

Table des matières

1 Cadre théorique	5
1 réseaux de neurones biologiques et artificielles	6
2 perceptron	7
3 Apprentissage de réseaux neuronals	8
3.1 Types d'apprentissage	8
3.2 Processus d'apprentissage supervisé de réseaux neuronals	8
3.2.1 Fonction d'activation	9
3.2.2 propagation vers l'avant ou Feed forward propagation	10
3.2.3 Erreur et cost function	10
3.2.4 Gradient descent	11
3.2.5 Rétropropagation	12
3.2.6 Batch et Batch size	13
3.2.7 Epochs	13
4 Regression et classification	13
5 réseau de neurones convolutifs	14
5.1 différence avec les réseaux neuronales classiques	14
5.2 L'image comme entrée	14
5.3 Architecture	15
5.3.1 Couche convolutive	16
5.3.2 Partie de classification	19
6 Apprentissage par transfert	20
7 vision par ordinateur	21
7.1 vision par ordinateur classique	21
7.2 vision par ordinateur moderne	21
7.3 Pipeline de vision par ordinateur	22
7.4 taches de cv	23
7.4.1 classification	23
7.4.2 Détection d'objets	26
7.5 OPENCV	30
7.6 introduction	30
7.7 Quelques image processing techniques avec opencv	30
2 Implémentation	34
8 étude	35

Introduction générale

Nul ne peut nier que la vie humaine est devenu de plus en plus liée à l'informatique et par suite la qualité de vie évolue de façon rapide et révolutionnaire. Une grande partie de cette évolution technologique est due à l'intelligence artificielle et ses applications dans la plupart des domaines de vie tel que :

- Médecine : Des machines capables de détecter les maladies et d'identifier les cellules cancéreuses et même capable de proposer des traitements aux malades jouant ainsi le rôle d'un médecin.
- Transport : Des voitures ou "Self-Driving cars" sont maintenant capables de circuler et détecter les signes de traffic indépendamment de l'intervention humaine
- Finance : Les banques et les bourses sont devenues capables d'analyser des données gigantesques plus efficacement et par suite ont réussi à obtenir une vue correcte et précise de leurs données avec un faible niveau d'erreur.

En tunisie, l'une des domaines qui n'a pas subi l'influence de l'informatique est le traffic routier. C'est dans ce cadre que s'inscrit ce projet de fin d'étude . Dans ce travail, Nous allons réaliser un système automatisé qui permet le contrôle des véhicules en temps réel et qui permet d'indiquer les voitures déclarées volées ou celles dont les papiers ne sont pas en règle. Nous allons dans un premier chapitre faire une étude sur des concepts et des termes parmi lesquelles nous citons :

- apprentissage à profond ou Deep Learning
- vision par ordinateur ou Computer Vision
- Transfer learning

Une fois, nous devons familiarisés avec ces concepts on passe au deuxième chapitre comprenant un étude préalable et l'architecture de solution proposée pour arriver en fin au dernier chapitre où nous montrons la réalisation de projet.

Chapitre 1

Cadre théorique

1 réseaux de neurones biologiques et artificielles

Les chercheurs ont essayé de comprendre le fonctionnement du cerveau humain et la façon dont il prend ses décisions et par suite simuler ce fonctionnement en créant un modèle mathématique du cerveau et l'intégrer dans les machines. En effet, le cerveau humain n'est qu'un réseau de neurones biologiques interconnectés. Les neurones interagissent avec leurs voisins en connectant les extrémités des axones d'une cellule aux dendrites d'une autre cellule, grâce à des points spécifiques appelés synapses. Si la somme des signaux d'entrée dans un neurone dépasse un certain seuil, alors le neurone envoie un signal d'action à l'axone puis transmet ce signal électrique le long de l'axone. Les réseaux de neurone artificielles simulent le mécanisme d'action des neurones vitaux du cerveau. Cela signifie qu'un réseau de neurones sera constitué d'un certain nombre de noeuds connecté entre eux et que les neurones artificiels imitent les neurones biologiques dans la façon dont ils génèrent et transmettent des signaux entre eux, et le réseau neuronal dans son ensemble doit imiter le travail du cerveau en termes de sa capacité à apprendre et acquérir de l'expérience. Plus l'expérience acquise, le plus fort soient les synapses entre les neurones.

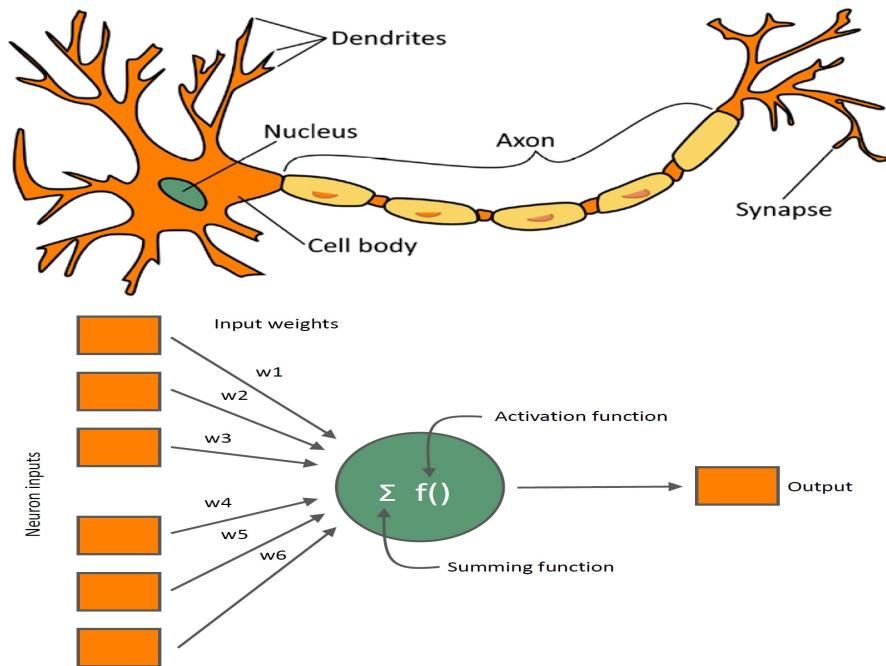


FIGURE 1.1 : Comparaison entre les cellules neuronales biologiques et artificielles

2 perceptron

On distingue 2 types :

- monocouche : C'est le premier réseau de neurones artificiels de l'histoire et son idée est la base de l'apprentissage en profondeur que nous connaissons aujourd'hui. Comprendre le simple réseau perceptron est nécessaire pour comprendre les réseaux de neurones actuellement utilisés. C'est un réseau neuronal monocouche dont l'architecture consiste de 4 composants :
 - valeurs d'entrées indépendants
 - des poids et des bias
 - somme nette
 - fonction d'activation

Single Layer Perceptron

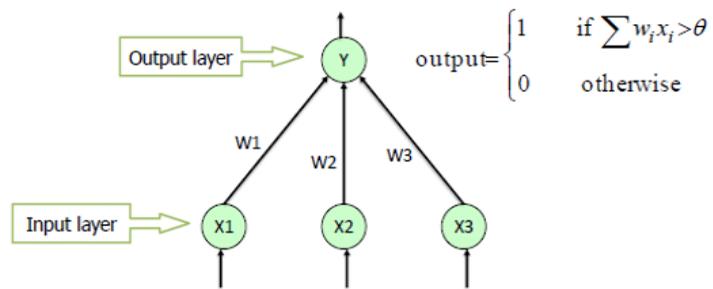


FIGURE 1.2 : -perceptron monocouche

Comme indiqué dans la figure, chaque valeur d'entrée est multipliée par son poids, ensuite une opération d'addition de tous les valeurs multipliées aura lieu pour donner un somme pondérée qui à son tour sera passé comme paramètre à une fonction d'activation produisant la sortie du perceptron. La fonction d'activation donne 0 comme résultat si la somme est inférieure ou égale à 0 sinon le résultat sera 1.

- multicouche ou Feed Forward Neural Network : Ce type consiste de plusieurs couche de neurones . Considérons par exemple un perceptron à 3 couches, la première couche sera la couche recevant les valeurs d'entrée, la couche intermédiaire sera la couche cachée et la dernière sera la couche de sortie. Chaque neurone d'une couche passe sa sortie vers les neurones de la couche suivante.

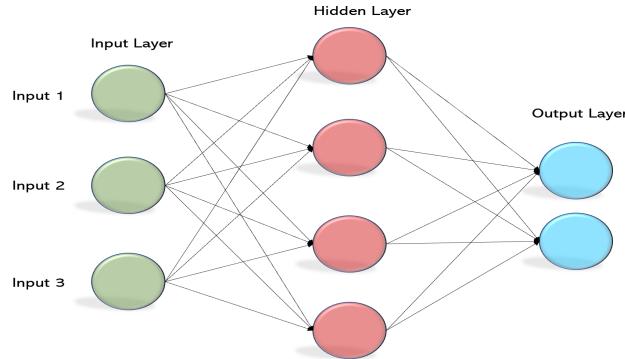


FIGURE 1.3 : perceptron multicouche

Nous pouvons augmenter le nombre de couches cachées autant que nous le souhaitons, pour rendre le modèle plus complexe en fonction de notre tâche.

3 Apprentissage de réseaux neuronaux

3.1 Types d'apprentissage

Les méthodes d'apprentissage d'un réseau de neurones se divisent en deux parties majeures à savoir :

- apprentissage supervisé : Ce type est basé sur l'idée de passer des données d'entraînement au réseau sous la forme d'une paire de valeurs, qui sont la valeur d'entrée et la valeur cible
- apprentissage non supervisé : Nous passons seulement les valeurs d'entrée au réseau neuronal sans y montrer d'exemples de ce qu'elle doit produire. Ici , le réseau détecte indépendamment les caractéristiques des formes et formats qui lui sont présentés et développe une représentation interne de ces formes sans connaissance préalable.

3.2 Processus d'apprentissage supervisé de réseaux neuronaux

L'apprentissage de réseau de neurone semble à la façon dont nous apprenons à faire les choses dans la vie courante. Prenons l'exemple d'un élève à l'école qui rédige des réponses fausses sur un sujet particulier. L'élève est ajusté et guidé plusieurs fois par son maître afin d'être capable de répondre correctement la prochaine fois sur le même sujet. De la même façon, le réseau de neurone a besoin d'un superviseur ou un entraîneur qui va montrer au réseau des exemples de ce que doivent être les résultats de prédiction face à une série de valeurs

d'entrée. Ces exemples peuvent être des images, texte ou chiffres. En effet le réseau neuronal artificiel n'est qu'un modèle paramétrique. Donc il est nécessaire de trouver les meilleures valeurs de paramètres de poids et bias pour que le modèle donne les résultats attendus par le créateur de ce réseau. En outre, l'entraînement de ce modèle nécessite une mesure de performance et un algorithme qui fait une optimisation mathématique pour l'amélioration de cette mesure. Le processus d'entraînement se déroule comme suit : Après sa création, le réseau de neurones commence avec des poids (w_0, w_1, \dots, w_n) et bias (b_0, b_1, \dots, b_n) aléatoirement initialisés portant des valeurs compris entre 0 et 1. Dès qu'on passe le premier exemple de valeurs de notre série d'exemples (x_0, x_1, \dots, x_n) au réseau chaque neurone de première couche calcule sa sortie $y = f(x) = wx + b$ et passe le résultat au fonction d'activation

3.2.1 Fonction d'activation

Cette fonction fait une transformation non linéaire à la sortie de la fonction $f(x)$ et décide si cette sortie va être transmis ou non vers les neurones connectés de la couche suivante . On distingue plusieurs types de fonction d'activation :

- Sigmoid :

Les valeurs de sortie sont comprises entre 1 et 0, ce qui rend ce type une solution idéale pour les modèles qui doivent produire une valeur de probabilité comme sortie, car les valeurs de probabilité sont toujours dans l'intervalle [0,1]. Cette fonction a la forme . Si sa sortie est supérieure à 0 donc pas d'activation de neurone c'est à dire la valeur de sortie sera passé aux neurones de couche suivante mais on n'aura pas d'activation si la sortie est 0. La fonction Sigmoid est préférable d'opérer en cas de classification binaire au niveau de neurones de couche de sortie

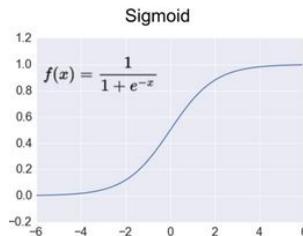


FIGURE 1.4 : Fonction d'activation Sigmoid

- Unité linéaire rectifiée(ReLU) : Cette fonction souvent implémenté dans les couches cachées du réseau de neurones.Si l'entrée est positive elle sera converti en 1 sinon elle sera converti en zéro ce qui résulte de la non activation de neurone. Cela signifie qu'à la fois, seuls quelques neurones sont activés, ce qui rend l'entraînement de réseau rapide et efficace. Cette fonction prend a forme

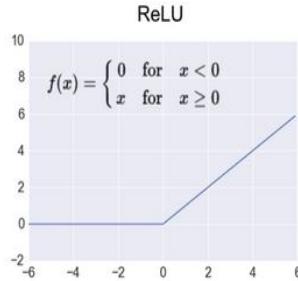


FIGURE 1.5 : Fonction d'activation ReLU

- Tanh ou Fonction hyperbolique tangente : Elle convert l'entrée en une valeur compris entre -1 et 1. Elle est souvent utilisé dans les couches intermédiaires d'un réseau neuronal. Sigmoid et tanh peuvent être dérivés l'un de l'autre

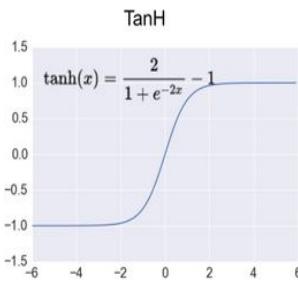


FIGURE 1.6 : Fonction d'activation Tanh

3.2.2 propagation vers l'avant ou Feed forward propagation

Chaque neurone d'une couche (d'entrée ou intermédiaire) calcule sa sortie. Si la fonction d'activation a activé le neurone, le résultat de l'activation va être la valeur d'entrée pour tous les neurones connectés de la couche suivante et va être multiplié par le poids de chacun de ces neurones. Ce processus continue pour tous les couches jusqu'on arrive à la couche de sortie. La valeur de sortie de chaque neurone de cette couche présente la probabilité que la classe associé à cet neurone.

3.2.3 Erreur et cost function

L'erreur est la différence entre prédiction et la valeur cible. Nous utilisons cette métrique pour évaluer la performance de réseau neuronal et à quelle degré les prédictions sont proches de valeurs réels.

3.2.4 Gradient descent

Maintenant que nous avons calculer la 'cost fonction', il est nécessaire d'adopter une stratégie d'optimisation pour minimiser l'erreur au maximum. C'est là où vient le nom de la Descente de gradient à la portée. Mais que signifie le terme "Gradient". Lex Fridman l'un des professeurs de MIT (1) explique "Le gradient mesure à quel point la sortie d'une fonction change si vous modifiez un peu les entrées".

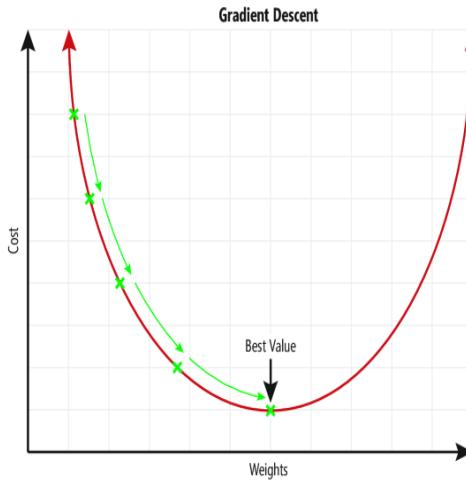


FIGURE 1.7 : déscente de gradient

En fait, Le gradient descent est un algorithme itératif d'optimisation qui sert à trouver le minimum local d'une fonction. A partir d'un poids aléatoire, le gradient commence à descendre pas à pas jusqu'à ce que la fonction du coût soit minimale à un poids particulier. La vitesse ou l'intensité de descente est contrôlée par un taux d'apprentissage ou 'learning rate' qui cause une descente rapide si sa valeur est grande et dans ce cas le réseau aura une dégradation au niveau de performance sinon la descente sera lente et par suite le réseau va apprendre presque correctement mais dans une durée longue d'où la nécessité de spécifier une valeur ni trop grande ni trop petite.

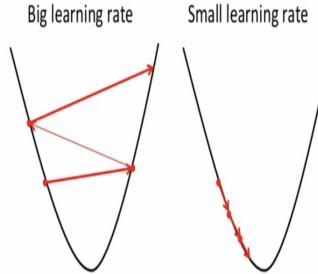


FIGURE 1.8 : L'influence de taux d'apprentissage sur le gradient déscente

Quand le gradient ne peut plus diminuer la valeur de fonction de coût à grande chose, on conclut que le réseau neuronal converge.

3.2.5 Rétropropagation

C'est une méthode de mise à jour des poids de tous les neurones d'un réseau partant de couche de sortie vers les couches précédentes dans une direction contraire de propagation vers l'avant.

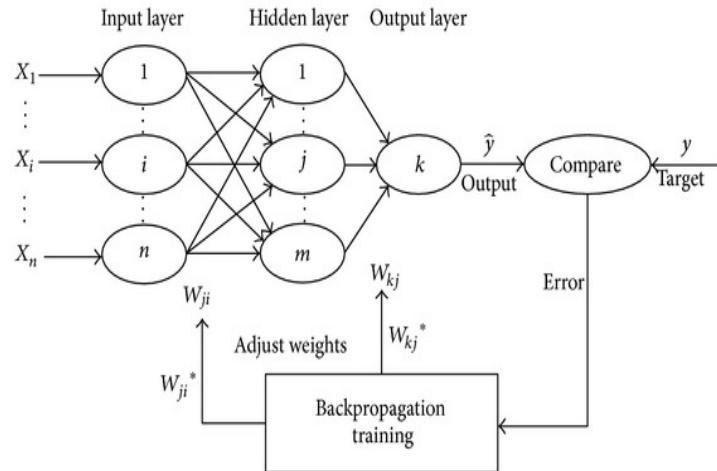


FIGURE 1.9 : rétropropagation

On peut décrire le principe de rétropropagation selon le schéma ci-dessus :

- Les valeurs d'entrée sont passées au réseau via la couche d'entrée
- Chaque neurone du réseau neuronal traite les données d'entrée avec les

valeurs résultantes des neurones de la couche précédente jusqu'à ce qu'un résultat soit généré par la couche de sortie.

- La valeur de sortie du réseau est comparée à la sortie attendue pour cette entrée. Le résultat est une valeur d'erreur qui représente l'écart entre l'entrée donnée et la valeur attendue. En se basant de cette valeur d'erreur, les poids de connexion dans le réseau sont ajustés progressivement, en partant de la couche de sortie, à travers les ou la couche cachée, et à la couche d'entrée, jusqu'à ce que la sortie correcte soit produite. Cette façon de mettre à jour des poids a pour effet d'enseigner au réseau comment produire la sortie correcte pour un entrée particulière.

3.2.6 Batch et Batch size

La taille du lot est un hyperparamètre de descente de gradient qui contrôle le nombre d'échantillons d'apprentissage à traiter avant la mise à jour des paramètres internes du modèle. Le nombre d'époques est un hyperparamètre de descente de gradient qui contrôle le nombre de passages complets à travers l'ensemble de données d'apprentissage.

3.2.7 Epochs

Un époque est achevé après faire un propagation en avant suivie par une rétro-propagation

4 Regression et classification

Ces techniques, faisant partie de l'apprentissage automatique supervisé ont comme objectif de faire des prédictions après avoir été entraînés sur des données connues. La modélisation prédictive peut être expliquée comme un problème mathématique consistant à trouver la meilleure fonction d'affectation (f) pour mapper les entrées (x) aux sorties (y), c'est à dire $f(x)=y$.

- Regression : Un algorithme de regression essaie de figurer une fonction (f) qui mappe une série de valeurs d'entrée à une série de valeurs cibles numériques ou continues. La regression peut être employé dans des cas où on veut savoir le prix d'une maison à partir ses caractéristiques tel que la surface, nombre d'étages et la distance au centre de ville. La regression linéaire et forêt d'arbres décisionnels sont des algorithmes de regression.
- Classification : Dans la classification, la sortie y est une catégorie, signifiant que la sortie de la classification est catégorielle. La classification peut être binaire, telle que la classification des e-mails en spam ou non spam, et il peut s'agir d'une classification multiple, telle que la classification des formes géométriques en rectangle ou triangle ou cercle et dans ce cas les trois formes peuvent par exemple être représentés par les nombres 0,1

et 2 respectivement. L'arbre de décision et le KNN sont des exemples d'algorithmes de classification.

5 réseau de neurones convolutifs

Les réseaux de neurones convolutifs repérentent un genre de réseaux de neurones artificiels. Ce type est un choix incontournable pour traiter des données d'entrée 2D ou 3D. Ces réseaux ont certaines caractéristiques qui imitent la façon dont l'oeil humain remarque des modèles particuliers dans les scènes et objets qu'elle observe dans la vie.

5.1 différence avec les réseaux neuronales classiques

Les réseaux de neurones classiques (feed forward neural network) reçoit une entrée et la passe à travers une suite de couches intermédiaires dont chaque couche est entièrement connectée à tous les neurones de la couche précédente. Le type d'entrée ne peut pas être assumé. Les neurones d'une couche cachée sont connectés à une petit nombre de neurones de la couche suivante au lieu de se connecter à tous les neurones. La sortie de réseau convolutif consiste d'un seul vecteur de scores de probabilités. L'entrée d'un tel réseau est explicitement une image.

5.2 L'image comme entrée

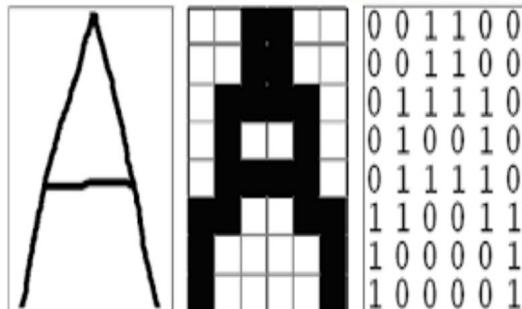


FIGURE 1.10 : L'image comme vue par l'ordinateur

Il est très important de comprendre à quoi l'image ressemble. Pour un ordinateur, une image est une matrice tridimensionnelle de valeurs compris entre 0 et 255. Ces valeurs ne sont que des pixels de couleur. L'image consiste detrois canaux de couleur si les images sont colorisés sinon on parle d'un seul canal si par exemple l'image est au niveau de gris.

5.3 Architecture

L'architecture de réseau convolutif ressemble à une collection de composants mis dans un ordre particulier. Les composants peuvent à leur tour être répétés plusieurs fois et peuvent également former ensemble des blocs de composants.

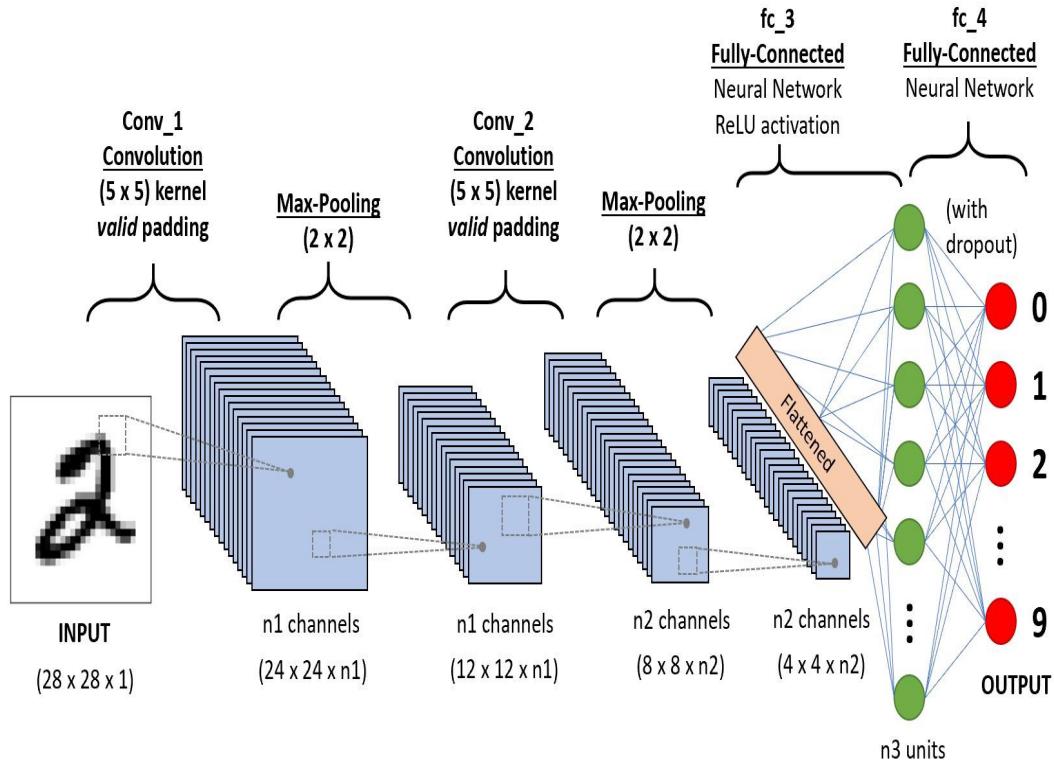


FIGURE 1.11 : L'architecture de réseau convolutionnel

En effet, pour arriver à avoir une prédiction d'un réseau convolutif , deux étapes majeures se déroulent :

- Convolution et Feature extraction : Ici des opérations de covolutions se déroulent afin d'analyser l'image et par conséquence identifier les différentes aspects et caractéristiques dans un processus appelé Extraction de caractéristiques
- Classification : Une couche entièrement connectée qui prédit la classe de l'image en fonction des caractéristiques extraites lors de l'étape de convolution.

5.3.1 Couche convulsive

- Filtre convolutif : La couche de convolution applique un filtre sur la matrice d'entrée dont le but est de chercher certains motifs ou attributs dans l'image, En fait plusieurs filtres peuvent être utilisés pour extraire différents caractéristiques ou Features. Le taille de matrice de filtre est petit pour parcourir l'ensemble de l'image et appliquer des opérations arithmétiques particuliers entre les valeurs du filtre et les pixels afin de prélever des attributs.

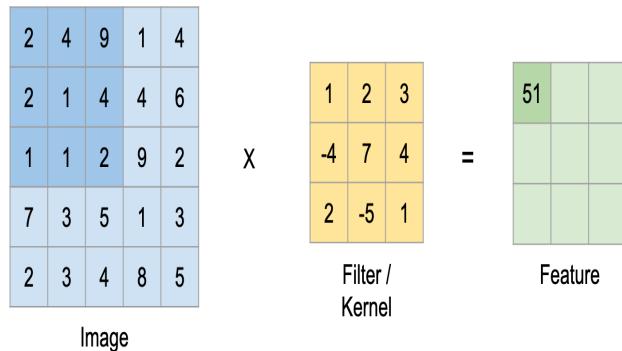


FIGURE 1.12 : fonctionnement d'un filtre

Comme indiqué précédemment les filtre sont conçus pour définir lignes,les bords et les couleurs spécifiques au sein de l'image d'entrée, tels que la définition de la ligne courbe dans la figure suivante,

- Stride : Stride contrôle la façon dont le filtre parcourt l'image après chaque opération de convolution. Lorsque sa valeur est de 1, le filtre se déplace un pixel à la fois. Lorsque la valeur de stride est 2, le filtre transit 2 pixels à la fois et ainsi de suite.
- padding : Prenons l'exemple d'une image en niveaux de gris de dimensions ($n \times n$) et un filtre de convolution de dimensions ($f \times f$). l'image résultante d'une opération de convolution a comme dimensions $(n - f + 1) \times (n - f + 1)$. Alors la taille de l'image se réduit dans durée courte prévenant d'avoir un nombre satisfaisant d'opérations de convolutions ce qui affecte le processus d'extraction de caractéristiques. On conclut que pour avoir une analyse plus précise de l'image d'entrée, il est préférable d'entourer l'image avec au moins une couche dont chaque élément ayant zéro comme valeur afin d'éviter des problèmes de performance.

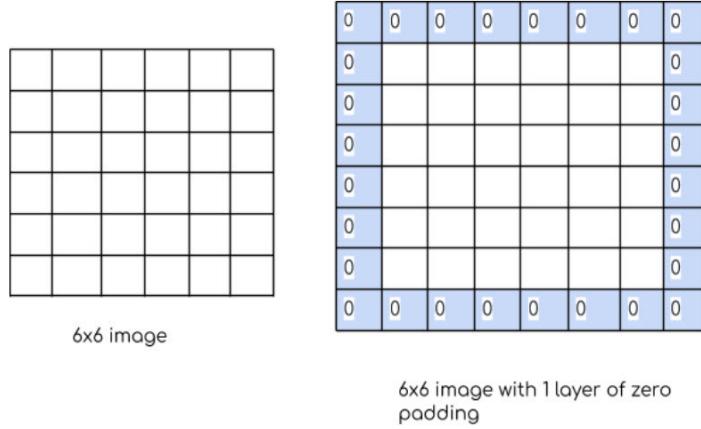


FIGURE 1.13 : Principe de Padding

- carte de caractéristiques de sortie ou Feature map : chaque filtre et le convoluons sur le volume d'entrée pour obtenir une seule carte d'activation. Les cartes de caractéristiques capturent le résultat de parcourt des filtres sur l'image d'entrée. L'utilité de carte de caractéristiques est de capturer les caractéristiques détectés. Les dimensions de cette carte sont les mêmes de filtre.
- pooling : La couche de pooling permet la diminution de taille des cartes d'activation. Cela réduit non seulement quantité de paramètres et de calcul nécessaire dans le réseau, mais empêche le réseau de tomber dans un état de surapprentissage ainsi que certaines attributs détectés après l'utilisation des filtres devient plus robustes et par suite on n'a pas à se soucier de son endroit dans l'image. Le pooling peut être réalisé par l'emploi de l'une des deux méthodes suivantes :
 - Max pooling : Spécifie les valeurs maximales dans chaque carte d'activation. En effet, il suffit de faire glisser une petite fenêtre pas à pas sur toutes les parties de l'image et de prendre la valeur maximum de cette fenêtre à chaque pas.
 - average pooling : Calcule la moyenne arithmétique de l'ensemble de valeurs dans une fenêtre..

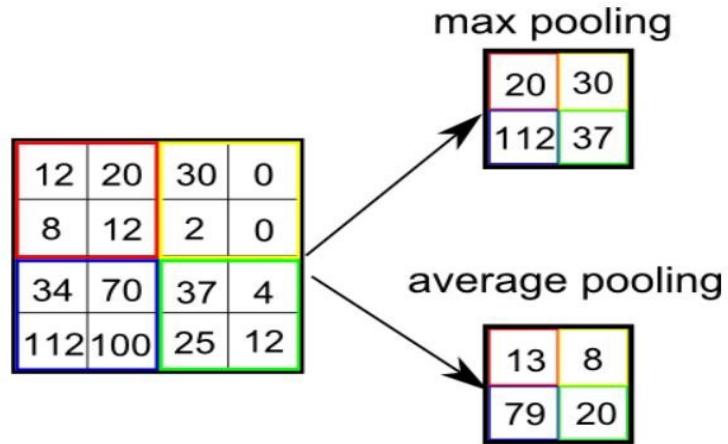


FIGURE 1.14 : Principe de pooling

En effet, le Max-pooling est la technique la plus utilisée. Elle réduit la taille de carte d'activation et maintient les valeurs les plus élevées dans chaque carte d'activation, en réduisant la taille de la carte. Les deux figures suivantes expliquent les deux fonctions de pooling.

- ReLu : Une pièce importante de l'architecture de réseau convolutif est l'Unité linéaire rectifiée ou ReLU. Cette fonction d'activation reçoit la matrice résultante de couche de pooling et remplace chaque valeur négatif par zéro.
- Flatten : L'opération d'aplatissement convertit tout l'ensemble de matrices résultantes de la partie convolutive en un seul tableau à une dimension pour les passer à la couche entièrement connectée responsable de la classification.

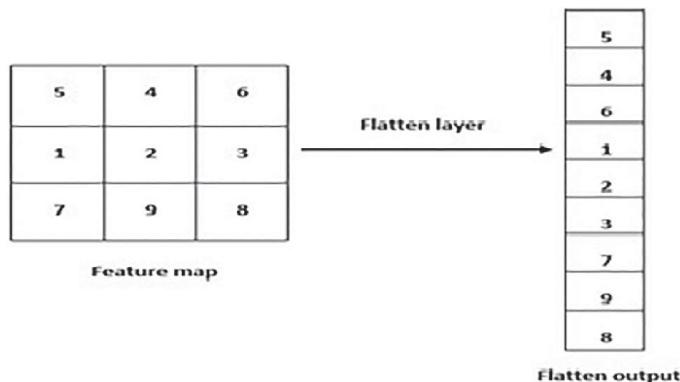


FIGURE 1.15 : Opération d'aplatissement

La partie convolutionnelle peut être construite en keras comme suit :

```
[ ] model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

FIGURE 1.16 : construction de partie d'extraction d'attributs avec keras

5.3.2 Partie de classification

C'est la dernière dans l'ordre de composants d'un réseau convolutif. Ce n'est qu'un perceptron multicouche dont les neurones sont entièrement connectés à tous les noeuds de la couche précédente. La classification de l'image d'origine est faite par cette partie. L'entrée de cette couche sera le vecteur des caractéristiques produit par la couche de flatten et la sortie sera un vecteur de score de probabilités. Les poids de cette couche jouent un rôle crucial dans le processus de vote pour la classe ayant la plus grande probabilité. La partie de classification peut être construite en keras comme suit :

```
[ ] model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

FIGURE 1.17 : construction de partie de classification avec keras

- La couche Dropout est un masque qui annule la contribution de certains neurones vers la couche suivante et laisse tous les autres inchangés. Nous pouvons appliquer une couche Dropout au vecteur d'entrée, auquel cas elle annule certaines de ses fonctionnalités ; mais nous pouvons également l'appliquer à une couche cachée, auquel cas il annule certains neurones cachés. Les couches d'abandon sont importantes dans l'entraînement des CNN car elles empêchent le surapprentissage des données d'entraînement

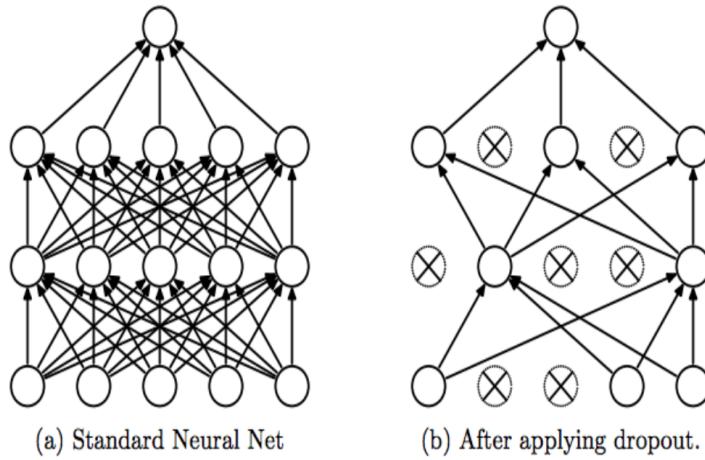


FIGURE 1.18 : Dropout

6 Apprentissage par transfert

Le processus d'entrainer un réseau de neurones sur des grands "dataset" peut prendre une longue durée allant jusqu'à des semaines. Ce temps immense d'entraînement peut être réduit si on fait recours aux poids modèles pré-entraînés précédemment en appliquant l'apprentissage par transfert. L'apprentissage par transfert consiste à partir d'un modèle pré-entraîné, il n'y aura pas besoin d'entraîner un nouveau modèle à partir de zéro. Les poids obtenus à partir des modèles pré-entraînées peuvent être réutilisés dans d'autres tâches de vision par ordinateur. Un modèle pré-entraîné a subi un entraînement sur l'un des "dataset" de volume massif. Ces modèles peuvent être utilisés directement pour faire des prédictions sur de nouvelles tâches ou intégrés dans le processus de formation d'un nouveau modèle. L'inclusion des modèles pré-entraînés dans un nouveau modèle réduit le temps d'apprentissage et l'erreur de généralisation. L'apprentissage par transfert est particulièrement utile au cas où nous n'avons qu'un dataset de petit taille. Dans ce cas, il sera utile d'utiliser les poids des modèles pré-entraînés pour initialiser les poids du nouveau modèle.

En général , L'apprentissage par transfert consiste à utiliser un modèle développé pour une tâche comme point de départ d'un modèle sur une autre tâche.



```
model = tf.keras.applications.MobileNet(  
    input_shape=None,  
    alpha=1.0,  
    depth_multiplier=1,  
    dropout=0.001,  
    include_top=True,  
    weights="imagenet",  
    input_tensor=None,  
    pooling=None,  
    classes=1000,  
    classifier_activation="softmax",  
)
```

FIGURE 1.19 : Exemple en keras d'apprentissage par transfert avec le modèle Mobilenet

7 vision par ordinateur

7.1 vision par ordinateur classique

La vision par ordinateur classique compter sur un effort manuel important afin d'élaborer des techniques de classification. Pour reconnaître une table, on doit indiquer explicitement à la machine qu'une table possède certains caractéristiques ou attributs qui sont principalement 4 pieds et une surface horizontale et chacun de ces composants consiste d'un arrangement particulier de pixels. Pour une large période de temps, les ingénieurs ont suivi ce processus traditionnel de traduire manuellement les composants d'objets de vie en des formes particuliers de pixels et les passer au machine pour qu'elle soit capable de reconnaître correctement ces objets. Cet approche est rigide, inflexible et difficile à développer.

7.2 vision par ordinateur moderne

Les réseaux neuronels convolutifs ont révolutionnés la vision par ordinateur en éliminant une grande partie de rôle de développeur. Au lieu d'indiquer explicitement à la machine les caractéristiques d'une fleur ou d'un animal, on construit un réseau neuronel et on le donne juste des images contenant tout objet qu'on souhaite reconnaître et on lance un processus d'entraînement de ce réseau créé et au long de plusieurs itérations, le réseau tente de distinguer tout seul et sans intervention humaine les caractéristiques d'un objet quelconque et par suite élaborer une sorte d'un guide général pour détecter cet objet dans tout nouvelle image d'entrée. A la fin d'entraînement de réseau, l'ensemble de poids et l'architecture de réseau tout ensemble constituent un modèle qui peut être testé sur des nouveaux images et vidéos.

7.3 Pipeline de vision par ordinateur

un système de vision suit un enchaînement d'étapes permettant l'analyse et le traitement des images d'entrée. Ces étapes forment un flux de vision par ordinateur ou Computer vision pipeline. Au début le système commence par l'acquisition d'images, fait un certain traitement de données, effectue des opérations d'analyse et détermination des caractéristiques particuliers , pour passer enfin à l'étape de prédiction sur la base des attributs d'image extraits.

Le processus de figure est expliqué comme suit : Le programme reçoit une image comme entrée d'un dispositif d'imagerie comme une caméra. Ce l'entrée est généralement capturée sous la forme d'une image ou d'une séquence d'images formant une vidéo. L'image reçue va ensuite être soumise à des opérations de prétraitement ayant comme but de standardiser tous les images d'entrée. Quelques étapes de prétraitement sont : (itemize) le changement de dimensions d'un image, rotation, changement de forme ou transformation de l'image à partir de une couleur à l'autre, comme de la couleur à l'échelle de gris. Cette étape est cruciale car l'élaboration une solution précise ne peut être faite que lorsque toute nouvelle image acquise doit être dans la même forme de toutes les images précédentes.

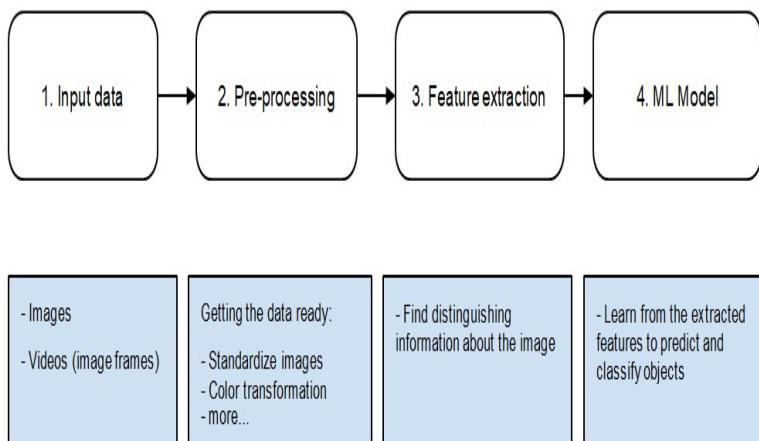


FIGURE 1.20 : Pipeline de vision par ordinateur

Dans cette étape, le système remarque certaines caractéristiques dans l'image permettant d'y distinguer des objets. Ces caractéristiques ou attributs ne sont que des informations sur la couleur ou forme d'un objet particulier. Par exemple, quelque attributs qui permettent l'identification d'une personne sont la taille de corps , la forme des oreilles et des yeux est la longueur de cou. La sortie de ce processus est un vecteur de caractéristiques qui est une liste de formes qui identifient l'objet. Les attributs extraits sont soumis à un modèle de classification qui analyse le vecteur de caractéristiques issue de phase précédente et fait une prédiction enfin de la classe de l'image. Prenons par exemple une image de

bicyclette. Les attributs extractés inclues des pédales des roues. Les pédales n'existent pas dans les voitures ni sont des organes de corps humain donc le système élimine tout objet loin d'être conforme avec les attributs existantes pour s'approcher enfin au objet qui est le plus proche de ces attribus.

7.4 tâches de cv

7.4.1 classification

La classification d'image est le processus résultant à attribuer une classe ou un label à un vecteur de pixels d'une image à partir d'un ensemble prédéfini de classes. Une image ne peut avoir qu'une seule classe. Afin de classifier un ensemble d'images en catégories, la relation entre les données et les classes dans lesquelles ils sont classés doivent être bien compris par le modèle de classification et c'est pour ça que ce dernier doit être entraîné. On peut citer quelques modèles pré-entraînés qui sont utilisé pour la classification d'image :

- vgg16 : Développé par le Visual Graphics Group de l'Université d'Oxford, VGG-16 l'un des modèles les plus adopté par l'industrie quand aux tâches de classification d'images. Son architecture est présenté comme suit : figure : dans cette architecture , 13 couches convolutives sont empilées les unes après les autres et 3 couches denses responsable de classification. Le nombre de filtres dans les couches de convolution suit un schéma croissant). Les caractéristiques informatives sont obtenues par max pooling couches appliquées à différentes étapes de l'architecture. Les couches denses comprennent chacune 4096, 4096 et 1000 noeuds. Les inconvénients de cette architecture sont qu'elle est lente à former et produit le modèle avec une très grande taille. Comme vous pouvez le voir, le modèle est de nature séquentielle et utilise de nombreux filtres. A chaque étape, des petits filtres 3×3 sont utilisés pour réduire le nombre de paramètres toutes les couches cachées utilisent la fonction d'activation ReLU. Même dans ce cas, le nombre de paramètres est de 138 milliards, ce qui en fait un modèle plus lent et beaucoup plus volumineux à entraîner que les autres.



FIGURE 1.21 : Architecture de vgg16

- Resnet50 : Une convolution avec une taille de noyau de $7*7$ et 64 noyaux différents le tout avec une foulée de taille 2 nous donnant 1 couche. Ensuite, nous voyons une mise en commun maximale avec également une taille de foulée de 2. Dans la prochaine convolution, il y a un noyau $1 * 1,64$ qui suit un noyau $3 * 3,64$ et enfin un noyau $1 * 1 256$. Ces trois couches sont répétées au total 3 fois, ce qui nous donne 9 couches dans cette étape. Ensuite, nous voyons un noyau de $1 * 1 128$, puis un noyau de $3 * 3 128$ et enfin un noyau de $1 * 1 512$. Cette étape a été répétée 4 fois, ce qui nous a donné 12 couches dans cette étape. Après cela, il y a un noyau de $1 * 1 256$ et deux autres noyaux avec $3 * 3 256$ et $1 * 1 1024$ et cela est répété 6 fois, ce qui nous donne un total de 18 couches. Et puis encore un noyau de $1 * 1 512$ avec deux autres de $3 * 3 512$ et $1 * 1 2048$ et cela a été répété 3 fois, ce qui nous a donné un total de 9 couches. Après cela, nous faisons un pool moyen et le terminons avec une couche entièrement connectée contenant 1000 noeuds et à la fin une fonction softmax, ce qui nous donne 1 couche. Nous ne comptons pas réellement les fonctions d'activation et les couches de pooling max/moyenne. donc en totalisant cela, cela nous donne un réseau convolutif profond de $1 + 9 + 12 + 18 + 9 + 1 = 50$ couches.

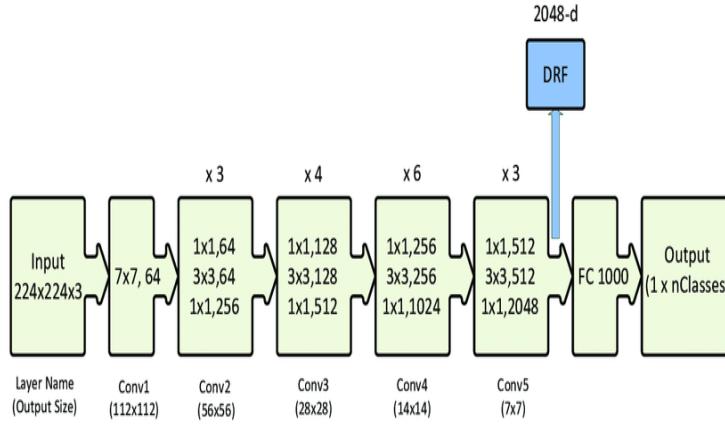


FIGURE 1.22 : Architecture de Resnet50

- Mobilenet : MobileNet est une architecture convolutionnelle fréquemment employé face au tâches de vision du monde réel. Au lieu d'utiliser des convolutions standards, MobileNet adopte des convolutions séparables en profondeur ou "depthwise separable convolutions" afin d'obtenir un modèle plus léger à l'exception du première couche qui utilise la convolution classique. Toutes les couches sont suivies d'une normalisation de batchs et d'une fonction ReLU. La couche finale est entièrement connectée sans aucune non-linéarité et alimente le softmax pour la classification.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32 \text{ dw}$	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64 \text{ dw}$	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128 \text{ dw}$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128 \text{ dw}$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256 \text{ dw}$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256 \text{ dw}$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512 \text{ dw}$	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512 \text{ dw}$	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024 \text{ dw}$	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

FIGURE 1.23 : Architecture de Mobilenet

Mobilenet inclut deux hyperparamètres globaux qui permettent encore une diminution de coût de calcul. multiplicateur de largeur l : Ce para-

mètre permet d'amincir le réseau à chaque couche. Pour une couche et un multiplicateur de largeur l donnés, le nombre de canaux d'entrée M devient lM et le nombre de canaux de sortie N devient lN . multiplicateur de resolution r : appliqué à l'image d'entrée et la représentation interne de chaque couche est ensuite réduite par le même multiplicateur. En pratique, nous définissons implicitement r en définissant la résolution d'entrée.

7.4.2 Détection d'objets

La tâche de classification d'image traite une image ne contenant qu'un seul objet, et l'objectif de cette tâche est de l'identifier. D'une autre part la vision par ordinateur cherche à atteindre des niveaux humains de compréhension d'où nous devons ajouter de la complexité aux réseaux convolutifs afin qu'ils devient capables de d'identifier plusieurs objets et déterminer leurs emplacements dans une seule image. divisez-le en régions plus petites et étiquetez chaque région avec une classe afin qu'un nombre variable d'objets dans une image donnée puisse être localisé et étiqueté (figure 1.7). Vous pouvez imaginer qu'une telle tâche est un prérequis de base pour des applications comme les systèmes autonomes. Avant de présenter quelques exemples de modèles il faut savoir qu'une architecture de détection d'objets comporte quatre composants :

- Proposition de région : un algorithme permettant la génération des régions d'intérêts (RoI) à traiter ultérieurement par les composants suivants de modèle. Ce sont des régions que le réseau pense qu'ils peuvent contenir un objet ; la sortie est un grand nombre de cadres (bounding box), dont chacun a un score. Les boîtes avec des scores élevés sont ensuite transmises à travers le réseau.
- Extraction d'attributs (features) et prédictions : les propriétés visuelles sont extraites pour chaque cadre. Ils sont évalués et il est déterminé si et quels objets sont présents dans les propositions sur la base de caractéristiques visuelles (par exemple, un composant de classification d'objets).
- Suppression non maximale (NMS) : Souvent à cette étape le modèle peut trouver des cadres (bounding box) multiples sur le même objet. NMS est une solution pour éviter la détection répétée de la même instance en combinant des boîtes qui se chevauchent dans une seule boîte englobante pour chaque objet.

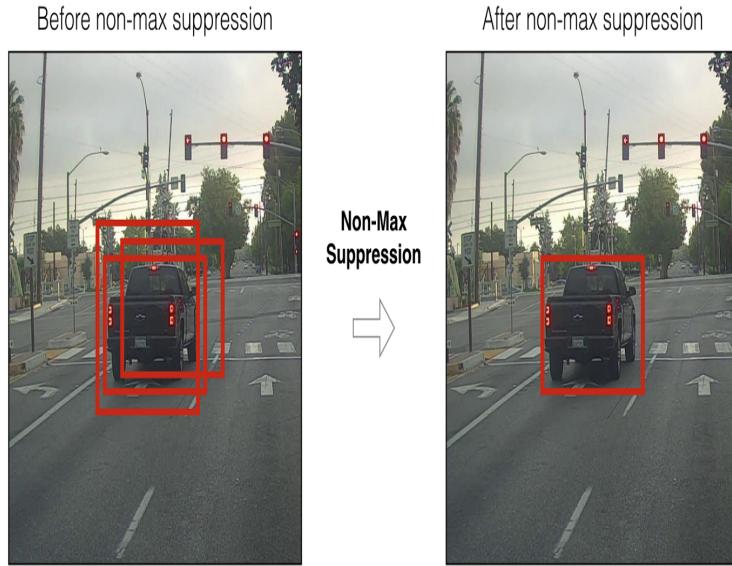


FIGURE 1.24 : Suppression non maximale (NMS)

- Métriques d'évaluation : Comme les modèles de classification , les modèles de détection possèdent des métriques pour évaluer leurs performances de détection. Dans cette section, nous expliquerons les métriques les plus populaires :
 - FRAMES PAR SECONDE (FPS) POUR MESURER LA VITESSE DE DÉTECTION
 - PRÉCISION MOYENNE (MAP) POUR MESURER LA PRÉCISION DU RÉSEAU
 - INTERSECTION SUR L'UNION (IOU) pour mesurer le chevauchement entre deux cadres de délimitation : entre le box vrai et le box prédit

Parmi les modèles pré-entraînés les plus populaires on cite :

- Rcn (Region-based Convolutional Neural Networks) : L'idée principale consiste de deux étapes principales. Au premier lieu, le modèle fait une recherche sélective, il identifie quelques régions d'intérêts candidats. Par suite le modèle dans une deuxième étape extrait les caractéristiques et attributs de chaque région indépendamment pour les passer à la classification. Le modèle R-CNN se compose de quatre composants :
 - Extraction des régions d'intérêt : également connu sous le nom d'extraction de propositions de région. Ces régions ont une forte probabilité de contenir un objet. Un algorithme dit sélectif recherche scanne

l'image d'entrée pour trouver les régions qui contiennent des gouttes, et propose en tant que ROI à traiter par les prochains modules du pipeline. Les ROI proposés sont ensuite déformés pour avoir une taille fixe. Ils varient généralement en taille,

- Partie d'extraction de caractéristiques : un réseau convolutif pré-entraîné est appliqué sur chaque région proposé afin d'y extraire des caractéristiques particuliers.
- Module de classification—Nous formons un classificateur comme une machine à vecteurs de support (SVM), sur les caractéristiques extraites de l'étape précédente un algorithme d'apprentissage automatique traditionnel, pour classer les detections de candidats en fonction.
- Module de localisation : Dans ce module, nous voulons prédire l'emplacement et la taille de la délimitation

En général la famille R-CNN sont des détecteurs à plusieurs étages : le réseau d'abord prédit le score d'existence d'un objet de la région d'intérêt, puis passe cette boîte à travers un classificateur pour prédire la probabilité de classe. Ce processus est différent aux familles de modèles comme yolo et ssd qui nécessitent une seule phase de prédiction donnant comme résultat les scores pour la présence d'un d'objets et les probabilités de classe.

- SSD : Single-shot detector L'approche SSD utilise un réseau convolutif de propagation vers l'avant qui produit un ensemble de cadres (bounding boxes) de taille fixe et de scores pour la présence d'une classe d'objets instances dans ces cases, suivie d'une étape NMS (Non Maximum Suppression) pour produire les detections finales. L'architecture du modèle SSD consiste de 3 composants principaux :

- extraction des cartes de caractéristiques : C'est la tâche d'un réseau pré-entraîné utilisé pour classification d'images de haute qualité, qui est tronquée avant toute classification couches. VGG16 a été utilisé comme réseau de base en raison de ses bonnes performances dans les tâches de classification d'images de haute qualité mais VGG19 et ResNet peuvent être utilisés aussi et avec des bons résultats.
 - Couches d'entités à plusieurs échelles : une série de filtres de convolution est ajoutée après la base réseau. Ces couches diminuent progressivement de taille pour permettre des prédictions de detections à plusieurs échelles.
 - Suppression non maximale—NMS est utilisé pour éviter le chevauchement entre les rectangles autour d'un objet afin de garder qu'un rectangle pour chaque objet détecté.
- yolo l'un des algorithmes de détection d'objets les plus populaires pour la détection d'objets en raison de sa vitesse de détection, souvent démontrée en vidéo ou en caméra en temps réel entrée d'alimentation. Ce modèle

considère la détection d'objets comme un problème de régression produisant comme résultat les probabilités de classe et les coordonnées de boîtes encadrant tout objet détecté. YOLO utilise un réseau de neurones convolutifs (CNN) pour détecter des objets en temps réel. Comme son nom l'indique, yolo a besoin seulement propagation vers l'avant à travers le réseau afin de détecter des objets. Cela signifie que tous les prédictions sont effectuées en une seule exécution d'algorithme. Le réseau YOLO adopte le principe de diviser l'image d'entrée en une grille de cellules $S \times S$. Si le centre de la boîte de vérité au sol tombe dans une cellule, cette cellule est chargée de détecter l'existence de cet objet. Chaque cellule de la grille prédit le nombre B de cadres de délimitation et leur score de contenir des objets avec leurs prédictions de classe, comme suit :

- Coordonnées des cadres (bounding boxes) : Similaire aux détecteurs précédents, YOLO prédit quatre coordonnées pour chaque cadre de délimitation (bx , by , bw , bh), où x et y sont définis sur être des décalages d'un emplacement de cellule
- Score d'existence d'objet : indique la probabilité que la cellule contienne un objet. Le score d'objectivité est passé par une fonction sigmoïde pour être traité comme une probabilité avec une plage de valeurs comprise entre 0 et 1.
- Prédiction de classe : si le cadre de délimitation contient un objet, le réseau prédit la probabilité de K nombre de classes, où K est le nombre total de classes dans ton problème.

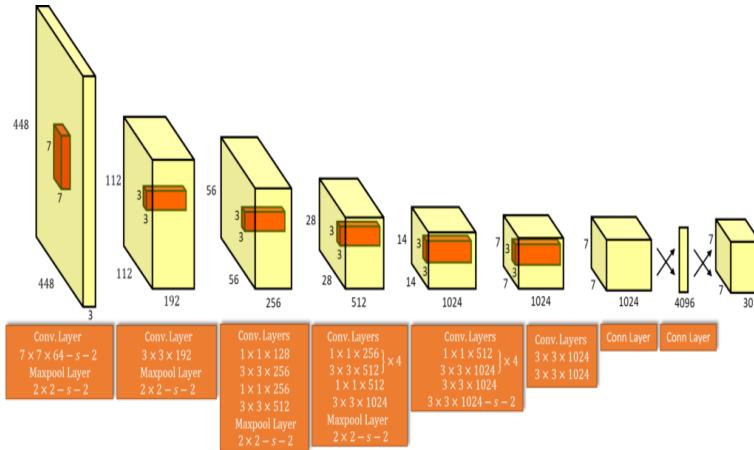


FIGURE 1.25 : Architecture de yolo

7.5 OPENCV

7.6 introduction

Lorsque nous connectons un appareil photo numérique à un ordinateur, nous pouvons obtenir les images capturées par l'appareil photo sous la forme d'une matrice (boîte de nombres). Maintenant, la question est que faire avec ces images obtenues. Nous pouvons stocker ces images, convertir cette séquence d'images en une vidéo, éditer ces images, ou même essayer de comprendre ce qu'est cette image (identifier l'objet ou reconnaître les visages, etc.). Pour faire tout cela, nous avons besoin d'une séquence de calculs mathématiques qui peuvent impliquer des calculs mathématiques. Il y a un besoin d'un ensemble d'outils pour effectuer ces opérations telles que l'édition et traitement d'images, la reconnaissance d'objets. La bibliothèque est écrite en C et C++ et fonctionne sous Linux, Windows et Mac OS X. OpenCV a été conçu pour l'efficacité de calcul et avec un fort accent sur les applications en temps réel. OpenCV est écrit en langage C optimisé et peut tirer parti des processeurs multicoeurs. L'un des objectifs d'OpenCV est de fournir une infrastructure de vision par ordinateur simple à utiliser qui aide les gens à créer rapidement des applications de vision assez sophistiquées. Open CV vous fournit un ensemble de bibliothèques très bien documentées et faciles à utiliser pour effectuer toutes ces opérations sur les images. Il s'agit d'un ensemble très élaboré de bibliothèques qui incluent également des modèles d'apprentissage automatique. Cela fait gagner beaucoup de temps aux développeurs. C'est l'une des meilleures bibliothèques de traitement d'images disponibles dans l'espace open source Opencv consiste de 4 composants principales :

- cv : le composant contient le traitement d'image de base et l'algorithme de vision par ordinateur de niveau supérieur
- ML est la bibliothèque d'apprentissage automatique, qui comprend de nombreux classificateurs statistiques et outils de clustering
- highGUI : contient des routines d'E/S et des fonctions pour stocker et charger des vidéos et des images.
- CXCore : contient les structures de données et algorithmes de base

7.7 Quelques image processing techniques avec opencv

- Changer les espaces colorimétriques : Il existe plus de 150 méthodes de conversion d'espaces colorimétriques disponibles dans OpenCV. Mais nous n'en examinerons que deux, qui sont les plus utilisées : BGR vers Gris et vice versa , BGRvers HSV et vice versa.

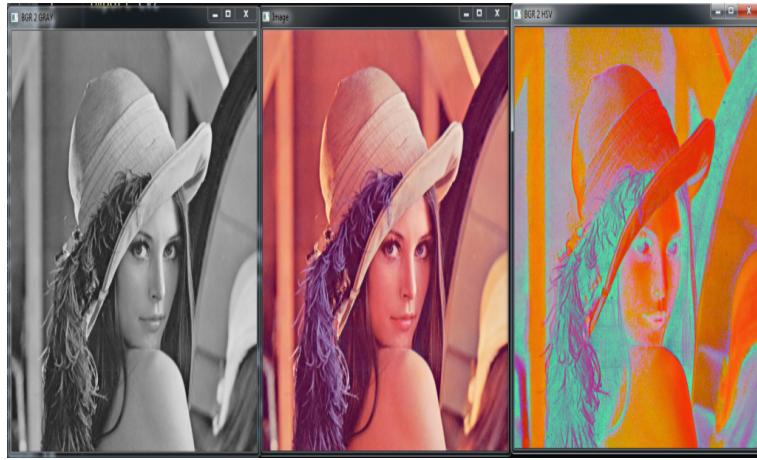


FIGURE 1.26 : types de couleurs

- seuillage d'image : Le seuillage est une technique dans OpenCV, qui est l'attribution de valeurs de pixels par rapport à la valeur de seuil fournie. Dans le seuillage, chaque valeur de pixel est comparée à la valeur de seuil. Si la valeur du pixel est inférieure au seuil, elle est fixée à 0, sinon, elle est fixée à une valeur maximale (généralement 255). Il existe plusieurs types de seuillage tel que seuillage simple, seuillage adaptatif et seuillage otshu.

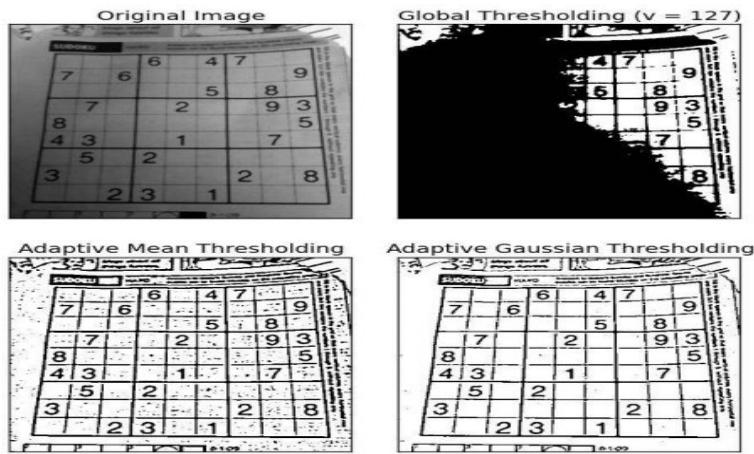


FIGURE 1.27 : types de seuillage

- Transformations morphologiques : Ce sont les techniques utilisées pour manipuler les tailles d'image, ces techniques ne sont utilisées qu'après avoir converti l'image en niveaux de gris. Ces techniques sont utilisées pour

éliminer le bruit, corriger les imperfections des données et créer des images claires. L'érosion et la dilatation sont principalement utilisées dans les techniques morphologiques.

- Erosion : Cette technique supprime les pixels de frontière de l'entrée en passant simplement le filtre sur l'image. Selon la taille du noyau, il supprime les pixels de frontière de l'image d'entrée. Il fonctionne de manière similaire à l'érosion du sol, ils ont donc appelé cette technique l'érosion.



FIGURE 1.28 : erosion

- Dilatation La technique ci-dessus supprime les pixels limites mais en dilatation. Il ajoute les pixels supplémentaires à l'entrée. Il est utilisé lorsque les pixels sont manquants dans l'image. En pratique générale, on applique l'érosion pour rétrécir l'image afin de supprimer les bruits, puis en appliquant la dilatation, il n'y aura pas de perte de pixels.



FIGURE 1.29 : dilation

Le problème avec cet approche est que dans plusieurs cas la plaque n'a pas des lignes claires et par suite, on échoue à détecter la plaque

- contours : Les contours peuvent être expliqués simplement comme une courbe joignant tous les points continus (le long de la frontière), ayant la même couleur ou intensité. Les contours sont un outil utile pour l'analyse de forme et la détection et la reconnaissance d'objets.

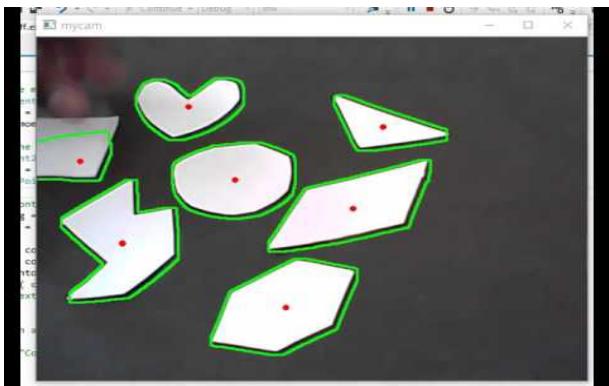


FIGURE 1.30 : contours

Chapitre 2

Implémentation

8 étude

Introduction Dans ce chapitre , nous avons comme but de montrer le contexte général et expliquer la structure générale du projet, ses détails et ses objectifs. Puis nous allons présenter les choix optimaux pour une bonne exécution du projet.

Contexte général Le sujet de projet s'intitule "gestion automatisé de traffic routier" dans le cadre de l'obtention de mastère proffesionel de science et données de l'institut supérieur de l'informatique de mahdia.

Problème général La vision par ordinateur est utilisé dans nombreux domaine notamment la surveillance et les systèmes des voitures automatisés. En tunisie, il existe plus de 2 millions de voituresce qui génère un traffic routier immense ce qui cause une besoin d'une solution automatisé de gestion de ce traffic.

Solution proposée Notre solution consiste d'un système de gestion automatisée de traffic routier qui va fonctionner en temps réel et identifier le numéro de matricule de chaque nouveau voiture afin de le comparer à un fichier csv pour déterminer si le numéro détecté correspond à une voiture volée ou dont les papiers ne sont pas réglés.

Outils matériels Ce projet sera réalisé sur un ordinateur portable ayant comme caractéristiques :

- Ram : 8 Go
- Processeur : i5 8ème génération
- carte graphique : gtx 1050 ti

Outils logiciels Après le choix matériel, il est nécessaire de spécifier l'environnement logiciel du travil et les technologies à utiliser.

- Anaconda : Anaconda est une distribution Python qui facilite l'installation de Python et d'un certain nombre de ses bibliothèques tierces les plus souvent utilisées de manière flexible sur une machine Windows ou Linux. Anaconda inclut
 - Le langage Python de base
 - Plus de 100 "paquets" Python (bibliothèques)
 - Spyder (IDE/éditeur - comme PyCharm) et Jupyter
 - conda, le propre gestionnaire de packages d'Anaconda, utilisé pour mettre à jour Anaconda et les packages

- Google Colaboratory : Colaboratory, ou « Colab », est un produit de Google Research permettant d'écrire et d'exécuter du code Python via le navigateur. c'est un choix incontournable pour les tâches de deep learning tel que l'entraînement des modèle au lieu de mettre notre machine locale en risque de surchauffage.
- CMAKE : CMake est un système de construction logicielle multiplate-forme. Il permet de vérifier les prérequis nécessaires à la construction, de déterminer les dépendances entre les différents composants d'un projet. Cet outil nous aidera à installer la bibliothèque Opencv et la configurer pour supporter la carte graphique.
- Visual studio : Cet outil va s'intégrer dans le processus de configuration d'Opencv avec Cmake
- Notepad++ : C'est un éditeur de texte libre générique, fonctionnant sous Windows, codé en C++, qui intègre la coloration syntaxique de code source pour les langages et fichiers.

Nous avons défini les différents choix matériels et logiciels et par suite nous allons aborder l'étude conceptuelle de projet :

Fonctionnement de système Le système repose sur une caméra qui visualise le trafic des voitures. Dans chaque image ou 'frame' du vidéo, le système tente de détecter tout plaque de matricule. Une fois une plaque est détecté, le système extrait une sous-image contenant la plaque à partir de l'image de vidéo. L'image de plaque extraite subit un certain traitement pour reconnaître le numéro de matricule. Ensuite le système effectue une recherche dans un fichier csv contenant tous les numéros de matricule avec chaque ligne indique si le numéro correspond à une voiture volée ou dont les papiers ne sont pas réglés. Si le numéro de matricule est susceptible il affiche le numéro en rouge dans l'image de vidéo en rouge juste au dessus de la plaque d'immatriculation sinon le numéro sera affiché en couleur vert.

Choix La réalisation de chaque étape de ce projet nécessite une choix d'une méthode optimale pour assurer un fonctionnement du système en toute fluidité et précision.

Choix détection plaque Détection de plaque d'immatriculation : La recherche de plaques de matricule se déroule à chaque image de vidéo ce qui nécessite d'éviter tout approche complexe ou nécessitant un calcul intense et par suite assurer une rapidité du fonctionnement système qui est mesuré par le nombre d'image par seconde ou Frames per second (fps). C'est pour ça que nous discutons les approches les plus probables à utiliser.

- haar cascade : Un Haar Cascade est fondamentalement un classificateur utilisé pour détecter un objet en se basant sur des filtres déterminées

manuellement au contraire au cas des CNN où les filtres sont calculés au long d'entraînement. Cette méthode a une vitesse d'exécution plus élevée, car les classificateurs basés sur Haar impliquent généralement moins de calculs. Cependant, étant donné que les caractéristiques Haar doivent être déterminées manuellement, il existe une certaine limite aux types de choses qu'il peut détecter. Si vous donnez au classificateur 'haar cascade' des caractéristiques de bord et de ligne, il ne pourra détecter que les objets avec des bords et des lignes clairs. Cette méthode est trop faible en termes de précision mais bien mieux en termes de vitesse/coût de calcul.

- Traitement d'image de vidéo : On effectue un ensemble d'opérations sur chaque frame de vidéo pour détecter la plaque de matricule :

- Redimensionnez l'image à la taille requise, puis mettez-la en niveaux de gris : La mise à l'échelle des gris est courante dans toutes les étapes de traitement d'image. Cela accélère les autres processus suivants car nous n'avons plus à traiter les détails de la couleur lors du traitement d'une image



FIGURE 2.1 : une image de voiture avant tout traitement

- L'étape suivante est intéressante où nous effectuons la détection des contours. Une des façons possibles consiste à utiliser la méthode canny edge de la bibliothèque OpenCV.
- Maintenant, nous pouvons commencer à chercher des contours sur notre image. Une fois les contours détectés, nous les trions du plus gros au plus petit et ne considérons que les 10 premiers résultats en ignorant les autres. Dans notre image, le compteur pourrait être tout ce qui a une surface fermée, mais de tous les résultats obtenus, le numéro de plaque d'immatriculation sera également présent car il s'agit également d'une surface fermée. Pour filtrer l'image de la plaque d'immatriculation parmi les résultats obtenus, nous allons parcourir tous les résultats et vérifier lequel a un contour en forme de rectangle avec

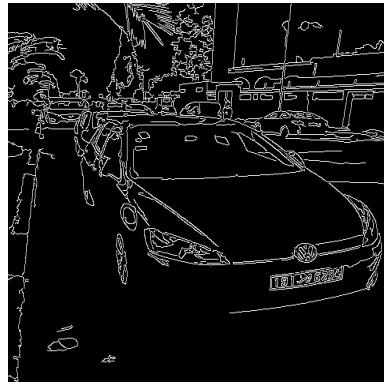


FIGURE 2.2 : Résultat de la fonction canny

quatre côtés et une figure fermée. Puisqu'une plaque d'immatriculation serait certainement une figure rectangulaire à quatre côtés. on remarque plusieurs cas d'échéance en utilisant cet approche pour multiple raisons :

- * Les bordures de plaques ne sont pas claires



FIGURE 2.3 : cas d'échéance

- * Dans une image, on n'a pas seulement une voiture mais aussi son environnement. Par suite on peut avoir des formes rectangulaires dont les mesures sont identiques ou près de celles du plaque d'immatriculation. Dans ce cas, même un filtrage des rectangles par leurs dimensions échoue certainement à distinguer le rectangle de plaques parmi les autres formes rectangulaire dans l'image.
- modèle de détection d'objets : Nous utilisons un modèle de détection d'objet pré-entraînée ce qui présente un apprentissage par transfert. L'appren-



FIGURE 2.4 : autre cas d'échéance

tissage par transfert est utile lorsque nous possédons une quantité de données insuffisantes pour un nouveau domaine que nous voulons gérer par un réseau de neurones et qu'il existe un grand ensemble de données préexistant qui peut être utilisé à notre problème. Ainsi, nous avons moins que 1 000 images de voitures avec plaques de matricule tunisienne, mais en exploitant un CNN existant, formé avec plus d'un million d'images, Nous pouvons obtenir de nombreuses définitions de fonctionnalités de moyen et de bas niveau.

Choix final Il apparaît qu'un modèle de détection d'objets pré-entraîné est le choix le plus robuste. Mais on doit spécifier exactement quel modèle à utiliser. Pour une opération de détection d'objet exécuté chaque en temps réel, il est obligatoire de choisir un modèle ayant un grand nombre de fps et un taux minimum de précision. Nous adopterons la dataset 'coco' (Common Objects in Context) comme critère de choix. CoCO est l'un des ensembles de données d'images les plus populaires avec plus de 200000 image annotés avec 80 classes. les images de COCO des objets de tous les jours capturés à partir de scènes de tous les jours. Cela ajoute du contexte aux objets capturés dans les scènes.



FIGURE 2.5 : Détection avec modèle de détection d'objet

Real-Time Object Detection on COCO

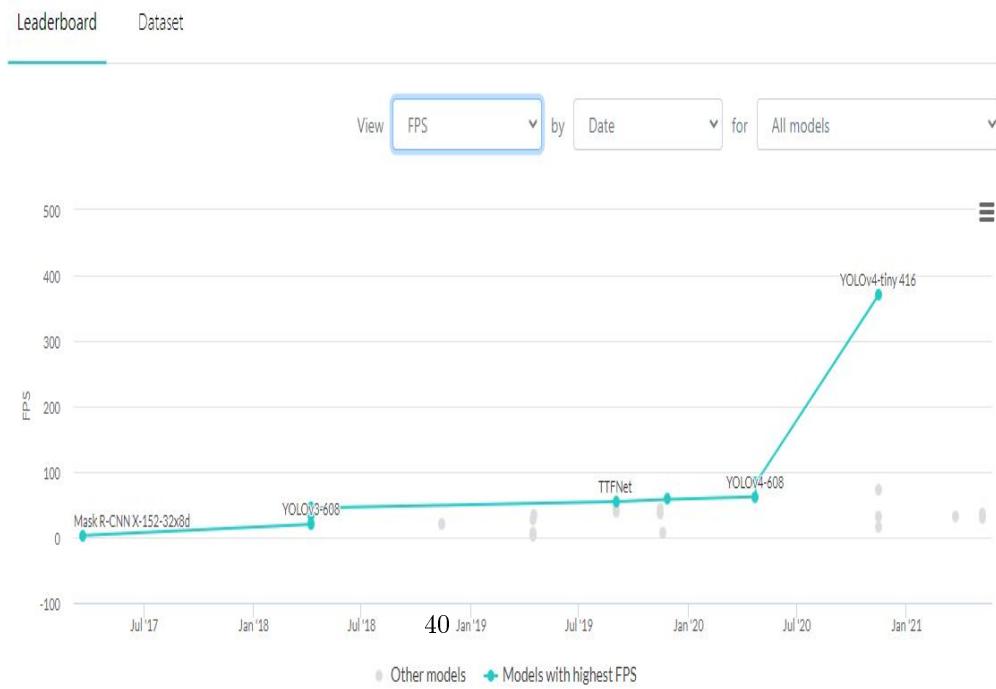


FIGURE 2.6 : Tri de modèles coco par fps

D'après la figure ci-dessus, en utilisant Yolov4-tiny on obtient 371 fps avec 288 fps loin du deuxième modèle. Avec une carte graphique rtx2080 ti obtient 443 fps. Donc ce modèle est le meilleur choix pour la détection de plaque d'immatriculation. en effet YOLOv4-tiny est la version compressée de YOLOv4. Il est proposé sur la base de YOLOv4 pour simplifier la structure du réseau et réduire les paramètres

Choix reconnaissance de matricule

- Contours + modèle de détection d'objets : Nous utilisons un modèle de détection d'objets pour détecter les zones d'intérêts dans l'image de plaque. Ces zones sont à gauche et à droite du mot "tounes".



FIGURE 2.7 : Modèle de détection de 2 zones à côté de mot tounes

On utilise les coordonnées résultantes de modèle de détection pour extraire deux images de l'image de plaque avec chacun représentant une zone d'intérêt.



FIGURE 2.8 : zone d'intérêt à gauche de mot tounes



FIGURE 2.9 : zone d'intérêt à droite de mot tounes

On fait la binarisation de chaque image de zone et on extrait les contours et on les tri de gauche à droite. On extrait les coordonnées de chaque contour et on extrait une sous-image contenant chaque chiffre. Enfin on passe chaque image de chiffre à un modèle de classification entraîné par exemple sur la dataset MNIST ou bien on utilise un outil OCR. Dans le cas de figure ci-dessus, on a réussi à extraire les chiffres dans la partie droite de plaque.



FIGURE 2.10 : approche efficace avec la zone d'intérêt à droite de mot tounes

D'une autre part, on n'a pas arrivé extraire les chiffres dans la zone d'intérêt gauche à cause de l'attachement entre deux chiffres adjacents.



FIGURE 2.11 : approche inéfficace avec la zone d'intérêt à gauche de mot tounes

Si on utilise une fonction OpenCV de seuillage adaptatif comme solution, on va rencontrer une autre difficulté. Les paramètres de fonction de seuillage optimales pour quelques cas de plaques tournent inutiles pour d'autres images de plaques. On peut même avoir des chiffres qui sont effacés entièrement ou partiellement. Alors il est impossible d'élaborer une combinaison de paramètres unique pour tous les situations. Une dernière solution qu'on peut tester pour cet approche est d'utiliser un outil OCR (reconnaissance optique de caractères) pour lire directement le numéro situé dans chaque zone d'intérêt sans l'obligation de passer par une étape d'extraction de contours. On utilise Tesseract-OCR, l'un des outils OCR les plus populaires et précis. Pour tirer le meilleur de cet outil, nous devons tout d'abord rendre chaque zone d'intérêt au niveaux de gris puis la binariser par une fonction de seuillage.



FIGURE 2.12 : Résultat de tesseract la zone d'intérêt à gauche de mot tounes

Tesseract a réussi à lire correctement le numéro situé dans cette image.



FIGURE 2.13 : Résultat de tesseract la zone d'intérêt à droite de mot tounes

Au contraire , Tesseract a échoué dans la reconnaissance du numéro situé dans cette la zone d'intérêt droite. Avec telle performance, on conclut que les outils tesseract ne sont pas utiles pour notre projet. en général Tesseract n'est adéquat qu'avec des images contenant un texte clair et nette avec aucune inclinaison et aucune bruit ce qui est impossible à garantir dans notre projet.

- Détection de chiffres + classification : Cet approche consiste de détecter seulement les chiffres numériques par un modèle de détection pré-entraîné. Puis on extrait le plus grand contour de chaque détection pour extraire seulement le chiffre. Ce modèle aura une seule classe "chiffre".



FIGURE 2.14 : extraction des nombres par un modèle de détection

Enfin chaque image de chiffre est passé comme entrée soit à un outil ocr ou un modèle de classification. Encore une fois, Tesseract montre une faible performance à cause de problème de seuillage.



FIGURE 2.15 : L'image de 3 est lit ":" par tesseract

On a le choix de passer chaque chiffre comme entrée à un modèle de classification entraîné sur un dataset massif de chiffres comme mnist. Voici un réseau convolutionnel en keras entraîné sur MNIST et qui atteint une précision de plus de 0.99.

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

FIGURE 2.16 : Préparation des données mnist

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
batch_size = 128
epochs = 15
#epochs = 50

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

FIGURE 2.17 : Construction de modèle

```
[ ] score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])

Test loss: 0.024083824828267097
Test accuracy: 0.9921000003814697
```

FIGURE 2.18 : Evaluation de modèle

- Détection de chiffres seulement Ici on utilise un modèle de détection pré-entraîné et on le réentraîne sur 11 classes qui seront les chiffres de 0 à 9 et le mot "toumes" Les noms de classes résultantes de chaque détection d'un chiffre seront assemblés dans un mot contenant le numéro de matricule.

Choix final L'Ocr performe terriblement dans la reconnaissance de numéro de matricule alors c'est un choix à éviter. Le choix de contours n'est pas approprié à notre projet car il est délicat aux situations de luminances différentes dans une seule image de plaque. Donc une solution qui dépend complètement au deep learning est un choix robuste. L'approche 3 représente une solution plus efficace que l'approche 2 car il regroupe la détection et la reconnaissance dans un seul modèle de détection au lieu de diviser la tâche en deux modèles comme le cas dans l'approche 3. Mais quel modèle exactement ? La modèle yolov4-tiny choisi pour la détection de plaque d'immatriculation peut être choisi également

Entrainement de modèles

Détection de plaque Pour cette tâche nous avons collecter des centaines d'image de voitures ayant un matricule tunisien. Cet dataset sera annotée par un logiciel d'annotation. Dans chaque image , on dessine un rectangle autour de plaque d'immatriculation et le logiciel va générer automatiquement un fichier texte qui contient des ligne dont chacun contient les coordonnées du rectangle dessiné et le nom de classe correspondante. Ici on aura une seule classe Lp représentant la plaque d'immatriculation.

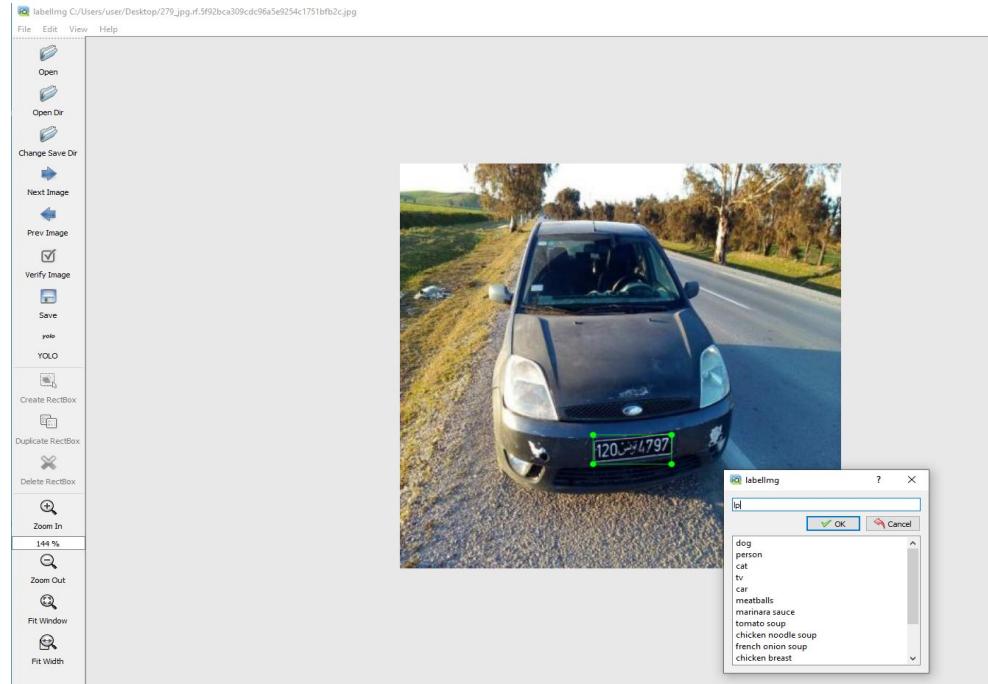


FIGURE 2.19 : Exemple d'annotation d'une image

Chaque image aura un fichier .txt sera généré automatiquement par LabelImg portant le même nom et contenant les coordonnées du rectangle dessiné sur l'image ainsi que les classes d'objets.

```
IMG_20210819_112901 - Bloc-notes
Fichier Edition Format Affichage Aide
15 0.037646 0.099095 0.061404 0.011239
15 0.366959 0.094709 0.025585 0.005208
15 0.859832 0.188596 0.078582 0.014254
```

FIGURE 2.20 : Fichier txt généré par LabelImg

Après finir l'annotation de tous les images. on répartit les images en 3 dossiers train contenant 75% des images, valid et test afin de les utiliser plus tard dans l'entraînement de notre modèle. On crée un fichier d'extension LABELS dans chaque dossier contenant les classes (labels). L'entraînement des modèles localement cause un grand risque de surchauffage de la machine locale utilisée. Une solution pour éviter tout risque est d'utiliser google colab. La première chose à faire après ouvrir une session colab est s'assurer que colab utilise une carte graphique au lieu d'un processeur pour exécuter le code python et par suite réduire le temps d'entraînement.

Notebook settings

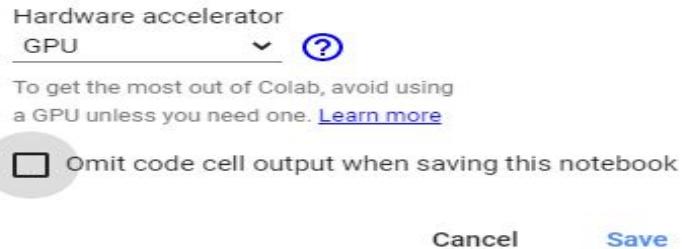


FIGURE 2.21 : Type d'exécution de colab

Ensuite on commence par télécharger le framework Darknet. Darknet est un framework de réseau de neurones open source écrit en C et CUDA. Il est rapide, facile à installer et prend en charge les calculs CPU et GPU.

```
[5] 2s
!git clone https://github.com/AlexeyAB/darknet

Cloning into 'darknet'...
remote: Enumerating objects: 15301, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 15301 (delta 0), reused 3 (delta 0), pack-reused 15298
Receiving objects: 100% (15301/15301), 13.67 MiB | 23.97 MiB/s, done.
Resolving deltas: 100% (10397/10397), done.
```

FIGURE 2.22 : Installation de Darknet

Une fois Darknet téléchargé, il faut l'installer et le configurer en se basant sur un fichier Makefile

```
[6] 1m
%cd /content/darknet/
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/g' Makefile
!sed -i "s/ARCH= -gencode arch=compute_60,code=sm_60/ARCH= ${ARCH_VALUE}/g" Makefile
!make
```

FIGURE 2.23 : Installation de Darknet

Par suite nous téléchargeons les poids du modèle pré-entraîné yolov4-tiny.

```

[7] #download the newly released yolov4-tiny weights
ls
%cd /content/darknet
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.weights
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29

/content/darknet
--2021-08-22 01:50:09-- https://github.com/AlexeyAB/darknet/releases/download/darknet\_yolo\_v4\_pre/yolov4-tiny.weights
Resolving github.com (github.com)... 192.30.255.112

```

FIGURE 2.24 : Téléchargement des poids de Yolov4-tiny

Ensuite nous faisons un "upload" de dossiers train, valid et test contenant les images et leurs annotations à notre session colab sous le dossier Darknet.

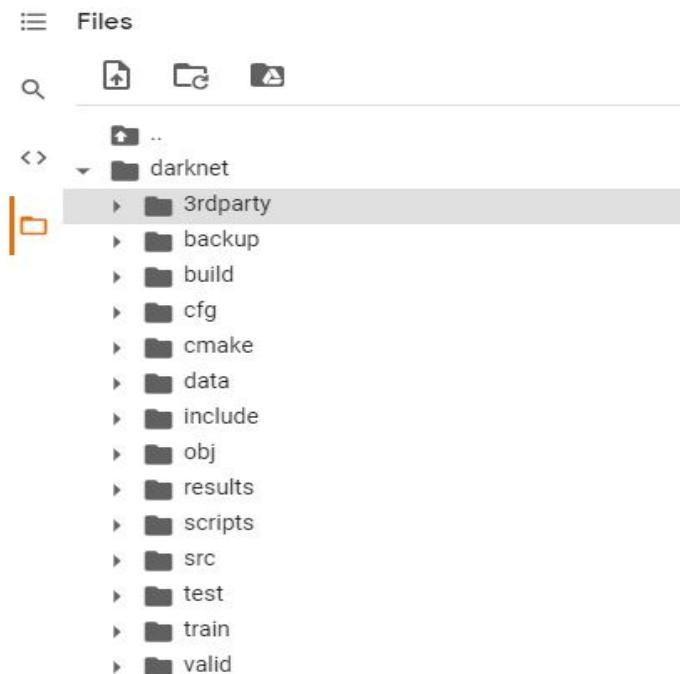


FIGURE 2.25 : Upload des dossiers train, valid et test au session colab

Puis nous configurons les répertoires des fichiers d'entraînement. Le bloc du code ci-dessous crée les fichiers train.txt, valid.txt et test.txt où le fichier train.txt a des chemins vers 75% des images, valid.txt a des chemins vers 10% et test.txt a des chemins vers 15% des images.

```

[9] %cd /content/darknet/
%cp train/_darknet.labels data/obj.names
mkdir data/obj

%cp train/*.jpg data/obj/
%cp valid/*.jpg data/obj/
%cp test/*.jpg data/obj/

%cp train/*.txt data/obj/
%cp valid/*.txt data/obj/
%cp test/*.txt data/obj/

with open('data/obj.data', 'w') as out:
    out.write('classes = 1\n')
    out.write('train = data/train.txt\n')
    out.write('valid = data/valid.txt\n')
    out.write('test = data/test.txt\n')
    out.write('names = data/obj.names\n')
    out.write('backup = backup/')

import os

with open('data/train.txt', 'w') as out:
    for img in [f for f in os.listdir('train') if f.endswith('.jpg')]:
        out.write('data/obj/' + img + '\n')

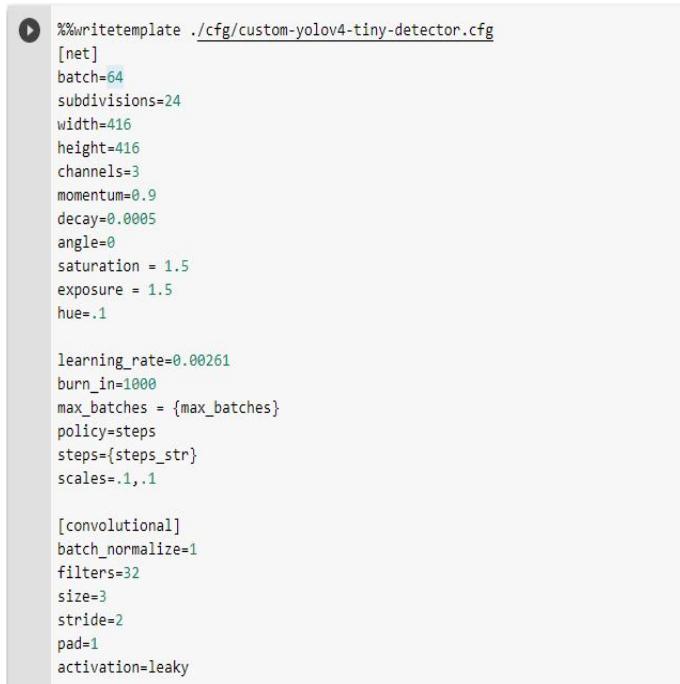
with open('data/valid.txt', 'w') as out:
    for img in [f for f in os.listdir('valid') if f.endswith('.jpg')]:
        out.write('data/obj/' + img + '\n')

with open('data/test.txt', 'w') as out:
    for img in [f for f in os.listdir('test') if f.endswith('.jpg')]:
        out.write('data/obj/' + img + '\n')

```

FIGURE 2.26 : Création des fichiers train.txt, valid.txt et test.txt

Par suite, nous définissons le fichier .cfg contenat l'architecture de notre modèle. En fait on ne vas pas rédiger de zéro mais on va apporter des modifications au fichier custom-yolov4-tiny-detector.cfg téléchargé depuis ce lien "<https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov4-tiny-custom.cfg>". nous allons apporter les modifications suivantes dans le fichier de configuration :



```
%%writetemplate ./cfg/custom-yolov4-tiny-detector.cfg
[net]
batch=64
subdivisions=24
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.00261
burn_in=1000
max_batches = {max_batches}
policy=steps
steps={steps_str}
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=2
pad=1
activation=leaky
```

FIGURE 2.27 : Extrait de fichier yolov4-tiny-custom.cfg

Enfin on lance le processus d’entraînement de notre modèle

```
[ ] !./darknet detector train data/obj.data cfg/custom-yolov4-tiny-detector.cfg yolov4-tiny.conv.29 -dont_show -map
```

FIGURE 2.28 : Lancement d’entraînement du modèle

Sous la répertoire Darkent, se trouve le fichier chart.png qui décrit la performance du modèle.

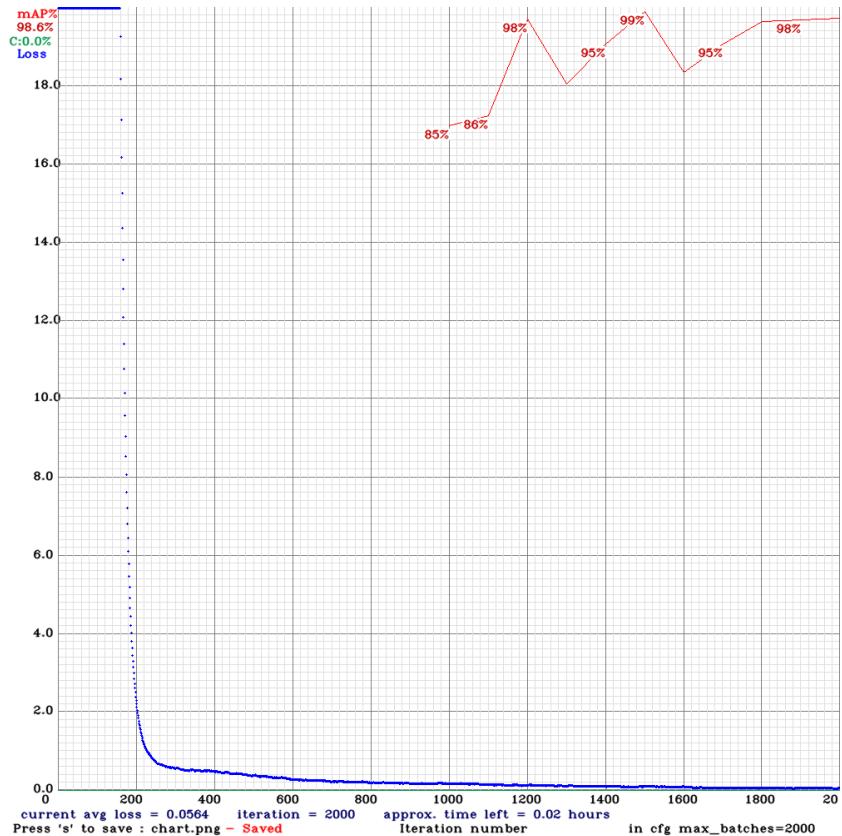


FIGURE 2.29 : Evaluation du modèle

Le modèle atteint une précision moyenne de 98.57 % ce qui présente une excellente résultat. Finalement nous avons 3 fichiers à télécharger

- custom-yolov4-tiny-detector.WEIGHTS : fichier contenant les nouveaux poids situé sous le dossier Darknet
- custom-yolov4-tiny-detector.cfg : fichier contenant l'architecture de modèle yolov4-tiny situé sous le dossier Darknet/cfg
- obj.NAMES : contenant les noms de classes

Reconnaissance de plaque Nous collectons pour cette tâche des centaines d'images de plaques de matricules tunisiens avec différentes angles d'inclinaison , degrés de luminances et taille de chiffres. Nous allons annoter ces images par labelImg et cette fois nous aurons 11 classes qui seront les chiffres de 0 à 9 et tn représentant le mot "tounes".

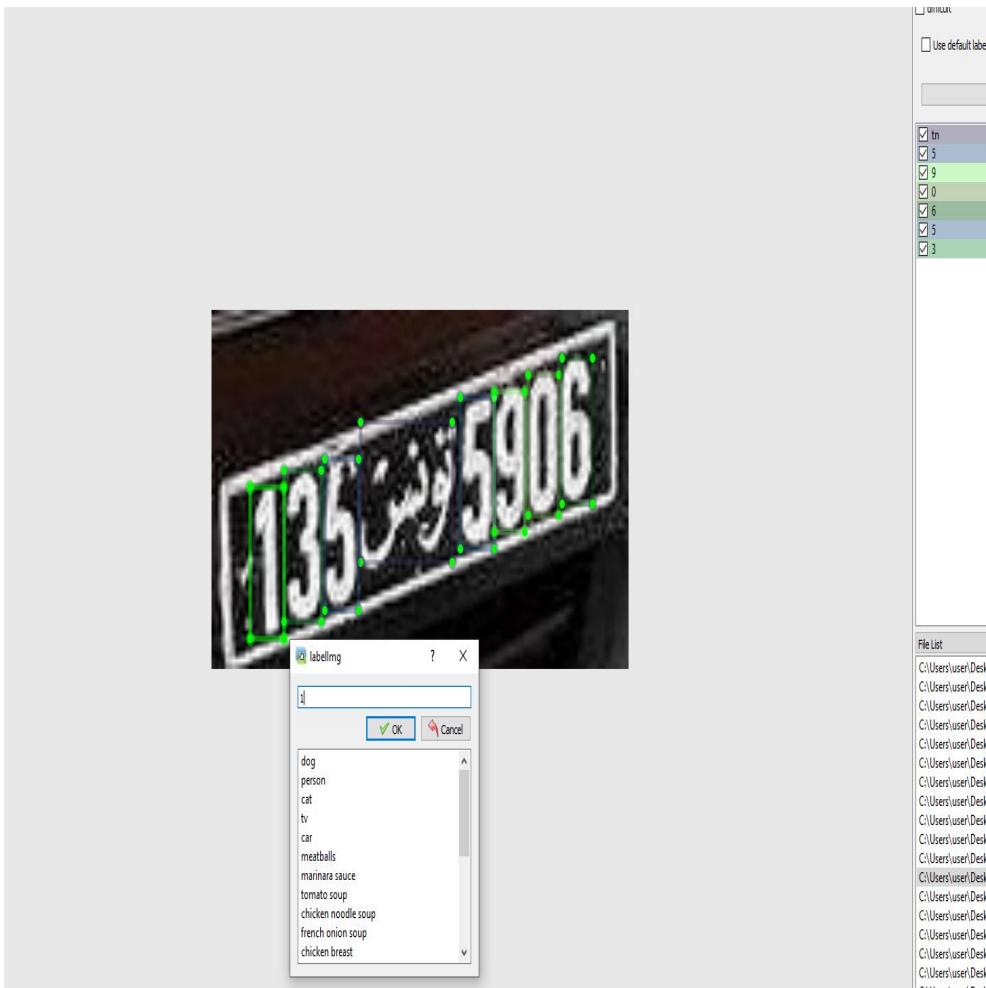


FIGURE 2.30 : Annotage des images de dataset

Par suite nous allons répéter le même processus suivi dans l'entraînement du modèle précédent sans aucun changement mais avec une nouvelle dataset. On lance l'entraînement du modèle. Après acheminer l'entraînement, le modèle atteint une précision moyenne de

```
mean_average_precision (mAP@0.5) = 0.999466
Saving weights to backup//custom-yolov4-tiny-detector_22000.weights
Saving weights to backup//custom-yolov4-tiny-detector_last.weights
Saving weights to backup//custom-yolov4-tiny-detector_final.weights
```

FIGURE 2.31 : Evaluation de nouveau modèle

Comme précédemment, nous aurons télécharger les 3 fichier de poids, cfg et names.

Configuration de Opencv La bibliothèque Opencv procure un outil pour mettre nos modèles entraînés en action. On parle ici de module DNN qui assure une exécution rapide Le module OpenCV DNN ne peut faire l'inférence des modèles de deep learning que sur vidéos et les images. On ne peut pas utiliser le module dnn pour l'entraînement d'un modèle. L'une des meilleures choses du module OpenCV DNN est qu'il est hautement optimisé pour les processeurs Intel. Nous pouvons obtenir de bons FPS lors de l'exécution d'inférences sur des vidéos en temps réel pour les applications de classification d'image et de détection d'objets ainsi que les applications de segmentation d'images. Nous obtenons souvent des FPS plus élevés avec le module DNN lorsque nous utilisons un modèle pré-entraîné à l'aide d'un framework spécifique. Par exemple, examinons la vitesse d'inférence de modèles détection pour différents frameworks.

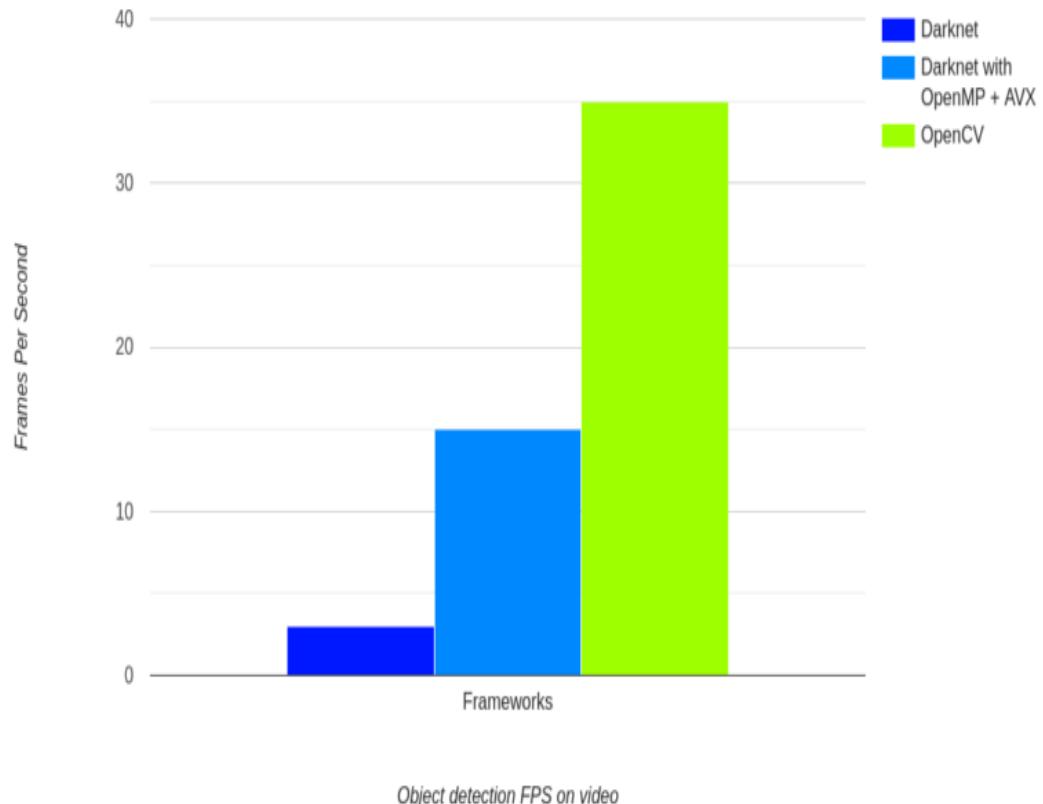


FIGURE 2.32 : fps obtenus avec l’inférence de chaque framework

Les benchmarks ci-dessus sont effectués en utilisant PyTorch 1.8.0, OpenCV 4.5.1 et TensorFlow 2.4. Tous les tests sont effectués sur Google Colab qui a des processeurs Intel Xeon 2.3Ghz. On peut voir que sur la même vidéo, le module DNN d’OpenCV tourne à 35 FPS alors que Darknet compilé avec OpenMP et AVX tourne à 15 FPS. Et Darknet (sans OpenMP ni AVX) Tiny YOLOv4 est le plus lent, fonctionnant à seulement 3 FPS. C’est une énorme différence étant donné que nous utilisons les modèles d’origine Darknet Tiny YOLOv4 dans les deux cas. Les quelques graphiques ci-dessus montrent OpenCV optimisé et à quelle vitesse il est pour l’inférence de réseau neuronal

Le module OpenCV DNN prend en charge de nombreux frameworks d’apprentissage en profondeur populaires. Voici les frameworks d’apprentissage en profondeur pris en charge par le module OpenCV DNN. Caffe, Tensorflow, Darknet, torch et pytorch Avec opencv dnn , pour faire l’inférence d’un modèle, pas besoin d’installer le framework associé à ce modèle.

Configuration de OpenCV pour utilisation de GPU Si nous ne faisons pas l'installation de OpenCV pour avoir le support GPU NVIDIA activé, OpenCV utilisera toujours le CPU pour l'inférence. La solution pour surmonter cette obstacle est de compiler OpenCV depuis sa source au lieu de le télécharger avec un simple commande python (par exemple pip install) . Ce processus d'installation manuelle a besoin de l'aide du logiciel CMAKE afin de permettre au module dnn d'utiliser les ressources de la carte graphique et par suite assurer un plus grand nombre de fps.

- On télécharge et installe Anaconda.
- On télécharge et install visual studio. Pendant l'installation il faut pas oublier de cocher la case "Desktop development with c++"

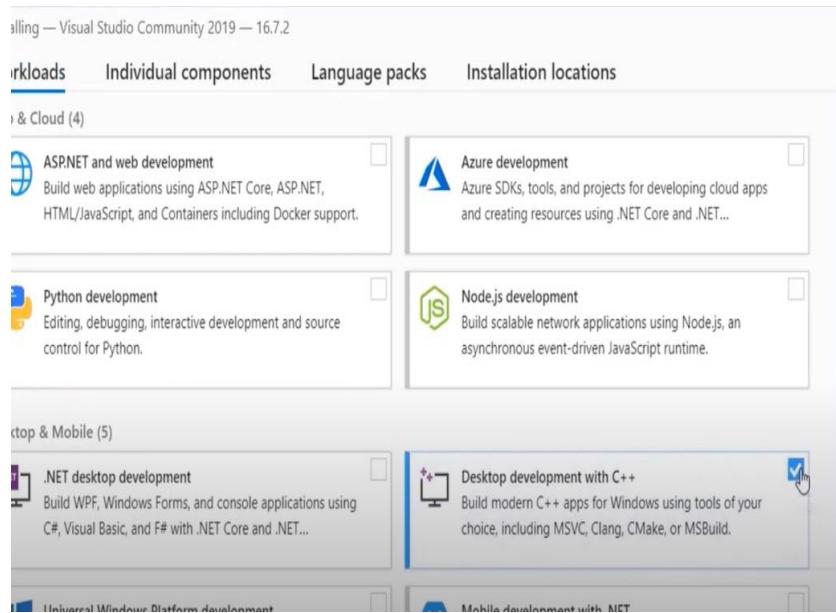


FIGURE 2.33 : installation de visual studio

- Tout d'abord , on télécharge Cuda 11.0 et on l'installe avec les options par défaut. Ensuite on télécharge cudnn sous forme un fichier d'extension .zip. On le décomprime et on copie tout son contenu sous le dossier de Cuda.
- On télécharge le logiciel Cmake et on l'installe toujours avec les paramètres par défaut.
- On crée un dossier sous "C :" et on le nomme par exemple "OpenCV_Build"
- On télécharge les fichiers source de OpenCV 4.5.1 depuis le siteweb officiel de OpenCV. Puis on télécharge OpenCV_contrib de la répertoire Github

associé à Opencv. Les deux fichiers qui ont l'extension .zip seront téléchargés dans le dossier OpenCV_Build créé précédemment et on extrait le contenu de chaque fichier

- On lance Cmake et on donne les chemin "C:/OpenCV_Build/opencv-4.5.1" et "C:/OpenCV_Build/build" dans les cases appropriés.

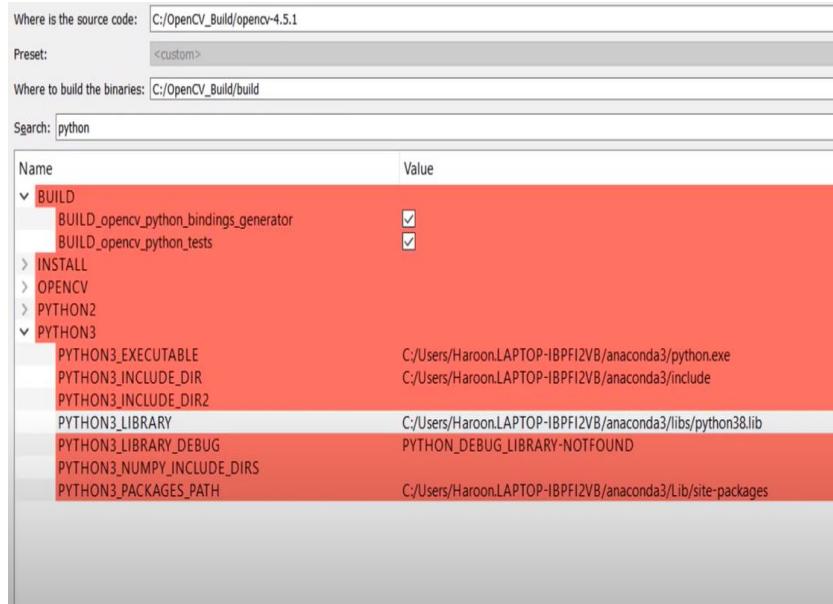


FIGURE 2.34 : Interface de Cmake

- Si on jette un coup d'oeil sur la section 'Opencv modules', on observe que la compilation depuis la source de Opencv ne peut pas être faite avec python3.

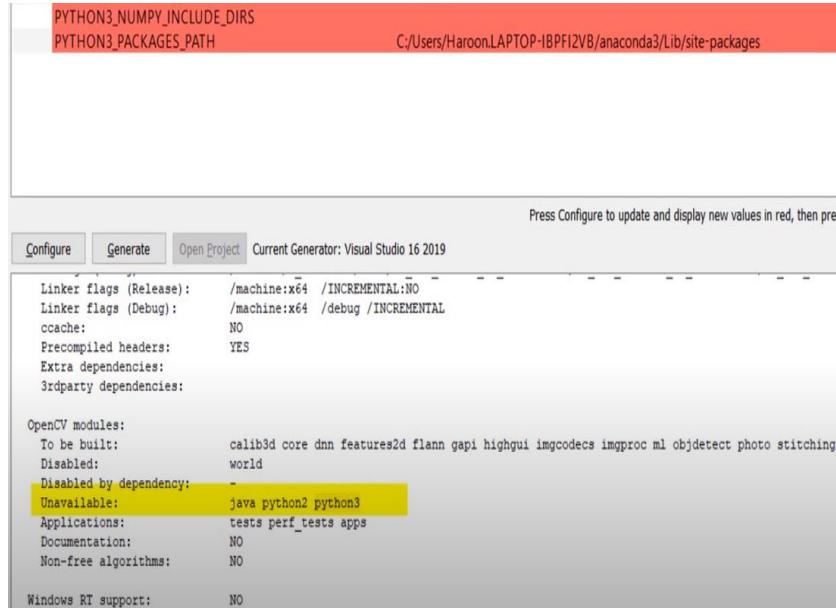


FIGURE 2.35 : Section Opencv modules

La solution pour ce problème est de mettre à jour la version de numpy pré-installée dans l'environnement de base de Anaconda.

```
(base) C:\WINDOWS\system32>pip install --upgrade numpy
Collecting numpy
  Using cached numpy-1.19.5-cp38-cp38-win_amd64.whl (13.3 MB)
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.19.2
    Uninstalling numpy-1.19.2:
      Successfully uninstalled numpy-1.19.2
Successfully installed numpy-1.19.5
(base) C:\WINDOWS\system32>
```

FIGURE 2.36 : Mise à jour de numpy

- Ensuite on retourne à l'interface de Cmake et on presse sur le bouton "Configure"
- Maintenant, on observe une autre fois la section 'Opencv modules' et on voit finalement qu'on peut réaliser la compilation depuis source de Opencv avec python3



FIGURE 2.37 : Section Opencv modules après mise à jour de numpy

- Puis, on aborde la configuration par cocher ces différentes paramètres
 - WITH_CUDA
 - BUILD_OPENCV_DNN
 - OPENCV_DNN_CUDA
 - ENABLE_FAST_MATH
 - BUILD_OPENCV_WORLD
- Dans la case "OPENCV_EXTRA_MODULES_PATH" on insère le chemin "C:/OpenCV_Build/opencv_contrib-4.5.1/modules" et on presse le bouton "Configure"
- Une fois la configuration terminée, on coche le paramètre "CUDA_FAST_MATH"
- Dans le paramètre "CUDA_ARCH_BIN", on trouve une série de nombres séparés par virgule et chacun représentant l'architecture d'un ensemble de cartes graphiques". En consultant la page de Cuda dans wikipédia, on trouve que 6.1 correspond à la carte graphique gtx1050ti de notre machine.

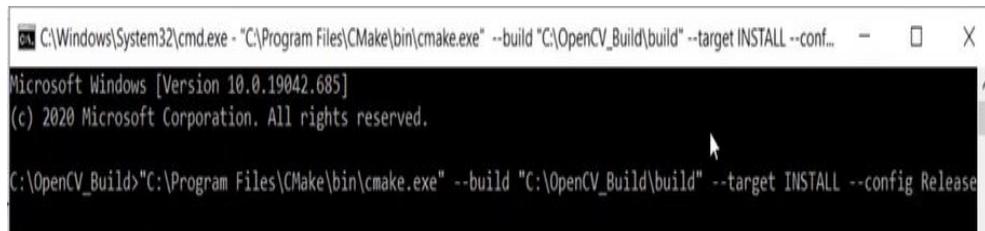
6.0		GP100	Quadro GP100
6.1	Pascal	GP102, GP104, GP106, GP107, GP108	Nvidia TITAN Xp, Titan X, GeForce GTX 1080 Ti, GTX 1080, GTX 1070 Ti, GTX 1070, GTX 1060, GTX 1050 Ti, GTX 1050, GT 1030, GT 1010, MX350, MX330, MX250, MX230, MX150, MX130, MX110
		GP108 ^[1]	
		GV100	NVIDIA TITAN V
7.0			Quadro GV100

FIGURE 2.38 : Page de Cuda en wikipédia

Donc on efface tous les nombre en laissant que 6.1

- Dans le paramètre "CMAKE_INSTALL_PREFIX", on insère le chemin suivant "C:/OpenCV_Build/install"
- Dans le paramètre "CMAKE_CONFIGURATION_TYPES" on élimine le mot "DEBUG"
- On exécute une autre fois le bouton "Configure". Après finir la configuration, on clique sur "Generate".

- Enfin, on ouvre la ligne de commande et on insère cet commande

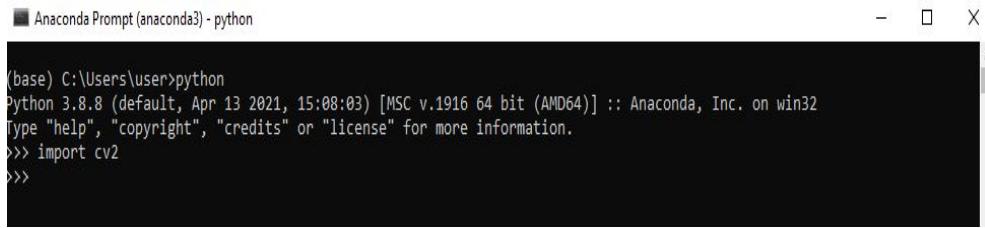


```
C:\Windows\System32\cmd.exe - "C:\Program Files\CMake\bin\cmake.exe" --build "C:\OpenCV_Build\build" --target INSTALL --config Release
```

FIGURE 2.39 : Commande d'installation d'Opencv dans windows

et on atteint jusqu'à son exécution s'achève

- On ouvre la console d'Anaconda et on se positionne dans l'environnement "Base". On exécute la commande "python" et on essaie d'importer opencv par cette ligne "import cv2".



```
(base) C:\Users\user>python
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>>
```

FIGURE 2.40 : Importation d'Opencv avec succès

Si on ne rencontre aucun erreur, cela signifie qu'on est dans le bon chemin et on peut utiliser le module DNN de Opencv en se basant sur les ressources de la carte graphique de notre machine au lieu des ressources du processeur et par suite on gagne plus de fps durant l'exécution de notre programme.

Toute configuration faite, nous pouvons avoir le module dnn capable d'utiliser le gpu

utilisation des moèles à l'aide de Opencv L'utilisation d'un modèle de détection avec DNN suit ce flux :

- Initialisation du module DNN avec paramètres et architecture de modèle et spécifier les noms de classes
- Effectuer la passe en avant sur l'image avec le module
- Traitement des résultats.

Avant d'être capable d'utiliser le module DNN, nous devons l'initialiser par une de ces fonctions :

- cv2.dnn.readNetFromTensorFlow pour initialiser des modèles associés à la framework Tensorflow
- cv2.dnn.readNetFromDarknet pour initialiser des modèles associés à la framework Darknet
- cv2.dnn.readNetFromTorch pour la création de modèles Pytorch

Ensuite nous devons spécifier les noms d'objets que notre modèle a été entraîné à identifier. Nous les stockons par exemple dans une liste nommée "classes".

Maintenant, pour exécuter notre en se basant sur le module cv2.dnn, nous devons transmettre les noms des couches pour lesquelles la sortie doit être calculée. La fonction getUnconnectedOutLayers() renvoie les indices des couches de sortie du réseau.

nous devons faire un pré-traitement de l'image d'entrée afin d'obtenir des prédictions correctes. Pour cette tâche, le module cv2.dnn nous fournit deux fonctions : blobFromImage et blobFromImages. Ces deux fonctions s'occupent de la mise à l'échelle, la soustraction moyenne et l'échange de canaux, ce qui est facultatif.

la fonction blobFromImage crée un blob en 4 dimensions à partir de l'image. En option, redimensionne et recadre l'image à partir du centre, soustrait les valeurs moyennes, met à l'échelle les valeurs par facteur d'échelle et permute les canaux bleu et rouge. En effet, Un blob est juste une collection d'images avec les mêmes dimensions spatiales (la largeur et la hauteur), la même profondeur (nombre de canaux), qui ont subi le même prétraitement. Regardons la signature de cette méthode : blob = cv.dnn.blobFromImage(image, scalefactor, size, mean, swapRB)

- image : C'est l'image d'entrée lise en utilisant la fonction cv2.imread
- scalefactor : cette valeur dont elle égale à 1 par défaut met l'image à l'échelle de la valeur fournie. 1 signifie qu'aucune mise à l'échelle n'est effectuée.
- size : Il s'agit de la taille à laquelle l'image sera redimensionnée. il s'agit de 416*416 pour les modèles yolo
- mean : L'argument moyen est assez important. Ce sont en fait les valeurs moyennes qui sont soustraites des canaux de couleur RVB de l'image. Cela normalise l'entrée et rend l'invariance d'entrée finale à différentes échelles d'éclairage.
- swapRB : OpenCV lit l'image au format BGR, et pour la détection d'objets, les modèles s'attendent généralement à ce que l'entrée soit au format RVB. Ainsi, l'argument swapRB fait une permutation entre les canaux R et B de l'image, la transformant au format RVB.

Après nous allons réaliser la propagation vers l'avant de l'entrée à travers le modèle model.setInput(blob) output = model.forward()

Nous sommes tous prêts à boucler les détections, supprimer les cadres de délimitation faibles qui se chevauchent à l'aide de fonction cv2.dnn.NMSBoxes() et obtenir les coordonnées de chaque rectangle de délimitation autour de chacun des objets détectés. Ce qui suit est le code pour boucler les détections.

```

Width = image.shape[1]
Height = image.shape[0]
class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.5
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence >= 0.1:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([round(x), round(y), round(w), round(h)])
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
#arr = np.empty((0, 2), dtype=object)
np.warnings.filterwarnings('ignore', category=np.VisibleDeprecationWarning)
dt=np.dtype('object,int')
arr = np.empty((0,2), dtype=dt)

for i in indices:
    i = i[0]
    box = boxes[i]
    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]

```

FIGURE 2.41 : détections résultantes du modèle

Structure de programme La réalisation du programme final suit ces étapes :

- Initialisation d'un modèle yolov4-tiny avec le module DNN. Cela implique l'importation de 2 fichiers .WEIGHTS et .cfg. Puis On cherche les noms de classes dans une liste.
- Capture vidéo avec la fonction cv2.videoCapture()
- Tant que le vidéo marche , le modèle de détection de plaque est exécuté chaque image ou "frame" de vidéo. Si dans un frame, une plaque d'immatriculation est détecté, nous dessinons un rectangle encadrant ce plaque.et

le système crée une nouvelle image à partir de la partie entouré par le rectangle.

- Ensuite, on passe l'image contenant seulement la plaque à une fonction qui va s'occuper de la détection des chiffres numériques et le mot "tounes" de la plaque d'immatriculation tunisienne. Cette fonction existe dans un fichier python externe. Dans cette fonction, on crée un nouvel modèle yolov4-tiny en suivant le même démarche suivi dans la création de modèle qui a la détection de plaque comme tâche.
- Enfin la fonction retourne la liste des noms d'objets détectés dans la plaque et triés de droite à gauche en se basant sur la position x de chaque objet comme critère de tri
- Le programme forme une mot à partir de liste retournée par la fonction de détection de chiffres. Ce mot va contenir le numéro de matricule sous forme xxtnyy. Par suite, il effectue une recherche une recherhce dans un fichier csv contenant tous les numéro de matricules tunisiennes. afin de vérifier si ce numéro correspond à une voiture volée ou dont les papiers ne sont pas réglés.
- Si le numéro de matricule détecté n'est pas susceptible, le système l'affiche au dessus de rectangle contenant la plaque d'immatriculation en vert. Si non il affiche le numéro de matrciule en rouge et enregistre l'image ou frmae courante.

Réisation La dernière tâche de ce projet est de tester tout le travail précédent. Pour cela nous allons tester le projet sur des vidéos enregistrés de circulation routière.



FIGURE 2.42 : Détection précise



FIGURE 2.43 : Autre cas de détection précise

Si on a en exécution seulement le modèle de détection de plaque d'immatriculation, on obtient 30 fps. Ce nombre de fps diminue fortement lorsque une plaque est détectée et soumise à un deuxième modèle de détections des chiffres. La précision de détection augmente quand la voiture (et par suite la plaque d'immatriculation) n'est pas trop loin de caméra.