# Caméra

## Introduction

La bibliothèque CameraX est un ajout à Android Jetpack qui facilite l'utilisation de la caméra/appareil photo au sein d'une application Android. Cette librairie a été présentée comme une couche d'abstraction à Camera2.

Jetpack est une suite de bibliothèques, d'outils et de conseils pour aider les développeurs à écrire des applications de haute qualité plus facilement. Jetpack comprend les bibliothèques de packages `androidx. *`.

Liste des classes de base de la bibliothèque CameraX

## Gradle

Pour commencer, il faut ajouter les dépendances CameraX au fichier `app/build.gradle` de l'application :

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"
    defaultConfig {
        applicationId "com.example.myapplicationcamera"
        minSdkVersion 21
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rule
s.pro'
        }
    }
    /*compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }*/
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    def cameraxVersion = "1.0.0-alpha02"
    implementation "androidx.camera:camera-core:${cameraxVersion}"
    implementation "androidx.camera:camera-camera2:${cameraxVersion}"
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

Pour terminer, il faut faire `File` ➔ `Sync Project with Gradle Files`.

# Le layout

Il faut utiliser un `TextureView` pour afficher la vue de la caméra. On va aussi ajouter un bouton pour effectuer une capture.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/
res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextureView
        android:id="@+id/view_finder"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <ImageButton
        android:id="@+id/boutonCapturer"
        android:layout_width="72dp"
        android:layout_height="72dp"
        android:layout_margin="24dp"
        app:srcCompat="@android:drawable/ic_menu_camera"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

*Remarque :* on pourrait ajouter un bouton pour sélectionner la caméra (*back* ou *front*).

# AndroidManifest.xml

Il faut ajouter la permission d'utiliser la caméra dans le fichier `AndroidManifest.xml`. Pour sauvegarder des captures, on peut aussi ajouter la permission `WRITE_EXTERNAL_STORAGE` :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplicationcamera">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
```

```
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

# Principe

CameraX fournit une approche basée sur les cas d'utilisation (*use case*). Les trois cas d'utilisation principaux sont :

- Aperçu (**Preview**) : l'affichage du flux de la caméra.
- Analyse d'image (**Image Analysis**) : le traitement du flux de la caméra.
- Capture d'image (**Image Capture**) : la capture photo / vidéo.

Les cas d'utilisation peuvent être combinés et actifs simultanément.

Les cas d'utilisation se configurent (PreviewConfig, ImageAnalysisConfig et ImageCaptureConfig) avec les méthodes `set()` fournies par le `Builder` et se finalisent avec la méthode `build()`.

Exemple de configuration pour le cas d'utilisation **Preview** :

```
Size screen = new Size(viewFinder.getWidth(), viewFinder.getHeight());

PreviewConfig previewConfig = new PreviewConfig.Builder()
    .setTargetResolution(screen)
    .setLensFacing(CameraX.LensFacing.FRONT)
    .build();

Preview preview = new Preview(previewConfig);
```

Il n'est plus nécessaire de redéfinir les méthodes `onResume()`, `onPause()`, … car CameraX gère un cycle de vie en appelant la méthode `bindToLifecycle()` et en lui précisant les cas d'utilisation à gérer.

Exemple pour le cas d'utilisation **Preview** seul : à chaque fois que celui-ci sera actif, on va recevoir `PreviewOutput` qui permettra de mettre à jour le `TextureView`

```
Size screen = new Size(viewFinder.getWidth(), viewFinder.getHeight()); //size of the screen

PreviewConfig previewConfig = new PreviewConfig.Builder()
        .setTargetResolution(screen)
        //.setLensFacing(CameraX.LensFacing.FRONT)
        .build();
Preview preview = new Preview(previewConfig);

preview.setOnPreviewOutputUpdateListener(new Preview.OnPreviewOutputUpdateListener()
{
    @Override
    public void onUpdated(Preview.PreviewOutput output)
```

```
    {
        // on met à jour le TextureView
        viewFinder.setSurfaceTexture(output.getSurfaceTexture());
    }
});

CameraX.bindToLifecycle((LifecycleOwner)this, preview); // ici un seul cas d'utilisation : previe
w
```

Pour le cas d'utilisation **Image Capture**, on va utiliser le bouton et appeler la méthode `takePicture()` :

```
...
ImageCaptureConfig imageCaptureConfig = new ImageCaptureConfig.Builder()
        .setCaptureMode(ImageCapture.CaptureMode.MIN_LATENCY)
        .setTargetRotation(getWindowManager().getDefaultDisplay().getRotation())
        .build();

final ImageCapture imageCapture = new ImageCapture(imageCaptureConfig);

findViewById(R.id.boutonCapturer).setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        File file = new File(Environment.getExternalStorageDirectory() + "/IMG_" + System.current
TimeMillis() + ".png");
        imageCapture.takePicture(file, new ImageCapture.OnImageSavedListener()
        {
            @Override
            public void onImageSaved(@NonNull File file)
            {
                Toast.makeText(getBaseContext(), file.getAbsolutePath(), Toast.LENGTH_LONG).show(
);
            }

            @Override
            public void onError(@NonNull ImageCapture.UseCaseError useCaseError, @NonNull String
message, @Nullable Throwable cause)
            {
                Toast.makeText(getBaseContext(), "Erreur capture : " + message, Toast.LENGTH_LONG
).show();
                if(cause != null)
                {
                    cause.printStackTrace();
                }
            }
        });
    }
});

CameraX.bindToLifecycle((LifecycleOwner)this, preview, imageCapture); // ici les deux cas d'utili
sation
```

Exemple pour le cas d'utilisation **Image Analysis** :

```
...
ImageAnalysisConfig imageAnalysisConfig = new ImageAnalysisConfig.Builder()
```

```
        //.setTargetResolution(new Size(1280, 720))
        .setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE) // la dernière im
age de la file d'attente, en supprimant toutes les images antérieures
        .build();

ImageAnalysis imageAnalysis = new ImageAnalysis(imageAnalysisConfig);

imageAnalysis.setAnalyzer( new ImageAnalysis.Analyzer()
{
        @Override
        public void analyze(ImageProxy image, int rotationDegrees)
        {
                // Faire le traitement d'image ici
                Log.v(TAG, "image := " + image.getWidth() + "x" + image.getHeight());
        }
});

CameraX.bindToLifecycle((LifecycleOwner)this, imageAnalysis, preview, imageCapture); // ici avec
les trois cas d'utilisation
```

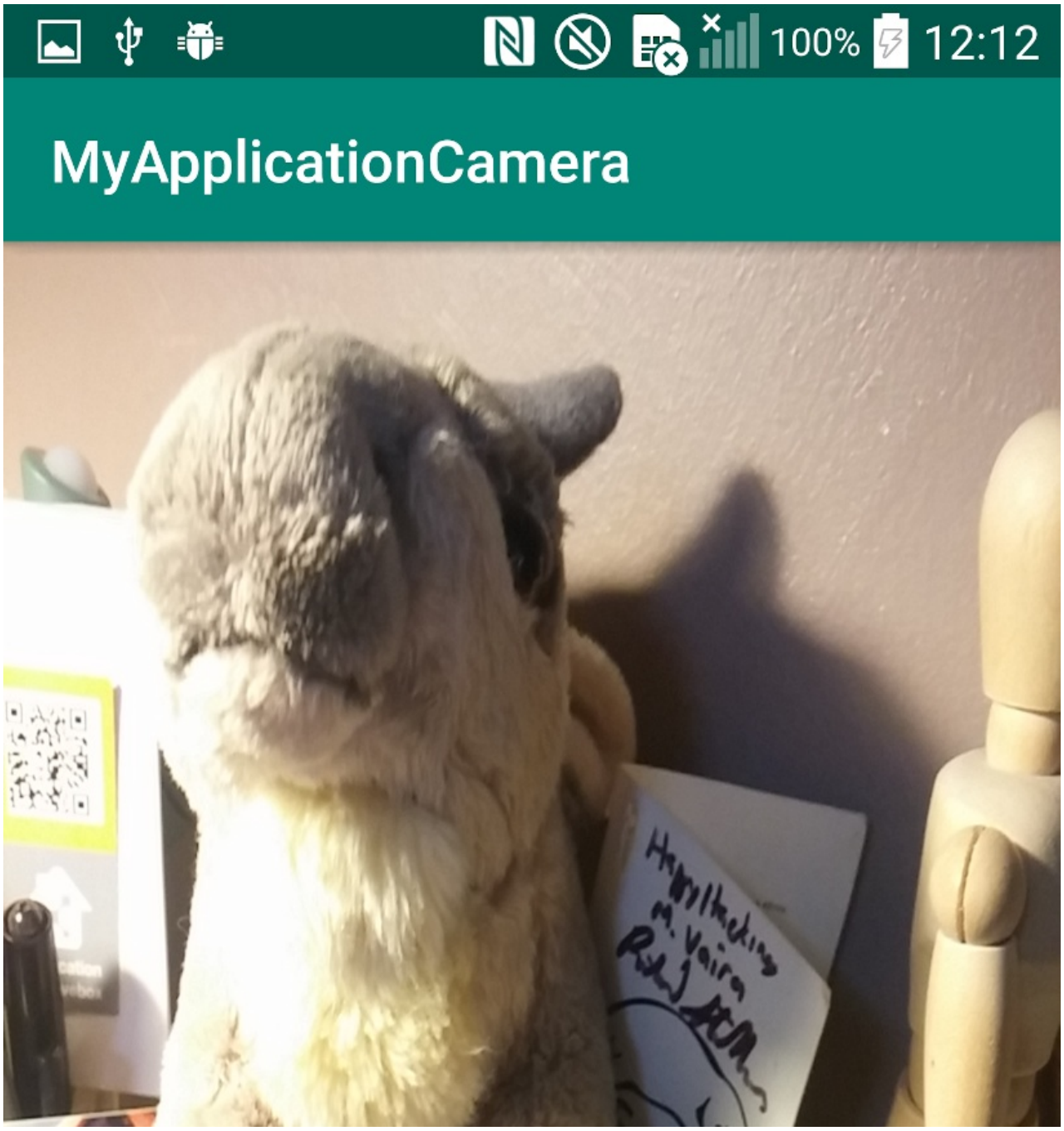
*Remarque :* la méthode `setImageReaderMode()` permet de choisir l'image à analyser à partir de la file d'attente. Cela peut être soit `ACQUIRE_LATEST_IMAGE` (la dernière image de la file d'attente) ou `ACQUIRE_NEXT_IMAGE` (l'image suivante de la file d'attente). Ces deux modes définissent le mode d'analyse d'image : bloquant ( `ACQUIRE_NEXT_IMAGE` ) ou non-bloquant ( `ACQUIRE_LATEST_IMAGE` ).

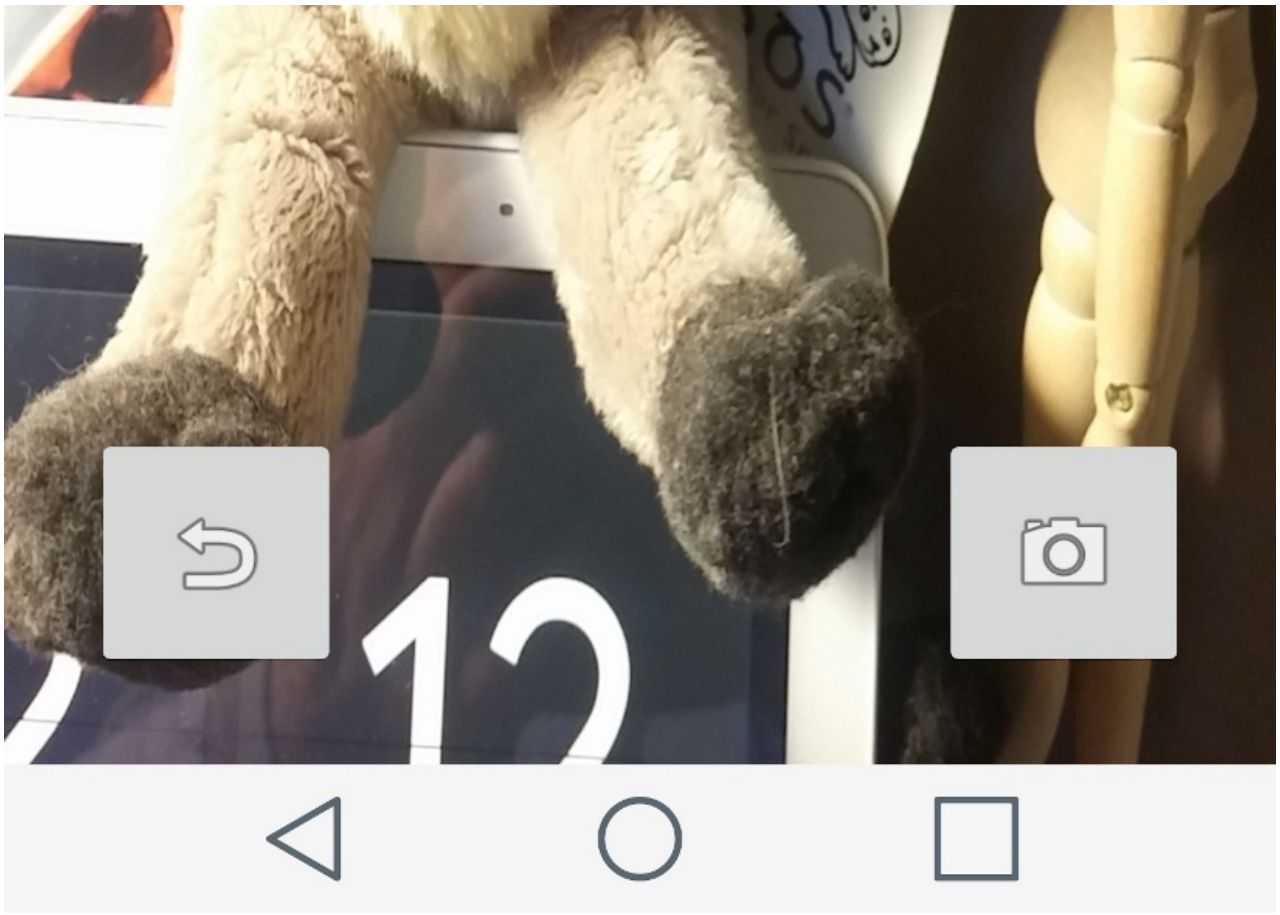
# Sélection de caméra

On va modifier le code de base pour intégrer un bouton de sélection de la caméra (avant et arrière) :

```
...
private CameraX.LensFacing choixCamera = CameraX.LensFacing.BACK;
...

private void demarrerCamera()
{
    demarrerCasUtilisationCameraX();

    ImageButton boutonSelectionner = findViewById(R.id.boutonSelectionner);
    boutonSelectionner.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View v)
        {
            if(choixCamera == CameraX.LensFacing.FRONT)
                choixCamera = CameraX.LensFacing.BACK;
            else
                choixCamera = CameraX.LensFacing.FRONT;
            demarrerCasUtilisationCameraX();
        }
    });
}

private void demarrerCasUtilisationCameraX()
{
        CameraX.unbindAll();

        Size screen = new Size(viewFinder.getWidth(), viewFinder.getHeight()); //size of the scre
en
```

```
        PreviewConfig previewConfig = new PreviewConfig.Builder()
                .setTargetAspectRatio(aspectRatio)
                .setTargetResolution(screen)
                .setLensFacing(choixCamera)
                .build();
        Preview preview = new Preview(previewConfig);

        ...

        //CameraX.bindToLifecycle((LifecycleOwner)this, imageAnalysis, preview, imageCapture);
        //CameraX.bindToLifecycle((LifecycleOwner)this, preview, imageCapture);
        CameraX.bindToLifecycle((LifecycleOwner)this, preview);
}
```





## OpenCV

**OpenCV** (*Open Computer Vision*) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. Cette bibliothèque est distribuée sous licence BSD.

Il est possible d'intégrer **OpenCV** dans une application Android pour réaliser un traitement d'image sur le flux de la caméra.

Il faut télécharger et importer le SDK OpenCV à partir du GitHub officiel. Le mieux est de l'ajouter dans les dépendances du fichier `app/build.gradle` de l'application :

```
dependencies {
    ...
    implementation 'com.quickbirdstudios:opencv:3.4.1'
    ...
}
```

On modifie ensuite le *layout* de l'application afin d'ajouter un `ImageView` qui servira à afficher le flux en niveau de gris.

Dans le code ci-dessous, on modifie le cas d'utilisation **Image Analysis** pour récupérer le *bitmap* à partir du `TextureView`.

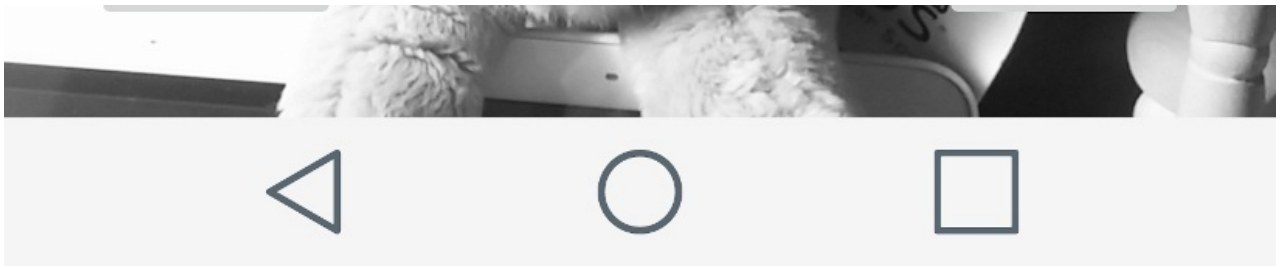
On utilise la fonction `Utils.bitmapToMat()` pour convertir l'objet `Bitmap` en objet `Mat`. L'analyse d'image se limite ici à convertir l'espace colorimétrique Mat d'un type à un autre (pour le niveau de gris, on choisit `Imgproc.COLOR_RGB2GRAY`) en utilisant `ImgProc.cvtColor()`. On revient à un objet `Bitmap`

avec `Utils.matToBitmap()` puis on l'affiche dans l' `ImageView`.

Le traitement nécessite l'utilisation des *threads*.

```
private ImageView imageBitmap;

...

imageBitmap = findViewById(R.id.image_bitmap);

...

HandlerThread threadAnalise = new HandlerThread("OpenCV");
threadAnalise.start();

ImageAnalysisConfig imageAnalysisConfig = new ImageAnalysisConfig.Builder()
                .setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE)
                .setCallbackHandler(new Handler(threadAnalise.getLooper()))
                .setImageQueueDepth(1)
                .build();

ImageAnalysis imageAnalysis = new ImageAnalysis(imageAnalysisConfig);

imageAnalysis.setAnalyzer( new ImageAnalysis.Analyzer()
{
    @Override
    public void analyze(ImageProxy image, int rotationDegrees)
    {
        final Bitmap bitmap = viewFinder.getBitmap();

        if(bitmap == null)
            return;

        Mat mat = new Mat();
        Utils.bitmapToMat(bitmap, mat);

        Imgproc.cvtColor(mat, mat, Imgproc.COLOR_RGB2GRAY); // conversion en niveau de gris
        Utils.matToBitmap(mat, bitmap);
        runOnUiThread(new Runnable()
        {
            @Override
            public void run()
            {
                imageBitmap.setImageBitmap(bitmap);
            }
        });
    }
});

...

CameraX.bindToLifecycle((LifecycleOwner)this, imageAnalysis, preview, imageCapture);
```

MyApplicationCamera



# ML Kit

ML Kit est un SDK mobile (intégré à Firebase) qui apporte l'expertise de Google en *machine learning* pour les applications Android (et iOS).

La partie API repose sur *Google Cloud Vision* avec l'avantage d'être disponible via le *cloud* ou directement en local. ML Kit embarque les modèles suivants :

- Reconnaissance de caractères (disponible en local et sur le *cloud*)
- Ajout de libellés à des images (disponible en local et sur le *cloud*)
- Détection de visage (uniquement en local)
- Lecture de codes-barres (uniquement en local)
- Reconnaissance de points de repère (uniquement sur le *cloud*)

Il faut tout d'abord ajouter Firebase au projet Android.

Pour utiliser l'API sur l'appareil mobile, il faut configurer l'application pour télécharger automatiquement le modèle ML. Pour cela, on ajoute la déclaration suivante au fichier `AndroidManifest.xml` de l'application (ici pour l'ocr et les *labels*) :

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

## Ajout de libellés

On ajoute la dépendance dans le fichier `app/build.gradle` de l'application :

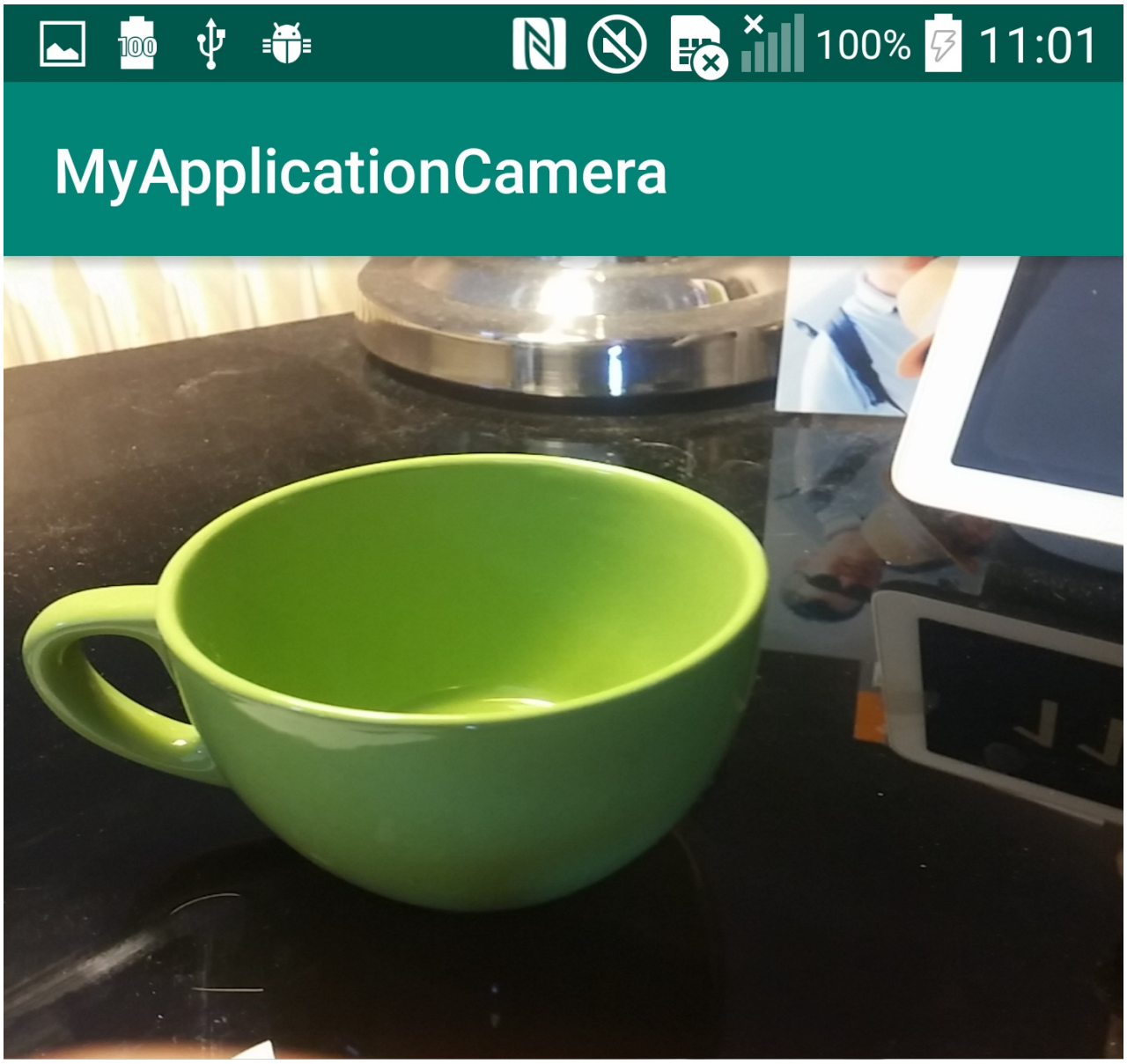
```
dependencies {
    ...
    implementation 'com.google.firebase:firebase-ml-vision:24.0.1'
    implementation 'com.google.firebase:firebase-ml-vision-image-label-model:19.0.0'
    ...
}
```

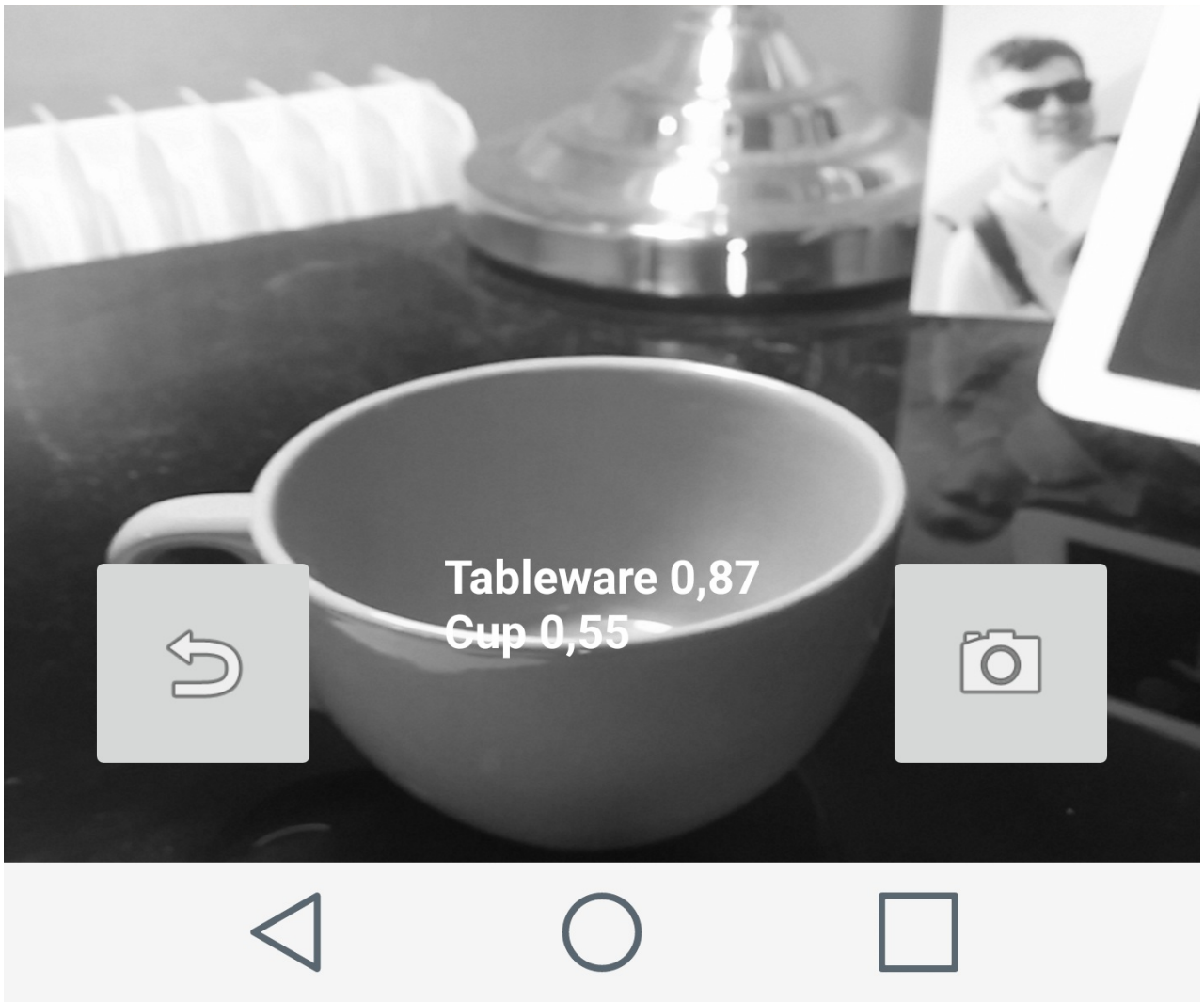
Pour Android, le principe est détaillé ici. On ajoute un `TextView` au layout pour afficher le *label* détecté pour l'image :

```
...
HandlerThread threadAnalise = new HandlerThread("Label");
threadAnalise.start();
```

```
ImageAnalysisConfig imageAnalysisConfig = new ImageAnalysisConfig.Builder()
                //.setTargetResolution(new Size(1280, 720))
                .setLensFacing(choixCamera)
                .setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE)
                .setCallbackHandler(new Handler(threadAnalise.getLooper()))
                .setImageQueueDepth(1)
                .build();

ImageAnalysis imageAnalysis = new ImageAnalysis(imageAnalysisConfig);

imageAnalysis.setAnalyzer( new ImageAnalysis.Analyzer()
{
    @Override
    public void analyze(ImageProxy image, int rotationDegrees)
    {
        Image mediaImage = image.getImage();
        if(mediaImage == null)
            return;
        int rotation = degreesToFirebaseRotation(rotationDegrees);

        FirebaseVisionImage visionImage = FirebaseVisionImage.fromMediaImage(mediaImage, rotation
);

        FirebaseVisionImageLabeler labeler = FirebaseVision.getInstance().getOnDeviceImageLabeler
();
        labeler.processImage(visionImage).addOnSuccessListener(new OnSuccessListener<List<Firebas
eVisionImageLabel>>()
        {
            @Override
            public void onSuccess(List<FirebaseVisionImageLabel> labels)
            {
                String labelTexte = "";
                for (FirebaseVisionImageLabel label: labels)
                {
                    labelTexte += label.getText() + " " + String.format("%.2f", label.getConfiden
ce()) + "\n";
                }
                if(!labelTexte.isEmpty())
                {
                    final String labelTexteUI = labelTexte;
                    runOnUiThread(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            label.setText(labelTexteUI);
                        }
                   });
                }
            }
        }).addOnFailureListener(new OnFailureListener()
        {
            @Override
            public void onFailure(@NonNull Exception e)
            {
            }
        });
    }
});
```

```
CameraX.bindToLifecycle((LifecycleOwner)this, imageAnalysis, preview, imageCapture);
...

private int degreesToFirebaseRotation(int rotationDegrees)
{
    switch (rotationDegrees)
    {
        case 0:
            return FirebaseVisionImageMetadata.ROTATION_0;
        case 90:
            return FirebaseVisionImageMetadata.ROTATION_90;
        case 180:
            return FirebaseVisionImageMetadata.ROTATION_180;
        case 270:
            return FirebaseVisionImageMetadata.ROTATION_270;
        default:
            throw new IllegalArgumentException("Erreur rotation !");
    }
}
```



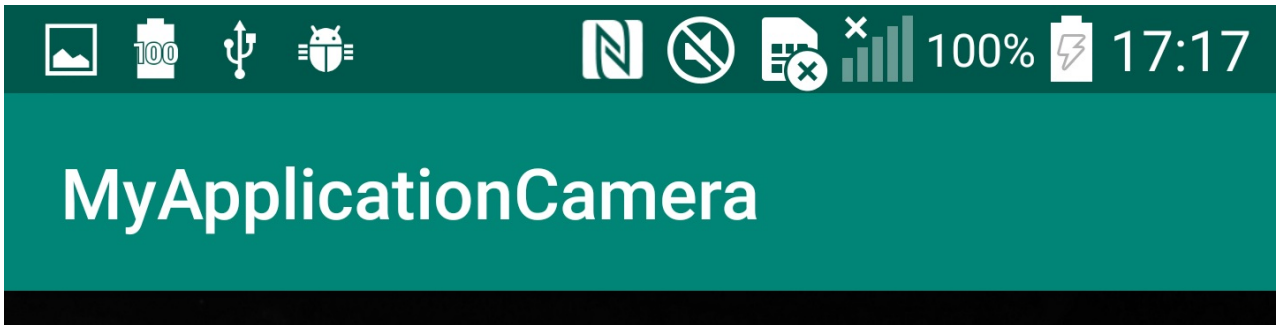



## Reconnaissance de caractères (ocr)

On ajoute la dépendance dans le fichier `app/build.gradle` de l'application :

```
dependencies {
    ...
    implementation 'com.google.firebase:firebase-ml-vision:24.0.1'
    ...
}
```

Pour Android, le principe est détaillé ici.



inistration Système UNIX ainistration Système UNIX

Travaux Pratiques Travaux Pratiques

uvelle machine virtuelle. Avant de lancer l'installati

achine virtuelle en mode pont (*bridge*). achine virtuelle en r

. Il faut : Il faut:

Volu ager). Volume Manager).

rsonnel. rsonnel.

ème de base, pas de paquets supplémentaires mo