





**MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET
DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOU D MAMMERI DE TIZI OUZOU
Faculté de Génie électriques & informatique
Département d'informatique**



Mémoire :

Système intelligent pour la classification de fruit

Réalisé par :

 SEHAKI Sofiane
 MONIR Marzouk

Promoteur :

 ACHOUR Hakim

Promotion : 2023/2024

Table des matières

Introduction générale.....	4
CHAPITRE 1 : état de l'art sur le système de tri de fruit.....	6
1. Introduction :	7
2. Travaux déjà réalisé :	7
2.1 DeepFruit [2] :	7
2.2 Méthode pour la détection des mangues [3] :	8
2.3 Fresh and Rotten Classification Application [4]:	8
2.4 Les dattes avec la vision artificielle [5]:	9
2.5 systèmes de tri de fruit mécanique [6]:	9
2.6 systèmes de trieur de pommes [7] :	10
2.7 trieuses de dattes automatique [8]:	10
3. Conclusion :	11
CHAPITRE 2 : Conception.....	12
1. Introduction :	13
2. Description globale du système :	13
3. Conception software :	14
3.1 Architecture du programme :	14
3.2 Classification :	15
3.3 Calcul Calibre :	25
4. Conception Hardware :	29
4.1 Choix de carte de développement à utiliser :	29
4.2 choix de caméra à utiliser :	32
5. Conclusion :	33
CHAPITRE 3 : Réalisation	34
1. Introduction :	35
3. Implémentation de MobileNet sur les cartes de développements :	37
3.1 Implémentation sur la Beagle Bone Black :	37
3.2 Les résultats obtenus :	38
3.3 Implémentation sur la Raspberry :	38
3.3 Les résultats obtenus :	39
4. Implémentation de MASK RCNN sur les cartes de développements :	40
4.1 implémentations sur la Raspberry :	40
4.3 choix du modèle et de carte à utiliser :	41
5. Résultat du calcul du calibre :	42
6. Conclusion :	42
7. Conclusion générale :	43
7.1 Synthèse :	43

7.2 Perspectives :.....	43
Bibliographie	44

Figure 1 Exemples de détection de deux fruits en utilisant DeepFruit [2].....	7
Figure 2 Plate-forme d'imagerie automatisée, impliquant un appareil GNSS (sur poteau), un projecteur à LED de 720 W et une caméra, alimentée par un onduleur-générateur et montée sur un véhicule agricole [3]	8
Figure 3 Les étapes de la méthode proposée pour la classification automatique des fruits des dattes. [5].....	9
Figure 4 système de tri du baies [6]	9
Figure 5 système de tri de pomme [7]	10
Figure 6 trieuse de dettes [8]	10
Figure 7 Schéma globale de notre système	13
Figure 8 architecture du programme	14
Figure 9 Architecture de MobileNetV2 [10]	16
Figure 10 Architecture de Mask RCNN [11]	17
Figure 11 Quelques images du dataset [12]	18
Figure 12 partie du programme de pré entraînement.....	19
Figure 13 architecture du modèle.....	20
Figure 14 Architecture finale du modèle entraîner.....	20
Figure 15 Choix d'optimiseur et de fonction d'erreur	21
Figure 16 Exemples d'optimiseurs [15]	21
Figure 17 une partie du code pour initialiser les hypers paramètre.....	22
Figure 18 Segmentation des objets dans les images Prétraitement [16]	23
Figure 19 La méthode load_custom.....	23
Figure 20 La méthode load_mask	23
Figure 21 La méthode count_classes	23
Figure 22 La fonction load_image_dataset.....	24
Figure 23 les choix des hypers paramètre.....	24
Figure 24 partie 1 du programme de calcul de calibre	25
Figure 25 Espace colorimétrique HSV [20]	26
Figure 26 Espace colorimétrique RVB [20]	26
Figure 27 formule mathématique de calcul de seuille [21]	27
Figure 28 Fonctionnement de la méthode cv2.RETR_TREE [22].....	27
Figure 29 partie 2 du programme de calcul du calibre	28
Figure 30 beagle bone black [23]	29
Figure 31 Raspberry Pi 4 [25]	30
Figure 32 Caméra ip [26]	32
Figure 33 webcam [27]	33
Figure 34 la précision du modèle	35
Figure 35 la fonction d'erreur du modèle	36
Figure 36 programme d'implémentation du modèle.....	37
Figure 37 programme de conversion	38
Figure 38 programme d'implémentation du modèle.....	39
Figure 39 Architecture du modèle MaskRCNN pour la segmentation d'instance avec une configuration simple.....	40
Figure 40 Chargement des poids pré-entraînés pour le modèle MaskRCNN à partir du fichier mask_rcnn_coco.h5	40
Figure 41 Détection d'instances sur une image à l'aide du modèle MaskRCNN	40

Introduction générale

L'agriculture est un domaine en constante évolution et très vaste, cependant il est confronté à de nombreux problèmes qui peuvent avoir un impact sur la qualité et la quantité de la production alimentaire. Deux problèmes majeurs se posent dans l'agriculture. Tout d'abord, il y a un manque de main d'œuvre qualifiée pour les activités difficiles qui nécessitent de l'expérience, de la concentration et une bonne condition physique. Ensuite, il y a le commerce agricole qui dépend des exigences des consommateurs. Ces derniers demandent plus que de simples quantités, ils exigent un rapport qualité-prix optimal, ce qui peut engendrer des problèmes de gestion et de conservation des produits agricoles.

Cependant, selon le livre [1] le marché de l'IA en agriculture était évalué à près de 518,7 millions en 2017 et devrait se développer de plus de 22,5% par an pour atteindre 2.6 milliards d'ici 2025. En particulier, la classification et la régression des fruits en fonction de différentes caractéristiques telles que la forme, la texture, la couleur et la qualité peuvent être effectuées de manière efficace. De telles applications de l'IA peuvent aider à améliorer la productivité et la qualité des produits agricoles tout en réduisant les coûts de main-d'œuvre et en répondant aux exigences des consommateurs.

Pour rependre à cette problématique on n'a pas pensé à concevoir un système qui permet la classification du fruit selon la qualité et le calibre et cela en se basant sur des techniques de l'intelligence artificielle.

Dans cette étude, nous allons aborder différents aspects de l'intelligence artificielle dans le domaine agricole. Nous commencerons par examiner l'état de l'art dans ce domaine, en explorant les avancées récentes et les applications de l'IA dans l'agriculture. Cette première partie nous permettra de mieux comprendre les différentes techniques et technologies utilisées.

Dans la deuxième partie, nous aborderons la conception de notre système. Nous discuterons des différentes composantes et fonctionnalités que nous souhaitons intégrer, en prenant en compte les besoins spécifiques de l'agriculture. Nous examinerons les différentes méthodes d'IA qui peuvent être utilisées, telles que l'apprentissage automatique, les réseaux neuronaux, et d'autres techniques pertinentes.

Dans la troisième partie, nous nous concentrerons sur les résultats obtenus. Nous présenterons les données et les métriques que nous avons collectées, en mettant en évidence les avantages et les limites de notre système. De plus, nous expliquerons les étapes nécessaires pour implémenter notre solution sur des cartes telles que le Raspberry Pi et la BeagleBone Black, qui sont couramment utilisées dans les projets d'IA embarquée.

CHAPITRE 1 : état de l'art sur le système de tri de fruit

1. Introduction :

Dans ce chapitre, nous aborderons les concepts clés de l'intelligence artificielle (IA) utilisés dans l'agriculture, en mettant l'accent sur le tri des fruits et légumes. Nous examinerons les algorithmes, les modèles et les jeux de données couramment utilisés dans ce domaine, ainsi que les applications spécifiques de l'IA dans l'agriculture.

2. Travaux déjà réalisé :

2.1 DeepFruit [2] :

C'est application basée sur l'apprentissage profond, c'est un système de détection qui fait partie d'une plateforme robotique agricole pour la récolte automatisée des fruits, cette application est basée sur architecture faster rcnn et entraîné sur un jeu de données obtenue par combinaison des informations des images de deux modalités RVB et PIR, l'application et d'une précision de 0.83 %.

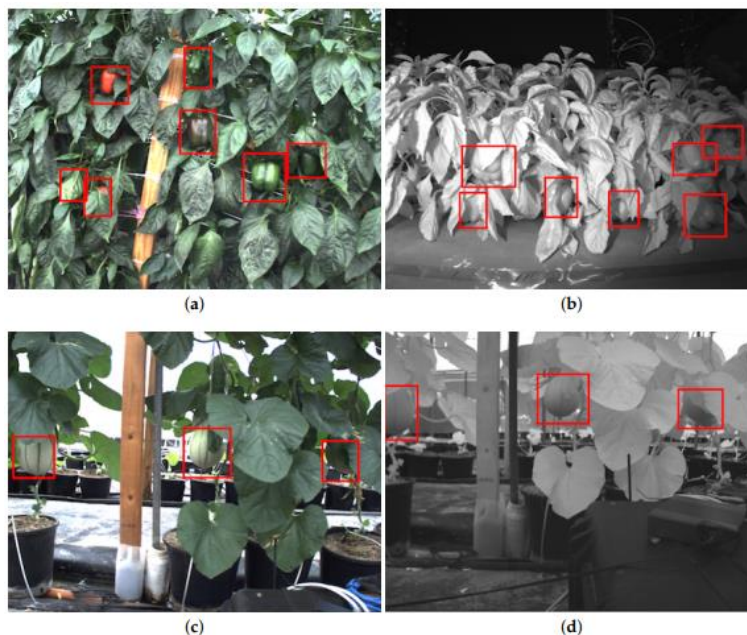


Figure 1 Exemples de détection de deux fruits en utilisant DeepFruit [2]

2.2 Méthode pour la détection des mangues [3] :

Il s'agit d'une application de suivi, détection et comptage des mangues sur les arbres en se basant sur l'apprentissage profond, dans cette méthode ils ont utilisé mangoyolo pour la détection et l'algorithme hongrois en plus de filtre kalman, la détection est faite à l'aide d'un flux vidéo de 10 fps (image par seconde), selon les résultats la méthode a détecté 2050 fruits ce qui fait 62% de la récolte.



Figure 2 Plate-forme d'imagerie automatisée, impliquant un appareil GNSS (sur poteau), un projecteur à LED de 720 W et une caméra, alimentée par un onduleur-générateur et montée sur un véhicule agricole [3]

2.3 Fresh and Rotten Classification Application [4]:

Ce travail concerne un modèle d'intelligence artificielle basé sur l'architecture CNN (réseaux de convolution) qui a été entraîné sur jeu de donnée contenant 3 types de fruits obtenus à partir de kaggle, ce modèle est d'une précision de 97,80%.

2.4 Les dattes avec la vision artificielle [5]:

Des chercheurs ont développé un modèle qui permet de classer les dattes par rapport à la variété et le niveau de maturité, ils ont utilisé l'algorithme GMM et une dataset de 5000 images de différentes variétés.

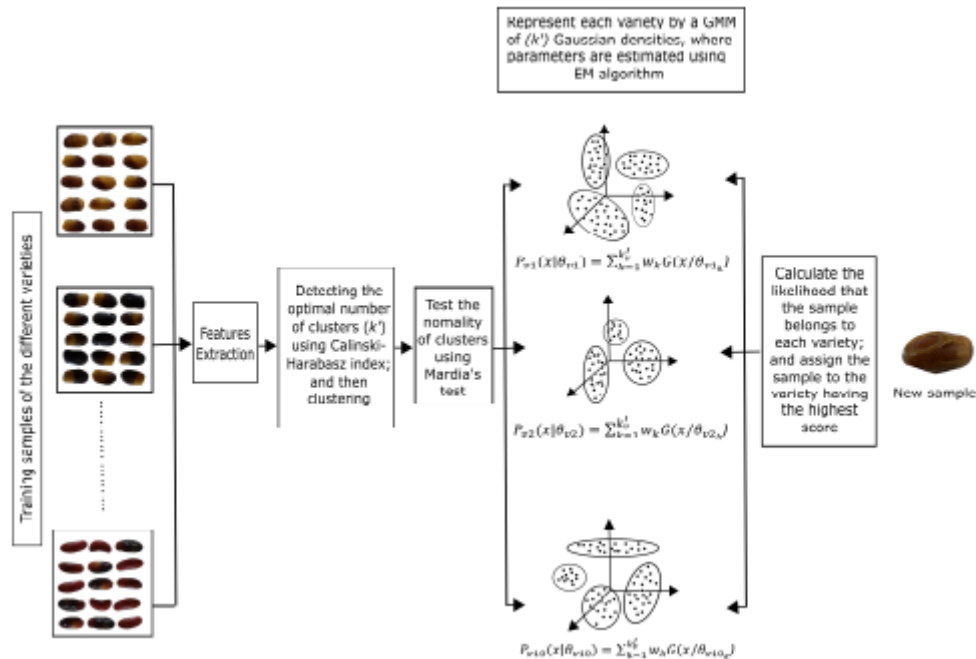


Figure 3 Les étapes de la méthode proposée pour la classification automatique des fruits des dattes. [5]

2.5 systèmes de tri de fruit mécanique [6]:

C'est un système qui permet de trier le fruit baies automatiquement selon le calibre. D'abord Les baies passent sur le tapis qui a des coupes-circuit et puis des petites baies tombent par ces coupes-circuit au convoyeur de décharge.



Figure 4 système de tri du baies [6]

2.6 systèmes de trieur de pommes [7] :

C'est une machine de haute performance qui permet de trier les fruits pomme selon le calibre.

Méthode de tri :

- L'équilibre idéal entre la simplicité et la performance
- Pas de transitions supplémentaires grâce à un seul transporteur de fruits intelligent
- Une vitesse de tri très élevée jusqu'à pas moins de 15 fruits par seconde
- De 2 à 10 lignes
- Un nombre illimité de sorties
- Facilement extensible grâce à la construction modulaire
- Une durée de vie prolongée grâce à des mises à jour modulaires
- Des fruits jusqu'à un calibre de 120 mm

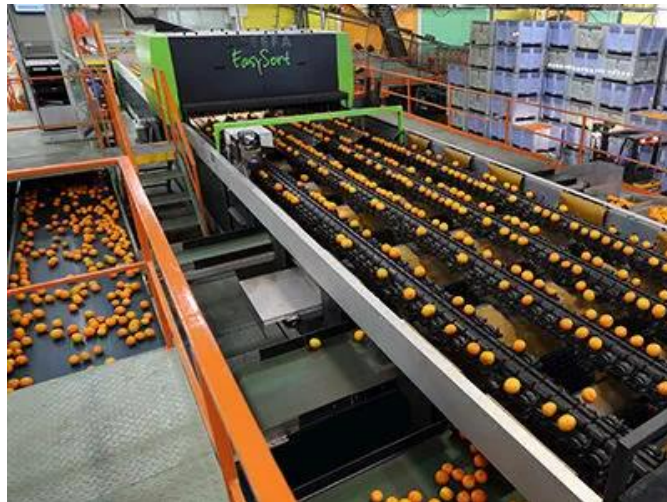


Figure 5 système de tri de pomme [7]

2.7 trieuses de dattes automatique [8]:

Cette machine à trier les dattes comprend principalement la partie d'alimentation automatique et de classification des rouleaux, une fois que les dattes sont introduites dans le seau d'alimentation, elles seront transportées vers la partie de tri par l'élévateur d'alimentation automatique.



Figure 6 trieuse de dattes [8]

3. Conclusion :

Dans ce chapitre nous avons étudié les différents travaux déjà réalisés, notamment pour la récolte et le tri automatique des fruits et légumes, nous aborderons dans le prochain chapitre la conception de notre système.

CHAPITRE 2 : Conception

1. Introduction :

Dans ce chapitre, nous allons concevoir un système qui permet la classification de fruit, en se basant sur l'intelligence artificielle. Pour résoudre le problème rencontré par les agriculteurs, ce chapitre est repartitionné en trois sections, d'abord nous allons discuter d'une façon globale sur le système en donnant des schémas, ensuite nous allons détailler la partie logiciel du système, enfin nous allons concevoir la partie matériel du système.

2. Description globale du système :

Ce système est repartitionné en trois tâches :

- Tapis roulant : il permet de déplacer les fruits d'un point à un autre à vitesse constante afin de faciliter la tâche de détection.
- Un dispositif de détection : comprenant une caméra connectée à une carte dotée d'un processeur assez puissant pour exécuter des programmes d'IA et de traitement d'image.
- Le dispositif de direction : le fruit se dirige vers sa destination finale selon la décision prise par le dispositif de détection.

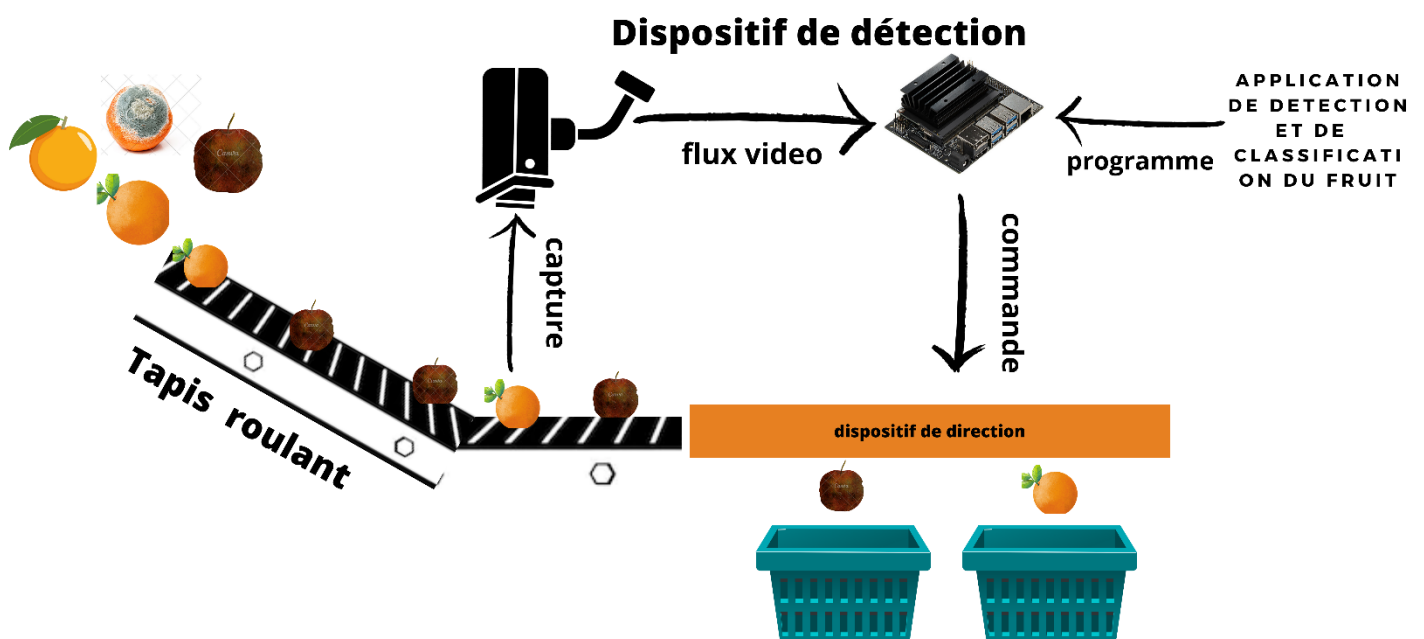


Figure 7 Schéma globale de notre système

3. Conception software :

3.1 Architecture du programme :

L'architecture de notre programme commence tout d'abord par la détection du fruit d'une manière synchroniser avec la vitesse du tapis roulant, ensuite les images seront envoyées vers les deux processus de classifications qui permettent de classier le fruit selon la classe au quelle il appartient soit de bonnes qualité ou de mauvaise, et le processus du calibre qui permet de calculer et de retourner le calibre du fruit, enfin les résultats de ces deux processus permettent de commander le dispositif de direction.

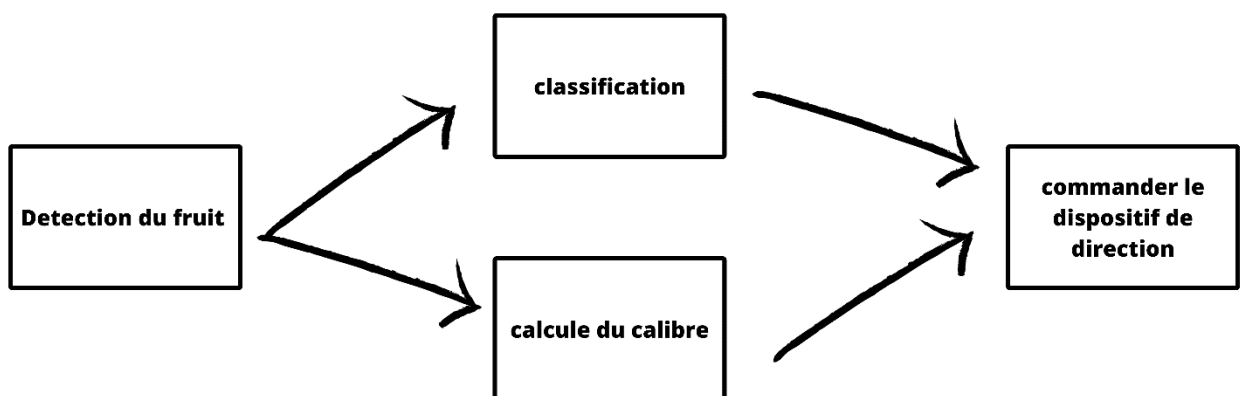


Figure 8 architecture du programme

3.2 Classification :

3.2.1 Outils à utilisés :

- TensorFlow [9]: TensorFlow est l'une des bibliothèques les plus populaires et performantes pour l'apprentissage en profondeur. Elle est largement utilisée dans l'industrie et la recherche. TensorFlow alimente de nombreuses applications d'envergure, notamment chez des géants de la technologie tels qu'Uber, Dropbox et Airbnb, C'est un framework open source qui offre une grande efficacité et des fonctionnalités puissantes pour le développement de modèles d'apprentissage automatique.
- PyTorch [9]: PyTorch est un framework populaire pour l'apprentissage en profondeur, qui a gagné en popularité ces dernières années, en particulier dans la communauté de la recherche en apprentissage automatique, il est largement apprécié pour sa facilité d'utilisation et son approche plus intuitive du développement de modèles.

Selon l'article [9] TensorFlow est reconnu pour sa capacité à gérer des projets d'apprentissage en profondeur complexes et à grande échelle, grâce à son modèle de calcul de flux de données. Il offre également une intégration avec d'autres outils et bibliothèques couramment utilisés dans l'écosystème de l'apprentissage automatique. Tant dit que PyTorch soit peut-être moins répandu dans l'industrie que TensorFlow, il est devenu le choix privilégié de nombreux chercheurs et praticiens pour sa flexibilité et sa facilité d'expérimentation.

Nous avons donc choisi d'utiliser TensorFlow comme framework pour l'apprentissage en profondeur. Grâce à sa large communauté connue et son utilisation dans de nombreuses applications industrielles.

3.2.2 Architecture des modèles à utiliser :

- MobileNetV2 [10] : MobileNetV2 est un modèle de classification développé par Google, distinct de MobileNetSSDV2. Il a été spécialement conçu pour répondre aux contraintes informatiques des appareils tels que les smartphones tout en fournissant des capacités de classification en temps réel. MobileNetV2 est largement utilisé pour des tâches de classification d'images sur des appareils avec des ressources limité.

L'architecture MobileNetV2 utilise une structure résiduelle inversée avec des couches de goulot d'étranglement pour les blocs résiduels, des convolutions légères pour filtrer les fonctionnalités dans la couche d'extension, et supprime les non-linéarités dans les couches étroites. Ces techniques permettent de réduire la complexité computationnelle, d'améliorer l'efficacité des ressources et de maintenir des performances de classification de haute qualité.

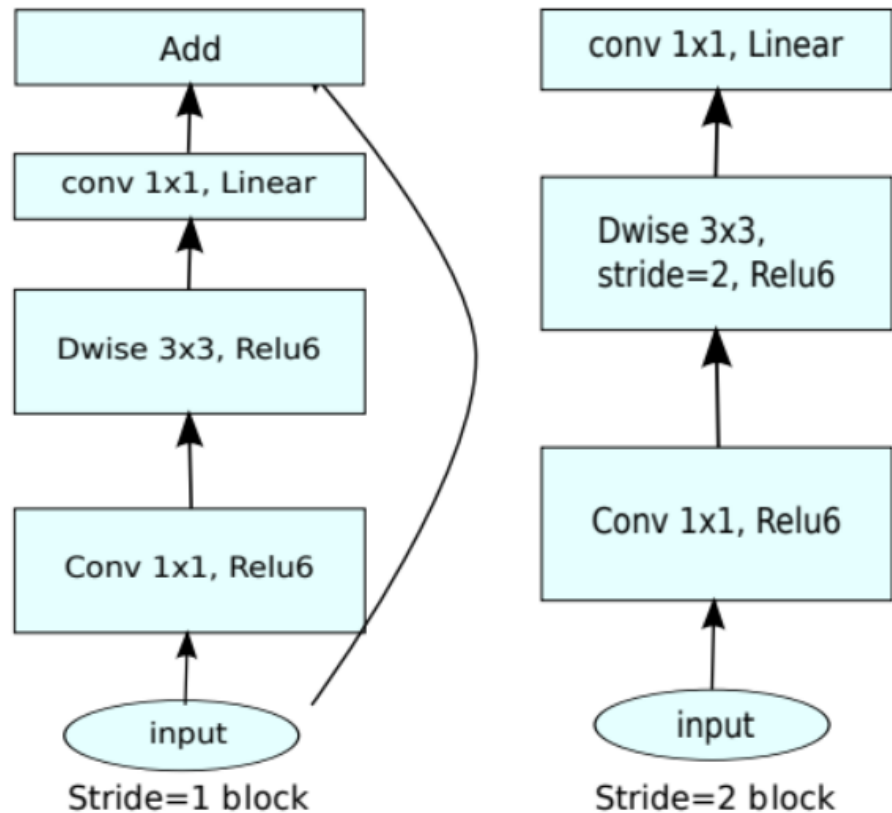


Figure 9 Architecture de MobileNetV2 [10]

- Mask RCNN [11]: Est un modèle de détection d'objets avancé qui combine les fonctionnalités de deux algorithmes puissants : Faster R-CNN et l'apprentissage de masques. Il est utilisé pour détecter et segmenter précisément les objets dans une image.

L'architecture Faster R-CNN combine l'utilisation du réseau de proposition de région (RPN) pour générer des régions d'intérêt, le réseau RoI Align pour générer et déformer les boîtes englobantes, dans une dimension fixe, Les caractéristiques déformées sont ensuite introduites dans des couches entièrement connectées pour effectuer une classification à l'aide de softmax et la prédiction de la boîte aux limites est encore affinée à l'aide du modèle de régression, et le classificateur de masque pour générer des masques précis pour chaque objet détecté. Cette architecture permet une détection et une segmentation précises des objets dans une image.

Mask RCNN

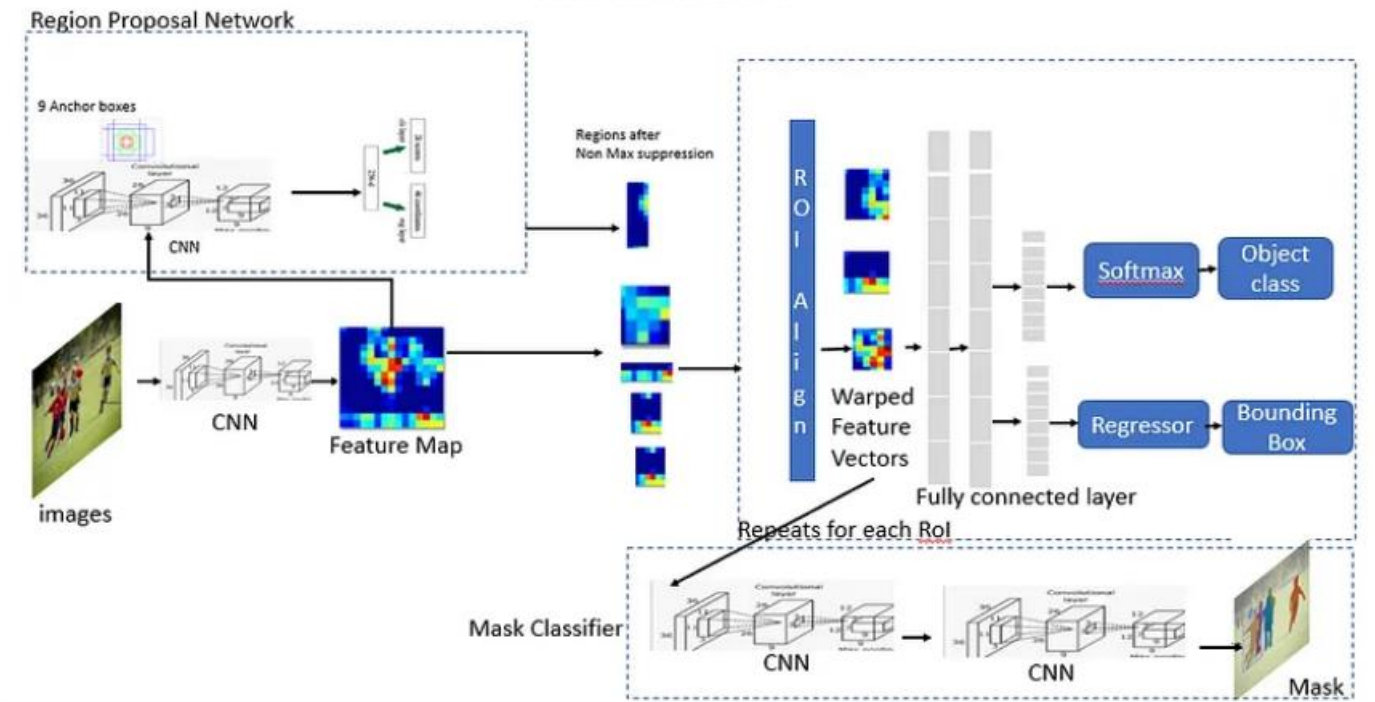


Figure 10 Architecture de Mask RCNN [11]

3.2.3 Entraînement du modèle basé sur Mobile Net :

Acquisition des données (images des fruits) : Nous avons extrait une dataset depuis kaggle [12] qui contient 2 dossiers (train,test), un pour l'entraînement et l'autre pour le test chaque'un de ces dossiers est compose de 6 dossiers, 3 pour la bonne qualité et 3 pour la mauvaise qualité de fruit (orange, banane , pomme)



Figure 11 Quelques images du dataset [12]

3.2.4 Chargement du data set sur Colab :

Nous avons établi une connexion entre Google Colab et Google Drive afin de charger et d'utiliser le jeu de données de manière efficace et sécurisée.

Prétraitement :

```
base_path = Path("/content/drive/MyDrive")
train_path = base_path/"dataset2/dataset/train"
test_path = base_path/"dataset2/dataset/test"

traindatagen = ImageDataGenerator(validation_split=0.3,
                                   preprocessing_function=preprocess_input)
testdatagen = ImageDataGenerator(preprocessing_function=preprocess_input)

classes = ["rottenbanana", "freshbanana", "rottenapples",
           "rottenoranges", "freshapples", "freshoranges"]
traingen = traindatagen.flow_from_directory(directory=train_path,
                                             target_size=(224,224),
                                             class_mode="categorical",
                                             classes=classes,
                                             batch_size=64,
                                             subset="training",
                                             shuffle=True,
                                             seed=42)

valgen = traindatagen.flow_from_directory(directory=train_path,
                                             target_size=(224,224),
                                             class_mode="categorical",
                                             classes=classes,
                                             batch_size=64,
                                             subset="validation",
                                             shuffle=True,
                                             seed=42)
```

Figure 12 partie du programme de pré entraînement

Nous avons utilisé la classe ImageDataGenerator pour faciliter le chargement des données dans notre modèle. Cette classe nous permet de prétraiter les images de manière efficace. Dans notre cas, nous avons défini deux instances de cette classe : traindatagen et testdatagen.

La variable traindatagen est utilisée pour charger les données d'entraînement. Nous avons spécifié une valeur de validation_split de 0.3, ce qui signifie que 30% des données d'entraînement seront réservées pour la validation. Cela nous permettra d'évaluer les performances de notre modèle pendant l'entraînement.

La variable testdatagen est utilisée pour charger les données de test. Ces données ne sont pas divisées en ensembles d'entraînement et de validation, car elles sont utilisées uniquement pour évaluer les performances finales de notre modèle une fois qu'il est entraîné.

Nous avons également utilisé la fonction `preprocess_input` pour effectuer un prétraitement supplémentaire sur les images chargées. Cette fonction est spécifique à MobileNet, un modèle de réseau de neurones convolutionnel que nous utilisons. Elle permet d'appliquer des transformations spécifiques aux images afin de les préparer de manière optimale pour l'entraînement et l'évaluation du modèle MobileNet.

Nous avons redimensionné Les images à une taille de 224x224 pixels (`target_size`). Le mode de classification est défini sur "categorical" pour indiquer que nous avons plusieurs classes d'images.

Les classes des images sont spécifiées dans la liste "classes". Dans notre cas, les classes sont "rottenbanana", "freshbanana", "rottenapples", "rottenoranges", "freshapples" et "freshoranges".

Architecture du modèle :

```
base_mdl.trainable = False
x = base_mdl.output
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(units=32,activation="relu")(x)
x = tf.keras.layers.Dense(units=16,activation="relu")(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = tf.keras.layers.Dense(units=len(classes),activation="softmax")(x)
mdl = tf.keras.models.Model(inputs=base_mdl.input,outputs=outputs)
mdl.summary()
```

Figure 13 architecture du modèle

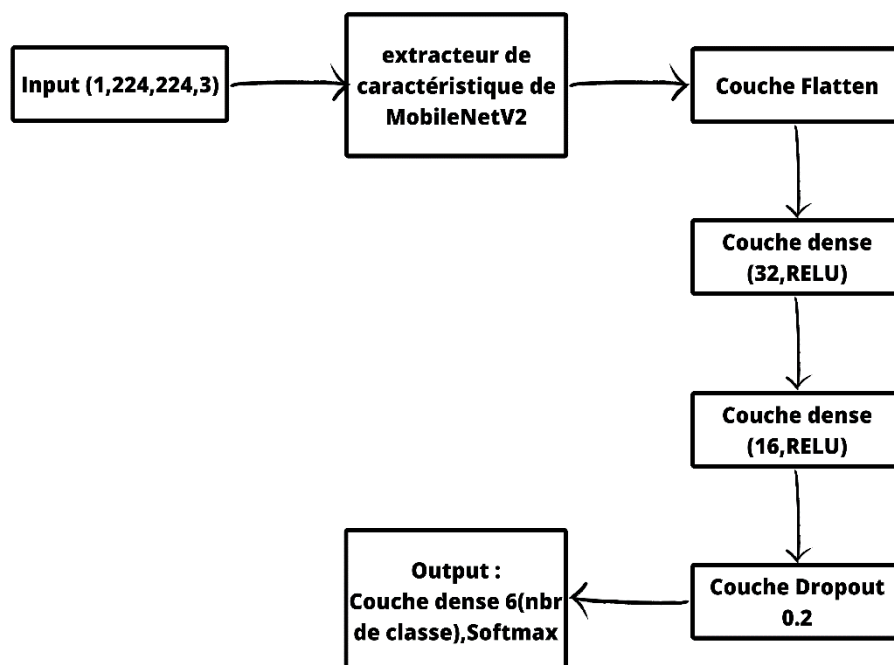


Figure 14 Architecture finale du modèle entrainer

Nous avons ajouté à l'extracteur de caractéristique pré entraîné sur imageNet une couche Flatten pour aplatir la carte de caractéristique mobileNet2, ensuite nous avons ajouté 2 couches de neurones entièrement connectés, une avec 32 neurones et l'autre avec 16, avec une fonction d'activation ReLU. Cette couche supplémentaire ajoute une autre transformation non linéaire pour capturer des informations plus spécifiques et complexes.

La couche Dropout, `tf.keras.layers.Dropout(0.2)`, est utilisée pour introduire de la régularisation dans le modèle. Elle désactive aléatoirement 20% des neurones de la couche entièrement connectée précédente pendant l'entraînement.

Enfin on n'a ajouté une couche qui utilise la fonction d'activation softmax pour obtenir des probabilités de classe en sortie. Elle est responsable de la prédiction finale du modèle, en assignant des probabilités à chaque classe possible.

Choix de l'optimiseur :

```
mdl.compile(loss="categorical_crossentropy",
            optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
            metrics=["accuracy"])
```

Figure 15 Choix d'optimiseur et de fonction d'erreur

La fonction de coût `categorical_crossentropy` [13] mesure la différence entre deux distributions de probabilité. Dans les réseaux de neurones, elle est utilisée pour mesurer la différence entre la distribution de sortie prédite par le réseau et la distribution de sortie réelle (étiquette de classe).

La fonction Adam [14] : L'optimisation Adam est une technique d'optimisation populaire pour l'apprentissage automatique qui met à jour les poids d'un réseau de neurones. Elle utilise des estimations adaptatives des moments du premier ordre (la moyenne) et du second ordre (la variance) des gradients pour ajuster le taux d'apprentissage de chaque paramètre.

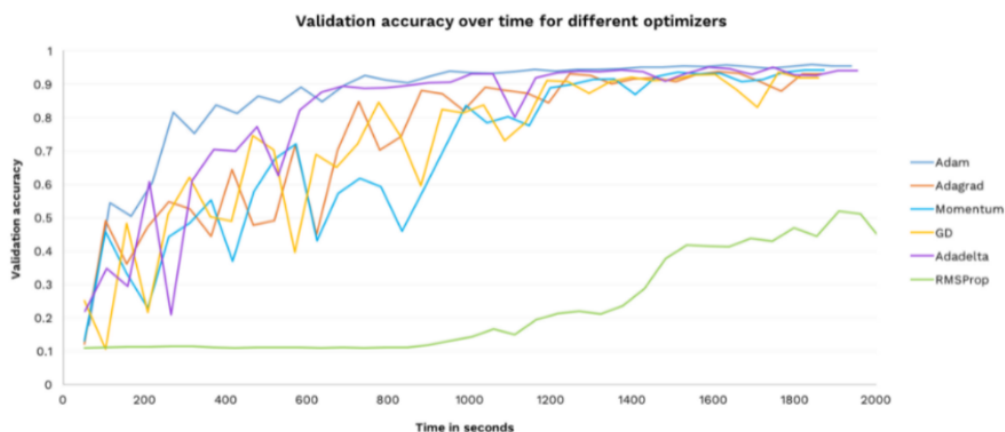


Figure 16 Exemples d'optimiseurs [15]

Pour compiler le modèle nous avons choisie :

« `categorical_crossentropy` » comme fonction d'erreur,

Voici la formule pour `categorical_crossentropy`:

`categorical_crossentropy` = $-\sum_i y_i \log(\hat{y}_i)$ [13]

Où :

y_i : est la probabilité réelle de la classe i

\hat{y}_i : est la probabilité prédite de la classe i par le réseau de neurones

La somme est calculée sur toutes les classes possibles

Nous avons également choisi l'optimiseur Adam [14] car une technique populaire, efficace et facile à implémenter dans le domaine de vision artificielle.

Initialisation des hypers paramètres :

```
hist = mdl.fit(traingen,  
               batch_size=64,  
               epochs=15,  
               validation_data=valgen)
```

Figure 17 une partie du code pour initialiser les hypers paramètre

Cette ligne de code entraîne un modèle de réseau de neurones (mdl) sur des données d'entraînement (traingen) pendant 15 epochs (itérations sur l'ensemble de données d'entraînement) avec un batch size de 64.

Plus précisément :

traingen est un générateur qui renvoie des lots de données d'entraînement de taille 64 à chaque itération.

mdl.fit() entraîne le modèle sur ces données.

epochs=15 signifie que le modèle verra 15 fois l'ensemble des données d'entraînement (en lots de 64).

validation_data=valgen permet de valider le modèle sur un ensemble de validation valgen à la fin de chaque epoch. Cela permet de surveiller les performances du modèle sur des données qu'il n'a pas vues pendant l'entraînement.

hist est l'historique d'entraînement retourné, qui contient des métriques telles que la loss (fonction de coût) et l'accuracy (précision) sur les ensembles d'entraînement et de validation à chaque epoch.

3.2.5 Entraînement du modèle basé sur Mask RCNN :

Détection et segmentation des objets :

En utilisant ReboFlow nous avons pu segmenter les objets (les fruits) dans les images, nous avons généré le fichier JSON qui est compatible avec l'architecture Mask RCNN

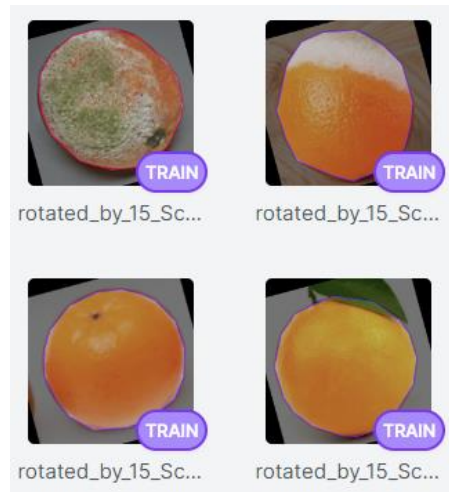


Figure 18 Segmentation des objets dans les images Prétraitement [16]

```
def load_custom(self, annotation_json, images_dir, dataset_type="train"):
    print("Annotation json path: ", annotation_json)
    json_file = open(annotation_json)
    coco_json = json.load(json_file)
    json_file.close()
    source_name = "coco_like"
```

Figure 19 La méthode load_custom

La méthode load_custom de cette classe est utilisée pour charger les données d'annotation à partir d'un fichier JSON au format COCO et les images correspondantes. Elle parcourt le fichier JSON et ajoute les noms de classe à l'aide de la méthode add_class de la classe parente. Ensuite, elle récupère les annotations pour chaque image et les ajoute à un dictionnaire.

```
def load_mask(self, image_id):

    image_info = self.image_info[image_id]
    annotations = image_info['annotations']
    instance_masks = []
    class_ids = []
```

Figure 20 La méthode load_mask

La méthode load_mask est utilisée pour charger les masques d'instance pour une image donnée. Les masques sont attendus sous la forme d'une matrice booléenne [hauteur, largeur, nombre d'instances]. La méthode parcourt les annotations de l'image, crée un masque pour chaque annotation en utilisant les coordonnées de segmentation et les remplit de manière appropriée. Les masques d'instance et les identifiants de classe correspondants sont collectés et retournés.

```
def count_classes(self):
    class_ids = set()
```

Figure 21 La méthode count_classes

La méthode count_classes est utilisée pour compter le nombre de classes uniques présentes

dans l'ensemble de données. Elle parcourt les identifiants de classe dans les annotations de chaque image et les ajoute à un ensemble. Finalement, elle retourne le nombre de classes uniques.

```
def load_image_dataset(annotation_path, dataset_path, dataset_type):  
    dataset_train = CustomDataset()  
    dataset_train.load_custom(annotation_path, dataset_path, dataset_type)  
    dataset_train.prepare()  
    return dataset_train
```

Figure 22 La fonction load_image_dataset

La fonction load_image_dataset utilise la classe CustomDataset pour charger les données d'annotation et les préparer pour l'entraînement ou la validation du modèle. Elle retourne l'instance de l'ensemble de données préparée.

Ces parties du code sont essentielles pour la préparation des données d'annotation dans le format COCO-like et pour fournir les masques d'instance nécessaires à l'entraînement du modèle Mask R-CNN.

Les choix des hypers paramètre :

```
model = modellib.MaskRCNN(mode="training", config=config,  
                           model_dir=MODEL_DIR)  
model.load_weights(COCO_MODEL_PATH, by_name=True,  
                  exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",  
                           "mrcnn_bbox", "mrcnn_mask"])  
model.train(dataset_train, dataset_val,  
            learning_rate=config.LEARNING_RATE,  
            epochs=1,  
            layers='heads')
```

Figure 23 les choix des hypers paramètre

Nous avons entraîné le modèle sur les jeux de données d'entraînement (dataset_train) et de validation (dataset_val) pendant 5 epochs, en se basant sur les poids du modèle pré entrainer sur le data set coco.

3.3 Calcul Calibre :

3.3.1 Outils utilisés pour la détection :

- Open cv [17] : c'est une bibliothèque python open source qui couvre un large éventail de domaines de vision par ordinateur ou l'analyse d'images scientifiques, tel que la détection du fruit et sa segmentation, il existe plusieurs versions de open cv, la plus utilisée c'est la version 2 de open cv.
- scikit-image [18] : scikit-image se concentre principalement sur le traitement d'images. Il offre des fonctionnalités tel que la manipulation d'images, ainsi que leurs segmentations, il est utilisé dans les domaines tel que la vision par ordinateur, l'imagerie médicale et l'analyse d'images scientifiques.

Selon la publication [19] open cv est une bibliothèque conçue pour l'efficacité des calculs et avec un fort accent sur les applications en temps réel et largement utilisés, elle a une grande communauté de plus de 45 000 personnes elle offre un éventail plus large de fonctionnalités, couvrant non seulement le traitement d'images, mais aussi d'autres aspects de la vision par ordinateur et de l'analyse d'images. Tant dit que scikit-image est une collection d'algorithmes spécialisés pour le traitement d'images, et offre des fonctionnalités avancées dans ce domaine.

Nous avons donc choisi d'utiliser open cv comme bibliothèque pour la vision par ordinateur et le traitement d'image grâce à son efficacité de calcul puissant et sa large communauté, pour la détection du fruit.

3.3.2 Choix de l'espace de couleur :

```
import cv2
import os
import numpy as np
pj = os.path.split(_file_)[0]
os = os.chdir(pj)
img = cv2.imread('image/7.jpg')
#pms = cv2.pyrMeanShiftFiltering(img,50,51)
h,s,v = cv2.split(cv2.cvtColor(img,cv2.COLOR_BGR2HSV))
#gray = cv2.Canny(s,30,100)
_,th = cv2.threshold(s,-1,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
cnt,_ = cv2.findContours(th,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
```

Figure 24 partie 1 du programme de calcul de calibre

- HSV :

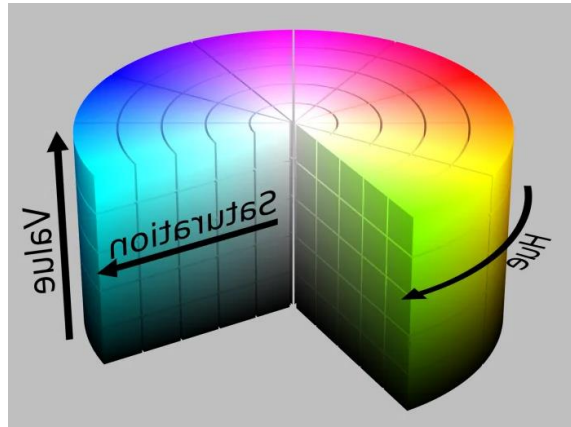


Figure 25 Espace colorimétrique HSV [20]

HSV est un modèle de couleur qui représente les couleurs en termes de teinte (hue), de saturation et de valeur (brightness).

La teinte (hue) correspond à la couleur elle-même (rouge, orange, jaune, vert, bleu, violet, etc.). Elle est représentée par un angle de 0 à 360 degrés.

La saturation correspond à l'intensité ou à la pureté de la couleur. Plus la saturation est élevée, plus la couleur est vive et intense. Une saturation de 0 correspond au gris.

La valeur (brightness) correspond à la luminosité de la couleur. Plus la valeur est élevée, plus la couleur est claire. Une valeur de 0 correspond au noir.

- RGB :

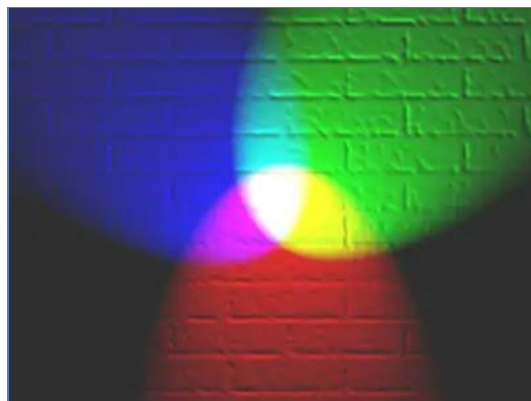


Figure 26 Espace colorimétrique RVB [20]

RGB signifie Red, Green, Blue. C'est l'espace natif des images numériques et des écrans, chaque couleur est représentée par un triplet de valeurs (R, G, B) correspondant aux intensités des canaux rouge, vert et bleu.

Nous avons choisi le modèle hsv que le modèle rgb pour deux raisons [20]:

- Les utilisateurs ont tendance à utiliser davantage le modèle HSV que le modèle RVB pour le traitement d'image et la vision par ordinateur.
- hsv il correspond plus à la perception humaine des couleurs que d'autres espaces comme RGB. Il permet par exemple de faire des seuillages sur la saturation ou la valeur pour isoler des objets d'une certaine couleur.

Otsu [21]: c'est une méthode de seuillage pour calculer automatiquement un seuil optimal (th) pour binariser l'image.

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Figure 27 formule mathématique de calcul de seuille [21]

Deux méthodes de seuillage existent [21]:

Manuelle : on fixe manuellement la valeur de seuil. Tous les pixels avec une valeur supérieure au seuil seront mis à 255 (blanc), les autres à 0 (noir).

Automatique : on utilise une méthode de seuillage adaptatif pour déterminer automatiquement le meilleur seuil. La méthode de Otsu est l'une des plus utilisées.

Dans notre cas nous avons des objets de plusieurs catégories donc nous avons choisi cv2.THRESH_OTSU qui est une méthode automatique pour choisir un seuil, car elle analyse l'histogramme de l'image et choisit un seuil qui minimise la variance intra-classe. C'est une méthode très utilisée car elle s'adapte bien aux images avec un fond complexe.

- La fonction cv2.findContours() [22]: permet de détecter les contours dans une image. Elle prend trois arguments :

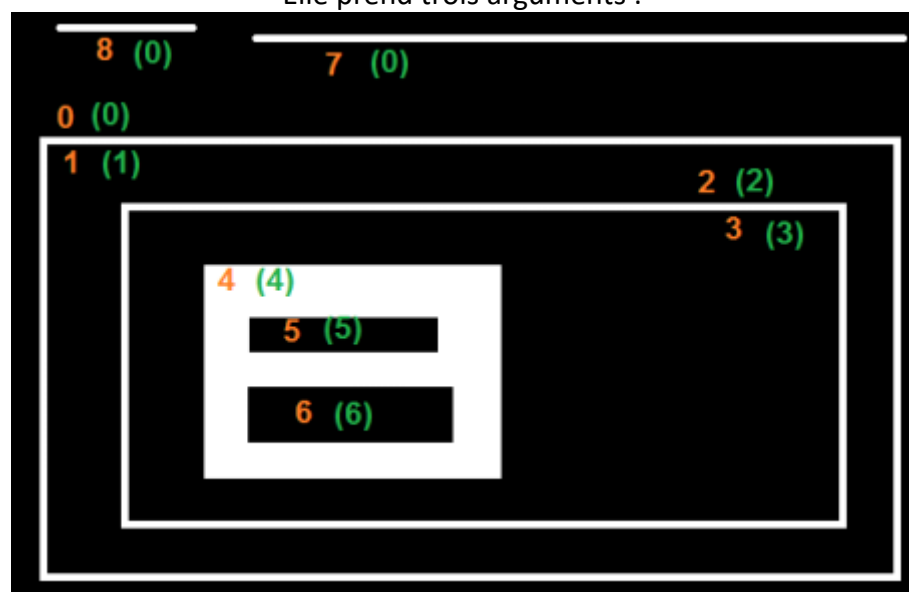


Figure 28 Fonctionnement de la méthode cv2.RETR_TREE [22]

cv2.RETR_TREE [22]: La méthode de récupération des contours, Nous on a utilisé RETR_TREE qui récupère tous les contours et reconstruit une hiérarchie complète des contours imbriqués.

cv2.CHAIN_APPROX_NONE [22] : La méthode d'approximation des contours. Nous on a utilisé CHAIN_APPROX_NONE qui stocke tous les points de contour.

```
for c in cnt :  
    if cv2.contourArea(c) > 500 :  
        cntm = c  
        cntm = cv2.approxPolyDP(cntm,0.2*cv2.arcLength(c,True),True)  
        print(cv2.arcLength(cntm,True))  
        cv2.drawContours(img,[cntm],-1,(255,0,0),2)  
        cv2.imshow('img',img)  
        cv2.waitKey(0)
```

Figure 29 partie 2 du programme de calcul du calibre

Nous avons parcouru la liste des contours dans les images et vérifier si l'aire du contour est supérieure à 500 pixels. Si oui, il s'agit probablement du contour du fruit. Ensuite nous avons approximés le contour sélectionné en utilisant cv2.approxPolyDP(), qui est une fonction approxime un contour à une séquence de segments de droite plus simples en supprimant les points redondants. Enfin nous avons calculé le périmètre du contour approximé en utilisant cv2.arcLength().

4. Conception Hardware :

4.1 Choix de carte de développement à utiliser :

- BeagleBoneBlack [23]: La BeagleBone Black est une plateforme open source idéale pour les makers, les étudiants et les développeurs. Elle permet de créer des projets en robotique, domotique, automatisation, etc. Son faible coût et sa grande flexibilité en font un outil très attrayant pour apprendre l'électronique et la programmation.

-



Figure 30 beagle bone black [23]

- Caractéristique [23] :
 - Processeur : AM335x 1 GHz ARM® Cortex-A8
 - 512 Mo de RAM DDR3
 - Stockage flash intégré eMMC 4 Go 8 bits
 - Accélérateur graphique 3D
 - Accélérateur à virgule flottante NEON
 - 2x microcontrôleurs PRU 32 bits
- Connectivité :
 - Client USB pour l'alimentation et les communications
 - Emplacement USB
 - Ethernet
 - HDMI
 - 2x embases 46 broches
- Compatibilité logicielle :
 - DebianName

- Android
- Ubuntu
- IDE Cloud9 sur Node.js avec bibliothèque BoneScript
- Et bien plus encore ...

Les pines de la beagle bone black :

Les broches GPIO (pines) de la BeagleBone Black permettent d'interfacer des périphériques externes. Elles sont utilisées pour la lecture et l'écriture de signaux numériques, la communication série, la PWM (modulation de largeur d'impulsion) et d'autres protocoles. Ces broches peuvent être configurées et contrôlées à l'aide du système d'exploitation installé sur la carte.

- Raspberry Pi 4 [24]:

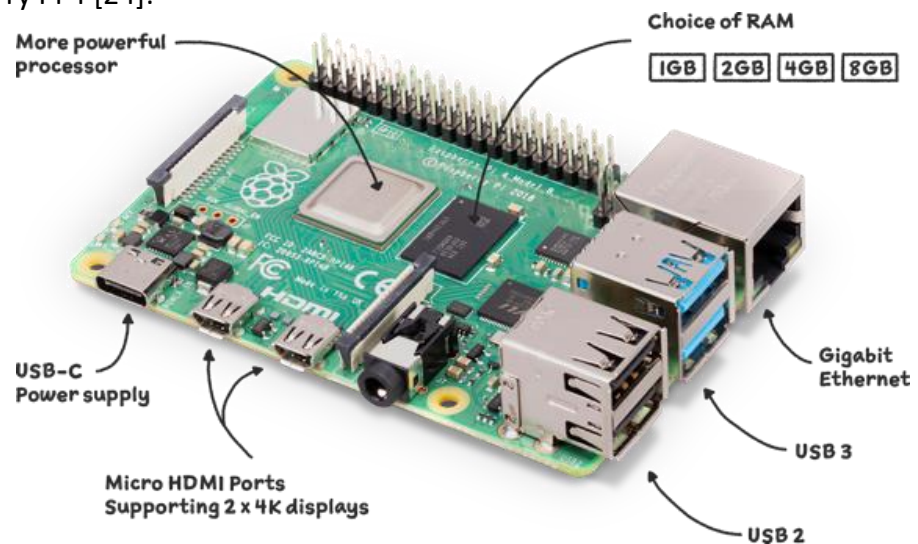


Figure 31 Raspberry Pi 4 [25]

Le Raspberry Pi 4 est un nano-ordinateur monocarte populaire et abordable, Le Raspberry Pi 4 est le modèle le plus puissant de la gamme Raspberry Pi, elle est utilisée dans une grande variété de projets, allant de l'informatique domestique, de l'automatisation résidentielle, de la robotique, de l'IoT, de l'apprentissage.

Caractéristiques [24]:

- Processeur Broadcom BCM2711 quad-core Cortex-A72 64-bit cadencé à 1.5GHz
- 2Go, 4Go ou 8Go de RAM LPDDR4
- Stockage eMMC de 32Go (modèle 8Go de RAM uniquement) ou carte microSD
- 2 ports micro HDMI pour double affichage 4K
- 2 ports USB 3.0 et 2 ports USB 2.0
- Port Ethernet Gigabit
- Wi-Fi 802.11ac et Bluetooth 5.0
- GPIO de 40 broches

- Alimentation via USB-C (5V/3A)
- Système d'exploitation Raspbian (Linux)
- Dimensions : 85 x 56 x 19.5 mm

Les pines :

Les broches GPIO (General Purpose Input/Output) sont des connecteurs à 40 broches qui permettent de contrôler des circuits électroniques externes comme des LED, des boutons, des moteurs, etc. Les broches GPIO offrent de nombreuses fonctionnalités, notamment :

- UART (Universal Asynchronous Receiver/Transmitter): Le UART permet la communication série asynchrone entre le Raspberry Pi et d'autres périphériques. Il utilise les broches GPIO 14 (TX) et 15 (RX).
- I2C (Inter-Integrated Circuit) : L'I2C est un protocole de communication série à deux fils qui permet au Raspberry Pi de communiquer avec plusieurs périphériques I2C. Il utilise les broches GPIO 2 (SDA) et 3 (SCL).
- SPI (Serial Peripheral Interface) : Le SPI est un protocole de communication série qui permet une communication rapide entre le Raspberry Pi et les périphériques compatibles SPI. Il utilise les broches GPIO 10 (MOSI), 9 (MISO), 11 (SCLK), 8 (CE0) et 7 (CE1).
- PWM (Pulse Width Modulation) : Certaines broches GPIO (12, 13, 18 et 19) prennent en charge la modulation de largeur d'impulsion matérielle.

4.2 choix de caméra à utiliser :

- Caméra IP [26]:



Figure 32 Caméra ip [26]

Caractéristique [26]:

- C'est une caméra de surveillance extérieure sans fil.
- Marque : Ctronics
- Technologie de connectivité : Sans fil, connectivité Wi-Fi
- Caractéristique spéciale : Vision nocturne infrarouge pour une surveillance dans l'obscurité
- Utilisation : Extérieure, résistante aux intempéries
- Source d'alimentation : Alimentation électrique, nécessite une prise de courant
- Protocole de connectivité : Wi-Fi, permet une connexion sans fil à un réseau Wi-Fi
- Type de contrôleur : Peut être contrôlée et surveillée à distance via une application pour Android
- Type de fixation : Montage mural, peut être fixée au mur ou au plafond
- Résolution d'enregistrement vidéo : 1080p full HD pour des images et vidéos claires.

- Webcam [27]:



Figure 33 webcam [27]

Caractéristique [27]:

- Marque : Trust
- Technologie de connectivité : USB
- Couleur : Noir
- Caractéristique spéciale : Micro intégré
- Technologie de capteur photo : Autre
- Type de caméscope : Webcam
- Nom de modèle : GXT 1160
- Composants inclus : Webcam Full HD
- Résolution d'enregistrement vidéo : 1080p
- Type de lentille : Zoom
- Résolution : 8MP (3840 x 2160)
- Streaming en direct sur : Twitch, Skype, YouTube, etc.
- Support intelligent avec pince intégrée
- Équilibre automatique des blancs
- Mise au point fixe

5. Conclusion :

Dans ce chapitre nous avons discuté sur la conception software et hardware de notre système, dans la partie software on n'a détaillé les différentes étapes pour concevoir le programme à implémenter dans la carte de développement, et dans la partie Hardware nous avons spécifié les différentes caractéristiques des cartes de développement et les cameras à utiliser, dans le prochain chapitre nous allons implémenter le programme et discuter les résultats obtenue des expériences.

CHAPITRE 3 : Réalisation

1. Introduction :

Dans ce chapitre nous nous pencherons sur les résultats obtenus. Nous présenterons les données et les métriques que nous avons collectées, en mettant en évidence les avantages et les limites de notre système. De plus, nous expliquerons les étapes nécessaires pour implémenter notre solution sur des cartes telles que le Raspberry Pi et la BeagleBone Black, qui sont couramment utilisées dans les projets d'IA embarquée.

2. Résultats d'entraînement de MobileNet :

Après avoir collecté les données et configurer notre architecture basée sur MobileNet nous avons lancé l'entraînement en 15 epochs sur les jeux de données extrait de kaggle, on n'a obtenu les résultats suivants :

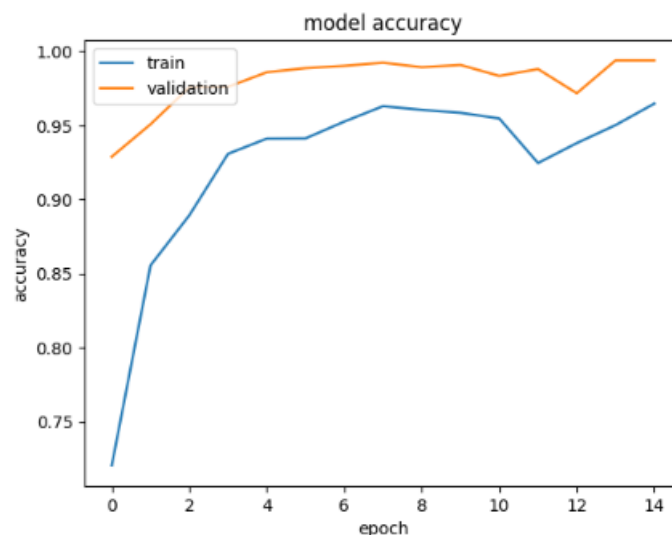


Figure 34 la précision du modèle

Les graphiques ci-dessus présentent la métrique de l'accuracy sur plusieurs epochs.

Au départ, la courbe d'entraînement augmente rapidement du premier epoch jusqu'au huitième, puis elle se stabilise jusqu'au onzième epoch. Cependant, il y a une perturbation au douzième epoch, après quoi la courbe recommence à augmenter pour atteindre une valeur de 0,98 % à la fin.

En ce qui concerne la courbe de validation, elle suit symétriquement la courbe d'entraînement et se stabilise à une valeur de 0,99 %.

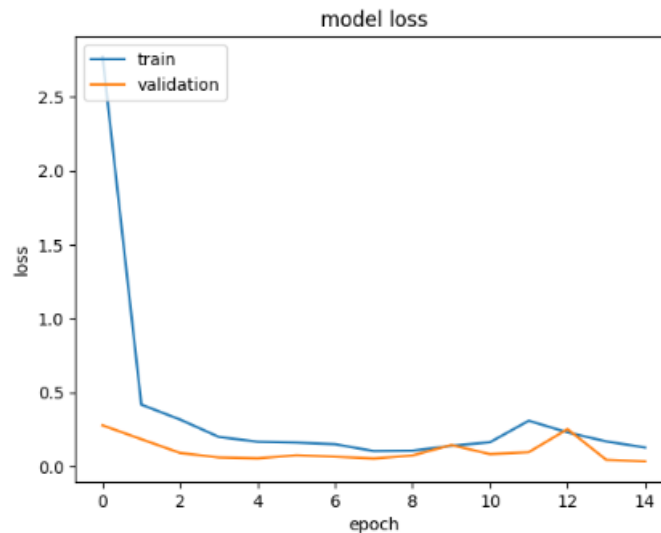


Figure 35 la fonction d'erreur du modèle

Les graphiques ci-dessus présentent la métrique de l'accuracy sur plusieurs epochs. Au départ, la courbe d'entraînement décroît rapidement du premier epoch jusqu'au huitième, puis il y a une perturbation au troisième epoch, après quoi la courbe recommence à décroître pour atteindre une valeur de 0,1 % à la fin.

En ce qui concerne la courbe de validation, elle suit symétriquement la courbe d'entraînement et se stabilise à une valeur de moins de 0,1 %.

Analyse des résultats obtenus dans les deux figures :

Les résultats obtenus jusqu'à présent sont encourageants, avec une augmentation progressive de l'accuracy et une bonne généralisation du modèle.

3. Implémentation de MobileNet sur les cartes de développements :

3.1 Implémentation sur la Beagle Bone Black :

- ❖ Conversion du modèle keras vers un modèle compatible opencv en utilisant l'outil tf2onnx

```
! pip install tf2onnx
```

```
! python -m tf2onnx.convert --saved-model mdl.pb --output mdl.onnx --opset 13
```

- ❖ Envoie du modèle convertis vers la BeagleBonBlack en utilisant l'outil fcp
- ❖ Installation de la bibiothèque OpenCV.
- ❖ Implémentation du modèle en utilisant la bibliothèque OpenCV

```
from time import time as t
import cv2
import numpy as np
# Charger le modèle ONNX
model_onnx_path = 'mdl.onnx'
net = cv2.dnn.readNetFromONNX(model_onnx_path)
img = cv2.imread('1.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (224, 224))
img = img.reshape((1, 224, 224, 3)).astype('float32')
img /= 127.5
img -= 1.0
# Préparer l'entrée pour le modèle
#input_image = cv2.imread('1.jpg')
#input_blob = cv2.dnn.blobFromImage(input_image, size=(224, 224),
#mean=(103.93, 116.77, 123.68), swapRB=True, crop=False)
classes = ["rottenbanana", "freshbanana", "rottenapples",
           "rottenoranges", "freshapples", "freshoranges"]
# Utiliser le modèle pour effectuer une prédiction
#input_blob.reshape((1, 224,))
t0 = t()
net.setInput(img)
output = net.forward()
t1 = t()
# Traiter la sortie du modèle (exemple : récupérer l'indice de la classe prédite)
index = np.argmax(output)
print(f"Indice de la classe prédite : {classes[index]} at {t1-t0}")
```

Figure 36 programme d'implémentation du modèle

Ce programme charge un modèle ONNX pré-entraîné, prépare une image d'entrée, utilise le modèle pour effectuer une prédiction et affiche le résultat avec l'indice de la classe prédite et le temps d'exécution.

3.2 Les résultats obtenus :

Taux de prédiction	17.62 secondes
Taux d'exécution	22.30 secondes
Ram	94.64 mégas octet

Selon la figure le modèle a prédit la classe de l'image en 17,62 secondes, le temps total de l'exécution du programme est de 22.30 secondes, il a utilisé 94.64 méga octets de ram, ce qui signifie que la BeagleBonBlack est moins performante pour exécuter un tel modèle de Deep Learning, car elle est dotée d'une ram de 512 mégas.

3.3 Implémentation sur la Raspberry :

- ❖ Téléchargement du système d'exploitation sur la Raspberry pi.
- ❖ Flash du système d'exploitation sur la carte mémoire de 32 giga en utilisant Raspberry imager, après avoir configuré cet outil pour autoriser les connexion SSH et la connexion du wifi automatiquement.
- ❖ Conversion du modèle keras vers un modèle compatible tensorflowLite

```
import tensorflow as tf
mdl = tf.keras.models.load_model('mdl.h5')
conv = tf.lite.TFLiteConverter.from_keras_model(mdl)
_mdl = conv.convert()
with open('mdl.tflite', 'wb') as f :
    f.write(_mdl)
```

Figure 37 programme de conversion

Ce programme charge un modèle Keras préalablement entraîné à partir d'un fichier, le convertit en un modèle TensorFlow Lite, puis enregistre le modèle converti dans un fichier. tflite.

- ❖ Chargement du modèle vers la Raspberry en utilisant l'outil SCP
- ❖ Installation de la bibliothèque tensorflowLite – runtime version 2.8.0.
- ❖ Installation de OpenCV.
- ❖ Implémentation du modèle en utilisant OpenCV et tensorflowLite

```

import cv2
from resource import gettrusage, RUSAGE_SELF
import tflite_runtime.interpreter as tfl
from time import time as t
import numpy as np
model_tflite_path = 'mdl45.tflite'
net = tfl.Interpreter(model_tflite_path)
img = cv2.imread('1.jpg')[:, :, ::-1]
img = cv2.resize(img, (224, 224))
img = img.reshape((1, 224, 224, 3)).astype('float32')
img /= 127.5
img -= 1
# Préparer l'entrée pour le modèle
classes = ["rottenbanana", "freshbanana", "rottenapples",
           "rottenoranges", "freshapples", "freshoranges"]
# Utiliser le modèle pour effectuer une prédiction
inpt = net.get_input_details()
outpt = net.get_output_details()
net.allocate_tensors()
t0 = t()
net.set_tensor(inpt[0]['index'], img)
net.invoke()
output = net.get_tensor(outpt[0]['index'])
t1 = t() - t0
# Traiter la sortie du modèle (exemple : récupérer l'indice de la classe prédite)
index = np.argmax(output)
print(f"Indice de la classe prédite : {classes[index]} at {t1} s")
print(gettrusage(RUSAGE_SELF))

```

Figure 38 programme d'implémentation du modèle

Ce code charge un modèle TensorFlow Lite, prétraite une image, effectue une prédiction en utilisant le modèle, affiche la classe prédite et le temps d'exécution, ainsi que l'utilisation des ressources système par le programme.

3.3 Les résultats obtenus :

Taux de prédiction	0.09 secondes
Taux d'exécution	0.82 secondes
Ram	96.03 mégas octet

Selon la figure le modèle a prédit la classe de l'image en 0.09 secondes, le temps total de l'exécution du programme est de 0.82 secondes, il a utilisé 96.03 méga octets de ram, ce qui signifie que la Raspberry est compatible et performante pour exécuter un tel de programme de Deep Learning, car elle est dotée d'une ram de 2 giga.

4. Implémentation de MASK RCNN sur les cartes de développements :

4.1 implémentations sur la Raspberry :

- ❖ Téléchargement du répertoire MASK RCNN depuis GitHub [28]
- ❖ Transfert du répertoire MASK RCNN vers la RASBERRY en utilisant la commande SCP.
- ❖ Installation des bibliothèques nécessaire pour le bon fonctionnement du MASK RCNN.
- ❖ Implémentation du modèle MASK RCNN.

```
model = mrcnn.model.MaskRCNN(mode="inference",  
                               config=SimpleConfig(),  
                               model_dir=os.getcwd())
```

Figure 39 Architecture du modèle MaskRCNN pour la segmentation d'instance avec une configuration simple

Ce code crée un modèle MaskRCNN pour la segmentation d'instance, en utilisant une configuration simple et en spécifiant le répertoire du modèle, ce modèle peut ensuite être utilisé pour effectuer des prédictions sur des images.

```
model.load_weights(filepath="mrcnn/mask_rcnn_coco.h5",  
                   by_name=True)
```

Figure 40 Chargement des poids pré-entraînés pour le modèle MaskRCNN à partir du fichier mask_rcnn_coco.h5

Ce code charge les poids pré-entraînés d'un modèle MaskRCNN à partir d'un fichier spécifié. Les poids pré-entraînés sont des paramètres appris à partir d'un grand ensemble de données et permettent au modèle de bénéficier de connaissances préalables pour résoudre des tâches similaires.

```
r = model.detect([image], verbose=0)
```

Figure 41 Détection d'instances sur une image à l'aide du modèle MaskRCNN

Ce code effectue la détection d'instances sur une seule image en utilisant le modèle MaskRCNN préalablement créé, La variable image contient l'image sur laquelle la détection doit être effectuée.

4.2 les résultats obtenus :

Taux de prédiction	44.39 secondes
Taux d'exécution	169 secondes
Ram	1300 mégas octet

Selon le tableau ci-dessus le modèle a prédit la classe de l'image en 44.39 secondes, le temps total de l'exécution du programme est de 169 secondes, il a utilisé 1300 méga octets de ram, ce qui signifie que la Raspberry n'est pas compatible pour exécuter un modèle de tel taille car elle est dotée d'une ram de 2 giga.

Remarque : Après l'exécution du modèle pré-entraîné sur la Raspberry, on déduit que ce modèle ne peut pas être exécuter sur le beagle Bone Black à cause de ces ressources limitées et qui sont inférieure aux ressources de la Raspberry.

4.3 choix du modèle et de carte à utiliser :

Nous avons choisi le modèle basé sur MobileNet sur plusieurs avantages :

- ❖ Léger et rapide : MobileNet est caractérisé par sa légèreté et son efficacité en termes de calculs. Il utilise des opérations de convolution profonde spécifiques qui réduisent le nombre de paramètres et de calculs nécessaires, ce qui le rend adapté aux dispositifs avec des ressources limitées en termes de mémoire et de puissance de calcul.
- ❖ Haute précision : Bien que MobileNet soit un modèle léger, il parvient à maintenir une précision raisonnable dans des tâches de classification d'images. Il a été conçu pour obtenir un équilibre entre la taille du modèle et ses performances.
- ❖ Adaptabilité : MobileNet est compatible avec différents frameworks de deep learning et peut être utilisé pour des tâches de classification d'images, de détection d'objets et de segmentation sémantique. Il peut être adapté à diverses applications sur des appareils mobiles, des systèmes embarqués et d'autres plateformes avec des contraintes de ressources similaires.

MobileNet offre une solution légère, rapide et précise pour le traitement d'images sur des appareils avec des ressources limitées. Il est adapté à une variété d'applications et peut être déployé sur une gamme de plates-formes, y compris la Raspberry Pi offrant ainsi qu'une certaine polyvalence.

5. Résultat du calcul du calibre :

Après avoir connecté la caméra à la Raspberry Pi, nous avons initié le flux vidéo pour capturer les images en temps réel. Ensuite, nous avons utilisé ces images pour effectuer le calibrage en pixels. Pour ce faire, nous avons procédé comme suit :

Mesure du calibre réel : Nous avons sélectionné un objet de référence dont nous connaissons les dimensions réelles. À l'aide d'un instrument de mesure précis, nous avons mesuré la taille de cet objet dans l'unité de notre choix, comme les centimètres.

Capture d'une image de l'objet : Nous avons placé l'objet de référence devant la caméra et capturé une image à partir du flux vidéo en cours. Cette image servira de base pour déterminer le calibre en pixels.

Calcul du rapport de calibrage : En divisant la mesure du calibre réel par le calibre en pixels, nous avons obtenu le rapport de calibrage, qui représente le nombre de pixels équivalents à une unité de mesure réelle (par exemple, le nombre de pixels par centimètre).

Ce processus de calibrage en pixels nous a permis d'établir une correspondance entre les dimensions réelles et les dimensions en pixels dans les images capturées. Cela nous a ensuite permis d'utiliser cette relation pour mesurer et analyser d'autres objets dans le flux vidéo.



6. Conclusion :

Dans ce chapitre, nous avons présenté les résultats de l'entraînement de notre modèle de reconnaissance d'images. Nous avons ensuite décrit les étapes nécessaires pour déployer ce modèle sur des cartes de développement, telles que la Raspberry Pi. Enfin, après avoir évalué plusieurs options, nous avons opté pour le modèle MobileNet et la carte Raspberry Pi, car cette combinaison a démontré à la fois une grande précision et une bonne compatibilité.

En résumé, ce chapitre a permis de valider notre approche en matière d'apprentissage automatique pour la reconnaissance d'images. Nous disposons désormais d'un modèle entraîné et optimisé, prêt à être intégré dans une solution embarquée.

7. Conclusion générale :

7.1 Synthèse :

L'agriculture joue un rôle essentiel dans notre société en nous fournissant la nourriture dont nous avons besoin. Les agriculteurs cultivent et récoltent des produits alimentaires tels que les fruits et les légumes que nous consommons chaque jour, ce qui nécessite un traitement (classification) généralement fait manuellement. Dans le cadre de ce mémoire, notre objectif était de développer un système intelligent pour la classification des fruits selon la qualité et le calibre.

Pour réaliser ce travail, nous avons d'abord fait une étude sur les travaux déjà réalisés en utilisant l'intelligence artificielle dans le domaine agricole, ce qui nous a permis d'avoir une vision globale sur les algorithmes et les jeux de données utilisés dans ce domaine.

Ensuite, nous avons entamé la partie conception où nous avons expliqué d'une façon globale les différentes parties du fonctionnement de notre système, puis nous avons détaillé la partie qui se charge de la détection et du calcul du calibre.

Enfin, nous avons discuté les résultats de l'implémentation des modèles sur les cartes de développement, et nous avons opté pour l'architecture MobileNet et la carte Raspberry PI pour la classification des fruits, car elle a démontré son efficacité et ses performances.

7.2 Perspectives :

Dans ce travail, nous nous sommes concentrés sur la partie détection et calcul du calibre. Par contrainte de temps, nous n'avons pas pu aborder les autres parties du système telles que le tapis roulant et le système de direction. Dans le cadre de travaux futurs, nous allons intégrer la partie détection et calcul du calibre sur le tapis roulant, et nous allons développer un système de direction pour diriger les fruits selon la qualité et le calibre.

Bibliographie

- [1 I. C. AgroTIC, DEEP LEARNING.
]
- [2 Z. G. F. D. B. U. T. P. a. C. M. Inkyu Sa *, «DeepFruits: A Fruit Detection System Using Deep,» 2016.
]
- [3 K. W. 2. A. K. 3. Zhenglin Wang 1, «Mango Fruit Load Estimation Using a Video Based MangoYOLO-Kalman Filter-
] Hungarian Algorithm Method».
- [4 S. S. S. P. V. R. R. C. S. B. Y. Dasari, «Fresh and Rotten Fruits Classification Using CNN and Transfer Learning,»
] November 2020.
- [5 M. L. K. O. Aiadi, «A new method for automatic date fruit classification,» August 2017.
]
- [6 [En ligne]. Available: <https://www.agriexpo.online/fr/prod/krebeck-gmbh-stalleinrichtungen-und-apparatebau/product-171092-79863.html>.
]
- [7 «product-180742-45989.html,» [En ligne]. Available: <https://www.agriexpo.online/fr/prod/greefa/product-180742-45989.html>.
]
- [8 [En ligne]. Available: <http://www.hnmiracle.org/news/the-delivery-of-the-red-date-sorting-machine-47414220.html>.
]
- [9 M. Shivanandhan. [En ligne]. Available: <https://www.freecodecamp.org/news/deep-learning-frameworks-compared-mxnet-vs-tensorflow-vs-dl4j-vs-pytorch/?fbclid=IwAR3V08-29rV9sOBmMpZFgyQ1-oHJDcbpUqLGIWrLaPcAPaHxnuimOIP4P6U>.
]
- [1 «mobilenet,» [En ligne]. Available: <https://roboflow.com/model/mobilenet-v2-classification?fbclid=IwAR0g76YmkNutR1gxOXKzsqrJxKk2UrThblJKuDexCe3vbXj1MGrrquGYWQ8>.
0]
- [1 R. Khandelwal. [En ligne]. Available: <https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-7983502fcad1>.
1]
- [1 kaggle, «datasets,» [En ligne]. Available: <https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification>.
2]
- [1 «fonction de calcul d'erreur,» [En ligne]. Available:
3] https://keras.io/api/losses/probabilistic_losses/#categorical_crossentropy-class.
- [1 «adam,» [En ligne]. Available:
4] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam?fbclid=IwAR1zuLdLwN4_2pFzWaU69TCSLemUcWgsoEeg6_0XzAezNTnWFSaAZdHUcy0.
- [1 M. P. Dr Bernard GIUSIANO, Deep Learning 2, Aix Marseille.
5]
- [1 «reboflow,» [En ligne]. Available: <https://public.roboflow.com/>.
6]
- [1 «open cv,» [En ligne]. Available: <https://opencv.org/>.
7]
- [1 «scikit-image,» [En ligne]. Available: <https://scikit-image.org/>.
8]
- [1 «open cv vs scikit,» [En ligne]. Available: <https://stackshare.io/stackups/opencv-vs-scikit-image?fbclid=IwAR3QFmg9n-Amr1szQ4e6qZrAeHSeTOGVjAJKkvlQH329EIE7twJ1m80i6Kk>.
9]
- [2 «HSV et RGB,» [En ligne]. Available: https://discover.hubpages.com/technology/Color-spaces-RGB-vs-HSV-Which-one-to-use?fbclid=IwAR0_5EXxIAmxTbtcrUXhI1gq49KIJOoOoY0rj9rBX5DIS_mcbuYOOi4xVtw.
0]
- [2 «otsu,» [En ligne]. Available: <https://theailearner.com/tag/otsu-method-opencv/>.
1]

- [2 «find contour,» [En ligne]. Available: https://docs.opencv.org/3.4.3/d9/d8b/tutorial_py_contours_hierarchy.html.
2]
- [2 «beagle bone black,» [En ligne]. Available: <https://beagleboard.org/black>.
3]
- [2 «rasbery,» [En ligne]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>.
4]
- [2 «rasberry-pi-4-model-b,» [En ligne]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>.
5]
- [2 «amera ip,» [En ligne]. Available: https://www.amazon.fr/Surveillance-Ext%C3%A9rieure-Mouvements-Automatique-Bidirectionnel/dp/B08TWDXYSC/ref=sr_1_5?__mk_fr_FR=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=2X6KIUGR4ZTTL&keywords=Cam%C3%A9ra+IP+WiFi+1080P+HD+Outdoor&qid=1687117093&s=electronics&spre.
6]
- [2 «webcam,» [En ligne]. Available: https://www.amazon.fr/Trust-Gaming-22397-Webcam-7int%C3%A9gr%C3%A9/dp/B0784BWH3H/ref=mp_s_a_1_3?keywords=webcam&qid=1687117830&sr=8-3&fbclid=IwAR0otp0h4Qdp3gsC8DZO7NEzqz6mXvbY2U6YCyGCYNJo3GWcQYbGm71mMI8.
7]
- [2 «mrcnn,» [En ligne]. Available: <https://github.com/mrk1992/mask-rcnn-tf2-us/blob/main/mrcnn/>.
8]
- [2 «product-171092-79863.html,» [En ligne]. Available: <https://www.agriexpo.online/fr/prod/krebeck-gmbh-9stalleinrichtungen-und-apparatebau/product-171092-79863.html>.
9]
- [3 H. Z. X. W. a. C. C. Hanwen Kang, «Real-Time Fruit Recognition and Grasping Estimation for Robotic Apple
0] Harvesting,» 2020 Oct 4.