



STUDENTS:

ANTHONY PULLA

KELLY PALTIN

COURSE:

COMPUTER VISION

TOPIC:

**FINAL INTEGRATIVE PROJECT - DEVELOPMENT OF A COMPUTER VISION
SYSTEM TO DETECT/CLASSIFY OBJECTS EMPLOYING DEEP LEARNING
AND FEATURE EXTRACTION.**

PROFESSOR:

ENG. VLADIMIR ROBLES

DATE:

JULY 31, 2024

ACADEMIC PERIOD:

64

FINAL INTEGRATIVE PROJECT

Introduction

In recent years, technology has advanced rapidly, especially in the field of computer vision. This area focuses on analyzing and understanding visual information and includes the study of biometrics, which uses physical or behavioral traits to identify individuals.

Currently, identifying people quickly and accurately is challenging because it requires time for a person or machine to verify their identity. Therefore, there is a need for reliable and automated facial detection systems that can help track people involved in crimes or locate missing persons. This research proposes methods to improve video quality through preprocessing techniques, which enhance lighting and resolution. This makes face detection more accurate, reduces false positives, and minimizes errors.

However, recognizing faces automatically is difficult due to changes in location, size, orientation, facial expressions, lighting, and other factors. The groundwork laid by Jones and Viola in 2001 with their object detection framework has been influential, even though their Viola-Jones algorithm is now considered outdated. It is still widely used for real-time face detection and has led to further advancements in the field. (Fernández, F., & Mariel, K., 2016)

Besides facial recognition, recognizing characters from images is another challenging task due to high variation and clutter. Histograms of Oriented Gradients (HOG) descriptors have been very effective in this area. By combining HOG descriptors at different scales, researchers have significantly improved the performance of character recognition tasks. (A. J. Newell & L. D. Griffin, 2011)

This essay explores the development of a computer vision system to detect and classify objects using deep learning and feature extraction techniques. It focuses on facial recognition using the Haar cascade method and making predictions from the MNIST dataset using HOG descriptors and a Multilayer Perceptron (MLP) neural network. Combining these technologies, in this project we aim to improve the accuracy and efficiency of detecting and classifying objects in various applications.

Problem Description

Part II (Final):

- Based on the application developed in Part I, you must incorporate two new functionalities listed below:

- **Face detection:**

- You must program a code that allows to perform face detection on the mobile device (C++ library) using Haar Cascades or a similar approach, so that it can detect not only the area where the face is, but also eyes, nose and mouth, as shown in Illustration 1.

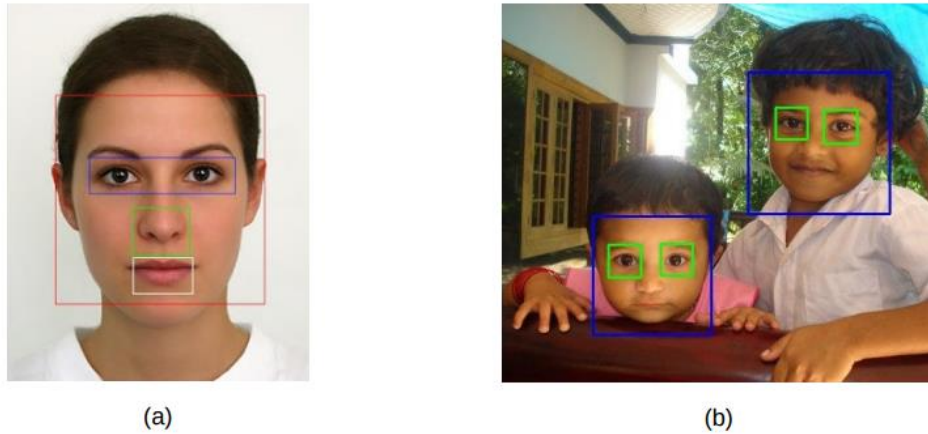


Illustration 1: Examples of the application of the Haar Cascades algorithm for face detection. Sources: Reddy, S. (2010) & Jaiswal, A. (2022).

- Once the face is detected, the captured images or video with the face, eyes, nose and mouth positions should be sent to the web server.

- The web server will have to apply an effect on the parts of the face, for example: place lenses or glasses, change the color of the eyes, etc. To do this, you must define what effect you want to apply and design how the effect will be implemented

▪ **Object classification using Histograms of Oriented Gradients (HOG):**

- An object classifier should be trained using the HOG descriptor and a Multilayer Perceptron Neural Network or similar.

- To do this, you must select a corpus of images on which the classification will be performed, extract the HOG descriptor and then train the neural network, as shown in Illustration 2, where the HOG is used to classify handwritten digits (MNIST corpus).

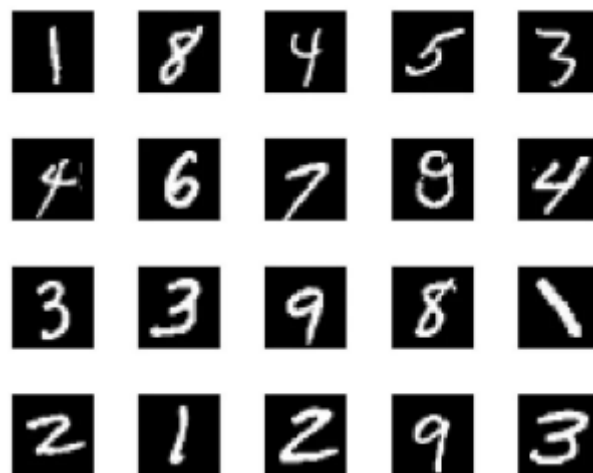
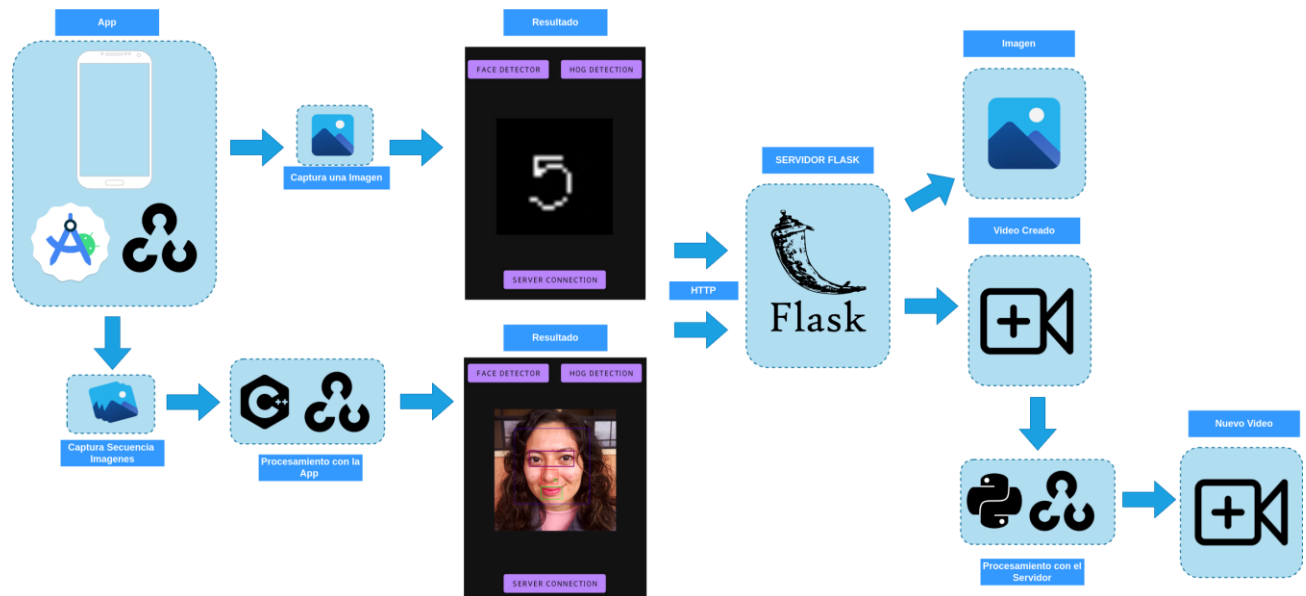


Illustration 2: Example of images from the MNIST corpus with which digit classification is performed using HOG. Source: Histogram of oriented gradients -- MNIST/MNIST with HOG features.Ipynb at master · pransen/ComputerVisionAlgorithms. (2020).

- Once the neural network is trained, it must implement the classifier in the web server, so that when a new image is loaded, the prediction is performed.

Proposed Solution

To face the proposed challenges, we started based on this explanatory diagram which outlines the main process that will be found in our Application, we will explain the code of all the processes made in further detail in the following report.



▪ Face detection using Haar Cascades:

Now in order to address the face detection part on our application, we will explain in detail how the code detects and marks facial features like the face, eyes, nose, and mouth in a video. The code uses the OpenCV library for image processing and is integrated with an Android application through JNI (Java Native Interface).

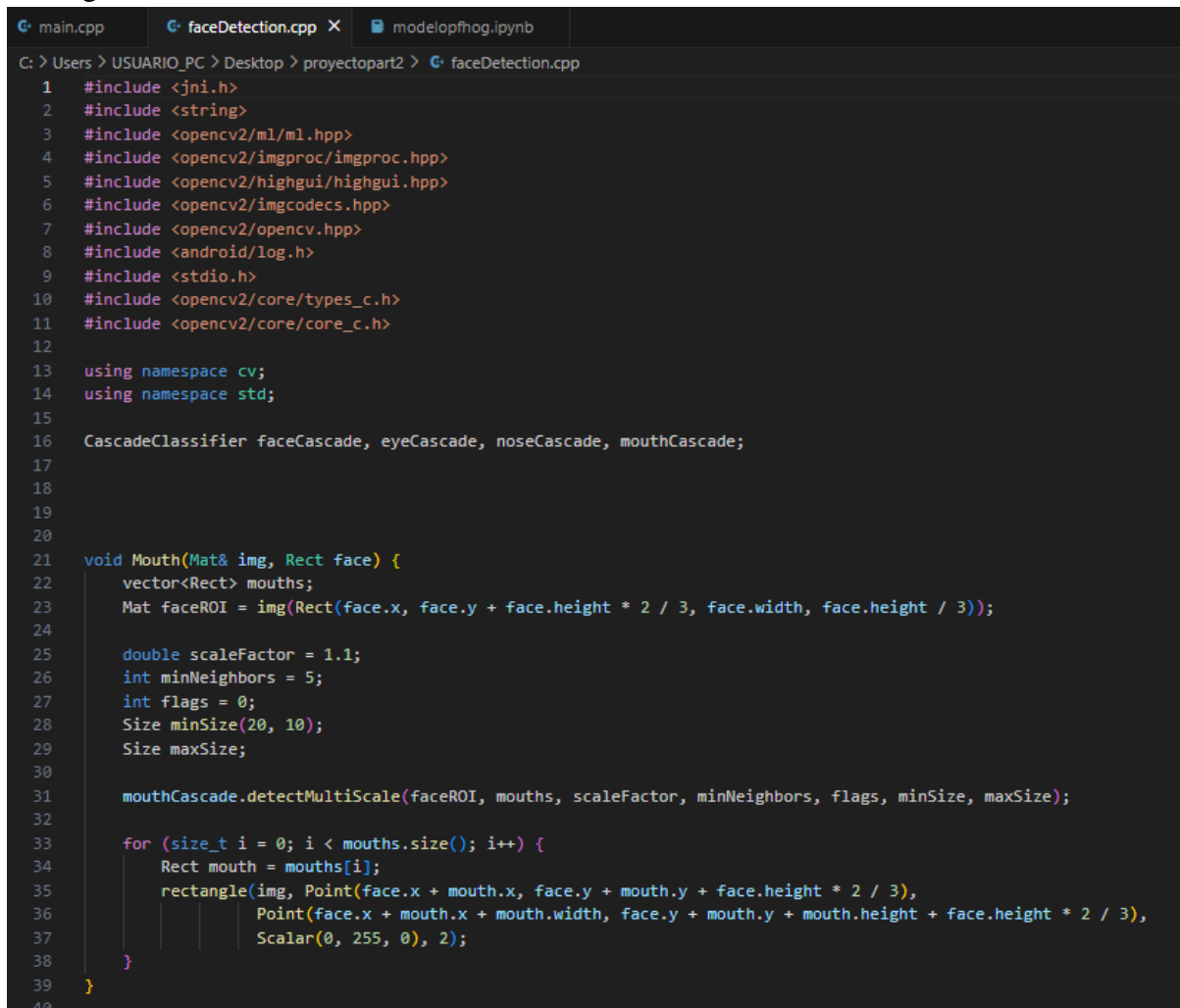
1. Setting Up Cascades and Loading the Cascade Files

The code initializes several cascade classifiers, which are pre-trained models used for detecting objects like faces and facial features. The function `Java_com_example_project_FaceDetector_nativeInitCascades` is a JNI method that connects the Java code with native C++ code. This function is called from Java to initialize the cascade classifiers. The cascade file paths are passed as `jstring` from Java. These are converted to C-style strings using `GetStringUTFChars` method, which allows C++ to read the string data.

The `faceCascade.load`, `eyeCascade.load`, `noseCascade.load`, and `mouthCascade.load` methods are called with their respective file paths. These methods load the pre-trained Haar cascade models into the program, making them ready for detecting specific features. After loading the cascades, the `ReleaseStringUTFChars` method is called to free the memory allocated for the C-style strings.

2. Detecting the Mouth

The Mouth function defines a region of interest (ROI) in the lower third of the face where the mouth is likely to be. This ROI is calculated as `Rect(face.x, face.y + face.height * 2 / 3, face.width, face.height / 3)`. The `mouthCascade.detectMultiScale` method is used to detect the mouth within the ROI. This method scans the ROI at multiple scales and returns a list of rectangles representing the detected mouths. Rectangles are drawn around detected mouths, adjusting the positions relative to the face coordinates. The rectangles are drawn using the `rectangle` function.



```
main.cpp | faceDetection.cpp X | modelopfhog.ipynb
C: > Users > USUARIO_PC > Desktop > proyectopart2 > faceDetection.cpp
1  #include <jni.h>
2  #include <string>
3  #include <opencv2/ml/ml.hpp>
4  #include <opencv2/imgproc/imgproc.hpp>
5  #include <opencv2/highgui/highgui.hpp>
6  #include <opencv2/imgcodecs.hpp>
7  #include <opencv2/opencv.hpp>
8  #include <android/log.h>
9  #include <stdio.h>
10 #include <opencv2/core/types_c.h>
11 #include <opencv2/core/core_c.h>
12
13 using namespace cv;
14 using namespace std;
15
16 CascadeClassifier faceCascade, eyeCascade, noseCascade, mouthCascade;
17
18
19
20
21 void Mouth(Mat& img, Rect face) {
22     vector<Rect> mouths;
23     Mat faceROI = img(Rect(face.x, face.y + face.height * 2 / 3, face.width, face.height / 3));
24
25     double scaleFactor = 1.1;
26     int minNeighbors = 5;
27     int flags = 0;
28     Size minSize(20, 10);
29     Size maxSize;
30
31     mouthCascade.detectMultiScale(faceROI, mouths, scaleFactor, minNeighbors, flags, minSize, maxSize);
32
33     for (size_t i = 0; i < mouths.size(); i++) {
34         Rect mouth = mouths[i];
35         rectangle(img, Point(face.x + mouth.x, face.y + mouth.y + face.height * 2 / 3),
36                 Point(face.x + mouth.x + mouth.width, face.y + mouth.y + mouth.height + face.height * 2 / 3),
37                 Scalar(0, 255, 0), 2);
38     }
39 }
40
```

3. Detecting the Nose

The Nose function defines a Region of Interest (ROI) in the middle part of the face where the nose is likely to be. This ROI is calculated as `Rect(face.x + face.width / 4, face.y + face.height / 4, face.width / 2, face.height / 2)`. The image is converted to grayscale, and histogram equalization is applied to the ROI to enhance the contrast. The `noseCascade.detectMultiScale` method is used to detect the nose within the ROI. This method scans the ROI at multiple scales and returns a list of rectangles representing the detected noses. If a nose is detected, a rectangle is drawn around it. The coordinates are adjusted relative to the original face coordinates. The rectangle is drawn using the `rectangle` function.

4. Detecting the Eyes

The Eyes function focuses on detecting eyes within the detected face. It defines a region of interest (ROI) within the face where the eyes are likely to be. This ROI is the upper half of the face, calculated as `Rect(face.x + face.width / 8, face.y, face.width * 3 / 4, face.height / 2)`.

The `eyeCascade.detectMultiScale` method is used to detect eyes within the ROI. This method also scans the ROI at multiple scales and returns a list of rectangles representing the detected eyes. If two eyes are detected, rectangles are drawn around them. The coordinates are adjusted relative to the original face coordinates. The rectangles are drawn using the `rectangle` function.

```
41 void Nose(Mat& img, Rect face)
42 {
43     vector<Rect> noses;
44     Mat gray;
45
46     cvtColor(img, gray, COLOR_BGR2GRAY);
47     equalizeHist(gray, gray);
48
49     Mat faceROI = gray(Rect(face.x + face.width / 4, face.y + face.height / 4, face.width / 2, face.height / 2));
50
51     noseCascade.detectMultiScale(faceROI, noses, 1.1, 5, 0 | CASCADE_SCALE_IMAGE, Size(30, 30));
52
53     if (!noses.empty()) {
54         Rect nose = noses[0];
55
56         int x = nose.x + face.x + face.width / 4;
57         int y = nose.y + face.y + face.height / 4;
58         int width = nose.width;
59         int height = nose.height;
60
61         rectangle(img, Point(x, y), Point(x + width, y + height), Scalar(255, 0, 0), 2);
62     }
63 }
64
65 void Eyes(Mat& img, Rect face)
66 {
67     vector<Rect> e;
68     Mat faceEyes = img(Rect(face.x + face.width / 8, face.y, face.width * 3 / 4, face.height / 2));
69     eyeCascade.detectMultiScale(faceEyes, e, 1.1, 5, 0 | CASCADE_SCALE_IMAGE, Size(30, 30));
70
71     if (e.size() == 2)
72     {
73         Rect eye1 = e[0];
74         Rect eye2 = e[1];
75
76         int x = min(eye1.x, eye2.x) + face.x + face.width / 8;
77         int y = min(eye1.y, eye2.y) + face.y;
78         int width = max(eye1.x + eye1.width, eye2.x + eye2.width) - min(eye1.x, eye2.x) + 10;
79         int height = max(eye1.y + eye1.height, eye2.y + eye2.height) - min(eye1.y, eye2.y) + 10;
80
81         rectangle(img, Point(x, y), Point(x + width, y + height), Scalar(135, 0, 116), 6);
82     }
83 }
```

5. Detecting Facial Features

The core functionality of the code is to detect and highlight different facial features in a video. This is achieved through several functions, each focusing on a specific feature.

The function `FacialFeatures` starts by converting the input image to grayscale using `cvtColor`. Grayscale conversion reduces the computational complexity because it removes color information, which is not needed for feature detection.

The `equalizeHist` function is applied to the grayscale image to enhance the contrast. This process spreads out the most frequent intensity values, which helps in better detection of facial features under varying lighting conditions.

The face detection is performed using the `faceCascade.detectMultiScale` method. This method scans the grayscale image at multiple scales to detect faces of different sizes. It

returns a list of rectangles, each representing a detected face. The most important parameters used for all the detections made on the face are as follows:

- `imgGris`: The input grayscale image.
- `faces`: The vector where detected face rectangles are stored.
- `scaleFactor`: Specifies how much the image size is reduced at each image scale.
- `minNeighbors`: Specifies how many neighbors each candidate rectangle should have to retain it.
- `flags`: Flags for the detection process.
- `minSize`: Minimum possible object size. Objects smaller than this are ignored.

Finally, for each detected face, a rectangle is drawn around it using the `rectangle` function. This visually marks the detected face in the image.

```
86 void FacialFeatures(Mat img) {
87     vector<Rect> faces;
88     Mat imgGris;
89     cvtColor(img, imgGris, COLOR_BGR2GRAY);
90     equalizeHist(imgGris, imgGris);
91     faceCascade.detectMultiScale(imgGris, faces, 1.1, 5, 0 | CASCADE_SCALE_IMAGE, Size(50, 50));
92     for (size_t i = 0; i < faces.size(); i++)
93     {
94         Rect face = faces[i];
95         rectangle(img, Point(face.x, face.y), Point(face.x + face.width, face.y + face.height), Scalar(0, 0, 255), 2);
96         Eyes(img, face);
97         Mouth(img, face);
98         Nose(img, face);
99     }
100 }
101
102 extern "C" JNIEXPORT jlong JNICALL
103 Java_com_example_project_CameraActivity_camprocess(JNIEnv* env, jobject, jlong mat) {
104     Mat& inputMat = *(Mat*)mat;
105     Mat processedMat = inputMat.clone();
106     cvtColor(processedMat, processedMat, COLOR_BGR2RGB);
107     rotate(processedMat, processedMat, ROTATE_90_CLOCKWISE);
108     return (jlong)(new Mat(processedMat));
109 }
110
111 extern "C"
112 JNIEXPORT jlong __attribute__((unused))
113 JNICALL
114 Java_com_example_project_CameraActivity_processFace(JNIEnv* env, jobject this, jlong mat) {
115     Mat& inputMat = *(Mat*)mat;
116
117     FacialFeatures(inputMat);
118     rotate(inputMat, inputMat, ROTATE_90_CLOCKWISE);
119     flip(inputMat, inputMat, 1);
120
121     return (jlong) &inputMat;
122 }
```

6. Processing Images in Android

The code also includes functions to process images, remember since we are working with videos, these become several hundred frames/images, within our Android Studio application, therefore these functions are exposed to Java through JNI.

The function `Java_com_example_project_CameraActivity_camprocess` converts the image from BGR to RGB color space using `cvtColor` and rotates it for proper orientation using `rotate`. This ensures the image is in the correct format and orientation for further processing. The processed image is returned to the Android application as a new `Mat` object.

The function `Java_com_example_project_CameraActivity_processFace` calls the `FacialFeatures` function to detect and mark facial features in the image. This function is

responsible for detecting faces, eyes, nose, and mouth and drawing rectangles around them. The image is rotated and flipped to match the orientation expected by the Android application. This ensures the detected features are displayed correctly in the mobile app and the processed image with detected features is returned to the Android application as a new Mat object.

By following these steps, the code successfully detects and highlights facial features in an image, leveraging the power of OpenCV's pre-trained cascade classifiers and integrating seamlessly with an Android Studio application through JNI.

```
124 extern "C"
125 JNIEXPORT void JNICALL
126 Java_com_example_project_FaceDetector_nativeInitCascades(JNIEnv *env, jobject this,
127                                                         jstring faceCascadePath,
128                                                         jstring eyeCascadePath,
129                                                         jstring noseCascadePath,
130                                                         jstring mouthCascadePath)
131 {
132     const char* faceCascadePathStr = env->GetStringUTFChars(faceCascadePath, nullptr);
133     const char* eyeCascadePathStr = env->GetStringUTFChars(eyeCascadePath, nullptr);
134     const char* noseCascadePathStr = env->GetStringUTFChars(noseCascadePath, nullptr);
135     const char* mouthCascadePathStr = env->GetStringUTFChars(mouthCascadePath, nullptr);
136
137     faceCascade.load(faceCascadePathStr);
138     eyeCascade.load(eyeCascadePathStr);
139     noseCascade.load(noseCascadePathStr);
140     mouthCascade.load(mouthCascadePathStr);
141
142     env->ReleaseStringUTFChars(faceCascadePath, faceCascadePathStr);
143     env->ReleaseStringUTFChars(eyeCascadePath, eyeCascadePathStr);
144     env->ReleaseStringUTFChars(noseCascadePath, noseCascadePathStr);
145     env->ReleaseStringUTFChars(mouthCascadePath, mouthCascadePathStr);
146 }
147
```

7. Filter Server


```

def face(paht_one,path_two,path_glasses,idx):
    img = cv2.imread(paht_one)
    img_original = cv2.imread(path_two)
    glasses = cv2.imread(path_glasses)

    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    purple_bgr = np.uint8([[135, 0, 116]])
    purple_hsv = cv2.cvtColor(purple_bgr, cv2.COLOR_BGR2HSV)[0][0]

    lower_purple = np.array([purple_hsv[0] - 10, 70, 50])
    upper_purple = np.array([purple_hsv[0] + 10, 255, 255])

    mask = cv2.inRange(hsv, lower_purple, upper_purple)

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours:
        largest_contour = max(contours, key=cv2.contourArea)

        x, y, width, height = cv2.boundingRect(largest_contour)

        eyes = img_original[y:y+height, x:x+width]

        alto, ancho = eyes.shape[:2]

        rez_glasees = cv2.resize(glasses, (ancho, alto))

        glasses_gris = cv2.cvtColor(rez_glasees, cv2.COLOR_BGR2GRAY)
        ret,mascara = cv2.threshold(glasses_gris,190,255,cv2.THRESH_BINARY)
        mascara = cv2.bitwise_not(mascara)

        img_and = cv2.bitwise_and(eyes,eyes,mask = mask)
        imgt_and = cv2.bitwise_and(rez_glasees,rez_glasees,mask=mascara)

        resultado = cv2.add(img_and,imgt_and)
        img_original[y:y+height, x:x+width] = resultado

        # Guardar la imagen resultante
        result_path = '/home/apulla/Descargas/server-files-flask-master/files/FaceImg/'+f"Frame{idx}.jpg"
        cv2.imwrite(result_path, img_original)

        print(f'Image with glasses saved to {result_path}')
    else:
        print("No se encontró un rectángulo púrpura en la imagen.")

```

This Python code uses OpenCV to detect a purple region in an image, determine if it corresponds to the eyes, and overlay glasses on that region. It reads three images: `img` (for purple region detection), `img_original` (unmodified copy), and `glasses` (image of the glasses). The `img` image is converted from BGR to HSV color space to facilitate purple color detection.

A binary mask is created to identify pixels within the purple color range. Contours are found using this mask, and the largest contour, assumed to be the eye region, is selected.

The bounding rectangle of this contour is extracted from `img_original` and resized to match the size of the glasses. The glasses image is converted to grayscale, and binary masks are created. These masks help combine the glasses with the eye region using bitwise operations.

The combined region is then inserted back into `img_original`, and the resulting image is saved. If no purple contour is found, a message is printed.

▪ Object classification using Histograms of Oriented Gradients (HOG):

Now in this section, we will explain how the code predicts numbers based on the MNIST dataset using HOG descriptors and a multilayer perceptron neural network (MLP). We'll break down the process step-by-step, explaining the purpose and function of each part of the code.

1. Reading the MNIST Dataset

The process of reading the MNIST dataset involves extracting the image and label data from the compressed files. The code uses the `gzopen` function to open these files and `gzread` to read the contents since this function handles compressed gzip files, which is the format of the MNIST dataset files.

You might notice that the code reads the "magic number" from the image file, which we learned is a specific number used to verify the file type. For the image file, the magic number should be 2051. For the label file, it should be 2049. These checks ensure that the correct files are being read. The number of images, rows, and columns are then read from the image file. Similarly, the number of labels is read from the label file.

The code reads each image into a `Mat` object, which is a matrix representation used by OpenCV to store image data. Each matrix is of size 28x28, corresponding to the pixel dimensions of the MNIST images. These matrices are stored in a vector called `images`. Each entry in this vector is a matrix representing one handwritten digit image. Each label is read as a single byte and stored in a vector called `labels`. Each entry in this vector is an integer representing the digit (0-9) that corresponds to the image in the same position in the `images` vector. This setup ensures that the images and labels are correctly paired and ready for further processing.

```
main.cpp x modelophog.ipynb
C: > Users > USUARIO_PC > Desktop > proyectopart2 > main.cpp
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <opencv2/opencv.hpp>
5  #include <zlib.h>
6  #include <arpa/inet.h>
7
8  using namespace std;
9  using namespace cv;
10
11 void readMNIST(const string& imageFile, const string& labelFile, vector<Mat>& images, vector<int>& labels) {
12     gzFile imageFileStream = gzopen(imageFile.c_str(), "rb");
13     gzFile labelFileStream = gzopen(labelFile.c_str(), "rb");
14
15     if (!imageFileStream || !labelFileStream) {
16         cerr << "Error al abrir los archivos" << endl;
17         return;
18     }
19
20     int magicNumber, numImages, numLabels, numRows, numCols;
21
22     gzread(imageFileStream, &magicNumber, sizeof(magicNumber));
23     magicNumber = ntohl(magicNumber);
24
25     if (magicNumber != 2051) {
26         cerr << "Número mágico incorrecto en archivo de imágenes" << endl;
27         return;
28     }
29
30     gzread(imageFileStream, &numImages, sizeof(numImages));
31     numImages = ntohl(numImages);
32
33     gzread(imageFileStream, &numRows, sizeof(numRows));
34     numRows = ntohl(numRows);
35
36     gzread(imageFileStream, &numCols, sizeof(numCols));
37     numCols = ntohl(numCols);
38
39     gzread(labelFileStream, &magicNumber, sizeof(magicNumber));
40     magicNumber = ntohl(magicNumber);
41
42     if (magicNumber != 2049) {
43         cerr << "Número mágico incorrecto en archivo de etiquetas" << endl;
44         return;
45     }
46
47     gzread(labelFileStream, &numLabels, sizeof(numLabels));
48     numLabels = ntohl(numLabels);
49
50     if (numImages != numLabels) {
51         cerr << "Número de imágenes y etiquetas no coinciden" << endl;
52         return;
53     }
}
```

2. Extracting HOG Descriptors

After loading all the images, the next step is to extract HOG descriptors, which will capture the shape and structure of objects in the images, giving us the descriptors of the numbers needed for the prediction. The HOG descriptor is initialized with specific parameters such as window size, block size, block stride, cell size, and the number of bins. These parameters determine how the image is divided into cells and how the gradients are computed and binned.

Then for each image, the HOG descriptor computes the gradient orientations and compiles them into a histogram for each cell. These histograms are then combined to form the HOG descriptor for the entire image. The computed HOG descriptors are stored in a vector of vectors called `hogDescriptors`, where each inner vector represents the HOG features of one image.

3. Saving HOG Descriptors to CSV

Finally, to make it easy to use the data for training the neural network, we decided to make the HOG descriptors and labels saved into a CSV file so that the MLP neural network can easily train itself to the value of each vector.

For each image, the code writes the corresponding label followed by the HOG descriptors. Each row in the CSV file starts with the label, followed by the HOG descriptor values separated by commas. This format ensures that each row corresponds to one image and its features, making it easy to load and use for training the neural network.

```
54
55     images.resize(numImages);
56     labels.resize(numImages);
57
58     for (int i = 0; i < numImages; ++i) {
59         Mat img(numRows, numCols, CV_8U);
60         gzread(imageFileStream, img.data, numRows * numCols);
61         images[i] = img;
62
63         unsigned char label;
64         gzread(labelFileStream, &label, 1);
65         labels[i] = label;
66     }
67
68     gzclose(imageFileStream);
69     gzclose(labelFileStream);
70 }
71
72 void extractHOGDescriptors(const vector<Mat>& images, vector<vector<float>>& hogDescriptors) {
73     HOGDescriptor hog(
74         Size(28, 28),
75         Size(14, 14),
76         Size(7, 7),
77         Size(14, 14),
78         9);
79
80     for (const auto& img : images) {
81         vector<float> descriptors;
82         hog.compute(img, descriptors);
83         hogDescriptors.push_back(descriptors);
84     }
85 }
86
87 void saveDescriptorsToCSV(const string& filename, const vector<vector<float>>& hogDescriptors, const vector<int>& labels) {
88     ofstream file(filename);
89
90     for (size_t i = 0; i < hogDescriptors.size(); ++i) {
91         file << labels[i];
92         for (const auto& val : hogDescriptors[i]) {
93             file << "," << val;
94         }
95         file << endl;
96     }
97
98     file.close();
99 }
100
101 int main() {
102     vector<Mat> trainImages;
103     vector<int> trainLabels;
104     vector<vector<float>> hogDescriptors;
105
106     readMNIST("train-images-idx3-ubyte.gz", "train-labels-idx1-ubyte.gz", trainImages, trainLabels);
107     extractHOGDescriptors(trainImages, hogDescriptors);
108     saveDescriptorsToCSV("hog_train.csv", hogDescriptors, trainLabels);
109
110     return 0;
111 }
112
```

4. Training the Neural Network

Once we have the HOG descriptors saved, the next step is to train a neural network using these features. The HOG descriptors and labels are loaded from the CSV file into a pandas DataFrame. The features (HOG descriptors) are separated from the labels. The features are stored in a matrix X , and the labels are stored in a vector y .

The data is split into training and testing sets using `train_test_split` from scikit-learn. This function randomly divides the data, ensuring that 80% is used for training and 20% is used for testing. This split allows the model to be trained on one set of data and evaluated on another, helping to assess its performance.

An MLP (Multilayer Perceptron) classifier is created using scikit-learn's `MLPClassifier`. The MLP is a type of pretrained neural network that consists of multiple layers of neurons. The classifier is configured with the following parameters:

- `solver`: 'adam' is used for optimizing the weights.
- `activation`: 'relu' (Rectified Linear Unit) is used as the activation function, which introduces non-linearity into the model.
- `alpha`: 0.001, which is a regularization term to prevent overfitting.
- `hidden_layer_sizes`: (512, 128, 10), specifying three hidden layers with 512, 128, and 10 neurons respectively.
- `random_state`: 1, ensures reproducibility of the results.
- `max_iter`: 500, the maximum number of iterations for training.

The MLP classifier is trained using the training data (X_{train} and y_{train}). The `fit` method is called, which adjusts the weights of the network to minimize the error in predicting the labels from the features. During training, the MLP learns to map the input features (HOG descriptors) to the correct output labels (digits). The backpropagation algorithm is used to update the weights in the network by propagating the error backward through the network and adjusting the weights to minimize this error.

After training, the model's performance is evaluated by making predictions on both the training set and the testing set using the `predict` method. The accuracy of the model is calculated by comparing the predicted labels to the true labels. The training accuracy indicates how well the model has learned the training data, while the testing accuracy indicates how well the model generalizes to new, unseen data. This detailed process ensures that the neural network is well-trained and evaluated, leveraging the strengths of HOG descriptors for feature extraction and the MLP for robust classification, and with that we have been able to achieve a 97% to 99% of accuracy in our model.

```
[ ] import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pickle
```

```
data = pd.read_csv('/content/drive/MyDrive/hog_train.csv', header=None)

X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf = MLPClassifier(solver='adam',
                    activation='relu',
                    alpha=.001,
                    hidden_layer_sizes=(512, 128, 10),
                    random_state=1,
                    max_iter=500)

clf.fit(X_train, y_train)

train_pred = clf.predict(X_train)
train_accuracy = np.mean(train_pred == y_train)

test_pred = clf.predict(X_test)
test_accuracy = np.mean(test_pred == y_test)

print("Training accuracy: {}".format(train_accuracy))
print("Testing accuracy: {}".format(test_accuracy))
```

```
Training accuracy: 0.9996875
Testing accuracy: 0.9793333333333333
```

5. Saving and Converting the Model

Finally, the trained model is saved and can be converted to different formats for deployment, but first since the model is a trained scikit-learn model it must be saved to a file using pickle. The saved model is then converted to ONNX (Open Neural Network Exchange) format using skl2onnx. This format allows the model to be used in different environments and frameworks, which is our case. We decided our ONNX model to be further converted to TensorFlow Lite format since we have worked with this type before and it has been quite easy to be imported to our Android Studio because it is optimized for mobile and embedded devices, making it suitable for deployment on a wide range of platforms.

By following these steps, the code successfully reads the MNIST dataset, extracts meaningful features using HOG descriptors, trains a neural network model, and prepares the model for deployment in various formats. This approach ensures that the system can accurately predict handwritten digits, leveraging the strengths of deep learning and feature extraction techniques

```

import pickle
import skl2onnx
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

model_filename = '/content/drive/MyDrive/mlp_model.pkl'
with open(model_filename, 'rb') as file:
    sklearn_model = pickle.load(file)

n_features = sklearn_model.coefs_[0].shape[0]

initial_type = [('float_input', FloatTensorType([None, n_features]))]
onnx_model = convert_sklearn(sklearn_model, initial_types=initial_type)

onnx_model_filename = '/content/drive/MyDrive/mlp_model.onnx'
with open(onnx_model_filename, 'wb') as file:
    file.write(onnx_model.SerializeToString())

print("Modelo ONNX guardado en: {}".format(onnx_model_filename))

```

Modelo ONNX guardado en: /content/drive/MyDrive/mlp_model.onnx

+ Código

+ Texto

```
[ ] !pip install tf2onnx
```

```

Requirement already satisfied: tf2onnx in /usr/local/lib/python3.10/dist-packages (1.16.1)
Requirement already satisfied: numpy>=1.14.1 in /usr/local/lib/python3.10/dist-packages (from tf2onnx) (1.25.2)
Requirement already satisfied: onnx>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from tf2onnx) (1.16.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from tf2onnx) (2.31.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from tf2onnx) (1.16.0)
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.10/dist-packages (from tf2onnx) (24.3.25)
Collecting protobuf<=3.20 (from tf2onnx)
  Downloading protobuf-3.20.3-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl.metadata (679 bytes)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->tf2onnx) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->tf2onnx) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->tf2onnx) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->tf2onnx) (2024.7.4)
Downloading protobuf-3.20.3-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.1 MB)
1.1/1.1 MB 17.4 MB/s eta 0:00:00
Installing collected packages: protobuf

```

Successfully installed protobuf-3.20.3

```
!python -m tf2onnx.convert --opset 13 --onnx /content/drive/MyDrive/mlp_model.onnx --output /content/drive/MyDrive/mlp_model_tf.pb
```

```

/usr/lib/python3.10/runpy.py:126: RuntimeWarning: 'tf2onnx.convert' found in sys.modules after import of package 'tf2onnx', but prior to e
warn(RuntimeWarning(msg))
usage: convert.py [-h] [--input INPUT] [--graphdef GRAPHDEF] [--saved-model SAVED_MODEL]
                  [--tag TAG] [--signature_def SIGNATURE_DEF]
                  [--concrete_function CONCRETE_FUNCTION] [--checkpoint CHECKPOINT]
                  [--keras KERAS] [--tfLite TFLITE] [--tfjs TFJS] [--large_model]
                  [--output OUTPUT] [--inputs INPUTS] [--outputs OUTPUTS]
                  [--ignore_default IGNORE_DEFAULT] [--use_default USE_DEFAULT]
                  [--rename-inputs RENAME_INPUTS] [--rename-outputs RENAME_OUTPUTS]
                  [--use-graph-names] [--opset OPSET] [--dequantize] [--custom-ops CUSTOM_OPS]
                  [--extra_opset EXTRA_OPSET] [--load_op_libraries LOAD_OP_LIBRARIES]
                  [--target {rs4,rs5,rs6,caffe2,tensorrt,nhw}] [--continue_on_error] [--verbose]
                  [--debug] [--output_frozen_graph OUTPUT_FROZEN_GRAPH]
                  [--inputs-as-nchw INPUTS_AS_NCHW] [--outputs-as-nchw OUTPUTS_AS_NCHW]
convert.py: error: unrecognized arguments: --onnx /content/drive/MyDrive/mlp_model.onnx

```

```

[ ] import tensorflow as tf

saved_model_dir = "/content/drive/MyDrive/mlp_model_tf"
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()

tflite_model_filename = '/content/drive/MyDrive/mlp_model.tflite'
with open(tflite_model_filename, 'wb') as file:
    file.write(tflite_model)

print("Modelo TensorFlow Lite guardado en: {}".format(tflite_model_filename))

```

Modelo TensorFlow Lite guardado en: /content/drive/MyDrive/mlp_model.tflite

To ensure that our model works we made predictions from 3 random images of the dataset and made the comparisons between the true label of the image and our predictions based on our model trained by HOG descriptors; we also created a confusion matrix with all of the images labels noticing that even though that thousands of the predictions made were true positives, there were a few false negatives and positives that fell through the cracks, therefore we had the following results.

```

random_indices = random.sample(range(len(X_test)), 3)
random_images = X_test[random_indices]
random_labels = y_test[random_indices]

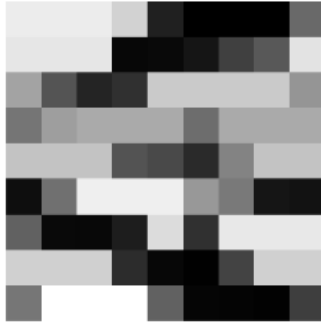
random_predictions = clf.predict(random_images)

for i in range(3):
    # Determine the correct dimensions for reshaping
    image_side = int(np.sqrt(random_images[i].size))
    image = random_images[i].reshape(image_side, image_side)

    plt.imshow(image, cmap='gray')
    plt.title(f"Etiqueta Verdadera: {random_labels[i]}, Predicción: {random_predictions[i]}")
    plt.axis('off')
    plt.show()

```

Etiqueta Verdadera: 0, Predicción: 0



Etiqueta Verdadera: 8, Predicción: 8



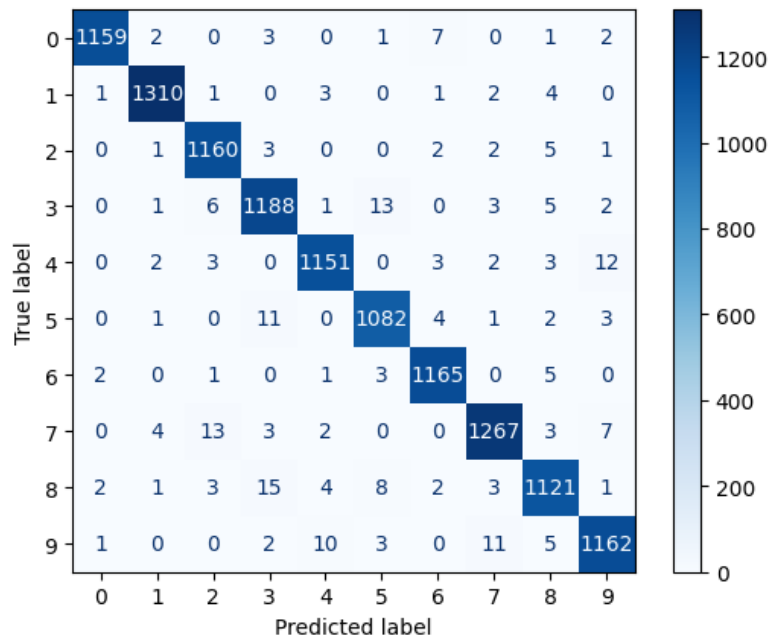

```

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

conf_matrix = confusion_matrix(y_test, test_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.arange(10))
disp.plot(cmap=plt.cm.Blues)
plt.show()

```



Now we enter the part in the code that processes a new image to compute Histogram of Oriented Gradients (HOG) descriptors. The core functionality of the code is to compute HOG descriptors for the new input image.

The core functionality of the code is to compute HOG descriptors is repeated once again for the input image. HOG descriptors capture the shape and texture of objects within an image by analyzing the distribution of gradient orientations with the specific parameters tailored for effective gradient computation as we saw before, we need the same values so that the new MLP model created can understand the input values of the vector for the prediction since the computeHOG function returns a vector of floating-point numbers representing these HOG descriptors. This vector encapsulates the essential structural information of the input image.

```

1  #include <opencv2/opencv.hpp>
2  #include <filesystem>
3  #include <iostream>
4  #include <string>
5  #include <jni.h>
6
7  using namespace cv;
8  using namespace std;
9
10 vector<float> computeHOG(const Mat &image) {
11     vector<float> features;
12     Size newSize(28, 28);
13     resize(image, image, newSize);
14
15     HOGDescriptor hog(
16         Size(28, 28),
17         Size(14, 14),
18         Size(7, 7),
19         Size(14, 14),
20         9);
21
22     hog.compute(image, features);
23
24     return features;
25 }
26

```

In addition to computing HOG descriptors, we also include the functions to preprocess images within an Android application. These functions are exposed to Java through JNI, allowing for seamless integration of native C++ code with the Android app. One such function is `Java_com_example_project_CameraActivity_processHog`, which preprocesses the image before computing HOG descriptors. This function clones the input image to ensure the original remains unchanged, converts the color space from BGR to RGB, resizes the image to 28x28 pixels, and rotates it 90 degrees clockwise. The image is then converted to grayscale, and a binary inverse threshold is applied to enhance the features, converting the grayscale image to a binary format where pixels are either black or white. The preprocessed image is then returned to the Android application.

Another important function is `Java_com_example_project_CameraActivity_computeHOG`, which computes HOG descriptors for the preprocessed image. It calls the `computeHOG` function with the input image to obtain the HOG descriptors. A new Java float array is created to store these descriptors, and the data is copied from the native C++ environment to the Java environment using `SetFloatArrayRegion`. This array is then returned to the Android application, making the HOG descriptors accessible for further processing or analysis.

This effectively combines the strengths of OpenCV and TensorFlow Lite, leveraging JNI for efficient native code execution. It enables real-time image processing and feature extraction

in an Android environment, providing a robust method for analyzing and interpreting visual information.

```
27
28 extern "C" JNIEXPORT jlong JNICALL
29 Java_com_example_project_CameraActivity_processHog(JNIEnv* env, jobject, jlong mat) {
30     Mat& inputMat = *(Mat*)mat;
31     Mat processedMat = inputMat.clone();
32     cvtColor(processedMat, processedMat, COLOR_BGR2RGB);
33     resize(processedMat, processedMat, Size(28,28));
34     rotate(processedMat, processedMat, ROTATE_90_CLOCKWISE);
35     Mat gray;
36
37     cvtColor(processedMat, gray, COLOR_RGB2GRAY);
38     threshold(gray, gray, 127, 255, THRESH_BINARY_INV);
39     return (jlong)(new Mat(gray));
40 }
41
42 extern "C" JNIEXPORT jfloatArray JNICALL
43 Java_com_example_project_CameraActivity_computeHOG(JNIEnv *env, jobject, jlong matAddr) {
44     Mat &image = *(Mat *)matAddr;
45     vector<float> features = computeHOG(image);
46
47     jfloatArray result = env->NewFloatArray(features.size());
48     if (result == nullptr) {
49         return nullptr;
50     }
51
52     env->SetFloatArrayRegion(result, 0, features.size(), features.data());
53     return result;
54 }
```

Lastly, we take all the final functions of preprocessing into our mobile app, This solution leverages the power of OpenCV and TensorFlow Lite within an Android application to process images, extract features, apply filters and perform real-time analysis. The integration of native C++ code through JNI (Java Native Interface) ensures efficient image processing.

```

public class
CameraActivity extends org.opencv.android.CameraActivity {
    String TAG = "CameraActivity";
    Mat mRgba;

    private static final String[] CLASS_NAMES = {"5", "1", "2", "3", "4", "0", "6", "7", "8", "9"};

    int frameHeight, frameWidth;
    private final List<Mat> framesList = new ArrayList<>();
    private final List<Mat> framesListOri = new ArrayList<>();
    private long lastFrameTime = 0;
    private final List<Mat> framesListHog = new ArrayList<>();

    private int frameCount = 0;
    private TextView fpsTextView;
    private Interpreter tflite;
    String className;
    List<Mat> framesCopy = new ArrayList<>();
    private boolean capturingFrames = false;
    native long camara(long mat);
    native long filterOne(long mat);
    native long filterTwo(long mat);
    native long filterThree(long mat);
    native long filterFour(long mat);
    native long processFace(long mat);
    native long processHog(long mat);
    native long camprocess(long mat);
    public native float[] computeHOG(long matAddr);
    CameraBridgeViewBase cameraBridgeViewBase;
    ImageView take_photo_btn;
    Mat frame;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_camera2);

        Intent intent = getIntent();
        String paramValue = intent.getStringExtra("param_key");

        fpsTextView = findViewById(R.id.fps_text_view);

        getPermission();
        cameraBridgeViewBase = findViewById(R.id.cameraView);

        cameraBridgeViewBase.setCvCameraViewListener(new CameraBridgeViewBase.CvCameraViewListener2() {
            @Override
            public void onCameraViewStarted(int width, int height) {
                frameHeight = height;
                frameWidth = width;
            }
        });
    }
}

```

The CameraActivity class is the core component of this solution, extending org.opencv.android.CameraActivity and handling the initialization and processing of camera frames. Upon activity creation, it sets the layout and retrieves parameters passed through an Intent to determine the mode of operation, either "Face" detection or "Hog" (Histogram of Oriented Gradients) feature extraction.

The activity initializes key UI components such as fpsTextView, memoryTextView, and cpuTextView to display real-time performance metrics. It also sets up the cameraBridgeViewBase, which is responsible for displaying the camera feed and managing frame processing. Permissions for camera access are requested and handled appropriately to ensure the app has the necessary access to capture images.

The cameraBridgeViewBase is configured with a listener to handle events such as starting and stopping the camera view and processing each camera frame. When the camera view starts, frame dimensions are initialized, and a handler is set up to update memory and CPU usage information. The onCameraFrame method processes each frame captured by the

camera, calculates frames per second (FPS), and updates the fpsTextView. Depending on the mode specified by paramValue, the frame is either processed for the face detection or prepared for the HOG feature extraction.

```

CameraActivity extends org.opencv.android.CameraActivity {
    protected void onCreate(Bundle savedInstanceState) {
        cameraBridgeViewBase.setCvCameraViewListener(new CameraBridgeViewBase.CvCameraViewListener2() {
            public void onCameraViewStarted(int width, int height) {
                frameHeight = height;
                frameWidth = width;
            }

            @Override
            public void onCameraViewStopped() {
            }

            @Override
            public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
                long currentTime = System.currentTimeMillis();
                if (lastFrameTime != 0) {
                    long elapsedTime = currentTime - lastFrameTime;
                    frameCount++;
                    if (elapsedTime >= 1000) {
                        final double fps = frameCount / (elapsedTime / 1000.0);
                        runOnUiThread(() -> fpsTextView.setText(String.format("FPS: %.2f", fps)));
                        frameCount = 0;
                        lastFrameTime = currentTime;
                    }
                } else {
                    lastFrameTime = currentTime;
                }

                mRgba = inputFrame.rgba();
                Mat img0 = mRgba.clone();
                if (paramValue.equals("Face")) {
                    frame = mRgba.t();
                    processFace(frame.nativeObj);
                    if (capturingFrames) {
                        framesList.add(frame);
                        framesListOri.add(img0);
                    }
                } else if (paramValue.equals("Hog")) {
                    frame = img0;
                }
                return frame;
            }
        });

        if (OpenCVLoader.initDebug()) {
            cameraBridgeViewBase.enableView();
        }
    }
}

```

In "Face" mode, the processFace native method is called to detect faces in the frame. If frames are being captured, the current frame and its original version are added to framesList and framesListOri, respectively. In "Hog" mode, the frame is directly assigned for processing. The take_photo_btn triggers actions based on the mode: starting frame capture for eight seconds in "Face" mode or predicting the model in "Hog" mode. The startCapturingFramesForEightSeconds method captures frames for eight seconds to create a video and logs the action. After capturing, it saves the frames using the saveFrame or save methods. The predicModel method processes the image to extract HOG features using the processHog native method and the computeHOG method. These HOG features are then used to run inference with a TensorFlow Lite model. The model is loaded using the loadModelFile

method, which maps the model file into memory for an efficient access.

```
take_photo_btn = findViewById(R.id.take_photo_btn);
if(paramValue.equals("Face")){
    take_photo_btn.setOnClickListener(v -> startCapturingFramesForEightSeconds(paramValue));
}else if(paramValue.equals("Hog")){
    take_photo_btn.setOnClickListener(v -> predicModel(frame));
}
}

private void startCapturingFramesForEightSeconds(String paramValue) {
    capturingFrames = true;
    Log.i(TAG, paramValue);
    new Handler().postDelayed(() -> {
        capturingFrames = false;
        if (paramValue.equals("Face")) {
            saveFrame(framesList, "FaceDetection");
            save(framesListOri, "OriginalVi");
        } else if (paramValue.equals("Hog")) {
            // Handle Hog case if needed
        }
    }, 8000); // Changed to 8000 for eight seconds
}

private void predicModel(Mat img) {
    Log.i(TAG, img.toString());
    Mat hog = img.clone();
    long mt = processHog(hog.nativeObj);
    Mat imgHog = new Mat(mt);

    float[] hogFeatures = computeHOG(imgHog.getNativeObjAddr());

    Log.d("HOG Features", "Number of HOG features: " + hogFeatures.length);
    Log.d("Model Info", "Input Shape: " + Arrays.toString(hogFeatures));

    try {
        tflite = new Interpreter(loadModelFile());
        int[] inputShape = tflite.getInputTensor(0).shape();
        int[] outputShape = tflite.getOutputTensor(0).shape();

        Log.d("Model Info", "Input Shape: " + Arrays.toString(inputShape));
        Log.d("Model Info", "Output Shape: " + Arrays.toString(outputShape));

        if (inputShape.length > 3) {
            int inputChannels = inputShape[3];
            Log.d("Model Info", "Input Channels: " + inputChannels);
        } else {
            Log.d("Model Info", "Input Shape does not have 4 dimensions, cannot get input channels.");
        }
    }
}
```

The inference results are analyzed to determine the predicted class name. The HOG features and model input/output shapes are logged for debugging. The inference results are analyzed to find the maximum value and corresponding class name. The saveHog method saves the image with the predicted class name to external storage and starts a new activity to display the results. The rest of the code indicates successful or failed accounts of either loading the model, saving the image, writing the image for external storage and overall permissions of access to avoid any problems during the transactions between the mobile app and the web server.

```

CameraActivity extends org.opencv.android.CameraActivity {
    private void predicModel(Mat img) {
        try {
            Log.d("Model Info", "Input Shape: " + Arrays.toString(inputShape));
            Log.d("Model Info", "Output Shape: " + Arrays.toString(outputShape));

            if (inputShape.length > 3) {
                int inputChannels = inputShape[3];
                Log.d("Model Info", "Input Channels: " + inputChannels);
            } else {
                Log.d("Model Info", "Input Shape does not have 4 dimensions, cannot get input channels.");
            }

            Log.d(TAG, "TensorFlow Lite model loaded successfully.");
        } catch (IOException e) {
            e.printStackTrace();
            Log.d(TAG, "TensorFlow Lite model loaded error.");
        }

        float[][] result = runInference(hogFeatures);
        for (int i = 0; i < result.length; i++) {
            StringBuilder sb = new StringBuilder();
            for (int j = 0; j < result[i].length; j++) {
                sb.append(result[i][j]).append(" ");
            }
            Log.d("HOGResult", "Result row " + i + ": " + sb.toString());
        }

        float maxValue = Float.MIN_VALUE;
        int maxRow = -1;
        int maxCol = -1;

        for (int i = 0; i < result.length; i++) {
            for (int j = 0; j < result[i].length; j++) {
                if (result[i][j] > maxValue) {
                    maxValue = result[i][j];
                    maxRow = i;
                    maxCol = j;
                }
            }
        }

        className = CLASS_NAMES[maxCol];

        Log.d("ResultAnalysis", "Best result value: " + maxValue);
        Log.d("ResultAnalysis", "Position: Row " + maxRow + ", Column " + maxCol);
        Log.d("ResultAnalysis ", "Class name: " + className);
        saveHog(imgHog, "Hog");
    }
}

```

```

private float[][] runInference(float[] bitmap) {

    float[][] output = new float[1][10];

    tfLite.run(bitmap, output);

    return output;
}

// Save images
private void saveFrame(List<Mat> mat, String name) {
    Intent intent = new Intent(this, FaceDetector.class);

    File filter = new File(getExternalFilesDir(null), name);
    if (!filter.exists()) filter.mkdirs();

    File[] files = filter.listFiles();
    if (files != null) {
        for (File file : files) {
            file.delete();
        }
    }

    for (int i = 0; i < mat.size(); i++) {
        long mt = camprocess(mat.get(i).nativeObj);
        Mat frame = new Mat(mt);
        String filename = "Frame" + i + ".jpg";
        File file = new File(filter, filename);
        boolean bool = Imgcodecs.imwrite(file.toString(), frame);
        frame.release();
        if (bool) {
            Log.i(TAG, "Image saved successfully: " + file);
            intent.putExtra("imagePath", file.getAbsolutePath());
        } else {
            Log.i(TAG, "Failed to write image to external storage");
        }
    }
    framesList.clear();
    startActivity(intent);
}

```

```

CameraActivity extends org.opencv.android.CameraActivity {

    private void saveHog(Mat mat, String name) {
        Intent intent = new Intent(this, FaceDetector.class);

        File filter = new File(getExternalFilesDir(null), name);
        if (!filter.exists()) filter.mkdirs();

        File[] files = filter.listFiles();
        if (files != null) {
            for (File file : files) {
                file.delete();
            }
        }

        Mat frame = mat.clone();
        String filename = className+".jpg";
        File file = new File(filter, filename);
        boolean bool = Imgcodecs.imwrite(file.toString(), frame);
        frame.release();
        if (bool) {
            Log.i(TAG, "Image saved successfully: " + file);
            intent.putExtra("imagePath", file.getAbsolutePath());
        } else {
            Log.i(TAG, "Failed to write image to external storage");
        }

        startActivity(intent);
    }

    private void save(List<Mat> matIng, String name) {
        File filter = new File(getExternalFilesDir(null), name);
        if (!filter.exists()) filter.mkdirs();

        File[] files = filter.listFiles();
        if (files != null) {
            for (File file : files) {
                file.delete();
            }
        }

        for (int i = 0; i < matIng.size(); i++) {
            long mt = camprocess(matIng.get(i).nativeObj);
            Mat frame = new Mat(mt);
            String filename = "Frame" + i + ".jpg";
            File file = new File(filter, filename);
            boolean bool = Imgcodecs.imwrite(file.toString(), frame);
            frame.release();
            if (bool) {
                Log.i(TAG, "SUCCESS writing image to external storage: " + file);
            } else {
                Log.i(TAG, "Failed to write image to external storage");
            }
        }
    }
}

```



```

private MappedByteBuffer loadModelFile() throws IOException {
    Resources resources = getResources();
    int modelResourceId = R.raw.mlp_model;
    FileInputStream inputStream = new FileInputStream(resources.openRawResourceFd(modelResourceId).getFileDescriptor());
    FileChannel fileChannel = inputStream.getChannel();
    long startOffset = resources.openRawResourceFd(modelResourceId).getStartOffset();
    long declaredLength = resources.openRawResourceFd(modelResourceId).getDeclaredLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength);
}

@Override
protected void onPause() {
    super.onPause();
    if (cameraBridgeViewBase != null) {
        cameraBridgeViewBase.disableView();
    }
}

@Override
protected List<? extends CameraBridgeViewBase> getCameraViewList() {
    return Collections.singletonList(cameraBridgeViewBase);
}

void getPermission(){
    if(checkSelfPermission(android.Manifest.permission.CAMERA)!= PackageManager.PERMISSION_GRANTED){
        requestPermissions(new String[]{android.Manifest.permission.CAMERA},101);
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if(grantResults.length > 0 && grantResults[0]!= PackageManager.PERMISSION_GRANTED){
        getPermission();
    }
}

```

The solution that we have demonstrated effectively combines the strengths of OpenCV and TensorFlow Lite, leveraging JNI for efficient native code execution. It enables real-time image processing and feature extraction in an Android environment, providing a robust method for analyzing and interpreting visual information. This detailed explanation covers the entire process, from initializing the camera and handling permissions to capturing and processing frames, running model inference, and saving the results to then present onto the server.

Results

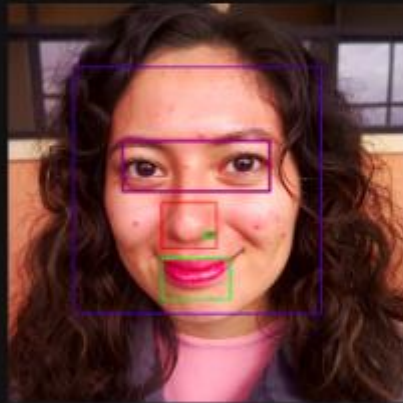
The following images represent what our code does in our App and how it then shows the results on our web server:

Project

Proyecto Interciclo

FACE DETECTOR

HOG DETECTION

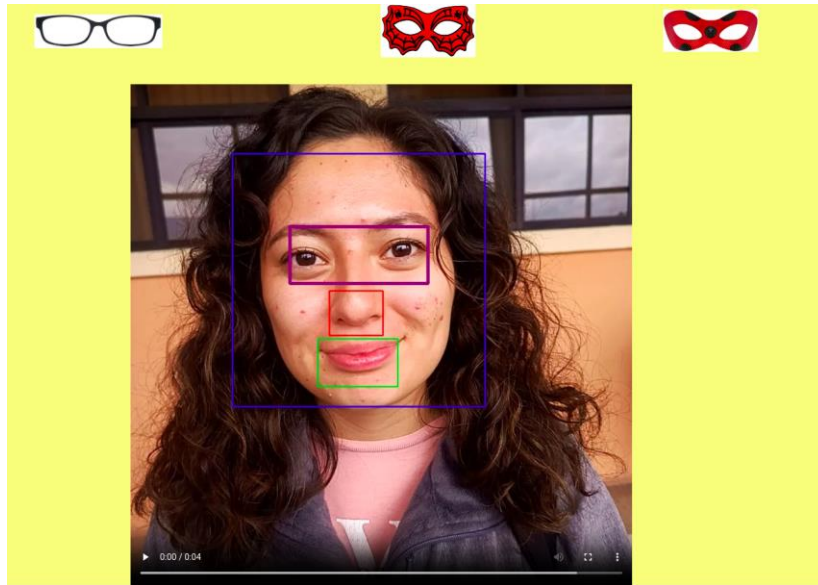


SERVER CONNECTION

SERVER PREDICTION

CPU: 230655MB
RAM: 2629MB / 7685MB

FPS: 1.52



Project

Proyecto Interciclo

FACE DETECTOR

HOG DETECTION



SERVER CONNECTION

SERVER PREDICTION

CPU: 230655MB
RAM: 2394MB / 7685MB

FPS: 21.38



Conclusions

On this project we focused on developing a computer vision system capable of detecting and classifying objects using deep learning and feature extraction. We developed an Android Studio application using Java and C++ with two main parts: facial detection and number classification.

In the first part, we used Haar cascades in C++ with OpenCV to detect faces in captured images. We then added glasses and two other types of masks to the detected face. This demonstrates the effectiveness of Haar cascades. We started with facial feature detection using the Haar cascade method. This technique allowed us to accurately identify faces, eyes, nose, and mouth in images. By integrating this capability into a mobile application through the Java Native Interface (JNI), we ensured smooth interaction between Java and native C++ code for efficient processing.

As we progressed in our project, we noticed that an essential part of the project was improving image quality to enhance detection accuracy. We used techniques like converting images to grayscale, applying histogram equalization, and adjusting lighting and resolution to try to minimize errors and reduce false positives, which resulted in crucial for accurate face detection and object classification.

In the second part, we focused on number classification with a neural network model. We processed the number images using Histograms of Oriented Gradients (HOG) to extract their features. With this, we trained on a Multilayer Perceptron (MLP) neural network on the MNIST dataset, enabling the system to predict handwritten digits accurately. We detailed the process of reading the dataset, extracting HOG descriptors, and training the neural network to ensure clarity in our TensorFlow lite model and integrated it into the application. The model can recognize and classify numbers, highlighting the use of machine learning on mobile devices.

Overall, this project showcased the effectiveness of combining deep learning and feature extraction for sophisticated computer vision applications while addressing the challenges of real-time image processing and feature extraction, paving the way for future advancements in computer vision. The techniques and technologies used here can be refined and expanded for broader applications, contributing to the development of intelligent systems that can interpret and respond to visual information. The successful integration of these methods within an Android environment, leveraging OpenCV and TensorFlow Lite, demonstrated the feasibility of deploying such systems on mobile devices.

To look further into detail at our code please follow the link: https://estliveupsedu-my.sharepoint.com/:u:/g/personal/kpaltin_est_ups_edu_ec/EY-8-2aS65EtiNVX30IbeEBppvtZqtlRCKWsBrOYLxUlw?e=Sa6ThU

Bibliography

- A. J. Newell and L. D. Griffin, "Multiscale Histogram of Oriented Gradient Descriptors for Robust Character Recognition," 2011 International Conference on Document Analysis and Recognition, Beijing, China, 2011, pp. 1085-1089, doi: 10.1109/ICDAR.2011.219.
- Build from source. (2024). Retrieved from TensorFlow website: <https://www.tensorflow.org/install/source>
- Dalal, N. (n.d.). Histogram of oriented gradients (HOG) for object detection. Retrieved from Stanford.edu website: http://vision.stanford.edu/teaching/cs231b_spring1213/slides/HOG_2011_Stanford.pdf
- Fernández, F., & Mariel, K. (2016). Modelo para optimizar la detección de rostros en secuencias de video (Universidad Nacional San Agustín de Arequipa). Retrieved from <https://repositorio.unsa.edu.pe/bitstreams/b90d2fb5-0e07-4b06-9cb3-a6eae46a6047/download>
- Histogram of oriented gradients -- MNIST/MNIST with HOG features.Ipynb at master · pransen/ComputerVisionAlgorithms. (2020). Retrieved from <https://github.com/pransen/ComputerVisionAlgorithms/blob/master/Histogram%20of%20Oriented%20Gradients%20--%20MNIST/MNIST%20with%20HOG%20Features.ipynb>
- Jaiswal, A. (2022, October 19). Face Detection using Haar-Cascade using Python. Retrieved from Analytics Vidhya website: <https://www.analyticsvidhya.com/blog/2022/10/face-detection-using-haar-cascade-using-python/>
- Kyaw, W. Y. (2021, May 10). Histogram of Oriented Gradients. Retrieved from Medium website: <https://waiyankyawmc.medium.com/histogram-of-oriented-gradients-90567ea6490a>
- Reddy, S. (2010). Face Detection Describes face detection using opencv. Retrieved from Blogspot.com website: <https://opencvfacedetect.blogspot.com/2010/10/face-detectionfollowed-by-eyese.h>
- S. S. Ali, J. H. Al' Ameri and T. Abbas, "Face Detection Using Haar Cascade Algorithm," 2022 Fifth College of Science International Conference of Recent Trends in Information Technology (CSCTIT), Baghdad, Iraq, 2022, pp. 198-201, doi: 10.1109/CSCTIT56299.2022.10145680.