كلية العلوم ببنزرت
Faculté des Sciences de Bizerte

2021

Lunette Amine

Réalisée par :
Marwa Lamouri
Farah Ben Hassen
Omar Killbi
Mohamed Amine Fetni

Projet Fédérateur

Département informatique

01/01/2021

I. Table des matières

Liste de figures

Introduction générale

Depuis quelques années, un nouveau lexique lié à l'émergence de l'intelligence artificielle dans notre société inonde les articles scientifiques, et il est parfois difficile de comprendre de quoi il s'agit. Lorsqu'on parle d'intelligence artificielle, on fait très souvent l'allusion aux technologies associées comme le Machine learning ou le Deep learning. Deux termes extrêmement utilisés avec des applications toujours plus nombreuses, mais pas toujours bien définis.

Le Machine Learning est un ensemble de techniques donnant la capacité aux machines d'apprendre, contrairement à la programmation qui consiste en l'exécution de règles prédéterminées. Il existe deux principaux types d'apprentissages en Machine Learning. L'apprentissage supervisé et non supervisé.

Le Deep learning ou apprentissage profond est l'une des technologies principales du Machine learning. Avec le Deep Learning, nous parlons d'algorithmes capables de mimer les actions du cerveau humain grâce à des réseaux de neurones artificielles. Les réseaux sont composés de dizaines voire de centaines de « couches » de neurones, chacune recevant et interprétant les informations de la couche précédente.

Les modèles de Deep learning ont tendance à bien fonctionner avec une grande quantité de données alors que les modèles d'apprentissage automatique plus classique cessent de s'améliorer après un point de saturation. Au fil des années, avec l'émergence du big data et de composants informatiques de plus en plus puissant, les algorithmes de Deep Learning gourmands en puissance et en données ont dépassé la plupart des autres méthodes. Ils semblent être prêt à résoudre bien des problèmes : Reconnaître des visages, vaincre des joueurs de go ou de poker, permettre la conduite de voitures autonomes ou encore la recherche de cellules cancéreuses.

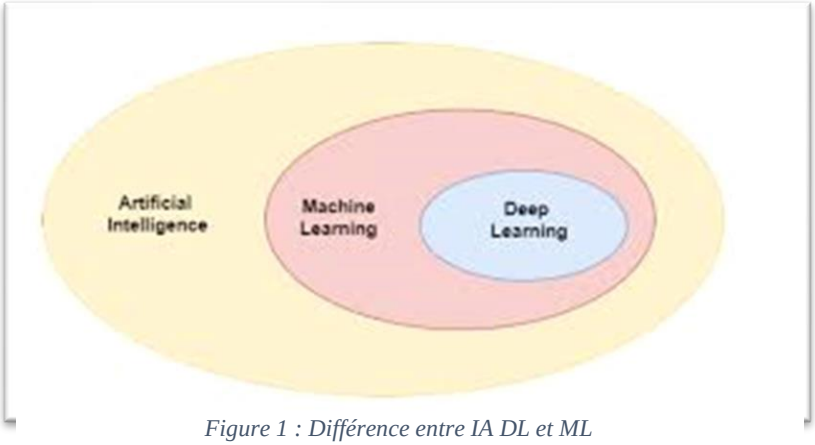


Figure 1 : Différence entre IA DL et ML

I. Détection des objets

1. Introduction :

Récemment, le domaine de l'intelligence artificielle a connu de nombreuses avancées grâce au deep learning et au traitement des images. Il est maintenant possible de reconnaître des images ou même de trouver des objets à l'intérieur d'une image ou d'un vidéo. Le traitement vidéo est devenu un outil indispensable pour de nombreuses applications précisément celles liées à la vision par ordinateur et la robotique. Le traitement basé sur la notion dite 'vision' remplace de plus en plus des techniques, longtemps utilisées. A titre d'exemples, citons les travaux liés à l'analyse routière où le traitement vidéo remplace l'ancienne technique connue par la boucle inductive. Ces détecteurs vidéo sont adoptés à cause de leur installation, opération, et maintenance simple, d'une part. D'autre part, ils permettent d'observer, surveiller et d'analyser des champs larges. La détection est également importante pour le traitement vidéo, aussi que pour la compression. Elle fournit les informations concernant la relation spatio-temporelle entre les objets dans l'image. Ces informations peuvent être exploitées dans des applications diverses tel que la télésurveillance, le suivi d'un objet bien défini, le traitement routier, …etc.

2. Problématique :

La déficience visuelle peut être dans certains moments de la vie une cause de souffrance. La personne handicapée ressent dès lors un besoin de parler de ce qui lui arrive et de confronter sa situation à celles rencontrées par d'autres personnes souffrant des mêmes problèmes visuels.

Une déficience visuelle peut conduire à une perte d'autonomie, de déplacement ou d'utilisation de moyens de communication dans la vie de tous les jours et beaucoup des accidents. La personne malvoyant ou aveugle se sent donc très souvent mise de côté, abandonnée, suite à cette exclusion, même si celle-ci ne se limite qu'à certaines activités sociales.

Dans le monde selon les estimations de France 24 :

- 253 millions de personnes présentent une déficience visuelle : 36 millions d'entre elles sont aveugles et 217 millions présentent une déficience visuelle modérée à sévère.
- 81% des aveugles ou des personnes qui présentent une déficience visuelle modérée ou sévère sont âgés de 50 ans et plus.

Les accidents auxquels ils sont exposés :

- 43% des accidents dans le monde causé par les aveugles ou des personnes qui présentent une déficience visuelle.

3. Solution :

D’après les statistiques que nous avons fournies dans la problématique, nous avons décidé de proposer une solution pour réduire les accidents et pour faciliter les déplacements des personnes malvoyantes.

Notre projet intitulé « Lunette-Amine » consiste à d’aider les malvoyants et d’assister les aveugles dans leurs déplacements et de les informer sur leur environnement afin de faciliter leur quotidien.

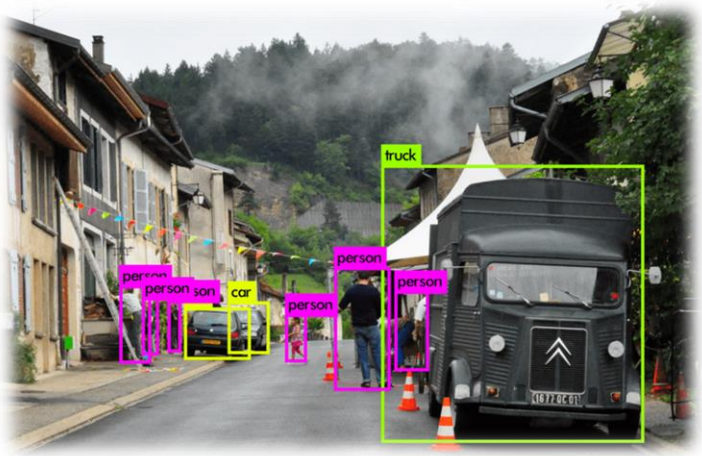


Figure 2 : Exemple de détection des objets

4. Conclusion :

Lunette –Amine pourraient bien changer la vie de millions de personnes aveugles.

Notre projet s’appuie sur la réalité virtuelle pour rendre au moins partiellement la vue aux aveugles.

L’idée est à terme de remplacer les cannes ou les chiens d’aveugles. Ceux-ci vous avertissent de ce qu’il y a juste devant vous mais vous fournir une image générale de votre environnement.

Lunette Amine s’appuient sur des technologies destinée à comprendre l’environnement en réalité virtuelle.

Tester et choisir un algorithme de détection d’objet(Deep learning) répond à notre besoin c’est notre première étape.

II. Etude des algorithmes des détections des objets

1. Introduction :

On ne peut pas parler de Deep Learning sans mentionner Alexnet. En effet, c'est l'un des pionniers du Deep Neural Net qui vise à classifier les images. Les algorithmes de détection d'objets s'améliorent continuellement avec les réseaux neuronaux profonds de classification d'images sur lesquels ils sont basés. Cependant, la détection d'objets reste un domaine de recherche actif et de nouveaux algorithmes sont également créés régulièrement.

Dans ce chapitre nous discutons les différents algorithmes de détections de objets avec détails et étude comparatif**.**

2. Choix de base des données :

Nous ne pouvons pas utiliser une dataset existante à cause de sa grande taille (plus de 32GO), donc nous allons créer notre propre dataset à partir des images téléchargées sur internet et en va tester l'efficacité de l'algorithme qui va être choisis de détection des objets à l'aide du cette base.

3. R-CNN :

a) Définition :

R-CNN a été le premier algorithme à appliquer le deep learning à la tâche de détection d'objets. Il bat les précédents de plus de 30 % par rapport au VOC2012 (Visual Object Classes Challenge) et constitue donc une amélioration considérable dans les domaines de la détection d'objets.
Comme mentionné précédemment, la détection d'objets présente deux difficultés : trouver des objets et les classer. C'est le but de R-CNN : diviser la tâche difficile de la détection d'objets en deux tâches plus faciles :

- Objects Proposal (trouver des objets)
- Region Classification (les comprendre)

La tâche d'Object Proposal est un domaine de recherche actif et, en 2013, plusieurs algorithmes étaient déjà performants. Nous ne détaillerons pas cette tâche ici, mais il est bon de savoir qu'il existe deux grandes familles d'algorithmes :

- Ceux qui regroupent les super-pixels (Selective Search, CPMC, MCG, …)
- Ceux basés sur une fenêtre coulissante (EdgeBoxes, …)

b) Principe :

R-CNN est en fait indépendant de l'algorithme d'Objects Proposal et peut utiliser n'importe laquelle de ces méthodes.
R-CNN prend en entrée les Regions Proposal (ou d'objets ou de boîtes). La plupart des algorithmes Regions Proposal produisent un grand nombre de régions (environ 2000 pour une image standard) et l'objectif de R-CNN est de trouver quelles régions sont significatives et les objets qu'elles représentent.

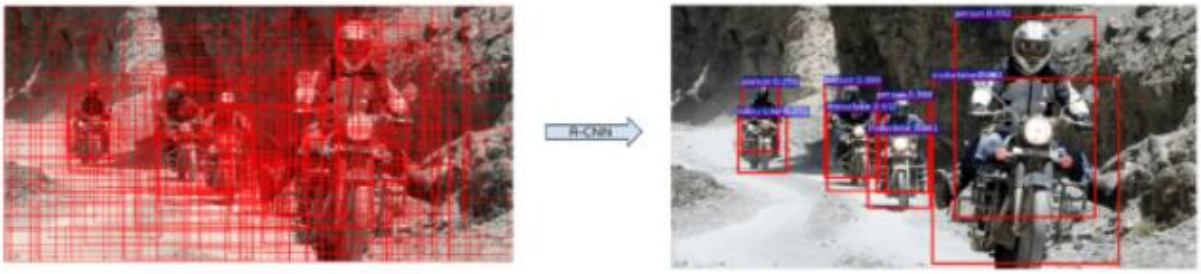


Figure 3 : R-CNN

R-CNN n'utilise pas directement un Alexnet sur toutes les propositions de régions car en plus de classifier une image, le modèle devrait pouvoir corriger l'emplacement d'une proposition de région si elle n'est pas correcte. Cependant, si vous vous souvenez d'Alexnet, nous avons vu que les couches entièrement connectées après les convolutions pouvaient en fait être remplacées par n'importe quel autre classificateur. C'est exactement ce qui est fait dans R-CNN. La partie convolutionnelle d'Alexnet est utilisée pour calculer les caractéristiques de chaque région, puis les SVM utilisent ces caractéristiques pour classer les régions. L'avantage de cette méthode est que le réseau neuronal (Alexnet) est déjà formé sur un énorme ensemble de données d'images et est très puissant pour concevoir les propositions de régions.

Avant cette étape de classification SVM, le réseau neuronal est affiné pour prendre en compte une nouvelle classe « fond », afin de distinguer les régions avec ou sans objets.
Les propositions de régions, qui sont des rectangles de différentes formes possibles, sont transformées en carrés de 227×227 pixels, la taille d'entrée requise par Alexnet. Elles sont ensuite traitées par le réseau et les valeurs obtenues sur la dernière carte de caractéristiques sont sorties. La région est alors devenue 4096 vecteurs d'entités.

Ces vecteurs d'entités codent les informations des images d'une manière bien plus efficace pour traiter la classification.

Ensuite, une stratégie de SVM à un seul repos est appliquée sur tous les vecteurs de régions. C'est-à-dire que si le modèle est entraîné à reconnaître 100 classes, alors 100 SVM binaires seront traités sur chaque région. En conservant le meilleur score parmi tous les classificateurs binaires, nous obtenons la classe d'objet détectée correspondante (ou le fond en fait). Même si les classificateurs sont tous capables de reconnaître une seule classe parmi les autres, les résultats sont bons car les caractéristiques extraites d'Alexnet sont partagées par toutes les classes.

Il y a ensuite une étape de régression des boîtes englobantes afin de corriger la localisation des propositions de régions qui n'étaient pas bonnes, par exemple si la boîte n'est pas bien centrée sur l'objet ou pas du bon rapport. Cette phase de régression produit des facteurs de correction aux coordonnées de la boîte englobante. Pour cette tâche, des régresseurs linéaires spécifiques à la classe sont formés sur les cartes de caractéristiques pour prédire les boîtes limites de la vérité de terrain.
Finalement, il existe un mécanisme permettant de ne garder que les meilleures régions. Si une région chevauche une autre de la même classe avec plus d'un certain pourcentage (environ 30% fonctionne assez bien), seule la région la mieux notée est conservée. Cela permet de conserver un nombre assez raisonnable de régions.

c) Inconvénient :

Cet algorithme est puissant et fonctionne très bien, mais il présente plusieurs inconvénients :

Il est très long. La proposition de région prend de 0,2 à plusieurs secondes selon la méthode, puis l'extraction et la classification des caractéristiques prennent à nouveau plusieurs secondes. Elle peut aller jusqu'à une minute sur un processeur pour une image.
Ce n'est pas un algorithme fluide. Il y a trois étapes différentes qui sont presque indépendantes et qui nécessitent donc une formation séparée.

4. YOLO V3 :

a) Définition :

YOLO (You only look once) est un système de détection d'objets en temps réel à la pointe de la technologie. Il s'agit d'un algorithme de reconnaissance et de localisation d'objets basé sur un réseau neuronal profond. Sa plus grande caractéristique est qu'il fonctionne très vite. Par exemple, si vous entrez une

image, le système affichera les objets qu'il contient et la position de chaque objet (le cadre rectangulaire contenant l'objet).

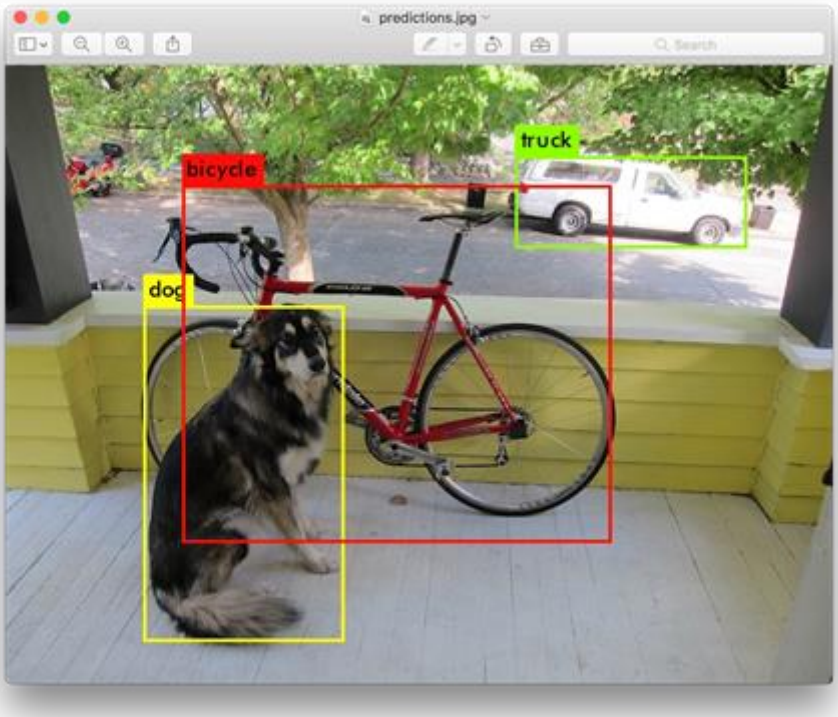


Figure 4: Exemple de détection d'objets

L'adresse du site YOLOv3 : https://pjreddie.com/darknet/yolo/

YOLOv3: une amélioration incrémentale

À 320 × 320, YOLOv3 fonctionne en 22 ms à 28,2 mAP[1], aussi précis qu'un SSD mais trois fois plus rapide. Quand nous regardons l'ancienne métrique de détection .5 IOU[2] mAP YOLOv3 est assez bonne. Il atteint 57,9 AP50 en 51 ms sur un Titan X, contre 57,5 AP50 en 198 ms par RetinaNet, des performances similaires mais 3,8 fois plus rapides.

b) Principe :

- **Prédiction de la boîte englobante (Bounding Box Prediction)**

YOLOv3 prédit un score d'objectivité pour chaque boîte englobante à l'aide de la régression logistique. Cela devrait être 1 si la boîte englobante antérieure chevauche un objet de vérité terrain de plus que toute autre boîte englobante

[1] Mean Average Precision: AP (Average precision) is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1.

[2] Intersection Over Union: IoU measures the overlap between 2 boundaries. We use that to measure how much our predicted boundary overlaps with the ground truth (the real object boundary). In some datasets, we predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.

antérieure. Si la boîte englobante antérieure n'est pas la meilleure mais chevauche un objet de vérité terrain de plus d'un certain seuil, nous ignorons la prédiction.

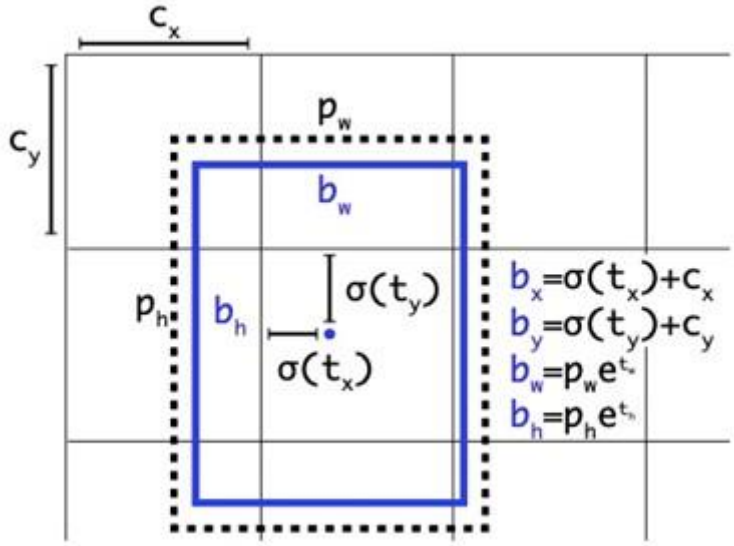


Figure 5 : Principe de la boîte englobante

Boîtes englobantes avec priors de dimension et prédiction d'emplacement. Nous prédisons la largeur et la hauteur de la boîte sous forme de décalages par rapport aux centres de gravité du cluster. Nous prédisons les coordonnées du centre de la boîte par rapport à l'emplacement de l'application du filtre à l'aide d'une fonction sigmoïde.

- **Prédiction de classe (Class Prediction)**

Chaque boîte prédit les classes que la boîte englobante peut contenir en utilisant une classification à étiquettes multiples. Nous n'utilisons pas de softmax, nous utilisons simplement des classificateurs logistiques indépendants. Nous utilisons la perte d'entropie croisée binaire pour les prédictions de classe.

- **Prédictions à toutes les échelles (Predictions Across Scales)**

YOLOv3 prédit des boîtes à 3 échelles différentes. Nous utilisons toujours le clustering k-means pour déterminer nos a priori de boîte englobante. Nous avons simplement choisi en quelque sorte 9 grappes et 3 échelles de manière arbitraire, puis divisons les grappes uniformément sur les échelles.

- **Extracteur de fonctionnalités (Feature Extractor)**

Darknet-53

Le réseau utilise des couches convolutives successives 3 × 3 et 1 × 1, mais dispose désormais également de connexions de raccourci et est nettement plus grand. Il a 53 couches convolutives.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Figure 6 : Structure du réseau Darknet

c) Inconvénient :

Le YOLOv3 indique un compromis entre la vitesse et la précision pour l'utilisation de YOLO par rapport à RetinaNet puisque le temps d'entraînement de RetinaNet est supérieur à YOLOv3. Cependant, la précision de la détection d'objets avec YOLOv3 peut être égale à la précision lors de l'utilisation de RetinaNet en ayant un ensemble de données plus important, ce qui en fait une option idéale pour les modèles pouvant être entraînés avec de grands ensembles de données.

5. YOLO V4 :

a) Définition :

YOLOv4 utilise de nouvelles fonctionnalités : WRC, CSP, CmBN, SAT, activation Mish, augmentation des données Mosaic, CmBN, régularisation DropBlock et perte CIoU, et combine certaines d'entre elles pour obtenir des

résultats de pointe : 43,5% AP (65,7% AP50) pour l'ensemble de données MS COCO à une vitesse en temps réel de ~65 FPS sur Tesla V100.

b) Principe :

➢ **Modèles de détection d'objets**

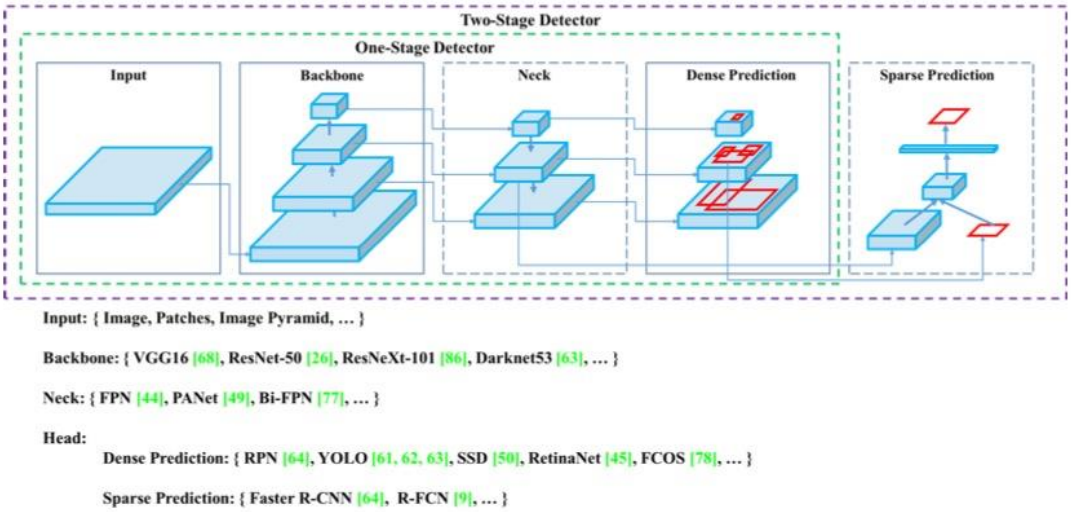


Figure 7 : Modèle de détection d'objets

➢ **Bag of freebies**

Augmentation des données : ajustements par pixel, simulation de problèmes d'occlusion d'objets, plusieurs images ensemble.

Résoudre le problème que la distribution sémantique dans l'ensemble de données peut avoir un biais.

Exprimer la relation du degré d'association entre différentes catégories avec la représentation dure à chaud : label smoothing.

➢ **Bag of specials**

Modules de plugin :

Elargissement du champ récepteur : SPP, ASPP, RFB

Introduction d'un mécanisme d'attention : attention par canal, attention ponctuelle

Intégration de fonctionnalités : SFAM, ASFF, BiFPN

Bonne fonction d'activation

Méthodes de post-traitement : NMS, soft NMS, DIoU NMS

Table 1: Parameters of neural networks for image classification.

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	**1058 K**	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	**27.6 M**	950 K	52 (26.0 FMA)	**66**
EfficientNet-B3 (ours)	512x512	**1311x1311**	12.0 M	668 K	11 (5.5 FMA)	26

- **Méthodologie**

Deux options de réseaux de neurones en temps réel :

• Pour le GPU, nous utilisons un petit nombre de groupes (1 - 8) dans les couches convolutives : CSPResNeXt50 / CSPDarknet53

• Pour VPU - nous utilisons la convolution groupée, mais nous nous abstenons d'utiliser des blocs Squeeze-and-excitation (SE) - en particulier, cela inclut les modèles suivants : EfficientNet-lite / MixNet / GhostNet / Mo- bileNetV3

- **Sélection d'architecture**

Nous choisissons le backbone CSPDarknet53, le module supplémentaire SPP, le col d'agrégation de chemin PANet et la tête YOLOv3 (à base d'ancre) comme architecture de YOLOv4.

- **Sélection de BoF et BoS**

Nous n'avons pas hésité à choisir DropBlock comme méthode de régularisation. En ce qui concerne le choix de la méthode de normalisation, puisque nous nous concentrons sur une stratégie de formation qui n'utilise qu'un seul GPU, syncBN n'est pas pris en compte.

- **Améliorations supplémentaires**

Afin de rendre le détecteur conçu plus adapté à la formation sur un seul GPU :

- Nous introduisons une nouvelle méthode d'augmentation des données de formation mosaïque et auto-adversaire (SAT)
- Nous sélectionnons des hyper-paramètres optimaux tout en appliquant des algorithmes génétiques
- Nous modifions certaines méthodes existantes pour rendre notre conception adaptée à un entraînement et une détection efficaces - SAM modifié, PAN modifié et normalisation en mini-lots croisés (CmBN)

c) Inconvénient :

- Difficultés à détecter les objets proches car chaque grille ne peut proposer que 2 cadres de délimitation.
- Lutte pour détecter les petits objets.

III. Analyse du Code

a) Introduction :

Dans ce chapitre, on va analyser les différentes parties du code de l'algorithme utilisé et qui a été testé sur Google Colab pour avoir des performances acceptables .

- Importation des bibliothèques :

```
# import dependencies
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from google.colab.patches import cv2_imshow
from base64 import b64decode, b64encode
import cv2
import numpy as np
import PIL
import io
import html
import time
import matplotlib.pyplot as plt
```

On va utiliser divers bibliothèques comme
numpy, matplotlib pour la manipulation des données.
cv2 , PIL pour le traitement des images.
Base64 pour codage binaire des images.

- Préparation de l'environnement :

```
# clone darknet repo
!git clone https://github.com/AlexeyAB/darknet
```

```
Cloning into 'darknet'...
remote: Enumerating objects: 15376, done.
remote: Total 15376 (delta 0), reused 0 (delta 0), pack-reused 15376
Receiving objects: 100% (15376/15376), 14.01 MiB | 8.21 MiB/s, done.
Resolving deltas: 100% (10339/10339), done.
```

```
# change makefile to have GPU, OPENCV and LIBSO enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

Dans cette partie, on a fait appel au réseau de neurone darknet qui nous permet de démarrer YOLOv4 avec les poids personnalisés.

Pour pouvoir utiliser YOLOv4, on a eu recours à des fonctions prédéfinies de darknet

```
# import darknet functions to perform object detections
from darknet import *
# load in our YOLOv4 architecture network
network, class_names, class_colors = load_network("cfg/yolov4-csp.cfg", "cfg/coco.data", "yolov4-csp.weights")
width = network_width(network)
height = network_height(network)

# darknet helper function to run detection on image
def darknet_helper(img, width, height):
  darknet_image = make_image(width, height, 3)
  img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  img_resized = cv2.resize(img_rgb, (width, height),
                           interpolation=cv2.INTER_LINEAR)

  # get image ratios to convert bounding boxes to proper size
  img_height, img_width, _ = img.shape
  width_ratio = img_width/width
  height_ratio = img_height/height

  # run model on darknet style image to get detections
  copy_image_from_bytes(darknet_image, img_resized.tobytes())
  detections = detect_image(network, class_names, darknet_image)
  free_image(darknet_image)
  return detections, width_ratio, height_ratio
```

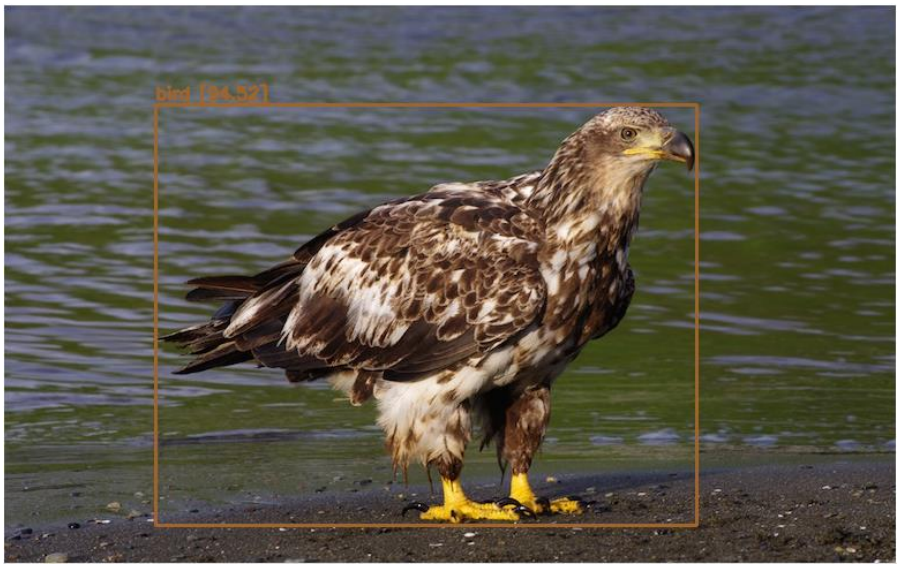
- Phase de test :

Après le test du modèle avec la fonction suivante :

```
# run test on person.jpg image that comes with repository
image = cv2.imread("data/eagle.jpg")
detections, width_ratio, height_ratio = darknet_helper(image, width, height)

for label, confidence, bbox in detections:
  left, top, right, bottom = bbox2points(bbox)
  left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int(right * width_ratio), int(bottom * height_ratio)
  cv2.rectangle(image, (left, top), (right, bottom), class_colors[label], 2)
  cv2.putText(image, "{} [{:.2f}]".format(label, float(confidence)),
                    (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    class_colors[label], 2)
cv2_imshow(image)
```

On a obtenu le résultat suivant :



Cette fonction nous aide à convertir des objet JavaScript en des images :

```
# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
  """
  Params:
          js_reply: JavaScript object containing image from webcam
  Returns:
          img: OpenCV BGR image
  """
  # decode base64 image
  image_bytes = b64decode(js_reply.split(',')[1])
  # convert bytes to numpy array
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
  # decode numpy array into OpenCV BGR image
  img = cv2.imdecode(jpg_as_np, flags=1)

  return img

# function to convert OpenCV Rectangle bounding box image into base64 byte string to be overlayed on video stream
def bbox_to_bytes(bbox_array):
  """
  Params:
          bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
  Returns:
        bytes: Base64 image byte string
  """
  # convert array into PIL image
  bbox_PIL = PIL.Image.fromarray(bbox_array, 'RGBA')
  iobuf = io.BytesIO()
  # format bbox into png for return
  bbox_PIL.save(iobuf, format='png')
  # format return string
  bbox_bytes = '{}'.format((str(b64encode(iobuf.getvalue()), 'utf-8')))

  return bbox_bytes
```

➢ YOLOv4 on Webcam Images

Dans cette partie, on a utilisé des fonctionnalités du google colab pour pouvoir accéder à la webcam et tester le modèle :

```
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
```

```
# get photo data
data = eval_js('takePhoto({})'.format(quality))
# get OpenCV format image
img = js_to_image(data)

# call our darknet helper on webcam image
detections, width_ratio, height_ratio = darknet_helper(img, width, height)

# loop through detections and draw them on webcam image
for label, confidence, bbox in detections:
  left, top, right, bottom = bbox2points(bbox)
  left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int(right * width_ratio), int(bottom * height_ratio)
  cv2.rectangle(img, (left, top), (right, bottom), class_colors[label], 2)
  cv2.putText(img, "{} [{:.2f}]".format(label, float(confidence)),
                  (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                  class_colors[label], 2)
# save image
cv2.imwrite(filename, img)

return filename
```

Et cette fonction permet de faire cette tâche :

```
try:
  filename = take_photo('photo.jpg')
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))
```

➢ YOLOv4 on Webcam Videos

Maintenant, on va créer une sorte de stream en utilisant la webcam comme input avec cette fonction :

```
# JavaScript to properly create our live video stream using our webcam as input
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;

    var pendingResolve = null;
    var shutdown = false;

    function removeDom() {
       stream.getVideoTracks()[0].stop();
       video.remove();
       div.remove();
       video = null;
       div = null;
       stream = null;
       imgElement = null;
       captureCanvas = null;
       labelElement = null;
    }

    function onAnimationFrame() {
      if (!shutdown) {
        window.requestAnimationFrame(onAnimationFrame);
      }
      if (pendingResolve) {
        var result = "";
        if (!shutdown) {
          captureCanvas.getContext('2d').drawImage(video, 0, 0, 640, 480);
          result = captureCanvas.toDataURL('image/jpeg', 0.8)
        }
        var lp = pendingResolve;
        pendingResolve = null;
        lp(result);
      }
```

```
}

async function createDom() {
  if (div !== null) {
    return stream;
  }

  div = document.createElement('div');
  div.style.border = '2px solid black';
  div.style.padding = '3px';
  div.style.width = '100%';
  div.style.maxWidth = '600px';
  document.body.appendChild(div);

  const modelOut = document.createElement('div');
  modelOut.innerHTML = "<span>Status:</span>";
  labelElement = document.createElement('span');
  labelElement.innerText = 'No data';
  labelElement.style.fontWeight = 'bold';
  modelOut.appendChild(labelElement);
  div.appendChild(modelOut);

  video = document.createElement('video');
  video.style.display = 'block';
  video.width = div.clientWidth - 6;
  video.setAttribute('playsinline', '');
  video.onclick = () => { shutdown = true; };
  stream = await navigator.mediaDevices.getUserMedia(
      {video: { facingMode: "environment"}});
  div.appendChild(video);

  imgElement = document.createElement('img');
  imgElement.style.position = 'absolute';
  imgElement.style.zIndex = 1;
  imgElement.onclick = () => { shutdown = true; };
  div.appendChild(imgElement);

  const instruction = document.createElement('div');
  instruction.innerHTML =
      '<span style="color: red; font-weight: bold;">' +
      'When finished, click here or on the video to stop this demo</span>';
  div.appendChild(instruction);
```

```
      instruction.onclick = () => { shutdown = true; };

      video.srcObject = stream;
      await video.play();

      captureCanvas = document.createElement('canvas');
      captureCanvas.width = 640; //video.videoWidth;
      captureCanvas.height = 480; //video.videoHeight;
      window.requestAnimationFrame(onAnimationFrame);

      return stream;
    }
    async function stream_frame(label, imgData) {
      if (shutdown) {
        removeDom();
        shutdown = false;
        return '';
      }

      var preCreate = Date.now();
      stream = await createDom();

      var preShow = Date.now();
      if (label != "") {
        labelElement.innerHTML = label;
      }

      if (imgData != "") {
        var videoRect = video.getClientRects()[0];
        imgElement.style.top = videoRect.top + "px";
        imgElement.style.left = videoRect.left + "px";
        imgElement.style.width = videoRect.width + "px";
        imgElement.style.height = videoRect.height + "px";
        imgElement.src = imgData;
      }

      var preCapture = Date.now();
      var result = await new Promise(function(resolve, reject) {
        pendingResolve = resolve;
      });
      shutdown = false;

      return {'create': preShow - preCreate,
              'show': preCapture - preShow,
              'capture': Date.now() - preCapture,
              'img': result};
    }
    ''')

  display(js)

def video_frame(label, bbox):
  data = eval_js('stream_frame("{}", "{}")'.format(label, bbox))
  return data
```

Et enfin , la fonction suivante va permettre de tester YOLOv4 sur un vidéo :

```
# start streaming video from webcam
video_stream()
# label for video
label_html = 'Capturing...'
# initialze bounding box to empty
bbox = ''
count = 0
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # convert JS response to OpenCV Image
    frame = js_to_image(js_reply["img"])

    # create transparent overlay for bounding box
    bbox_array = np.zeros([480,640,4], dtype=np.uint8)

    # call our darknet helper on video frame
    detections, width_ratio, height_ratio = darknet_helper(frame, width, height)

    # loop through detections and draw them on transparent overlay image
    for label, confidence, bbox in detections:
      left, top, right, bottom = bbox2points(bbox)
      left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int(right * width_ratio), int(bottom * height_ratio)
      bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bottom), class_colors[label], 2)
      bbox_array = cv2.putText(bbox_array, "{} [{:.2f}]".format(label, float(confidence)),
                       (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                       class_colors[label], 2)

    bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype(int)*255
    # convert overlay of bbox into bytes
    bbox_bytes = bbox_to_bytes(bbox_array)
    # update bbox so next frame gets new overlay
    bbox = bbox_bytes
```

Et voici le résultat obtenu :

b) Conclusion :

On constate que le modèle fonctionne correctement avec des performances acceptables. Ceci est le résultat voulu.

IV. Implémentation de code python sur carte Arduino

a) Introduction :

Dans ce dernier chapitre, on va parler sur l'implémentation de code python de chapitre précédent dans une carte à l'aide de Arduino. L'objectif est d'échanger des informations (code python) entre la carte Arduino UNO et l'ordinateur à travers le langage de programmation Python.

b) Relation entre Arduino et python :

La carte Arduino permet de faire l'acquisition d'un signal analogique par l'intermédiaire d'un capteur et de le convertir en signal numérique grâce à son CAN (10 bits). Il est ensuite possible de transférer ces données par le port série

vers l'ordinateur pour réaliser un traitement numérique avec **Python**.

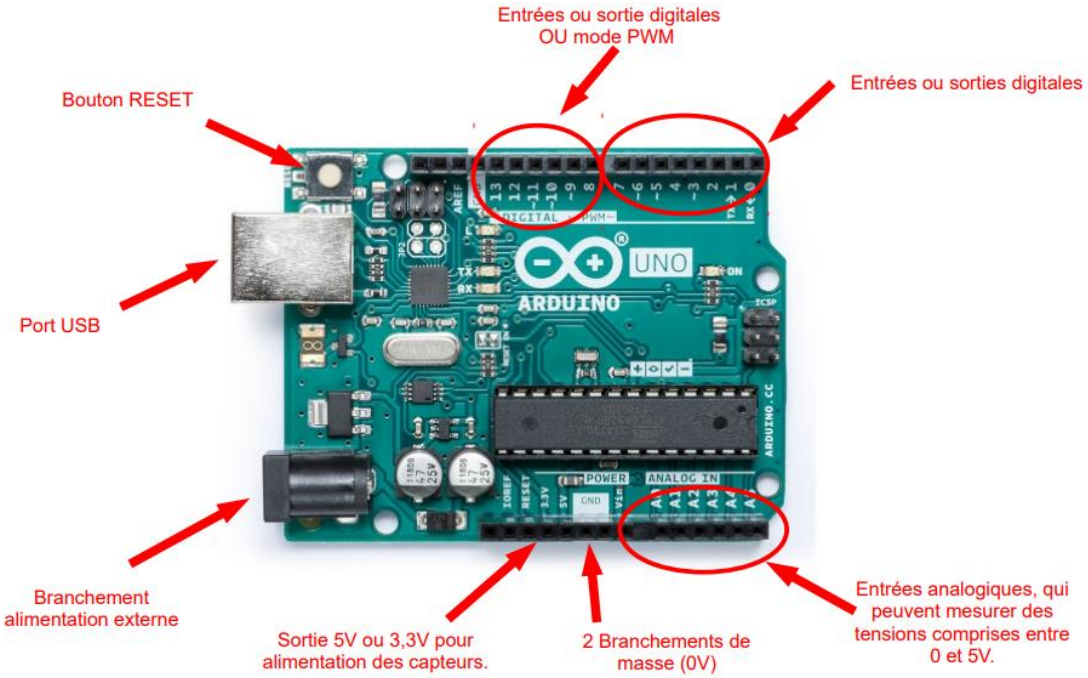
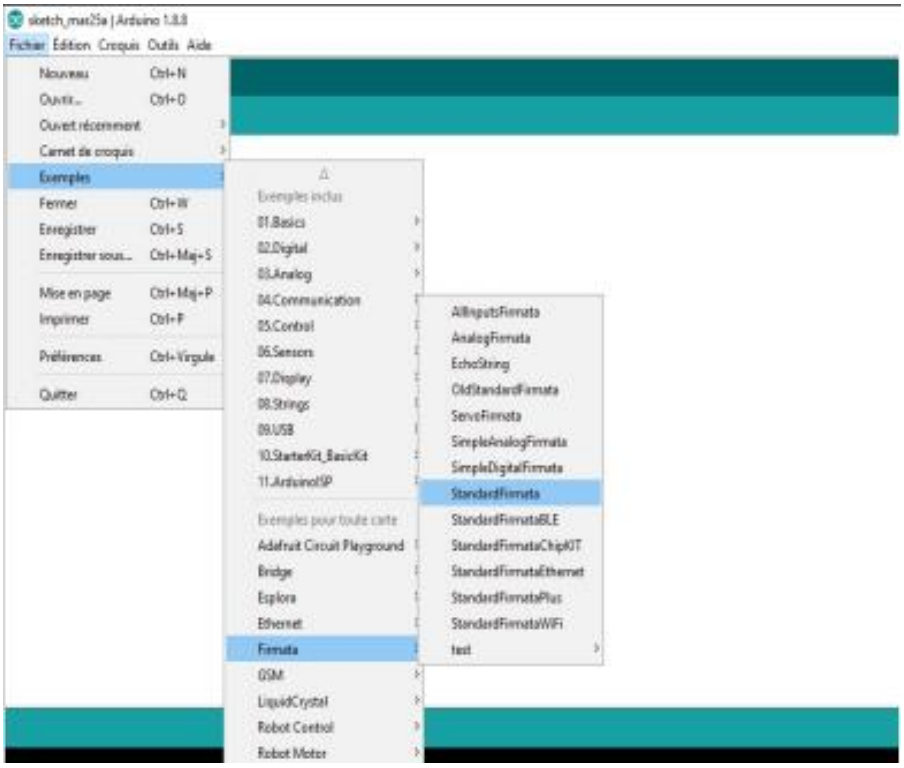


Figure 8 : Les différentes parties du carte Arduino

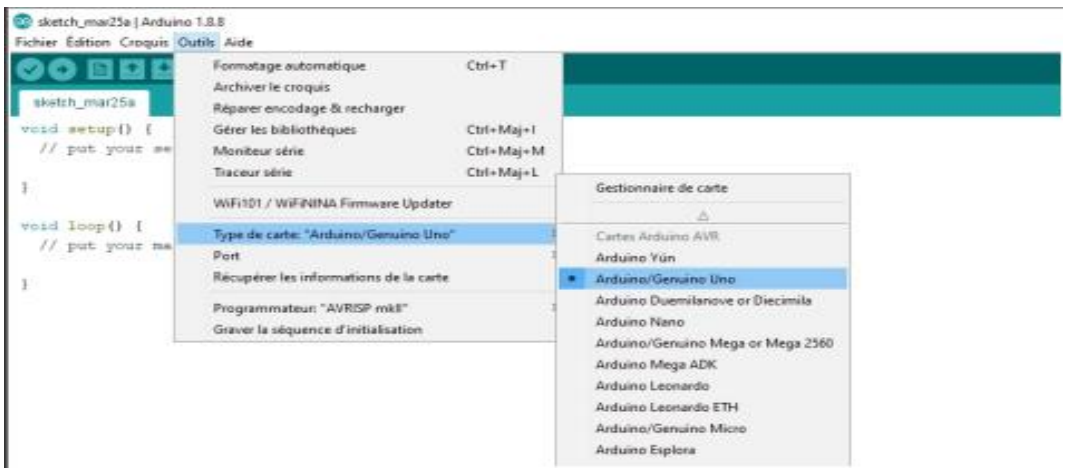
Etape 1 : **Installation et configuration du Firmware sur la carte**

La première étape consiste à installer un firmware différent sur la carte pour qu'elle puisse communiquer en Python.

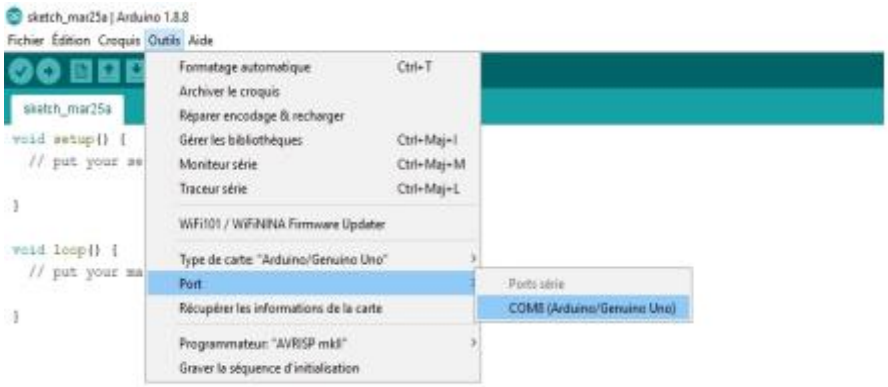
- Après l'installation de logiciel Arduino, on va commencer par la configuration de ce logiciel avec les données de format standars : fichier > Exemples > Firmata > StandardFirmata
  Comme montre la figure :



- Branchement de la carte Arduino Uno en USB. Choisir le type de carte



- Connexion de la carte avec le port : Outils > Port > COMX (la carte connectée apparaît dans la liste)



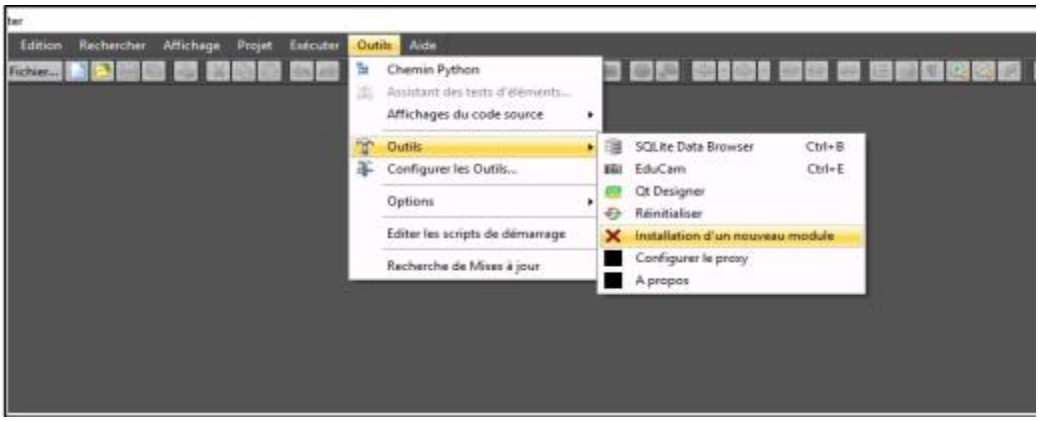
Il ne reste plus qu'à téléverser le microprogramme (firmware) sur la carte.

La carte est prête pour être utilisée avec un IDE Python comme EduPython. Ce firmware reste ensuite sur la carte le temps de l'activité, même si elle est déconnectée ou éteinte.

Étape 2 : Téléchargement de la bibliothèque Pyfirmata dans Edupython

Pour utiliser la carte avec EduPython, il faut installer la bibliothèque Pyfirmata, qui renferme les commandes Python compréhensibles par la carte Arduino.

Démarrer EduPython. Faire Outils > installation d'un nouveau module.



Un menu apparaît Choisir 2 et ensuite taper pyfirmata. Suivre les instructions. Une procédure similaire existe pour Thonny, Pyzo, Spyder.

Étape 3 : les principales fonctions de pyfirmata

Nous allons pouvoir commander la carte Arduino grâce au logiciel EduPython.

- Démarrer EduPython

- Brancher la carte Arduino Uno. En cas d'oubli, pour retrouver le port de communication (COM) de la carte, pour cela, il faut ouvrir le gestionnaire de périphériques dans le menu Démarrer de Windows. La carte reliée à l'ordinateur apparaît dans la liste, avec le numéro de port. Ici c'est COM8.

Dans Edupython, Nous pouvons taper un code qui sera reconnu par la carte, à

condition de le faire commencer par les lignes suivantes :

Importation du module pyfirmata

Choisir un nom pour la carte(« carte ») et son adresse (« COM8 »)

```
from pyfirmata import Arduino, util
import time

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()
```

- Ces deux lignes lancent la fonction mesure en temps réel

Les entrées et sorties sur la carte se définissent grâce à la fonction get_pin() Avec pour un choix de paramètres dans la parenthèse : analogique, digitale, pwm (non traité ici). Le numéro de branchement sur la carte (de 0 à 13) input , entrée, output et sortie.

- Remarques importantes :

Le module time doit être importé en début de code. Les noms de variables « carte », « acquisition », « entree1 », « sortie1 » ont été choisis par le rédacteur du code, il est donc possible de les nommer comme on le souhaite (si possible de manière explicite). La fonction carte.exit() à la fin du code, termine l'acquisition de mesures proprement.

```
entree1 = carte.get_pin('a:0:i')
sortie1 = carte.get_pin('d:13:o')
time.sleep(1.0)
```

Nous allons pouvoir commencer les mesures et actions avec les fonctions read() et write(valeur) Plusieurs cas sont possibles :

Entrée		**Sortie**
Analogique	**Digitale**	**Digitale**
read()	**read()**	**write(*valeur*)**
Lis la valeur de tension sur l'entrée analogique. Cette fonction renvoie un nombre type **Float** compris entre 0 et 1 qui correspond à une tension comprise entre 0V et 5V	Lis l'état de la tension sur l'entrée analogique. (type **Boolean**) **0** pour 0V (False) **1** pour 5V (True)	Impose une tension sur la sortie : 0V pour write(**0**) 5V pour write(**1**). *Il est possible d'utiliser* ***write****(False) ou* ***write****(True)*

Dernière étape : Nous avons tapé le code de l'algorithme sur l'environnement EduPython avec tous les entrées et sorties.

c) Conclusion :

Ce chapitre montre toutes les étapes nécessaires pour configurer et implémenter notre code python sur une carte Arduino. Nous arrêterons à ce niveau-là car les prochaines étapes sont possible mais hors de notre portée financièrement et scientifiquement.