الجمهورية الجزائرية الديمقراطية الشعبية

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي و البحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة أبي بكر بلقايد – تلمسان –

Université Aboubakr Belkaïd – Tlemcen –

Faculté de TECHNOLOGIE

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**

En : Télécommunications

Spécialité : Réseaux Mobiles et Services de Télécommunications

Par : HADBI Widad

Sujet

Reconnaissance d'une plaque d'immatriculation via un Smartphone Android

Soutenu publiquement, le 12 / 06 /2017, devant le jury composé de :

M. MERZOUGUI Rachid	MCA	Univ. Tlemcen	Président
M. MOUSSAOUI Djilali	MAA	Univ. Tlemcen	Examinateur
M. HADJILA Mourad	MCB	Univ. Tlemcen	Encadreur

Remerciements

Au terme de ce travail, je tiens à remercier

Notre ***DIEU***

tout puissant de m'avoir donné la foi, la force et le courage

Mon encadreur Monsieur ***HADJILA MOURAD***

Pour ses qualités humaines et professionnelles, pour son encadrement, ses directives,

Ses remarques constructives, et sa disponibilité.

Monsieur ***MERZOUGUI RACHID***

Pour avoir accepté de présider le jury de ma soutenance.

Monsieur ***MOUSSAOUI DJILALI*** *mon examinateur*

De me faire l'honneur d'être membre examinateur du jury

Monsieur ***FEHAM MOUHAMED***

Pour avoir accepté d'être étudiant au spécialité RMST

Tous les personnels qui me soutien

Pour leurs encouragements continus et leurs aides précieuses.

HADBI WIDAD

Je dédie ce travail à

Mes parents,
Mon fiancé,
Ma sœur IBTISSEM,
Et tous mes amis et ma famille.

HADBI WIDAD

Table des matières

Chapitre III: Réalisation et mise en œuvre

Table des figures

Liste des tableaux

Introduction générale

Introduction générale

Aujourd'hui, le téléphone mobile n'est pas seulement utilisé pour la communication, et l'échange des messages courts comme autre fois, De nouveaux usages sont apparus tels que les jeux, la lecture audio, etc. Sur un plan plus pratique, Le téléphone mobile dispose des variétés d'utilisations et l'appareil est devenu plus utile que jamais auparavant. L'essor du mobile a connu une évolution considérable en particulier dans les pays en développement. Les réseaux de télécommunication mobiles sont également en expansion : le nombre d'utilisateurs de mobile est en constante progression et la couverture territoriale est largement répandue.

Actuellement la société Apple à travers son Smartphone « iPhone», sa tablette PC « iPad » et son système d'exploitation « iPhone OS» est en forte concurrence avec la communauté Open Handset Alliance (OHA) qui englobe Google, Motorola, HTC, Samsung, etc. Cette dernière équipe ses terminaux mobiles par le système d'exploitation mobile « Android OS».

Grâce à la généralisation des téléphones portables tactiles à écrans larges ainsi qu'au développement des logiciels et des réseaux, les applications mobiles sont capables de satisfaire un large éventail de besoins, sans oublier un développement technologique continu qui en fait un outil encore plus essentiel.

L'intégration massive des technologies de l'information dans tous les aspects de la vie moderne a provoqué la demande de traitement des véhicules en tant que ressources conceptuelles dans les systèmes d'information. Étant donné qu'un système d'information autonome sans données n'a aucun sens, il était également nécessaire de transformer l'information sur les véhicules entre la réalité et les systèmes d'information. Cela peut être réalisé par un agent humain ou par un équipement intelligent spécial qui est capable de reconnaître les véhicules par leurs plaques d'immatriculation dans un environnement réel et de le refléter en ressources conceptuelles.

En raison de cela, diverses techniques de reconnaissance ont été développées et les systèmes de reconnaissance de plaque d'immatriculation sont utilisés aujourd'hui dans diverses applications de trafic et de sécurité, telles que le stationnement, l'accès et le contrôle aux frontières, ou le suivi des voitures volées.

Ce projet présente la conception d'un système ANPR, permettant la détection automatique des plaques d'immatriculation et la reconnaissance des caractères sans intervention humaine en utilisant les réseaux de neurones artificiels, SVM (Support Vector Machines) et la reconnaissance optique des caractères.

Pour répondre aux besoins et objectifs de ce projet de fin d'étude, j'ai organisé mon travail en trois chapitres :

- Le premier chapitre décrit les Smartphones, leurs systèmes d'exploitation et plus particulièrement le système Android.
- Le chapitre II présente les étapes nécessaires à la création d'une application pour la reconnaissance automatique des plaques numériques.
- Le chapitre III est une description détaillée de la conception et l'implémentation de système ANPR.

Chapitre I

Smartphones et le système d’exploitation mobile Android

I.1. Introduction

Le Smartphone est un compagnon de tous les instants et un véritable ordinateur de poche pour gérer le quotidien. Ce dernier est passé en l'espace de quelques années d'un outil professionnel à un objet de divertissement grand public.

Avec l'explosion des ventes de Smartphones ces dernières années, Android a pris une place importante dans la vie quotidienne.

Dans ce chapitre, nous allons essayer de voir les concepts de base des Smartphones et nous nous intéressons plus précisément à la plateforme de Google Inc. Android

I.2. Définition de Smartphone

Un Smartphone est un terme d'origine anglo-saxonne, signifiant littéralement «téléphone intelligent ». Il désigne un téléphone mobile évolué disposant des fonctions d'un assistant numérique personnel, d'un appareil photo numérique et d'un ordinateur portable. La saisie des données se fait le plus souvent par le biais d'un écran tactile ou d'un clavier **[1]**.

I.3. Historique

Paradoxalement à l'évolution des technologies, l'avènement du Smartphone a été relativement long à se mettre en place. Les principaux acteurs du marché (à l'époque, Nokia, Ericsson, Motorola) se faisant hésitant pour se lancer du fait des nombreuses contraintes de normalisation technique.

En effet, si à l'époque, chaque mobile GSM possédait son propre environnement d'exploitation, aucun d'entre eux n'était prêt à laisser rentrer Microsoft, à l'époque le seul acteur capable de lancer une plateforme dédiée mobile, permettant d'échanger des données et entièrement compatible avec les ordinateurs sous Windows **[1]**.

I.4. Les principaux acteurs du marché des Smartphones

Aujourd'hui, les acteurs principaux du marché des Smartphones, sont Apple, constructeur de l'iPhone et propriétaire de son OS (iOS), Google propriétaire de son OS Android (Open source), Microsoft, propriétaire de son environnement Windows Mobile et les nombreux constructeurs de Smartphones, anciens ou récents, ayant pratiquement tous abandonné leur environnement propriétaire, au profit de Windows Mobile ou d'Android **[1]**.

- Le tableau suivant nous donne la part du marché des principaux systèmes d'exploitation et le nombre d'expédition mondiale de Smartphones **[4]** :

Systèmes d'exploitation :	T4 2016		T4 2015	
	Smartphones	Part de marché	Smartphones	Part de marché
Android	352,669.9	81.7	325,394.4	80.7
iOS	77,038.9	17.9	71,525.9	17.7
Windows	1,092.2	0.3	4,395.0	1.1
BlackBerry	207.9	0.0	906.9	0.2
Autres	530.4	0.1	887.3	0.2
Totale :	**431,539.3**	**100.0**	**403,109.4**	**100.0**

Tableau I.1 Nombre des Smartphones et part de marché mondiale des OS mobiles

Dans le marché des systèmes d'exploitation pour Smartphones, l'Android de Google a étendu son avance en enregistrant 82% du marché total au quatrième trimestre de 2016.

Il est devenu l'OS mobile le plus utilisé du monde. Cela fait déjà plusieurs années que cette situation persiste.

I.5. Systèmes d'exploitation pour les Smartphones

Les systèmes d'exploitation mobiles (OS) peuvent être définis comme les logiciels permettant à un Smartphone ou un téléphone mobile basique de fonctionner. Ils permettent de ce fait aux utilisateurs de pouvoir passer un appel téléphonique, naviguer sur leurs téléphones parmi toutes les rubriques, télécharger des applications ou encore paramétrer et personnaliser leurs Smartphones **[2]**.

Il existe sur le marché plusieurs systèmes d'exploitation. Certains OS fonctionnent sur plusieurs marques de mobile, ils sont appelés système ouvert (comme Android), d'autres ne fonctionnent que sur le mobile vendu par le fabriquant, on parle alors de système fermé (iOS d'apple, Blackberry OS, etc) **[3]**.

I.5.1. IOS d'Apple Inc.

IOS est le nom du système d'exploitation mobile d'Apple qui est dérivé de Mac OS X. Il est utilisé par les générations successives d'iPhone depuis la sortie du premier modèle en 2007 puis par l'iPad depuis 2010 **[5]**.

I.5.2. Windows Phone de Microsoft

Windows Phone est le nom du système d'exploitation pour Smartphones que Microsoft a publié en octobre 2010. Il est venu remplacer Windows Mobile en introduisant une interface utilisateur totalement redessinée et pensée pour les terminaux à écran tactile. Celle-ci se compose de tuiles dynamiques dont l'affichage évolue en temps réel selon l'activité de l'application concernée **[6]**.

I.5.3. BlackBerry OS de Research In Motion (RIM)

BlackBerry OS est un système d'exploitation propriétaire pour téléphone mobile de la gamme BlackBerry, conçu par la société canadienne Research In Motion (RIM), maintenant connue sous le nom de Blackberry. Il s'agit d'un système multitâche. Il est caractérisé par le charme des claviers AZERTY, très pratiques pour l'envoi de messages écrits **[7]**.

I.5.4. Android

Android est un système d'exploitation open-source pour Smartphones, PDA et autres terminaux mobiles, conçu par Android, une start-up rachetée par Google en juillet 2005. Il existe d'autres types d'appareils possédant ce système d'exploitation tels que les téléviseurs et les tablettes.

Figure I.1 Logo d'Android

Afin de promouvoir ce nouveau système d'exploitation ouvert, Google a su fédérer autour de lui un consortium d'une trentaine d'entreprises : l'Open Handset Alliance (OHA) créée officiellement le 5 novembre 2007. Toutes ces entreprises interviennent, plus ou moins directement, dans le marché de la téléphonie mobile **[8]**.

I.6. Etude de la plateforme Android

Depuis sa création, la popularité d'Android a toujours été croissante. C'est au quatrième trimestre 2010 qu'Android devient le système d'exploitation mobile le plus utilisé au monde, devançant Symbian (le système d'exploitation de Nokia avant qu'ils optent pour Windows Phone). Désormais, on le retrouve non seulement dans les tablettes et Smartphones, mais aussi dans les téléviseurs, les consoles de jeux, les appareils photos, etc **[9]**.

Pour cela, nous avons choisi le système Android pour l'utiliser comme système de mobile qu'on veut contrôler.

1.6.1. Différentes versions

Android est régulièrement mis à jour. La première version d'Android a été distribuée fin 2007. Depuis, de nombreuses versions ont vu le jour.

La figure suivante nous montre les différentes versions d'Android :

Figure I.2 Evolution des versions d'Android

En 2007, le 5 novembre, l'OHA a été officiellement annoncée, ainsi que son but : développer des standards open sources pour appareil mobile. Le premier standard annoncé a été Android, une plateforme pour appareils mobiles basée sur un kernel linux 2.6.

En octobre 2008, apparait la première version d'Android qui n'avait pas reçu de nom. Cette version s'est avérée être la β du système.

Android 1.5 « CupCake » a été publiée le 30 avril 2009. Cette version corrigea le manque d'API et rendit le système plus utilisable.

À partir de celle-ci, il a été décidé de nommer les différentes versions du système avec des noms de gâteaux (peut-être que les entreprises étaient gourmandes), en suivant l'ordre alphabétique.

Android 1.6, 2.0 et 2.1 « Donut » et **« Eclair »** ont apporté d'importantes améliorations respectivement sur les fonctionnalités et sur l'interface graphique du système.

Android 2.2 « Froyo » a fortement mis l'accent sur la synergie avec Internet. L'envoi d'applications et de liens instantanés depuis un ordinateur est désormais possible. Aussi, Google annonce-t-elle que le navigateur chrome intégré à Android 2.2 est le navigateur mobile le plus rapide au monde grâce à l'intégration du moteur JavaScript V8.

Android 2.3 « Gingerbread » n'a pas apporté grand-chose, mais surtout des améliorations sur la prise en charge et le support de la VoIP, le NFC, le capteur frontal sur les appareils concernés et un passage au système de fichiers ext4 que les amateurs de GNU/Linux connaissent fort bien.

Android 3.0 « Honeycomb » est spécialement étudié pour les tablettes tactiles. Les premiers modèles ont été annoncés au CES 2011. On y apprend quelques nouveautés comme la prise en charge de la vidéo-conférence via Gtalk, la nouvelle interface Gmail ou encore le lecteur de livre électronique Google.

Android 4.0 « Ice Cream Sandwich » a rajouté encore plus de fonctionnalités aux terminaux. Pour le développement, cette nouvelle version d'Android a proposé de nouveaux composants permettant de réaliser des applications avec une ergonomie plus adaptée aux tablettes tactiles. **[8]**.

Android 4.1 « Jelly Bean » une version basée sur le noyau Linux 3.0.31 et dont la principale nouveauté est une amélioration des fonctionnalités et des performances de l'interface utilisateur.

Android 4.4 « KitKat » a été annoncé le 31 octobre 2013 d'abord sur le tout nouveau Nexus 5 puis déployé sur les autres terminaux. Il s'agit encore aujourd'hui de l'une des versions d'Android les plus couramment utilisées.

Android 5.0 « Lollipop » a été officiellement nommé ainsi le 15 octobre 2014 par Google. De nombreuses mises à jour sont depuis en cours vers cette nouvelle version majeure qui fut disponible publiquement le 3 novembre, et qui étend sa disponibilité sur de nouveaux supports telles que la télévision, la voiture ou les montres connectées.

Android 6.0 « Marshmallow » est une nouvelle version d'Android qui symbolise tout d'abord le passage à une nouvelle unité, mais aussi à une nouvelle friandise. Un changement assez radical pour une version qui était supposée originellement être la 5.2 **[10]**.

Android 7.0 « Nougat » a été annoncée lors de la Google I/O 2016, Android Nougat (précédemment appelée Android N avant de recevoir un nom définitif) correspond à Android 7.0. Android 7.0 fut rapidement suivi d'une version 7.1.

Depuis la sortie d'Android Nougat, Google a modifié le rythme de mises à jour en passant aux mises à jour trimestrielles, afin de sortir une nouvelle version du système chaque trimestre **[11]**.

- Le tableau suivant nous montre les versions d'Android les plus utilisés **[12]**:

Version	Nom	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.1%
4.1.x	Jelly Beans	16	4.0%
4.2.x		17	5.9%
4.3		18	1.7%
4.4	KitKat	19	22.6%
5.0	Lollipop	21	10.1%
5.1		22	23.3%
6.0	Marshmallow	23	29.6%
7.0	Nougat	24	0.5%
7.1		25	0.2%

Tableau I.2 Distribution des différentes versions d'Android

Les principales versions d'Android déployées restent Kitkat, Lollipop et Marshmallow. La plus ancienne des trois, Android Kitkat (4.4), représente même encore 22,6% du parc.

Lollipop (5.x) compte pour 33,4% du parc Android installé, talonnée par Marshmallow qui représente 29,6% de cette base.

1.6.2. La philosophie et les avantages d'Android

1.6.2.1. Open source

Le contrat de licence pour Android respecte les principes de l'open source, c'est-à-dire que vous pouvez à tout moment télécharger les sources et les modifier selon vos goûts ! Bon, je ne vous le recommande vraiment pas, à moins que vous sachiez ce que vous faites... Notez au passage qu'Android utilise des bibliothèques open source puissantes, comme par exemple SQLite pour les bases de données et OpenGL pour la gestion d'images 2D et 3D.

1.6.2.2. Gratuit (ou presque)

Android est gratuit, autant pour vous que pour les constructeurs. S'il vous prenait l'envie de produire votre propre téléphone sous Android, alors vous n'auriez même pas à ouvrir votre porte-monnaie (mais bon courage pour tout le travail à fournir !). En revanche, pour poster vos applications sur le Play Store, il vous en coûtera la modique somme de 25$. Ces 25$ permettent de publier autant d'applications que vous le souhaitez, à vie !

1.6.2.3. Facile à développer

Toutes les API mises à disposition facilitent et accélèrent grandement le travail. Ces APIs sont très complètes et très faciles d'accès. De manière un peu caricaturale, on peut dire que vous pouvez envoyer un SMS en seulement deux lignes de code (concrètement, il y a un peu d'enrobage autour de ce code, mais pas tellement).

1.6.2.4. Facile à vendre

Le Play Store (anciennement Android Market) est une plateforme immense et très visitée ; c'est donc une mine d'opportunités pour quiconque possède une idée originale ou utile.

1.6.2.5. Flexible

Le système est extrêmement portable, il s'adapte à beaucoup de structures différentes. Les Smartphones, les tablettes, la présence ou l'absence de clavier ou de trackball, différents processeurs... On trouve même des fours à micro-ondes qui fonctionnent à l'aide d'Android ! Non seulement c'est une immense chance d'avoir autant d'opportunités, mais en plus Android est construit de manière à faciliter le développement et la distribution en fonction des composants en présence dans le terminal (si votre application nécessite d'utiliser le Bluetooth, seuls les terminaux équipés de Bluetooth pourront la voir sur le Play Store).

1.6.2.6. Ingénieux

L'architecture d'Android est inspirée par les applications composites, et encourage par ailleurs leur développement. Ces applications se trouvent essentiellement sur internet et leur principe est que vous pouvez combiner plusieurs composants totalement différents pour obtenir un résultat surpuissant. Par exemple, si on combine l'appareil photo avec le GPS, on peut poster les coordonnées GPS des photos prises **[9]**.

1.6.3. Architecture d'Android

Android est conçue pour des appareils mobiles au sens large. Nullement restreint aux téléphones, il ouvre d'autres possibilités d'utilisation des tablettes, des ordinateurs portables, des bornes interactives, des baladeurs…

La plate-forme Android est composée de différentes couches :

• un noyau Linux qui lui confère notamment des caractéristiques multitâches ;

• des bibliothèques graphiques, multimédias ;

• une machine virtuelle Java adaptée : la Dalvik Virtual Machine ;

• un framework applicatif proposant des fonctionnalités de gestion de fenêtres, de téléphonie, de gestion de contenu... ;

• des applications dont un navigateur web, une gestion des contacts, un calendrier… **[13]**

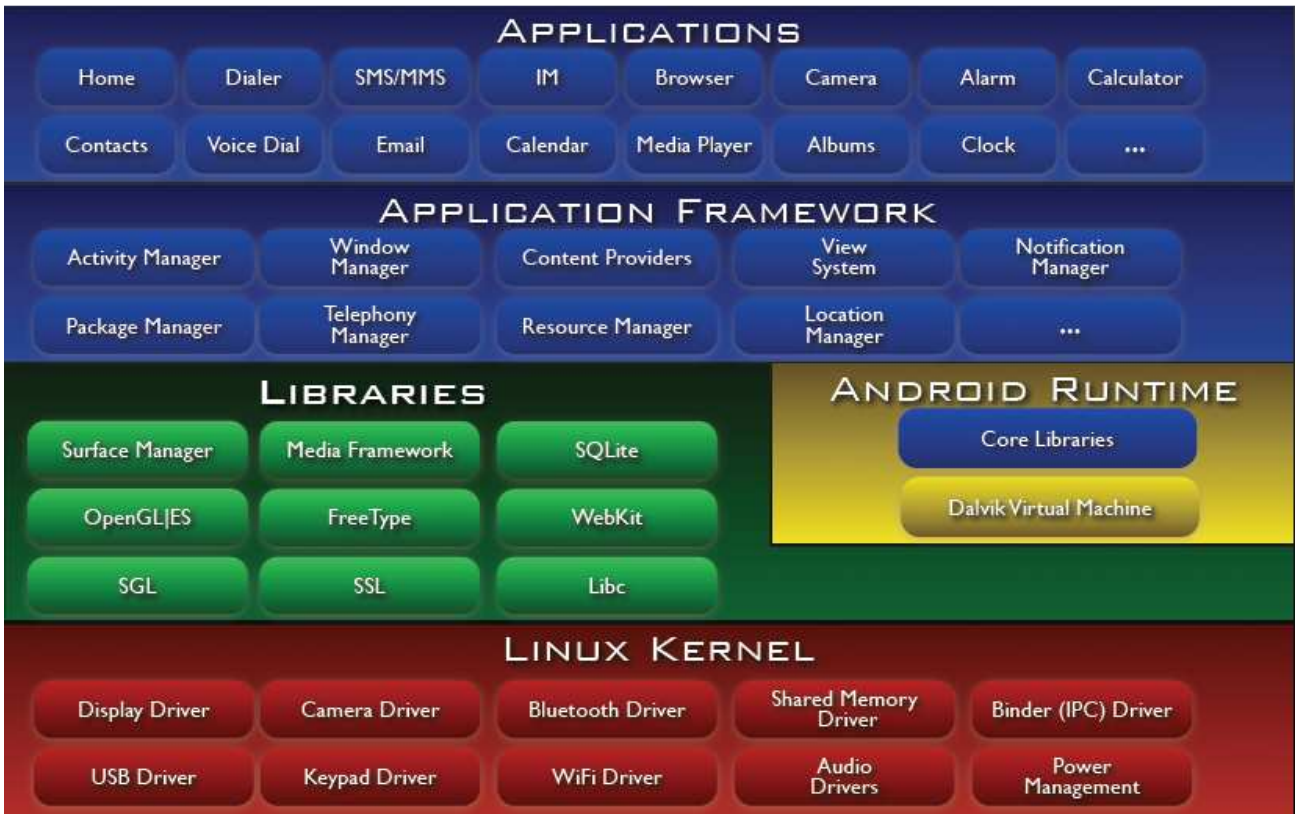

Figure I.3 Architecture de système Android

1.6.3.1. Premier niveau : noyau linux

Android est basé sur un kernel linux 2.6 → 3.2 mais ce n'est pas linux. Il ne possède pas de système de fenêtrage natif (X window system). La glibc n'étant pas supportée, Android utilise une libc customisée appelée Bionic libc.

Enfin, Android utilise un kernel avec différents patches pour la gestion de l'alimentation, le partage mémoire, etc. permettant une meilleure gestion de ces caractéristiques pour les appareils mobiles.

Android n'est pas linux mais il est basé sur un kernel linux. Pourquoi sur un kernel linux ? Le kernel linux a un système de gestion mémoire et de processus reconnu pour sa stabilité et ses performances.

- Le model de sécurité utilisé par linux, basé sur un système de permission, est connu pour être robuste et performant. Il n'a pas changé depuis les années 70.
- Le kernel linux fournit un système de drivers permettant une abstraction avec le matériel. Il permet également le partage de librairies entre différents processus, le chargement et le déchargement de modules à chaud.
- le kernel linux est entièrement open source et il y a une communauté de développeurs qui l'améliorèrent et rajoutent des drivers.

C'est pour les points cités ci-dessus que l'équipe en charge du noyau a décidé d'utiliser un kernel linux.

1.6.3.2. Deuxième niveau :

1.6.3.2.1. Bibliothèques C/C++

Android dispose d'un ensemble de librairies C / C++ utilisées par les différents composants du système Android. Elles sont offertes aux développeurs à travers le framework Android. En voici quelques-unes :

Système de bibliothèque C – une mise en œuvre dérivée de BSD de la bibliothèque C standard du système (libc), destinés aux systèmes embarqués basés sur Linux.

Comme cela a été dit précédemment, Android ne supporte pas la glibc, donc les ingénieurs d'Android ont développé une librairie C (libc) nommé Bionic libc. Elle est optimisée pour les appareils mobiles et a été développée spécialement pour Android.

Les ingénieurs d'Android ont décidé de développer une libc propre à la plateforme Android car ils avaient besoin d'une libc légère (la libc sera chargée dans chaque processus) et rapide (les appareils mobiles ne disposent pas de CPU puissant).

La Bionic libc a été écrit pour supporter les CPU ARM, bien que le support x86 soit présent. Il n'y pas de support pour les autres architectures CPU telles que PowerPC ou MIPS. Néanmoins, pour le marché des appareils mobiles, seulement l'architecture ARM est importante. Cette libc est sous licence BSD. Elle reprend une grande partie du code des glibc issue d'OpenBSD, FreeBSD et NetBSD.

Ces caractéristiques importantes sont :

- Elle pèse environ 200Ko, soit la moitié de la glibc.
- L'implémentation des pthreads (POSIX thread) a été complètement réécrite pour supporter les threads de la machine virtuelle Dalvik. De ce fait, la Bionic libc ne supporte pas les threads POSIX.
- Les exceptions C++ et les "wide char" ne sont pas supportés.
- Médiathèques – basée sur PacketVideo de OpenCore ; les librairies permettant la lecture et l'enregistrement audio et vidéo, ainsi que la gestion des fichiers image, y compris MPEG4, H.264, MP3, AAC, AMR, JPG et PNG.

  Le schéma ci-dessous décrit tous les éléments de l'architecture de ces médiathèques :

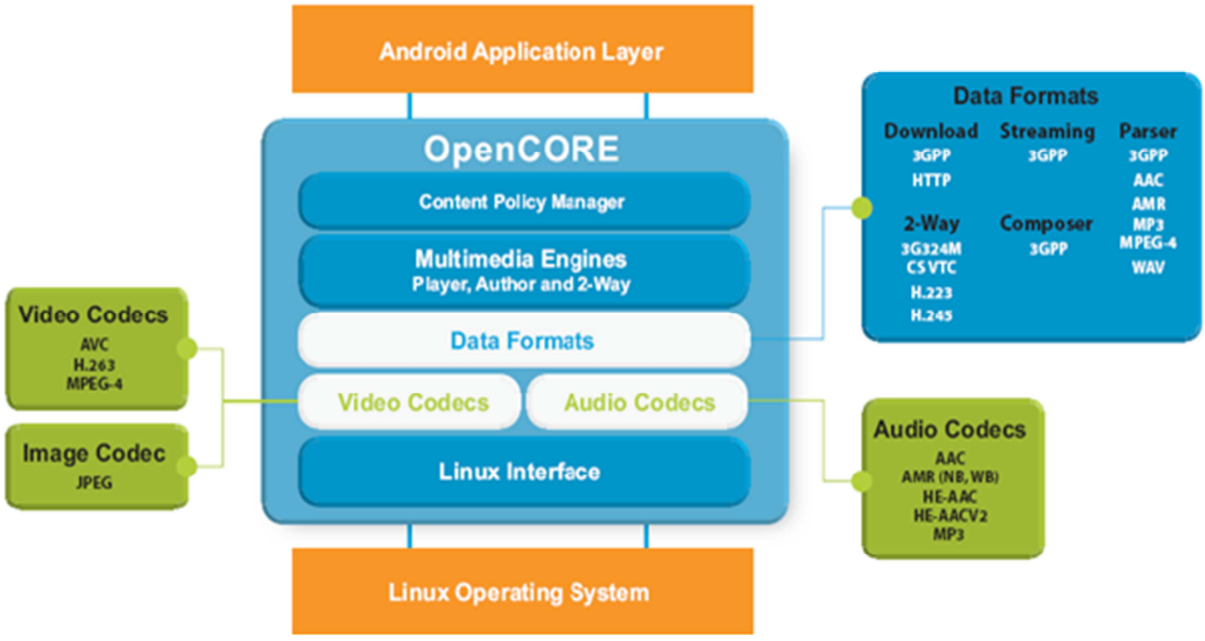

Figure I.4 Architecture de ces médiathèques

- Surface Manager – gère l'accès au sous-système d'affichage et de façon transparente.
- LibWebCore – Le navigateur web présent dans Android est basé sur le moteur de rendu sous licence BSD WebKit.

- WebKit est un moteur de rendu, qui fournit une "fondation" sur laquelle on peut développer un navigateur web. Il a été originellement dérivé par Apple du moteur de rendu KHTML pour être utilisé par le navigateur web Safari et maintenant il est développé par KDE project, Apple, Nokia, Google et d'autres. WebKit est composé de deux librairies : WebCore et JavascriptCore qui sont disponibles sous licence GPL.

  WebKit supporte le CSS, Javascript, DOM, AJAX. La dernière version a obtenu 100% au test Acid 3. La version de WebKit présent dans Android a été légèrement modifiée pour s'adapter aux appareils mobiles. Ainsi, le moteur de rendu basé sur WebKit présent dans Android supporte l'affichage sur une colonne.

- SGL – le moteur graphique 2D.
- Bibliothèques 3D – une implémentation basée sur OpenGL ES 1.0 API ; les bibliothèques utilisent l'accélération 3D matérielle (si disponible).
- FreeType – bitmap et vectoriel de rendu de police.
- SQLite – un moteur de base de données relationnelles puissant et léger, disponible pour toutes les applications.

1.6.3.2.2. Moteur d'exécution Android : Dalvik

Android inclut un ensemble de librairies de base offrant la plupart des fonctionnalités disponibles dans les bibliothèques de base du langage de programmation Java.

Chaque application Android s'exécute dans son propre processus, avec sa propre instance de la machine virtuelle Dalvik. Dalvik a été écrit pour que le dispositif puisse faire tourner plusieurs machines virtuelles de manière efficace. La machine virtuelle Dalvik exécute des fichiers dans l'exécutable Dalvik (.DEX), un format optimisé pour ne pas encombrer la mémoire. La machine virtuelle est la base de registres et fonctionne grâce aux classes compilées par un compilateur Java et transformées dans le format DEX.

La machine virtuelle Dalvik s'appuie sur le noyau Linux pour les fonctionnalités de base tels que le filetage et la gestion de la mémoire de bas niveau.

1.6.3.3. Troisième niveau : Framework applicatif

En fournissant une plateforme de développement ouverte, Android offre aux développeurs la possibilité de créer des applications extrêmement riches et innovantes. Les développeurs sont libres de profiter du matériel périphérique et informations sur la localisation d'accès, exécuter des services d'arrière-plan, définir des alarmes, ajouter des notifications à la barre d'état, etc.

Les développeurs ont un accès complet au même framework API utilisé par les applications de base. L'architecture d'application est conçue pour simplifier la réutilisation des composants ; n'importe quelle application peut publier ses capacités et n'importe quelle autre application peut alors faire usage de ces capacités (soumis à des contraintes de sécurité appliquées par le framework). Ce même mécanisme permet aux composants d'être remplacés par l'utilisateur.

Toutes les applications sous-jacentes forment un ensemble de services et de systèmes, y compris:

- Un jeu extensible de vues qui peuvent être utilisées pour construire une application.
- Des fournisseurs de contenu qui permettent aux applications d'accéder aux données d'autres applications (tels que les Contacts), ou de partager leurs propres données.
- Un gestionnaire de ressources.
- Un gestionnaire de notification qui permet à toutes les demandes d'afficher des alertes personnalisées dans la barre d'état.
- Un gestionnaire d'activité qui gère le cycle de vie des applications et propose une navigation commune.

1.6.3.4. Quatrième niveau : applications

Android est fourni avec un ensemble d'applications dont un client email, une application SMS, un calendrier, un service de cartographie, un navigateur, etc. tous écrits en JAVA **[8]**.

I.7. Conclusion

Dans ce chapitre, nous avons identifié les Smartphones et leurs principaux systèmes d'exploitation en détaillant le système Android, tout en présentant un bref historique, les fonctionnalités que nous pouvons trouver sur ce système d'exploitation et l'architecture d'Android, à savoir les principaux composants du système.

Chapitre II

Reconnaissance des plaques d'immatriculation à l'aide de SVM et des réseaux de neurones

II.1. Introduction

Dans ce chapitre nous présentons les étapes nécessaires à la création d'une application pour la reconnaissance automatique des plaques numériques (ANPR). Il existe différentes approches et techniques basées sur des situations différentes, par exemple, les caméras infrarouges, les positions fixes des voitures, les conditions de luminosité, etc.

Nous pouvons procéder à la construction d'une application ANPR pour détecter les plaques d'immatriculation d'automobile dans une photographie prise entre 2 et 3 mètres d'une voiture, dans un état de lumière ambigu et avec un sol non parallèle avec de faibles distorsions de perspective de la plaque de l'automobile.

Le but principal de ce chapitre est de nous présenter la segmentation d'images et l'extraction des traits, la base de reconnaissance de modèles et deux algorithmes de reconnaissance de modèles importants : **Support Vector Machines** et **Artificial Neural Networks**.

Dans ce chapitre, nous aborderons:

- ANPR.
- Détection de plaques.
- Reconnaissance des plaques.

II.2. Introduction à l'ANPR

Reconnaissance automatique de plaques Numériques (ANPR), également connu sous le nom : Reconnaissance Automatique du licence des plaques (ALPR), Identification automatique des véhicules (AVI) ou Reconnaissance des plaques d'immatriculation (CPR), est une méthode de surveillance qui utilise la reconnaissance optique des caractères (OCR) et d'autres méthodes telles que les segmentations et la détection pour lire les plaques d'immatriculation.

Les meilleurs résultats dans un système ANPR peuvent être obtenus avec une caméra infrarouge (IR), car les étapes de segmentation pour la détection et la segmentation OCR sont faciles, propres et minimisent les erreurs.

Cela est dû aux lois de la lumière, la base étant que l'angle d'incidence est égal à l'angle de réflexion; Nous pouvons voir cette réflexion de base quand nous voyons une surface lisse comme un miroir plan. La réflexion sur les surfaces rugueuses telles que le papier conduit à un type de réflexion connu sous le nom de réflexion diffuse ou de dispersion.

La plupart des plaques d'immatriculation ont une caractéristique spéciale appelée rétro-réflexion, la surface de la plaque est faite avec un matériau qui est recouvert des milliers des minuscules hémisphères qui font réfléchir la lumière à la source comme nous pouvons le voir dans la Figure suivante:

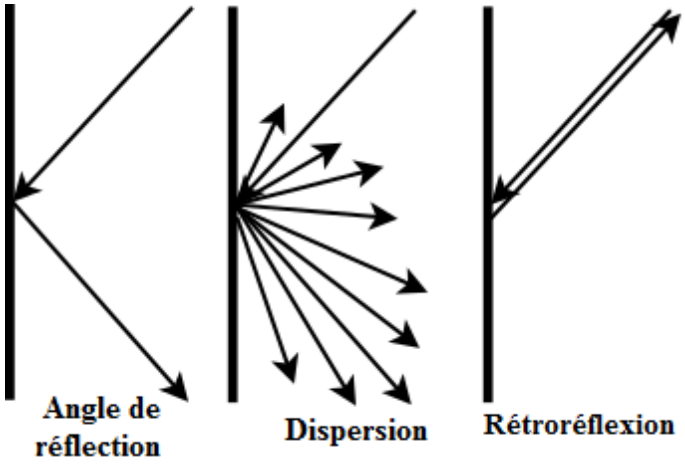

Figure II.1 Schéma de la réflexion, dispersion et la rétroflexion

Si nous utilisons une caméra avec un filtre couplé à un projecteur infrarouge structuré, nous pouvons récupérer uniquement la lumière infrarouge, puis nous avons une image de très haute qualité pour segmenter et ensuite détecter et reconnaître le nombre de plaque qui est indépendante de toute lumière comme le montre la figure suivante:

Figure II.2 Plaque d'immatriculation

Chaque pays a des différentes tailles et spécifications de la plaque d'immatriculation; Il est utile de connaître ces spécifications afin d'obtenir les meilleurs résultats et de réduire les erreurs. Les algorithmes utilisés dans ce chapitre sont destinés à expliquer les bases de l'ANPR et les spécifications pour les plaques d'immatriculation de l'Espagne, mais nous pouvons les étendre à n'importe quel pays ou spécification.

Dans ce chapitre, nous travaillerons avec les plaques d'immatriculation de l'Espagne. En Espagne, il existe trois tailles et formes différentes de plaques d'immatriculation; Nous n'utiliserons que la plus commune (grande) plaque d'immatriculation qui est de 520 x 110 mm.

Deux groupes de caractères sont séparés par un espace de 41 mm, puis une largeur de 14 mm sépare chaque caractère individuel. Le premier groupe de caractères comporte quatre chiffres et le second groupe comporte trois lettres sans les voyelles A, E, I, O, U, ni les lettres Ñ ou Q; Tous les caractères ont des dimensions de 45 x 77 mm. Ces données sont importantes pour la segmentation des caractères car nous pouvons vérifier le caractère et les espaces pour vérifier que nous obtenons un caractère et aucun autre segment d'image. Voici une figure d'une telle plaque d'immatriculation: ANRP

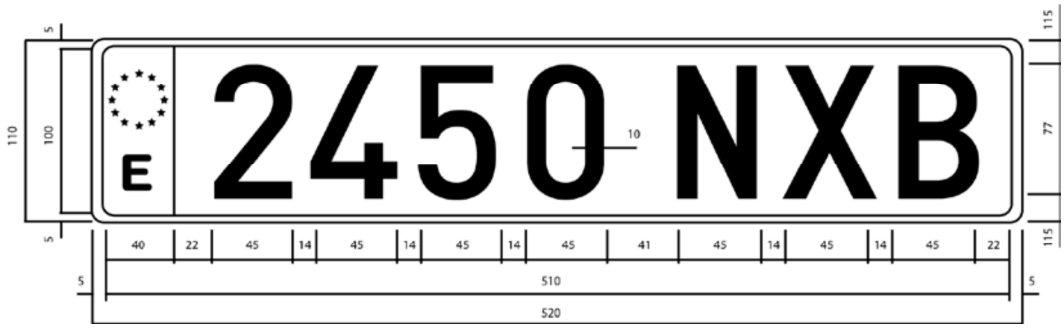

Figure II.3 Dimension de la plaque d'immatriculation

II.3. Algorithme ANPR

Avant d'expliquer le code ANPR, nous devons définir les principales étapes et tâches de l'algorithme ANPR. L'ANPR est divisé en deux étapes principales: la détection des plaques et la reconnaissance des plaques. La détection des plaques a pour but de détecter l'emplacement de la plaque dans l'ensemble du cadre du caméra. Lorsqu'une plaque est détectée dans une image, le segment de plaque est transmis à la deuxième reconnaissance de plaque d'échelon qui utilise un algorithme de reconnaissance optique de caractères pour déterminer les caractères alphanumériques sur la plaque.

Dans la figure suivante, nous pouvons voir les deux étapes principales d'algorithme, la détection des plaques et la reconnaissance des plaques. Après ces étapes, le programme tire sur le cadre du caméra les caractères de la plaque qui ont été détectés. Les algorithmes peuvent retourner des mauvais résultats ou même pas de résultat:

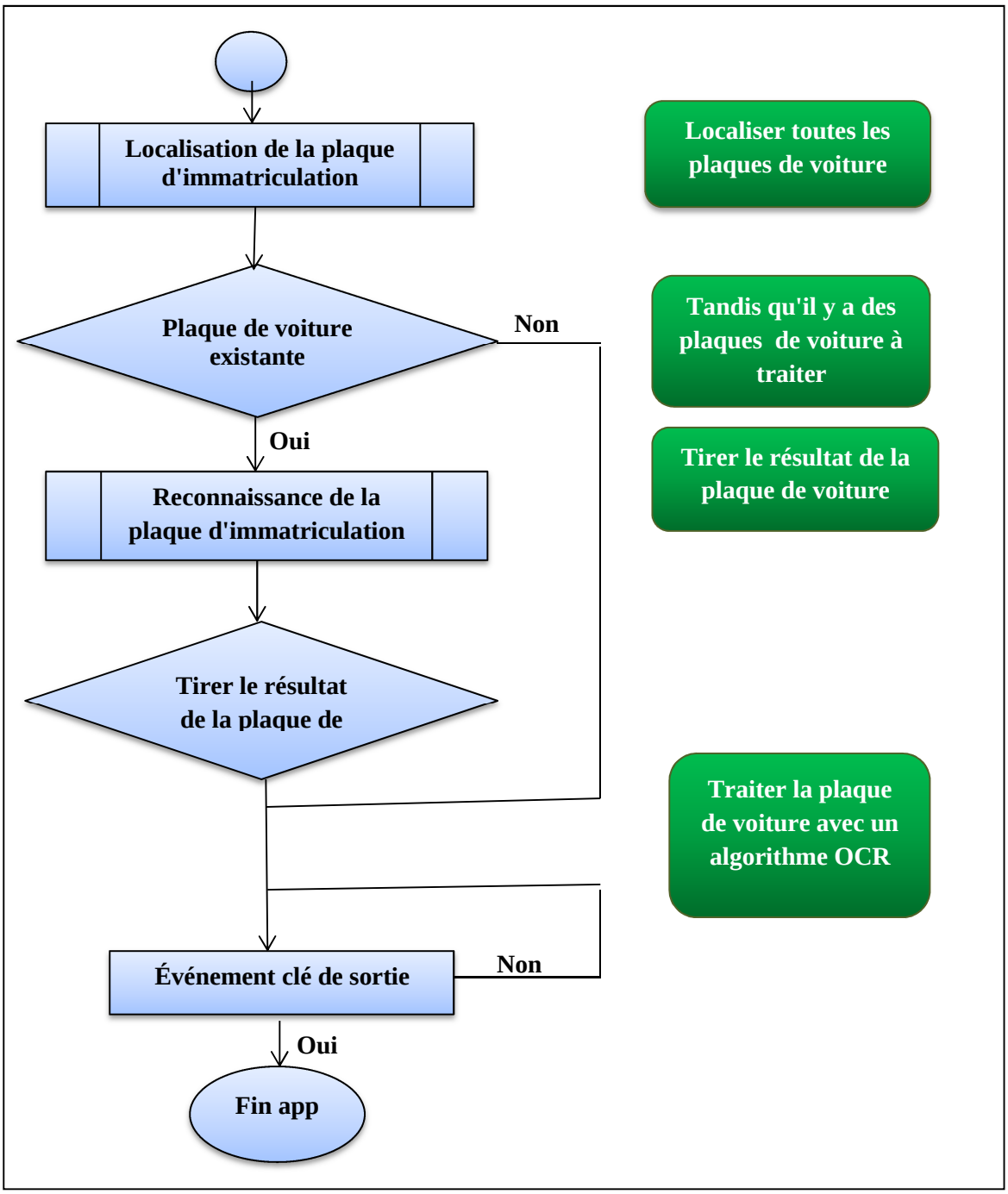

Figure II.4 Etapes principales d'algorithme ANPR

A chaque étape illustrée dans la figure précédente, nous définirons trois étapes supplémentaires qui sont couramment utilisées dans les algorithmes de reconnaissance de formes:

- **Segmentation:** Cette étape détecte et supprime chaque segment d'intérêt dans l'image.
- **Extraction de caractéristiques:** Cette étape extrait de chaque segment un ensemble de caractéristiques.
- **Classification:** Cette étape extrait chaque caractère de l'étape de reconnaissance de plaque, ou classe chaque segment d'image en "plaque" ou "pas de plaque" dans l'étape de détection de plaque.

La figure suivante montre les étapes de reconnaissance des formes dans l'algorithme d'application entier :

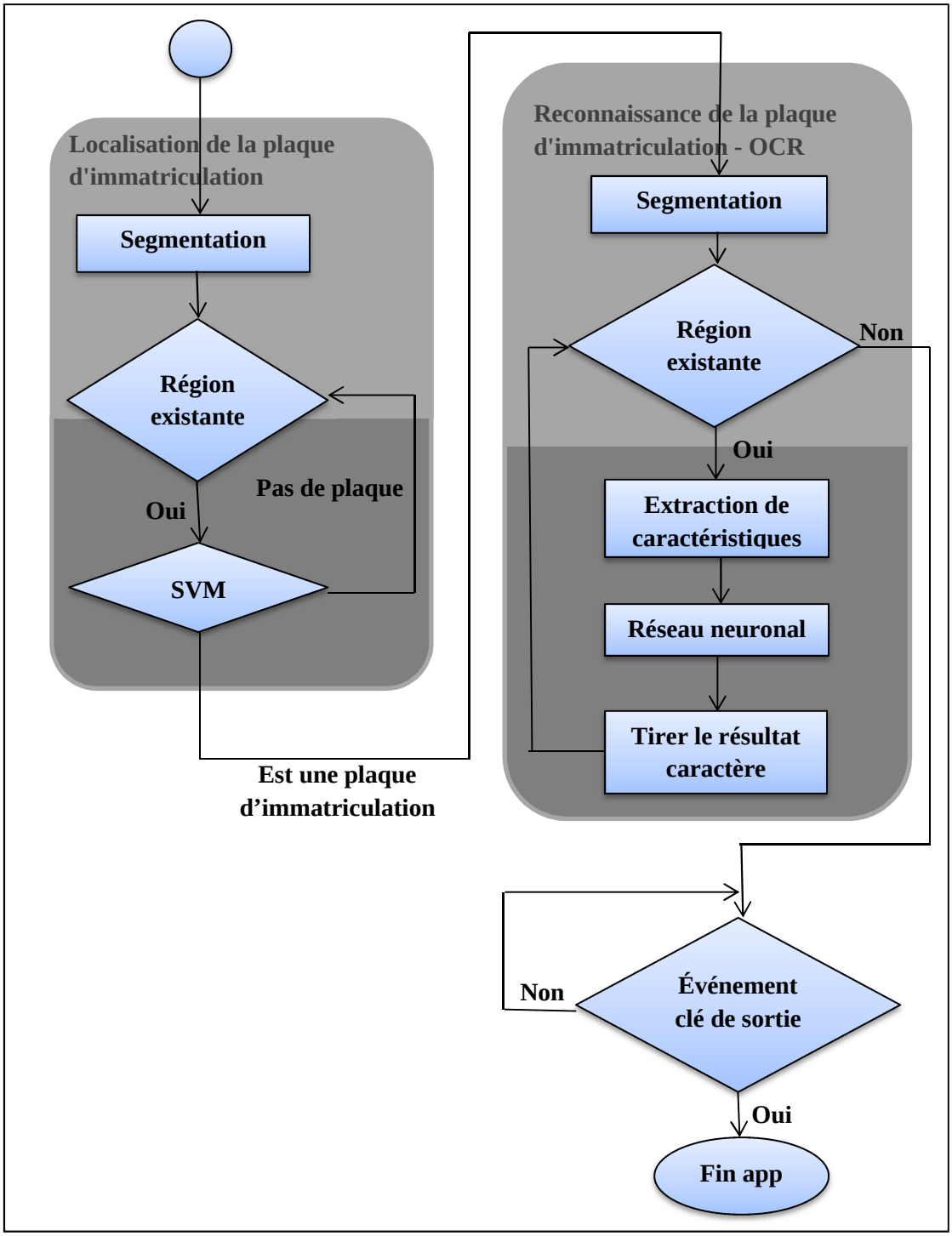

Figure II.5 Etapes de reconnaissance des formes dans l'algorithme ANPR

Mis à part l'application principale, dont le but est de détecter et de reconnaître le numéro d'immatriculation d'une voiture, nous allons brièvement expliquer deux autres tâches qui ne sont généralement pas expliquées:

- Comment former un système de reconnaissance de formes.
- Comment évaluer un tel système.

Cependant, ces tâches peuvent être plus importantes que l'application principale elle-même, car si nous ne formons pas correctement le système de reconnaissance de formes, notre système peut échouer et ne pas fonctionner correctement; Différents modèles ont besoin de différents types de formation et d'évaluation. Nous devons évaluer notre système dans différents environnements, conditions et avec différentes fonctionnalités pour obtenir les meilleurs résultats. Ces deux tâches sont parfois utilisées ensemble car différentes fonctionnalités peuvent produire des résultats différents que nous pouvons voir dans la section d'évaluation.

II.4. Détection de la plaque

Dans cette étape, nous devons détecter toutes les plaques dans le cadre de l'appareil photo actuel. Pour ce faire, nous le divisons en deux étapes principales: la segmentation et la classification des segments. L'étape de caractéristique n'est pas expliquée car nous utilisons le segment d'image comme une fonction vectorielle.

Dans la première étape (segmentation), nous appliquons différents filtres, opérations morphologiques, algorithmes de contour, et des validations pour récupérer les parties de l'image qui pourraient avoir une plaque.

Dans la deuxième étape (classification), nous appliquons un classificateur SVM (**Support Vector Machine**) à chaque segment d'image. Avant de créer notre application principale, nous nous entraînons avec deux classes différentes : plaque et non-plaque. Nous travaillons avec des images parallèles en couleur à vision frontale de 800 pixels de large et prises à 2-4 mètres d'une voiture.

Ces exigences sont importantes pour assurer une segmentation correcte. Nous pouvons effectuer la détection si nous créons un algorithme d'image à plusieurs échelles.

Dans l'image suivante, nous avons montré tous les processus impliqués dans la détection des plaques:

- Filtre de Sobel.
- Fonctionnement de seuil.
- Fermer l'opération morphologique.
- Masque d'une zone remplie.
- Plaques détectées possibles marquées en rouge (images caractéristiques).
- Plaques détectées après le classificateur SVM.

Figure II.6 Etapes de la détection de la plaque d'immatriculation

II.4.1. Segmentation

La segmentation est le processus de division d'une image en plusieurs segments. Ce processus consiste à simplifier l'image, à analyser et à faciliter l'extraction des fonctions.

Une caractéristique importante de la segmentation des plaques est le nombre élevé des bords verticaux dans une plaque d'immatriculation en supposant que l'image a été prise à l'avant et que la plaque n'est pas tournée et qu'elle ne présente pas de distorsion de perspective. Cette fonctionnalité peut être exploitée lors de la première étape de segmentation pour éliminer les régions qui n'ont pas de bords verticaux.

Avant de trouver des arêtes verticaaux, nous devons convertir l'image couleur en image en niveaux de gris (car la couleur ne peut pas nous aider dans cette tâche) et supprimer le bruit possible généré par la caméra ou autre bruit ambiant. Nous allons appliquer un flou gaussien de 5 x 5 et supprimer le bruit. Si nous n'appliquons pas une méthode de suppression du bruit, nous pouvons obtenir beaucoup de bords qui produisent une fausse détection.

// convertir l'image en gris
Mat img_gray;
cvtColor(input, img_gray, CV_BGR2GRAY);
blur(img_gray, img_gray, Size(5,5));

Pour trouver les bords verticaux, nous utiliserons un filtre de **Sobel** et trouver la première dérivée horizontale. La dérivée est une fonction mathématique qui nous permet de trouver les bords verticaux sur une image. La définition d'une fonction **Sobel** dans OpenCV est:
void Sobel(InputArray src, OutputArray dst, int ddepth, int xorder,
int yorder, int ksize=3, double scale=1, double delta=0, int
borderType=BORDER_DEFAULT)

Ici, **ddepth** est la profondeur de l'image de destination, **xorder** est l'ordre de la dérivée par x, **yorder** est l'ordre de la dérivée par y, **ksize** est la taille du noyau de 1, 3, 5 ou 7, **scale** est un facteur facultatif pour les valeurs dérivées calculées, **delta** est une valeur optionnelle ajoutée au résultat, et **borderType** est la méthode d'interpolation des pixels.
Pour notre cas, nous pouvons utiliser un xorder = 1, yorder = 0, et un ksize = 3:
// Trouver des lignes verticaux. Les plaques de voiture ont une densité élevée de lignes verticaux.
Mat img_sobel;
Sobel(img_gray, img_sobel, CV_8U, 1, 0, 3, 1, 0);

Après un filtre de **Sobel**, nous appliquons un filtre de seuil pour obtenir une image binaire avec une valeur de seuil obtenue par la méthode d'**Otsu**. L'algorithme d'**Otsu** a besoin d'une image d'entrée de 8 bits et la méthode d'**Otsu** détermine automatiquement la valeur de seuil optimale:
// image de seuil.
Mat img_threshold;
threshold(img_sobel, img_threshold, 0, 255, CV_THRESH_OTSU+CV_THRESH_
BINARY);

Pour définir la méthode d'**Otsu** dans la fonction de seuil, si nous combinons le paramètre de type avec la valeur **CV_THRESH_OTSU**, le paramètre de valeur de seuil est ignoré. Lorsque la valeur de **CV_THRESH_OTSU** est définie, la fonction de seuil renvoie la valeur de seuil optimale obtenue par l'algorithme d'**Otsu**.

En appliquant une opération morphologique étroite, nous pouvons supprimer des espaces vides entre chaque ligne de bordure verticale, et relier toutes les régions qui ont un grand nombre de bords. Dans cette étape, nous avons les régions possibles qui peuvent contenir des plaques.

D'abord nous définissons notre élément structural à utiliser dans notre opération morphologique. Nous utiliserons la fonction **getStructuringElement** pour définir un élément rectangulaire structurel avec une taille de dimension 17 x 3 dans notre cas; Cela peut être différent dans d'autres tailles d'image.

Mat element = getStructuringElement(MORPH_RECT, Size(17, 3));

Et utiliser cet élément structurel dans une opération morphologique étroite en utilisant la fonction **morphologyEx**.

morphologyEx(img_threshold, img_threshold, CV_MOP_CLOSE, element);

Après l'application de ces fonctions, nous avons des régions dans l'image qui pourraient contenir une plaque; Toutefois, la plupart des régions ne contiendront pas de plaques d'immatriculation. Ces régions peuvent être divisées par une analyse de composants connectés ou en utilisant la fonction **findContours**. Cette dernière fonction récupère les contours d'une image binaire avec différentes méthodes et résultats. Nous avons seulement besoin d'obtenir les contours externes avec n'importe quelle relation hiérarchique et tout résultat d'approximation polygonale.

```
// Trouver les contours des plaques possibles
vector< vector< Point> > contours;
findContours(img_threshold,
contours, // un vecteur de contours
CV_RETR_EXTERNAL, // récupérer les contours externes
CV_CHAIN_APPROX_NONE); // tous les pixels de chaque contour
```

Pour chaque contour détecté, il faut extraire le rectangle de délimitation de la zone minimale. OpenCV fait apparaître la fonction **minAreaRect** pour cette tâche. Cette fonction renvoie une classe de rectangle tournant appelée **RotatedRect**. Ensuite, en utilisant un littérateur vectoriel sur chaque contour, nous pouvons obtenir le rectangle tourné et faire quelques validations préliminaires avant de classer chaque région :

```
// Commencer à itérer sur chaque contour trouvé
vector<vector<Point> >::iterator itc= contours.begin();
vector<RotatedRect> rects;

// Supprime le patch qui n'a pas de limites internes de format et de surface.
while (itc!=contours.end()) {
// Création d'un rectangle d'objet
RotatedRect mr= minAreaRect(Mat(*itc));
if( !verifySizes(mr)){
itc= contours.erase(itc);
}else{
++itc;
rects.push_back(mr);
}
}
```

Nous effectuons des validations de base sur les régions détectées en fonction de sa superficie et de son rapport d'aspect. Nous considérons seulement qu'une région peut être une plaque si le rapport d'aspect est approximativement 520/110 = 4.727272 (largeur de plaque divisée par hauteur de plaque) avec une marge d'erreur de 40 pour cent et une aire basée sur un minimum de 15 pixels et maximum de 125 Pixels pour la hauteur de la plaque. Ces valeurs sont calculées en fonction de la taille de l'image et de la position du caméra :

```
bool DetectRegions::verifySizes(RotatedRect candidate ){

float error=0.4;
// plaque de voiture Espagnole taille: 52x11 aspect 4,7272
const float aspect=4.7272;
// Définit une zone min et max. Tous les autres correctifs sont rejetés
int min= 15*aspect*15; // zone minimale
```

```
int max= 125*aspect*125; // surface maximale
// Obtenir uniquement des correctifs qui correspondent à un ratio de respect.float rmin= aspect-aspect*error;
float rmax= aspect+aspect*error;
int area= candidate.size.height * candidate.size.width;
float r= (float)candidate.size.width / (float)candidate.size.height;
if(r<1)
r= 1/r;
if(( area < min || area > max ) || ( r < rmin || r > rmax )){
return false;
}else{
return true;
}
}
```

Nous pouvons faire plus d'améliorations en utilisant la propriété de fond blanc de la plaque d'immatriculation. Toutes les plaques ont la même couleur de fond et nous pouvons utiliser un algorithme de remplissage pour récupérer le rectangle tourné pour un recadrage précis.

La première étape pour recadrer la plaque d'immatriculation est d'obtenir plusieurs graines près du dernier centre de rectangle tourné. Ensuite, obtenir la taille minimale de la plaque entre la largeur et la hauteur, et l'utiliser pour générer des graines aléatoires près du centre de patch.

Nous voulons sélectionner la région blanche et nous avons besoin de plusieurs graines pour toucher au moins un pixel blanc. Ensuite, pour chaque graine, nous utilisons une fonction **floodFill** pour dessiner une nouvelle image de masque pour stocker la nouvelle région de culture la plus proche:

```
for(int i=0; i< rects.size(); i++){
// Pour un meilleur recadrage de rectangle pour chaque case possible
// Effectuer un algorithme d'inondation parce que la plaque a un fond blanc
// Ensuite, nous pouvons récupérer plus clairement le contour
circle(result, rects[i].center, 3, Scalar(0,255,0), -1);
```

```
// obtenez la taille minimale entre largeur et hauteur
float minSize=(rects[i].size.width < rects[i].size.height)?rects[i].
size.width:rects[i].size.height;
minSize=minSize-minSize*0.5;
// initialiser rand et obtenir 5 points autour du centre pour floodfill algorithme
srand ( time(NULL) );
// Initialiser les paramètres et les variables de floodfill
Mat mask;
mask.create(input.rows + 2, input.cols + 2, CV_8UC1);
mask= Scalar::all(0);
int loDiff = 30;
int upDiff = 30;
int connectivity = 4;
int newMaskVal = 255;
int NumSeeds = 10;
Rect ccomp;
int flags = connectivity + (newMaskVal << 8 ) + CV_FLOODFILL_FIXED_
RANGE + CV_FLOODFILL_MASK_ONLY;
for(int j=0; j<NumSeeds; j++){
Point seed;
seed.x=rects[i].center.x+rand()%(int)minSize-(minSize/2);
seed.y=rects[i].center.y+rand()%(int)minSize-(minSize/2);
circle(result, seed, 1, Scalar(0,255,255), -1);
int area = floodFill(input, mask, seed, Scalar(255,0,0), &ccomp,
Scalar(loDiff, loDiff, loDiff), Scalar(upDiff, upDiff, upDiff), flags);
}
```

La fonction **floodFill** remplit une composante connectée avec de la couleur dans une image de masque à partir d'un point de graine et définit la différence de luminosité / couleur maximale inférieure et supérieure entre le pixel à remplir et les pixels voisins ou pixel de graine:

int floodFill(InputOutputArray image, InputOutputArray mask, Point seed, Scalar newVal, Rect* rect=0, Scalar loDiff=Scalar(), Scalar upDiff=Scalar(), int flags=4)

Le paramètre **newVal** est la nouvelle couleur que nous voulons mettre dans l'image lors du remplissage. Les paramètres **loDiff** et **upDiff** sont les différences maximales de luminosité / couleur supérieures inférieures et maximales entre le pixel à remplir et les pixels voisins ou pixel de graine.

Le paramètre **flag** est une combinaison de:

- **Bits inférieurs:** Ces bits contiennent une valeur de connectivité, 4 (par défaut) ou 8, utilisée dans la fonction. La connectivité détermine quels voisins d'un pixel sont considérés.
- **Bit supérieur:** Il peut s'agir de 0 ou d'une combinaison des valeurs suivantes: **CV_FLOODFILL_FIXED_RANGE** et **CV_FLOODFILL_MASK_ONLY**. **CV_FLOODFILL_FIXED_RANGE** définit la différence entre le pixel actuel et le Pixel d'amorçage. **CV_FLOODFILL_MASK_ONLY** remplit uniquement le masque d'image et ne modifie pas l'image elle-même.

Une fois que nous avons un masque de récolte, nous obtenons un rectangle de zone minimale à partir des points de masque d'image et vérifions à nouveau la taille valide. Pour chaque masque, un pixel blanc obtient la position et utilise la fonction **minAreaRect** pour récupérer la région de culture la plus proche:

```
// Vérifiez le nouveau masque de remplissage de remplissage pour un correctif.
// Obtenir tous les points détectés pour un minimum tourné Rect
vector<Point> pointsInterest;
Mat_<uchar>::iterator itMask= mask.begin<uchar>();
Mat_<uchar>::iterator end= mask.end<uchar>();
for( ; itMask!=end; ++itMask)
if(*itMask==255)
pointsInterest.push_back(itMask.pos());
RotatedRect minRect = minAreaRect(pointsInterest);
if(verifySizes(minRect)){
…
```

Maintenant que le processus de segmentation est terminé et que nous avons des régions valides, nous pouvons recadrer chaque région détectée, supprimer toute rotation possible, recadrer la région de l'image, redimensionner l'image et égaliser la lumière des régions d'image recadrées.

Tout d'abord, nous devons générer la matrice de transformation avec **getRotationMatrix2D** pour supprimer les rotations possibles dans la région détectée. Nous devons faire attention à la hauteur, parce que la classe **RotatedRect** peut être retournée et tournée à 90 degrés, donc nous devons vérifier l'aspect du rectangle, et si elle est inférieure à 1, puis le faire tourner de 90 degrés :

```
// Obtenir une matrice de rotation
float r= (float)minRect.size.width / (float)minRect.size.height;
float angle=minRect.angle;
if(r<1)
angle=90+angle;
Mat rotmat= getRotationMatrix2D(minRect.center, angle,1);
```

Avec la matrice de transformation, nous pouvons maintenant faire tourner l'image d'entrée par un affine (Une transformation affine en géométrie est une transformation qui prend des lignes parallèles à des lignes parallèles) avec la fonction **warpAffine** où nous définissons les images d'entrée et de destination, la matrice de transformation, la taille de sortie (identique à l'entrée dans notre cas) et quelle méthode d'interpolation à utiliser. Nous pouvons définir la méthode de bordure et la valeur de bordure si nécessaire:

```
// Créer et faire pivoter l'image
Mat img_rotated;
warpAffine(input, img_rotated, rotmat, input.size(), CV_INTER_CUBIC);
```

Après avoir tourné l'image, nous recadrons l'image avec **getRectSubPix**, qui récolte et copie une portion d'image de la largeur et de la hauteur données centrées dans un point. Si l'image a été tournée, nous devons changer les tailles de largeur et de hauteur avec la fonction d'échange de C ++.

```
// Recadrer l'image
Size rect_size=minRect.size;
if(r < 1)
```

```
swap(rect_size.width, rect_size.height);
Mat img_crop;
getRectSubPix(img_rotated, rect_size, minRect.center, img_crop);
```

Les images coupées ne sont pas bonnes pour l'utilisation dans la formation et la classification car elles n'ont pas la même taille. De plus, chaque image contient des conditions lumineuses différentes, augmentant ainsi leurs différences relatives. Pour résoudre ce problème, nous redimensionnons toutes les images à la même largeur et la même hauteur et appliquons une légère répartition de l'histogramme:

```
Mat resultResized;
resultResized.create(33,144, CV_8UC3);
resize(img_crop, resultResized, resultResized.size(), 0, 0, INTER_
CUBIC);
// Égaliser l'image recadrée
Mat grayResult;
cvtColor(resultResized, grayResult, CV_BGR2GRAY);
blur(grayResult, grayResult, Size(3,3));
equalizeHist(grayResult, grayResult);
```

Pour chaque région détectée, nous stockons l'image recadrée et sa position dans un vecteur:

```
output.push_back(Plate(grayResult,minRect.boundingRect()));
```

II.4.2. Classification

Après avoir prétraité et segmenté toutes les parties possibles d'une image, nous devons maintenant décider si chaque segment est (ou n'est pas) une plaque d'immatriculation. Pour ce faire, nous allons utiliser un algorithme **Support Vector Machine** (SVM).

SVM est un algorithme de reconnaissance de modèle inclus dans une famille d'algorithmes d'apprentissage supervisé initialement créés pour la classification binaire.

L'apprentissage supervisé est un algorithme d'apprentissage machine qui apprend par l'utilisation des données étiquetées. Nous avons besoin de former l'algorithme avec une quantité de données qui est étiquetée; Chaque ensemble de données doit avoir une classe. La SVM crée un ou plusieurs hyperplans qui sont utilisés pour discriminer chaque classe de données.

Un exemple classique est un ensemble de points 2D qui définit deux classes; Le SVM recherche la ligne optimale qui différencie chaque classe:

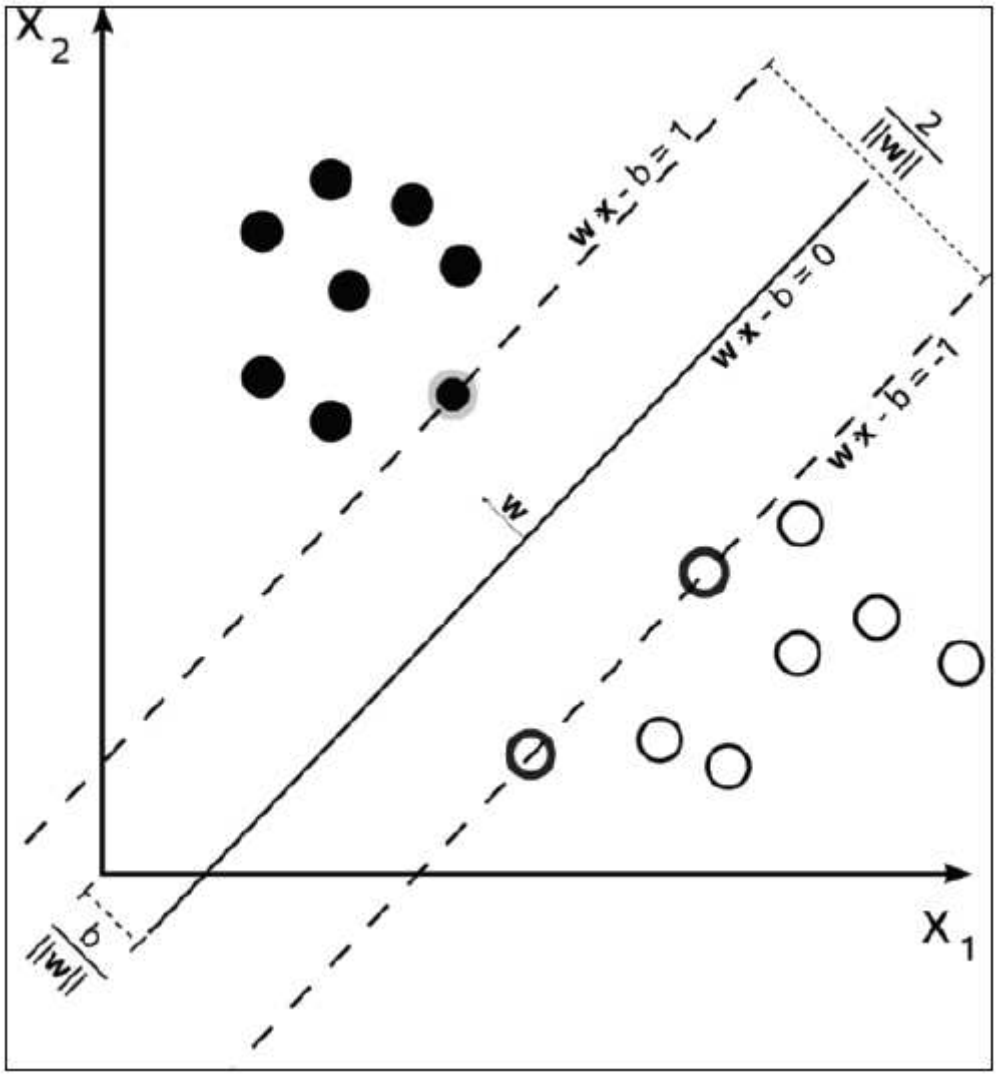

Figure II.7 L'hyperplan optimal (au milieu) et la marge maximale

La marge est la distance entre l'hyperplan et les échantillons les plus proches. Ces derniers sont appelés **vecteurs supports**.

La première tâche avant toute classification est de former notre classificateur; Ce travail se fait avant de commencer l'application principale et on l'appelle une formation hors ligne. Ce n'est pas un travail facile car il nécessite une quantité suffisante de données pour former le système, mais un plus grand ensemble de données n'implique pas toujours les meilleurs résultats. Dans notre cas, nous n'avons pas assez de données car il n'existe pas de base de données publique sur les plaques d'immatriculation. Pour cette raison, nous devons prendre centaines photos des voitures, puis prétraiter et segmenter toutes les photos.

Nous avons formé notre système avec 75 images de plaque d'immatriculation et 35 images sans plaques de 144 x 33 pixels. Nous pouvons voir un échantillon de ces données dans l'image suivante. Ce n'est pas un grand ensemble de données, mais il est suffisant pour obtenir des résultats décents pour nos besoins. Dans une application réelle, nous aurions besoin de former avec plus de données:

Figure II.8 Images plaque et non plaque

Pour comprendre facilement comment l'apprentissage machine fonctionne, nous procédons à l'utilisation des caractéristiques des pixels des image de l'algorithme classificateur (gardez à l'esprit, il existe de meilleures méthodes et fonctionnalités pour former un SVM, tels que l'analyse des composants principaux, la transformée de Fourier, l'analyse de texture, et ainsi de suite).

Nous devons créer les images pour former notre système à l'aide de la classe **DetectRegions** et définir la variable **savingRegions** sur **true** afin de sauvegarder les images. Nous pouvons utiliser le script **segmentAllFiles.sh** pour répéter le processus sur tous les fichiers image sous un dossier.

Pour faciliter cette tâche, nous stockons toutes les données de formation d'image qui sont traitées et préparées, dans un fichier XML à utiliser directement avec la fonction SVM. L'application **trainSVM.cpp** crée ce fichier en utilisant les dossiers et le nombre des fichiers images.

Les données de formation pour un algorithme OpenCV d'apprentissage machine sont stockées dans une matrice N x M avec N échantillons et M fonctions. Chaque ensemble de données est enregistré en tant que ligne dans la matrice de formation. Les classes sont stockées dans une autre matrice avec une taille N x 1, chaque classe étant identifiée par un numéro de flotteur.

OpenCV a un moyen simple de gérer un fichier de données au format XML ou JSON avec la classe **FileStorage**. Cette classe permet de stocker et de lire des variables et des structures OpenCV ou nos variables personnalisées. Avec cette fonction, nous pouvons lire la matrice de formation-données et les cours de formation et l'enregistrer dans **SVM_TrainingData** et **SVM_Classes**:

```
FileStorage fs;
fs.open("SVM.xml", FileStorage::READ);
Mat SVM_TrainingData;
Mat SVM_Classes;
fs["TrainingData"] >> SVM_TrainingData;
fs["classes"] >> SVM_Classes;
```

Maintenant, nous devons définir les paramètres SVM qui définissent les paramètres de base à utiliser dans un algorithme SVM; Nous utiliserons la structure **CvSVMParams** pour la définir. Il s'agit d'une cartographie faite aux données d'apprentissage pour améliorer sa ressemblance avec un ensemble de données linéairement séparable. Cette cartographie consiste à augmenter la dimensionnalité des données et se fait efficacement à l'aide d'une fonction noyau. Nous choisissons ici les types **CvSVM :: LINEAR**, ce qui signifie qu'aucune application n'est effectuée:

```
// Définir les paramètres SVM
CvSVMParams SVM_params;
SVM_params.kernel_type = CvSVM::LINEAR;
```

Nous créons et formons ensuite notre classificateur. OpenCV définit la classe **CvSVM** pour l'algorithme Support Vector Machine et nous l'initialisons avec les données de formation, les classes et les données de paramètres:

```
CvSVM svmClassifier(SVM_TrainingData, SVM_Classes, Mat(), Mat(), SVM_
params);
```

Notre classificateur est prêt à prédire une image recadrée possible en utilisant la fonction de prédiction de notre classe SVM; Cette fonction renvoie l'identificateur de classe **i**. Dans notre cas, nous étiquetons une classe de plaques de 1 et aucune classe de plaques avec 0. Ensuite, pour chaque région détectée qui peut être une plaque, nous utilisons SVM pour classer comme une plaque ou pas de plaque, et d'enregistrer que les réponses correctes. Le code suivant est une partie de l'application principale, appelée traitement en ligne:

```
vector<Plate> plates;
for(int i=0; i< possible_regions.size(); i++)
{
Mat img=possible_regions[i].plateImg;
Mat p= img.reshape(1, 1);//convert img to 1 row m features
p.convertTo(p, CV_32FC1);
int response = (int)svmClassifier.predict( p );
if(response==1)
plates.push_back(possible_regions[i]);
}
```

II.5. Reconnaissance des plaques

La deuxième étape de la reconnaissance des plaques d'immatriculation vise à récupérer les caractères de la plaque d'immatriculation avec la reconnaissance optique des caractères(OCR). Pour chaque plaque détectée, on procède à la segmentation de la plaque pour chaque caractère, et on utilise un réseau neuronal artificiel (ANN : **Artificial Neural Network**), algorithme d'apprentissage machine pour reconnaître le caractère. Également dans cette section, nous allons apprendre à évaluer un algorithme de classification.

II.5.1. Segmentation OCR

En premier lieu, nous obtenons un segment d'image en plaque comme entrée pour la fonction OCR de segmentation avec un histogramme égalisé, nous avons ensuite besoin d'appliquer un filtre de seuil et d'utiliser cette image de seuil comme entrée d'un algorithme **Find contours**; Nous pouvons voir ce processus dans la figure suivante:

Figure II.9 Etapes de la reconnaissance des caractères

Ce processus de segmentation est codé comme suit:

```
Mat img_threshold;
threshold(input, img_threshold, 60, 255, CV_THRESH_BINARY_INV);
if(DEBUG)
imshow("Threshold plate", img_threshold);
Mat img_contours;
img_threshold.copyTo(img_contours);
```

```
// Rechercher les contours des caractères possibles
vector< vector< Point> > contours;
findContours(img_contours,
contours, // un vecteur de contours
CV_RETR_EXTERNAL, // récupérer les contours externes
CV_CHAIN_APPROX_NONE); // tous les pixels de chaque contour
```

Nous utilisons le paramètre **CV_THRESH_BINARY_INV** pour inverser la sortie de seuil en transformant les valeurs d'entrée noires en blanc et les valeurs d'entrée blanches en noir. Ceci est nécessaire pour obtenir les contours de chaque caractère, car l'algorithme des contours recherche des pixels blancs.

Pour chaque contour détecté, nous pouvons effectuer une vérification de taille et supprimer toutes les régions où la taille est plus petite ou l'aspect n'est pas correct. Dans notre cas, les caractères ont un aspect de 45/77, et nous pouvons accepter une erreur de 35 pour cent d'aspect pour les caractères tournés ou déformés. Si une superficie est supérieure à 80%, nous considérons que cette région est un bloc, et pas un caractère. Pour le comptage de la zone, nous pouvons utiliser la fonction **countNonZero** qui compte le nombre de pixels avec une valeur supérieure à 0:

```
bool OCR::verifySizes(Mat r)
{
// Poids 45x77
float aspect=45.0f/77.0f;
float charAspect= (float)r.cols/(float)r.rows;
float error=0.35;
float minHeight=15;
float maxHeight=28;
// Nous avons un rapport d'aspect différent pour le nombre 1, et il peut être ~ 0,2
float minAspect=0.2;
float maxAspect=aspect+aspect*error;
// zone de pixels
Float area = countNonZero (r);
```

```
// zone bb
float bbArea=r.cols*r.rows;
//% de pixels dans la zone
float percPixels=area/bbArea;
if(percPixels < 0.8 && charAspect > minAspect && charAspect <
maxAspect && r.rows >= minHeight && r.rows < maxHeight)
return true;
else
return false;
}
```

Si un caractère segmenté est vérifié, nous devons le prétraiter pour définir la même taille et la même position pour tous les caractères et l'enregistrer dans un vecteur avec la classe **CharSegment** auxiliaire. Cette classe enregistre l'image de caractère segmentée et la position que nous avons besoin pour ordonner les caractères car l'algorithme **Find Contour** ne retourne pas les contours dans l'ordre requis.

II.5.2. Extraction de caractéristiques

L'étape suivante pour chaque caractère segmenté est d'extraire les fonctions de formation et de classement de l'algorithme du réseau neuronal artificiel.

Contrairement à l'étape d'extraction de caractéristique de détection de plaque qui est utilisée dans SVM, nous n'utilisons pas tous les pixels d'image; Nous allons appliquer des caractéristiques plus communes utilisées dans la reconnaissance optique de caractères contenant des histogrammes d'accumulation horizontale et verticale et un échantillon d'image à basse résolution. Nous pouvons voir cette fonction plus graphiquement dans l'image suivante, où chaque image a une résolution de 5 x 5 et les accumulations d'histogramme:

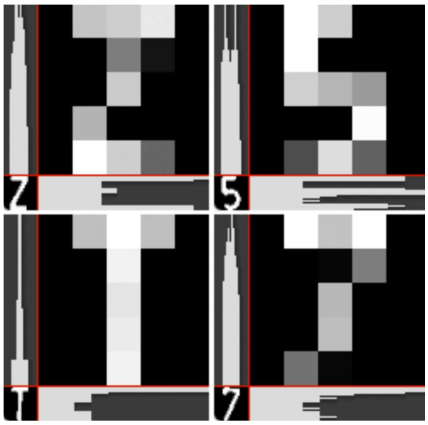

Figure II.10 Différentes caractères

Pour chaque caractère, nous comptons le nombre de pixels d'une ligne ou d'une colonne avec une valeur non nulle en utilisant la fonction **countNonZero** et la stockons dans une nouvelle matrice de données appelée **mhist**. Nous le normalisons en recherchant la valeur maximale dans la matrice de données en utilisant la fonction **minMaxLoc** et divisons tous les éléments de **mhist** par la valeur maximum avec la fonction **convertTo**. Nous créons la fonction **ProjectedHistogram** pour créer les histogrammes d'accumulation qui ont comme entrée une image binaire et le type d'histogramme dont nous avons besoin, horizontal ou vertical:

```
Mat OCR::ProjectedHistogram(Mat img, int t)
{
int sz=(t)?img.rows:img.cols;
Mat mhist=Mat::zeros(1,sz,CV_32F);
for(int j=0; j<sz; j++){
Mat data=(t)?img.row(j):img.col(j);
mhist.at<float>(j)=countNonZero(data);
}
// Normaliser l'histogramme
double min, max;
minMaxLoc(mhist, &min, &max);
if(max>0)
mhist.convertTo(mhist,-1 , 1.0f/max, 0);
return mhist;
}
```

D'autres caractéristiques utilisent une image d'échantillon à basse résolution. Au lieu d'utiliser toute l'image de caractère, nous créons un caractère de basse résolution, par exemple 5 x 5. Nous formons le système avec 5 x 5, 10 x 10, 15 x 15 et 20 x 20 caractères, puis évaluez celui qui revient le meilleur résultat afin que nous puissions l'utiliser dans notre système. Une fois que nous avons toutes les fonctionnalités, nous créons une matrice de M colonnes par une ligne où les colonnes sont les caractéristiques:

```
Mat OCR::features(Mat in, int sizeData)
{
// Caractéristiques de l'histogramme
Mat vhist=ProjectedHistogram(in,VERTICAL);
Mat hhist=ProjectedHistogram(in,HORIZONTAL);
// Faible caractéristique des données
Mat lowData;
resize(in, lowData, Size(sizeData, sizeData) );
int numCols=vhist.cols + hhist.cols + lowData.cols *
lowData.cols;
Mat out=Mat::zeros(1,numCols,CV_32F);
// Attribuer des valeurs à la fonction
int j=0;
for(int i=0; i<vhist.cols; i++)
{
out.at<float>(j)=vhist.at<float>(i);
j++;
}
for(int i=0; i<hhist.cols; i++)
{
out.at<float>(j)=hhist.at<float>(i);
j++;
}
for(int x=0; x<lowData.cols; x++)
{
for(int y=0; y<lowData.rows; y++)
{
out.at<float>(j)=(float)lowData.at<unsigned char>(x,y);
```

```
j++;
}
}
return out;
}
```

II.5.3. Classification OCR

Dans l'étape de classification, nous utilisons un réseau neuronal artificiel, algorithme d'apprentissage machine. Plus précisément, un **Multi-Layer Perceptron** (MLP), qui est l'algorithme ANN le plus couramment utilisé.

MLP consiste en un réseau de neurones avec une couche d'entrée, une couche de sortie et une ou plusieurs couches masquées. Chaque couche possède un ou plusieurs neurones connectés à la couche précédente et suivante.

L'exemple suivant représente un perceptron à 3 couches (c'est un classificateur binaire qui mappe une entrée de vecteur à valeur réelle à une seule sortie de valeur binaire) avec trois entrées, deux sorties et la couche cachée comprenant cinq neurones:

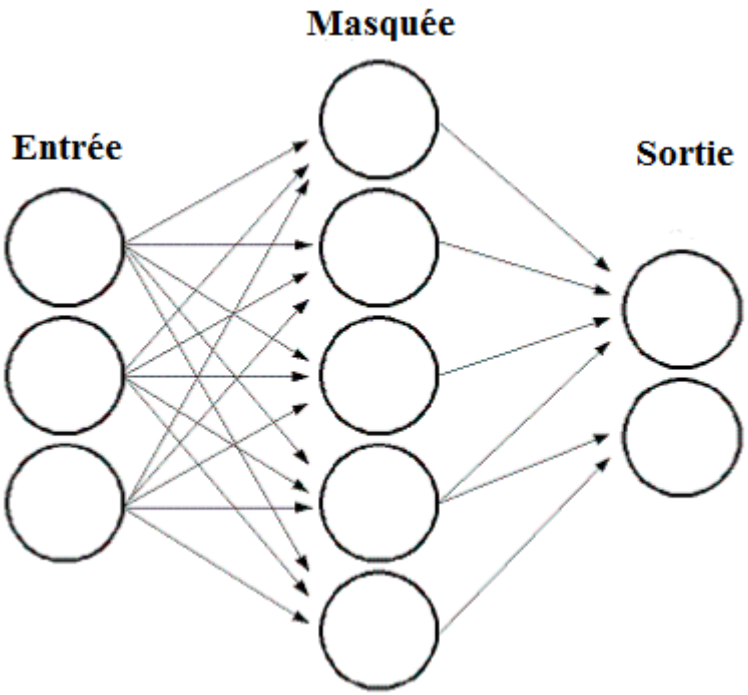

Figure II.11 Vue simplifiée d'un réseau artificiel de neurones

Tous les neurones d'un MLP sont semblables et chacun a plusieurs entrées (les neurones liés précédemment) et plusieurs liaisons de sortie avec la même valeur (les neurones liés suivants). Chaque neurone calcule la valeur de sortie comme une somme des entrées pondérées plus un terme de polarisation et est transformée par une fonction d'activation sélectionnée:

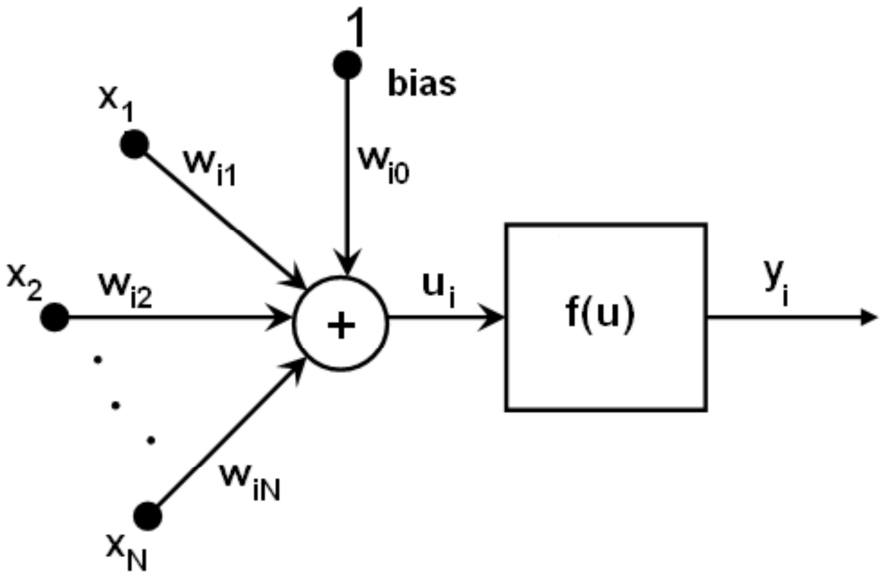

Figure II.12 Structure d'un neurone artificiel

Il existe trois fonctions d'activation largement utilisées: Identité, Sigmoïde et Gaussienne; La fonction d'activation par défaut est la fonction sigmoïde. Elle a une valeur alpha et bêta définie à 1:

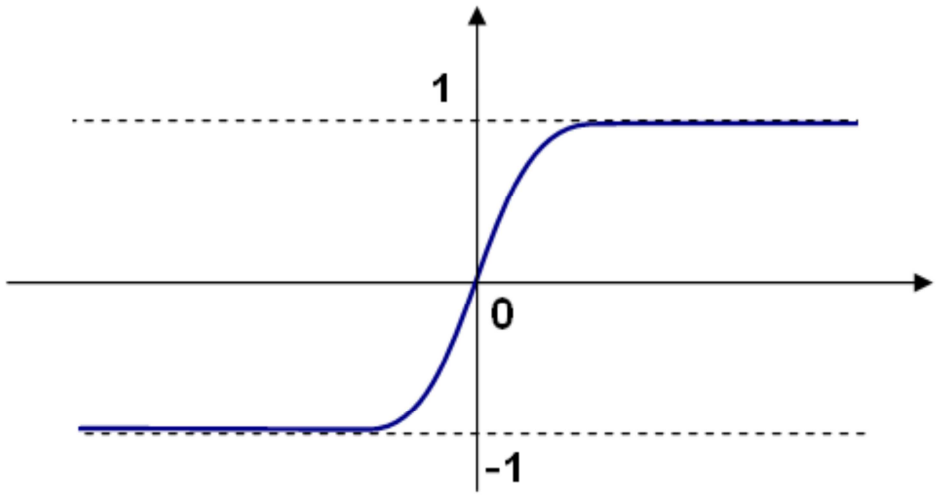

Figure II.13 La fonction sigmoïde

Un réseau formé par ANN a un vecteur d'entrée avec des caractéristiques. Il passe les valeurs à la couche masquée et calcule les résultats avec les poids et la fonction d'activation. Il passe les sorties plus en aval jusqu'à ce qu'il obtienne la couche de sortie qui a le nombre de classes de neurones.

Le poids de chaque couche, des synapses et du neurone est calculé et appris en formant l'algorithme ANN. Pour former notre classificateur, nous créons deux matrices de données comme nous l'avons fait dans la formation SVM, mais les étiquettes de formation sont un peu différentes. Au lieu d'une matrice N x 1 où N représente les lignes de données d'apprentissage et 1 est la colonne, nous utilisons l'identificateur de numéro d'étiquette. Nous devons créer une matrice N x M où N est la donnée de formation / échantillons et M est les classes (10 chiffres + 20 lettres dans notre cas), et placer 1 dans une position (i, j) si la ligne de données i est classé avec la classe j.

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Figure II.14 Matrices de données N*M

Nous créons une fonction **OCR :: train** pour créer toutes les matrices nécessaires et former notre système, avec la matrice des données de formation, la matrice des classes et le nombre de neurones cachés dans les couches masquées. Les données de formation sont chargées à partir d'un fichier XML comme nous l'avons fait pour la formation SVM.

Nous devons définir le nombre de neurones dans chaque couche pour initialiser la classe ANN. Pour notre échantillon, nous n'utilisons qu'une couche cachée, puis nous définissons une matrice de 1 rangée et 3 colonnes. La position de la première colonne est le nombre de fonctionnalités, la position de la deuxième colonne est le nombre de neurones cachés dans la couche masquée et la position de la troisième colonne est le nombre de classes.

OpenCV définit une classe **CvANN_MLP** pour ANN. Avec la fonction **create**, nous pouvons initier la classe en définissant le nombre de couches et de neurones, la fonction d'activation et les paramètres alpha et bêta:

```
void OCR::train(Mat TrainData, Mat classes, int nlayers)
{
Mat layerSizes(1,3,CV_32SC1);
layerSizes.at<int>(0)= TrainData.cols;
layerSizes.at<int>(1)= nlayers;
layerSizes.at<int>(2)= numCharacters;
ann.create(layerSizes, CvANN_MLP::SIGMOID_SYM, 1, 1); //ann is
global class variable
// Préparer les classes de train
// Créer un mat avec n données formées par m classes
Mat trainClasses;
trainClasses.create( TrainData.rows, numCharacters, CV_32FC1 );
for( int i = 0; i < trainClasses.rows; i++ )
{
for( int k = 0; k < trainClasses.cols; k++ )
{
// Si la classe de données i est identique à une classe k
if( k == classes.at<int>(i) )
trainClasses.at<float>(i,k) = 1;
else
```

```
trainClasses.at<float>(i,k) = 0;
}
}
Mat weights( 1, TrainData.rows, CV_32FC1, Scalar::all(1) );
// Apprendre le classificateur
ann.train( TrainData, trainClasses, weights );
trained=true;
}
```

Après la formation, nous pouvons classer toute fonction de plaquette segmentée en utilisant la fonction **OCR :: classify**:

```
int OCR::classify(Mat f)
{
int result=-1;
Mat output(1, numCharacters, CV_32FC1);
ann.predict(f, output);
Point maxLoc;
double maxVal;
minMaxLoc(output, 0, &maxVal, 0, &maxLoc);
// Nous devons savoir où dans output est le max val, le x (cols) est la classe.
return maxLoc.x;
}
```

La classe **CvANN_MLP** utilise la fonction **predict** pour classer un vecteur de caractéristique dans une classe. Contrairement à la fonction de classification SVM, la fonction de prédiction de l'ANN renvoie une ligne dont la taille est égale au nombre de classes avec la probabilité d'appartenir à la caractéristique d'entrée de chaque classe.

Pour obtenir le meilleur résultat, nous pouvons utiliser la fonction **minMaxLoc** pour obtenir la réponse maximale et minimale et la position dans la matrice. La classe de notre caractère est spécifiée par la position x d'une valeur plus élevée:

Figure II.15 Résultat

Pour terminer chaque plaque détectée, nous commandons ses caractères et renvoyons une chaîne à l'aide de la fonction **str ()** de la classe Plate et nous pouvons la dessiner sur l'image originale:

```
string licensePlate=plate.str();
rectangle(input_image, plate.position, Scalar(0,0,200));
putText(input_image, licensePlate, Point(plate.position.x, plate.
position.y), CV_FONT_HERSHEY_SIMPLEX, 1, Scalar(0,0,200),2);
```

II.6. Conclusion

Dans ce chapitre, nous avons appris comment fonctionne le programme de reconnaissance automatique des plaques d'immatriculation et ses deux étapes importantes: la localisation des plaques et la reconnaissance des plaques.

Dans la première étape, nous avons appris à segmenter une image à la recherche des segments où l'on peut avoir une plaque, et comment utiliser une simple heuristique et l'algorithme **Support Vector Machine** pour faire une classification binaire pour les segments avec des plaques et pas de plaques.

Dans la deuxième étape, nous avons appris à segmenter avec l'algorithme **Find Contours**, extraire le vecteur de caractéristiques de chaque personnage et utiliser un réseau neuronal artificiel pour classer chaque entité dans une classe de caractères.

Chapitre III

Réalisation et mise en œuvre

III.1. Introduction

Après avoir compléter l'étude conceptuelle dans le chapitre précédent, nous allons aborder la partie implémentation dans ce qui suit. Nous commençons par présenter l'environnement de de développement et l'installation, et ensuite, la réalisation.

Dans ce chapitre nous aborderons les sujets suivants :

- L'installation du pack de développement Android Tegra
- Comprendre comment fonctionne le kit de développement natif (NDK)
- La création de projet Android avec OpenCV
- La réalisation de l'application.

III.2. Installation du pack de développement Android Tegra

TADP a été diffusé par NVIDIA pour que la préparation pour l'environnement de développement Android soit transparente.

NVIDIA a publié TADP version 3.0r4 pour prendre en charge Android SDK (23.0.2), NDK (r10c) et OpenCV pour Tegra 2.4.8.2, qui est un SDK OpenCV4Android régulier étendu avec des optimisations spécifiques à Tegra.

III.3. Téléchargement et installation de TADP

Pour obtenir TADP, il faut visiter https://developer.nvidia.com/tegra-android-development-pack et suivre les étapes pour devenir un développeur enregistré ; C'est une adhésion gratuite.

Une fois que l'adhésion est activée, il faut connecter et téléchargez la version correspondant au système d'exploitation. NVIDIA prend en charge les systèmes d'exploitation suivants:

- Windows 64 bits.
- Mac OS X.
- Ubuntu Linux (32/64-bit).

Dans mon cas, j'ai Windows 7 64 bits sur ma machine. Pour l'installation d'Ubuntu, TADP aura besoin d'avoir des privilèges root, alors il faut assurer de le faire. Une fois que le téléchargement du programme d'installation de TADP est terminé, lancer-le et effectuer les étapes suivantes :

1. Suivre les instructions à l'écran après avoir lu et accepté le contrat de licence.

2. Choisir le type d'installation. Sélectionner une installation personnalisée et cliquer sur le bouton Suivant :

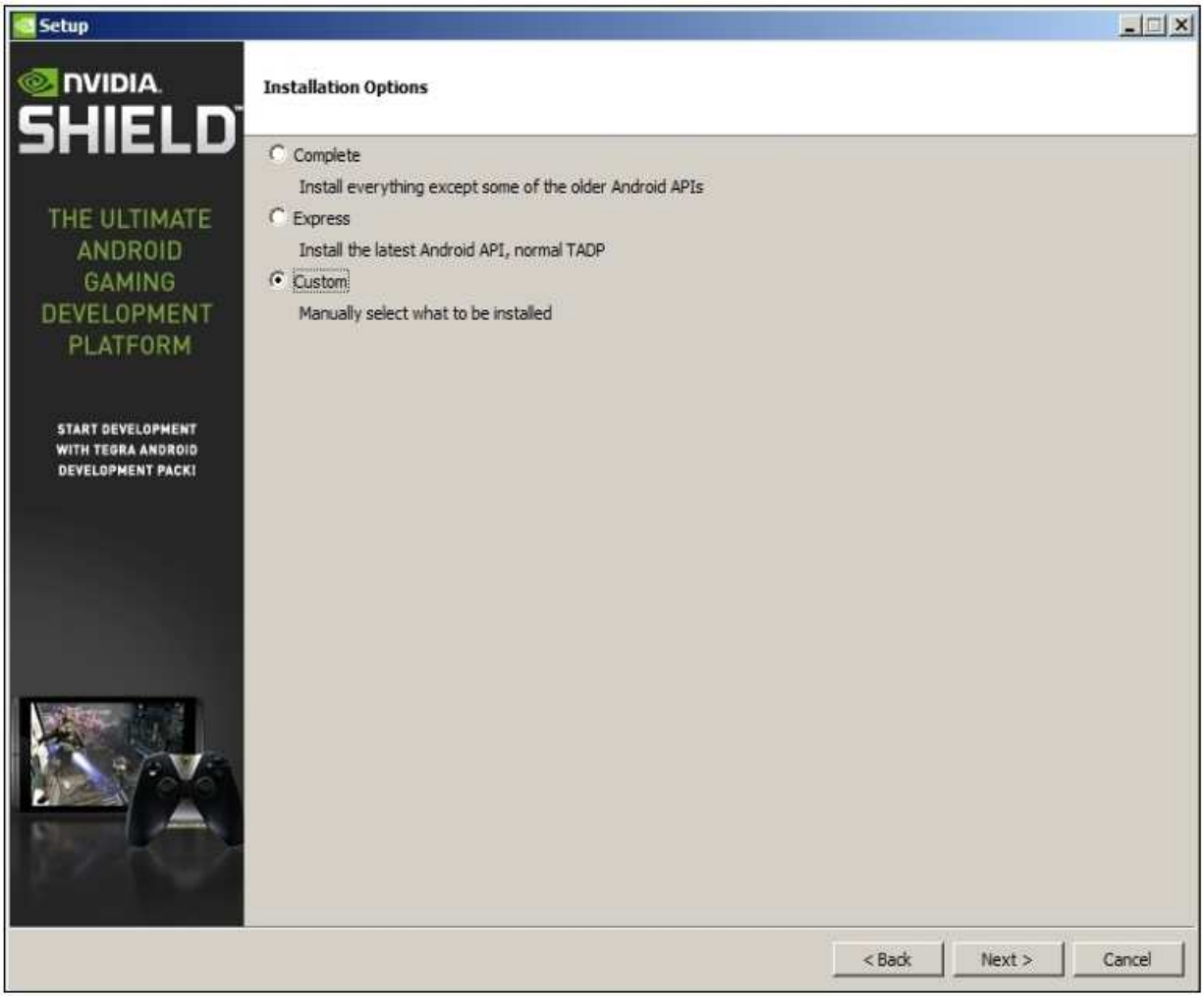

Figure III.1 Installation TADP

3. Sélectionner les composants à installer comme décrit et cliquer sur le bouton Suivant:

Figure III.2 Composants à installer

4. Nommer l'installation et télécharger le répertoire.

5. Cliquez sur le bouton Suivant. Le programme d'installation va commencer à télécharger tous les composants sélectionnés ; Cela peut prendre un certain temps en fonction de la connexion Internet.

6. Une fois le téléchargement terminé, cliquer sur Suivant pour commencer l'installation des composants sélectionnés.

7. Sélectionner l'action de post-installation souhaitée et cliquer sur le bouton Terminer.

III.4. Configuration TADP

Après que TADP télécharge et installe tout, il faut faire une configuration après l'installation afin d'assurer que tout fonctionnera correctement.

III.4.1. Installation des images du système émulateur

Pour chaque plate-forme Android SDK installée, il faut installer une image système au cas de vouloir exécuter un émulateur avec cette plate-forme SDK comme cible.

Pour ce faire, il faut juste suivre ces simples étapes :

1. Accéder au répertoire d'installation sélectionné lors de l'installation de TADP.

2. Ouvrir le dossier SDK; Dans ce cas, c'est Android-sdk-windows.

3. Exécuter le Gestionnaire SDK.

4. Pour chaque Android X.X installé, sélectionner une image système pour l'émulateur, comme ARM EABI V7a System Image :

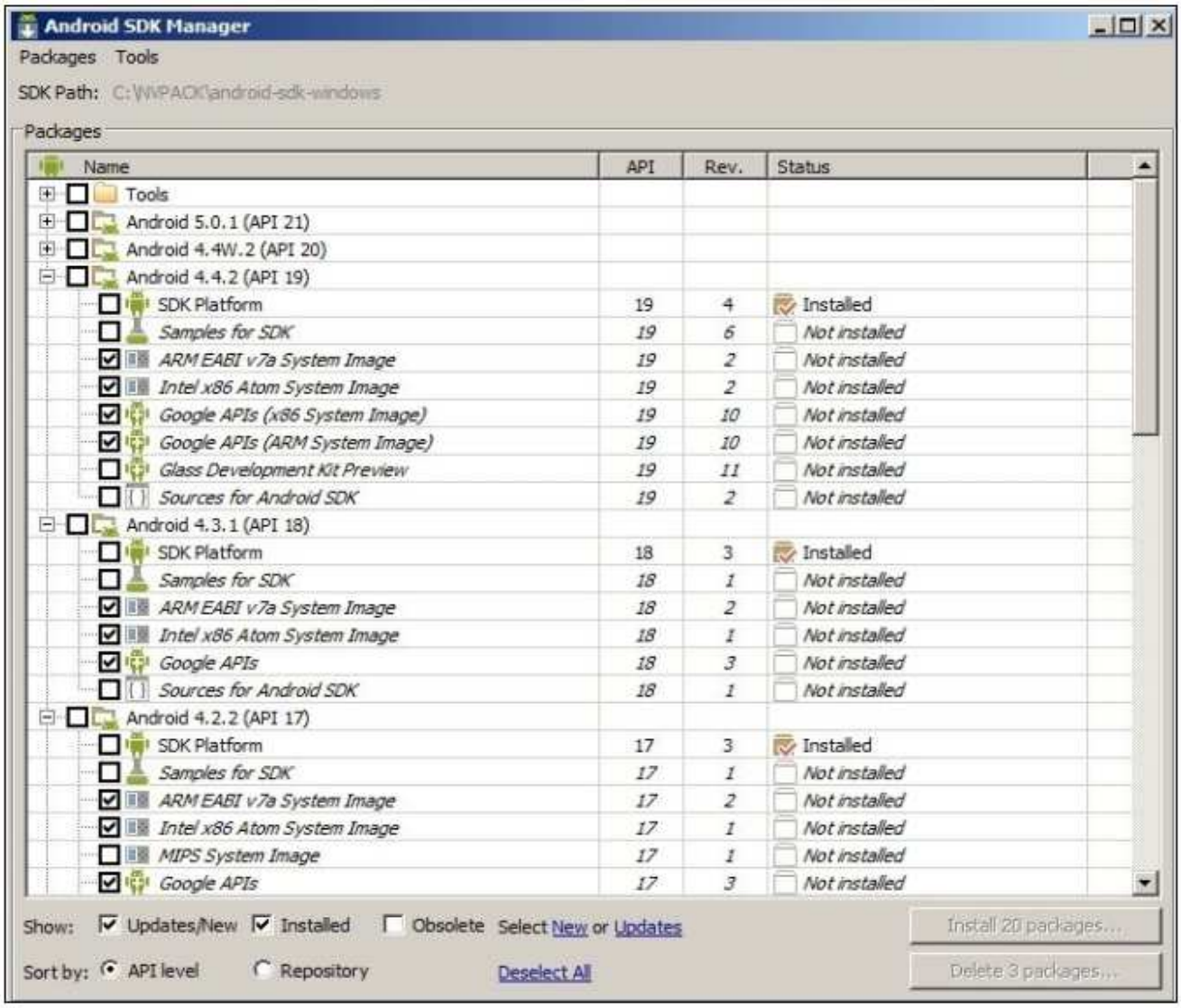

Figure III.3 Android SDK Manager

5. Cliquer sur Installer les packages.

6. Lire et accepter le contrat de licence pour les composants sélectionnés.

7. Cliquer sur Installer.

III.4.2. Configuration d'Eclipse pour fonctionner avec NDK

Maintenant, il faut configurer Eclipse pour qu'il fonctionne avec NDK afin de pouvoir créer des applications natives directement à partir d'Eclipse :

1. Lancer Eclipse à partir du répertoire d'installation spécifié plus tôt.

2. Ouvrir la fenêtre | Préférences.

3. Dans le volet sur le côté gauche, ouvrir AndroidTree.

4. Sélectionner le nœud d'arbre appelé NDK.

5. Dans le volet droit, cliquer sur Parcourir et sélectionner le répertoire NDK ; il se trouve sous le répertoire d'installation.

6. Cliquer sur OK.

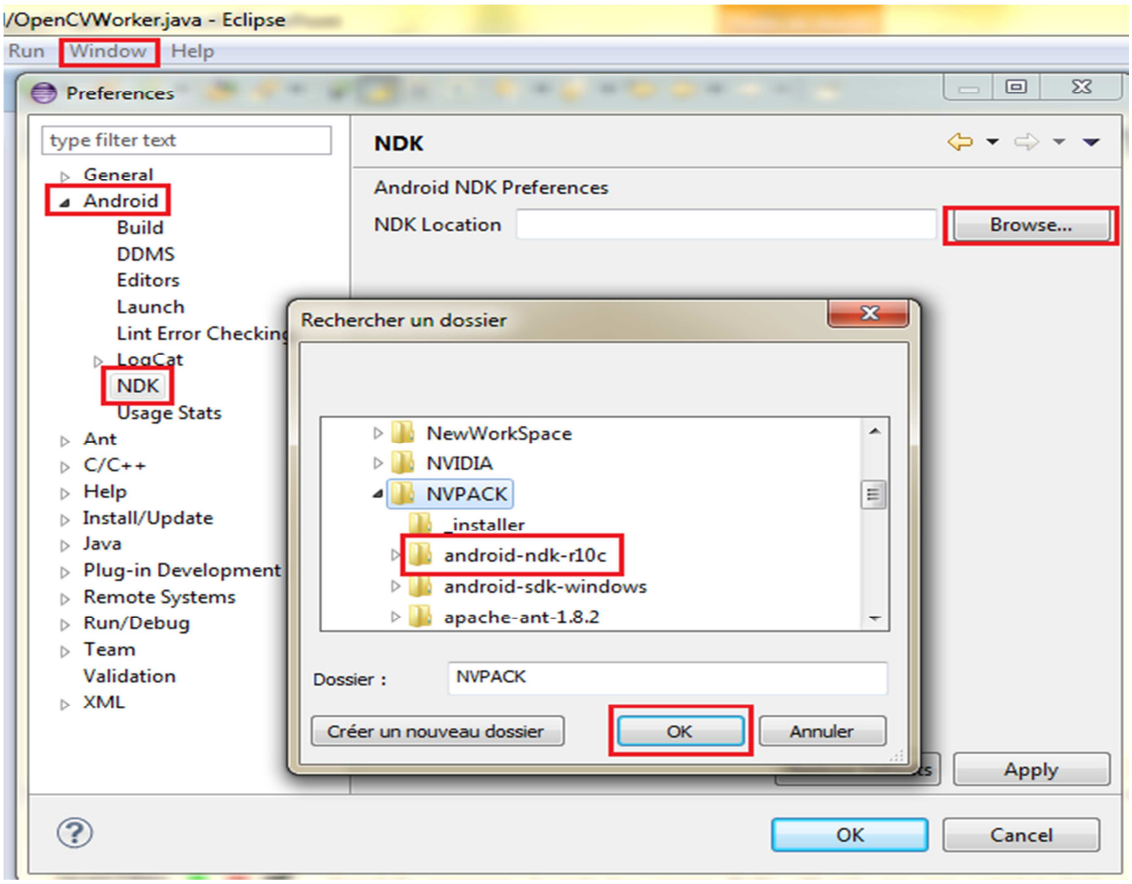

Figure III.4 Configurer Eclipse avec NDK

III.4.2.1. Vérification NDK

Comme les bibliothèques OpenCV sont écrites en C / C ++, la première étape pour vérifier que l'environnement fonctionne est d'assurer la possibilité de pouvoir exécuter des applications Android qui utilisent le code natif :

1. Lancer Eclipse.

2. Dans le répertoire d'installation NDK, dans mon cas, C: \ NVPACK \ android-ndk-r10c \ - importer le projet d'exemple hello-jni à partir du dossier samples.

3. Cliquer avec le bouton droit de la souris sur le projet HelloJni.

4. Dans le menu contextuel, choisir Outils Android | Ajouter un support natif.

5. Assurer que le nom de la bibliothèque est défini sur hello-jni ; Il faut l'appeler par défaut.

6. Démarrez l'émulateur.

7. Cliquer avec le bouton droit de la souris sur le projet hello-jni dans l'explorateur de projet. Dans le menu contextuel, choisir Exécuter en tant que | Application Android.

Dans la sortie de console, il devrait y avoir une liste de fichiers .so ; Ce sont les bibliothèques partagées natives que NDK ait construit à l'aide de l'interface binaire d'application (ABI), qui définit exactement comment le code machine doit être considéré.

Android NDK prend en charge différentes architectures. Par défaut, .so sera construite pour ARM EABI en plus de MIPS et x86 si elles sont spécifiées dans le fichier application.mk. Si tout se déroule en douceur, l'émulateur devrait avoir une application fonctionnant comme suit :

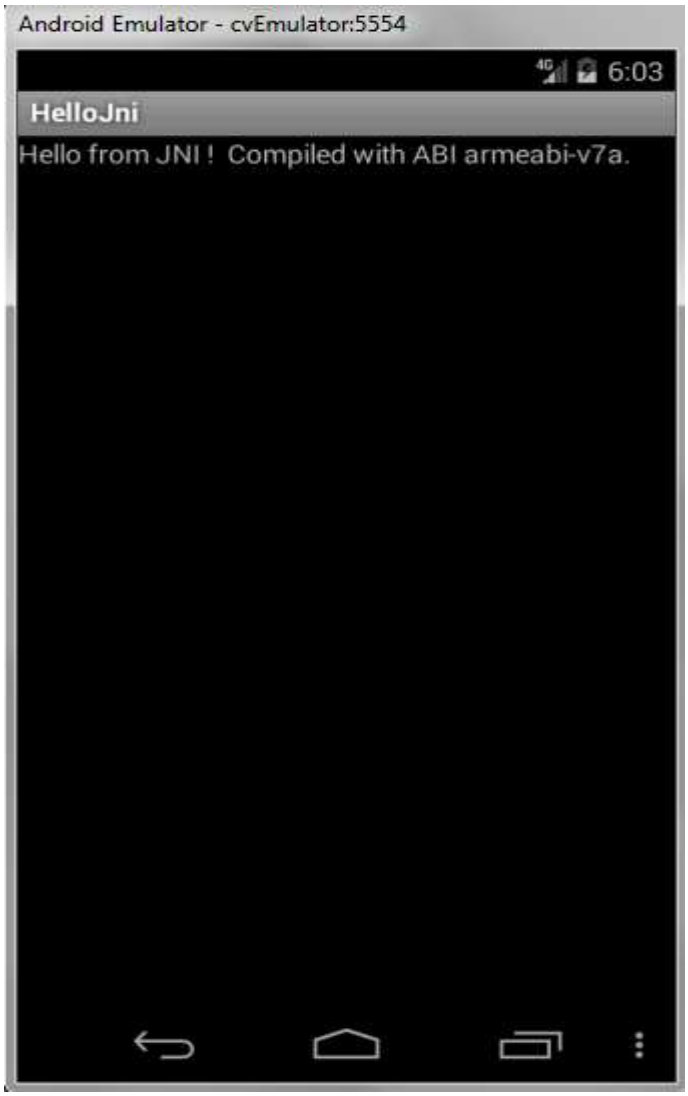

Figure III.5 Exécution HelloJni

Cette application est très simple et un bon point de contrôle pour vérifier la possibilité de pouvoir invoquer le code natif de l'application Android.

Fondamentalement, ce qui est affiché sur l'écran de l'émulateur est une chaîne renvoyée du code natif et affichée par le cadre Android dans une vue de texte.

III.4.2.2. Comprendre comment NDK fonctionne

La programmation avec Android NDK consiste essentiellement à écrire un code à la fois dans Java et dans des langages natifs tels que C, C ++ et assembleur.

Le code Java s'exécute sur Dalvik Virtual Machine (VM), alors que le code natif est compilé sur des fichiers binaires s'exécutant directement sur le système d'exploitation. Java Native Interface (JNI) agit comme le pont qui réunit les deux mondes. Cette relation entre le code Java, Dalvik VM, le code natif et le système Android peut être illustrée par le schéma suivant :

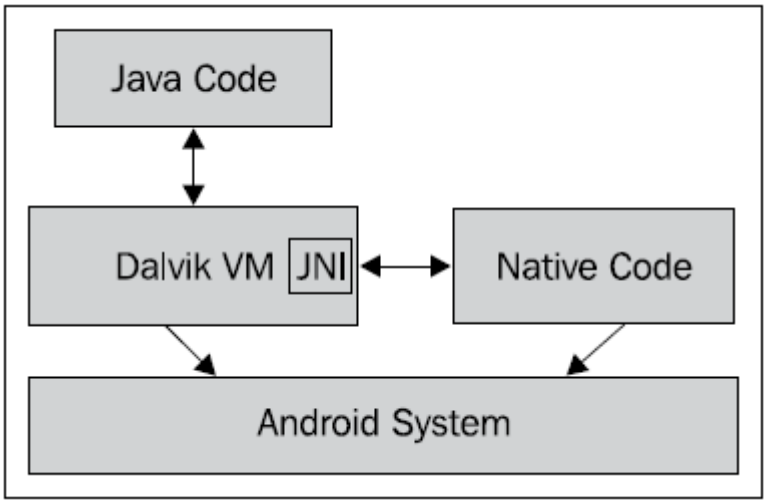

Figure III.6 JNI

La flèche dans le diagramme indique quelle partie initie l'interaction. Les deux Dalvik VM et Native Code fonctionnent sur le système Android. Ils ont besoin du système pour fournir l'environnement d'exécution. JNI fait partie de Dalvik VM, qui permet à Native Code d'accéder aux champs et d'appeler des méthodes au Code Java.

III.4.2.2.1. JNI

JNI est l'acronyme de Java Native Interface. C'est une technologie qui permet également à Java Code d'invoquer des méthodes natives mises en œuvre dans Native Code. Par conséquent, JNI facilite la communication bidirectionnelle entre Native Code et Java Code.

L'inconvénient majeur de cette technologie est d'annuler la portabilité du code Java. En contre partie, cette technologie peut être très utile dans plusieurs cas :

- pour des raisons de performance.
- utilisation de composants éprouvés déjà existants.

La mise en œuvre de JNI nécessite plusieurs étapes :

- la déclaration et l'utilisation de la ou les méthodes natives dans la classe Java.
- la compilation de la classe Java.

- la génération du fichier d'en-tête avec l'outil javah.
- l'écriture du code natif en utilisant entre autres les fichiers d'en-tête fournis par le JDK et celui généré précédemment.
- La compilation du code natif sous la forme d'une bibliothèque.

Le format de la bibliothèque est donc dépendante du système d'exploitation pour lequel elle est développée : **.dll** pour les systèmes de type Windows, **.so** pour les système de type Unix, ...

III.4.2.2.2. Passage de paramètres et renvoi d'une valeur

Plusieurs types sont prédéfinis par JNI pour toutes les primitives et les objets fréquemment utilisés :

Primitive Java	Type native
boolean	jboolean
byte	jbyte
char	jchar
double	jdouble
int	jint
float	jfloat
long	jlong
short	short
void	void

Tableau III.1 Primitives type Java et type native

Objet Java	Objet C
java.lang.object	jobject
java.lang.String	jstring
java.lang.Class	jclass
java.lang.Throwable	jthrowable
type de base pour les tableaux	jarray
int[]	jintArray
long[]	jlongArray
float[]	jfloatArray
double[]	jdoubleArray
Object[]	jobjectArray
boolean[]	jbooleanArray
byte[]	jbyteArray
char[]	jcharArray
short[]	jshortArray

Tableau III.2 Objet Java et objet C

III.4.3. Création de projet avec OpenCV

III.4.3.1. OpenCV

La bibliothèque de logiciels Open Source Computer Vision (OpenCV) possède plus de 2500 algorithmes optimisés. La bibliothèque comprend un ensemble complet d'algorithmes de vision par ordinateur et d'algorithmes d'apprentissage. Elle existe depuis une décennie et publiée sous la licence Berkeley Software Distribution (BSD), ce qui facilite l'utilisation et la modification du code par les utilisateurs.

OpenCV pour Android prend en charge l'accès à ses fonctions grâce à son API native et à son API Java wrappers. Dans le cas d'une API native, il faut définir la bibliothèque native en utilisant NDK Android et inclure les bibliothèques OpenCV. Ensuite, il faut appeler la bibliothèque native à partir du code Java en utilisant Java Native Interface (JNI).

L'autre option consiste à utiliser les wrappers Java OpenCV directement dans le code Java en utilisant les importations Java habituelles. Ce qui se passera, c'est que les wrappers de Java canaliseront vos appels vers les bibliothèques OpenCV natives à l'aide de JNI.

III.4.3.2. Création du projet

Pour créer notre projet dans Eclipse, nous avons suivi les étapes suivantes :

1. Lancer Eclipse et créer un nouvel espace de travail.

2. Créer un nouveau projet Android et nommez votre application ANPR.

3. Définir la version minimale du SDK. Pour créer avec OpenCV4Android SDK, la version SDK minimale est 11 ; Cependant, il est fortement recommandé d'utiliser l'API 15 ou supérieure. Dans mon cas, j'ai utilisé l'API 15.

4. Sélectionner Target SDK. Dans mon cas, je l'ai configuré à l'API 23. Cliquer sur Suivant.

5. Autorisez Eclipse à créer une nouvelle activité vide et à le nommer MainActivity avec un layout nommée activity_main.

6. Importer le projet de bibliothèque OpenCV dans l'espace de travail. Naviguer vers Fichier | Importer | Code Android existant dans l'espace de travail.

7. Sélectionner le répertoire racine d'OpenCV4Android SDK. Désélectionner tous les exemples de projets et sélectionner uniquement la bibliothèque OpenCV et cliquer sur Terminer.

Figure III.7 Importer OpenCV

8. Référencer la bibliothèque OpenCV à partir du projet Android. Cliquer sur Projet | Propriétés. Sélectionner le nœud d'arbre Android dans le volet latéral gauche et dans le volet latéral droit, cliquer sur Ajouter dans la section Bibliothèque, puis sur OK.

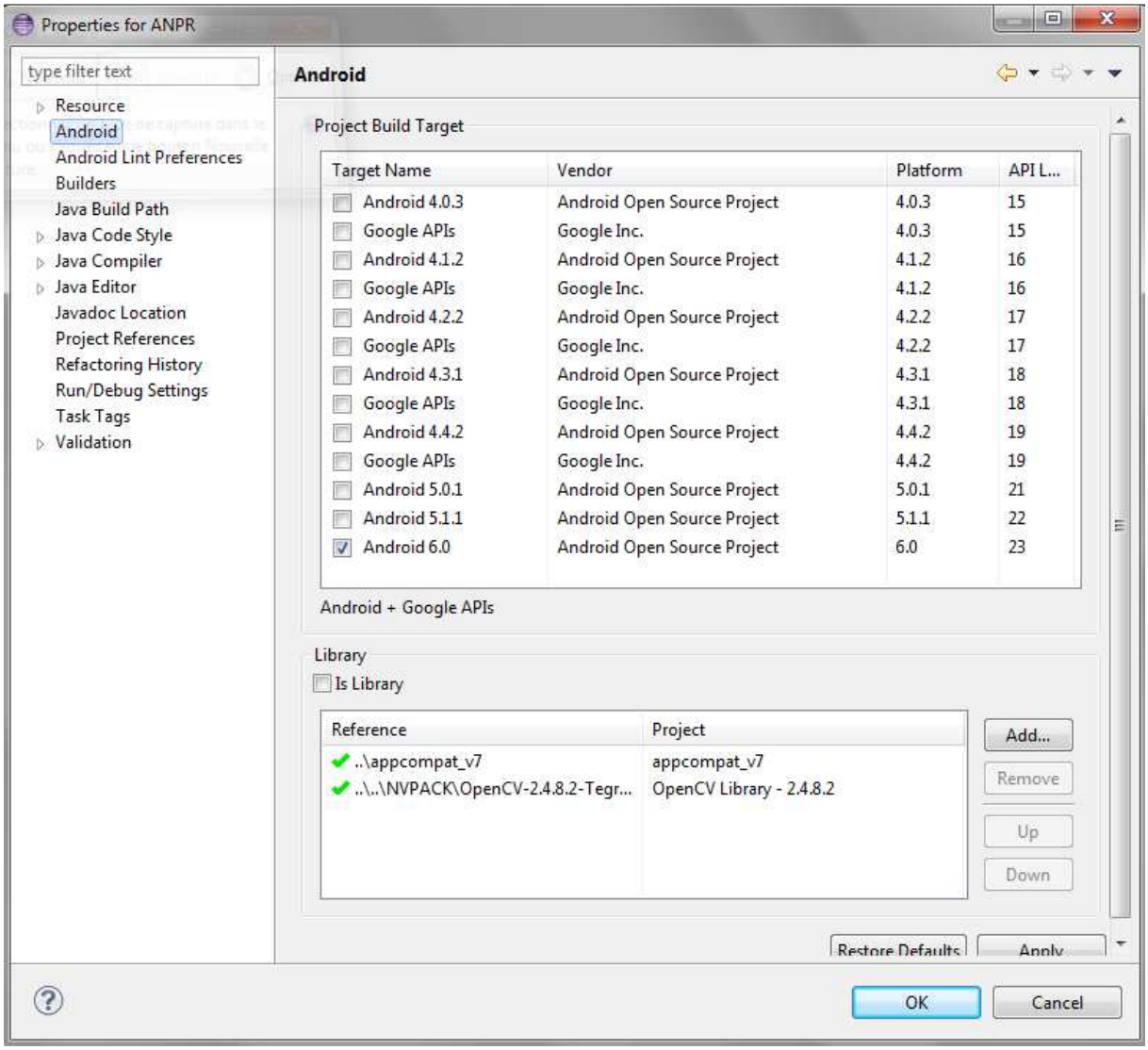

Figure III.8 Propriétés de projet

9. Ajouter le support natif au projet. Cliquer sur Projet | Android Tools | Add Native Support. Nommer la bibliothèque et cliquer sur Finish.

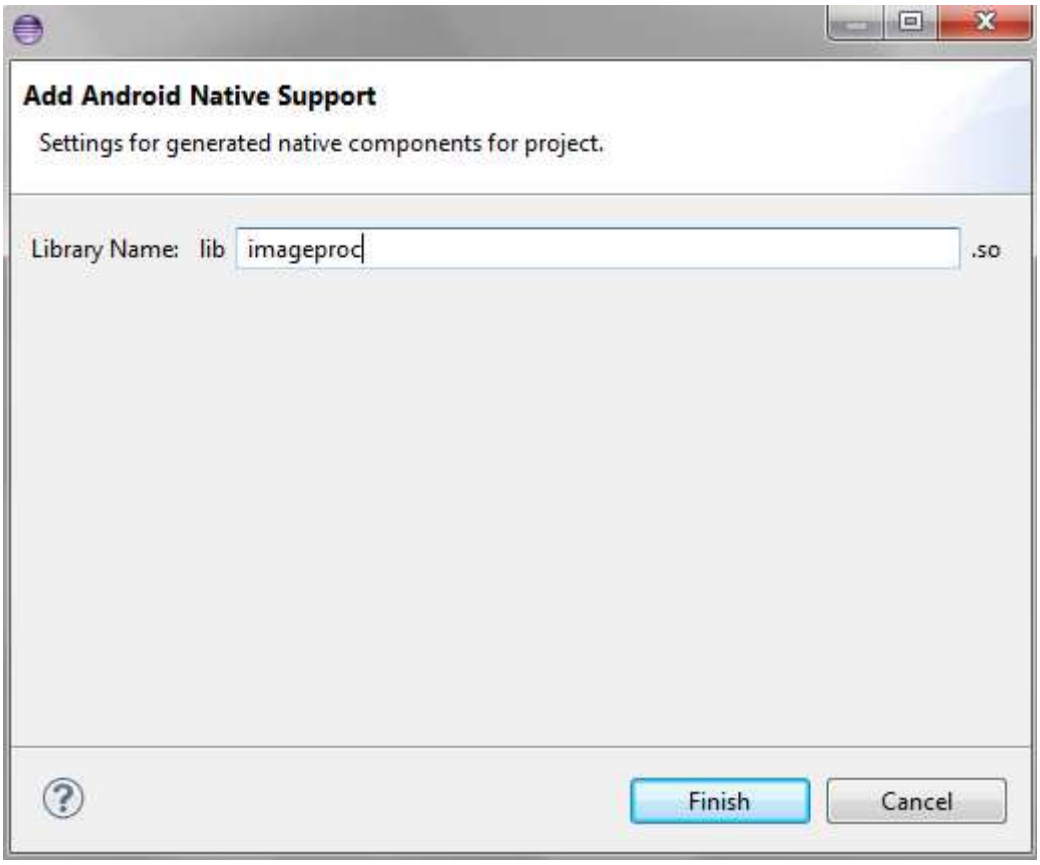

Figure III.9 Ajouter Android Native Support

III.4.3.3. Un aperçu de NDK

Le dossier racine du projet créé comporte les sous-répertoires suivants :

jni/

libs/

res/

src/

AndroidManifest.xml

project.properties

Ici, les dossiers liés à NDK sont les suivants:

- Le dossier jni contiendra la partie native de votre application. En d'autres termes, il s'agit du code source C / C ++ avec les scripts de construction NDK tels que Android.mk et Application.mk, qui sont nécessaires pour créer les bibliothèques natives.
- Le dossier libs contiendra les bibliothèques natives après une compilation réussie.

III.5. Réalisation d'ANPR

Le code de notre application est divisé en deux parties, une partie écrite en langage natif C++ et l'autre en Java.

III.5.1. Code native :

Cette partie écrite en C++ contient toutes les étapes nécessaires à la création de l'application ANPR qui sont détaillées dans le chapitre précédent. Ces classes doivent être mises dans le dossier jni. Puis il faut compiler le code natif sous la forme d'une bibliothèque partir de l'invite de commande en utilisant la commande ndk-build après avoir spécifie l'emplacement du projet :

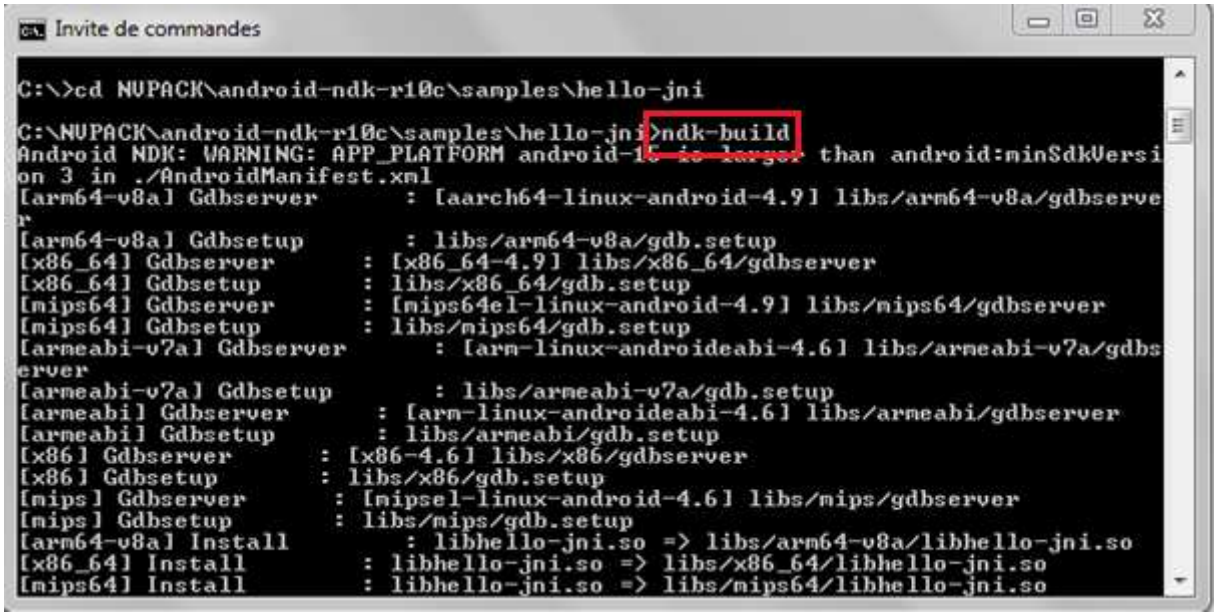

Figure III.10 Compilation du code natif

III.5.2. Code Java :

Dans cette partie, il y a deux classes plus l'interface de l'application :

- La classe « CarPlateDetection » : contient la méthode native imageproc. Imageproc.cpp rassemble toutes les méthodes natives qui sont appelées par le code Java.
- La classe « MainActivity » c'est la classe principale. Elle permet d'effectuer toutes les opérations du code natif sur une image qui est déjà dans le dossier « Drawable », puis afficher le résultat dans la zone du texte " m_text" lors d'un clique sur le bouton "bouton".
- L'interface de l'application : C'est l'interface principale, elle s'affiche lors du lancement de l'application. Elle continent la photo qui contient la plaque et un bouton. En cliquant sur le bouton, le résultat s'affichera dans un instant sur l'image.

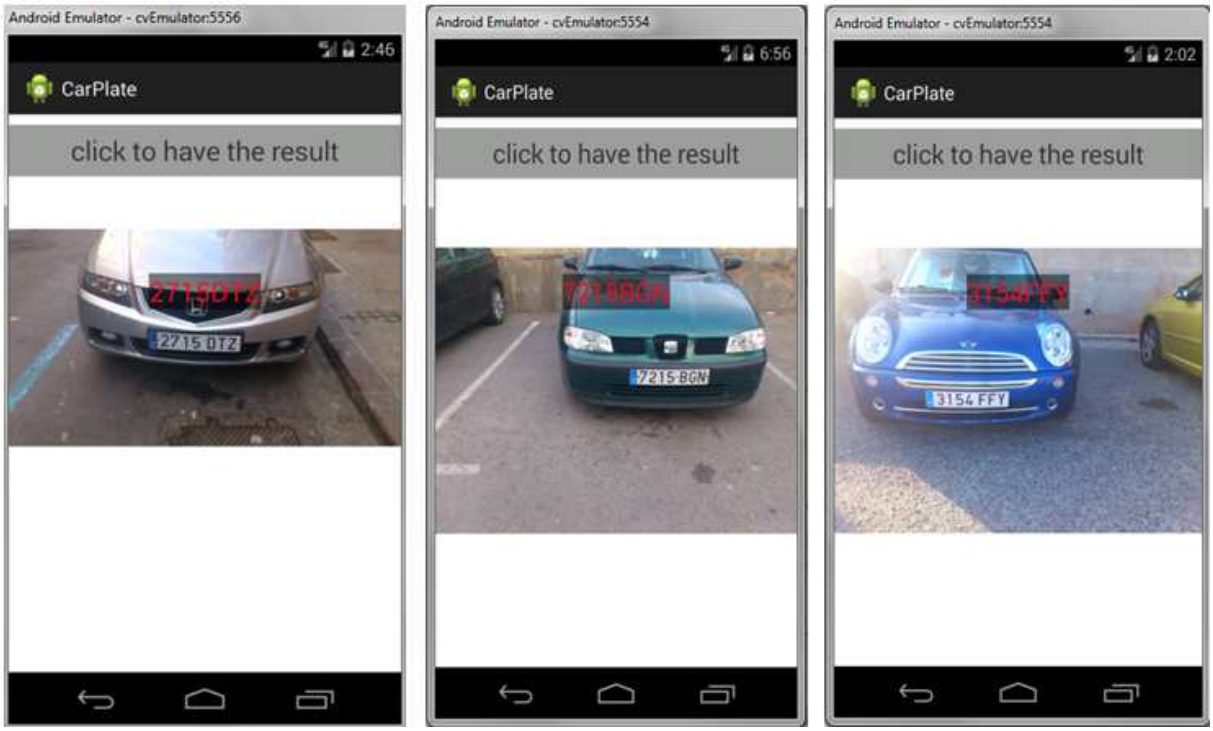

Figure III.11 Résultat

III.6. Conclusion

Dans ce chapitre, nous avons pu présenter l'environnement de développement et l'installation de tous les outils nécessaires pour l'implémentation d'une application ANPR. Nous avons exposé ainsi le résultat de développement à l'aide des captures d'écran.

Conclusion générale

Conclusion générale

Mon projet intitulé **«Reconnaissance d'une plaque d'immatriculation via un Smartphone Android»** consiste à la conception et la réalisation d'une application de reconnaissance automatique des plaques d'immatriculation nommée ANPR. ANPR est divisée en deux parties principales : la détection de la plaque et la reconnaissance de la plaque. La détection a pour but de détecter la position de la plaque dans tout le cadre de la caméra. Quand une plaque sera détectée sur une image, le segment contenant cette plaque est passé à une seconde étape de reconnaissance qui utilise l'algorithme OCR pour déterminer les caractères alphanumériques de la plaque.

J'ai choisi ce thème pour l'intérêt qu'il porte dans le domaine des réseaux informatiques et aussi pour enrichir ma connaissance dans l'espoir de l'appliquer dans la vie.

Durant ma recherche, j'ai remarqué que ce projet n'est pas riche en documentation. Sa préparation m'a beaucoup appris, puisqu'elle a permis de valider les connaissances acquises.

Ce mémoire est donc composé de trois parties. Dans la première, j'ai décrit les Smartphones, leurs systèmes d'exploitation et plus particulièrement le système Android. Puis, j'ai présenté les étapes nécessaires à la création d'une application pour la reconnaissance automatique des plaques d'immatriculation. Finalement, j'ai décrit la conception et l'implémentation du système ANPR.

Le champ d'action de mon sujet, étant très vaste, je n'ai pas la prétention de l'avoir totalement épuisé, mais j'espère avoir répondu aux questions que j'ai en train de me poser, dans le temps et dans l'espace, laissant l'occasion à d'autres personnes pouvant traiter les autres aspects de la question.

Enfin, ce travail s'inscrit dans le cadre des projets innovants et a un impact socioéconomique développé au sein de la Faculté de Technologies, Université de Tlemcen.

En perspective, mon application peut être améliorée en la rendant utilisable pour une large gamme de pays dont les dimensions de la plaque seront introduites au début suivant le choix du pays pour lequel s'applique cette application.

Références & Bibliographie

Références & Bibliographie

[1] Définition et historique du Smartphone - SCT TELECOM
http://www.sct-telecom.fr/glossaire/smartphone/

[2] Les systèmes d'exploitation des smartphones - Mon Petit Mobile
http://www.monpetitmobile.com/choisir-mobile/systemes-exploitation-smartphones

[3] Comment paramétrer manuellement mon mobile - Digicel
www.digicel3gplus.com/Smartphones_Parametrage.pdf?v2

[4] Gartner (Février 2017)
http://www.gartner.com/newsroom/id/3609817

[5] Smartphone, Apple, iOS - Futura Tech
http://www.futura-sciences.com/tech/definitions/smartphone-ios-15211/

[6] Définition, Windows phone – Emag (Electronic magazin)
http://www.emag.tn/tech/tutos/definition-windows-phone-windows-10-mobile/

[7] BlackBerry OS - Wikipédia
https://fr.wikipedia.org/wiki/BlackBerry_OS

[8] Khaoula MRABET & Nessrine TRABELSI - « Application de messagerie simple sur Android : Rapport de projet de VAP RSM »

http://www-public.tem-tsp.eu/~afifi/rapport%20projet%20RSM%20android.pdf

[9] Par AndroWiiid & Frédéric Espiau – « Créez des applications pou Android »
http://nquantin.free.fr/SiteDuZero_android.pdf

[10] Toute l'histoire et la chronologie d'Android - PHONANDROID
http://www.phonandroid.com/toute-l-histoire-et-la-chronologie-d-android-dossier.html

[11] Android Nugat - Wikipédia
https://fr.wikipedia.org/wiki/Android_Nougat

[12] ZDNet (Février 2017)
http://www.zdnet.fr/actualites/samsung-porte-android-nougat-sur-galaxy-s7-et-s7-edge-trop-lent-39847110.htm

[13] Damien Guignard, Julien Chable & Emmanuel Robles – « Programmation Android De la conception au déploiement avec le SDK Google Android 2 »
http://www.cegepsherbrooke.qc.ca/~gagnonju/Fichiers255/Programmation_Android_-_De_la_conception_au_deploiement_avec_le_SDK_Google_Android_2.pdf

Résumé

La reconnaissance automatique des plaques d'immatriculation (ANPR) est un système très précis capable de lire des plaques d'immatriculation de véhicule sans intervention humaine.

L'objectif du présent travail consiste à concevoir et réaliser une application ANPR grâce à l'utilisation des réseaux de neurones artificiels, SVM (Support Vector Machines) et la reconnaissance optique des caractères pour convertir l'image en texte ; se terminant par afficher le texte décodé associé à cette plaque.

Mots clés :

Android, machine à vecteurs de support, réseaux de neurones, reconnaissance optique des caractères, ANPR

Abstract

Automatic License Plate Recognition (ANPR) is a very accurate system that can read vehicle license plates without human intervention.

The aim of this work is to design and realize an ANPR application by using neural networks, Support vector machines and optical character recognition to convert the image into text; Ending by displaying the associated decoded text of that plate.

Keywords:

Android, vector support machine, neural networks, optical character recognition, ANPR.

ملخص

التعرف التلقائي للوحات المركبات (ANPR) هو نظام دقيق جدا يمكن أن يقرأ لوحات ترخيص المركبات دون تدخل بشري.

الهدف من هذا العمل هو تصميم وتحقيق تطبيق ANPR من خلال استخدام الشبكات العصبية، شعاع الدعم الالي والتعرف الضوئي على الحروف لتحويل الصورة إلى نص. ينتهي من خلال عرض النص الخاص باللوحة.

كلمات البحث:

الروبوت، شعاع الدعم الآلي، الشبكات العصبية، التعرف الضوئي على الحروف، ANPR