Tema:

Practica 3.2

Docente:

PHD. Vladimir Robles

Materia:

Vision por Computador

Nombre:

Kevin Japa

Pedro Orellana

Fecha:

3 de Julio de 2024

El histograma de gradientes orientados (HoG)

Es un descriptor de características que se utiliza para definir una imagen por las intensidades de los píxeles y las intensidades de los gradientes de los píxeles. Los gradientes definen los bordes de una imagen, por lo que la extracción del descriptor de características HoG es lo mismo que la extracción de bordes. El histograma de gradientes orientados genera gradientes en cada punto de la imagen, lo que proporciona invariancia a las oclusiones, la iluminación y los cambios de expresión.

Pasos para Calcular el Hog:

- **Gradientes de la Imagen:** Calcular los gradientes de la imagen en la dirección x, y. Estos gradientes capturan cambios en la intensidad de la imagen.

- **Orientación y Magnitud:** Con los gradientes Calculados, calculamos la magnitud y la orientación de cada píxel.

- **Celda de Gradientes:** Dividimos la imagen en celdas pequeñas. Dentro de cada celda, crear un histograma de las orientaciones de los gradientes.

- **Bloques Normalizados:** Agrupamos las celdas en bloques más, Normalizamos el histograma en cada bloque para hacer el descriptor menos sensible a los cambios en iluminación y contraste.

- **Descriptor HoG:** Concatenamos los histogramas normalizados de todos los bloques para formar el descriptor final de la imagen.

Codigo Utilizado:

Creacion y Entrenamiento del Modelo

```
#include <opencv2/opencv.hpp>
#include <opencv2/ml.hpp>
#include <iostream>
#include <vector>
#include <filesystem>
#include <map>
#include <fstream>
#include <numeric>

using namespace cv;
using namespace cv::ml;
using namespace std;
namespace fs = std::filesystem;
```

```
void loadImagesAndLabels(const string& directory, vector<Mat>& images,
vector<int>& labels, map<string, int>& labelMap, map<int, string>&
reverseLabelMap) {
    int currentLabel = 0;
    for (const auto& entry : fs::directory_iterator(directory)) {
        if (entry.is_directory()) {
            string className = entry.path().filename().string();
            if (labelMap.find(className) == labelMap.end()) {
                labelMap[className] = currentLabel;
                reverseLabelMap[currentLabel] = className;
                currentLabel++;
            }
            int label = labelMap[className];
            for (const auto& imgEntry : fs::directory_iterator(entry.path()))
{
                if (imgEntry.path().extension() == ".jpg" ||
imgEntry.path().extension() == ".png") {
                    Mat img = imread(imgEntry.path().string(),
IMREAD_GRAYSCALE);
                    if (!img.empty()) {
                        Mat resizedImg;
                        resize(img, resizedImg, Size(64, 64));
                        images.push_back(resizedImg);
                        labels.push_back(label);
                    }
                }
            }
        }
    }
}
```

Con esta funcion cargarmos las imágenes con las etiquetas del dataset que contiene 10 imágenes de cada una de las categorias, el dataset, tiene las 4 variables: Google, Neflix, Zoom, Tiktok, que se realizo una recolecion de imágenes de internet para el entrenamiento del Modelo.
Luego Iteramos sobre la carpeta del dataset para dar la etiqueta a cada subdirectorio, cabe recalcar que las imágenes las leemos en escala de grises y redimencionamos la imgenes para tenerlas en un tamaño general (64 x 64) y almacenamos la imagen en un vector, junto con sus respectivas etiquetas.

```
int main() {
    string trainingDir = "dataset-logos";
    vector<Mat> images;
    vector<int> labels;
    map<string, int> labelMap;
    map<int, string> reverseLabelMap;
    loadImagesAndLabels(trainingDir, images, labels, labelMap,
reverseLabelMap);
```

Definimos el Directorio para el entrenamiento y utilizamos la funcion generada anteriormente(loadImagesAndLabels) cargamos las imágenes con sus label, con los vectores.

```
    HOGDescriptor hog(
        Size(64, 64),
        Size(16, 16),
        Size(8, 8),
        Size(8, 8),
        9
    );
```

Configuramos el Descriptor de HoG con los parametros, Tamano de ventana, Bloque, el paso del Bloque, la Celda y el Numero de bins.

```
    vector<Mat> descriptors;
    for (const auto& image : images) {
        vector<float> descriptor;
        hog.compute(image, descriptor);
        descriptors.push_back(Mat(descriptor).clone());
    }

    Mat trainingData;
    vconcat(descriptors, trainingData);
    trainingData = trainingData.reshape(1, descriptors.size());
    Mat trainingLabels(labels);
```

Luego de Calcular los descriptores de cada imagen, almacenamos en un vector de matrices, concatenamos en una sola matriz y las reorganizamos para que cada fila concatene para cada descriptor

```
    Ptr<SVM> svm = SVM::create();
    svm->setType(SVM::C_SVC);
    svm->setKernel(SVM::LINEAR);
    svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6));
    svm->train(trainingData, ROW_SAMPLE, trainingLabels);

    svm->save("svm_logo_model.yml");
```

Creamos el modelo SVM, y configuramos los parametros del modelo, en este caso usamos de tipo SVM, con el kernel Lineal, y entrenamos el modelo utilizando el dataset cargado anteriormente con sus etiquetas.
Este modelo entrenado lo guardamos en un archivo xml, para luego utilizarlo en la segunda fase de las predicciones.

```
    ofstream labelFile("label_map.txt");
    for (const auto& pair : reverseLabelMap) {
        labelFile << pair.first << " " << pair.second << endl;
    }
    labelFile.close();
```

```
    cout << "Modelo SVM entrenado y guardado en svm_logo_model.yml" << endl;
    cout << "Mapa de etiquetas guardado en label_map.txt" << endl;
    cout << "Tamaño del descriptor: " << trainingData.cols << endl;
```

Guardamos el modelo de las etiquetas, para usarlo en la segunda parte

```
    Mat predictedLabels;
    svm->predict(trainingData, predictedLabels);

    int numClasses = reverseLabelMap.size();
    Mat confusionMatrix = Mat::zeros(numClasses, numClasses, CV_32S);
    for (int i = 0; i < trainingLabels.rows; i++) {
        int actual = trainingLabels.at<int>(i);
        int predicted = predictedLabels.at<float>(i);
        confusionMatrix.at<int>(actual, predicted)++;
    }

    cout << "Matriz de confusión:" << endl;
    cout << confusionMatrix << endl;

    vector<int> truePositives(numClasses, 0);
    vector<int> falsePositives(numClasses, 0);
    vector<int> falseNegatives(numClasses, 0);

    for (int i = 0; i < numClasses; i++) {
        truePositives[i] = confusionMatrix.at<int>(i, i);
        falsePositives[i] = sum(confusionMatrix.col(i))[0] - truePositives[i];
        falseNegatives[i] = sum(confusionMatrix.row(i))[0] - truePositives[i];
    }

    double accuracy = sum(confusionMatrix.diag())[0] / trainingLabels.rows;
    cout << "Accuracy: " << accuracy << endl;

    for (int i = 0; i < numClasses; i++) {
        double precision = (truePositives[i] + falsePositives[i]) > 0 ?
truePositives[i] / double(truePositives[i] + falsePositives[i]) : 0;
        double recall = (truePositives[i] + falseNegatives[i]) > 0 ?
truePositives[i] / double(truePositives[i] + falseNegatives[i]) : 0;
        cout << "Clase " << reverseLabelMap[i] << ": Precision = " <<
precision << ", Recall = " << recall << endl;
    }

    return 0;
}
```

Finalmente realizamos el calculo de las metricas del modelo entrenado, mandamos a generar la matriz de confusion, para comparar las etiquetas predichas con las reales. Y las imprimimos en consola.

Resultados Obtenidos

```
(base) kevinjapa@MacBook-Pro-de-Kevin TareaHogLogos % make run
./vision.bin
Modelo SVM entrenado y guardado en svm_logo_model.yml
Mapa de etiquetas guardado en label_map.txt
Tamaño del descriptor: 1764
Matriz de confusión:
[11, 0, 0, 0;
 0, 11, 0, 0;
 0, 0, 8, 0;
 0, 0, 0, 9]
Accuracy: 1
Clase google: Precision = 1, Recall = 1
Clase netflix: Precision = 1, Recall = 1
Clase zoom: Precision = 1, Recall = 1
Clase tiktok: Precision = 1, Recall = 1
(base) kevinjapa@MacBook-Pro-de-Kevin TareaHogLogos %
```

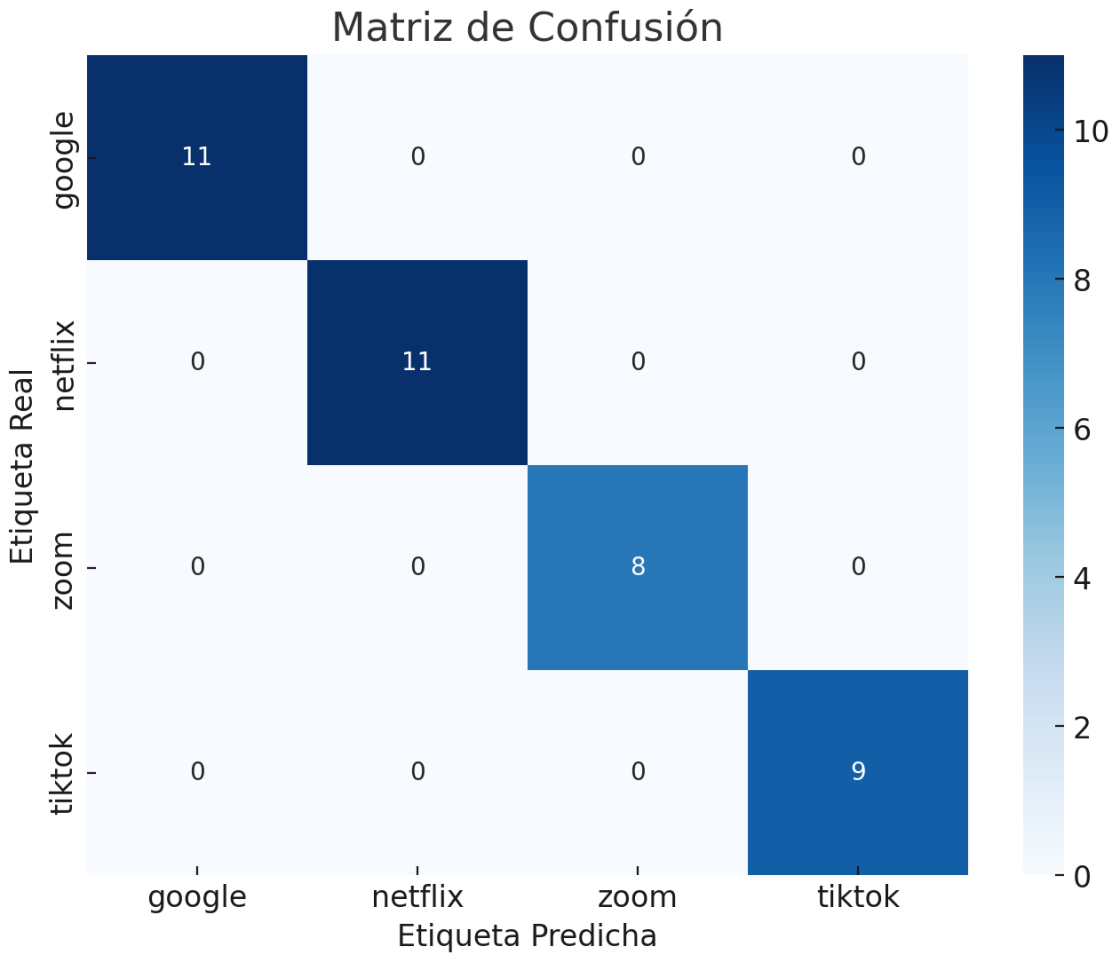
El tamano del descriptor es de 1764 caracteristicasm que se extrayeron del descriptor

Google: 11 Imágenes de google fueron clasificadas correctamente

Netflix: 11 imágenes fuero clasificadas

Zoom: 8 imágenes fueron clasificas

TikTok: 9 imágenes fueron clasificadas



Con respecto a las metricas de calidad tenemos que el modelo clasifica de manera correcta, al igual de cada una de las etiquetas

Cargar el Modelo y Realizar Predicciones

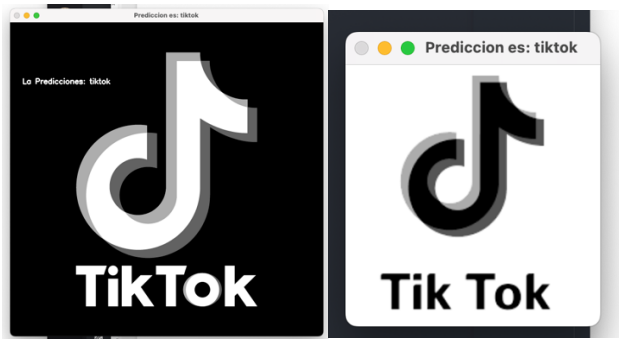
```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/ml/ml.hpp>
#include <opencv2/objdetect.hpp>
#include <fstream>
#include <map>

using namespace std;
using namespace cv;
using namespace ml;

map<int, string> loadLabelMap(const string& filename) {
    map<int, string> labelMap;
    ifstream file(filename);
    int label;
    string className;
    while (file >> label >> className) {
        labelMap[label] = className;
    }
    return labelMap;
}

Casgamos el mapa de etiquetas desde el archivo de texto y lo almacenamos en un
mapa de tipo (map<int, string>).

int main() {
    string imagePath = "tiktok.png";
    Mat image = imread(imagePath, IMREAD_GRAYSCALE);
    Mat resizedImage;
    resize(image, resizedImage, Size(64, 64));

Cargamos la imagen con la que se realizara la nueva predccion en escala de
grises, y la imagen la redimencionamos al tamano que recibe el descriptor, que
es una imagen de 64 x 64.

    Ptr<SVM> svm = SVM::load("svm_logo_model.yml");

Cargamos el modelo svm que se genero en el modelo entrenado(.xml)

    HOGDescriptor hog(
        Size(64, 64),
        Size(16, 16),
        Size(8, 8),
        Size(8, 8),
        9
    );
```

```
Configuramos el descriptor HoG con los parametros que se utilizaron para el
entrenamiento del modelo.

    vector<float> descriptor;
    hog.compute(resizedImage, descriptor);

    Mat descriptorMat = Mat(descriptor).reshape(1, 1);
    descriptorMat.convertTo(descriptorMat, CV_32F);

Calculamos los descriptores de la imagen que se va a realizar la prediccion y
realizamos la prediccion con el modelo cargado

    int predictedLabel = svm->predict(descriptorMat);
    map<int, string> labelMap = loadLabelMap("label_map.txt");
    if (labelMap.find(predictedLabel) != labelMap.end()) {
        cout << "La imagen ha sido clasificada como: " <<
labelMap[predictedLabel] << endl;
    } else {
        cout << "Etiqueta desconocida: " << predictedLabel << endl;
    }

    waitKey(0);
    destroyAllWindows();

    return 0;
}
```

Predicciones

Etiqueta Google

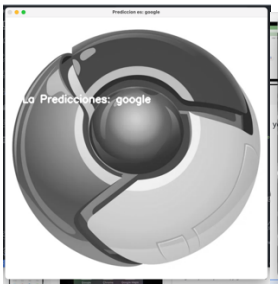
Predicciones Correctas



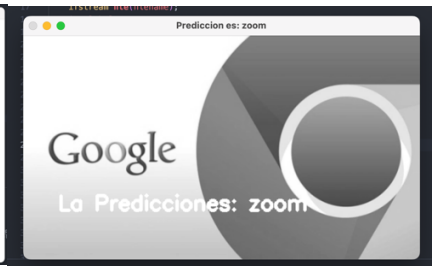
Predicciones Incorrectas

Etiqueta google

Predicciones Correctas

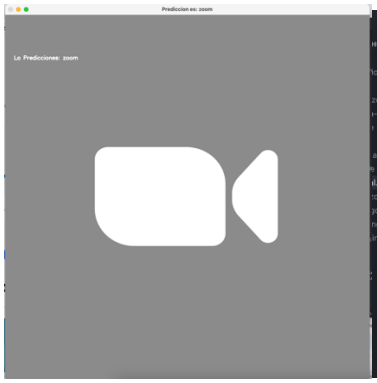


Predicciones Incorrectas

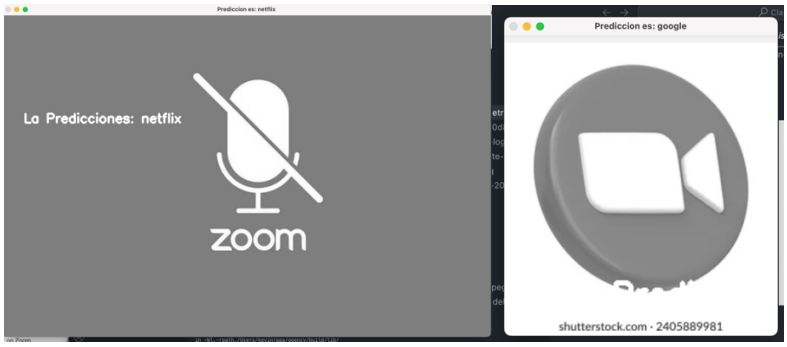


Etiqueta Zoom

Predicciones Correctas

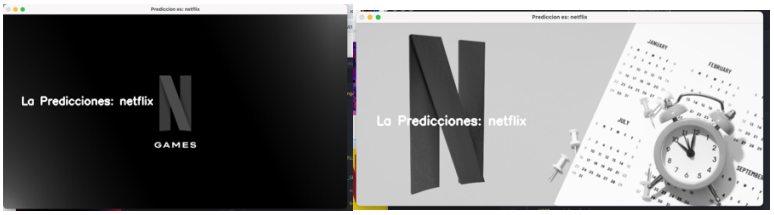


Predicciones Incorrectas

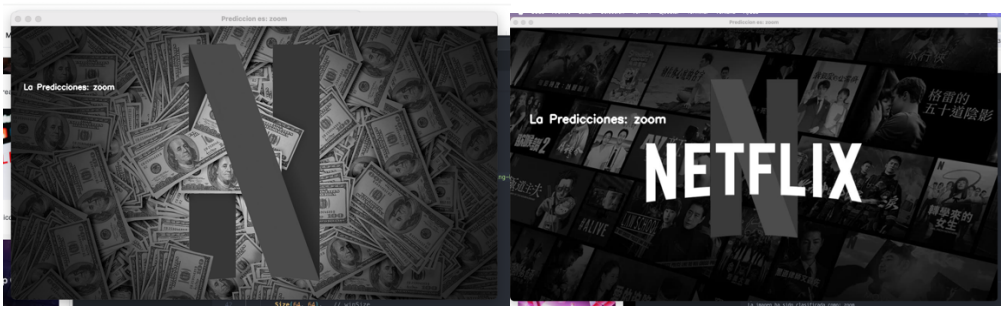


Netflix

Predicciones Correctas



Predicciones Incorrectas



Referencias

Malick, A. (2020, julio 15). *Histogram of oriented gradients (HOG) for multiclass image classification and image recommendation*. The Startup. https://medium.com/swlh/histogram-of-oriented-gradients-hog-for-multiclass-image-classification-and-image-recommendation-cf0ea2caaae8

(S/f). Sciencedirect.com. Recuperado el 4 de julio de 2024, de https://www.sciencedirect.com/topics/computer-science/histogram-of-oriented-gradient#:~:text=involved%20in%20classification.-,Histogram%20of%20oriented%20gradients%20(HoG)%20is%20a%20f

eature%20descriptor%20used,the%20same%20as%20extracting%20edg
es.