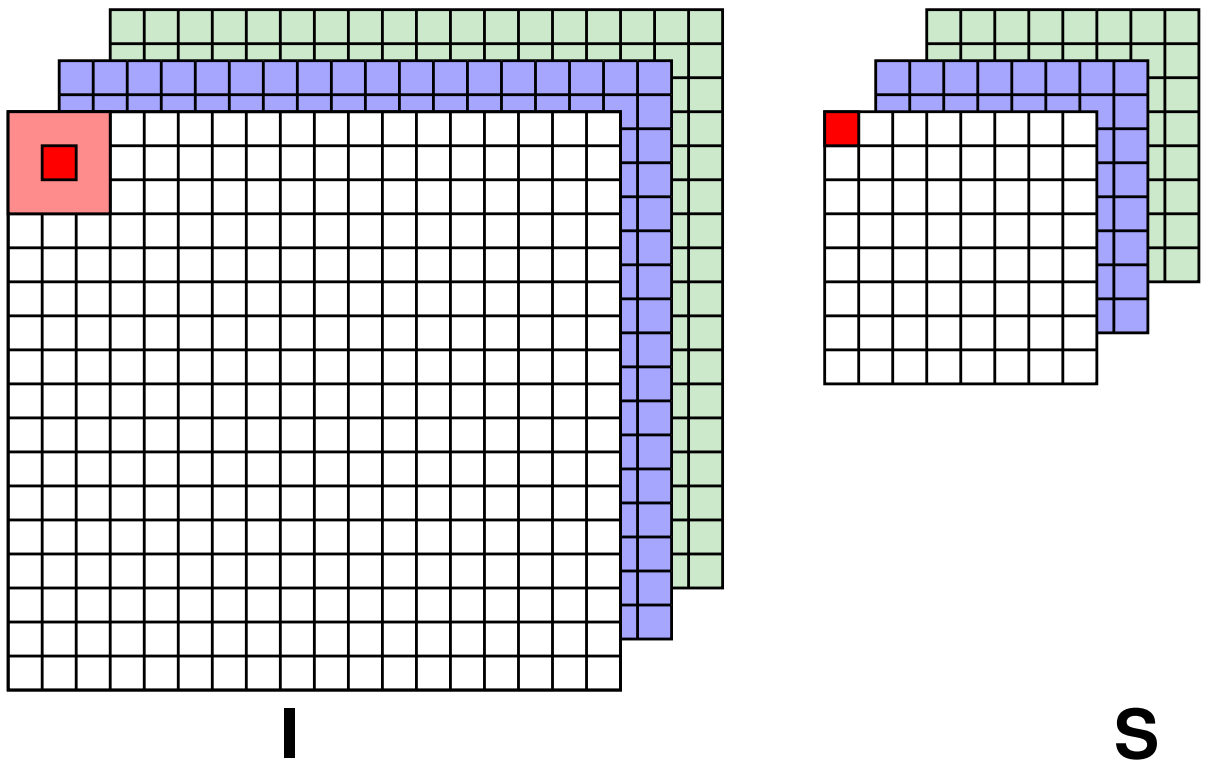
Par rapport à la sigmoïd du perceptron, les avantages de RELU sont:

- Facile à implémenter
- Dérivable (presque) partout, à dérivée non-nulle même pour les valeurs extrêmes
  - L'apprentissage est facilité

MaxPool

Maxpool est l'autre fonction non-linéaire de choix:

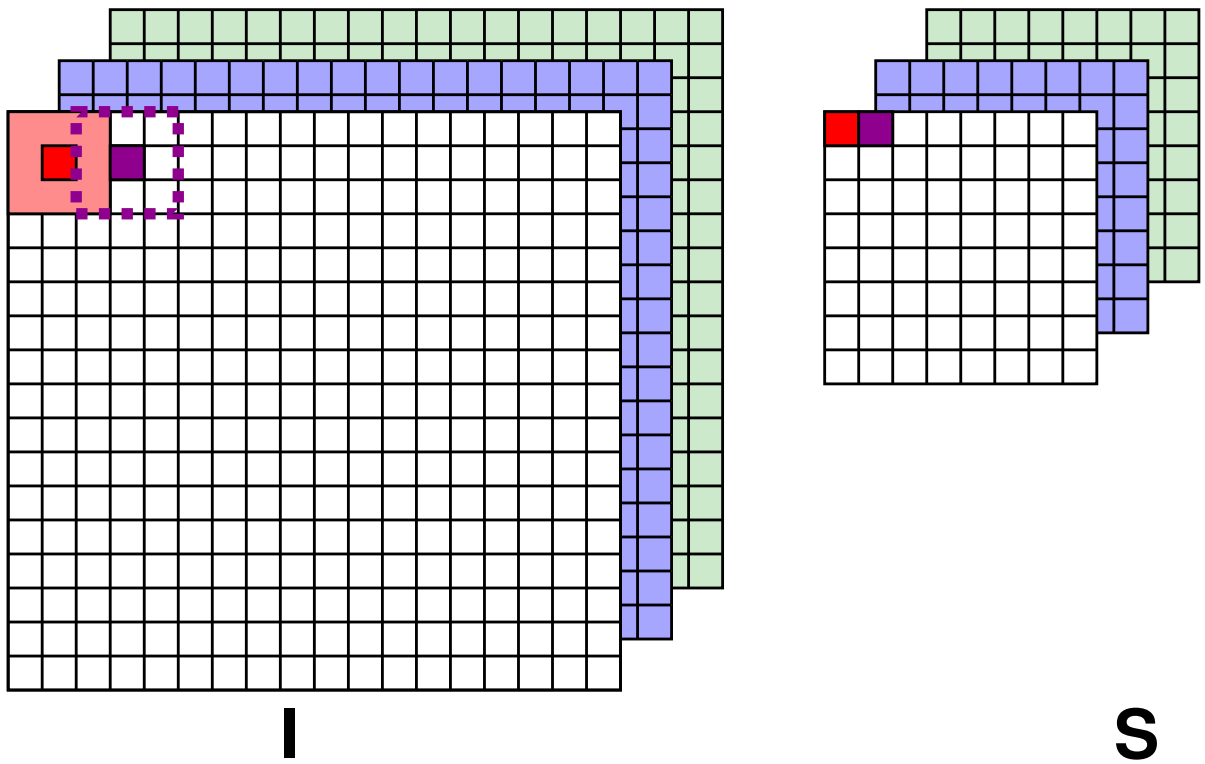
- Sous-échantillonnage pour la réduction des données
- Remplace un groupe de données par son maximum (ou médian, etc . . .)



MaxPool

Maxpool est l'autre fonction non-linéaire de choix:

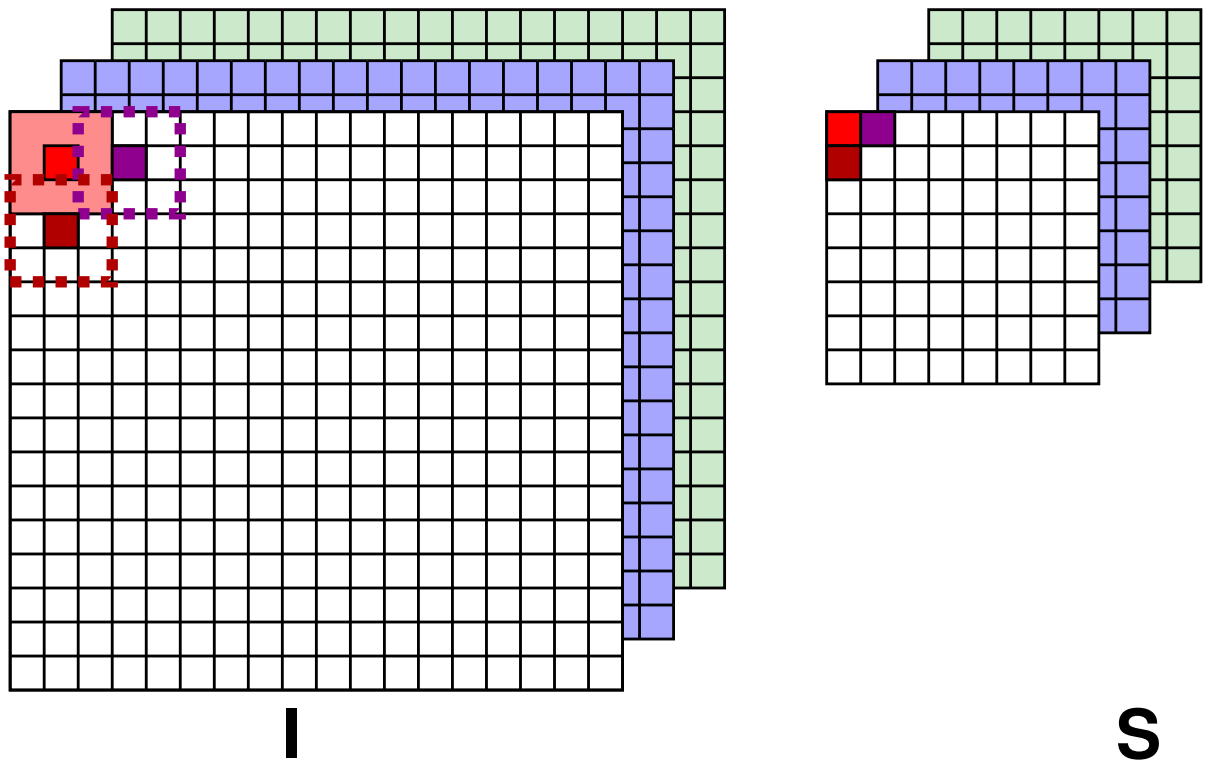
- Sous-échantillonnage pour la réduction des données
- Remplace un groupe de données par son maximum (ou médian, etc . . .)



MaxPool

Maxpool est l'autre fonction non-linéaire de choix:

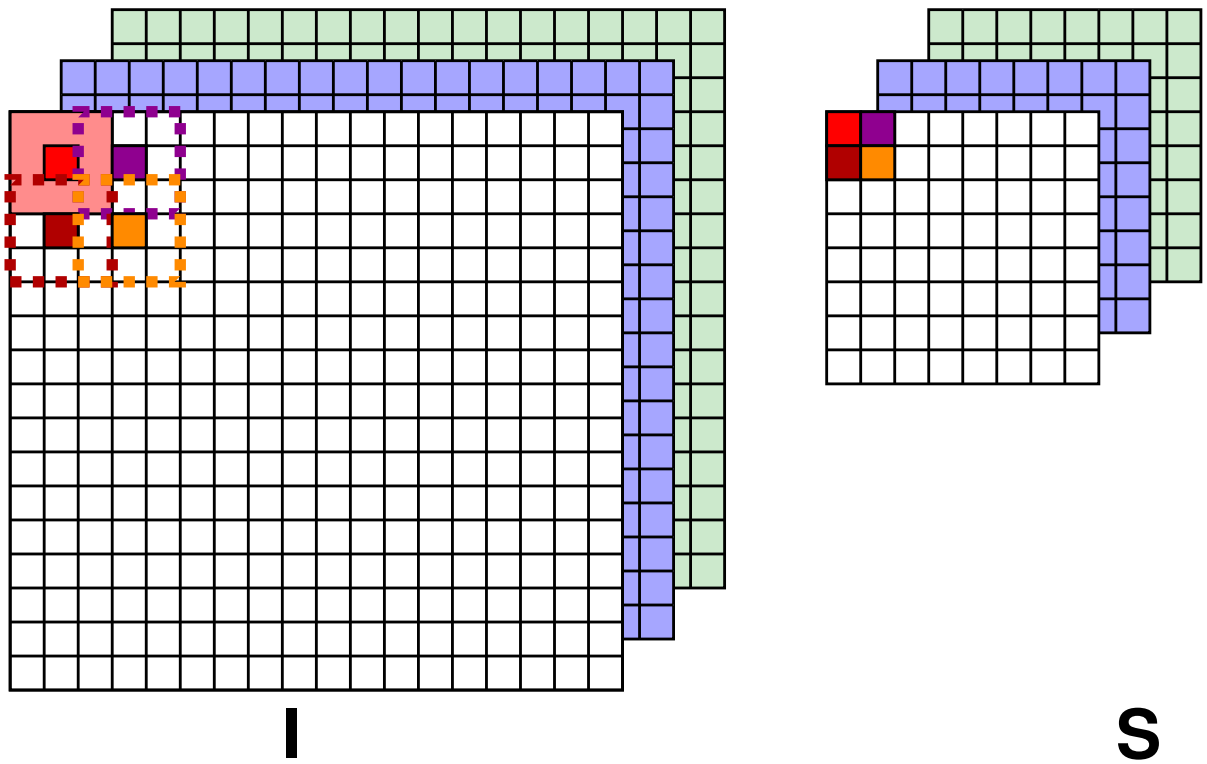
- Sous-échantillonnage pour la réduction des données
- Remplace un groupe de données par son maximum (ou médian, etc . . .)



MaxPool

Maxpool est l'autre fonction non-linéaire de choix:

- Sous-échantillonnage pour la réduction des données
- Remplace un groupe de données par son maximum (ou médian, etc . . .)



SoftMax

Softmax est utilisé pour les tâches de classification car le CNN ne produit pas de valeurs binaires telles qu'une appartenance à une classe:

$$\mathrm{softmax}(z)_i = \frac{\exp z_i}{\sum_j \exp z_j}$$

Avec z un vecteur de probabilité pour chacune des classes potentielles. $\mathrm{softmax}(z)_i$ fait 'émerger' la classe la plus probable.

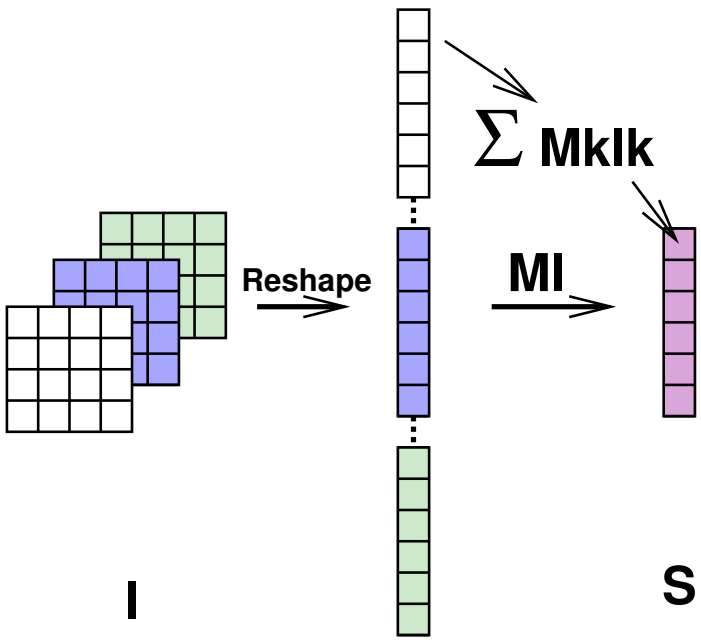
Remarque: **Softmax** provient du monde bayesien et est lié aux techniques d'optimisations statistiques basées sur le maximum de log-vraisemblance (log-likelyhood).

Complètement connectée

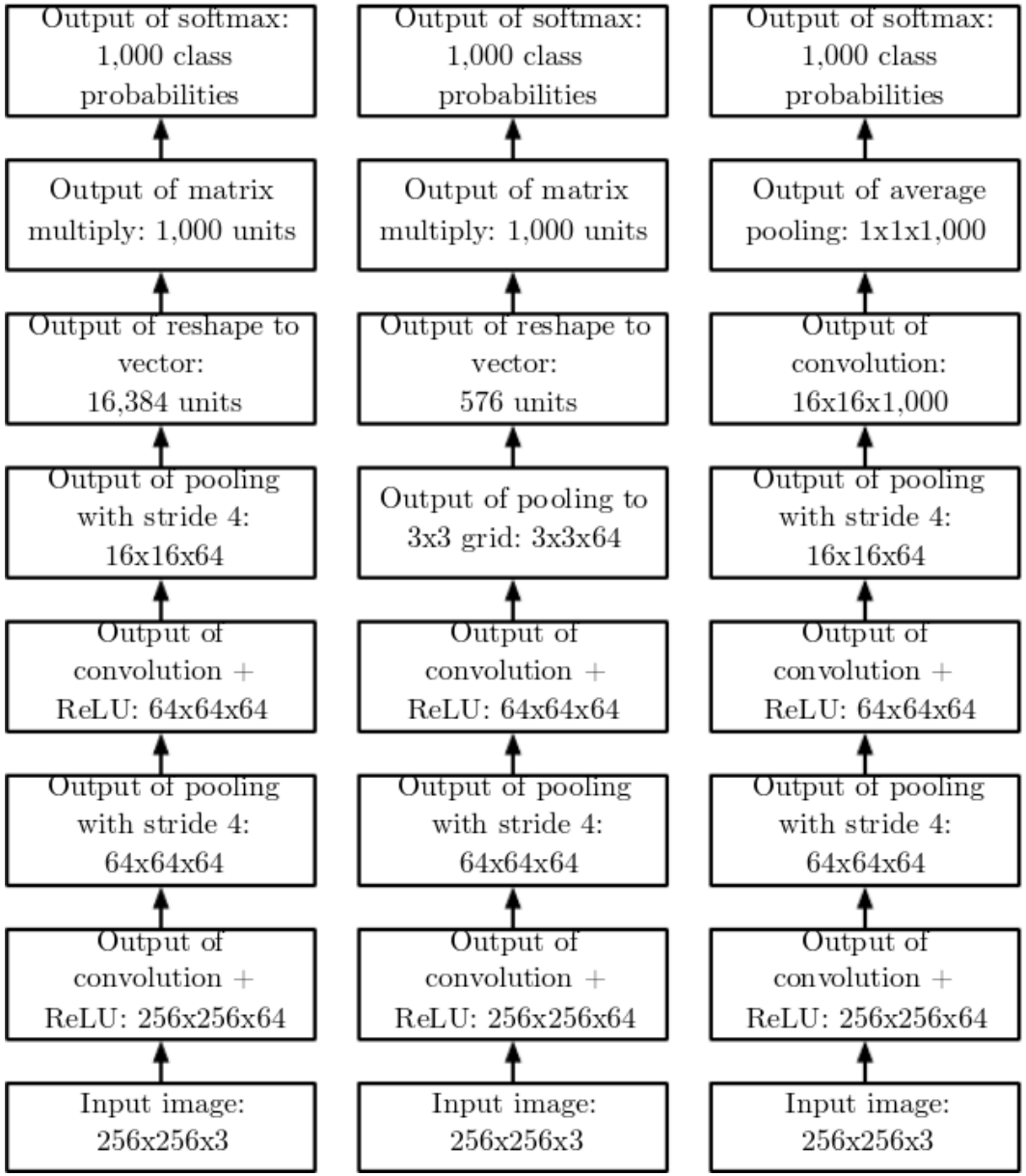
Une couches '**Fully connected**' ou **perceptron** est une combinaison linéaire de tous les élements de la couche précédente: c'est un produit matrice - vecteur $S = M.I, S_i = \sum_k M_{i,k} I_k$

La feature map d'entrée (3D) est transformée en un vecteur (reshape), le vecteur de sortie peut être transformé en feature map si besoin.

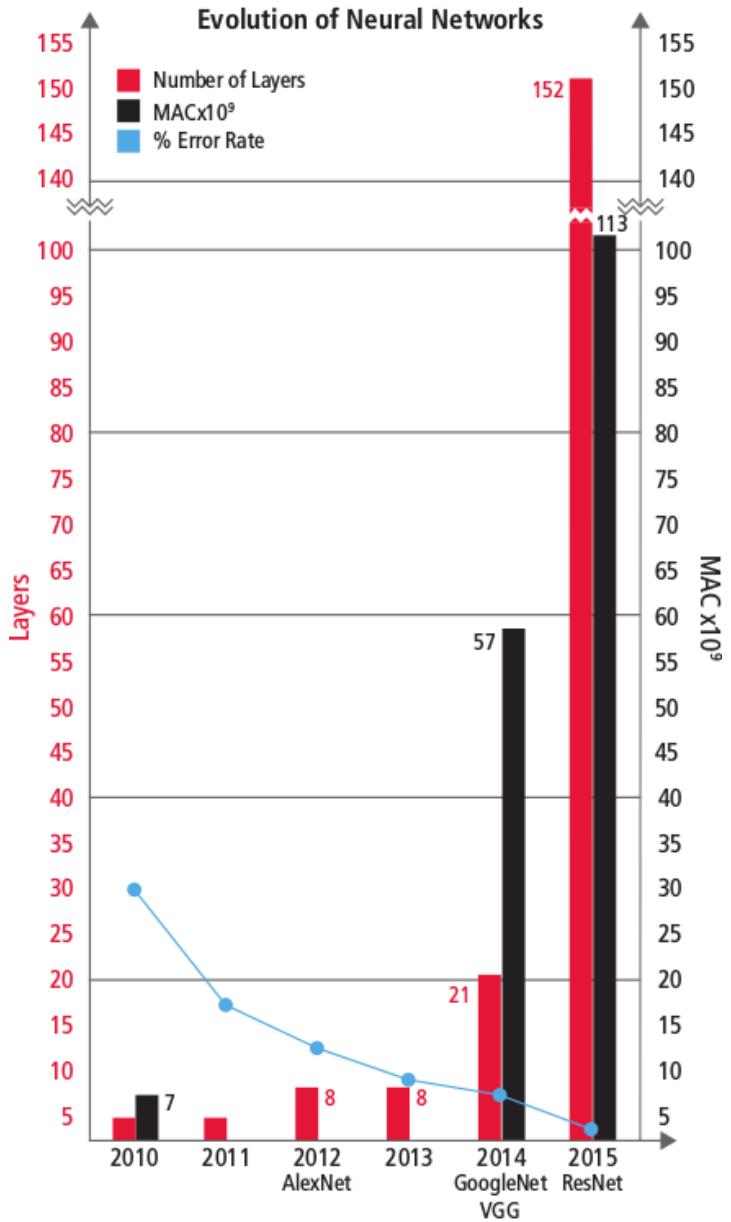
Un RNA constitué exclusivement de perceptrons est dit Multi-Layer Perceptron - MLP.



Exemples de CNN

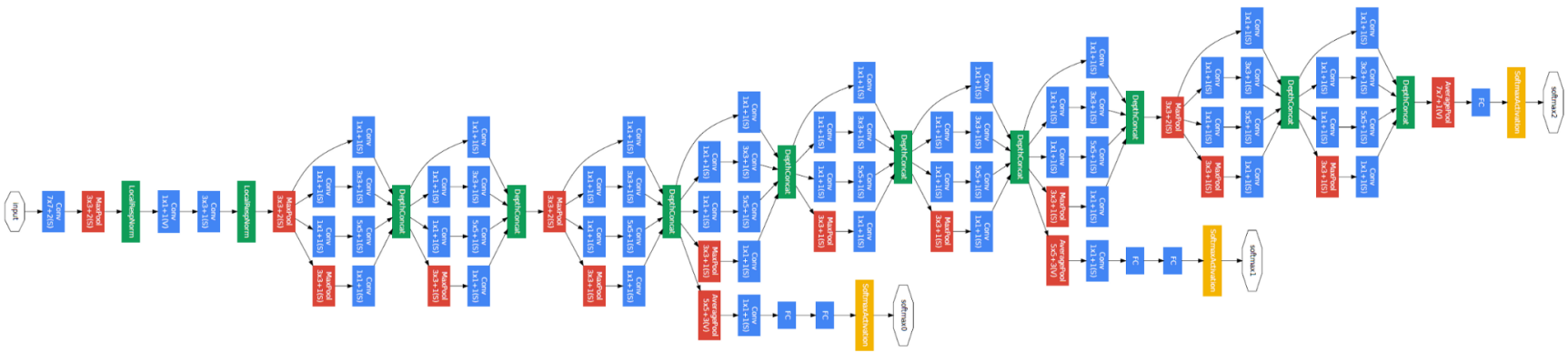


Complexité des CNN



GoogleNet - Architecture

Le réseau GoogleNet (Inception V1), fait 22 couches.
Quelques couches effectuent la **normalisation** des features map.



D'après https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7

GoogleNet - Puissance de calcul

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Plan

- Projet d'intégration des SoC
- Réseau de neurones CNN
- Détails d'organisation
- CNN cible : CIFAR10

Documentation et fichiers dans

`/tp-fmr/smancini/SEI_SoC_CNN/`

Organisation

Projet par **binôme**

- Séances : 40 heures
- Au moins 24 heures non encadrées

Le vendredi matin, jusqu'à fin Janvier.

Evaluation:

- Rapport + interview 20 mn , à la dernière séance
- Réalisation minimale pour valider le module: convolution d'image sur FPGA, flux vidéo 'live'

Phase 1: modèle fonctionnel

Lors de la première phase, 3 première séances, les objectifs sont:

- Implémenter le CNN en Python
- Lire les coefficients et effectuer une détection
- Valider le CNN sur plusieurs images de référence et calculer le taux de détection moyen

Quelques conseils:

- Commencer par des fonctions simples
- Valider votre logiciel ou matériel sur des jeux de coefficients 'fictifs', dont les valeurs vous permettent de vérifier les calcul
- Utiliser des coefficients appris
- Tester la performance moyenne du réseau de neurones

Phase 2: modèle virgule fixe et HLS

Lors de la deuxième phase, séances 3 à 5, les objectifs sont:

- Coder le CNN en C++ pour la HLS
  - Représenter les nombres en virgule fixe
  - Séparer le benchmark de l'unité de calcul
- Valider le CNN en virgule fixe et vérifier les formats des nombres (précision, débordements, . . .)
- Faire des tests de HLS et évaluer la performance/surface
- Optionnel: utilisez un code intermédiaire C++ en virgule flottante

Quelques conseils:

- Vérifier votre compréhension de la virgule fixe sur des cas d'école
- Commencer à tester la HLS sur des fonctions simples, pour comprendre ce qu'il se passe

Phase 3: HLS et intégration

Lors de la trosième phase, séances 6 à 9, les objectifs sont:

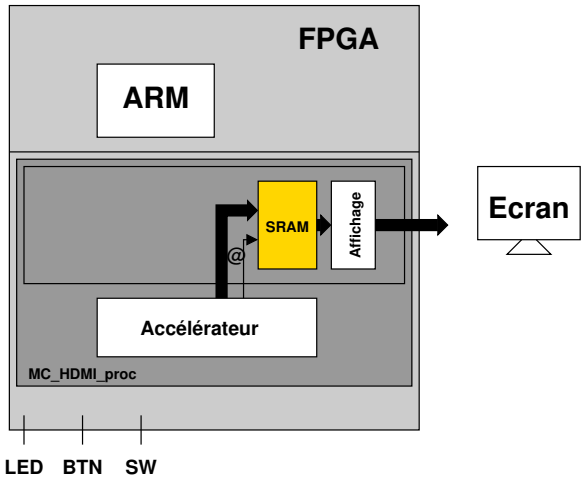
- Effectuer la HLS du CNN complet et valider le résultat
- Evaluer la surface et temps de calcul
- Intégrer le réseau de neurone sur la carte Zybo Z7
- Optimisez le HW produit
  - En utilisant les options de HLS (pipeline, déroulement de boucles)
  - Eventuellement en modifiant le code C++ pour spécifier une architecture plus rapide

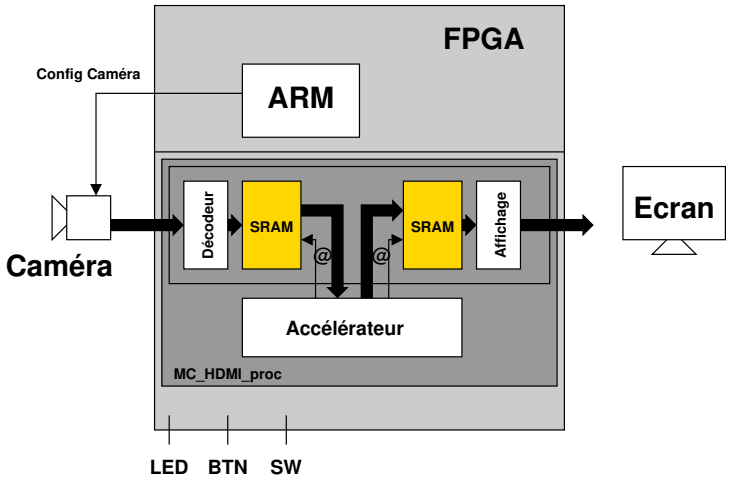
Quelques conseils:

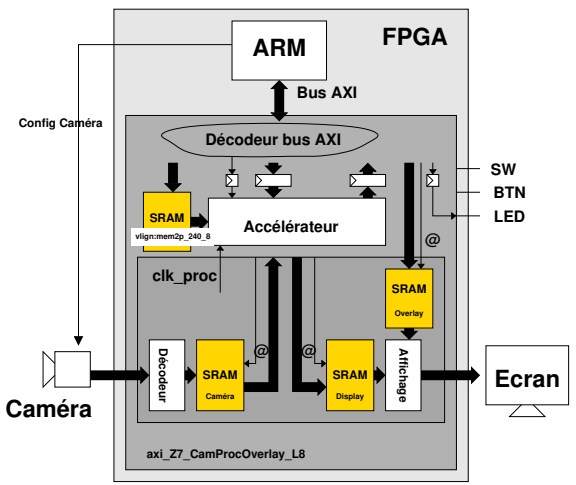
- Identifier les métriques produites par chaque outil
- Associer ces métriques aux paramètres de conception accessibles

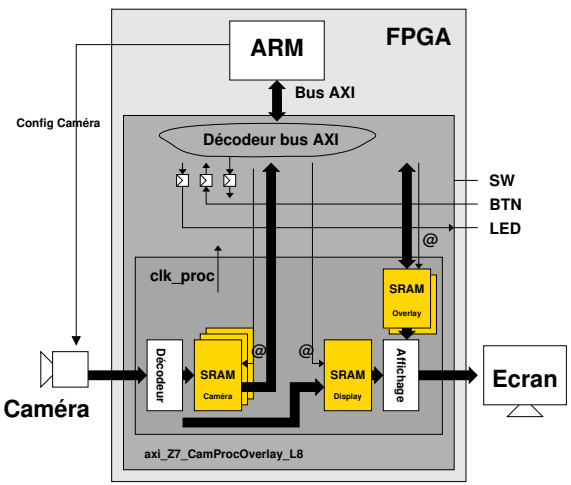
Plateformes de prototypage

Des canevas de plateforme de prototypage sont fournis.









Evaluation

La dernière séance est consacrée à l'évaluation.
Chaque binôme est interviewé en séance pendant 20 minutes:

- Présentation du travail fait
- Questions/réponses
- Démonstration sur carte FPGA du résultat

Plan

- Projet d'intégration des SoC
- Réseau de neurones CNN
- Détails d'organisation
- CNN cible : CIFAR10

CIFAR 10

CIFAR 10 est un réseau de neurone CNN de démonstration. Il est appris pour la base de données CIFAR10: des images de taille 32×32 appartenant à 10 classes.

Quelques sites d'intérêt :

- CIFAR 10
- Tensor Flow et tutoriel CIFAR10
  Tensor Flox est un environnement pour l'apprentissage et l'inférence.

- Le réseau de base de Tensor Flow est modifié pour ce projet!
- Le fichiers nécessaires sont tous sur le répertoire du projet.

CIFAR 10 database

Le format de la base de donnée du site CIFAR 10

Binary version

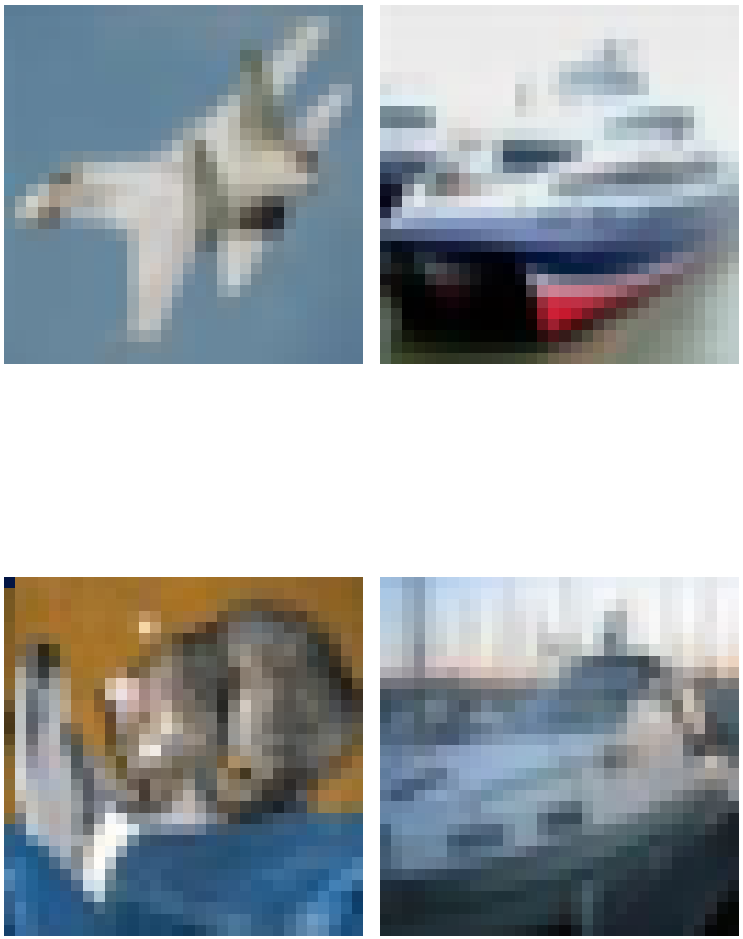
The binary version contains the files `data_batch_1.bin, data_batch_2.bin, ...` as well as `test_batch.bin`. Each of these files is formatted as follows:

```
<1 x label><3072 x pixel>
...
<1 x label><3072 x pixel>
```

In other words, the first byte is the label of the first image, which is a number in the range 0-9. The next 3072 bytes are the values of the pixels of the image. The first 1024 bytes are the red channel values, the next 1024 the green, and the final 1024 the blue. The values are stored in row-major order, so the first 32 bytes are the red channel values of the first row of the image.

Each file contains 10000 such 3073-byte "rows" of images, although there is nothing delimiting the rows. Therefore each file should be exactly 30730000 bytes long.

CIFAR 10 database



CIFAR 10 : utilitaires

Fichiers dans

`/tp-fmr/smancini/SEI_SoC_CNN/cifar10_data`

Exemple de script pour lire une image:

```
nb=n*3073+1
cat cifar10_data/cifar-10-batches-bin/test_batch.bin |
        dd count=3072 bs=1 skip=<n> > toto.raw
display -size 32x32 -depth 8 -interlace Plane  rgb:toto.raw
convert -size 32x32 -depth 8 -interlace Plane  rgb:toto.raw cifar10_voilier.png
convert -size 32x32 -depth 8 -interlace Plane  rgb:toto.raw cifar10_voilier_bin.ppm
convert -size 32x32 -depth 8 -interlace Plane  rgb:toto.raw -compress none cifar10_voilier
```

Adaptation des images

Les images 32x32 sont 'coupées' à 24x24, la coupe étant centrée.

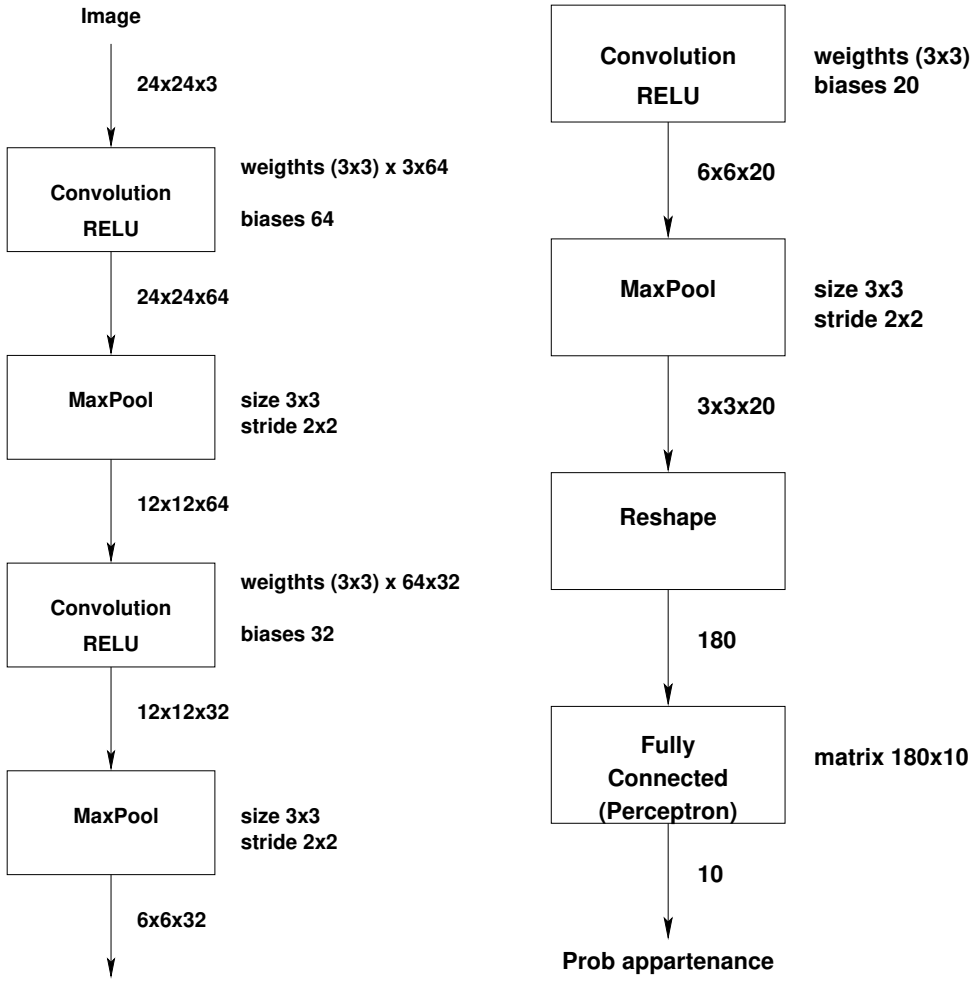
Attention: pour la détection, une image m doit être normalisée:

$$\mu = \frac{1}{N}\sum m_{i,j} \quad ; \quad \sigma = \sqrt{\frac{1}{N}\sum (m_{i,j} - \mu)^2}$$

$$m'_{i,j} = (m_{i,j} - \mu) / \max(\sigma, \frac{1}{\sqrt{N}})$$

- Le normalisation est appliquée sur toutes les composantes de l'image car une normalisation par canaux séparés modifierait la couleur des objets.

CNN 'basique' pour CIFAR10



reshape : transforme la matrice en vecteur, dans l'ordre cannonique

Format des données : **HWC** HeigthxWidthxCanal , les 'canaux' sont en 'premier'.

CNN 'basique' pour CIFAR10

Les coefficients sont dans

`CNN_coeff_3x3.txt`

et

`CNN_coeff_5x5.txt`

SEI - SoC - CNN

S. Mancini

Plan Détaillé

CNN cible : CIFAR10

- CIFAR 10
- CIFAR 10 database
- CIFAR 10 database
- CIFAR 10 : utilitaires
- Adaptation des images
- CNN 'basique' pour CIFAR10
- CNN 'basique' pour CIFAR10