

Pont Arthur
Lemaire Louis
Lambert Rémi
Tristan Rivet
Gaël Ousset
Guillaume Dalle

3A SEI SoC
Décembre 2023
Janvier 2024

SYSTÈMES EMBARQUÉS NUMÉRIQUES - 5PMESEN7

Rapport de Projet



SOMMAIRE :

Introduction Générale	3
Présentation globale du Projet	3
Mise en Contexte	3
Présentation Structurelle du projet	4
Les différentes Phases du projet	4
La Politique de travail	4
Méthode de Tests	5
Implémentation des Solutions	5
A. Reading images from the SD card*	5
B. Image pre-processing	6
C. Image display ("bypass" filter)	7
D. Development of the Sobel filter	10
E. Development of the convolutional neural network (CNN) CIFAR-10	11
Développement Technique	11
Test du CNN	12
Comparatif avec le projet Architecture	13
Pistes d'amélioration du CNN	13
F. Interruptions via push buttons	14
Résultats et pistes d'amélioration	15
Les résultats sur carte	15
Les perspectives d'amélioration	17
Compétences Acquises	17
Annexes	18

Introduction Générale

Cette première section vise à contextualiser le projet, mais également à justifier son existence; en effet, outre le côté obligatoire de la matière pour valider un diplôme d'ingénieur, il est intéressant de comprendre en quoi travailler sur ce projet nous est utile et nous le sera également.

Ce projet a été mis en place du 7 Novembre 2023 au 25 Janvier 2024 et a été encadré par Mounir Benabdenbi. Il a été commencé d'une archive très complète, avec des zones à trous devant être complétées par les étudiants.

Présentation globale du Projet

Ce projet intègre une approche matériel-logiciel (HW/SW) pour développer le logiciel permettant de configurer le matériel d'un FPGA, afin d'afficher différentes images, afin d'y appliquer des filtres ou un réseau de neurones.

Avant de revenir sur chaque notion précisément, nous pouvons dire de manière simplifiée que ce projet consiste à mettre à disposition une Intelligence Artificielle permettant d'identifier la classe d'un objet ou d'un animal reconnu sur une photo, ou de présenter les images de manière simple ou filtrée.

Concrètement, voici ce qu'il se passe quand on lance le programme une fois celui-ci chargé sur le FPGA. Tout d'abord, le programme va lire toutes les images ppm et va les stocker en mémoire. Puis, notre programme va convertir toutes les images en échelle de gris. Ensuite, il va appliquer le filtre sobel sur toutes les images. On autorise ensuite les interruptions avant d'afficher la première image sans filtre, en mode "bypass". Enfin, notre programme va tourner indéfiniment en changeant de filtre appliqué ou d'image sélectionnée à chaque pression de bouton (ceci étant géré via des interruptions). On retrouve la sortie terminale du FPGA (récupérée via picocom) illustrant le début du fonctionnement du programme.

On pourra donc imaginer que ce système, développé en software, permettra d'être facilement portable : le code tient sur une carte SD et on n'a besoin simplement d'un FPGA pour le faire fonctionner.

Mise en Contexte

Le projet utilise ces différents éléments : il se base sur un micro-processeur RISC-V, très utilisé aujourd'hui avec architecture de jeu d'instructions ouverte et basée sur un ensemble d'instructions réduit. Le système sera implémenté sur FPGA Nexys A7 de Digilent. On implémentera un filtre Bypass, qui vise simplement à afficher l'image sans traitement. On proposera également l'implémentation d'un filtre Sobel, qui est un traitement d'image utilisé pour détecter les contours dans une image. Enfin, le CNN (Convolutional Neural Network) mis à disposition sera un CIFAR-10 : ce dataset est une collection d'images très utilisées dans le cadre de la création de CNN. Elle comporte assez peu de classes différentes, et

possède une grande variété d'images recensées et prêtes à être utilisées. Les différentes classes sont les suivantes : avion, voiture, oiseau, chat, cerf, chien, grenouille, cheval, bateau et camion. Nous prendrons des images ppm en entrée.

Présentation Structurelle du projet

Travailler sur un sujet de TP ou travailler sur un projet ambitieux plus long terme n'appellent pas la même structure et les mêmes ressources. Il est nécessaire pour travailler efficacement d'avoir une vraie méthodologie du travail et un découpage des tâches : cette section vise à présenter notre vision structurelle du projet.

Les différentes Phases du projet

Pour la réalisation de ce projet, nous avons dû réaliser plusieurs échéances. Tout d'abord, 6 étapes ont été effectuées en parallèle, toutes nécessaires pour implémenter le projet sur le FPGA :

- A. Reading images from the SD card
- B. Image pre-processing
- C. Image display ("bypass" filter)
- D. Development of the Sobel filter
- E. Development of the convolutional neural network (CNN) CIFAR-10
- F. Interruptions via push buttons

De plus, afin de pouvoir valider l'implémentation, plusieurs tests ont été effectués sur notre implémentation. Suite à la validation de l'implémentation, un compte-rendu fut réalisé dans le but de ne pas seulement rendre un projet fonctionnel mais aussi transmissible, lisible et compréhensible. Ce rapport a été coécrit par l'intégralité des membres du groupe; à noter que ce rapport ne remplace en aucun cas le README, totalement indépendant et essentiel pour faire fonctionner le projet par une personne tierce au projet. Une présentation orale accompagnée de diapositives fournies dans l'archive a également été donnée.

La Politique de travail

Afin de pouvoir avancer en groupe, nous avons fait le choix d'utiliser un Git. Cet outil a été utilisé afin de pouvoir partager nos codes, être certains de ne pas avoir de régressions, mais également de travailler en branches propres et suivies. Afin de réaliser régulièrement des points entre nous, nous nous sommes fixés des réunions en présentielle, généralement à la fin des cours. La plupart des codeurs ont utilisé l'outil Microsoft VSCode, permettant un travail simplifié dans un environnement propre. Nos PC personnels étaient sous Windows et/ou Ubuntu, l'un permettant seulement de coder, l'autre permettant également de compiler les codes à distance, sans forcément se rendre au CIME.

Enfin, nous avons utilisé l'outil Notion, très utile dans la gestion de projet, afin de pouvoir avoir un suivi des rôles affectés et des deadlines de chacun par exemple.



To Do List

Table + Filter Sort ⚡ Search ↻ ... New ▾

Nature des Rendus		...	Check	+ ...	
Aa	Name	Tags	Check	+ ...	
	Rapport	DOING	<input type="checkbox"/>		
	Diapo	DONE	<input type="checkbox"/>		
	Archive de Code	DONE	<input checked="" type="checkbox"/>		

Exemple de page du logiciel Notion

Méthode de Tests

Comme toutes les fonctions étaient regroupées dans un même fichier *applications.c*, nous avons commencé par compléter tous les trous avec des solutions simples mais fausses. Par exemple, pour une fonction de redimensionnement, il suffit de remplir la matrice de sortie avec des 0. Cela permet de pouvoir compiler et ainsi de tester chacun son morceau de code séparément des autres.

Ensuite, une fois que chacun a complété sa partie et qu'elle fonctionne seule de son côté, nous les regroupons et vérifions la compilation. Enfin nous exécutons le flot d'implémentation pour le FPGA et vérifions que le boot se fait bien grâce à picocom puis nous essayons d'afficher les images sur l'écran.

Implémentation des Solutions

Cette section du rapport vise à détailler techniquement les solutions mises en œuvre, partie par partie, afin de mettre en place un système total fonctionnel.

A. Reading images from the SD card*

Il s'agit de la première étape. Celle-ci est destinée à lire des images de format PPM(Portable PixMap) à partir d'une carte SD, à en extraire les valeurs des pixels, et à les stocker dans un tableau global. La fonction `read_pic(int n_image, int *tab_size, int *tab_width, int *tab_length, uint8_t *global_tab)` joue ce rôle.

La lecture du fichier est réalisé avec `f_open` tel que :

```
fr = f_open( &fil , file_name , FA_READ
```

Après avoir ouvert le fichier, le code lit les deux premiers caractères de l'en-tête pour vérifier s'ils correspondent à 'P3', ce qui indiquerait que le fichier est au format PPM P3. Cette vérification est importante car le traitement des données d'image dépend du format spécifique du fichier. Il faut utiliser la fonction `f_read` afin de pouvoir lire les premiers caractères et on effectue une comparaison sur ces caractères.

Le code saute ensuite le commentaire et lit les dimensions de l'image (longueur et largeur). Puis, il parcourt le reste du fichier ligne par ligne, en extrayant les valeurs des pixels. Chaque valeur de pixel est convertie de chaîne de caractères en entier à l'aide de la fonction `My_atoi` et stockée dans le tableau pixels.

L'extraction se réalise par l'intermédiaire d'un tableau `pixels`. Après avoir extrait toutes les valeurs des pixels de l'image, celles-ci sont copiées dans un tableau global (`global_tab`). Ce tableau est utilisé pour stocker les données de plusieurs images, d'où l'indexation basée sur `n_image`.

Une fois que toutes les données nécessaires ont été lues et stockées, le fichier est fermé à l'aide de la fonction `f_close` et l'espace alloué pour les tableaux est libéré. Cette étape est cruciale pour libérer les ressources et éviter les fuites de mémoire ou les conflits de fichiers.

B. Image pre-processing

L'étape de pre-processing a deux buts principaux. Le premier porte sur l'affichage de l'image sur l'écran qui va se faire en nuances de gris, il faut donc passer d'une image RGB à une image en nuances de gris. Deuxièmement, il permet d'adapter le format de l'image aux prérequis du réseau de neurones.

La conversion en niveaux de gris est faite assez facilement, il suffit d'appliquer la formule de luminance UIT-R BT 601 de l'œil humain à chaque pixel RGB afin de retrouver son niveau de gris : $Y = 0,3R + 0,57G + 0,11B$, avec Y le signal de luminance, R le niveau de rouge, G le niveau de vert et B le niveau de bleu. Cette conversion va permettre d'afficher l'image sur l'écran LCD afin de pouvoir comparer les résultats du réseau avec ce que l'on voit.

Ensuite, comme les images sont de base en 640x480 pixels, il a fallu les passer en 24x24 pixels pour être acceptées par le réseau de neurones. Pour cela, la fonction va parcourir chaque pixel de l'image cible (redimensionnée) et y placer la moyenne des pixels contenus dans une fenêtre 10x10 de l'image source, et ce pour chaque couleur. Ainsi, on obtient une image 24x24 centrée à partir d'une image 640x480 pixels.

Puis, on préfère utiliser des tenseurs plutôt que des images pour le réseau de neurones, il faut donc transformer les images (R0G0B0 R1G1B1 ...) en tenseurs (R0R1... G0G1... B0B1...), pour cela on parcourt juste l'image source avec la boucle sur les canaux à l'extérieur et en remplissant le tenseur au fur et à mesure grâce à un index.

Enfin, le CNN a besoin d'images normalisées pour s'exécuter correctement, il faut donc appliquer une normalisation. Dans ce projet on choisit la normalisation suivante :

$$m'_{ij} = \frac{m_{ij} - \mu}{\max(\sigma - \frac{1}{\sqrt{N}})}$$

Cependant, comme on code pour un système embarqué, on préfère utiliser une implémentation en Assembleur de la racine carrée plutôt que celles disponibles dans les bibliothèques C standards. Cette implémentation utilise l'instruction *fsqrt.s* et en voici le code exact :

```
float __ieee754_sqrtf(float x)
{
    asm("fsqrt.s %0, %1"
        : "=f"(x)
        : "f"(x));
    return x;
}
```

Finalement, après toutes ces étapes, nous obtenons un tenseur lisible par le réseau de neurones convolutifs qui va pouvoir en déceler ses mystères (catégorie du CIFAR-10).

C. Image display ("bypass" filter)

Cette partie est à la fois la plus simple à comprendre, et à la fois une des plus essentielles car elle sert en effet pour le “bypass” filter, mais elle sert aussi indirectement pour toutes les autres fonctions.

Cette section a plusieurs entrées ou composantes, mais celles que nous utiliserons pour expliquer le fonctionnement seront les suivantes (on ne détaillera pas ici chaque entrée) :

- le pointeur image “classique” *ptr_selected_img*
- le pointeur image “sobel” *ptr_selected_img_filtered*
- le filtre choisi *filter_nb*
- la classe de l'image calculée par le cnn *result*
- la constante *OVERLAY_SIZE*, valant 176*80, qui est équivalente à la taille d'une étiquette dans le fichier *overlay.c* (Annexe 2)

La décomposition du fichier *overlay.c* a été faite avec un rapide code python (*Projet_SE/lowrisc-chip-DATE2020-DEMO/pontar_python/compt_overlay.py*) servant à déterminer la constante *OVERLAY_SIZE* (Annexe 3).

En fonction de la classe et du filtre, on utilisera la fonction *on_screen()* de manière différente.

Dans le cas BYPASS, on affecte le pointeur référant à l'image “simple”.

```

case BYPASS:
| on_screen(filter_nb, 0, ptr_selected_img);

```

On calculera l'emplacement du début de l'étiquette "bypass" afin de pouvoir l'afficher (10*OVERLAY_SIZE car "bypass" est la 11ème étiquette).

```

if (mode == BYPASS) //Sélection de l'étiquette en fonction du filtre choisi
{
    printf("\nPainting BYPASS overlay.\n");
    //L'image à l'indice 10 correspond à l'overlay du bypass
    ptr_labels_overlay = ptr_labels_overlay + 10*OVERLAY_SIZE ; // on decale pour sauter les etiquettes des classes du CNN
}

```

Dans le cas du SOBEL, on fera exactement la même manipulation en utilisant le pointeur de l'image filtrée, c'est-à-dire le *ptr_selected_img_filtered*.

```

case EDGE_DETECTOR:
| on_screen(filter_nb, 0, ptr_selected_img_filtered);

```

On prendra ensuite la 12ème étiquette, contenant le texte "filter1".

```

else if (mode == EDGE_DETECTOR)
{
    printf("\nPainting the FILTER overlay\n");
    //L'image à l'indice 11 correspond à l'overlay du edge detector
    ptr_labels_overlay = ptr_labels_overlay + 11*OVERLAY_SIZE ; //apres les etiquettes des classes
}

```

Pour le cas "CNN", on lance l'exécution du réseau de neurones afin de récupérer le résultat de ce dernier (la classe de l'image).

Le pointeur *image* est celui de l'image "simple" car on effectue ici aucun traitement sur l'image affichée.

```

case CNN_CLASSIFIER:
// In this case we visualize the image, while computing ...
display_ptr = (uint64_t *) (TAB_GS[img_in_number - 1]);
for (y = 0; y < 480; ++y)
{
    for (x = 0; x < 640 / 8; ++x)
    {
        hid_new_vga_ptr[x + y * 640 / 8] = *display_ptr;
        display_ptr++;
    }
}
// Launch the CNN
int result = perform_cnn(img_in_number) ;
// When finished, show the LABEL as an overlay.
on_screen(filter_nb, result, ptr_selected_img);
break;
}
}

```

On calcule ensuite l'emplacement du début de l'étiquette avec un calcul très simple car les labels sont rangés dans l'ordre (emplacement 0 pour la classe 0). On obtient donc l'étiquette correspondante au traitement précédent.

```

else if (mode == EDGE_DETECTOR)
{
    printf("\nPainting the FILTER overlay\n");
    //L'image à l'indice 11 correspond à l'overlay du edge detector
    ptr_labels_overlay = ptr_labels_overlay + 11*OVERLAY_SIZE ; //apres les etiquettes des classes
}

```

Enfin, pour tous les filtres, l'étape suivante est donc d'afficher, selon la zone (176 x 80 en haut à gauche ou non), soit l'étiquette adéquate récupérée, soit l'image, avec ou sans traitement.

```

for (y = 0; y < 480; ++y)           //Affichage de l'image
{
    for (x = 0; x < 640 / 8; ++x)
    {
        if (x < OVERLAY_X && y < OVERLAY_Y )
        { //on verifie si on est dans la zone de l'etiquette
            hid_new_vga_ptr[x + y * 640 / 8] = (*ptr_labels_overlay);
            ptr_labels_overlay++;
        }
        else
        {
            hid_new_vga_ptr[x + y * 640 / 8] = (*ptr_image);
        }
        ptr_image++;
    }
}

```

On peut résumer ce fonctionnement à l'aide d'un schéma assez simple.

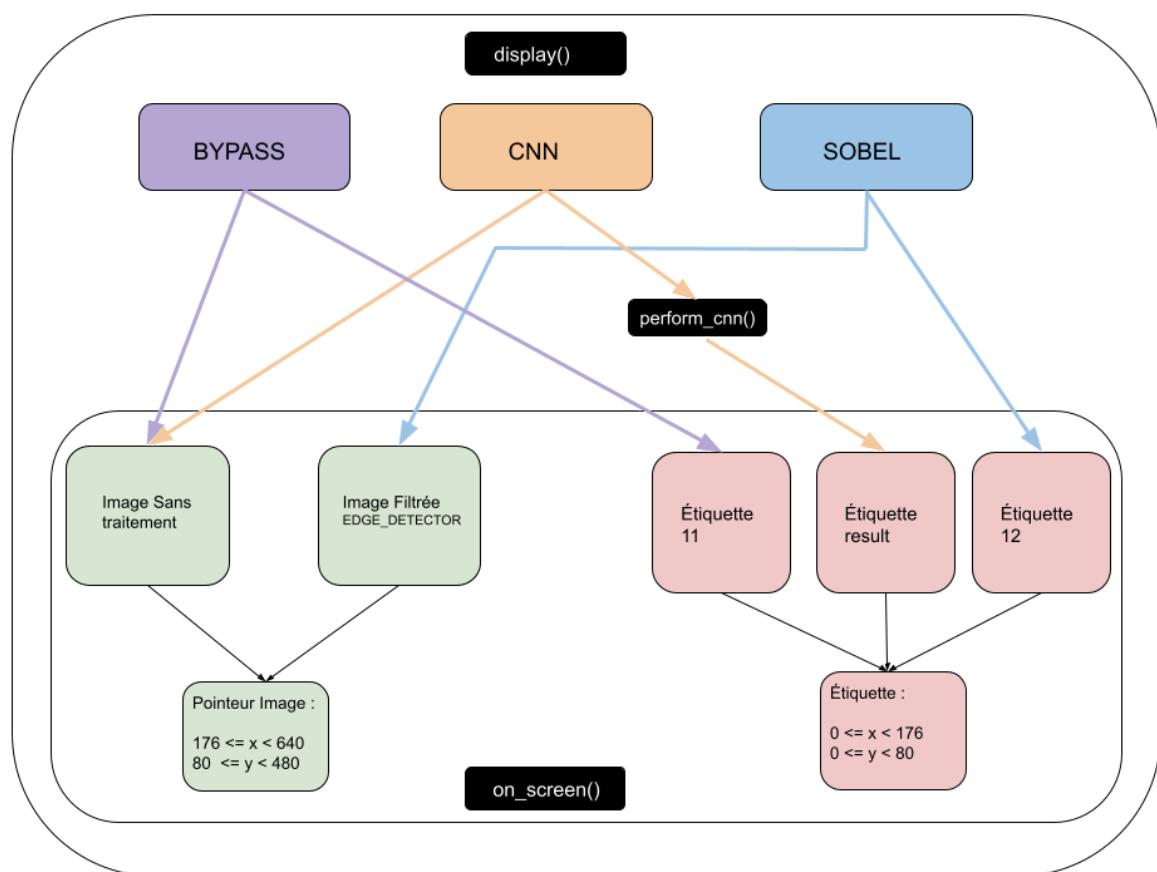


Schéma récapitulatif du fonctionnement de l'affichage

D. Development of the Sobel filter

Un filtre Sobel est un type de filtre de traitement d'image utilisé pour détecter les contours dans une image. Il porte le nom de ses inventeurs, Irwin Sobel et Gary Feldman, qui ont introduit cette technique dans un article en 1968. Le filtre Sobel est particulièrement utilisé dans le domaine du traitement d'image et de la vision par ordinateur.

Le filtre Sobel est souvent utilisé pour la détection des contours en effectuant une opération de convolution sur une image en niveaux de gris. Il utilise deux masques (ou noyaux/kernels) distincts, un pour la détection des contours horizontaux et l'autre pour la détection des contours verticaux. Ces masques sont appliqués à l'image originale en effectuant une convolution, qui est essentiellement une opération de pondération des pixels.

Les masques Sobel ressemblent généralement à ceux-ci :

kernel pour les contours horizontaux (Gx) :

:

$$\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

kernel pour les contours verticaux (Gy)

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

En appliquant ces masques à une image, on obtient deux images résultantes, représentant respectivement les gradients horizontaux (GX) et verticaux (GY) de l'image d'origine. Ces images peuvent ensuite être combinées pour former une nouvelle image représentant les contours détectés.

Cependant, pour simplifier les choses, dans ce projet nous utiliserons un seul noyau qui est le suivant :

$$\begin{matrix} -0.125 & -0.125 & -0.125 \\ -0.125 & 1 & -0.125 \\ -0.125 & -0.125 & -0.125 \end{matrix}$$

Ainsi, nous n'avons pas besoin de nous préoccuper de combiner les 2 matrices résultantes.

Le filtre Sobel est largement utilisé dans des applications telles que la détection de bordures, la segmentation d'images et d'autres tâches de traitement d'image où la détection des contours est importante.

E. Development of the convolutional neural network (CNN) CIFAR-10

Développement Technique

Un réseau neuronal convolutif (CNN) pour CIFAR-10 est un modèle d'apprentissage profond spécialement conçu pour la classification d'images à partir du jeu de données CIFAR-10, qui comprend 10 classes d'objets différents. Dans notre cas, ces 10 classes sont : airplane, automobile, bird, cat, deer, dog, frog, horse, ship et truck.

Le réseau de neurones est réalisé en différentes étapes. Chaque étape correspond à un bloc précis qui réalise une fonction spécifique. Les différents types de blocs sont :

1. **Convolution** : Applique des filtres pour détecter des motifs et caractéristiques spécifiques dans l'image, en effectuant une opération de convolution entre le noyau (filtre) et les régions locales de l'image.
2. **ReLU (Rectified Linear Unit)** : Introduit une non-linéarité en remplaçant toutes les valeurs négatives par zéro.
3. **Maxpool** : Réduit la dimension de l'image en conservant uniquement les valeurs maximales dans des régions prédéfinies, permettant une réduction de la complexité computationnelle tout en préservant les caractéristiques les plus importantes.
4. **Reshape** : Modifie la forme (dimensions) du tenseur sans changer la distribution des valeurs, généralement utilisé pour préparer les données avant de les fournir à une autre couche du réseau.
5. **Perceptron** : Couche entièrement connectée où chaque neurone est connecté à tous les neurones de la couche précédente, réalisant une transformation linéaire suivie d'une fonction d'activation pour produire des sorties adaptées à la tâche spécifique du réseau.

Ces blocs sont appliqués selon les étapes décrites dans le schéma ci-dessous. Cela permet de prendre en entrée une image de 24*24 pixels sur 3 canaux (RGB) et de produire en sortie un vecteur de taille 10 où chaque valeur correspond à la probabilité que la classe soit présente dans l'image. Pour prédire le résultat, il suffit de regarder l'indice de la valeur la plus élevée et de faire correspondre à la liste des classes. Pour exécuter la succession d'étapes, nous avons une fonction par blocs. Nous avons déclaré 2 matrices statiques de taille 64*24*24 et 64*12*12 qui correspondent aux 2 matrices les plus grosses nécessaires, puis chaque fonction "joue au ping-pong" en interchangeant à chaque étape les 2 matrices entre l'entrée et la sortie.

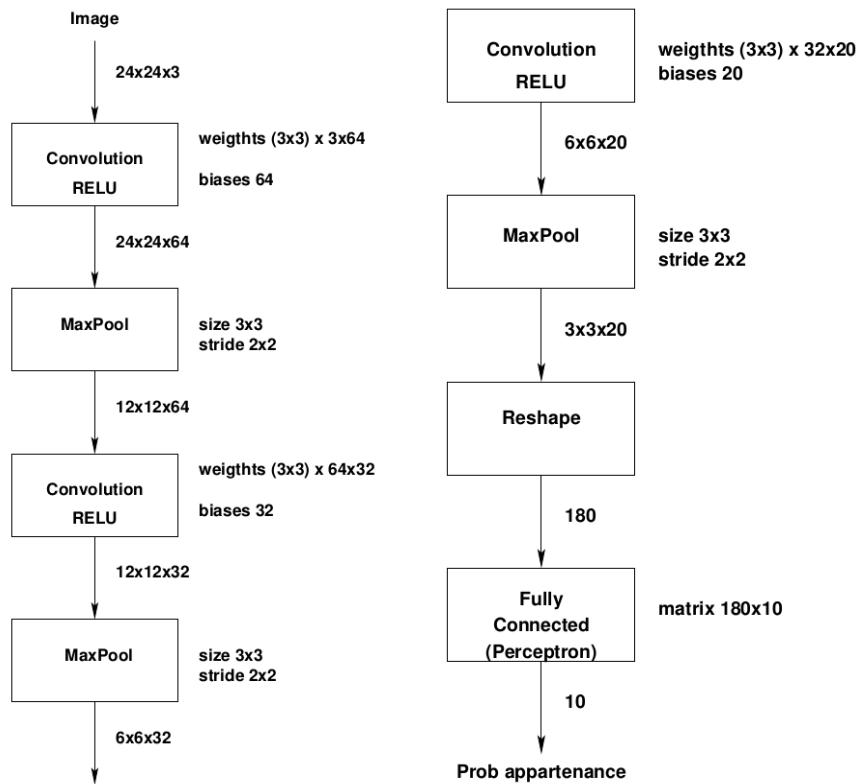


Schéma bloc du fonctionnement du CNN

La réalisation de ce réseau à principalement été basée sur le projet similaire réalisé dans la matière d'architecture. Dans le projet d'architecture, nous avons réalisé 4 différentes versions du CNN. Une première version pour comprendre l'algorithme en python. Une version C++ en utilisant des doubles (virgule flottante). Une version en C++ avec le type ac_fixed (virgule fixe) et une version en C++ pour catapult. Par la suite les 3 versions en C++ ont été mergé ensemble pour ne former plus qu'une version avec des macros de compilation. C'est de la version C++ virgule flottante dont nous sommes repartis. Nous avons changé les types pour correspondre au type donné dans l'ébauche du projet dans les différents fichiers Header.

Test du CNN

La partie CNN a pu être testée séparément grâce à une structure de dossier bien ordonnée comme suit :

- bin** : Ce dossier est utilisé pour stocker les fichiers binaires générés lors de la compilation du projet.
- build** : C'est le répertoire où les fichiers intermédiaires générés lors du processus de compilation sont stockés.
- src** : Ce dossier contient les fichiers source de votre projet.
- include** : Ce dossier est souvent utilisé pour stocker les fichiers d'en-tête (headers).
- testCpp** : Ce dossier contient des fichiers de test pour vérifier le bon fonctionnement des différentes parties du code.

6. **CMakeLists.txt** : Il s'agit d'un fichier de configuration pour CMake, un outil de génération de build. Ce fichier spécifie comment le projet doit être construit, quelles sont les sources, les fichiers d'en-tête, les dépendances, etc.
7. **Readme.md** : Pour finir, un Readme permet d'expliquer comment compiler cette partie.

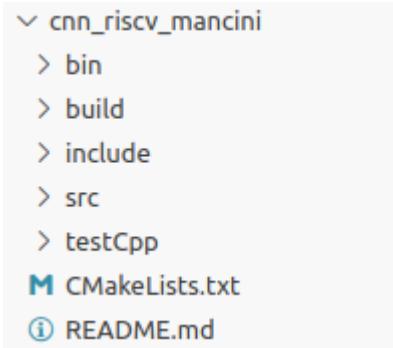


Schéma de l'organisation des dossiers/fichiers

Le test peut être compilé de la façon suivante :

1. Création du répertoire build : `mkdir build`
2. Accéder au répertoire build : `cd build`
3. Génération des fichiers de configuration : `cmake ..`
4. Compilation : `make`

Cela permet de compiler tous les fichiers de tests (ici seulement 1). Le CNN peut ainsi être testé. Le test donne un résultat de 76% de taux de réussite sur 100 images. L'exécution du test se fait via le `imagesMem.h` aussi présent dans `testCpp` contenant 100 images comme elles seraient générées en sortie du système d'image preprocessing. La sortie peut être comparée aux attentes aussi présentes dans le fichier de 100 images pour connaître le pourcentage de réussite du CNN.

Comparatif avec le projet Architecture

La principale différence avec CNN du projet architecture vient du fait que nous avons utilisé ici une virgule flottante (type float) alors que sur le projet architecture, nous avons utilisé une virgule fixe. Une autre différence est que dans le CNN du projet architecture, nous pouvions paralléliser des tâches pour réduire le temps de calcul, ce qui est impossible ici sur un projet exécuté en software. Cette différence peut être vue sur le temps de calcul. Sur le projet d'architecture, nous avions un résultat en moins d'une seconde (avec peu d'optimisation) alors qu'ici nous avons un résultat en quelques secondes et nous n'avons pas d'optimisation possible pour améliorer ce résultat.

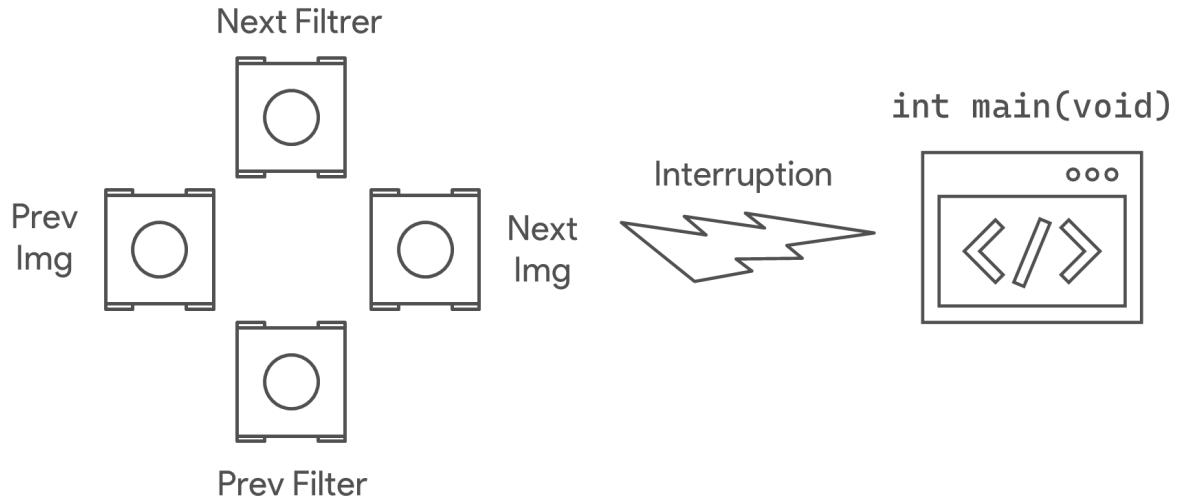
Pistes d'amélioration du CNN

Une amélioration qui n'a pas été amenée dans cette version du CNN aurait été de stocker les différents poids et biais sur la carte SD au lieu de les stocker directement dans le programme comme actuellement. Cela impliquerait quelques petites modifications comme le

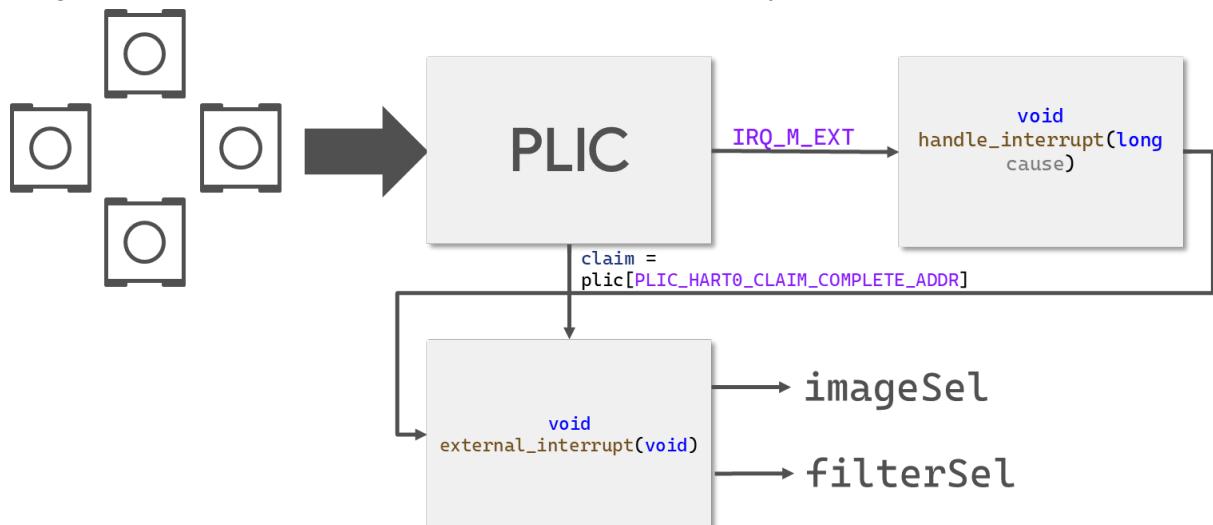
changement de la fonction top level pour la remettre dans son état d'origine suggéré par la trame du projet.

F. Interruptions via push buttons

L'objectif de cette tâche est d'implémenter l'usage des boutons poussoirs présents sur la carte afin de changer l'image à l'écran ou le filtre appliqué à celle-ci. De plus, chaque action associée à bouton poussoir doit être gérée par interruption.



Pour cela on doit utiliser le Platform Level Interrupt Controller (PLIC) de notre système RISC-V. Celui-ci permet de recevoir des signaux d'interruption extérieur, dans notre cas provenant d'un bouton poussoir, afin de les prendre en compte et de les identifier en fonction de sa nature. Dans notre cas, cela va nous permettre de connaître quel bouton poussoir a été appuyé. La fonction handle_interrupt quant à elle, va rédiger notre interruption externe vers external_interrupt qui va ensuite s'occuper de modifier les deux variables glocales imageSel et filterSel en fonction de quel bouton a été appuyé.



Pour cela on a eu trois fonctions à modifier. La première était init_csr qui a comme objectif de réinitialiser tous les Control Status Registers, comme on a pu voir en TP.

La deuxième est enable_plic_interruptions qui a pour but d'activer les interruptions. Pour cela il y a plusieurs modifications à faire :

- Dans un premier temps, on doit activer les priorités pour chaque boutons.
- Ensuite on doit définir le seuil de priorité à 0. Ce seuil est lié aux identifiants des priorités. Par exemple le bouton west à l'ID 1, l'est la 2 etc... De plus, toutes les interruptions qui ont une priorité en dessous du seuil sont désactivées.
- Pour continuer on doit vider la file d'attente de priorité.
- On active ensuite les interruptions 1, 2, 3, et 4.
- On indique au système que des interruptions externes peuvent survenir.
- Enfin on active globalement les interruptions.

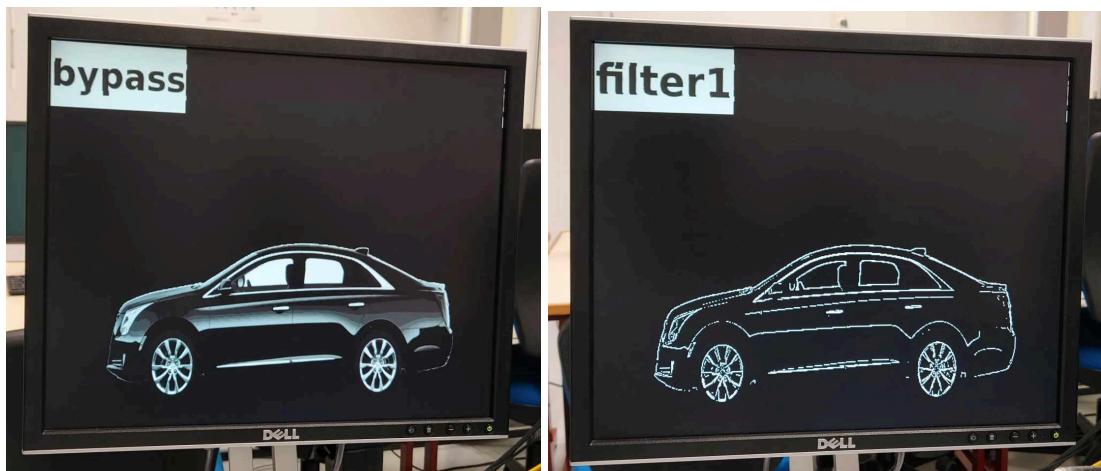
La dernière fonction modifiée était la fonction external_interrupt qui s'occupe de modifier les valeur des variables imageSel et filterSel. Dans un premier temps, elle récupère la nature de l'interruption afin de savoir quel bouton a été appuyé et on coupe les interruptions. Ensuite s'il n'y a plus de rebond on effectue la modification des valeurs. Pour finir on complète l'interruption en réécrivant le claim dans le PLIC et on réactive les interruptions.

Résultats et pistes d'amélioration

Après avoir détaillé le fonctionnement technique, il est important de pouvoir tirer des conclusions de ce projet : pour ceci, il est nécessaire de passer par une analyse de nos résultats et une remise en question, en détaillant les améliorations possibles.

Les résultats sur carte

Nos résultats sur la carte FPGA du filtre sobel sont assez bons. Nous avons de bons élèves comme les images simples sur fond uni.

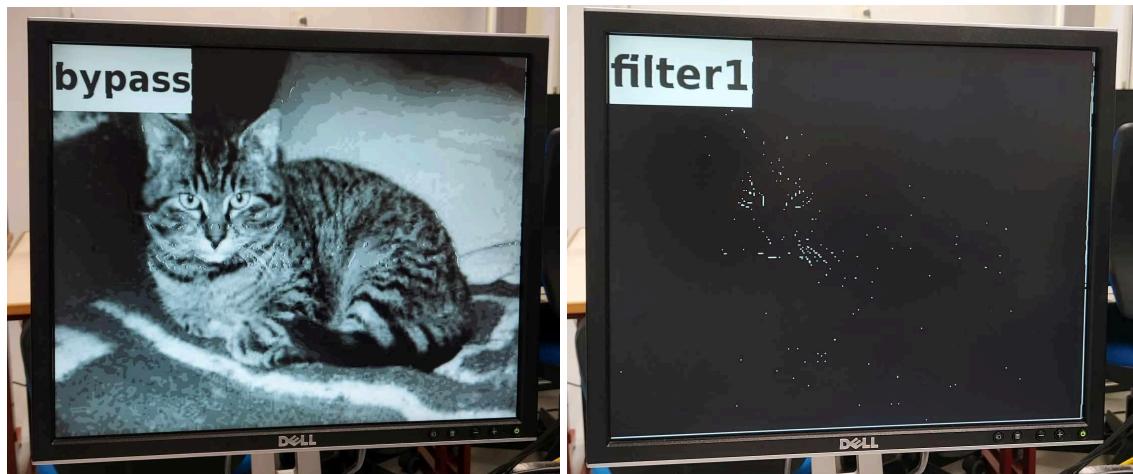


Avec et sans filtre sobel sur une image de voiture sur fond noir.



Avec et sans filtre sobel sur une image d'oiseau sur fond blanc.

Cependant, dès que les images deviennent un peu trop complexes, ou que le fond n'est pas uni, le filtre ne reconnaît plus grand chose.



Avec et sans filtre sobel sur une image de chat sur un lit.

Les résultats sur carte donnée par le CNN sont très satisfaisants. Sur la totalité des 10 images, 2 images ne sont pas détectées correctement.



Détection des photos (avec à gauche une détection correct et à droite une détection incorrecte)

Les perspectives d'amélioration

Le premier axe d'amélioration de ce projet serait le temps de "chargement" du module. En effet, le logiciel est conçu de manière à d'abord charger les images, puis les traiter et enfin donner la main à l'utilisateur. Une optimisation de la lecture des images aurait pu être envisagée ou bien simplement de laisser la main à l'utilisateur dès que la première image est chargée. On pourrait alors commencer à générer des interruptions pendant le chargement des autres images par exemple pour effectuer le traitement cnn ou convolutif du filtre sobel. Cependant, une telle solution nous aurait demandé de nombreuses heures supplémentaires.

Une des pistes d'amélioration pour le sobel filter aurait été d'utiliser 2 kernels de convolution : un kernel vertical et un kernel horizontal. Bien que nos résultats sur le filtre sobel soient satisfaisants, ils auraient sans doute été meilleurs sur les images plus "difficiles".

Le CNN pourrait être amélioré en stockant les biais et les poids sur la carte SD comme expliqué dans la partie Piste d'amélioration du CNN

Conclusion Globale

Dans l'ensemble, le projet démontre le succès du CNN dans son application spécifique, bien qu'il subsiste des opportunités d'amélioration vues ci-dessus. L'expérience acquise au cours de ce projet a été enrichissante, nous incitant à une réflexion approfondie et à des comparaisons. Par exemple, nous avons pu nous rendre compte que le software est plus rapide, et surtout plus simple à implémenter, car il nécessite seulement de placer une carte SD dans le FPGA. La répartition des différentes parties du code au sein du groupe a favorisé une communication efficace, renforçant la rapidité et l'efficacité du déroulement du projet. En outre, cette initiative a été une occasion d'améliorer nos compétences en programmation, que ce soit en C ou en compréhension de la conception matérielle sur FPGA. Ce projet constitue ainsi une étape significative dans notre parcours d'apprentissage et de développement professionnel.

Compétences Acquises

La réalisation du projet de groupe en système embarqué numérique a été une expérience enrichissante. Dans le cadre de la compétence "Concevoir ou réaliser des solutions techniques, théoriques ou expérimentales, permettant de répondre à un cahier des charges", nous avons travaillé de manière collaborative pour produire une solution fonctionnelle à un problème technique complexe. Chaque membre du groupe a contribué à une tâche spécifique, s'appuyant sur ses connaissances pour mettre en œuvre la conception de la technique nécessaire. Nous avons respecté les contraintes réglementaires et le temps imparti en adoptant une méthodologie d'évaluation rigoureuse pour obtenir des résultats analysés et compris. Le choix des outils les mieux adaptés a été crucial, tout comme les tests rigoureux de la solution proposée pour assurer sa fiabilité. Enfin, nous avons rendu la solution compréhensible et réutilisable par d'autres, en documentant soigneusement le processus et en assurant une compréhension claire de notre travail, démontrant ainsi notre

capacité à répondre efficacement à un cahier des charges complexe dans le domaine de la microélectronique numérique.

Nous avons aussi pu enrichir notre compétence "Mettre en œuvre une démarche de recherche fondamentale ou appliquée à des fins d'innovation" ainsi que ses sous-compétences de manière significative. Nous avons dimensionné et mis en œuvre la solution technique la plus pertinente en utilisant le SoC RISC-V sur FPGA, justifiant nos choix en fonction des exigences du projet. Malgré les contraintes réglementaires et temporelles, nous avons su respecter les délais tout en reconnaissant la contribution de chacun au sein du groupe. La documentation prospective a permis d'anticiper les défis et les verrous potentiels. Enfin, nous avons valorisé les résultats de manière adaptée aux enjeux et aux publics, démontrant ainsi notre compréhension approfondie du projet.

Enfin, nous avons pu consolider nos compétences en coopération, démontrant un engagement collectif dans la réalisation du système embarqué numérique. Chacun a su trouver sa place et a agi en tant que moteur dans les différentes composantes du projet technique. En respectant scrupuleusement des contraintes externes et des exigences réglementaires, nous avons assuré la fonctionnalité du projet, en utilisant notamment des outils de gestion de projets tels que Notion. La compréhension de ces contraintes, ainsi que la réalisation de tests approfondis et la gestion globale du projet, ont été des éléments clés de notre réussite. En somme, ce projet nous a permis d'acquérir de manière collective des compétences cruciales pour travailler harmonieusement dans des projets techniques d'envergure.

Annexes

Number of images to read : 14, MIN = 1 MAX = 14

Loading 1.ppm

Le fichier 1.ppm est un fichier ppm P3.

File size: 4301323 and image size : 640 * 480 = 307200

n_image = 1

Closing file 1.ppm

Loading 2.ppm

Le fichier 2.ppm est un fichier ppm P3.

File size: 4301323 and image size : 640 * 480 = 307200

n_image = 2

Closing file 2.ppm

Loading 3.ppm

Le fichier 3.ppm est un fichier ppm P3.

File size: 4301323 and image size : 640 * 480 = 307200

n_image = 3

Closing file 3.ppm

Loading 4.ppm

Le fichier 4.ppm est un fichier ppm P3.

File size: 4301323 and image size : 640 * 480 = 307200

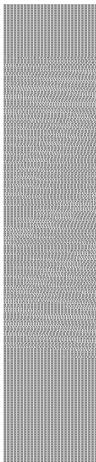
n_image = 4

Closing file 4.ppm

Loading 5.ppm
Le fichier 5.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 5
Closing file 5.ppm
Loading 6.ppm
Le fichier 6.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 6
Closing file 6.ppm
Loading 7.ppm
Le fichier 7.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 7
Closing file 7.ppm
Loading 8.ppm
Le fichier 8.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 8
Closing file 8.ppm
Loading 9.ppm
Le fichier 9.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 9
Closing file 9.ppm
Loading 10.ppm
Le fichier 10.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 10
Closing file 10.ppm
Loading 11.ppm
Le fichier 11.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 11
Closing file 11.ppm
Loading 12.ppm
Le fichier 12.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 12
Closing file 12.ppm
Loading 13.ppm
Le fichier 13.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200
n_image = 13
Closing file 13.ppm
Loading 14.ppm
Le fichier 14.ppm est un fichier ppm P3.
File size: 4301323 and image size : 640 * 480 = 307200

```
n_image = 14
Closing file 14.ppm
Affichage image numero : 1 480*640=307200
Affichage image numero : 2 480*640=307200
Affichage image numero : 3 480*640=307200
Affichage image numero : 4 480*640=307200
Affichage image numero : 5 480*640=307200
Affichage image numero : 6 480*640=307200
Affichage image numero : 7 480*640=307200
Affichage image numero : 8 480*640=307200
Affichage image numero : 9 480*640=307200
Affichage image numero : 10 480*640=307200
Affichage image numero : 11 480*640=307200
Affichage image numero : 12 480*640=307200
Affichage image numero : 13 480*640=307200
Affichage image numero : 14 480*640=307200
Starting filtering!
Filtering done !
```

Annexe 1 - Sortie picocom du FPGA



Annexe 2 - Structure du fichier overlay.c, avec des sections de 176 x 80 pixels codé en nuances de gris

```

optimizedtab = array([[[0 for _ in range(176)]for _ in range(80)]for _ in range(12)])
rangetab = []

i=34 #offset de declaration
word = ''
while (i < len(overlays_values)):
    if (overlays_values[i] != ","):
        if (overlays_values[i] != "}"):
            word = word + overlays_values[i]
        else :
            num=int(word)
            rangetab.append(num)
    else :
        num=int(word)
        rangetab.append(num)
        word = ''
    i = i+1

```

*Annexe 3 - Extrait du code Python compt_overlay.py pour décrypter
la composition du fichier overlay.c*