

Informe Práctica Profesional II

Sistema de Clasificación de Acciones de Personas

Profesor: Javier Ruiz del Solar
Guía: Patricio Loncomilla
Fecha: 29 de Abril de 2016

Índice General

1. Introducción	3
2. Marco Teórico	4
2.1. Deep Learning	4
2.1.1. Redes Neuronales Convolucionales	4
3. Metodología	6
3.1. Procesamiento Base de Datos	6
3.2. Implementación del Sistema	7
4. Resultados y Análisis	11
5. Conclusiones	13
6. Anexos	15

Índice de figuras

1.	Arquitectura de la red neuronal convolucional Lenet-5	4
2.	Imágenes del conjunto de prueba del desafío VOC 2012	6
3.	Ejemplo de imágenes a las cuales se le cortaron las porciones que corresponden a personas	6
4.	Arquitectura de la red neuronal convolucional usada para clasificar acciones de personas	7
5.	Esquema de red neuronal sin dropout (izquierda) y con dropout (derecha)	8
6.	Ejecución del primer código implementado	9
7.	Ejemplo de pruebas realizadas en el sistema final	10

Índice de Tablas

1.	Distribución de las clases en el conjunto de prueba	11
2.	Average Precision del sistema medido en el conjunto de prueba del desafío PASCAL VOC 2012	11
3.	Average Precision de otros sistemas para PASCAL VOC 2012	11
4.	Matriz de confusión del sistema implementado	12

Índice de Códigos

1.	Análisis del conjunto de prueba	15
2.	Sistema funcionando con la cámara	16

1. Introducción

El trabajo realizado consistió en implementar un sistema de clasificación de 10 acciones de personas, correspondiente a: hablar por teléfono, tocar un instrumento, leer, andar en bicicleta o motocicleta, montar a caballo, correr, tomar una fotografía, usar un computador, caminar y saltar. El conjunto con el cual se entrenó y probó la red fue obtenido desde la competencia PASCAL VOC 2012

Para la clasificación se utilizó una red neuronal convolucional, la cual está compuesta por 16 capas y el tamaño de sus parámetros entrenables es de 217 MB. La función de activación que se utilizó en cada capa fue una ReLU, además el modelo consideró capas de pooling, normalización, completamente conectadas y dropout.

Para utilizar la red se implementó un código en python usando el framework Caffe, con el cual se logró acceder a las salidas de las distintas capas, en particular la última que corresponde a las 10 salidas del sistema, una por cada clase. El sistema implementado es capaz de analizar imágenes que son captadas con una cámara.

2. Marco Teórico

2.1. Deep Learning

Deep Learning es un área de *Machine Learning*, la cual usa algoritmos de aprendizaje que efectúan procesamientos no lineales en múltiples capas, logrando la extracción de características en cada uno de los niveles. Además es posible con estos algoritmos realizar aprendizaje supervisado y no supervisado.

2.1.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN por su sigla en inglés) son un tipo de arquitectura profunda, que permite el entrenamiento supervisado de un sistema. Algunas de sus características es que son más fáciles de entrenar y poseen menos parámetros que estimar, en comparación con otras arquitecturas profundas.

Una de las principales deficiencias de las redes neuronales en tareas relacionadas con reconocimiento de imágenes y patrones de voz, es que no poseen invariancia ante traslaciones, escalamientos y distorsiones de la señal de entrada. En CNN la invariancia al desplazamiento se obtiene forzando la replicación de configuraciones de pesos.

Existen algunas arquitecturas clásicas de CNN, por ejemplo, la red LENET-5, la cual es descrita en [1]. Esta red es utilizada para clasificar números escritos a mano y se pueden identificar tres tipos de capas distintas, convolucional, submuestreo y completamente conectada.

La red LENET-5 posee 6 capas (Figura 1), sin contar la capa de entrada y la de salida. La primera capa es una convolucional (C1), la cual está compuesta por 6 planos, los cuales reciben el nombre de *feature maps*, estos a su vez están compuesto por unidades, las cuales son forzadas a tener pesos idénticos solo dentro del mismo *feature map*.

Otro aspecto importante en las capas de una CNN, es el llamado campo receptivo, que son las dimensiones de la entrada a una unidad. Para el caso de C1 el campo receptivo es de 5x5 y proviene desde la imagen de entrada, en consecuencia, las unidades de los *feature maps* de C1 poseen 25 coeficientes entrenables más un bias, en total C1 cuenta con 156 parámetros entrenables.

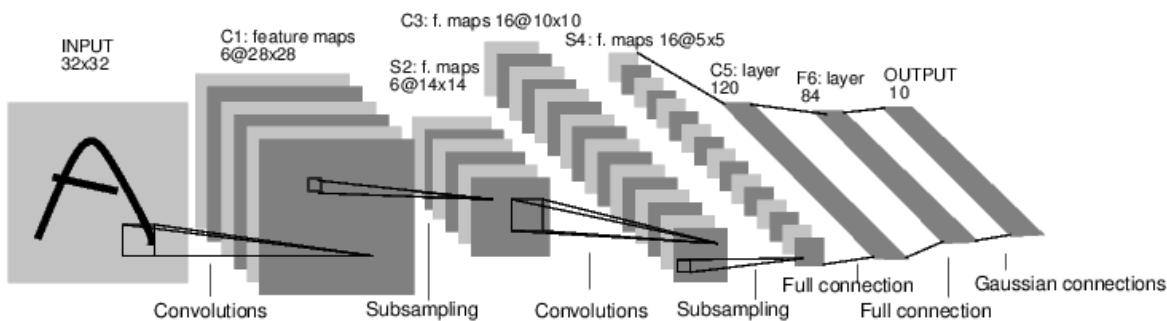


Figura 1: Arquitectura de la red neuronal convolucional Lenet-5

La capa S2 corresponde a una capa de sub muestreo, con campos receptivos de 2x2 sin solapamiento. Cada unidad calcula el promedio de sus 4 entradas, luego multiplica el resultado por un coeficiente entrenable más un bias y pasa el resultado por una función sigmoidea. En total se tienen 6 feature maps de 14x14, con 12 parámetros entrenables en total.

La capa C3 posee 16 feature maps de 10x10, con campos receptivos de 5x5. Cada feature map de la capa C3 combina un conjunto de feature maps de la capa S2, es decir, la conexión entre S2 y C3 no es completa, de esta forma se tienen 1.516 pesos entrenables para la capa C3. La siguiente capa es la S4 que posee campos receptivos de 2x2 sin solapamiento, además posee 16 feature maps de 5x5, por lo que se tienen 32 parámetros entrenables.

La capa C5 cuenta con campos receptivos desde S4 de 5x5, por lo que en este caso los 120 feature maps son de 1x1, esta capa está completamente conectada con la anterior por lo que se tienen 48.120 variables entrenables. La capa F6 posee 84 unidades y está completamente conectada a C5, por lo cual posee 10.164 pesos. La capa de salida está compuesta por unidades RBF (radial basis function), una por cada clase, las salidas de esta unidad se calculan como sigue:

$$y_i = \sum_j (x_j - w_{ij})^2 \quad (1)$$

En la ecuación (1) x_j es el vector de entradas a las unidades y w_{ij} es el vector de pesos o de parámetros, el cual para esta red fue creado manualmente.

3. Metodología

3.1. Procesamiento Base de Datos

La base de datos con la cual se entrenó la red y con la que se probó se obtuvo desde el proyecto PASCAL VOC, específicamente desde el desafío del 2012 en la categoría de clasificar acciones de personas, en la Figura 2 se aprecia un ejemplo de cada clase, las cuales son: hablar por teléfono, tocar un instrumento, leer, andar en bicicleta o motocicleta, montar a caballo, correr, tomar una fotografía, usar un computador, caminar y saltar.



Figura 2: Imágenes del conjunto de prueba del desafío VOC 2012

Para realizar la prueba de la red primero se realizó un procesamiento a la base de datos de prueba, ya que, se necesitaba solo la imagen de una persona y no los objetos que la rodeaban, por lo que, se utilizaron las anotaciones (provistas por el desafío VOC 20012) de las cajas que rodeaban a una persona, para analizar solamente esta porción de la imagen. En la Figura 3 (a) y (b) se comparan la imagen original y la resultante luego de cortar la porción correspondiente a una persona.



Figura 3: Ejemplo de imágenes a las cuales se le cortaron las porciones que corresponden a personas

3.2. Implementación del Sistema

Se exploraron dos sistemas, el primero adapta RCNN para clasificar las acciones usando más de una región de la imagen[2], no tan solo la persona, sino que también lo que la rodea. Este trabajo no fue utilizado, por no disponer de la capacidad computacional necesaria para los requerimientos del modelo.

El sistema implementado se basó en el trabajo de reconocimiento de acciones de personas de G.Gkioxari et al [3], se utilizó una arquitectura de red neuronal convolucional entrenada por ellos. Específicamente el modelo está compuesto por 23 capas (Figura 4) considerando las funciones de activación.

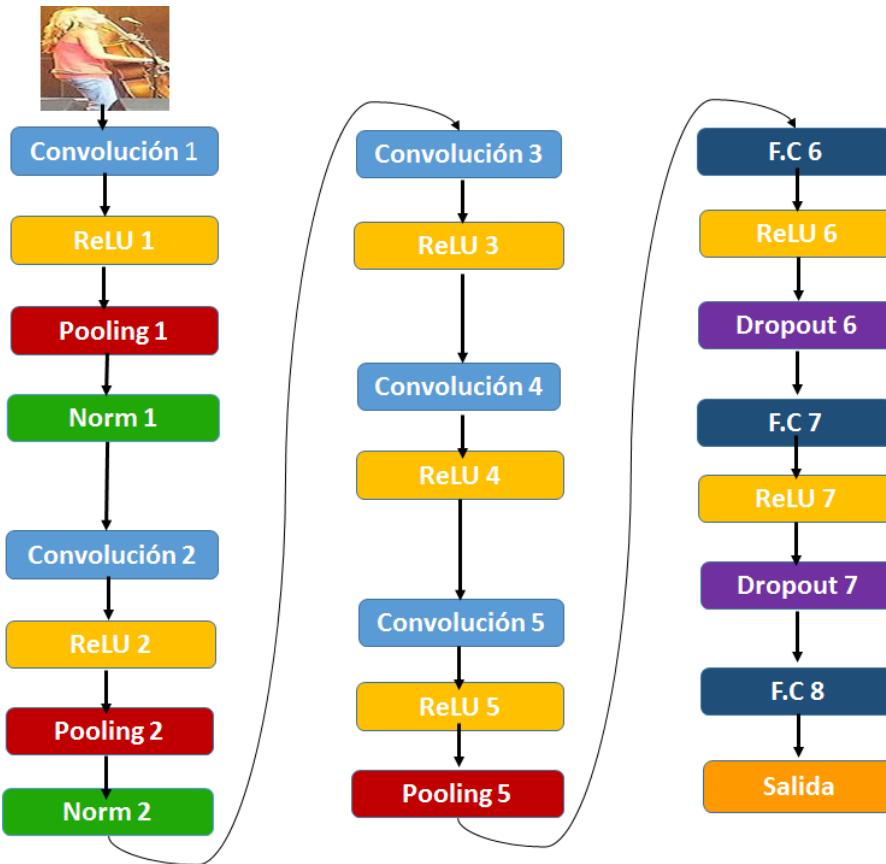


Figura 4: Arquitectura de la red neuronal convolucional usada para clasificar acciones de personas

El modelo que compartió en un principio G.Gkioxari fue uno en el cual no se consideraba la capa 8 ni la de softmax, ya que, en su trabajo los mejores resultados se obtuvieron conectando un clasificador SVM en la capa 7. Se le solicitaron los pesos del SVM entrenado pero no los tenía, por lo que después de un tiempo y tras algunas implementaciones propias de la capa 8 y de la capa de softmax, las cuales no fueron correctas, G.Gkioxari facilitó la arquitectura completa de la red.

La red cuenta con 5 capas convolucionales seguidas por una función de activación ReLU (rectified linear unit) cada una, esta función se calcula como $f(x) = \max(0, x)$. También hay tres capas de pooling, que

se configuran para extraer el máximo desde regiones de 3×3 con paso de 2 pixeles. Además la arquitectura posee dos capas de normalización, las cuales son antecedidas por capas de pooling.

Las capas finales de la red son del tipo completamente conectadas, seguidas por una función de activación ReLU y una capa de Dropout, la cual en cada etapa del entrenamiento retira nodos individuales con probabilidad $1-p$, los cuales no participan del entrenamiento, esto ayuda a evitar el sobre ajuste de la red [4]. En la Figura 5 se muestra un esquema de una red neuronal estándar y la misma red luego de aplicar dropout.

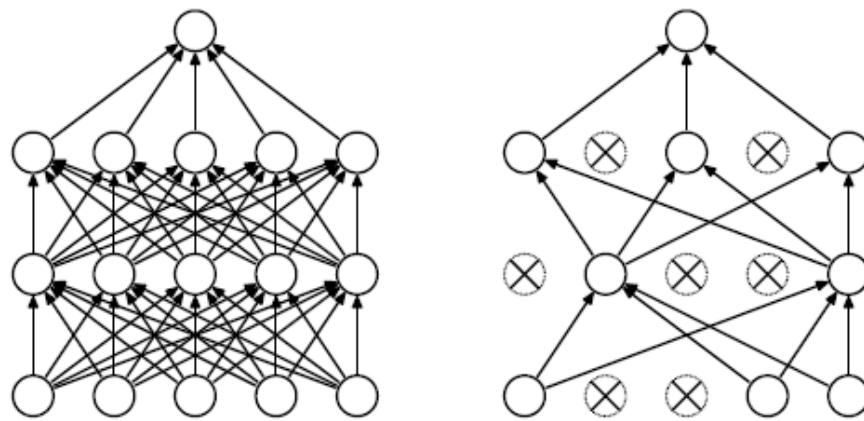
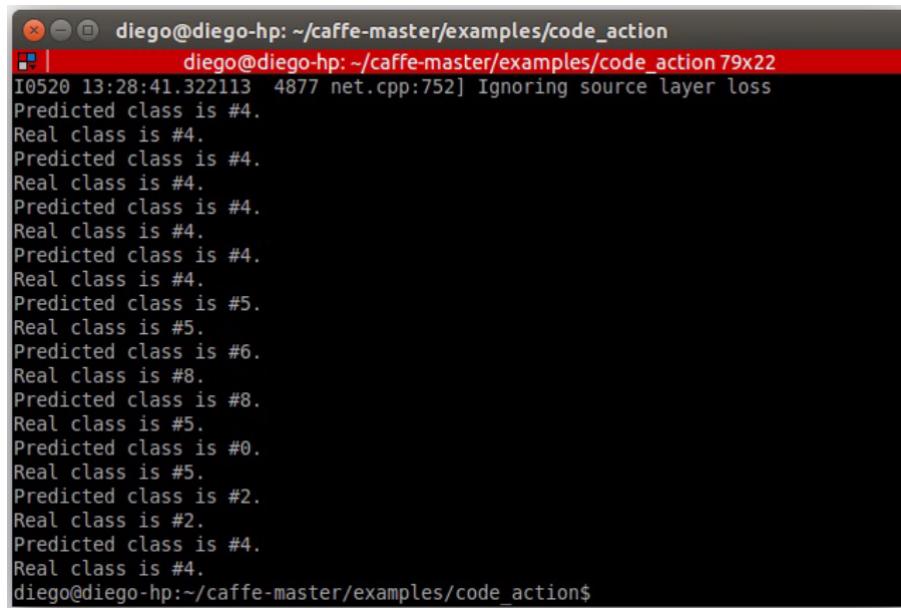


Figura 5: Esquema de red neuronal sin dropout (izquierda) y con dropout (derecha)

Se elaboraron principalmente dos scripts en python que permiten probar la red, utilizando el framework Caffe. El primero de estos lo que realiza es leer un archivo .txt en el cual están grabados los nombres de las imágenes de acciones de personas pertenecientes al conjunto de prueba, además este archivo posee la clase real asociada a cada imagen.

Al ejecutar el código primer código se muestra la clase predicha por la red y la clase a la cual efectivamente corresponden las imágenes analizadas. La clasificación realizada por la red y la etiqueta real son guardadas para el posterior análisis de los resultados. En la Figura 6 se expone lo que aparece en el terminal cuando se ejecuta el script.



```
diego@diego-hp: ~/caffe-master/examples/code_action
I0520 13:28:41.322113 4877 net.cpp:752] Ignoring source layer loss
Predicted class is #4.
Real class is #4.
Predicted class is #5.
Real class is #5.
Predicted class is #6.
Real class is #8.
Predicted class is #8.
Real class is #5.
Predicted class is #0.
Real class is #5.
Predicted class is #2.
Real class is #2.
Predicted class is #4.
Real class is #4.
diego@diego-hp:~/caffe-master/examples/code_action$
```

Figura 6: Ejecución del primer código implementado

El segundo código realizado permite que un usuario muestre a la cámara del computador una imagen de alguna acción dentro de las 10 clases antes mencionadas o bien que él efectúe alguna acción frente a la cámara, luego presionando la tecla 'q' el sistema comienza con el análisis de la acción que le fue mostrada entregando la clase predicha por este.

En la Figura 7 se muestran dos ejemplos con los cuales se probó el sistema, uno en el cual el usuario se encuentra hablando por teléfono y el sistema reconoce esta acción, por su parte la Figura 7 (b) corresponde a una imagen que se le muestra al sistema, el cual reconoce de forma correcta la clase en cuestión.



Figura 7: Ejemplo de pruebas realizadas en el sistema final

4. Resultados y Análisis

Una vez implementado el sistema de detección de actividades de personas se procedió a comprobar su funcionamiento, para esto se utilizó el conjunto de imágenes de prueba del desafío PASCAL VOC 2012. Este conjunto cuenta con 5360 imágenes correspondientes a las 9 clases, en la Tabla 1 se detalla la cantidad de imágenes de cada clase presente en el conjunto de prueba.

Acción	Nº Imágenes	%
Hablar por teléfono	455	8.48
Tocar un Instrumento	636	11.86
Leer	552	10.29
Andar en Bicicleta o Motocicleta	594	11.08
Montar a caballo	540	10.07
Correr	583	10.87
Tomar una fotografía	453	8.45
Usar computador	449	8.37
Caminar	601	11.21
Saltar	497	9.27

Tabla 1: Distribución de las clases en el conjunto de prueba

Utilizando el VOCdevkit 2012 se obtuvo el *Average Precision (AP)* para cada clase, los resultados obtenidos se detallan en la Tabla 2, mencionar que el AP es la principal cantidad medida para determinar la efectividad de los sistemas que participaron en el desafío de PASCAL VOC 2012.

	Hablar por teléfono	Tocar Instrumento	Leer	Andar en Bicicleta	Montar a caballo	Correr	Tomar Fotografía	Usar Computador	Caminar	Saltar	mAP
AP	43.0	70.5	37.7	86.1	82.5	79.9	44.1	52.0	56.6	69.9	62.23

Tabla 2: Average Precision del sistema medido en el conjunto de prueba del desafío PASCAL VOC 2012

En promedio se obtuvo un AP de 62.23 lo cual es un resultado que se acerca al del trabajo de referencia [3] y resulta menor, como se esperaba, ya que, en el sistema de clasificación citado se utilizan las capas convolucionales para la extracción de características y luego se clasifican las 10 clases con SVM en cascada, logrando un mAP de 70.5. Lo anterior ocurre porque al usar un clasificador SVM se obtiene un menor sobre-ajuste que al hacer la clasificación con capas completamente conectadas.

En la Tabla 3 se muestran los líderes de la competencia VOC 2012, más los resultados obtenidos por el paper usado de base. Los dos trabajos con mejores resultados (mAP de 70.2 y 70.5) fueron obtenidos con redes convolucionales.

	Hablar por teléfono	Tocar Instrumento	Leer	Andar en Bicicleta	Montar a caballo	Correr	Tomar Fotografia	Usar Computador	Caminar	Saltar	mAP
Standford	75.7	44.8	66.6	44.4	93.2	94.2	87.9	38.4	70.6	75.6	69.1
Oxford	77.0	50.4	65.3	39.5	94.1	95.9	87.7	42.7	68.8	74.5	69.6
Oquad et al [5]	74.8	46.0	75.6	45.3	93.5	95.0	86.5	49.3	66.7	69.5	70.2
Action R-CNN	76.2	47.4	77.5	42.2	94.9	94.3	87.0	52.9	66.5	66.5	70.5

Tabla 3: Average Precision de otros sistemas para PASCAL VOC 2012

Además con el total del conjunto de prueba se obtuvo la matriz de confusión del sistema, desde la cual se tiene un porcentaje de 63.68 % de correcta clasificación. Se observa además que el mayor error en la clasificación sucedió cuando la clase real correspondía a leer, equivocándose el sistema con la acción de usar computador y en menor medida con la de tocar un instrumento. Esto se puede deber a que en las imágenes de prueba estas acciones se realizan mayoritariamente con las personas sentadas, por lo cual, las características extraídas no lograron generar un mayor grado de complementariedad entre ellas.

		Predicho									
		1	2	3	4	5	6	7	8	9	10
Real	1	237 (4.42 %)	41 (0.76 %)	44 (0.82 %)	5 (0.09 %)	13 (0.24 %)	5 (0.09 %)	48 (0.89 %)	27 (0.50 %)	25 (0.46 %)	10 (0.18 %)
	2	27 (0.50 %)	460 (8.58 %)	44 (0.82 %)	11 (0.20 %)	18 (0.33 %)	3 (0.05 %)	10 (0.18 %)	30 (0.55 %)	19 (0.35 %)	14 (0.26 %)
	3	54 (0.98 %)	70 (1.30 %)	213 (3.97 %)	6 (0.11 %)	31 (0.57 %)	8 (0.14 %)	18 (0.33 %)	122 (2.27 %)	16 (0.29 %)	15 (0.27 %)
	4	1 (0.01 %)	10 (0.18 %)	8 (0.14 %)	505 (9.42 %)	34 (0.63 %)	5 (0.09 %)	3 (0.05 %)	3 (0.05 %)	17 (0.31 %)	8 (0.14 %)
	5	5 (0.09 %)	10 (0.18 %)	9 (0.16 %)	45 (0.83 %)	441 (8.22 %)	1 (0.01 %)	10 (0.18 %)	7 (0.13 %)	7 (0.13 %)	5 (0.09 %)
	6	6 (0.11 %)	3 (0.05 %)	20 (0.37 %)	12 (0.22 %)	17 (0.31 %)	419 (7.81 %)	15 (0.27 %)	3 (0.05 %)	70 (1.30 %)	18 (0.33 %)
	7	45 (0.83 %)	55 (1.02 %)	22 (0.41 %)	20 (0.37 %)	35 (0.65 %)	6 (0.11 %)	192 (3.58 %)	20 (0.37 %)	42 (0.78 %)	16 (0.29 %)
	8	35 (0.65 %)	43 (0.80 %)	73 (1.36 %)	2 (0.03 %)	8 (0.14 %)	5 (0.09 %)	15 (0.27 %)	257 (4.79 %)	5 (0.09 %)	6 (0.11 %)
	9	19 (0.35 %)	21 (0.39 %)	18 (0.33 %)	41 (0.76 %)	32 (0.59 %)	52 (0.97 %)	27 (0.50 %)	14 (0.26 %)	349 (6.51 %)	28 (0.52 %)
	10	9 (0.16 %)	17 (0.31 %)	15 (0.27 %)	25 (0.46 %)	15 (0.27 %)	23 (0.42 %)	15 (0.27 %)	9 (0.16 %)	27 (0.50 %)	342 (6.38 %)

Tabla 4: Matriz de confusión del sistema implementado

5. Conclusiones

Se comprobó que una arquitectura de red neuronal convolucional es capaz de mejorar los resultados previos en lo que respecta a la clasificación de las 10 acciones de personas propuestas por el desafío VOC 2012. Además se obtuvo que utilizar las primeras capas para seleccionar características y luego usar un clasificador SVM en cascada mejora el rendimiento, ya que, el sobreajuste es menor.

Mencionar que para utilizar modelos de redes neuronales se requiere una amplia capacidad de cómputo, ya que, estos resultan pesados por la gran cantidad de parámetros entrenables. El proceso que más capacidad requiere es el de entrenamiento, ya que, probar la red con un conjunto de imágenes resulta rápido, en este trabajo el sistema tomaba en promedio 2 s en generar una salida para una imagen de entrada.

Para lograr mejorar los resultados se tiene que poner énfasis en técnicas para disminuir el sobreajuste de la red, sin que aumente excesivamente el tiempo de entrenamiento, además puede resultar conveniente analizar los objetos que rodean a una persona para determinar la acción que está efectuando.

El trabajo futuro que se debe hacer para continuar con este sistema es implementar un detector de personas más el de acciones, puesto que por ahora se clasifican las imágenes recortadas a las porciones que corresponden a una persona.

Bibliografía

- [1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition.
- [2] Georgia Gkioxari, Ross Girshick, and Jitendra Malik. Contextual Action Recognition with R*CNN
- [3] Georgia Gkioxari, Bharath Hariharan, Ross Girshick, and Jitendra Malik. R-CNNs for Pose Estimation and Action Detection
- [4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting
- [5] M.Oquad, L.Bottou, I.Laptev, and J.Sivic. Learning and transferring mid-level image representations using convolutional neuronal networks. In *CVPR*, 2014.

6. Anexos

Código 1: Análisis del conjunto de prueba

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4
5  import caffe
6  import numpy as np
7  from StringIO import StringIO # para guardar arreglos en txt
8  import cPickle as pickle # para guardar
9  import matplotlib.pyplot as plt
10 import xml.etree.ElementTree # para abrir xml
11
12 # Make sure that caffe is on the python path:
13 caffe_root='/home/diego/caffe-master'
14 import sys
15 sys.path.insert(0, caffe_root + 'python')
16 print(caffe_root + 'python')
17
18 plt.rcParams['figure.figsize'] = (10, 10)
19 plt.rcParams['image.interpolation'] = 'nearest'
20 plt.rcParams['image.cmap'] = 'gray'
21
22 import os
23 caffe.set_mode_cpu()
24
25 # Prueba en la red de detección de acciones Net 2
26 # .prototxt es la arquitectura de la red
27 # pascal_finetune_HumanNet2_iter_10000 son los pesos de la red
28 net = caffe.Net('pascal_finetune_fc8.prototxt',
29                  'pascal_finetune_HumanNet2_iter_10000',
30                  caffe.TEST)
31
32 # input preprocessing: 'data' is the name of the input blob == net.inputs[0]
33 transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})
34 transformer.set_transpose('data', (2,0,1))
35 transformer.set_raw_scale('data', 255) # the reference model operates on images in
36 [0,255] range instead of [0,1]
37 transformer.set_channel_swap('data', (2,1,0)) # the reference model has channels in
38 # BGR order instead of RGB
39
40 # set net to batch size of 50
41 # .reshape(bach,nº de canales (=3 si es RGB),height , width)
42 net.blobs['data'].reshape(1,3,227,227)
43
44 # leer archivo txt
45 from PIL import Image # para usar libreria de imágenes
46 import numpy as num
47 arch=open('test2.txt','r')# leer txt conjunto de test
48 c=0
49 linea=arch.readline()# leer una linea del archivo test2.txt
50 lista=linea.split()# pasar la linea a una lista

```

```

49
50 while linea!="":
51     net.blobs['data'].data[...] = transformer.preprocess('data', caffe.io.load_image('/
52         home/diego/caffe-master/examples/VOC2012-test2/'+lista[0]+ '_'+lista[1]+ '.jpg'))
53         # cargar imagen para ser analizada
54     out = net.forward()# salida desde la ultima capa de la red
55     c=c+1
56
57     print("Predicted class is #{}.".format(out['prob'].argmax()))# prob es el nombre de
58         # la ultima capa del modelo de red cnn
59         # argmax es para
60         # obtener la salida
61         # con maxima
62         # probabilidad
63
64     print("Real class is #{}.".format(lista[2]))
65     # leer la linea siguiente
66     linea=arch.readline();
67     lista=linea.split();
68
69     # Para el programa para no analizar todas las imagenes
70     if c>=10:
71         break

```

Código 2: Sistema funcionando con la cámara

```

1  #!/usr/bin/env python
2  # coding: utf-8
3  import numpy as np
4  import cv2
5  from cv2 import *
6
7  # Capturar imagen
8  cap=cv2.VideoCapture(0)
9
10 while (True):
11     ret ,frame=cap.read()
12     cv2.imshow('video1',frame)
13     if cv2.waitKey(1) & 0xFF==ord('q'):#terminar al precionar q
14         imwrite("img.jpg",frame) #save image
15         break
16 cap.release()
17 cv2.destroyAllWindows()
18
19
20
21
22 # Analizar imagen tomada
23
24 import caffe
25 import numpy as np
26 from StringIO import StringIO # para guardar arreglos en txt
27 import cPickle as pickle # para guardar
28 import matplotlib.pyplot as plt
29 import xml.etree.ElementTree # para abrir xml

```

```

30
31 # Make sure that caffe is on the python path:
32 caffe_root='/home/diego/caffe-master'
33 import sys
34 sys.path.insert(0, caffe_root + 'python')
35 print(caffe_root + 'python')
36
37 plt.rcParams[ 'figure.figsize' ] = (10, 10)
38 plt.rcParams[ 'image.interpolation' ] = 'nearest'
39 plt.rcParams[ 'image.cmap' ] = 'gray'
40
41 import os
42 caffe.set_mode_cpu()
43
44 # Prueba en la red de detección de acciones Net 2
45 # .prototxt es la arquitectura de la red
46 # pascal_finetune_HumanNet2_iter_10000 son los pesos de la red
47 net = caffe.Net('pascal_finetune_fc8.prototxt',
48                  'pascal_finetune_HumanNet2_iter_10000',
49                  caffe.TEST)
50
51 # input preprocessing: 'data' is the name of the input blob == net.inputs[0]
52 transformer = caffe.io.Transformer({ 'data': net.blobs[ 'data' ].data.shape})
53 transformer.set_transpose('data', (2,0,1))
54 transformer.set_raw_scale('data', 255) # the reference model operates on images in
      [0,255] range instead of [0,1]
55 transformer.set_channel_swap('data', (2,1,0)) # the reference model has channels in
      BGR order instead of RGB
56
57 # set net to batch size of 50
58 # .reshape(bach,n de canales (=3 si es RGB),height , width)
59 net.blobs[ 'data' ].reshape(1,3,227,227)
60
61 # leer archivo txt
62 from PIL import Image # para usar libreria de imagenes
63 import numpy as num
64
65 net.blobs[ 'data' ].data[...] = transformer.preprocess('data', caffe.io.load_image('img.
      jpg'))# cargar imagen para ser analizada
66 out = net.forward()# salida desde la ultima capa de la red
67 print("Predicted class is #{}.".format(out[ 'prob' ].argmax()))# prob es el nombre de la
      ultima capa del modelo de red cnn
68
      # argmax es para
      # obtener la salida
      # con maxima
      # probabilidad
69 nc=out[ 'prob' ].argmax()
70 clases=['Hablando por telefono','Tocando un instrumento','Leyendo','Andando en
      Bicicleta','Montando a Caballo','Corriendo','Tomando una fotografía','Usando el
      computador','Caminando','Saltando']
71 print(clases[nc])

```