🎨 Real-Time Edge Detection Android App

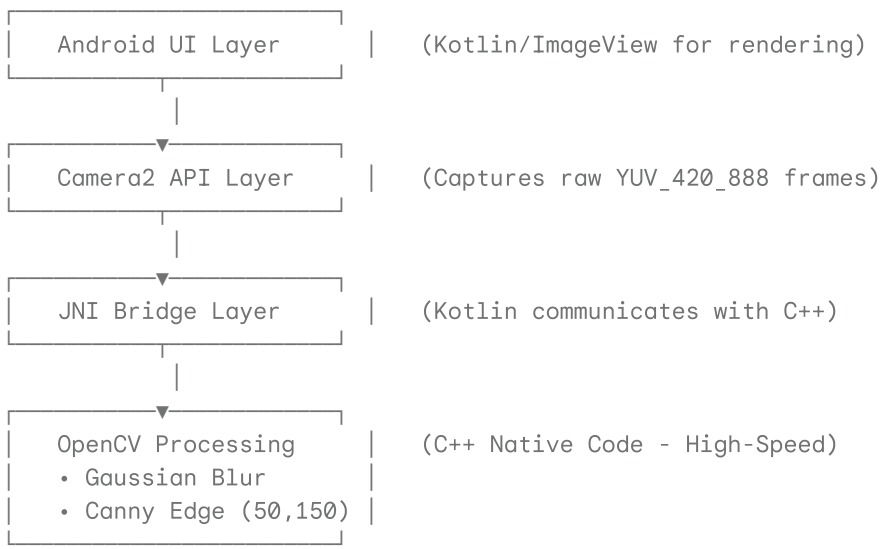
A high-performance Android application that performs **real-time Canny edge detection** on a camera feed, leveraging **OpenCV C++** via **JNI/NDK** for superior native performance.

📱 Core Features

- ⚡ **Real-Time Performance:** Achieves **20-30 FPS** edge detection directly on the device.
- 🚀 **Native Acceleration: OpenCV C++** implementation optimized and accessed via **JNI/NDK**.
- 📷 **Modern Camera:** Uses the high-performance **Camera2 API** for frame acquisition.
- 💾 **Frame Export:** Saves processed edge-detected images to device storage.
- 📊 **Live Metrics:** Immediate visual feedback with an integrated **FPS counter**.
- ⚙️ **Optimization:** Processes frames at **640x480** for balanced quality and speed.

📐 Technical Architecture

The application adopts a layered architecture to maximize processing efficiency by offloading computer vision tasks to the native C++ layer.



🛠️ Technologies & Dependencies

Android/Kotlin Layer

Component	Details

Language	Kotlin
Min/Target SDK	API 24 (Android 7.0) / API 36
Camera Interface	**Camera2 API**
Build Tools	Gradle 8.7 + CMake 3.22.1

Native/C++ Layer

Component	Details
Language	**C++17**
Vision Library	**OpenCV 4.10.0**
Native Integration	**Android NDK r26 / JNI**

Documentation

Component	Details
Web Viewer	TypeScript, HTML5 + CSS3 (For statistics and architecture)

📋 Prerequisites & Setup

Requirements

- **Android Studio:** Hedgehog (2023.1.1) or later
- **JDK:** 17
- **Android SDK:** API 24-36
- **NDK:** r26 or later
- **CMake:** 3.22.1+
- **OpenCV Android SDK:** 4.10.0
- **Node.js:** 16+ (Optional for web viewer)

Installation Steps

1. Clone Repository

```
git clone [https://github.com/yourusername/EdgeDetectionApp.git](https://github.com/you
cd EdgeDetectionApp
```

2. Download and Configure OpenCV

1. Download **OpenCV 4.10.0 for Android** from opencv.org.
2. Extract the SDK. For Windows, assume the path is `C:\OpenCV-android-sdk\` .

3. Copy Native Libraries (`.so` files)

Copy the necessary shared object files from the OpenCV SDK into the project's native libraries directory to ensure compatibility:

```
# Source: [OpenCV-android-sdk]/sdk/native/libs/
# Destination: EdgeDetectionApp/app/src/main/jniLibs/
```

Required Structure:

```
jniLibs/
  ├── arm64-v8a/
  │   ├── libopencv_java4.so
  │   └── libc++_shared.so
  └── armeabi-v7a/
      ├── libopencv_java4.so
      └── libc++_shared.so
```

4. Verify CMake Configuration

Ensure the `OpenCV_DIR` variable in `app/src/main/cpp/CMakeLists.txt` correctly points to your OpenCV installation:

```
set(OpenCV_DIR "C:/OpenCV-android-sdk/sdk/native/jni")
```

5. Build in Android Studio

1. Open the project folder in **Android Studio**.
2. Allow Gradle sync to complete.
3. Run **Build → Clean Project**, then **Build → Rebuild Project**.

📱 Running the App

For optimal performance, a **physical device is recommended**.

Launching on Device

1. Enable **Developer Options** and **USB Debugging** on your device.
2. Connect the device and select it from the Android Studio run dropdown.
3. Click the green **Run** button (▶️).

Usage Flow

1. **Launch App** and **Grant Camera Permissions**.
2. **View Edges:** Real-time Canny edge detection begins instantly.
3. **Monitor:** Check the **FPS counter** for performance metrics.
4. **Save Frame:** Tap the "Save Frame" button to export the current processed image to `Gallery → Pictures → EdgeDetection`.

🎯 Key Component Overview

File	Language	Function
`MainActivity.kt`	Kotlin	UI management, permission handling, FPS display, and frame saving logic.
`CameraManager.kt`	Kotlin	Configures the **Camera2 API**, manages the capture session, and provides raw `YUV_420_888` frames.
`NativeProcessor.kt`	Kotlin/JNI	The **JNI bridge** class that loads the native library and declares C++ functions for Kotlin access.
`native-lib.cpp`	C++	Contains the **OpenCV processing pipeline** (YUV conversion, Gaussian Blur, Canny edge detection).

🔧 Algorithm Details and Code

The core image processing is handled in **C++** for speed.

Canny Edge Detection Pipeline

The pipeline transforms the raw camera frame into the final edge-detected output:

1. **Color Conversion**: $\text{YUV\_420\_888} \rightarrow \text{RGB} \rightarrow \textbf{Grayscale}$
2. **Noise Reduction**: **Gaussian Blur** (5×5 kernel, $\sigma = 1.5$)
3. **Edge Detection**: **Canny Algorithm** (Thresholds: 50 to 150)
4. **Display**: Final conversion of the edge map back to RGB for UI rendering.

C++ Processing Code Snippet

```
// ... native function entry ...

// Convert YUV to RGB (assuming NV21/YUV_420_888 format)
cv::Mat yuvMat(height + height/2, width, CV_8UC1, inputBytes);
cv::Mat rgbMat;
cv::cvtColor(yuvMat, rgbMat, cv::COLOR_YUV2RGB_NV21);

// Convert to grayscale
cv::Mat grayMat;
cv::cvtColor(rgbMat, grayMat, cv::COLOR_RGB2GRAY);

// Apply Gaussian blur for noise reduction
cv::Mat blurredMat;
cv::GaussianBlur(grayMat, blurredMat, cv::Size(5, 5), 1.5);

// Apply Canny edge detection
cv::Mat edgesMat;
cv::Canny(blurredMat, edgesMat, 50, 150);

// Convert back to RGB for display on Android UI
cv::Mat resultMat;
cv::cvtColor(edgesMat, resultMat, cv::COLOR_GRAY2RGB);

// ... copy resultMat back to jbyteArray output ...
```

⚡ Performance Metrics

Metric	Value	Detail
FPS	**20-30**	Real-time performance on device.
Processing Time	3-5ms/frame	Time spent executing C++ native code.
Resolution	640x480	Fixed processing resolution.
Canny Thresholds	Low: 50, High: 150	Default parameters.
Memory Usage	~50MB	Runtime memory profile (approximate).

🐛 Troubleshooting

Problem	Potential Solution
`libopencv_java4.so not found`	**Missing JNILibs:** Ensure `libopencv_java4.so` and `libc++_shared.so` are copied to `app/src/main/jniLibs/` for both `arm64-v8a` and `armeabi-v7a`.

App crashes on launch	Check **Logcat** for an `UnsatisfiedLinkError`. Verify all `.so` files are present and **Clean/Rebuild** the project.
Black screen / No detection	1. Check Logcat for Camera2 errors. 2. Manually **grant Camera Permission** in device settings. 3. Verify native libraries loaded successfully.
CMake configuration failed	Double-check that `OpenCV_DIR` in `CMakeLists.txt` is the **correct, absolute path** to the extracted OpenCV SDK.

Development Notes

Extending Algorithms

To add a new C++ processing function (e.g., Sobel edge detection):

1. **C++ Implementation** (`native-lib.cpp`):

```
extern "C" JNIEXPORT void JNICALL
Java_com_example_edgedetectionapp_NativeProcessor_sobelEdgeDetection(
    JNIEnv* env, jobject, jbyteArray input,
    jint width, jint height, jbyteArray output) {
    // Sobel Implementation Here
}
```

2. **Kotlin Declaration** (`NativeProcessor.kt`):

```
external fun sobelEdgeDetection(
    input: ByteArray,
    width: Int,
    height: Int,
```