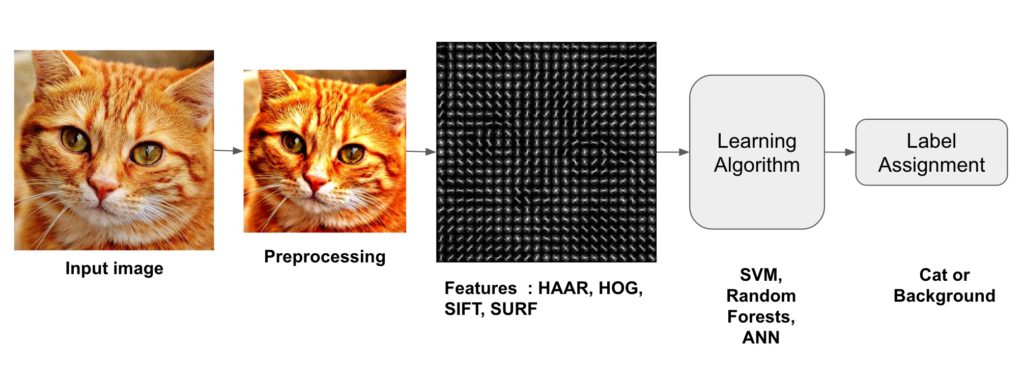
**Reconocimiento Facial basado en Deep Learning usando Dlib**

En este laboratorio usaremos técnicas avanzadas de reconocimiento facial de “deep learning” basadas en Dlib. El modelo de Dlib es más preciso que el de la librería OpenFace, que es mejor en el método de “Eigenfaces”, “Fisherfaces” y “LBP”. Este tiene una presición de 99.38% en benchmark que son las mediciones que se les hace a los modelos para saber que tan exactos son.

**Abriendo la caja negra**

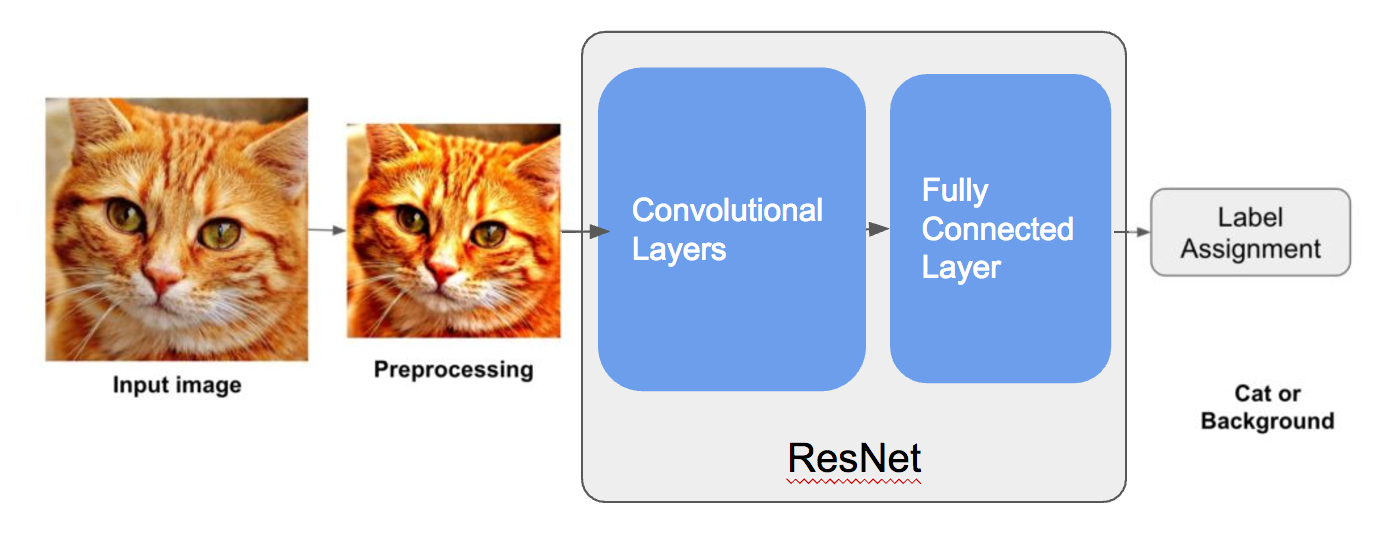
Establecer métodos de Deep learning es establecer una caja negra, los detalles de como realiza sus cálculos y demás la red neuronal dependen de los ajustes de los hiperparámetros. Hablaremos un poco para entender como el reconocimiento facial funciona.

Sin embargo, es importante refrescar que hace un método tradicional de clasificación de imagenes como se muestra en la siguiente figura.



En un clasificador de imagenes tradicional, el proceso de entrada a salida se le llama pipeline o lo mismo la línea de ejecución de inicio a fin. Inicialmente se convierte la imagen en un vector de características (o lo equivalente a un punto) en un espacio dimensional superior. Esto es hecho calculando el vector de características por HOG (Histograma de Gradientes) para una sección de la imagen. Una ves la imagen fue representada como un punto dimensional superior en el espacio, podemos utilizar algoritmos de aprendizaje como SVM (Máquinas de Soporte Vectorial) para particional los espacios usando hiperplanos que separan los puntos representando las diferentes clases.

Aun cuando la superficie del modelo superior de Deep Learning luce diferente hay similitudes conceptuales. La siguiente figura revela el modelo de Deep Learning utilizado por el reconocedor facial de Dlib. La arquitectura está basada en la popular red ResNet.



Como la mayoría de las CNN (Redes Neuronales Convolucionales), ResNet contiene un banco de redes convolucionales seguidas de una capa totalmente conectada.

El banco de redes convolucionales produce un vector de características de alta dimensión espacial justo como el descriptor HOG. Así que, ¿Cuál es la diferencia entre el banco de capas convolucionales y el descriptor HOG? Las diferencias más importantes son:

* HOG es un descriptor fijo. Existe una receta exacta para calcular el descriptor, por otro lado, un banco de redes convolucionales contiene varios filtros convolcionales. Estos filtros son aprendidos por la data. Así que como HOG, ellos se adaptan basados del problema a la mano.
* La red convolucional totaltemte conectada (FC) hace el mismo trabajo que el clasificador SVM en acercamientos similares. Este clasifica las características del vector. De hecho, algunas veces la capa FC es reemplazada por una SVM.

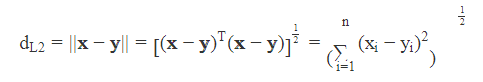
Antes de seguir adelante, evaluemos un concepto llamado métrica.

**¿Qué es la métrica?**

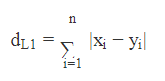
Usualmente, cuando la palabra “distancia” entre dos puntos fue mencionada, también la distancia euclidiana entre ellos. Por ejemplo, para una distancia en puntos en el espacio tenemos los puntos ( 1, 0, 1) and ( 1, 3, 5) y su distancia es



En general si tenemos un vector dimensional xx y yy, la distancia L2 (también llamada distancia euclideana) esta dada por:



Sin embargo, en matemáticas, la distancia (conocido como métrica) tiene una definición mucho mayor. Por ejemplo, una distancia diferente es llamada L1. Esta es la suma de los valores absolutos de los elementos entre dos vectores.



Las siguientes reglas definen cuando una función envuelve dos vectores que pueden ser llamados métrica. Una distancia mapeada como d(xx, yy) es llamada métrica si.

* La distancia entre dos puntos es mayor o igual a cero d(xxThe distance between any two points is greater than or equal to zero



* Un punto tiene distancia cero de si mismo. 
* La distancia desde xx a yy is la misma distancia desde yy a xx.  
  
* Inecuación triangular: Para cada uno de los siguientes tres puntos xx, yy y zz, la siguiente inecuación es correcta.   
  

**La métrica en Deep Learning**

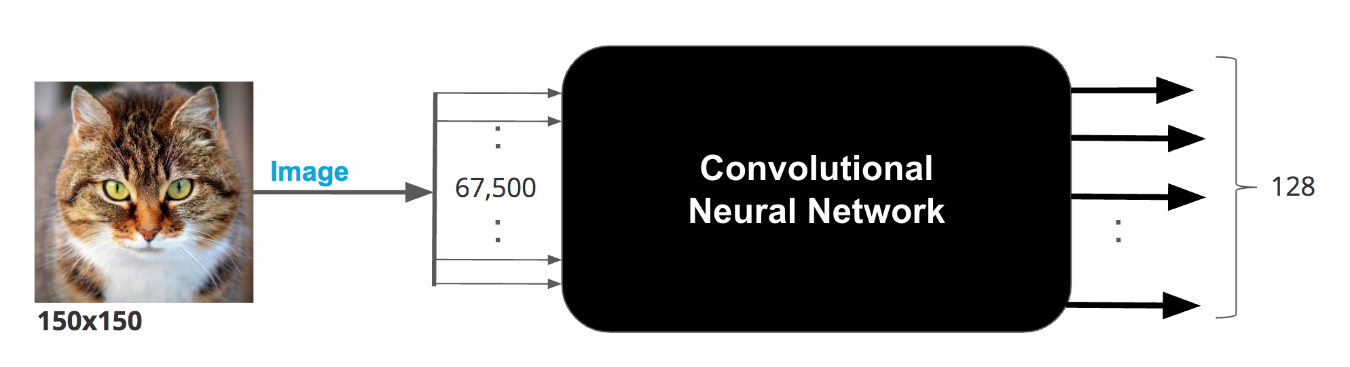
Cualquier imagen puede ser vectorizada simplemente almacenando los pixels en un largo vector. Este vector representa un punto en el espacio dimensional superior. Sin embargo, este espacio no es muy bueno para medir distancias. En una aplicación de reconocimiento facial, los puntos representado dos diferentes imágnes de la misma persona pueden ser muy distantes y los puntos representando imágenes de dos diferentes personas pueden ser muy cercanos.

Cuando se usa una técnica llamada PCA (Componentes de Analisis Principal) para reducir dimensionalidad, esperamos que las distancias a reducir dimension especial sean las más significativas. Similarmente cuando la “discriminación lineal de Fisher” se ejecuta buscamos encontrar donde las distancias se hacen significativas. Ambos acerpcamientos trabajan a cierta extensión pero el desempeño no siempre es excepcional.

La métrica, en deep learning, es una clase de técnica que usa deep learning para aprender que una menor dimensionalidad de metrica especial donde dos imágenes representadas por puntos como imágenes de la misma clase son enclaustraas juntas y las imágenes de diferente clase son catalogadas a parte. Conceptualmente, el objetivo es muy similar a la “discriminante lineal de Fisher” pero en práctica el resultado es vastamente superior porque en direccionar la reducción de dimensionalidad de los pixeles en el espacio, la red convolucional primeramente es calculada con características significativas las cuales son implícitas usando la creación de la métrica en el espacio.

Tenemos entonces que podemos usar la misma arquitectura CNN que utilizamos para clasificación de imágenes para Deep learning

En la siguiente figura la red convolucional que está entrenada para una imagen de 150x150 (la cual es la misma para el vector de tamaño150x150x3 = 67,500 ) y la probabilidad de salida corresponde a 128 animales de diferentes clases está muy lejana.

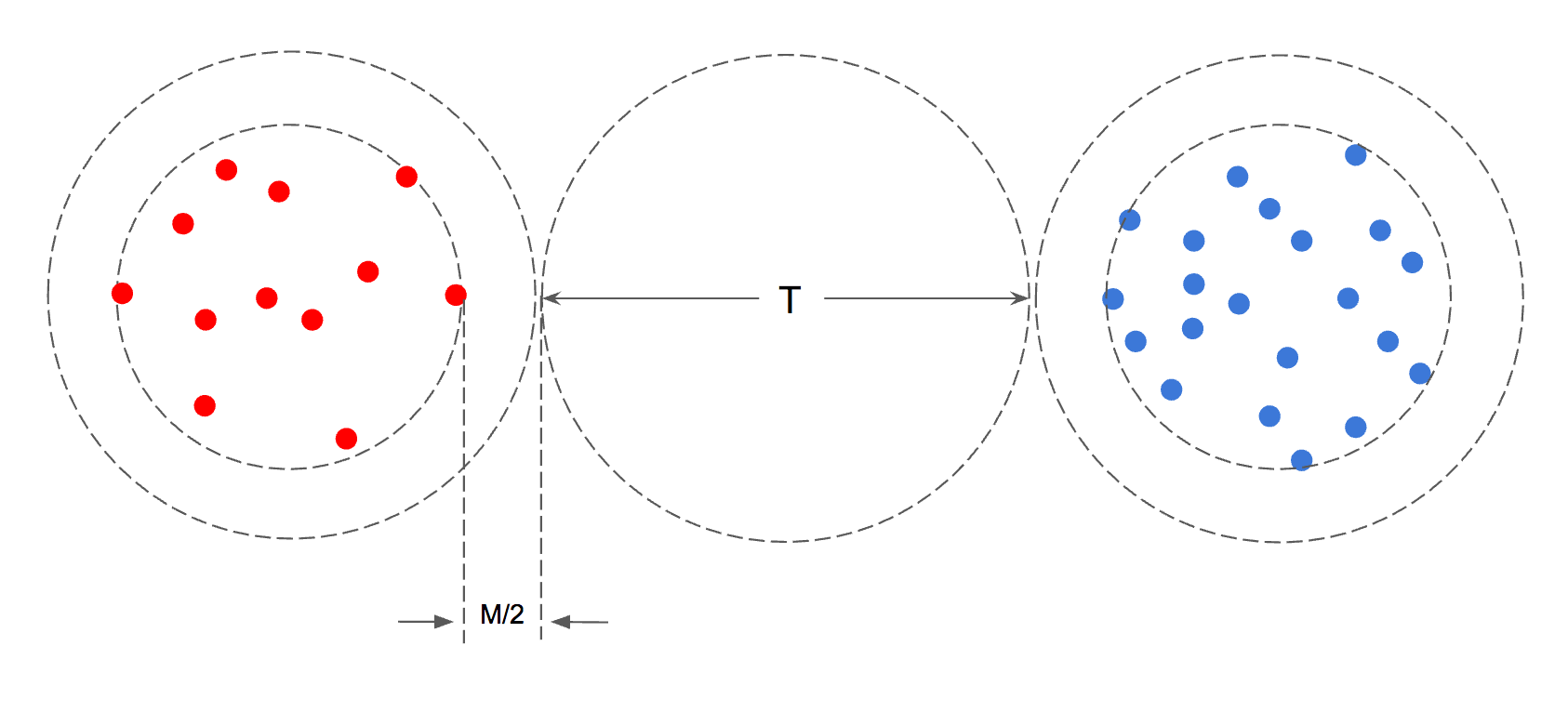


En otras palabas, ud. puede insertar una imagen y la salida es un punto de 128 dimensiones en el espacio. Si ud quiere encontrar cuan cerca está relacionada una imagen con la otra ud puede simplemente pasar ambas imágenes a la red convolucional y obtener dos puntos en el espacio dimensional de 128. Ud. puede comparar dos puntos usando simple distancia euclidiana entre ellos.

**La pérdida métrica**

Millones de imagenes son tipicamente utilizadas para entrenar un sistema de producción de CNN. Obviamente estas millones de imágenes no pueden ser insertadas simmultaneamente a la red convolucional para actualizar los pesos (w). El entrenamiento es realizado iterativamente usando un pequeño batch de imágenes al tiemo. Este pequeño batch se le llama mini-batch.

Como se menciono anteriormente en la sección previa necesitamos definir una función de pérdida en donde la red convolucional apunta a una salida de 128 dimensiones espaciales. La función de pérdiad es definida sobre todos los pares de imágenes del mini-batch.



La perdida métrica definida por Dlib. Para simplicidad el concepto es mostrado en dos parámetros 1) Umbral (T) y 2) Margen. Los puntos azules y rojo representan imágenes de dos diferentes clases. Para una pérdida métrica que sea 0, la distancia máxima entre uno de los dos puntos de la misma clase debe ser (T – M) y la mínima distancia entre dos puntos de diferentes clases debe ser (T + M)

Siendo p1 y p2 represntando dos puntos correspondientes a dos imagens I1 e I2 in el espacio dimensional de 128 posiciones. Si las imágenes corresponden a la misma clase entonces la pérdida está dada por:

max(0,||p1−p2||−T+M)

Por otro lado, si las imagenes I1 e I2 tienen dos diferentes clases etiquetadas su contribución a la función de pérdida es:

max(0,T−||p1−p2||+M)

La figura anterior muetra como la función de pérdidad prefiere embeberlas en donde las imagenes de las mismas clases son arregladas en modo cluster juntas y las imagenes de diferentes clases son separadas por un largo margen.

**Minería Negativa**

En un mini-batch hay muchos pares no encontrados (imagenes de diferentes clases) que similitudes de imagenes de parejas (imagenes de la misma clase). Es importante tomar este desbalance en cuenta cuando se calcula la función de pérdida de la métrica. Si hay N pares que coinciden y comparten la misma clase en un mini-batch entonces el algoritmo incluye solamente la Nesima peor par no coincidente en la computación de la función de pérdida. En otras palabras, ingresar minería negativa en el mini-batch toma los peores pares que no hacen coincidir.

**Entrenamiento**

El reconocedor facial de Dlib utiliza acerca de 3M de imagenes con 7845 diferentes identidades. Estas imagenes fueron obtenidas combinando y limpiando los datasets disponibles. Es un proceso tedioso que le tomo a Davis King, el autor de la librería, mucho tiempo. El entrenamiento tomo cerca de 1 día en un GPU Titan X.

Afortunadamente el modelo es compartido pro Davis y no necesitamos reentrenarlo.

**Inscripción**

La inscripción de una persona es simplemente realizada pasando algunas imagenes de la persona a traves de la red obtenida para 128 caracteristicas descriptivas dimensionales correspondientes a cada imagen, dada la imagen de la misma persona, podemos verificar si la misma persona por verificar su distancia está inscrita con la cara entre las 128 dimenciones del espacio.

Nota: Antes que nada debe tener la siguiente arquitectura de carpetas…

Recordar que la carpeta common es del laboratorio anterior y los archivos dentro de la carpeta python serán los que realizará en este laboratorio.

Folder\_De\_Proyecto

common

face\_recognition

models

dlib\_face\_recognition\_resnet\_model\_v1.dat

python

enrollDlibFaceRec.py

testDlibFaceRecVid.py

data

videos

video1.mp4

images

faces

<carpeta\_nombre\_de\_persona1>

Img01.jpg, img02.jpg, img03.jpg, etc.

<carpeta\_nombre\_de\_persona2>

Img01.jpg, img02.jpg, img03.jpg, etc.

*Archivo enrollDlibFaceRec.py*

**import os**

**import dlib**

**import cv2**

**import numpy as np**

**try:**

**import cPickle # Python 2**

**except ImportError:**

**import \_pickle as cPickle # Python 3**

Inicializamos el detector de caras de Dlib, el detector de marcas facials y el reconocedor de objetos en redes neuronales.

**# Path del modelo de puntos facials**

**PREDICTOR\_PATH = '../../common/shape\_predictor\_68\_face\_landmarks.dat'**

**FACE\_RECOGNITION\_MODEL\_PATH = '../models/dlib\_face\_recognition\_resnet\_model\_v1.dat'**

**# inicialzia el detector facial, landmark y reconocedor**

**faceDetector = dlib.get\_frontal\_face\_detector()**

**shapePredictor = dlib.shape\_predictor(PREDICTOR\_PATH)**

**faceRecognizer = dlib.face\_recognition\_model\_v1(FACE\_RECOGNITION\_MODEL\_PATH)**

Las imágenes de personas que vamos ha hacer enroll están estructuradas de la siguiente manera:

carpeta\_de\_proyecto/data/images/faces en este folder ponemos subfolders de nombres de personas, todas las que queramos detector. Tendremos este mapeo de imágenes y su correspondiente etiqueta para usarla finalmente como prueba.

**# Preparamos el training data**

**# La data esta organizada en estructura**

**# faces tiene subfolders y cada subfolder**

**# tiene informacion de cada persona**

**faceDatasetFolder = '../data/images/faces'**

**# lee subfolder en "faces"**

**subfolders = []**

**for x in os.listdir(faceDatasetFolder):**

**xpath = os.path.join(faceDatasetFolder, x)**

**if os.path.isdir(xpath):**

**subfolders.append(xpath)**

**# nameLabelMap es el diccionario en el cual las llaves**

**# y valores enteros están etiquetados como persona**

**# esta etiqueta contiene caracteres enteros corespondiente a imagenes**

**nameLabelMap = {}**

**labels = []**

**imagePaths = []**

**for i, subfolder in enumerate(subfolders):**

**for x in os.listdir(subfolder):**

**xpath = os.path.join(subfolder, x)**

**if x.endswith('jpg'):**

**imagePaths.append(xpath)**

**labels.append(i)**

**nameLabelMap[xpath] = subfolder.split('/')[-1]**

El proceso de inscripción es de 1 a 1. Convertir la imagen de RGB a BGR porque Dlib usa BGR como formato por defecto.

**# Procesa las imagenes una por una**

**# Almacenaremos los descriptors facials en un ndarray (faceDescriptors)**

**# y sus correspondientes etiquetas en un diccionario (index)**

**index = {}**

**i = 0**

**faceDescriptors = None**

**for imagePath in imagePaths:**

**print("processing: {}".format(imagePath))**

**# read image and convert it to RGB**

**img = cv2.imread(imagePath)**

**# detectar caras**

**faces = faceDetector(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))**

Detecta caras en una imagen. Para cada cara computaremos un descriptor facial. Note que hemos asumido que una imagen sencilla tendrá caras de caras de una persona. Pero in un escenario práctico ud tendrá imágenes de multiples personas en una imagen. En tal caso ud tiene que mantener el mapeo de cada descrptor facial correspondiendo su etiqueta para cada landmark facial que se quiera.

**print("{} cara(s) encontrada".format(len(faces)))**

**# para cad acara encontrada**

**for k, face in enumerate(faces):**

**# Encuentra la marca facial para cad acara**

**shape = shapePredictor(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB), face)**

**# Convierte marca facial de Dlib en format de lista de (x,y) puntos**

**landmarks = [(p.x, p.y) for p in shape.parts()]**

Calcula el descriptor facial usando landmarks. Este es una arreglo multidimensional vectorial (128) representando una cara. Convertiremos un descriptor facial de Dlib en formato a un arreglo numpy para fácil cálculo matemático y computación de escritura en el disco.

Para cada descriptor facial salvaremos cada correspondiente etiqueta. En orden de que el descriptor facial para cada cara en faceDescriptors sea un arreglo de numpy de filas y parecido al orden de etiquetas en la variable index.

**# Computa el descriptor facial usando una red neuronal definida en dlib.**

**# Es un vector de 128D que describe la cara en img identificado por forma.**

**faceDescriptor = faceRecognizer.compute\_face\_descriptor(img, shape)**

**# Convierte el face descriptor de dli ben format a lista, entonces a un arreglo de numpy**

**faceDescriptorList = [x for x in faceDescriptor]**

**faceDescriptorNdarray = np.asarray(faceDescriptorList, dtype=np.float64)**

**faceDescriptorNdarray = faceDescriptorNdarray[np.newaxis, :]**

**# Stack de face descriptors de 1x128 para cada cara en imagen como filas**

**if faceDescriptors is None:**

**faceDescriptors = faceDescriptorNdarray**

**else:**

**faceDescriptors = np.concatenate((faceDescriptors, faceDescriptorNdarray), axis=0)**

**# guarda la etiqueta de esta cara en índice. La usaremos luego para identificar**

**# el nombre de la persona correspondiente al descriptor almacenado en NumPy**

**index[i] = nameLabelMap[imagePath]**

**i += 1**

Guarda el descriptor en el disco.

**# Salva en disco**

**np.save('descriptors.npy', faceDescriptors)**

**# índice de imágenes de archivos en el mismo orden del descriptor facial**

**with open('index.pkl', 'wb') as f:**

**cPickle.dump(index, f)**

Corra el enroll desde una terminal dentro de la carpeta python como:

**python enrollDlibFaceRec.py**

Una vez entrenado el sistema puede abrir el video o una captura cambiando la siguiente línea y copiando el siguiente código en la línea

**cam = cv2.VideoCapture("../data/videos/video1.mp4") # para un video llamado video 1**

**cam = cv2.VideoCapture(0) # para entrada de cámara**

Archivo testDlibFaceRecVid.py

**#!/usr/bin/python**

**#**

**import os,sys,time**

**import dlib**

**import cv2**

**import numpy as np**

**# Ruta al landmark y modelo de face recognition**

**PREDICTOR\_PATH = '../../common/shape\_predictor\_68\_face\_landmarks.dat'**

**FACE\_RECOGNITION\_MODEL\_PATH = '../models/dlib\_face\_recognition\_resnet\_model\_v1.dat'**

**SKIP\_FRAMES = 1**

**THRESHOLD = 0.5**

**# Inicializa el detector facial, landmarks y reconocedor facial**

**faceDetector = dlib.get\_frontal\_face\_detector()**

**shapePredictor = dlib.shape\_predictor(PREDICTOR\_PATH)**

**faceRecognizer = dlib.face\_recognition\_model\_v1(FACE\_RECOGNITION\_MODEL\_PATH)**

**# carga descriptores e índice de archivos generados durante enroll**

**index = np.load('index.pkl')**

**faceDescriptorsEnrolled = np.load('descriptors.npy')**

**# crea un objeto de video captura**

**cam = cv2.VideoCapture(0)**

**count = 0**

**# procesa tramas de camara o video**

**while True:**

**t = time.time()**

**# captura trama**

**success, im = cam.read()**

**# salir si no puede leer una trama de cámara o video**

**if not success:**

**print('cannot capture input from camera')**

**break**

**# Procesaremos las tramas a intervalos**

**# of SKIP\_FRAMES to increase processing speed**

**if (count % SKIP\_FRAMES) == 0:**

**# convierte BGR a RGB**

**# porque Dlib utiliza ese formato**

**img = cv2.cvtColor(im, cv2.COLOR\_BGR2RGB)**

**# detect acaras en imagen**

**faces = faceDetector(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))**

**# Ahora que ha sido encontrado**

**for face in faces:**

**# Find facial landmarks for each detected face**

**shape = shapePredictor(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB), face)**

**# find coordinates of face rectangle**

**x1 = face.left()**

**y1 = face.top()**

**x2 = face.right()**

**y2 = face.bottom()**

**# Calcula el descriptor usando la red neuronal de dlib**

**# usando un facial landmark shape**

**faceDescriptor = faceRecognizer.compute\_face\_descriptor(img, shape)**

**# Convierte el face descriptor de dlib a lista y a un arreglod e numpy**

**faceDescriptorList = [m for m in faceDescriptor]**

**faceDescriptorNdarray = np.asarray(faceDescriptorList, dtype=np.float64)**

**faceDescriptorNdarray = faceDescriptorNdarray[np.newaxis, :]**

**# Calcula la distancia euclidiana entre el face descriptor y la cara detectada**

**# en la trama actual de todas las caras descritas cuando fue calculado en el enroll**

**distances = np.linalg.norm(faceDescriptorsEnrolled - faceDescriptorNdarray, axis=1)**

**# Calcula la distancia minima del indice de cada cara**

**argmin = np.argmin(distances) # indice**

**minDistance = distances[argmin] # distancia minima**

**# Dlib especifica en general que dos caras tienen descriptor de cara euclideana si**

**# la distancia entre ellas es menor de 0.6 es entonces la misma cara**

**# de lo contrario es diferente persona**

**# Este umbral varia dependiendo del número de imagenes activadas**

**# y varias variaciones (iluminacion, calidad de camara) entre**

**# imagenes inscriptas y peticiones de imagenes**

**# El umbral se especifico arriba y es 0.5**

**# Si la distancia minima es menor que el umbral**

**# encontrar el nombre de la persona en el indice**

**# de lo contrario la persona es “unknown”**

**if minDistance <= THRESHOLD:**

**label = index[argmin]**

**else:**

**label = 'unknown'**

**print("time taken = {:.3f} seconds".format(time.time() - t))**

**# Dibuja un rectangulo para la cara**

**cv2.rectangle(im, (x1, y1), (x2, y2), (0, 0, 255))**

**# Dibuja un circulo para la cara**

**center = (int((x1 + x2)/2.0), int((y1 + y2)/2.0))**

**radius = int((y2-y1)/2.0)**

**color = (0, 255, 0)**

**cv2.circle(im, center, radius, color, thickness=1, lineType=8, shift=0)**

**# Escribe texto en la imagen especificando la identidad de la persona min dist.**

**org = (int(x1), int(y1)) # izquierda inferior del texto**

**font\_face = cv2.FONT\_HERSHEY\_SIMPLEX**

**font\_scale = 0.8**

**text\_color = (255, 0, 0)**

**printLabel = '{} {:0.4f}'.format(label, minDistance)**

**cv2.putText(im, printLabel, org, font\_face, font\_scale, text\_color, thickness=2)**

**# Show muestra resultado**

**cv2.imshow('webcam', im)**

**# Cierra si ESC es presionado**

**k = cv2.waitKey(1) & 0xff**

**if k == 27:**

**break # ESC presionado**

**# Contador usando tramas**

**count += 1**

**cv2.destroyAllWindows()**

Corra el test desde una terminal dentro de la carpeta python como:

**python testDlibFaceRecVideo.py**

Objetivos

* Cumplir las fases superiores detectando su cara (50)
* Portar el model al raspberry pi (50)
  + Debe detectar la cara suya
  + Debe detectar la cara listada adicional que le toque

|  |  |  |
| --- | --- | --- |
| ID | Nombre | Caras de Futbolistas |
| 1 | AGUILAR ACEVEDO GABRIEL EDUARDO | Cristiano Ronaldo |
| 2 | BELFORD PETERS SHANTIEL AISHA | Eden Hazard |
| 3 | CAMPOS MONTOYA KADIR | Romelu Lukaku |
| 4 | CASTILLO ARAUZ ELIECER VALENTIN | Luis Suarez |
| 5 | CHUNG ZOU ERNESTO | Leonel Messi |
| 6 | FERMAN GONZALEZ ADRIAN ANTONIO | Gareth Bale |
| 7 | GONZALEZ BONILLA RODNEY ALEX | Koke |
| 8 | MEJIAS STABILITO JUAN LUIS | Neymar Jr |
| 9 | MENDOZA LEIVA ALDAIR ANTONIO | Edison Cavani |
| 10 | MORENO URRIOLA ROGELIO ROBERTO | Zlatan Ibrahimovic |
| 11 | PEÑA ABREGO ALCIDES | Mauro Icardi |
| 12 | QIU CHEN MICHAEL | Paulo Dybala |
| 13 | ROMERO RAMIREZ PEDRO ENRIQUE | Ivan Perisic |
| 14 | RUIZ CORONADO DYLAND CHRISTIAN | Heung Min Son |
| 15 | SALCEDO BORJA DANIELA | Mohamed Salah |
| 16 | TELLEZ PEREZ PAOLO RICARDO | Roberto Firmno |
| 17 | VERGARA PAREDES MIGUEL ANGEL | Joshua Kimmich |
| 18 | ZAMBRANO VIDAL CESAR AUGUSTO | Arturo Vidal |
| 19 | CHRISTIAN CRUZ | Antoine Griezmann |