UMONS

Certificat d'université en

Intelligence Artificielle

27-28 MARS 2021 (Hôtel Utopia)

3ÈME ÉDITION DU WORKSHOP RÉSIDENTIEL

Certificat IA UMONS

HackIA'21

Système Edge IA pour maisons et villes intelligentes

Ingénieurs | Informaticiens | Doctorants

Coachs Techniques :

Pr. Sidi Ahmed MAHMOUDI, UMONS

Ir. Yassine AMKRANE, UMONS

Adriano GUTTADAURIA, UMONS

Membres de Jury :

Pr. Pierre MANNEBACK, UMONS

Pr. Thierry DUTOIT, UMONS

Pr. Xavier SIEBERT, UMONS

Pr. Souhaib BEN TAIEB, UMONS

Dr. Sandrine BROGNAUX, UMONS

Membre représentant, Accenture

Nombre de participants : 12
Nombre de groupes : 02

Hôtel Utopia
Chaussée Brunehault 392,
7050 Masnuy-Sait-Jean

065 84 87 85

UMONS Université de Mons | POLYTECH MONS | Faculté des Sciences | numediart INSTITUT DE RECHERCHE POUR LES TECHNOLOGIES CREATIVES DE L'UMONS | inforTech UMONS RESEARCH INSTITUTE FOR INFORMATION TECHNOLOGY AND COMPUTER SCIENCE | accenture

Atelier d'Intelligence Artificielle (I-TSIA-202)

Système IA embarqué pour maisons et villes intelligentes

Edge AI System for Homes and Smart Cities

Table des matières


Atelier d'Intelligence Artificielle (I-TCTS-202)

Système IA embarqué pour maisons et villes intelligentes

Edge AI System for Smart Cities and Smart Homes


1. Enoncé

L'objectif de ce workshop est de développer un système d'intelligence artificielle embarqué sur des ressources Edge IA (matériel proche des capteurs de collecte de données). Le système s'appuiera sur les techniques et modèles Deep Learning vues et développés durant les défis du certificat IA. Ces modèles seront combinés pour fournir un module « Edge AI » appliqué aux vidéos capturées en temps réel. Chaque groupe aura le choix entre deux possibilités : module **« Edge AI »** pour maisons intelligentes ou un module **« Edge AI »** pour villes intelligentes.

1.1. Développement d'un module « Edge AI » pour maisons intelligentes

Développer un support pour maisons intelligentes avec une application utilisant des techniques et modèles Deep Learning pour détecter et localiser différents objets à partir d'images capturées via la caméra. Ce module devra utiliser deux ou trois modèles :

1.1.1. Modèle « Face »

Le premier modèle servira comme système d'authentification faciale grâce à la reconnaissance de visages. Il permettra d'autoriser l'utilisateur à employer les autres applications (modèles IA pour maisons intelligente) si la personne est à la fois reconnue et autorisée (via son visage).

1.1.2. Modèle « Fire »

Le deuxième modèle (de **classification**) consiste à détecter les feux ou fumées (ex. détecter des cas d'oubli de feu de gazinière allumée sans raison, présence de fumée, etc.) à partir d'images capturées via la caméra. Vous pouvez utiliser et adapter vos modèles développés lors du défi 1 du certificat IA.

1.1.3. Modèle « Personal »

Le 3ème modèle consiste à **localiser** les objets personnels (clés, trousseaux de clés, télécommande, smartphone, portefeuille, etc.). Là aussi, vous pouvez repartir du modèle de localisation développé durant le défi 1 avant de l'augmenter pour détecter et localiser plusieurs objets, un réentrainement est nécessaire avec plus de classes.

1.1.4. Modèle « Movement »

Ce modèle consiste à reconnaitre et classifier les mouvements (marche, marche rapide, chute, dispute, etc.) dans une scène filmée par la caméra. Pour réaliser ce modèle, il est conseillé d'utiliser des bases de données publiques présentant des séquences vidéo annotées avec différents types de mouvements.

1.2. Développement d'un module « Edge AI » pour villes intelligentes

Développer un support pour villes intelligentes avec une application utilisant des techniques et modèles Deep Learning pour détecter et localiser différents objets ou situations à partir d'images capturées via la caméra. Ce module devra aussi utiliser deux ou trois modèles :

1.2.1. Modèle « Face »

Le premier modèle servira comme système d'authentification faciale grâce à la reconnaissance de visages. Il permettra d'autoriser l'utilisateur à employer les autres applications (modèles IA pour villes intelligente) si la personne est à la fois reconnue et autorisée (via son visage).

1.2.2. Modèle « Fire »

Le deuxième modèle (de classification) consiste à détecter les feux ou fumées dans des forêts proches de la ville intelligente à partir d'images capturées via la caméra. Vous pouvez utiliser et adapter vos modèles développés lors du défi 1 du certificat IA.

1.2.3. Modèle « Suspect »

Ce modèle (de localisation) consiste à détecter la présence d'un ou plusieurs objets suspects ou non autorisés (ex. bâtons, sacs isolés, armes, etc.) dans une scène filmée par la caméra.

1.2.4. Modèle « Movement »

Le troisième modèle consiste reconnaitre et classifier les mouvements (accident, chute, dispute, etc.) dans une scène filmée par la caméra. Pour réaliser ce modèle, il est conseillé d'utiliser des bases de données publiques présentant des séquences vidéo annotées avec différents types de mouvements. Un exemple est accessible ici.

Note : chaque groupe devra développer/intégrer **trois modèles** au moins. Si un groupe le souhaite, il peut augmenter la solution par d'autres modèles mais cela est **optionnel**. Chaque groupe peut aussi intégrer d'autres idées innovantes pour améliorer la solution tout en gardant l'objectif principal du projet choisi.

2. Phases du Workshop

Afin de répondre à l'énoncé décrit ci-dessus et d'organiser un partage de tâches entre les membres de chaque groupe, nous vous proposons de suivre les étapes suivantes :

- **Partie 0 :** choix du projet ;
- **Partie 1 :** développement et entraînement des modèles de classification et de localisation ;
- **Partie 2 :** développement du programme de test (inférence) à partir de séquences vidéo, solution embarquée sur la ressource Edge : Jetson Xavier ;
- **Partie 3 :** optimisation/compression de modèles **et/ou** analyse d'explicabilité de la solution « **XAI** ».

2.1. Partie 0 : choix du projet

Chaque groupe devra donner son choix de projet parmi les deux possibilités :

a. Module **« Edge AI »** pour maisons intelligentes
b. Module **« Edge AI »** pour villes intelligentes

Choix à compléter ici pour le **Mercredi 17/03/2021** après concertation dans chaque groupe.

2.2. Partie 1 : développement et entraînement des modèles

Le choix du projet vous permettra de sélectionner les modèles à développer. Les entraînements peuvent être réalisés avec les mêmes frameworks utilisés lors du défi 1 (Python, Tensorflow, Keras, Pytorch, OpenCV, etc.). En termes de ressources, vous pouvez utiliser Google Colab ou Floydhub. Chaque groupe disposera, si besoin, de 50 heures de calcul avec Floydhub. Les principaux modèles à développer et entrainer sont :

2.2.1. Modèle « Face »

Ce modèle permettra d'identifier et reconnaître les visages des personnes présentes dans la scène. Pour cela, nous vous recommandons d'utiliser l'architecture FaceNet permettant de détecter et reconnaître des personnes (Nom de la personne). Bien sûr, il faudra entrainer le modèle avec une base de visages personnalisée (exemple : visages des différents membres du groupe). Cette partie peut être réalisée en trois sous-étapes :

a. **Collecte et préparation de la base de visages :** pour simplifier la collecte de données, nous vous proposons de travailler sur vos visages en sauvegardant une vidéo pour chaque membre du groupe. Ensuite, il faudra extraire les images de ces vidéos en utilisant le programme « video_to_image.py ». Pour lancer le programme, il faudra :
   - Se connecter à votre environnement de travail (PC personnel, Google Colab, Floydhub, etc.)
   - Via le terminal, aller sur le dossier contenant vos vidéos sauvegardées
   - Lancer la commande suivante : **python video_to_image.py**

   Vous devrez avoir six dossiers, dont chacun contient les images d'un membre du groupe. Il faudra les copier dans le dossier « PERSONS » récupérable en cliquant ici. Vous aurez donc 14 classes (8 classes pour les membres du service IILIA et 6 classes représentant les membres de groupe). Veuillez garder la confidentialité des images.

b. **Extraction et localisation des visages :** consiste à extraire les visages à partir des images collectées plus haut (Fig. 1), une étape primordiale pour la reconnaissance des visages. L'extraction se fera via un algorithme de détection, fourni dans le projet facent. Pour lancer ce programme, il faudra lancer la commande :

   python src/align/align_dataset_mtcnn.py PERSONS PERSONS_ALIGNED

   Notons que l'ensemble des étapes et données décrites ci-dessous sont mises en place dans notre programme « face_recognition_tf2.ipynb » que nous avons adapté pour les versions 2.x de tensorflow.

- align_dataset_mtcnn.py : représente le programme de détection (crop) de visages ;
- PERSONS : représente le dossier contenant les photos de personnes (voir plus haut) ;
- PERSONS_ALIGNED : représente le résultat (visages détectés), nous vous conseillons de vérifier les images résultantes étant donné que cet algorithme n'est pas efficace à 100%. Ce dossier présente donc notre base d'apprentissage pour la reconnaissance de visages.

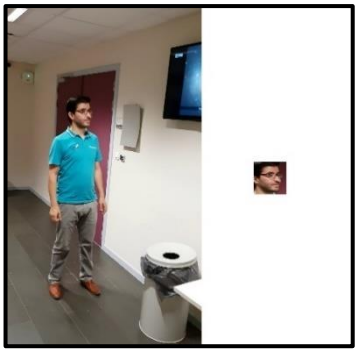

FIGURE 1 : EXEMPLE DE DÉTECTION ET LOCALISATION DE VISAGE

c. **Reconnaissance des visages :** une fois base de visages préparée, nous pouvons utiliser l'algorithme de reconnaissance de visages en utilisant le modèle FaceNet, pré-entraîné par Google sur une base de 16 millions de visages. Vous êtes donc invités à utiliser la technique de « *Transfer Learning* » pour entraîner ce modèle avec votre base de visages. Pour lancer cet apprentissage, il faudra lancer la commande **:**

```
python src/classifier.py TRAIN FPMS_PERSONS_ALIGNED/ 20170511-185253.pb
model_checkpoints/my_classifier.pkl --batch_size 100
```

Le résultat de cet apprentissage est représenté par le modèle « **my_classifier.pkl** ». Si vous voulez lancer l'entrainement à nouveau, il faut lancer la commande suivante :

```
python src/train_softmax.py --logs_base_dir logs/facenet --models_base_dir . --data_dir
FPMS_PERSONS_ALIGNED/ --model_def models.inception_resnet_v1 --optimizer ADAM
```

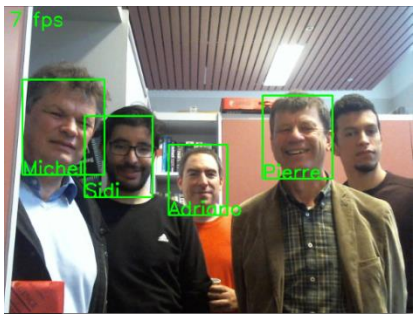

FIGURE 2 : EXEMPLE DE RECONNAISSANCE DE VISAGES

d. **Reconnaissance de visages en temps réel :** il faudra tester le modèle développé ci-dessus « **My_classifier.pkl** » avec les images provenant, de la webcam ou d'une vidéo enregistrée. Vous pouvez utiliser le programme « **real_time_face_recognition.py** » ou « **offline_face_recognition.py** » fourni avec le dossier « facenet/contributed ».

2.2.2. Modèle « Fire »

Il s'agit du modèle de classification développé lors du défi 1, vous pouvez utiliser/adapter vos modèles du défi 1 en fonction de votre choix.

2.2.3. Modèle « Personal »

Ce modèle à consiste à détecter et localiser les objets personnels (clés, télécommande, smartphone, portefeuille, etc.). Vous pouvez repartir du modèle de localisation développé durant le défi 1 pour le réentraîner avec une base de données contenant plusieurs classes d'objets annotées (télécommande, smartphone, portefeuille, etc.). N'hésitez pas à utiliser des bases de données publiques ou des modèles déjà pré-entrainés. Pour faciliter la tâche, nous partageons avec vous une base de données annotée, que vous pouvez augmenter ou réduire, ainsi que les éléments nécessaires pour l'annotation (si vous souhaitez annoter vous-même) :

a. **Base de données annotée d'objets personnels « BD_Personal » :** accessible via ce lien. Cette base de données est représentée par deux sous dossiers :
   - **Train :** 7200 images représentant différentes classes + un fichier « annotations.csv » (Fig. 3)
   - **Test :** 800 images représentant différentes classes + un fichier « annotations.csv »

Notons que la base de données fournie a été générée automatiquement à l'aide de notre programme python **« mix_objects_background.py »** permettant de déposer de manière aléatoire les objets sur les images de background en sauvegardant les coordonnées de positions d'objets (annotations).

b. **Eléments de génération et annotation de la base de données :** vous pouvez aussi annoter et adapter la base de données en utilisant les fichiers partagés via ce lien :
   - **Background :** images de background ne contenant pas d'objet spécifique ;
   - **Objets :** 09 sous dossiers dont chacun contient des images avec fond transparent d'un type d'objet ;
   - **Programme d'annotation « mix_objects_background.py »** : pour mixer les objets et background.

FIGURE 3 : EXEMPLE D'IMAGE ANNOTÉE AVEC DEUX OBJETS PERSONNELS « CHAPEAU ET PORTEFEUILLE »

2.2.4. Modèle « Suspect »

De la même manière avec laquelle vous avez travaillé lors du défi 1, vous pouvez développer un modèle (de classification ou localisation) permettant de détecter la présence d'un ou plusieurs objets suspects (exp. bâton, arme, couteau, etc.) dans une image. Il faudra donc utiliser la technique du « *Transfer Learning* » pour exploiter un modèle, pré-entrainé, et l'adapter par rapport à votre base de données, le but étant d'identifier la présence d'un ou plusieurs objets suspects dans l'image de la vidéo. N'hésitez pas à utiliser des bases de données publiques ou des modèles déjà pré-entrainés. Pour faciliter la tâche, nous partageons avec vous une base de données annotée, que vous pouvez augmenter ou réduire, ainsi que les éléments nécessaires pour l'annotation si vous souhaitez annoter vous-même :

c. **Base de données annotée d'objets suspects « BD_Suspect » :** accessible via ce lien. Cette base de données est représentée par deux sous dossiers :
   - **Train :** 7200 images représentant différentes classes + un fichier « annotations.csv » (Fig. 4)
   - **Test :** 800 images représentant différentes classes + un fichier « annotations.csv »

Notons que la base de données fournie a été générée automatiquement à l'aide de notre programme python **« mix_objects_background.py** » permettant de déposer de manière aléatoire les objets sur les images de background en sauvegardant les coordonnées de positions d'objets (annotations).

d. **Eléments de génération et annotation de la base de données :** vous pouvez aussi annoter et adapter la base de données en utilisant les fichiers partagés via ce lien :
   - **Background :** images de background ne contenant pas d'objet spécifique ;
   - **Objets :** 09 sous dossiers dont chacun contient des images avec fond transparent d'un type d'objet ;
   - **Programme d'annotation « mix_objects_background.py** » : pour mixer les objets et background.

FIGURE 4 : EXEMPLE D'IMAGE ANNOTÉE AVEC UN OBJET SUSPECT « DRONE »

2.2.5. Modèle « Mouvement »

Ce modèle a pour objectif de reconnaître des mouvements (exp. marche, marche rapide, fuite, chute, dispute, etc.) à partir de séquences vidéo. Etant donnée qu'une vidéo est toujours représentée par une succession d'images, les modèles de classification d'images pré-entrainés pourront être utilisés pour reconnaitre les mouvements. Toutefois, ces modèles ne prennent pas en compte l'information temporelle ce qui entraine des pertes de précisions. Dans ce contexte, il existe d'autres types de réseaux de neurones profonds (réseaux de neurones récurrents, réseaux de neurones convolutionnels 3D, etc.) qui peuvent combiner les informations spatiales, temporelles (ainsi que les caractéristiques de mouvements) pour générer des modèles capables d'identifier les mouvements avec une précision satisfaisante. Nous vous invitons à consulter les bases de vidéos publiques (Fig. 5) ainsi que des projets de classifications de vidéos partagées en ligne. A l'issue de cette analyse, vous pouvez sélectionner un modèle pré-entrainé avant de l'appliquer à vos données.

a. Exemple de bases de vidéos publiques annotées :

- **HMDB51 :** 6766 clips et 51 classes
- **UCF-101 :** 13320 clips, 27 heures de vidéo, 101 classes
- **Kinetics :** 306425 clips, 400 à 700 classes

BD	Année	Nb. actions	Nb. clips/ action	Clips	Vidéos
HMDB51	2011	51	min. 102	6766	3312
UCF101	2012	101	min. 101	13320	2500
Thumos ('15)	2013	101	/	/	> 20700
Sports-1M	2014	487	1000-3000	1000000	1000000
ActivityNet	2015	200	env. 141	28108	19994
Youtube-8M	2016	4800	/	8264650	8264650
Kinetics	2017	400	min. 400	306245	306245
HACS	2018	200	/	1550000	504000

FIGURE 5 : COMPARAISON ENTRE LES BASES DE VIDÉOS

b. Exemple d'architectures « GitHub » de classification de vidéos (et modèles pré-entrainés) :

- Approche Two-Stream
- Approche LRCN « Long-term Recurrent Convolutional Network »
- Approches I3D et C3D
- Approche SlowFast
- Approche TSM
- Approche R(2+1)D

Nous partageons avec vous aussi un travail d'étudiants « **video_classsification_mmaction.ipynb** » (basé sur l'outil **mmaction**) permettant d'utiliser des modèles de classification de vidéos à l'aide des architectures « TSM », « TSN » et « I3D ».

2.3. Partie 2 : portage de solution sur la ressource Edge : Jetson Xavier

Après validation des modèles, il faudra les porter vers la ressource Edge Nividia Jetson Xavier afin de développer une solution embraquée et appliquée aux vidéos capturées via la caméra connectée à la carte (via un port USB). L'idée serait de combiner les modèles pour fournir un module Edge AI pour maisons ou villes intelligentes en fonction de votre choix. Les figures 6 et 7 illustrent les programmes à développer et compléter (Edge AI systems for Smart Homes and Smart Cities).

```
#        ******************        EDGA AI module For Smart Homes        ******************        #
# face : model de reconnaissance facile
# fire : model de détéction de feu, fumée, etc.
# personal : modèle de détection ou localisation d'objets personnels
# movement : modèle de reconnaissance de mouvements/actions (facultatif)
cap = cv2.VideoCapture(0)    # Capture de la vidéo en temps réel
    while(cap.isOpened()):
        frame = cap.read(); # lecture de chaque image de la vidéo capturée
        faces = face_recognition.identify(frame)    # Modèle de reconnaissance faciale (à lancer pendant 10 secondes par exemple)
        if (face.name in liste)                     # liste : membres authorisés à utiliser le module
        FirePred1 = fire.predict(x)[0] ;
        PersonalPred1 = personal.predict(x)[0] ;
        .....................................       # Facultatif : autres modèles aux choix
        }
cap.release()
cv2.destroyAllWindows()
```

FIGURE 6: "EDGE AI" ALGORITHM FOR SMART HOMES

```
#        ******************        EDGA AI module For Smart Cities        ******************        #
# face : model de reconnaissance facile
# fire : model de détéction de feu, fumée, etc.
# suspect : modèle de détection ou localisation d'objets suspects
# movement : modèle de reconnaissance de mouvements/actions (facultatif)
cap = cv2.VideoCapture(0)    # Capture de la vidéo en temps réel
    while(cap.isOpened()):
        frame = cap.read(); # lecture de chaque image de la vidéo capturée
        faces = face_recognition.identify(frame)    # Modèle de reconnaissance faciale (à lancer pendant 10 secondes par exemple)
        if (face.name in liste)                     # liste : membres authorisés à utiliser le module
        FirePred1 = fire.predict(x)[0] ;
        SuspectPred1 = suspect.predict(x)[0] ;
        MvtPred1 = movement.predict(x)[0] ;         # Facultatif
        }
cap.release()
cv2.destroyAllWindows()
```

FIGURE 7: "EDGE AI" ALGORITHM FOR SMART CITIES

FIGURE 8 : SYSTÈME IA UTILISANT LA RESSOURCE EDGE "JETSON XAVIER"

Vous avez le libre choix de proposer une interface graphique ou pas et avec l'outil de votre choix (tkinter, PyQt, etc.). Notons que chaque groupe disposera d'une carte Jetson Xavier préconfigurée avec les différentes librairies Python, Tensorflow, Keras, OpenCV, etc. Les modalités d'accès aux environnements préconfigurés seront fournies durant le Workshop.

2.4. Partie 3 : optimisation/compression de modèles et/ou analyse d'explicabilité de la solution « XAI »

Pour ceux qui veulent aller plus loin pour ce Workshop, nous vous proposons d'optimiser votre solution en travaillant sur les points suivants :

2.4.1. **Analyser les performances :** des modèles en termes de précision, temps de calcul et espace mémoire ;

2.4.2. **Optimiser et compresser les modèles :** à l'aide des techniques d'élagage « pruning » de réseaux de neurones, quantification et convolutions séparables :

a. **Elagage (pruning) :** les réseaux de neurones profonds sont caractérisés par un nombre de paramètres (hyperparamètres) très élevé dont certains peuvent présenter une redondance inutile. Des techniques d'élagage (pruning) [1] et de Dropout peuvent être utilisées pour identifier et garder uniquement les neurones et connexion les plus significatives afin de générer des modèles à taille réduite préservant une précision suffisante. Il faudra veiller à garder un bon compromis en taille mémoire, temps de calcul et précision des modèles. Les réseaux compressés sont conçus pour avoir moins de paramètres et utiliser moins de mémoire. La figure 9 illustre le principe du pruning.

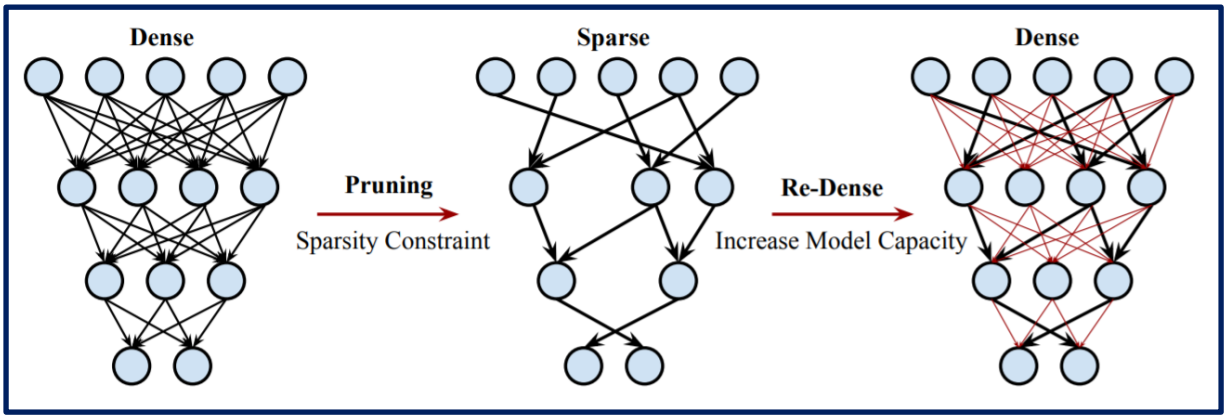

Figure 9 : illustration du principe d'élagage (pruning) de réseaux de neurones profonds [1]

b. **La quantification :** consiste à appliquer un processus d'approximation sur les réseaux de neurones afin de représenter les poids avec des nombre à faible nombre de bits sachant que les poids sont initialement représentés par des nombres en virgules flottantes [2]. Ce processus permet de réduire considérablement la taille mémoire et le temps de calcul des réseaux de neurones profonds. La figure 10 illustre un exemple de quantification de poids d'un réseau de neurones. Le nombre de bits doit être fixé en veillant à ce que la précision initiale reste maintenue car une réduction très grande de la précision des valeurs de poids risquera d'affecter négativement la précision du modèle.

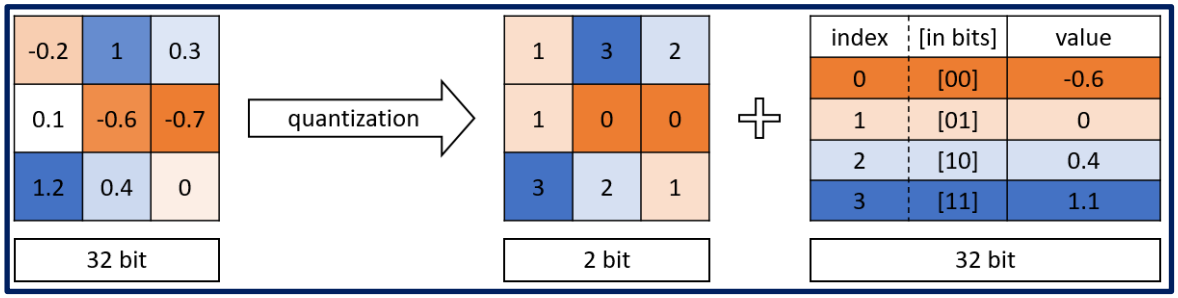

Figure 10 : illustration du principe de quantification

c. **Convolutions séparables :** dans le cas d'application des réseaux de neurones profonds sur des images, nous serons fort probablement amenés à utiliser des couches convolutionelles pour l'extraction de caractéristiques d'images au sein du réseau. Cependant, ces opérations sont très coûteuses en calcul et en mémoire. Dans ce contexte, Google a proposé en 2017 une nouvelle technique de convolution appelée « Depthwise Seperate Convolution » qui consiste à appliquer les filtres de convolutions séparément à chaque canal de l'image contrairement à la convolution classique qui applique un filtre sur l'ensemble des canaux [3]. Les 3 images résultant de cette opération (Depthwise Separate Convolution) seront ensuite combinées pour fournir une image caractéristique (Figure 11). Cette approche permet de réduire de manière significative les temps de calcul et espace mémoire tout en ayant un faible impact sur la précision. Cette technique est utilisée dans l'architecture de classification d'images « MobileNet » connue d'ailleurs par un faible temps de calcul et un espace mémoire réduit.

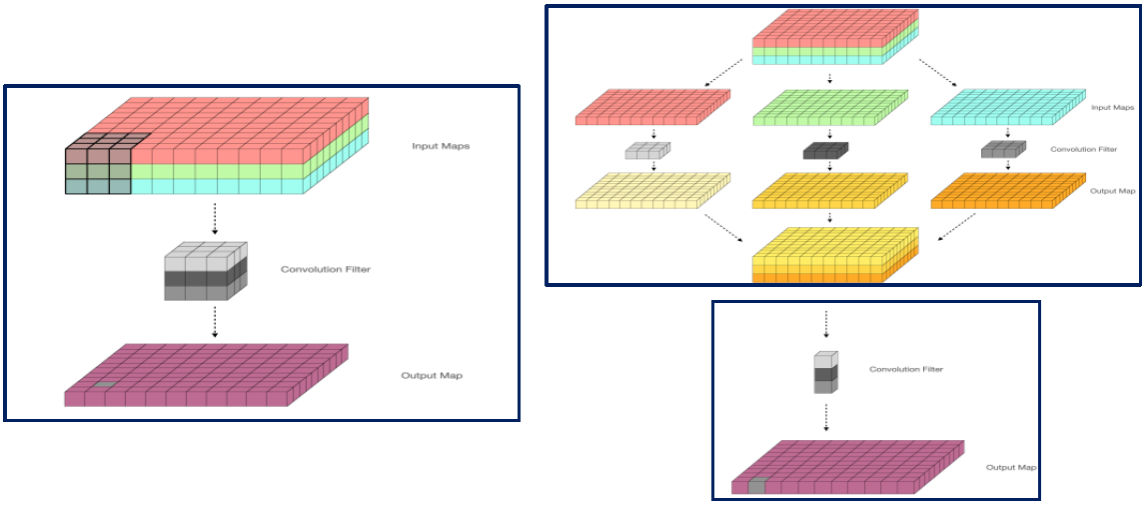

Figure 11 : convolution classique vs. Convolutions séparables

Nous partageons avec vous deux exemples de travaux d'étudiants « model_compresion_MNIST.ipynb » et « model_pruning_Cat_Dog.ipynb » permettant d'appliquer les techniques de prunning et quantification sur un simple réseau de neurones profond.

2.4.3. Expliquer et interpréter vos modèles : à l'aide d'une approche d'« ***Explainable AI, XAI*** » et techniques de visualisation de caractéristiques extraites à partir de réseaux de neurones. L'idée est de calculer, quantifier et visualiser la contribution et influence de chaque caractéristique et le faire transparaitre dans le résultat final du modèle. Récemment, un certain nombre de méthodes d'interprétation de réseaux de neurones profonds ont été proposées où l'interprétation est principalement représentée par la visualisation des parties de l'image ayant une forte ou faible contribution pour la décision [4]. Nous pouvons identifier trois principales catégories d'approches d'explicabilité de réseaux de neurones profonds :

a. Explicabilité basée sur les perturbations : par le remplacement, permutation ou occlusion des caractéristiques ou groupe de caractéristiques afin de calculer la différence (prédiction). Une faible différence signifie que les caractéristiques remplacées (permutées ou occlues) sont moins importantes et vice versa.

b. Explicabilité basée sur l'analyse de rétropropagation : cette approche ne s'appuie pas sur les résultats de modèles mais plutôt sur la structure interne des modèles. Le principe est d'identifier la saillance des caractéristiques d'entrée et couches intermédiaires en analysant les gradients transmis de la sortie vers l'entrée pendant l'apprentissage du réseau de neurones. Il existe dans la littérature plusieurs variantes de méthodes utilisant cette approche tels que : Gradient*Input, Integrated Gradient, Guided Backpropagation, Deconvolutional Network, Layer-wise Relevance Propagation (LRP) [5], DeepLIFT, DeepTaylorDecomposition, Class Activation Mapping (CAM), Grad-CAM [6], etc.

c. Explicabilité basée sur les modèles Proxy : les modèles proxy permettent de réduire les complexités de modèles Machine et Deep Learning afin d'avoir en résultat des modèles ayant des comportements et résultats similaires aux modèles initiaux mais utilisant des architectures simples dont le fonctionnement est plus facile à expliquer. Les principales méthodes utilisant cette approche sont : LIME [7] (modèle linéaire), arbre de décision et DeepRed [8], etc.

Nous partageons avec vous un exemple de programme « **model_explanation_visualisation.ipynb** » permettant d'appliquer les différents techniques d'explicabilité sur un réseau de neurones convolutionnel CNN.

3. Conclusion

Nous tenons à rappeler que l'objectif de ce Workshop est de mettre en œuvre des modèles de classification d'images/vidéos et de localisation d'objets avant de les embarquer dans un module « Edge AI » pour maisons ou villes intelligentes. Nous vous invitons à commencer par le développement d'un module utilisant un nombre minimal de modèles (3) avant de le porter vers la carte Jetson Xavier en offrant un traitement temps réel à partir d'images capturées via la webcam. A l'issue de cela, vous pourrez intégrer les autres tâches optionnelles choisies (compression, explicabilité, intégration d'autres modèles, envoie d'alertes et notifications, etc.) ou proposées par vos soins.

Références :

[1] Han, S., Pool, J., Narang, S., Mao, H., Gong, E., Tang, S., ... & Dally, W. J. (2016). Dsd: Dense-sparse-dense training for deep neural networks. arXiv preprint arXiv:1607.04381.

[2] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

[3] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[4] Vilone, G., & Longo, L. (2020). Explainable artificial intelligence: a systematic review. arXiv preprint arXiv:2006.00093.

[5] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, "Layer-wise relevance propagation: an overview," in Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Springer, 2019, pp. 193-209.

[6] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision (pp. 618-626).

[7] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Model-agnostic interpretability of machine learning". arXiv preprint arXiv:1606.05386.

[8] Zilke, J. R., Mencía, E. L., & Janssen, F. (2016, October). Deepred–rule extraction from deep neural networks. In International Conference on Discovery Science (pp. 457-473). Springer, Cham.