UMONS

Certificat d'Université en

Intelligence Artificielle

29 - 30 AVRIL 2023 (Hôtel Utopia)

5ÈME ÉDITION DU WORKSHOP RÉSIDENTIEL

Certificat IA UMONS
HackIA'23

Système *Edge AI*
Pour Villes Intelligentes

Ingénieurs **Informaticiens** **Doctorants**

COACHS TECHNIQUES

Pr. Sidi Ahmed MAHMOUDI	*UMONS*
Ir. Jean-Sébastien LERAT	*UMONS*
Ir. Mohamed BENKEDADRA	*UMONS*

MEMBRES DE JURY

Pr. Thierry DUTOIT	*UMONS*
Pr. Pierre MANNEBACK	*UMONS*
Pr. Xavier SIEBERT	*UMONS*
Pr. Souhaib BENTAIEB	*UMONS*
Pr. Stéphane DUPONT	*UMONS*
Pr. Mohammed BENJELLOUN	*UMONS*

Nombre de Participants 15
Nombre de Groupes 03

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

065 84 87 85

UMONS POLYTECH MONS Faculté des Sciences numediart inforTech accenture

Atelier d'Intelligence Artificielle (I-ISIA-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


Objectif : développer un système d'intelligence artificielle embarqué sur des ressources *Edge AI* (matériel proche des capteurs de collecte de données). Le système s'appuiera sur les modèles Deep Learning développés durant les défis du certificat IA. Ces modèles seront combinés pour fournir un module « *Edge AI* » appliqué aux vidéos capturées en temps réel. Le résultat serait représenté par un module **« *Edge AI* »** au service des villes intelligentes.

1. Développement d'un module « *Edge AI* » pour villes intelligentes :

Développer un support pour villes intelligentes avec une application utilisant des modèles Deep Learning pour détecter et localiser différents objets à partir d'images capturées via la caméra. Ce module devra utiliser 3 modèles au moins :

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 ville intelligente) si la personne est à la fois reconnue et autorisée (via son visage).

1.2. 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 adapter vos modèles du défi 1 du certificat IA. Des prétraitements d'images peuvent être appliqués pour améliorer la qualité de détection et généraliser. Vous pourrez aussi (facultatif) localiser les feux en utilisant une architecture de détection au lieu de la classification.

1.3. Modèle « Object : Suspect »

Le 3ème modèle consiste à **localiser** les objets personnels (armes, bâton, couteau, drone, appareil photo, grenade, gilet jaune, personne, etc.). Là aussi, vous pouvez repartir du modèle de localisation développé durant le défi 1 avant de l'adapter pour détecter et localiser plusieurs objets, un réentrainement est nécessaire avec plus de classes.

1.4. Modèle « Movement »

Ce modèle consiste à reconnaître et classifier les mouvements (marche, marche rapide, chute, dispute, etc.) dans une scène filmée par la caméra (un exemple est accessible ici). Pour réaliser ce modèle, vous avez le choix entre :

- Utiliser une base de données publique (UCF, Kinetics, etc.) et tester avec les classes de votre choix ;
- Utiliser une base de données privée liée à un problème de reconnaissance d'actions.

Chaque groupe est libre de personnaliser (ou pas) le modèle de détection d'actions selon son souhait (par exemple : compter le nombre d'occurrence d'actions, reconnaître les visages après détection d'action dangereuse, etc.)

Note 01 : 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 reste **optionnel**. Chaque groupe peut aussi intégrer d'autres idées innovantes pour améliorer la solution tout en gardant l'objectif principal du projet.

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 et identification des classes pour les modèles « Face » et « Movement » ;
- **Partie 1 :** développement et entraînement des différents modèles ;
- **Partie 2 :** développement du programme de test (inférence + notification) à partir de séquences vidéo sur la ressource embarquée « Edge AI » : **Jetson Xavier** ;
- **Partie 3 :** optimisation/compression d'au moins un modèle ;
- **Partie 4 (facultative) :** analyse d'explicabilité de la solution « **XAI** ».

2.1. Partie 0 : choix du projet

Chaque groupe devra :

a. Choisir les modèles à développer (03 modèles au moins) ;

b. Identifier les classes à intégrer pour les modèles « Face » et « Movement » (si ce dernier est choisi) ;

c. Choisir la ou les méthodes de compression : Pruning, Quantization, Knowledge distillation, etc. ;

d. Marquer son souhait ou non d'appliquer les méthodes d'explicabilité « XAI ».

Choix à compléter ici pour le **lundi 17/04/2023** après concertation dans chaque groupe.

	Edge AI module for smart cities (Partie obligatoire)			Edge AI module for smart cities (Choisir un modèle au moins)		Optimisation & compression (Choisir au moins une méthode)			Explibabilté (facultatif) (Choisir le modèle à explique)	
	Modèle Face	Modèle Fire	Notifications (sms, email, etc.)	Modèle Suspect	Modèle Mouvement	Pruning	Quantization	Knowledge Distillation	Modèle Fire	Modèle Mouvement
Exemple	x	x	x		x	x		x	x	
Groupe 1	x	x	x							
Groupe 2	x	x	x							
Groupe 3	x	x	x							

2.2. Partie 1 : développement et entraînement des modèles

Après avoir choisi les modèles et options à développer, les entraînements peuvent être réalisés sous Python avec le Framework **Pytorch** (pour le Deep Learning) et la librairie **OpenCV** (pour le traitement d'images). Ces derniers sont déjà installés sur la carte Jetson Xavier qui sera fourni durant le Workshop. En termes de ressources, vous pouvez utiliser Google Colab Pro, vast.ai ou paperspace. Chaque groupe disposera, si besoin, de 50h de calcul avec vast.ai ou de 2 abonnements (1 mois) avec Colab Pro ou paperspace. Les principaux modèles à développer 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 entraîner le modèle avec une base de visages personnalisée (exemple : **visages des différents membres du groupe**). Cette partie peut être réalisée en quatre 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 13 classes (8 classes pour les membres du service ILIA et 5 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 facenet. Pour utiliser cet algorithme, il suffit d'appeler la fonction « **mtcnn** » reprise dans le notebook « face_recognition_torch.ipynb **».** Ce notebook présente l'ensemble des étapes nécessaires pour entraîner et sauvegarder le modèle de reconnaissance faciale. Notons que :
   - La fonction « mtcnn » : représente le programme de détection (crop) de visages ;
   - PERSONS : représente le dossier contenant les photos de personnes (voir plus haut) ;
   - PERSONS_CROPPED : 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 la 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* » reprise dans le même notebook pour entraîner ce modèle avec votre base de visages.

```
trans = transforms.Compose([transforms.ToTensor()])
dataset = datasets.ImageFolder(dst_dir, transform=trans)

face_model = InceptionResnetV1(
    classify=True,
    pretrained='vggface2',
    num_classes=len(dataset.classes)
).to(device)

optimizer = optim.Adam(face_model.parameters(), lr=0.001)
scheduler = MultiStepLR(optimizer, [5, 10])

trainloader = DataLoader(dataset,shuffle=True,batch_size=BATCH_SIZE)
criterion = torch.nn.CrossEntropyLoss()

face_model.train()
for epoch in range(EPOCHS):
  running_loss = 0.0
  counter = 0
  for (i, data) in enumerate(trainloader):
    inputs, labels = data
    inputs, labels = inputs.to(device), labels.to(device)
    # zero the parameter gradients
    optimizer.zero_grad()
    # forward + backward + optimize
    outputs = face_model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    # print statistics
    running_loss += loss.item()
    counter += 1
  print(f'train [{epoch + 1}, {i + 1:5d}] loss: {running_loss / counter:.3f}')

torch.save(face_model, 'facenet.pt')
```

Le résultat de cet apprentissage est représenté par le modèle « **facenet.pt** ». Si vous voulez lancer l'entraînement de nouveau, il faut remplacer le chargement du modèle pré-entraîné de google :

```
face_model = InceptionResnetV1(
    classify=True,
    pretrained='vggface2',
    num_classes=len(dataset.classes)
).to(device)
```

par le modèle qui a été sauvegardé sous le fichier « **facenet.pt** » :

```
face_model = torch.load('facenet.pt')
```

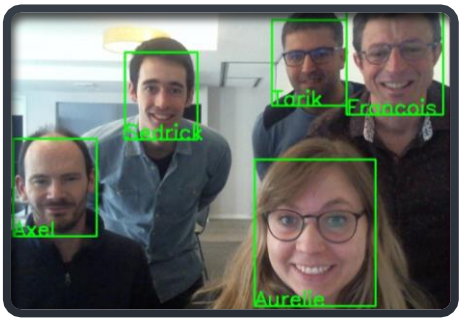


FIGURE 2 : EXEMPLE DE RECONNAISSANCE DE VISAGES

d. **Reconnaissance de visages en temps réel :** il faudra tester le modèle développé ci-dessus « **facenet.pt** » avec les images provenant, de la webcam ou d'une vidéo enregistrée. Vous pouvez utiliser le programme « face_recognition.py » qui prend en paramètre un modèle ainsi qu'une vidéo optionnelle. Par exemple, la commande ***python face_recognition.py facenet.pt*** permet de charger le modèle sauvegardé dans le fichier « **facenet.pt** » et d'identifier les personnes en face de la webcam. La commande ***python face_recognition.py facenet.pt video.mp4*** identifie les personnes du fichier vidéo « **video.mp4** » au lieu de la webcam.

2.2.2. Modèle « Fire »

Il s'agit du modèle de classification développé lors du défi 1, vous pouvez utiliser, adapter et améliorer vos modèles du défi 1 sans oublier de les porter **sous Pytorch** (Framework à utiliser durant le Workshop). Vous pouvez vous appuyer sur ce notebook **« fire_detection_training_torch.ipynb »** qu'il faudra bien évidemment adapter avec votre architecture, données et choix optimaux de paramètres.

2.2.3. Modèle « Object : Suspect »

De la même manière avec laquelle vous avez travaillé lors du défi 1, vous pouvez développer un modèle permettant de détecter la présence d'un ou plusieurs objets suspects (bâton, arme, couteau, etc.) dans une image. Il faudra 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-entraînés. Pour faciliter la tâche, nous partageons une base de données annotée, que vous pouvez augmenter/réduire, ainsi que les éléments nécessaires pour l'annotation si vous souhaitez annoter vous-même :

a. **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. 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 UN OBJET SUSPECT « DRONE »

Note 02 : un challenge en ligne « ***leaderboard*** » sera lancé durant le Workshop pour vous permettre d'évaluer vos modèles « ***face*** » et « ***suspect*** » avec des bases de données de test. Les liens seront fournis durant Workshop.

2.2.4. Modèle « Mouvement »

Ce modèle a pour objectif de reconnaître des mouvements (exp. marche, marche rapide, fuite, chute, 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 pourront servir à 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, CNN 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. 4) et projets de classification de vidéos existants. 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 4 : COMPARAISON ENTRE LES BASES DE VIDÉOS

b. **Catégories d'architectures « GitHub » de classification de vidéos (et modèles pré-entrainés) :** nous avons identifié quatre (04) catégories de réseaux de neurones profonds permettant la détection d'actions :

- **Approche convolutionnelle :** exploitant les convolutions 2D, 3D ainsi que les réseaux de neurones récurrents « RNN » afin de prendre en compte l'information spatiale et temporelle des actions : C3D, LRCN, R(2+1D), etc.
- **Approche Two-Stream :** proposant des réseaux de neurones composés de deux parties fusionnant leurs résultats respectifs pour la détection d'actions. Généralement, la première partie permet de prendre en compte la représentation visuelle des objets tandis que la deuxième partie permet de prendre en considération la représentation de mouvements : Two Stream, I3D, MARS, etc.
- **Approche temporelle :** permettant de prendre en compte l'information temporelle sans utiliser les réseaux de neurones récurrents ou les vecteurs de mouvements. Cette approche s'appuie sur l'extraction de changements à court, moyen et long terme entre les images (frames) successive dans une vidéo : TSN, TSM, SlowFast, etc.
- **Approche « Transformers » :** utilisant des réseaux « transformer » avec le mécanisme d'attention permettant de quantifier l'importance de de chaque partie des données d'entrée. Les transformes initialement proposés pour le traitement de langage (TLN) étaient étendus vers la vision par ordinateur (Vit, Visual Transformes) comme conséquence des résultats très promoteurs obtenus jusqu'au là avec transformes pour le TLN. Dans le contexte de reconnaissance d'actions, 02 principales architectures existent : R(2+1)D BERT et Timesformer. La figure 5 illustre dans un tableau la comparaison de précision entre les architectures citées ci-dessus avec la base de données publique UCF101.

Deep Learning methods	Convolutional	C3D
		LRCN
		R(2+1D)
	Two-Stream	Two Stream
		I3D
		MARS
	Temporal	TSN
		TSM
		SlowFast
	Transformers	R(2+1)D BERT
		Timesformer

FIGURE 5 : COMPARAISON DE PRÉCISION DES MODÈLES DEEP LEARNING DE RECONNAISSANCE D'ACTIONS AVEC LA BD UCF101

Nous partageons avec vous aussi des exemples de travaux d'étudiants (utilisant mmaction2) pour vous inspirer :

- Classification d'actions avec les architectures « **TSM** », « **TSN** » et « **I3D** » : voir ce notebook : « **video_classsification_mmaction.ipynb** »
- Classification d'actions avec l'architecture « **TSM** » sur la BD « Kinetics400 » : voir ce notebook : « **TSM_action_recognition_kinetics_400.ipynb** »
- Classification d'actions avec l'architecture « **Timesformer** » : voir ce notebook : « **TimeSformer_action_recognition.ipynb** »

Note 03 : le choix d'architecture de classification **devra** prendre en considération **la précision, le temps de calcul**, la **consommation mémoire** et l'**explicabilité** (si cette option est choisie). Notons que le modèle offrant la meilleure précision pour la BD « UCF101 » n'offrira pas forcément la meilleure précision pour les autres bases de données.

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 villes intelligentes en fonction de votre choix. La figure 6 illustre un exemple de programmes à développer et compléter (Edge AI System for Smart Cities)

```
#        ******************        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 6: "EDGE AI" ALGORITHM FOR SMART CITIES

FIGURE 7 : 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 : Pyorch, OpenCV, Sickit-learn, Matplotlib, 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 »

Afin d'avoir une solution optimale, nous vous proposons de l'optimiser en travaillant sur les points suivants :

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

2.4.2. Optimiser et compresser les modèles : à l'aide des techniques d'élagage « pruning » de réseaux de neurones, quantification, de distillation de connaissances et de 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 entre taille mémoire, temps de calcul et précision des modèles. Les réseaux compressés sont conçus pour avoir moins de paramètres et utiliser moins de mémoire. Pour faire le pruning, il faudra réfléchir à quatre éléments principaux :

- Comment élaguer « **How to prune** » ? : pour définir la stratégie d'élagage par couche [10] ;
- Ou « **Where to prune** » ? : pour définir le taux d'élagage par couche ;
- Quoi « **What to prune** » ? : en fonction de la magnitude, gradient, etc.
- Quand « **When to prune** » ? : durant ou après l'entrainement.

La figure 8 illustre des modes d'élagage permettant de répondre à la dernière question « When to prune ?».

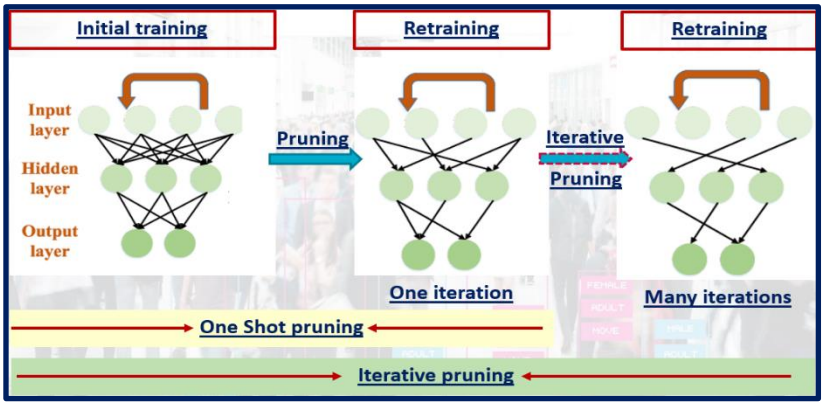


Figure 8 : illustration des modes d'élagage (pruning) de réseaux de neurones profonds [1]

b. La quantification : consiste à appliquer un processus d'approximation sur les réseaux de neurones afin de représenter les poids avec des nombres 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 9 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.

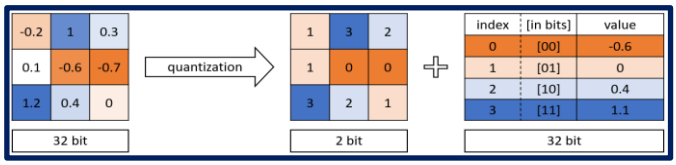


Figure 9 : illustration du principe de quantification

c. **Distillation de connaissances « Knowledge distillation » :** consiste à développer un petit réseau de neurones « **Student** » qui pourra apprendre durant l'entrainement à partir d'un grand réseau de neurones « **Teacher** ». L'objectif est d'avoir un réseau de neurones simplifié mais qui prendra des décisions semblables aux décisions d'un réseau de neurones grand et complexe. La figure 12 illustre le processus de cette approche.

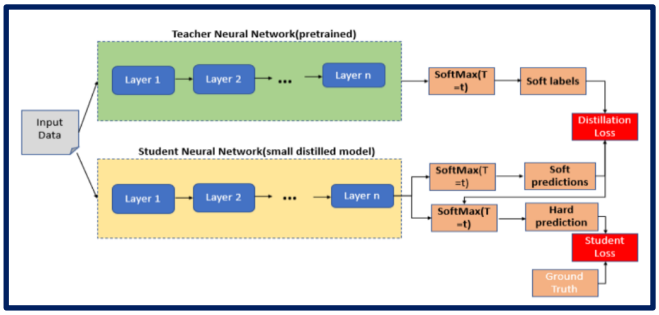


FIGURE 10 : PROCESSUS DE LA MÉTHODE DE DISTILLATION DE CONNAISSANCES

d. **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 10). 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 « **MobileNet** » connue d'ailleurs par un faible temps de calcul et un espace mémoire réduit.

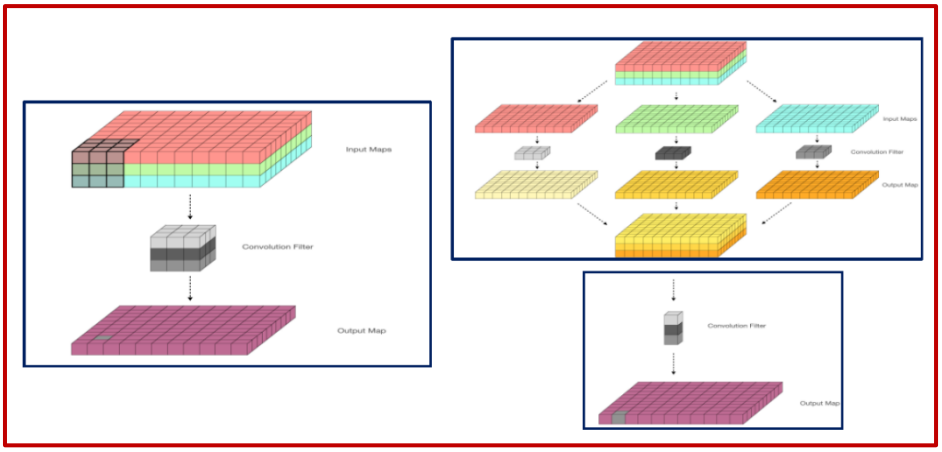


Figure 10 : convolution classique vs. Convolutions séparables

Nous partageons avec vous quelques liens vers des tutoriels pouvant vous aider pour la compression :

- **Pruning :** lien → mode d'application du pruning sur un DNN développé avec Pytorch.
- **Knowledge Distillation :** lien → exemple d'application de la KD sur un DNN développé sous Pytorch.
- **Quantization :** lien → mode d'application de la quantization sous Pytorch.

2.4.3. Facultatif : 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 [12]. Dans ce contexte, une nouvelle méthode « RISE, Randomized Input Sampling for Explantation of Black-Box Models » [13]. Cette dernière calcule des masques basés sur un suréchantillonnage de masques binaires remplis de manière aléatoire et crée une carte thermique basée sur la pertinence de chaque masque pour la prédiction. Ces méthodes sont plus lourdes en temps de calcul que la plupart des méthodes post-hoc car elles nécessitent de remplacer les valeurs d'entrée et de calculer le résultat des modèles à chaque itération.

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 méthodes CAM : les méthodes « CAM » sont appliquées aux réseaux de neurones convolutifs et ont la particularité d'exploiter les couches d'activation (généralement les dernières couches convolutives) pour produire des cartes de saillance. La mtéhode Grad-CAM [14] est la plus largement utilisée et pondère les cartes d'activation par les gradients obtenus par une rétropropagation à partir du neurone de sortie d'une classe choisie à la couche convolutive. Il existe de nombreuses variantes qui combinent des activations de différentes couches (Layer-CAM [15], Poly-CAM [16]), agréger les résultats pour l'entrée image à différentes échelles (CAMERAS [17]), ou utiliser les activations comme masques pour prédire leur importance (Score-CAM [18]).

d. Explicabilité basée sur les modèles Proxy : permettent de réduire la complexité de modèles Machine et Deep Learning afin d'avoir 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.

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 détection d'objets avant de les embarquer dans un module « ***Edge AI*** » pour 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 (ou proche) à partir d'images capturées via la webcam. A l'issue de cela, vous pourrez intégrer les autres tâches choisies (compression, explicabilité, intégration d'autres modèles, etc.) et mettre en œuvre l'envoi d'alertes et notifications. Vous avez aussi la possibilité d'intégrer d'autres options imaginées par vos soins. Nous vous proposons de quantifier les résultats de vos modèles choisis et développés en complétant le tableau 1.

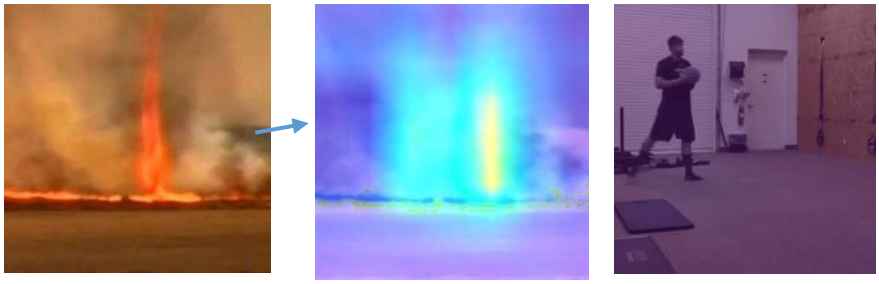
Note : il est possible de modifier le tableau en fonction de vos choix d'options (par exemple, une colonne d'évaluation de la méthode d'explicabilité pour ceux qui choisissent l'option « **XAI** »)

	Taille de frames	Précision (Acc)	Taille du modèle .pt (MB)	FPS	Classement LeaderBoard
Modèle « Face »					-------
Modèle « Fire »					
Modèle « Suspect »					
Modèle « Movement »					------

TABLEAU 1 : EVALUATION DES MODÈLES EN TERMES DE PRÉCISION, TEMPS DE CALCUL ET ESPACE MÉMOIRE

4. Quelques exemples :

Localisation de feu



Détection explicable de feu et d'action « Trowing ball »

Bibliographie

[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.

[9] Jianping Gou, Baosheng Yu, Stephen J. Maybank Dacheng Tao, "Konowledge Distillation: A survey" . Vol 129, pp 1789-1819, 2021.

[10] Sajid Anwar, Kyuyeon Hwang, Wonyong Sung, "Structured Pruning of Deep Convolutional Neural Networks", *ACM Journal on Emerging Technologies in Computing Systems,* Vol. 13, Issue 3, July 2017 Article No.: 32pp 1–18. https://doi.org/10.1145/3005348

[11] J. -H. Luo, H. Zhang, H. -Y. Zhou, C. -W. Xie, J. Wu and W. Lin, "ThiNet: Pruning CNN Filters for a Thinner Net," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 10, pp. 2525-2538, 1 Oct. 2019, doi: 10.1109/TPAMI.2018.2858232.

[12] Zeiler, M.D.; Fergus, R. "Visualizing and understanding convolutional networks". *In Proceedings of the European conference on computer vision. Springer,* 2014, pp. 818–833.

[13] Petsiuk, V.; Das, A.; Saenko, K. "RISE: Randomized Input Sampling for Explanation of Black-box Models". *In Proceedings of the BMC*, 2018.

[14] Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. "Grad-cam: Visual explanations from deep networks via gradient-based localization". *In Proceedings of the ICCV*, 2017.

[15] Jiang, P.T.; Zhang, C.B.; Hou, Q.; Cheng, M.M.; Wei, Y. "LayerCAM: Exploring hierarchical class activation maps for localization". *IEEE Trans. Image Process. 2021*, 30, 5875–5888.

[16] Englebert, A.; Cornu, O.; Vleeschouwer, C.D. "Poly-CAM: High resolution class activation map for convolutional neural networks". *arXiv:2204.13359v2* 2022.

[17] Jalwana, M.A.; Akhtar, N.; Bennamoun, M.; Mian, A. "CAMERAS: Enhanced resolution and sanity preserving class activation mapping for image saliency". *In Proceedings of the CVPR*, 2021.

[78] Wang, H.; Wang, Z.; Du, M.; Yang, F.; Zhang, Z.; Ding, S.; Mardziel, P.; Hu, "X. Score-CAM: Score-weighted visual explanations for convolutional neural networks". *In Proceedings of the CVPR Worshop on TCV, 2020.*