ESTUDIANTES:

ANTHONY PULLA

KELLY PALTIN

ASIGNATURA:

VISION POR COMPUTADOR

TEMA:

CLASIFICACIÓN DE OBJETOS EMPLEANDO LA TÉCNICA DE HISTOGRAMA DE GRADIENTES ORIENTADOS (HOG).

DOCENTE:

ING. VLADIMIR ROBLES

FECHA:

04 DE JUlIO DE 2024

PERIODO ACADÉMICO:

64

PRACTICA #3.2 – Clasificación de objetos empleando la técnica de Histograma de Gradientes Orientados (HOG).

Desarrollar un programa que permita calcular el Histograma de Gradientes Orientados (HOG) de acuerdo al tutorial "Histogram of Oriented Gradients (HOG) for Multiclass Image Classification and Image Recommendation" de zonas de interés en una imagen a fin de clasificar logos, para ello deberá considerar los siguientes aspectos:

1. Entrenar un clasificador basado en Máquinas de Soporte Vectorial (SVM) en OpenCV C++ para realizar detección de objetos en el Logo-Dataset. Un ejemplo de dataset se puede observar en la Ilustración 1:

Ilustración 1. Ejemplo de imágenes que pertenecen al Logo-Dataset

2. Cada grupo (de máximo 2 personas) de escoger una de las siguientes combinaciones o grupos de imágenes para entrenar su clasificador (ningún grupo puede la misma opción que los demás grupos).

ID	Combinación
1	microsoft-text,microsoft-image,google-image,telegram
2	paypal-text,discord,telegram,netflix-image
3	ebay,snapchat,tiktok-image,telegram
4	whatsapp-image,youtube-text,youtube-image,netflix-image
5	tiktok-image-2,zoom-text,google-image,netflix-image
6	spotify,zoom-image,snapchat,facebook-image
7	twitter,zoom-text,amazon-text,yahoo
8	apple,yahoo,zoom-image,telegram
9	paypal-image,microsoft-image,youtube-text,netflix-image
10	tiktok-text,discord,amazon-text,youtube-image
11	facebook-text,amazon-text,snapchat,netflix-image
12	google-text,amazon-text,facebook-image,youtube-image
13	dhl,microsoft-image,yahoo,facebook-image
14	ikea,snapchat,linkedin,tiktok-image
15	amazon-image,instagram,discord,tiktok-image
16	whatsapp-text,snapchat,youtube-image,telegram
17	netflix-text,instagram,yahoo,youtube-image

Tabla 1. Grupos de imágenes con las que debe entrenarse el clasificador

Nuestro grupo trabajó con el grupo de imágenes del ID 12: Google, Amazon, Facebook, YouTube.

3. Deberá desarrollar un informe que contenga la siguiente información:

▪ Matriz de confusión correspondiente a la clasificación (puede usar las otras categorías para probar el clasificador).

▪ Código fuente desarrollado en C++ y OpenCV.

▪ Ejemplos de clasificaciones correctas e incorrectas (colocar al menos 2 imágenes por categoría).

Código del GitHub: https://github.com/anthonypulla126/Task-3.2/tree/main

Comando para ejecutar el programa:

./Task3 "ruta de la img a predecir"

Declaración de HOG y otras funciones.

```
HOGDescriptor hog(
    Size(32, 32),
    Size(16, 16),
    Size(8, 8),
    Size(8, 8),
    9
);

pair<string, bool> containsKeyword(string filename) {
    if (filename.find("google") != std::string::npos) { return {"google", true};}
    else if (filename.find("amazon") != std::string::npos) { return {"amazon", true};}
    else if (filename.find("facebook") != std::string::npos) { return {"facebook", true};}
    else if (filename.find("youtube") != std::string::npos) { return {"youtube", true};}
    else { return {"", false}; }
}

vector<float> HOG(string filename){
    Mat img = imread(filename,IMREAD_GRAYSCALE);
    resize(img, img, Size(32, 32), 0, 0, INTER_LINEAR);
```

```
vector<float> HOG(string filename){
    Mat img = imread(filename,IMREAD_GRAYSCALE);
    resize(img, img, Size(32, 32), 0, 0, INTER_LINEAR);

    Mat blurred;
    GaussianBlur(img, blurred, Size(5, 5), 0);

    hog.compute(blurred, descriptors);

    return descriptors;
}

void SVM(const Ptr<SVM>& svm, const string& filename) {
    svm->save(filename);
    cout << "Modelo SVM guardado con éxito" << endl;
}
```

Función para la matriz de confusión

```
void visualizeConfusionMatrix(int TP, int TN, int FP, int FN, const string& className) {
    Mat confusionMat(800, 800, CV_8UC3, Scalar(255, 255, 255));
    rectangle(confusionMat, Point(100, 100), Point(700, 700), Scalar(0, 0, 0), 2);
    line(confusionMat, Point(100, 400), Point(700, 400), Scalar(0, 0, 0), 2);
    line(confusionMat, Point(400, 100), Point(400, 700), Scalar(0, 0, 0), 2);

    putText(confusionMat, "Actual Values", Point(300, 70), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);
    putText(confusionMat, "Predicted     Values", Point(200, 450), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2, true);

    putText(confusionMat, "Negative:", Point(120, 150), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);
    putText(confusionMat, "Positive:", Point(520, 150), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);

    putText(confusionMat, "Negative:", Point(120, 500), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2, true);
    putText(confusionMat, "Positive:", Point(520, 500), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2, true);

    putText(confusionMat, "TN: " + to_string(TN), Point(150, 300), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);
    putText(confusionMat, "FP: " + to_string(FP), Point(450, 300), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);
    putText(confusionMat, "FN: " + to_string(FN), Point(150, 550), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);
    putText(confusionMat, "TP: " + to_string(TP), Point(450, 550), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);

    putText(confusionMat, "Confusion Matrix for: " + className, Point(150, 750), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);

    imshow("Confusion Matrix", confusionMat);
    waitKey(0);
    destroyAllWindows();
}
```

```
int main(int argc, char** argv) {

    if (argc < 2) {
        cout << "Usage: " << argv[0] << " <test image path>" << endl;
        return -1;
    }

    string test_image_path = argv[1];
    string path = "../Logos";
    vector<vector<float>> hog_features;

    for (const auto& entry : fs::directory_iterator(path)) {
        std::string filename = entry.path().filename().string();
        auto result = containsKeyword(filename);
        if (result.second) {
            vector<float> descriptor = HOG(entry.path());
            labels.push_back(result.first);
            hog_features.push_back(descriptor);
        }
    }
```

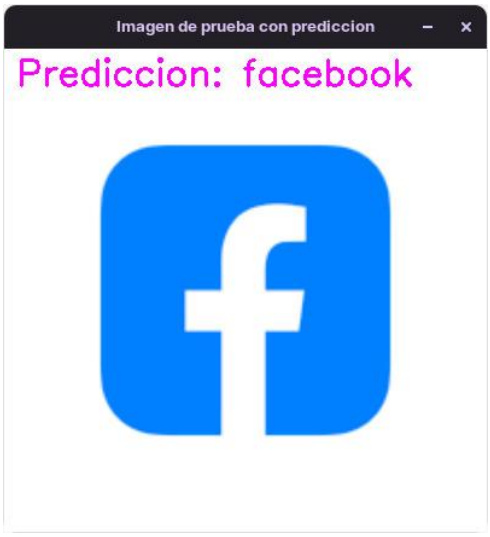
```cpp
    map<string, int> labelMap;
    int labelCounter = 0;
    for (const auto& label : labels) {
        if (labelMap.find(label) == labelMap.end()) {
            labelMap[label] = labelCounter++;
        }
    }

    vector<int> numericLabels(labels.size());
    transform(labels.begin(), labels.end(), numericLabels.begin(),
        [&](const string& label) { return labelMap[label]; });

    Mat hogMat(hog_features.size(), hog_features[0].size(), CV_32FC1);
    for (size_t i = 0; i < hog_features.size(); ++i) {
        for (size_t j = 0; j < hog_features[i].size(); ++j) {
            hogMat.at<float>(i, j) = hog_features[i][j];
        }
    }

    Ptr<TrainData> data = TrainData::create(hogMat, ROW_SAMPLE, Mat(numericLabels));
    data->setTrainTestSplitRatio(0.8, true);
```

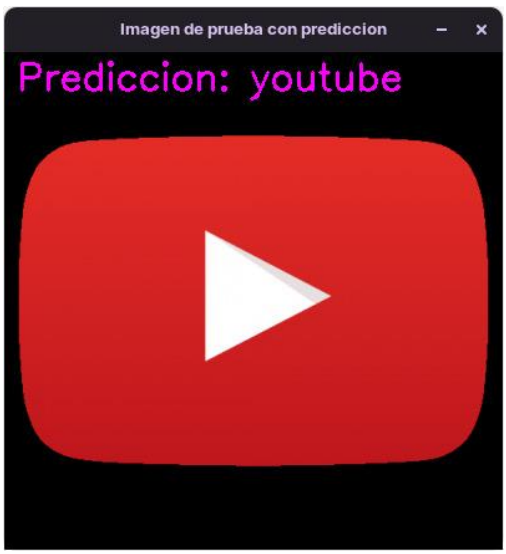
```cpp
    Ptr<SVM> svm = SVM::create();
    svm->setKernel(SVM::POLY);
    svm->setDegree(3);
    svm->setGamma(0.1);
    svm->setC(1);
    svm->train(data->getTrainSamples(), ROW_SAMPLE, data->getTrainResponses());

    Mat predictions;
    svm->predict(data->getTestSamples(), predictions);

    int correct = 0;
    for (int i = 0; i < predictions.rows; ++i) {
        if (predictions.at<float>(i, 0) == data->getTestResponses().at<int>(i, 0)) {
            correct++;
        }
    }
    float accuracy = (float)correct / predictions.rows;
    cout << "Precisión del modelo SVM: " << accuracy << endl;

    saveSVM(svm, "svm_classifier.xml");
```

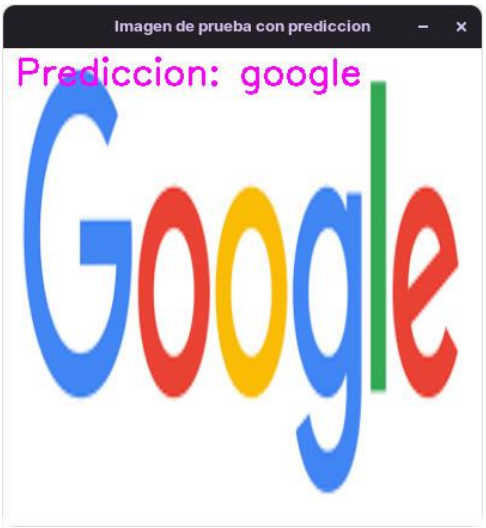
```cpp
Mat i = imread(test_image_path);
vector<float> test_descriptor = HOG(test_image_path);

Mat test_mat(1, test_descriptor.size(), CV_32FC1);
memcpy(test_mat.data, test_descriptor.data(), test_descriptor.size() * sizeof(float));

normalize(test_mat, test_mat, 0, 1, NORM_MINMAX);

float prediction = svm->predict(test_mat);

string predicted_label;
for (const auto& pair : labelMap) {
    if (pair.second == (int)prediction) {
        predicted_label = pair.first;
        break;
    }
}

resize(i, i, Size(400, 400));
putText(i, "Prediccion: " + predicted_label, Point(10, 30), FONT_HERSHEY_SIMPLEX, 1.0, Scalar(255, 0, 255), 2);
imshow("Imagen de prueba con prediccion", i);
```

```
    int TP = 0, TN = 0, FP = 0, FN = 0;
    int actual_label = labelMap[predicted_label];
    for (int j = 0; j < data->getTestResponses().rows; ++j) {
        int actual = data->getTestResponses().at<int>(j, 0);
        int predicted = predictions.at<float>(j, 0);
        if (actual == actual_label && predicted == actual_label) {
            TP++;
        } else if (actual != actual_label && predicted != actual_label) {
            TN++;
        } else if (actual != actual_label && predicted == actual_label) {
            FP++;
        } else if (actual == actual_label && predicted != actual_label) {
            FN++;
        }
    }

    visualizeConfusionMatrix(TP, TN, FP, FN, predicted_label);

    waitKey(0);
    destroyAllWindows();

    return 0;
}
```

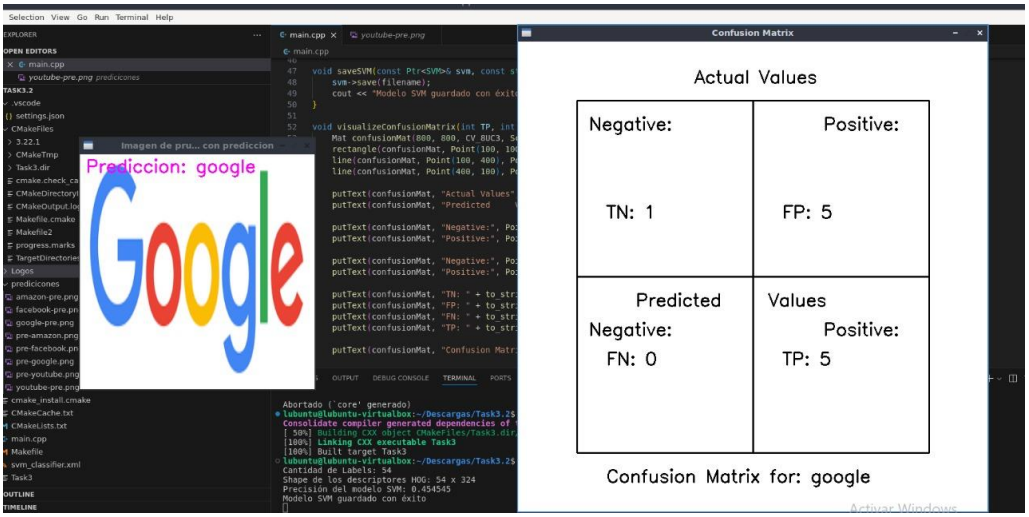
Resultados

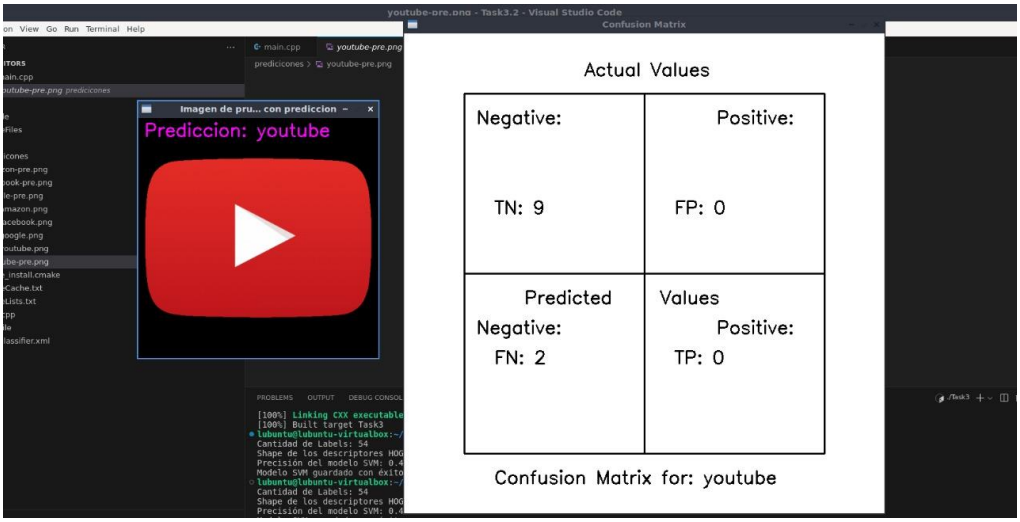






Matriz de confusión





Link al Video Demostrativo:

https://estliveupsedu-my.sharepoint.com/:v:/g/personal/kpaltin_est_ups_edu_ec/Ee9USDZ3mf9KhfgtzbF4MVkBHaF6hL3nHkp9hg0j862j8w?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=qDM4mt

Referencias:

- Malick, A. (2020). Histogram of oriented gradients (HOG) for multiclass image classification and image recommendation. Medium. Recuperado de: https://medium.com/swlh/histogram-of-oriented-gradients-hog-for-multiclass-image-classification-and-image-recommendation-cf0ea2caaae8
- KOUSTUBHK. (2021). Logos [Data set]. Popular brand Logos - Image Dataset. Recuperado de https://www.kaggle.com/datasets/kkhandekar/popular-brand-logos-image-dataset