ESTUDIANTES:

ANTHONY PULLA

KELLY PALTIN

ASIGNATURA:

VISION POR COMPUTADOR

TEMA:

RECONOCIMIENTO DE FORMAS USANDO MOMENTOS DE HU Y MOMENTOS DE ZERNIKE Y CLASIFICACIÓN DE IMÁGENES USANDO PATRONES BINARIOS LOCALES (LBP)

DOCENTE:

ING. VLADIMIR ROBLES

FECHA:

27 DE JUNIO DE 2024

PERIODO ACADÉMICO:

64

PRACTICA #3.1 – RECONOCIMIENTO DE FORMAS USANDO MOMENTOS DE HU Y MOMENTOS DE ZERNIKE Y CLASIFICACIÓN DE IMÁGENES USANDO PATRONES BINARIOS LOCALES (LBP)

Parte 1. Desarrollar una aplicación móvil que permita calcular los momentos invariantes de HU y los momentos de Zernike. Para ello, deberá realizar las siguientes tareas:

1. Trabajar con el corpus UPS-Writing-Skills que contiene diversas imágenes que representan 3 figuras geométricas (círculo, triángulo y cuadrado) que trazaron niños con y sin necesidades educativas especiales, como las que se muestran en la Ilustración 1:

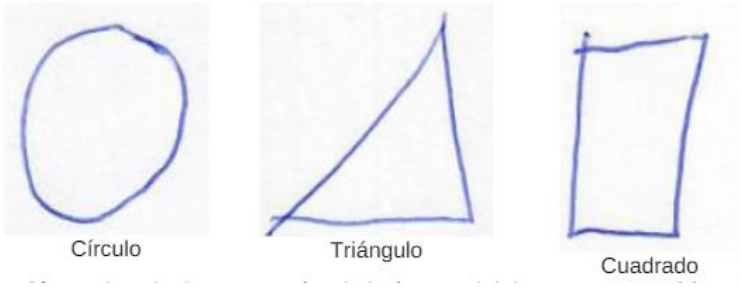


Ilustración 1. Ejemplo de 4 categorías de imágenes del dataset "UPS-Writing-Skills"

2. Debe crear una aplicación móvil en una librería nativa de C++ donde realizará todos los cálculos y referenciará al corpus y cuando se le muestre una imagen indicará qué tipo de figura es, presentando en pantalla la etiqueta correspondiente ("triángulo", "círculo" o "cuadrado").

3. Debe preprocesar las imágenes, convirtiéndolas en imágenes a blanco y negro, donde el trazo debe estar con color blanco y el fondo negro. Puede usar funciones de skeltización o rellenar el área interior de la figura geométrica. Debe buscar aplicar alguna técnica que permita mejorar los resultados. Se sugiere revisar el siguiente artículo:

• https://link.springer.com/chapter/10.1007/978-3-031-19647-8_22

4. Debe calcular los momentos invariantes de HU y los momentos de Zernike y con ellos realizar la clasificación usando la distancia Euclídea como se vio en clase.

5. Para el caso de los momentos de Zernike, puede implementar código de terceros en la librería nativa, como por ejemplo:

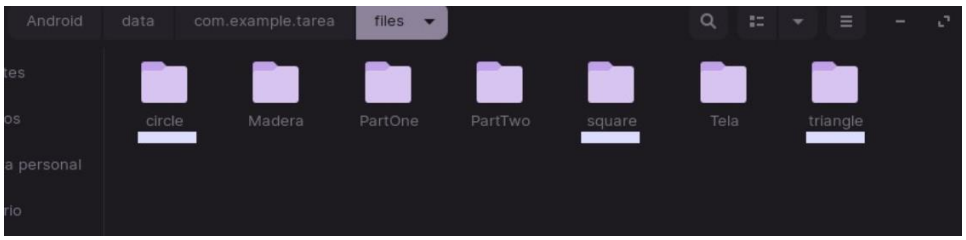
➢ https://github.com/chris-allan/pychrm/blob/master/src/textures/zernike/zernike.cpp

Código de la Parte 1 del Programa:

```java
private void processFrame(Mat img, String paramValue) {
    Log.i(TAG, msg: "Entro a saveFrame con paramValue: " + paramValue);
    if(paramValue.equals("PartOne")){
        cvtColor(img, img,COLOR_BGR2RGB);
        String as = Classification(img.getNativeObjAddr());
        String asZernike = momentsZernike(img.getNativeObjAddr());
        String moments = momentResult();
        long imgProcess = matMascara();
        Mat newFrameImg = new Mat(imgProcess);
        File filter = new File(getExternalFilesDir( type: null), paramValue);
        if (!filter.exists()) filter.mkdirs();
        String filename = "Imagen.jpg";
        String nameimg = "ImgProcess.jpg";
        File file = new File(filter, filename);
        File fileTwo = new File(filter, nameimg);
        Imgcodecs.imwrite(file.toString(), img);
        Imgcodecs.imwrite(fileTwo.toString(), newFrameImg);

        Log.i(TAG, msg: "Imagenes guardada correctamente: " + file.getAbsolutePath());
        Log.i(TAG, msg: "Imagen Clasificada en: " + as);

        Intent intent = new Intent( packageContext: this, PartOne.class);
        intent.putExtra( name: "imagePath", file.getAbsolutePath()); // Asegúrate de pasar una String
        intent.putExtra( name: "clasificacion", as);
        intent.putExtra( name: "momentsZernike",asZernike);
        intent.putExtra( name: "moments",moments);
        intent.putExtra( name: "imageFilter",fileTwo.getAbsolutePath());
        startActivity(intent);
```

Codigo de C++ de la Parte 1

Procesamiento del corpus de imágenes en el almacenamiento interno del dispositivo.



Función para calcular la distancia Euclidiana

```java
// Calcular la distancia Euclídea
double distanciaEuclidea(const double momentosHu[7], const double meanHu[7]) {
    double suma = 0;
    for (int i = 0; i < 7; i++) {
        suma += ((meanHu[i] - momentosHu[i]) * (meanHu[i] - momentosHu[i]));
    }
    return sqrt( x: suma);
}
```

Función para el procesamiento del corpus con la distancia Euclidiana

```cpp
extern "C" JNIEXPORT jdouble JNICALL
MainActivity.processCorpus(JNIEnv* env, jobject MainActivity , jstring ruta) {
    Mat inputMat, imgGray, mask;
    double distMedia = 0;
    const char *ph = env->GetStringUTFChars( string: ruta,  isCopy: nullptr);
    std::vector<std::string> imagePaths;
    String sd = std::string( s: ph) + "/*.PNG";
    cv::glob( pattern: sd,  &: imagePaths,  recursive: false);
    for (const auto& path : string const & : imagePaths) {
        Mat img = imread( filename: path);
        resize( src: img,  dst: img,  dsize: Size( width: 200,  height: 200));
        cvtColor( src: img,  dst: imgGray,  code: COLOR_BGR2GRAY);
        threshold( src: imgGray,  dst: mask,  thresh: 200,  maxval: 255,  type: THRESH_BINARY);
        bitwise_not( src: mask,  dst: maskInverted);
        Moments momentos = moments( array: maskInverted,  binaryImage: true);
        double momentosHu[7];
        HuMoments( moments: momentos,  hu: momentosHu);
        huMomentsList.push_back({ .hu[0]: momentosHu[0],  .hu[1]: momentosHu[1],  .hu[2]: momentosHu[2],  .hu[3]: momentosHu[3],
                                  .hu[4]: momentosHu[4],  .hu[5]: momentosHu[5],  .hu[6]: momentosHu[6]});
        mome += momentos.m00;
    }
    for (const auto& hu : HuMoment const & : huMomentsList) {
        for (int i = 0; i < 7; ++i) {
            meanHuMoments.hu[i] += hu.hu[i];}
    }
    for (double & i : meanHuMoments.hu) {
        i /= huMomentsList.size();}
    double distanciaEuclidiaTotal = 0;
    for (const auto& hu : HuMoment const & : huMomentsList) {
        distanciaEuclidiaTotal += distanciaEuclidea( momentosHu: hu.hu,  meanHu: meanHuMoments.hu);}
    distMedia = distanciaEuclidiaTotal / huMomentsList.size();
    std::string fullPath( s: ph);
    size_t pos = fullPath.find_last_of( c: '/');
    std::string nombre = (pos == std::string::npos) ? fullPath : fullPath.substr( pos: pos + 1);
    huMomentsMean.push_back(meanHuMoments);
    resultados.push_back({nombre, distMedia});
    huMomentsList.clear();
    meanHuMoments = {};
    return (jdouble)distMedia;
}
```

Funciones para el cálculo de Zernike

```cpp
// Función para preprocesar la imagen y extraer los píxeles no cero
void preprocess_image(const cv::Mat& img, std::vector<double>& X, std::vector<double>& Y, std::vector<double>& P, double& psum, double& moment10,
                      double& moment00, double& moment01) {
    int rows = img.rows;
    int cols = img.cols;

    X.clear();
    Y.clear();
    P.clear();
    psum = 0.0;
    moment10 = 0.0;
    moment00 = 0.0;
    moment01 = 0.0;

    for (int y = 0; y < rows; y++) {
        for (int x = 0; x < cols; x++) {
            double intensity = img.at<uchar>( i0: y,  i1: x);
            if (intensity != 0) {
                Y.push_back(y + 1);
                X.push_back(x + 1);
                P.push_back(intensity);
                psum += intensity;
                moment10 += (x + 1) * intensity;
                moment00 += intensity;
                moment01 += (y + 1) * intensity;
            }
        }
    }
}
```

```cpp
extern "C"
JNIEXPORT jstring JNICALL
Java_com_example_tarea_viewCamara_momentsZernike(JNIEnv *env, jobject,jobject mat) {
    cv::Mat imgGray, mask;
    cv::Mat& zernike = *(cv::Mat*)mat;
    cv::Mat zernikeTest = zernike;
    cv::cvtColor( src: zernikeTest,  dst: zernikeTest,  code: cv::IMREAD_GRAYSCALE);
    std::vector<double> X, Y, P;
    double psum, moment10, moment00, moment01;
    preprocess_image( img: zernikeTest,  &: X,  &: Y,  &: P,  &: psum,  &: moment10,  &: moment00,  &: moment01);
    double D = 15;
    double R = std::min(zernikeTest.rows, zernikeTest.cols) / 2.0;
    double m10_m00 = moment10 / moment00;
    double m01_m00 = moment01 / moment00;
    std::vector<double> zvalues;
    mb_Znl(X, Y, P, D, m10_m00, m01_m00, R, psum,  &: zvalues);
    std::vector<double> first7ZernikeMoments( first: zvalues.begin(),  last: zvalues.begin() + 12);
    for (size_t i = 0; i < first7ZernikeMoments.size(); i++) {
        LOGI("Zernike Moments %f", first7ZernikeMoments[i]);
    }

    cv::String ad = "Zernike 1:(" + std::to_string( val: first7ZernikeMoments[0]) + ")\n"
                + "Zernike 2:(" + std::to_string( val: first7ZernikeMoments[1]) + ")\n"
                + "Zernike 3:(" + std::to_string( val: first7ZernikeMoments[2]) + ")\n"
                + "Zernike 4:(" + std::to_string( val: first7ZernikeMoments[3]) + ")\n"
                + "Zernike 5:(" + std::to_string( val: first7ZernikeMoments[4]) + ")\n"
                + "Zernike 6:(" + std::to_string( val: first7ZernikeMoments[5]) + ")\n"
                + "Zernike 7:(" + std::to_string( val: first7ZernikeMoments[6]) + ")\n"
                + "Zernike 8:(" + std::to_string( val: first7ZernikeMoments[7]) + ")\n"
                + "Zernike 9:(" + std::to_string( val: first7ZernikeMoments[8]) + ")\n"
                + "Zernike 10:(" + std::to_string( val: first7ZernikeMoments[9]) + ")\n"
                + "Zernike 11:(" + std::to_string( val: first7ZernikeMoments[10]) + ")\n"
                + "Zernike 12:(" + std::to_string( val: first7ZernikeMoments[11]) + ")\n";

    return env->NewStringUTF( bytes: ad.c_str());
}
```

```cpp
// Definir función de factorial
double factorial(int n) {
    if (n == 0) return 1.0;
    double result = 1.0;
    for (int i = 1; i <= n; ++i) {
        result *= i;
    }
    return result;
}

void mb_Znl(const std::vector<double>& X, const std::vector<double>& Y, const std::vector<double>& P, double D, double m10_m00,
            double m01_m00, double R, double psum, std::vector<double>& zvalues) {
    static double LUT[MAX_LUT];
    static int n_s[MAX_Z], l_s[MAX_Z];
    static char init_lut = 0;

    double x, y, p;
    int i, m, theZ, theLUT, numZ = 0;
    int n = 0, l = 0;

    std::vector<std::complex<double>> sum( n: MAX_Z);
    std::complex<double> Vnl;

    assert(D == MAX_D);

    if (!init_lut) {
        theZ = 0;
        theLUT = 0;
```

```
if (!init_lut) {
    theZ = 0;
    theLUT = 0;
    for (n = 0; n <= MAX_D; n++) {
        for (l = 0; l <= n; l++) {
            if ((n - l) % 2 == 0) {
                for (m = 0; m <= (n - l) / 2; m++) {
                    LUT[theLUT] = pow( lcpp_x: -1.0,  lcpp_y: m) * (factorial( n: n - m) / (factorial( n: m) * factorial( n: (n - 2 * m + l) / 2) *
                            factorial( n: (n - 2 * m - l) / 2)));
                    theLUT++;
                }
                n_s[theZ] = n;
                l_s[theZ] = l;
                theZ++;
            }
        }
    }
    init_lut = 1;
}

for (n = 0; n <= D; n++) {
    for (l = 0; l <= n; l++) {
        if ((n - l) % 2 == 0) {
            sum[numZ] = std::complex<double>( re: 0.0,  im: 0.0);
            numZ++;
        }
    }
}
```

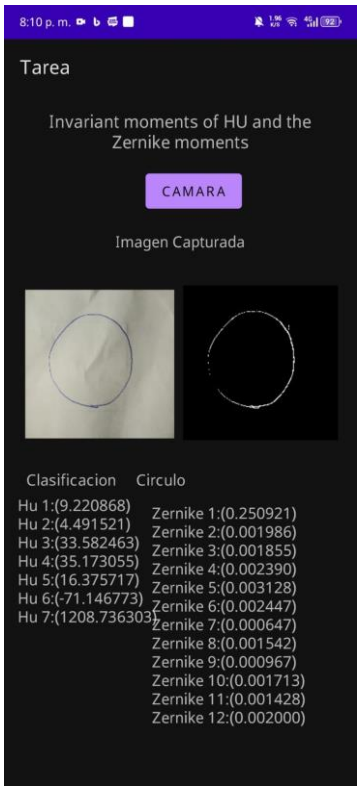
```
    for (i = 0; i < X.size(); i++) {
        x = (X[i] - m10_m00) / R;
        y = (Y[i] - m01_m00) / R;
        double sqr_x2y2 = sqrt( x: x * x + y * y);
        if (sqr_x2y2 > 1.0) continue;

        p = P[i] / psum;
        double atan2yx = atan2(y, x);
        theLUT = 0;
        for (theZ = 0; theZ < numZ; theZ++) {
            n = n_s[theZ];
            l = l_s[theZ];
            Vnl = std::complex<double>( re: 0.0,  im: 0.0);
            for (m = 0; m <= (n - l) / 2; m++) {
                Vnl += (std::polar( rho: 1.0,  theta: l * atan2yx) * LUT[theLUT] * pow( lcpp_x: sqr_x2y2,  lcpp_y: (n - 2 * m)));
                theLUT++;
            }
            sum[theZ] += std::conj( c: Vnl) * p;
        }
    }

    zvalues.resize( sz: numZ);
    for (theZ = 0; theZ < numZ; theZ++) {
        sum[theZ] *= ((n_s[theZ] + 1) / PI);
        zvalues[theZ] = std::abs( c: sum[theZ]);
    }
}
```

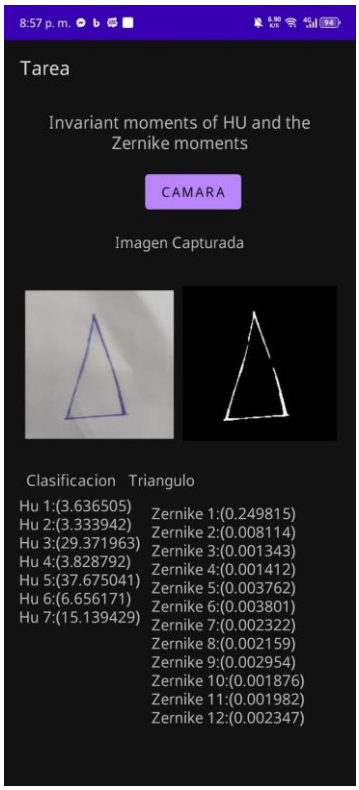
```
#include ...

void mb_Znl(const std::vector<double>& X, const std::vector<double>& Y, const std::vector<double>& P, double D, double m10_m00, double m01_m00,
            double R, double psum, std::vector<double>& zvalues);


#endif //TAREA_ZERNIKE_H
```

Clasificación para una nueva imagen procesándola y clasificándola.

```cpp
extern "C" JNIEXPORT jstring JNICALL
Java_com_example_tarea_viewCamara_Classification(JNIEnv* env, jobject, jobject mat) {
    Mat imgGray, mask;
    Mat& inputMat = *(Mat*)mat;

    resize( src: inputMat,  dst: inputMat,  dsize: Size( width: 200,  height: 200));
    rotate( src: inputMat,  dst: inputMat,  rotateCode: ROTATE_90_CLOCKWISE);
    cv::cvtColor( src: inputMat,  dst: imgGray,  code: COLOR_BGR2GRAY);
    threshold( src: imgGray,  dst: mask,  thresh: 120,  maxval: 255,  type: THRESH_BINARY);
    bitwise_not( src: mask,  dst: maskInverted);

    Moments mo = moments( array: maskInverted,  binaryImage: true);
    double momet[7];
    HuMoments( moments: mo,  hu: momet);

    for (int i = 0; i < 7; ++i) {
        momentResult[i] = momet[i];}

    std::vector<double> disEuList;
    for (const auto& huMoment :HuMoment const & : huMomentsMean) {
        double result = distanciaEuclidea( momentosHu: momet,  meanHu: huMoment.hu);
        LOGI("El resultado euclidiano es: %f", result);
        disEuList.push_back(result);}

    // Encuentra la menor diferencia
    double minDiferencia = std::numeric_limits<double>::max();
    std::string figura;
    for (size_t i = 0; i < disEuList.size(); ++i) {
        double diferencia = std::abs( lcpp_x: disEuList[i] - resultados[i].distMedia);
        if (diferencia < minDiferencia) {
            minDiferencia = diferencia;
            figura = resultados[i].nombre;}
    }

    LOGI("La figura clasificada es: %s", figura.c_str());
    String name = figura.c_str();
    return env->NewStringUTF( bytes: figura.c_str());
}
```

Resultados con nuevas imágenes desde la aplicación.





Repositorio de GitHub

https://github.com/anthonypulla126/Task-3.1

Parte 2. Desarrollar una aplicación móvil que permita clasificar una región de interés de una imagen, dada su textura. Para ello deberá tomar en cuenta lo siguiente (ver Ilustración 2):

1. Programar un método que permita convertir una imagen de un espacio de color en el espacio CIELab.

2. Programar un método que dada una imagen o región de interés permita calcular el descriptor LBP. Con ello, deberá almacenar el histograma en un archivo o base de datos.

3. Deberá calcular el descriptor LBP para al menos 10 imágenes distintas de dos tipos de textura: clase 1 y clase 2.

4. Cuando se le presente al dispositivo móvil una imagen de textura, deberá usar su clasificador basado en el descriptor LBP para identificar a qué clase pertenece la imagen (clase 1 o clase 2).

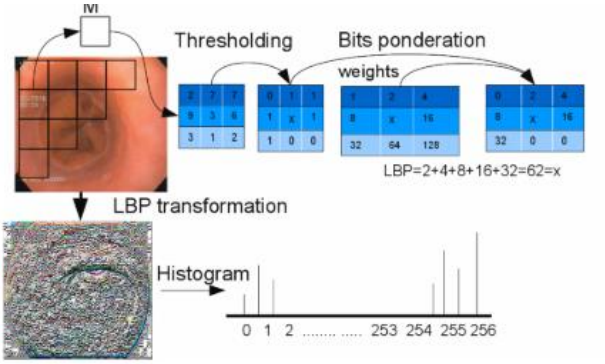


Ilustración 2. Ejemplo de clasificación de una zona del esófago donde existe hernial hiatal. Fuente: https://ieeexplore.ieee.org/document/7036342

Código de LBP.hpp:

```
#ifndef PROJECT_LBP_H

#define PROJECT_LBP_H


#include <opencv2/opencv.hpp>

#include <vector>


class LBP {

public:

    LBP();

    int* LBP8(const int* data, int rows, int columns);

    std::vector<int> calcularLBP(cv::Mat imagen);
```

```
    cv::Mat calcularLBPImage(cv::Mat imagen);
private:
    inline void compab_mask_inc(const int* p, int bit, unsigned int &value, int &cntr);
};
#endif // LBP_HPP
```

Codigo del LBP.cpp (Ming-Kuei Hu, 1962):

```
#include <opencv2/opencv.hpp>
#include <vector>
#include <iostream>
#include <cstring>
#include "LBP.h"
#include <jni.h>
#include <string>
using namespace cv;
using namespace std;

LBP::LBP(){

}

inline void LBP::compab_mask_inc(const int* p, int bit, unsigned int &value, int &cntr)
{
    value |= (*p >= cntr) << bit;
}

int* LBP::LBP8(const int* data, int rows, int columns){
    const int
            *p0 = data,
            *p1 = p0 + 1,
            *p2 = p1 + 1,
```

```
        *p3 = p2 + columns,
        *p4 = p3 + columns,
        *p5 = p4 - 1,
        *p6 = p5 - 1,
        *p7 = p6 - columns,
        *center = p7 + 1;
int r, c, cntr;
unsigned int value;
int* result = (int*)malloc(256*sizeof(int));
memset(result, 0, 256*sizeof(int));
for (r = 0; r < rows - 2; r++) {
    for (c = 0; c < columns - 2; c++) {
        value = 0;
        cntr = *center - 1;
        compab_mask_inc(p0, 0, value, cntr);
        compab_mask_inc(p1, 1, value, cntr);
        compab_mask_inc(p2, 2, value, cntr);
        compab_mask_inc(p3, 3, value, cntr);
        compab_mask_inc(p4, 4, value, cntr);
        compab_mask_inc(p5, 5, value, cntr);
        compab_mask_inc(p6, 6, value, cntr);
        compab_mask_inc(p7, 7, value, cntr);
        center++;
        result[value]++;
        p0++;
        p1++;
        p2++;
        p3++;
        p4++;
        p5++;
```

```
            p6++;
            p7++;
        }
        p0 += 2;
        p1 += 2;
        p2 += 2;
        p3 += 2;
        p4 += 2;
        p5 += 2;
        p6 += 2;
        p7 += 2;
        center += 2;
    }
    return result;
}

vector<int> LBP::calcularLBP(Mat imagen) {
    int *datos = (int *)malloc(imagen.rows * imagen.cols * sizeof(int));
    for (int i = 0, k = 0; i < imagen.rows; i++) {
        for (int j = 0; j < imagen.cols; j++) {
            datos[k++] = imagen.at<uchar>(i, j);
        }
    }
    int *res = this->LBP8(datos, imagen.rows, imagen.cols);
    vector<int> histo(256, 0);
    for (int i = 0; i < 256; i++) {
        histo[i] = res[i];
        cout << res[i] << "||";
    }
    cout << endl;
```

```
    free(datos);
    free(res);
    return histo;
}

Mat LBP::calcularLBPImage(Mat imagen) {
    Mat lbpImage = Mat::zeros(imagen.rows - 2, imagen.cols - 2, CV_8UC1);
    int *datos = (int *)malloc(imagen.rows * imagen.cols * sizeof(int));
    for (int i = 0, k = 0; i < imagen.rows; i++) {
        for (int j = 0; j < imagen.cols; j++) {
            datos[k++] = imagen.at<uchar>(i, j);
        }
    }
    const int* p0 = datos;
    const int* p1 = p0 + 1;
    const int* p2 = p1 + 1;
    const int* p3 = p2 + imagen.cols;
    const int* p4 = p3 + imagen.cols;
    const int* p5 = p4 - 1;
    const int* p6 = p5 - 1;
    const int* p7 = p6 - imagen.cols;
    const int* center = p7 + 1;

    for (int r = 0; r < imagen.rows - 2; r++) {
        for (int c = 0; c < imagen.cols - 2; c++) {
            unsigned int value = 0;
            int cntr = *center - 1;
            compab_mask_inc(p0, 0, value, cntr);
            compab_mask_inc(p1, 1, value, cntr);
            compab_mask_inc(p2, 2, value, cntr);
```

```
            compab_mask_inc(p3, 3, value, cntr);
            compab_mask_inc(p4, 4, value, cntr);
            compab_mask_inc(p5, 5, value, cntr);
            compab_mask_inc(p6, 6, value, cntr);
            compab_mask_inc(p7, 7, value, cntr);
            lbpImage.at<uchar>(r, c) = value;
            p0++;
            p1++;
            p2++;
            p3++;
            p4++;
            p5++;
            p6++;
            p7++;
            center++;
        }
        p0 += 2;
        p1 += 2;
        p2 += 2;
        p3 += 2;
        p4 += 2;
        p5 += 2;
        p6 += 2;
        p7 += 2;
        center += 2;
    }
    free(datos);
    return lbpImage;
}
```

Código de Part-Two.cpp:

```
#include <opencv2/opencv.hpp>
#include "LBP.h"
#include <android/log.h>
#include <iostream>
#include <fstream>  // Asegúrate de incluir esta biblioteca
#include <jni.h>
#include <string>
using namespace std;
using namespace cv;

#define LOG_TAG "Part-Two"
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO, LOG_TAG, __VA_ARGS__)

std::vector<vector<int>> clases;

Mat resultHistogram;
Mat resultLBP;

void saveLBPHistogram(vector<int> histo, const string& filename) {

    std::ofstream archivo(filename);
    if (archivo.is_open()) {
        for (int val : histo) {
            archivo << val << " ";
        }
        archivo.close();
    }
}
```

```
vector<int> averageHistograms(vector<string> files) {
    vector<int> avgHistogram(256, 0);
    int numFiles = files.size();
    LOGI("El tamaño del files es: %lu", files.size());

    for (const string& file : files) {
        ifstream infile(file);
        int val;
        for (int i = 0; i < 256; i++) {
            infile >> val;
            avgHistogram[i] += val;
        }
        infile.close();
    }
    for (int i = 0; i < 256; i++) {
        avgHistogram[i] /= numFiles;
    }
    return avgHistogram;
}

void matHistograma(const vector<int>& histo) {
    int histSize = 256;
    int hist_w = 512; int hist_h = 400;
    int bin_w = cvRound((double)hist_w / histSize);
    Mat histImage(hist_h, hist_w, CV_8UC1, Scalar(0, 0, 0));

    for (int i = 1; i < histSize; i++) {
        line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(histo[i - 1])),
             Point(bin_w*(i), hist_h - cvRound(histo[i])),
             Scalar(255, 255, 255), 2, 8, 0);
```

```
    }

    resultHistogram = histImage;
}

extern "C"
JNIEXPORT jstring JNICALL
Java_com_example_tarea_MainActivity_processImg(JNIEnv *env, jobject, jstring ruta)
{
    LBP* lbp = new LBP();
    Mat inputMat, imgGray, mask;

    const char *ph = env->GetStringUTFChars(ruta, nullptr);
    std::vector<std::string> imagePaths;
    std::vector<std::string> imgHisto;

    String sd = std::string(ph) + "/*.jpg";
    cv::glob(sd, imagePaths, false);
    for (const auto& path : imagePaths) {
        Mat img = imread(path, IMREAD_GRAYSCALE);
        vector<int> histo = lbp->calcularLBP(img);
        saveLBPHistogram(histo, path + "_histogram.txt");
    }

    String his = std::string(ph) + "/*.txt";
    cv::glob(his, imgHisto, false);
    // Promediar histogramas de entrenamiento
    vector<int> resultHistory = averageHistograms(imgHisto);
    clases.push_back(resultHistory);
```

```
    LOGI("El tamaño de clases es: %lu", clases.size());
    return env->NewStringUTF("Procesamiento de imagen exitoso.");
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_tarea_viewCamara_ClassificationMaterial(JNIEnv* env, jobject,
jobject mat) {
    LBP* lbp = new LBP();
    Mat imgGray, mask;
    Mat& textura = *(Mat*)mat;
    Mat tex = textura;

    std::string name;

    // Convert input image to grayscale
    cv::cvtColor(tex, tex, COLOR_BGR2GRAY);

    // Calculate LBP image and histogram
    Mat lbpImage = lbp->calcularLBPImage(tex);
    vector<int> textureHistogram = lbp->calcularLBP(tex);

    resultLBP = lbpImage;

    // Generate histogram image
    matHistograma(textureHistogram);

    // Check if classes vector has at least two classes
    if (clases.size() < 2) {
        LOGI("Not enough classes for comparison");
        return env->NewStringUTF("Not enough classes for comparison");
    }
```

```
    // Compare histograms and classify
    double distanciaClass1 = norm(clases[0], textureHistogram, NORM_L2);
    double distanciaClass2 = norm(clases[1], textureHistogram, NORM_L2);
    LOGI("Distancia a Clase 1 (Madera): %f", distanciaClass1);
    LOGI("Distancia a Clase 2 (Tela de ropa): %f", distanciaClass2);

    if (distanciaClass1 < distanciaClass2) {
        name = "Clase 1 (Madera)";
    } else {
        name = "Clase 2 (Tela de ropa)";
    }

    return env->NewStringUTF(name.c_str());
}

extern "C" JNIEXPORT jlong JNICALL
Java_com_example_tarea_viewCamara_matLBP(JNIEnv* env, jobject) {
    return (jlong)(new Mat(resultLBP));
}

extern "C" JNIEXPORT jlong JNICALL
Java_com_example_tarea_viewCamara_matHistogram(JNIEnv* env, jobject) {
    return (jlong)(new Mat(resultHistogram));
}
```

Código de PartTwo.java:

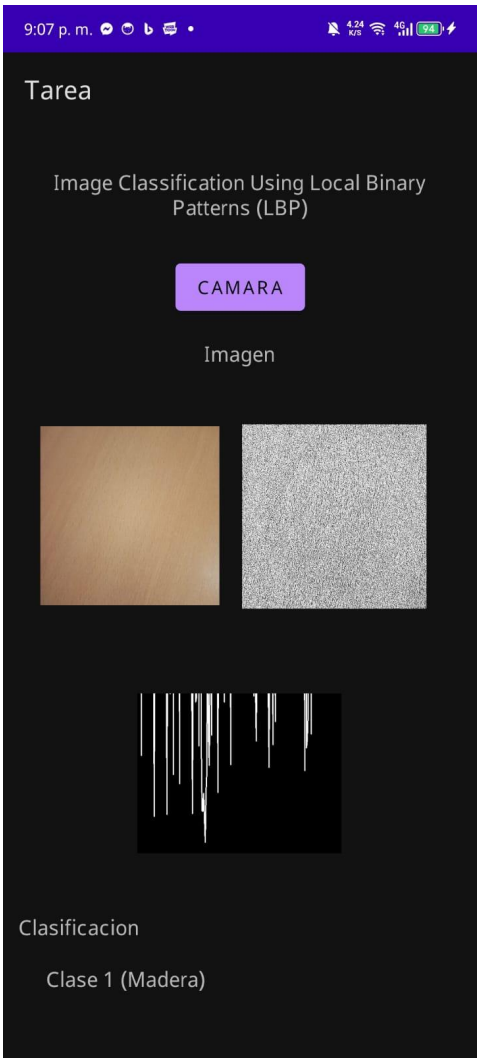
```
package com.example.tarea;

import androidx.appcompat.app.AppCompatActivity;
```

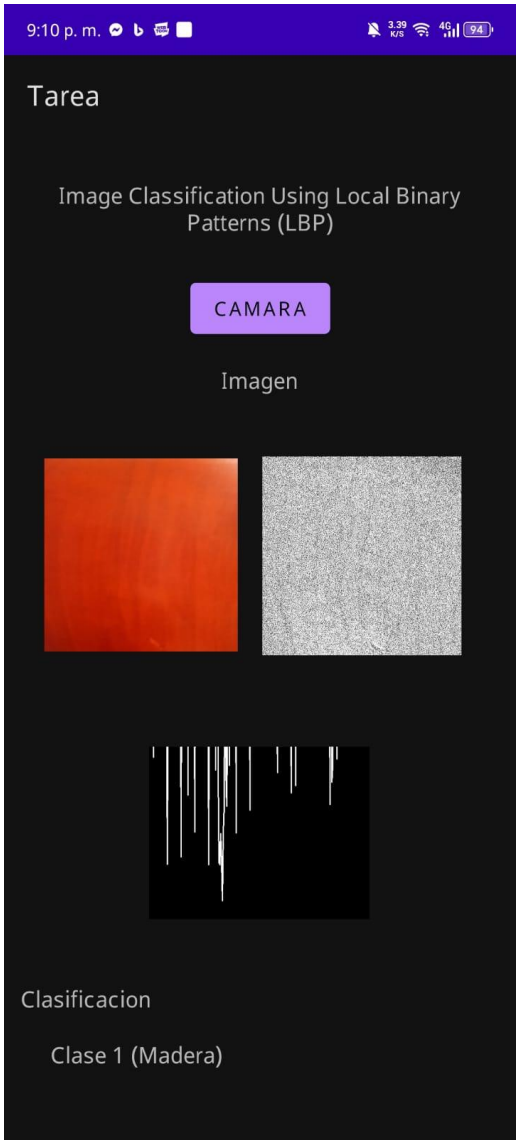
```
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

import com.example.tarea.databinding.ActivityMainBinding;

public class PartTwo extends AppCompatActivity {

    Button btn;
    private TextView classificationTextView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ActivityMainBinding binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        setContentView(R.layout.activity_part_two);
        btn= findViewById(R.id.button3);
        btn.setOnClickListener(v -> {
            Intent intent = new Intent(PartTwo.this, viewCamara.class);
            intent.putExtra("param_key", "PartTwo");
            startActivity(intent);
        });
```

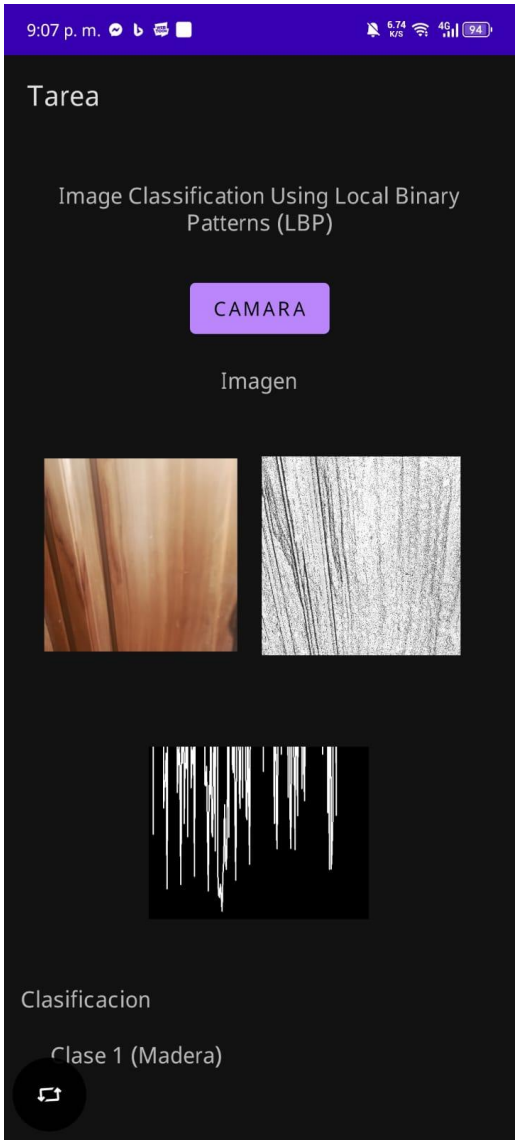
```
    }

    @Override
    protected void onResume() {
        super.onResume();
        Intent intent = getIntent();
        String imagePath = intent.getStringExtra("imagePath");
        String imageLBP = intent.getStringExtra("imageLBP");
        String imageHistogram = intent.getStringExtra("imageHistogram");
        String result = intent.getStringExtra("clasificacion");

        if (imagePath != null) {
            ImageView imageView = findViewById(R.id.image_view);
            Bitmap bitmap = BitmapFactory.decodeFile(imagePath);
            imageView.setImageBitmap(bitmap);

            ImageView imgLBP = findViewById(R.id.image_view3);
            Bitmap bit = BitmapFactory.decodeFile(imageLBP);
            imgLBP.setImageBitmap(bit);

            ImageView imgHistogram = findViewById(R.id.image_view4);
            Bitmap bi = BitmapFactory.decodeFile(imageHistogram);
            imgHistogram.setImageBitmap(bi);

            classificationTextView = findViewById(R.id.textView4);
            classificationTextView.setText(result);
        }
    }
}
```

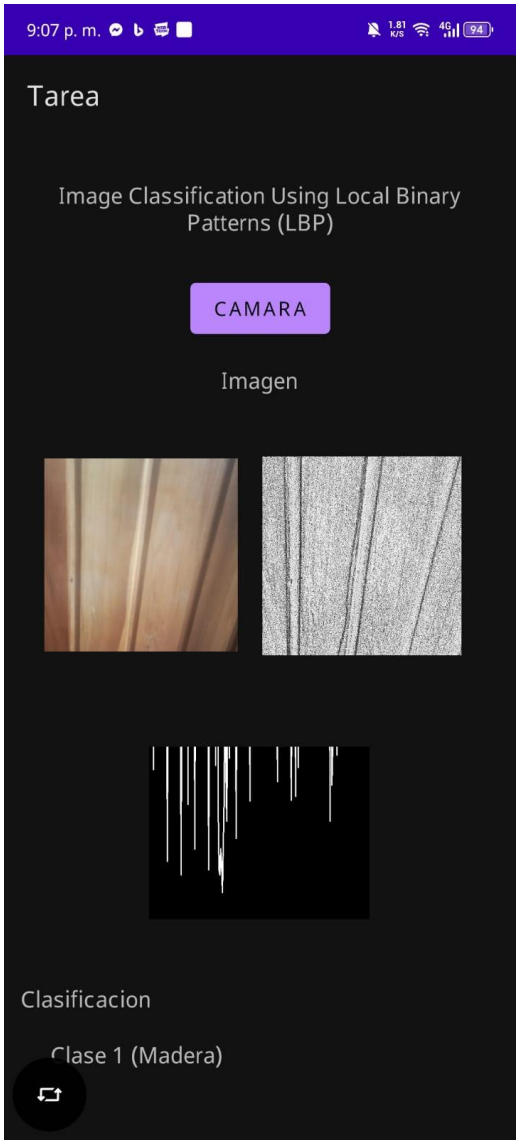
Resultados en el Dispositivo Móvil de las clases de textura donde clase 1 es madera y clase 2 es tela de ropa comprobada sobre 10 imágenes distintas utilizando LBP:

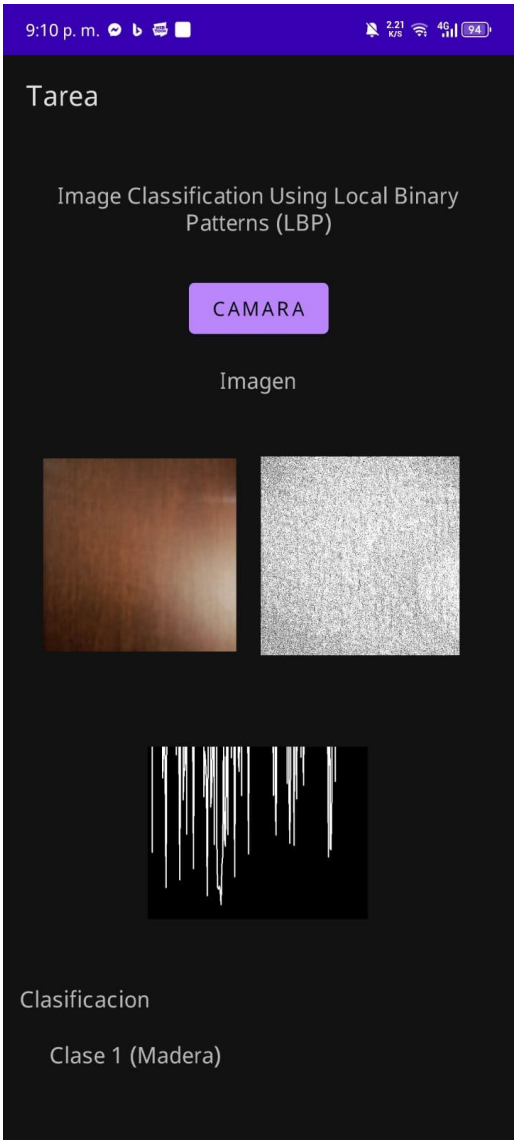
CLASE 1 (MADERA):



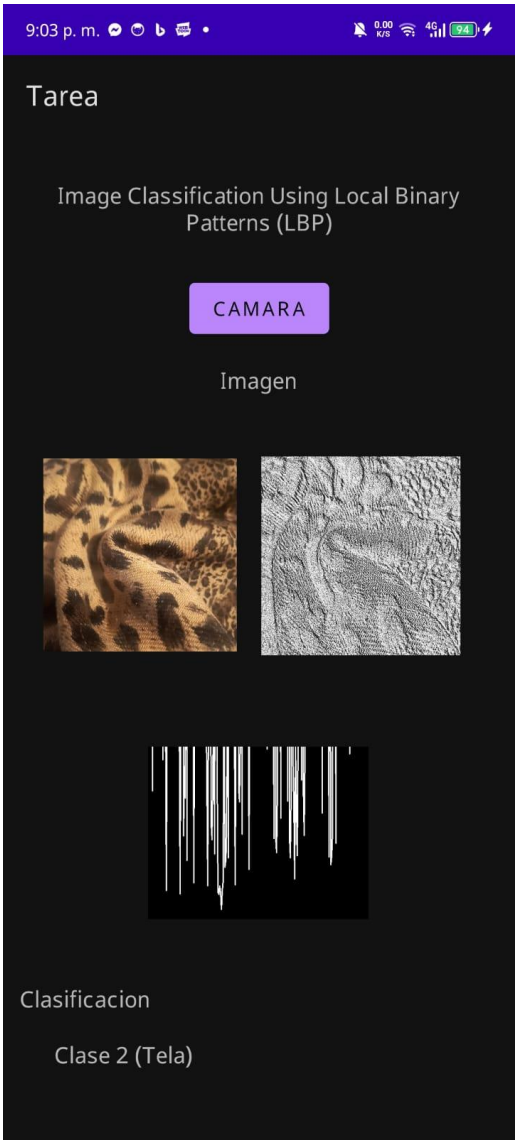

9:10 p. m.
Tarea
Image Classification Using Local Binary Patterns (LBP)
CAMARA
Imagen
Clasificacion
Clase 1 (Madera)

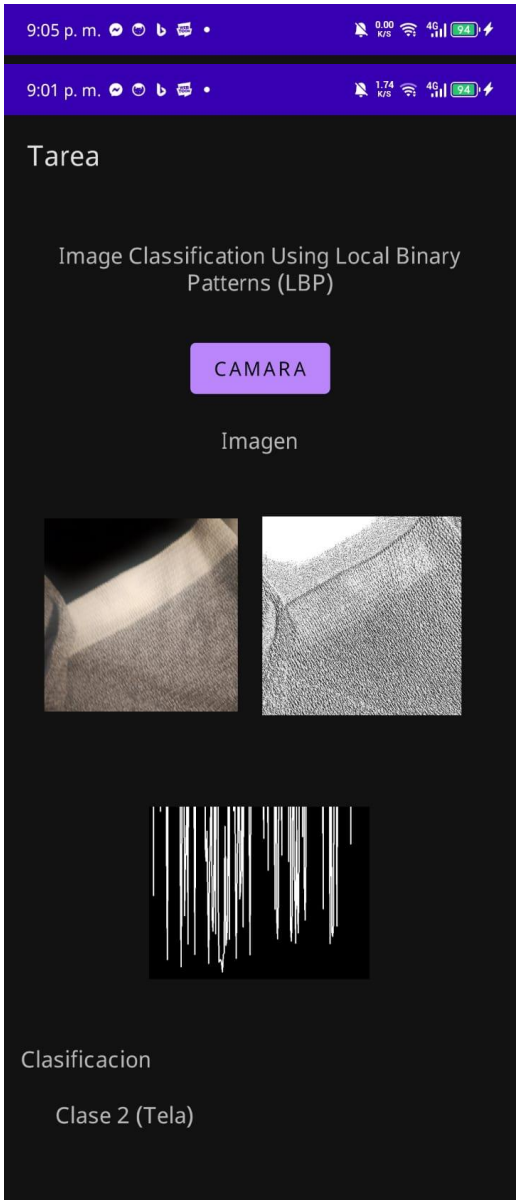

9:07 p. m.
Tarea
Image Classification Using Local Binary Patterns (LBP)
CAMARA
Imagen
Clasificacion
Clase 1 (Madera)

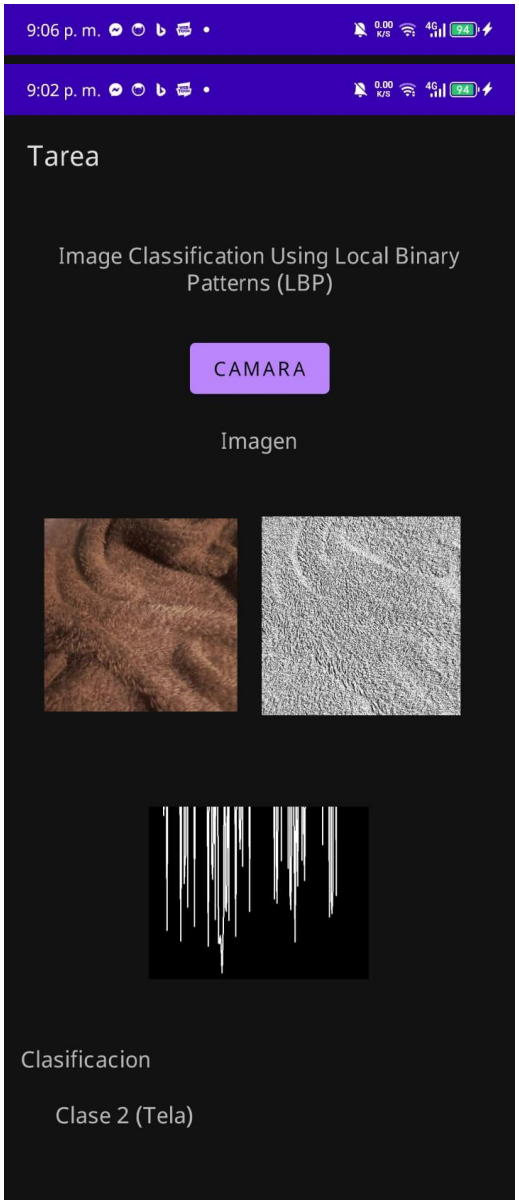

9:07 p. m.
Tarea
Image Classification Using Local Binary Patterns (LBP)
CAMARA
Imagen
Clasificacion
Clase 1 (Madera)


9:10 p. m.
Tarea
Image Classification Using Local Binary Patterns (LBP)
CAMARA
Imagen
Clasificacion
Clase 1 (Madera)

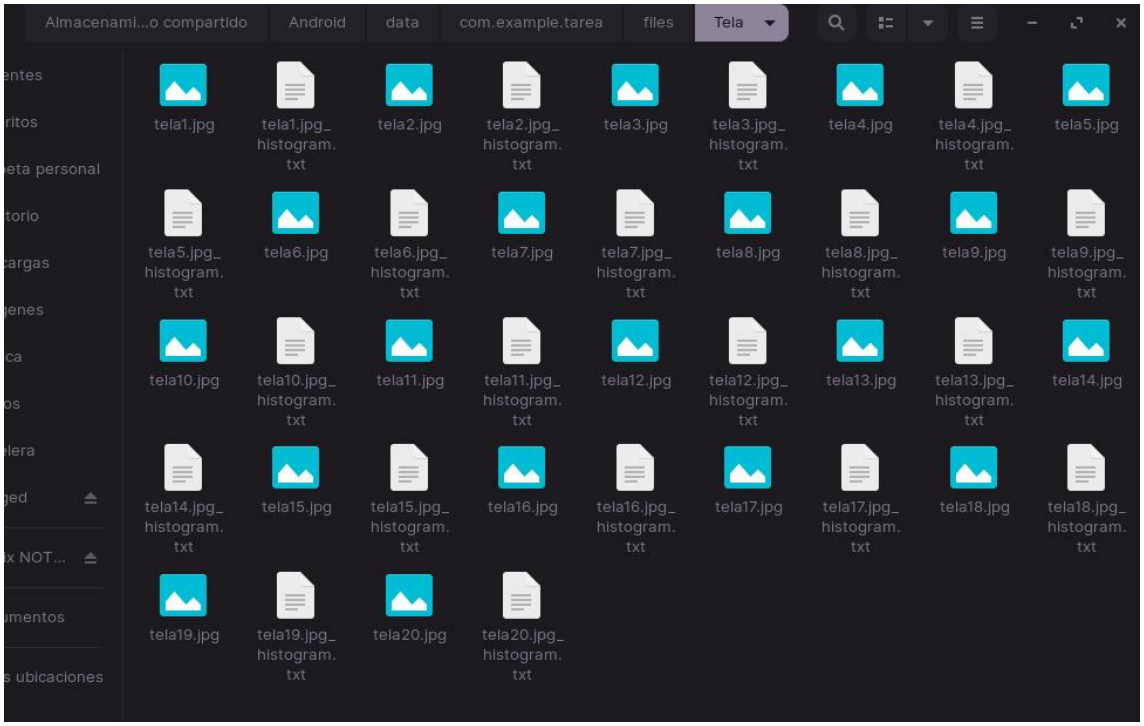
CLASE 2 (TELA):

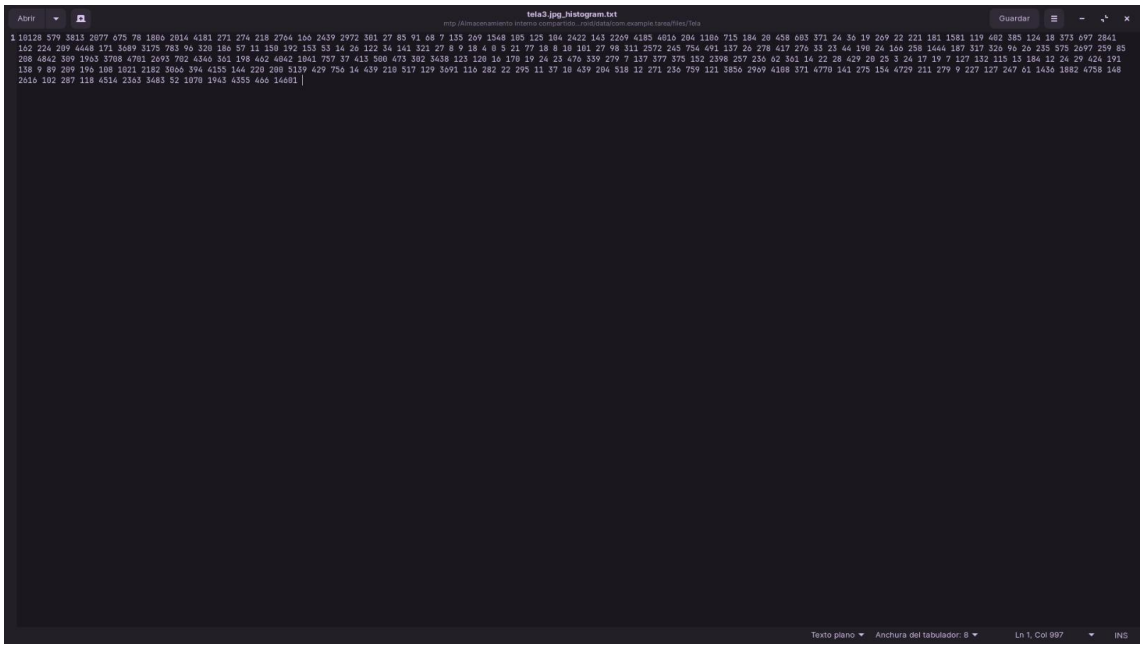

9:03 p. m.
Tarea
Image Classification Using Local Binary Patterns (LBP)
CAMARA
Imagen
Clasificacion
Clase 2 (Tela)





Demostración del histograma de las primeras imágenes (en este caso), se guardan en un archivo de .txt dentro de la misma carpeta del proyecto:


Almacenami...o compartido
Android
data
com.example.tarea
files
Tela
tela1.jpg
tela1.jpg_histogram.txt
tela2.jpg
tela2.jpg_histogram.txt
tela3.jpg
tela3.jpg_histogram.txt
tela4.jpg
tela4.jpg_histogram.txt
tela5.jpg
tela5.jpg_histogram.txt
tela6.jpg
tela6.jpg_histogram.txt
tela7.jpg
tela7.jpg_histogram.txt
tela8.jpg
tela8.jpg_histogram.txt
tela9.jpg
tela9.jpg_histogram.txt
tela10.jpg
tela10.jpg_histogram.txt
tela11.jpg
tela11.jpg_histogram.txt
tela12.jpg
tela12.jpg_histogram.txt
tela13.jpg
tela13.jpg_histogram.txt
tela14.jpg
tela14.jpg_histogram.txt
tela15.jpg
tela15.jpg_histogram.txt
tela16.jpg
tela16.jpg_histogram.txt
tela17.jpg
tela17.jpg_histogram.txt
tela18.jpg
tela18.jpg_histogram.txt
tela19.jpg
tela19.jpg_histogram.txt
tela20.jpg
tela20.jpg_histogram.txt


Abrir
tela3.jpg_histogram.txt
Guardar
1 10128 579 3813 2077 675 78 1886 2014 4181 271 274 218 2764 166 2439 2972 301 27 85 91 68 7 135 269 1548 105 125 184 2422 143 2269 4185 4016 204 1186 715 184 20 458 603 371 24 36 19 269 22 221 181 1581 119 402 385 124 18 373 697 2841
Texto plano
Anchura del tabulador: 8
Ln 1, Col 997
INS

Link de acceso al Video de Demostración del funcionamiento correcto de las Partes 1 y 2 de la Práctica: https://estliveupsedu-my.sharepoint.com/:v:/g/personal/kpaltin_est_ups_edu_ec/ERt0xWuTb_1Arqyr2OXfQSoBKlpJhWd4WSClRioidulkgg?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=5tGW2J

Repositorio de GitHub del Código Completo de la Práctica:

https://github.com/anthonypulla126/Task-3.1

Referencias:

- Allan, C. (2013). *Src/textures/zernike/zernike.Cpp at master · chris-allan/pychrm*. Recuperado de https://github.com/chris-allan/pychrm/blob/master/src/textures/zernike/zernike.cpp
- Boland, M. V. (1999). zernike.cpp. Recuperado de Cmu.edu website: https://murphylab.web.cmu.edu/publications/boland/boland_node85.html
- Ming-Kuei Hu, "Visual pattern recognition by moment invariants," in IRE Transactions on Information Theory, vol. 8, no. 2, pp. 179-187, February 1962, doi: 10.1109/TIT.1962.1057692.