Workshop - ML Kit Text Recognition (OCR)

DAA

Auteurs

- Duruz Florian
- Ferreira Silva Sven
- Richard Aurélien

13 janvier 2026

Introduction

L'objectif de ce workshop est d'explorer **ML Kit Text Recognition** dans l'écosystème Android. Cette technologie permet d'extraire du texte depuis des images de manière simple et efficace, sans nécessiter de connaissances approfondies en machine learning. Nous allons présenter les fonctionnalités de base et un exemple concret d'implémentation.

Qu'est-ce que ML Kit ?

ML Kit est une bibliothèque de machine learning mobile développée par Google qui permet d'intégrer facilement des fonctionnalités d'intelligence artificielle dans vos applications Android et iOS. ML Kit fournit des modèles pré-entraînés et prêts à l'emploi que vous pouvez intégrer avec quelques lignes de code, sans avoir besoin d'être expert en machine learning.

Les avantages de ML Kit

- **Facilité d'intégration** : Quelques lignes de code suffisent pour ajouter des fonctionnalités IA
- **Performance optimisée** : Google gère l'optimisation pour les appareils mobiles
- **Gratuit pour l'essentiel** : Les APIs on-device sont gratuites et illimitées
- **Fonctionne offline** : Le traitement se fait directement sur l'appareil
- **Cross-platform** : Compatible Android et iOS

Text Recognition : les deux versions disponibles

ML Kit propose deux versions principales pour la reconnaissance de texte :

Version On-device (locale)

- Le traitement se fait **directement sur le smartphone**
- **Pas besoin de connexion Internet**
- **Gratuit et illimité**
- Très bonne précision pour du texte imprimé
- Support de nombreuses langues (Latin, Chinois, Japonais, Coréen, etc.)
- **C'est cette version que nous utilisons dans ce workshop**

Version Cloud (en ligne)

- Le traitement se fait sur les **serveurs Google**
- **Nécessite une connexion Internet**
- Précision maximale
- Support de 100+ langues
- **Payant** après quota gratuit (1000 requêtes/mois)
- Nécessite une clé API Google Cloud

Pour ce workshop, nous utilisons la version On-device qui est amplement suffisante pour la majorité des cas d'usage.

Implémentation dans Android

Étape 1 : Configuration du projet

Ajoutez la dépendance ML Kit dans votre fichier `build.gradle` (niveau app) :

```
dependencies {
    // Pour la reconnaissance de texte Latin (anglais, français, etc.)
    implementation 'com.google.mlkit:text-recognition:16.0.0'
}
```

Note : Si vous avez besoin de reconnaître d'autres scripts (chinois, japonais, etc.), utilisez les dépendances spécifiques :

```
// Pour le chinois
implementation 'com.google.mlkit:text-recognition-chinese:16.0.0'

// Pour le japonais
implementation 'com.google.mlkit:text-recognition-japanese:16.0.0'

// Pour le coréen
implementation 'com.google.mlkit:text-recognition-korean:16.0.0'
```

Téléchargement automatique du modèle : Le modèle ML Kit (environ 10-20 MB) se télécharge automatiquement lors de la première utilisation. Pour forcer le téléchargement dès l'installation de l'app, ajoutez ceci dans votre `AndroidManifest.xml` :

```
<application ...>
    <meta-data
        android:name="com.google.mlkit.vision.DEPENDENCIES"
        android:value="ocr" />
</application>
```

Effet : L'application sera légèrement plus lourde au téléchargement, mais fonctionnera immédiatement sans attendre le téléchargement du modèle.

Étape 2 : Permissions nécessaires

Ajoutez les permissions dans votre `AndroidManifest.xml` :

```
<!-- Permission obligatoire pour utiliser la caméra -->
<uses-permission android:name="android.permission.CAMERA" />

<!-- Permissions pour lire des images depuis la galerie -->
<!-- Pour Android 12 et inférieur -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

<!-- Pour Android 13+ -->
<uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />

<!-- Déclaration de la fonctionnalité caméra (optionnelle) -->
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

Explication des permissions

CAMERA :

- **Obligatoire si** : Vous voulez capturer des photos directement depuis l'app
- **Conséquence si absente** : L'app crash si vous essayez d'ouvrir la caméra
- **Besoin de demande runtime** : OUI (depuis Android 6.0)

READ_EXTERNAL_STORAGE / READ_MEDIA_IMAGES :

- **Obligatoire si** : Vous voulez permettre à l'utilisateur de sélectionner une image depuis sa galerie
- **Conséquence si absente** : L'utilisateur ne pourra pas accéder à ses photos
- **Besoin de demande runtime** : OUI (depuis Android 6.0)

uses-feature camera avec required="false" :

- **Effet** : Votre app reste disponible sur le Play Store même pour les appareils sans caméra (tablettes, etc.)
- **Conséquence si absent** : Votre app ne sera visible que sur les appareils avec caméra
- **Recommandation** : Mettez `required="false"` sauf si votre app ne peut absolument pas fonctionner sans caméra

Gestion des permissions au runtime (obligatoire pour Android 6.0+)

Depuis Android 6.0, il ne suffit pas de déclarer les permissions dans le manifest. Vous devez **demander explicitement l'autorisation à l'utilisateur** au moment d'utiliser la fonctionnalité :

```
class MainActivity : AppCompatActivity() {
    private val REQUEST_CAMERA_PERMISSION = 100
    private val REQUIRED_PERMISSIONS = arrayOf(
        android.Manifest.permission.CAMERA
    )
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Vérifier si les permissions sont accordées
        if (!allPermissionsGranted()) {
            // Demander les permissions à l'utilisateur
            ActivityCompat.requestPermissions(
                this, REQUIRED_PERMISSIONS, REQUEST_CAMERA_PERMISSION
            )
        }
    }

    private fun allPermissionsGranted() = REQUIRED_PERMISSIONS.all {
        ContextCompat.checkSelfPermission(baseContext, it) ==
PackageManager.PERMISSION_GRANTED
    }

    // Cette méthode est appelée après que l'utilisateur ait répondu à la demande
    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == REQUEST_CAMERA_PERMISSION) {
            if (allPermissionsGranted()) {
                // L'utilisateur a accepté, vous pouvez utiliser la caméra
                Toast.makeText(this, "Permission accordée",
Toast.LENGTH_SHORT).show()
            } else {
                // L'utilisateur a refusé
                Toast.makeText(this, "Permission refusée - Impossible d'utiliser
la caméra", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

Point important : Si l'utilisateur refuse la permission, votre fonctionnalité ne marchera pas.

Étape 3 : Préparer l'image

ML Kit fonctionne avec des objets `InputImage`. Voici les méthodes les plus courantes pour créer un `InputImage` :

Depuis un Bitmap (le plus simple pour débuter) :

```
val image = InputImage.fromBitmap(bitmap, 0)
// Le "0" représente la rotation (0 = pas de rotation)
```

Depuis un fichier sélectionné par l'utilisateur :

```
val image = InputImage.fromFilePath(context, uri)
// uri = l'adresse du fichier (obtenu depuis la galerie par exemple)
```

Étape 4 : Initialiser le recognizer

Le recognizer est l'objet qui va effectuer la reconnaissance de texte. Voici comment le créer :

```
class TextRecognitionActivity : AppCompatActivity() {

    // Initialiser le recognizer pour la version ON-DEVICE (locale)
    private val recognizer =
TextRecognition.getClient(TextRecognizerOptions.DEFAULT_OPTIONS)

    // Si vous voulez utiliser la version CLOUD , ce serait :
    // private val recognizer =
TextRecognition.getClient(CloudTextRecognizerOptions.Builder().build())

    override fun onDestroy() {
        super.onDestroy()
        // Important : libérer les ressources quand l'activité est détruite
        recognizer.close()
    }
}
```

Explication de TextRecognizerOptions.DEFAULT_OPTIONS

`TextRecognizerOptions.DEFAULT_OPTIONS` configure le recognizer avec les paramètres par défaut optimaux :

- **Traitement on-device** (local sur l'appareil)
- **Script Latin** (alphabet occidental : A-Z, a-z, 0-9, ponctuation)
- **Fonctionne offline** (pas besoin d'Internet)
- **Gratuit et illimité**

Alternative pour d'autres langues :

```
// Pour le chinois
val recognizer =
TextRecognition.getClient(ChineseTextRecognizerOptions.Builder().build())

// Pour le japonais
val recognizer =
TextRecognition.getClient(JapaneseTextRecognizerOptions.Builder().build())

// Pour le coréen
```

```
val recognizer =
TextRecognition.getClient(KoreanTextRecognizerOptions.Builder().build())
```

Pourquoi DEFAULT_OPTIONS suffit pour commencer ? Parce qu'il couvre la majorité des cas d'usage (texte en français, anglais, espagnol, etc.) et ne nécessite aucune configuration supplémentaire.

Étape 5 : Effectuer la reconnaissance

Voici un exemple complet qui permet de sélectionner une image et d'effectuer la reconnaissance :

```
class TextRecognitionActivity : AppCompatActivity() {

    private lateinit var imageView: ImageView
    private lateinit var resultTextView: TextView
    private lateinit var selectButton: Button
    private val recognizer =
TextRecognition.getClient(TextRecognizerOptions.DEFAULT_OPTIONS)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_text_recognition)

        imageView = findViewById(R.id.image_view)
        resultTextView = findViewById(R.id.result_text)
        selectButton = findViewById(R.id.button_select)

        selectButton.setOnClickListener {
            selectImageFromGallery()
        }
    }

    private fun selectImageFromGallery() {
        val intent = Intent(Intent.ACTION_PICK)
        intent.type = "image/*"
        startActivityForResult(intent, REQUEST_IMAGE_PICK)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == REQUEST_IMAGE_PICK && resultCode == RESULT_OK) {
            data?.data?.let { uri ->
                // Charger l'image sélectionnée
                val bitmap = MediaStore.Images.Media.getBitmap(contentResolver,
uri)
                imageView.setImageBitmap(bitmap)

                // Lancer la reconnaissance
                processImage(bitmap)
            }
```

```
        }
    }

    private fun processImage(bitmap: Bitmap) {
        // Créer un InputImage depuis le Bitmap
        val image = InputImage.fromBitmap(bitmap, 0)

        // Lancer la reconnaissance
        recognizer.process(image)
            .addOnSuccessListener { visionText ->
                // Succès : afficher le texte détecté
                displayResults(visionText)
            }
            .addOnFailureListener { e ->
                // Erreur : afficher un message
                Toast.makeText(this, "Erreur: ${e.message}",
Toast.LENGTH_SHORT).show()
                Log.e("TextRecognition", "Erreur de reconnaissance", e)
            }
    }

    private fun displayResults(text: Text) {
        if (text.text.isEmpty()) {
            resultTextView.text = "Aucun texte détecté"
            return
        }

        // Afficher simplement tout le texte détecté
        resultTextView.text = text.text
    }

    override fun onDestroy() {
        super.onDestroy()
        recognizer.close()
    }

    companion object {
        private const val REQUEST_IMAGE_PICK = 1001
    }
}
```

Layout XML correspondant

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <Button
```

```
        android:id="@+id/button_select"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Sélectionner une image" />

    <ImageView
        android:id="@+id/image_view"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:layout_marginTop="16dp"
        android:scaleType="centerInside"
        android:background="@android:color/darker_gray" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_marginTop="16dp">

        <TextView
            android:id="@+id/result_text"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Aucun résultat"
            android:textSize="16sp"
            android:padding="8dp" />
    </ScrollView>

</LinearLayout>
```

Cas d'usage pratiques

Scanner de cartes de visite

Extraire automatiquement nom, téléphone, email depuis une photo de carte de visite.

Numérisation de documents

Transformer des documents papier en texte éditable pour archivage ou recherche.

Traduction en temps réel

Combiner Text Recognition avec l'API Translation de ML Kit pour traduire du texte depuis la caméra.

Accessibilité

Lire à haute voix le texte détecté pour aider les personnes malvoyantes (avec Text-to-Speech).

Extraction de données

Détecter et extraire des informations spécifiques (codes produits, numéros de série) depuis des photos.

Points d'attention importants

Taille des images

ML Kit fonctionne mieux avec des images de **résolution moyenne** (environ 1000-2000 pixels de largeur). Des images trop grandes :

- Ralentissent le traitement
- Consomment plus de mémoire
- N'améliorent pas la précision

Solution : Redimensionner les images trop grandes avant traitement.

Qualité de l'image

Pour obtenir de bons résultats, l'image doit être :

- **Nette** (pas floue)
- **Bien éclairée** (pas trop sombre)
- **Sans reflets** ni ombres gênantes
- Avec un **bon contraste** entre le texte et le fond

Libération des ressources

Toujours fermer le recognizer dans `onDestroy()` pour libérer la mémoire :

```
override fun onDestroy() {
    super.onDestroy()
    recognizer.close()
}
```

Conséquence si oublié : Fuite de mémoire qui peut ralentir ou crasher l'application.

Gestion du premier lancement

Attention ! Si le téléchargement du modèle n'as pas été forcé dans le manifest, la première utilisation peut échouer car le modèle est en cours de téléchargement.

Comparaison : On-device vs Cloud

Critère	On-device (ce workshop)	Cloud API
Connexion Internet	❌ Non requise	✅ Requise
Coût	✅ Gratuit illimité	⚠️ Payant après 1000/mois
Vitesse	✅ Rapide (<1s)	⚠️ Variable (dépend du réseau)
Précision	✅ Très bonne	✅ Excellente
Configuration	✅ Simple	⚠️ Nécessite clé API

Critère	On-device (ce workshop)	Cloud API
Vie privée	✅ Totale (local)	⚠ Données envoyées à Google

Conclusion

ML Kit Text Recognition offre une solution simple et efficace pour ajouter de la reconnaissance de texte dans vos applications Android. Avec seulement quelques lignes de code, vous pouvez :

- Extraire du texte depuis des photos
- Numériser des documents
- Créer des fonctionnalités d'accessibilité
- Le tout gratuitement et sans connexion Internet

Les points clés à retenir :

- ✅ **Simple** : Peu de code nécessaire
- ✅ **Gratuit** : La version on-device est illimitée
- ✅ **Offline** : Fonctionne sans Internet
- ✅ **Performant** : Résultats en moins d'une seconde
- ⚠ **Permissions** : Géré au runtime
- ⚠ **Qualité** : L'image doit être nette et bien éclairée

Bibliographie

- Documentation officielle ML Kit - Text Recognition
- Exemples de code Google
- ChatGPT a été utilisé pour la correction syntaxique et l'amélioration de certaines formulations et la mise en page (notamment les icônes)