
Universidad
Tecmilenio®

Universidad Tecmilenio

Proyectos de Aprendizaje Profundo

Clasificador personalizado que se adapte mejor a sus imágenes específicas

Detección de objetos dentro de una imagen

Introducción

Las computadoras solo "ven" matrices de números en lugar de imágenes, y lo hacen a través de álgebra lineal. La clasificación de imágenes es una tarea en la que se utilizan algoritmos de visión computacional y aprendizaje profundo para extraer el significado de una imagen digitalizada.

En el aprendizaje profundo (AP), una red neuronal convolucional (CNN, por sus siglas en inglés) es una clase de red neuronal que se ha utilizado con gran éxito en tareas de visión computacional, especialmente en el reconocimiento y detección de objetos.

La detección de objetos es una tarea que se subdivide en dos: clasificación y localización. Existen distintas arquitecturas con las que puede resolverse esta tarea, como R-CNN, Fast R-CNN, entre otras. Estas técnicas requieren de un poder de procesamiento importante durante el entrenamiento de la red, por lo que se recomienda utilizar GPU para agilizarlo.

Explicación

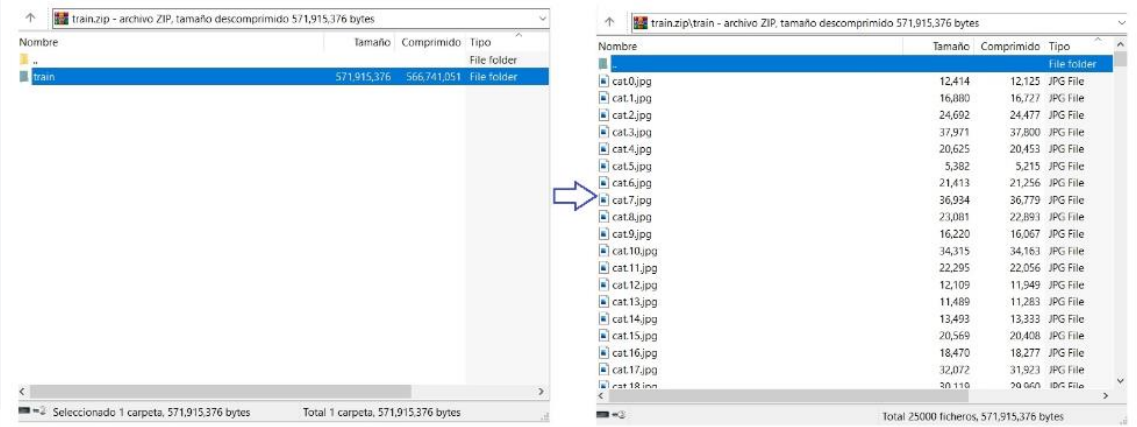
Pasos para construir un clasificador adaptable

En este tema se desarrollará un clasificador de imágenes, cuyo objetivo será determinar a qué clase o categoría pertenece la imagen de entrada.

La forma en la que se resolverá este problema de clasificación será entrenando a una red neuronal artificial con cierta cantidad de imágenes de gatos y perros, para que la red sea capaz de predecir la clase a la que pertenece una imagen nueva de esta naturaleza (con perros o gatos). Se utilizará la librería Keras (Gulli y Pal, 2017) en Python para construir la red neuronal convolucional.

Proyecto: construcción de clasificador personalizado

La primera etapa de este proyecto consiste en la preparación de los conjuntos de datos de entrenamiento y prueba. Después de descargar el archivo: train.zip de Dogs vs. Cats de Kaggle, se debe descomprimir en una carpeta y distribuir su contenido de tal forma que Keras pueda trabajar con este (Kaggle, s.f.; Keras, s.f.). Al descomprimir el archivo train.zip se creará una carpeta de nombre train que incluye 25,000 imágenes de perros y gatos. En la figura siguiente, se muestra un ejemplo del contenido del archivo zip.



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Se crearán las carpetas train y test, y dentro de cada una de ellas, perros (dogs) y gatos (cats), que tendrán los archivos de cada categoría de la carpeta original descargada. En la siguiente figura , se muestran las instrucciones para realizar este proceso de forma automática en Python. Antes, nota que debe crearse primero una carpeta llamada dataset. Esta carpeta debe estar vacía y ubicada en el mismo lugar donde se encuentra la carpeta train original y el archivo que contiene al script.

preproimg.py > ...

```python
# Etapa 1 - Preparación de los conjuntos de datos
# Organización del conjunto de datos descargado para trabajar con flow_from_directory() de Keras
from os import makedirs
from os import listdir
from shutil import copyfile
from random import seed
from random import random

dataset_home = 'dataset/'

# Se crean dos carpetas para los datos a utilizar
subdirs = ['train/', 'test/']
for subdir in subdirs:
    # Se crean subdirectorios para cada categoria
    labeldirs = ['dogs/', 'cats/']
    for labldir in labeldirs:
        newdir = dataset_home + subdir + labldir
        makedirs(newdir, exist_ok=True)
# Semilla de generador de numeros aleatorios
seed(1)
# Se selecciona de forma aleatoria el 25% de los datos
# como conjunto de pruebas
val_ratio = 0.25
# Se transfieren los datos a la estructura de carpetas
src_directory = 'train/'
for file in listdir(src_directory):
    src = src_directory + '/' + file
    dst_dir = 'train/'
    if random() < val_ratio:
        dst_dir = 'test/'
    if file.startswith('cat'):
        dst = dataset_home + dst_dir + 'cats/'  + file
        copyfile(src, dst)
    elif file.startswith('dog'):
        dst = dataset_home + dst_dir + 'dogs/'  + file
        copyfile(src, dst)

```

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

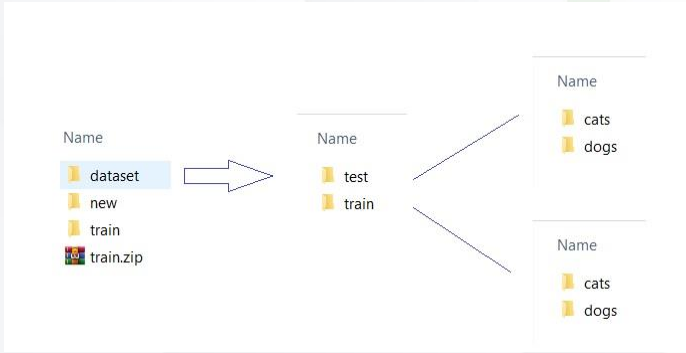
Explicación

Es necesario asegurarse de que se tienen instaladas las librerías requeridas en Python para este proyecto: Theano, Tensorflow y Keras (Theano, s.f.; Tensorflow, s.f.; Keras, s.f.).

La etapa de construcción de la CNN incluye cuatro pasos. En la siguiente figura se muestran las instrucciones para realizarla.

En Keras existen dos formas de inicializar la red neuronal, ya sea como una secuencia de capas o como un grafo. En este proyecto, se define al modelo como una red secuencial. Dado que se trabajará con imágenes, se definen capas convolucionales en dos dimensiones (Conv2D); de igual manera para la capa de reducción (MaxPooling2D).

El primer paso dentro de esta etapa es agregar la capa convolucional, después se agrega la capa de reducción, continuando con el vector que tendrá las imágenes después de la reducción y, por último, se crea la capa completamente conectada. Para un mayor entendimiento de los parámetros en cada instrucción, se recomienda consultar la documentación de Keras. Al finalizar la construcción de la CNN, se compila indicando qué se utilizará, en este ejemplo, el algoritmo de gradiente descendente, la función de error de entropía cruzada y como métrico de desempeño, la precisión.



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

cnn.py > ...

```python
import sys
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from matplotlib import pyplot

# Etapa 2 - Construcción de la red neuronal convolucional (CNN)

#Inicializar la red neuronal
clasificador = Sequential()

# Paso 1. Convolución
clasificador.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

# Paso 2. Pooling (Reducción)
clasificador.add(MaxPooling2D(pool_size = (2, 2)))

# Segunda capa convolucional
clasificador.add(Conv2D(32, (3, 3), activation = 'relu'))
clasificador.add(MaxPooling2D(pool_size = (2, 2)))

# Paso 3 - Arreglo "plano" para los elementos de la matriz de imagenes
clasificador.add(Flatten())

# Paso 4 - Red completamente conectada
clasificador.add(Dense(units = 128, activation = 'relu'))
clasificador.add(Dense(units = 1, activation = 'sigmoid'))

# Compilación de la CNN
clasificador.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Explicación

A continuación, se realiza un proceso de incremento de datos utilizando la función **ImageDataGenerator()**. Básicamente se crean más imágenes a partir del conjunto de entrenamiento ejecutando operaciones como rotación, reflejo, difuminado, etc. En la siguiente figura se muestran las instrucciones a realizar en esta etapa.

```
cnn.py > ...
43
44  # Etapa 3 - Entrenamiento de la red
45
46  from keras.preprocessing.image import ImageDataGenerator
47
48  gendatos_entrenamiento = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)
49
50  gendatos_prueba = ImageDataGenerator(rescale = 1./255)
51
52  datos_entrenamiento = gendatos_entrenamiento.flow_from_directory('dataset/train', target_size = (64, 64),
53                                                   batch_size = 64,
54                                                   class_mode = 'binary')
55
56  datos_prueba = gendatos_prueba.flow_from_directory('dataset/test', target_size = (64, 64),
57                                               batch_size = 64,
58                                               class_mode = 'binary')
59
60  historico = clasificador.fit_generator(datos_entrenamiento, steps_per_epoch = len(datos_entrenamiento), epochs = 15,
61                         validation_data = datos_prueba, validation_steps = len(datos_prueba))
```

En la figura a continuación, se muestran las instrucciones para la evaluación del modelo.

```
cnn.py > ...
63  # Etapa 4 - Evaluación del modelo
64
65  _, acc = clasificador.evaluate_generator(datos_prueba, steps=len(datos_prueba), verbose=0)
66  print('> %.3f' % (acc * 100.0))
67
68  # Grafica de la función de perdida (error)
69  pyplot.subplot(211)
70  pyplot.title('Cross Entropy Loss')
71  pyplot.plot(historico.history['loss'], color='blue', label='train')
72  pyplot.plot(historico.history['val_loss'], color='orange', label='test')
73
74  # Grafica de la Precisión
75  pyplot.subplot(212)
76  pyplot.title('Classification Accuracy')
77  pyplot.plot(historico.history['accuracy'], color='blue', label='train')
78  pyplot.plot(historico.history['val_accuracy'], color='orange', label='test')
79
80  filename = sys.argv[0].split('/')[-1]
81  pyplot.savefig(filename + '_plot.png')
82  pyplot.close()
```

Estas pantallas se obtuvieron directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

En la figura siguiente, se muestran las instrucciones requeridas para la última etapa del proyecto, es decir, la predicción con datos nuevos.

Se seleccionaron, sin utilizar criterio alguno, las imágenes 10000.jpg y 10005.jpg del archivo test1.zip que corresponden a un perro y un gato, respectivamente. Y al ejecutar las líneas de código, el modelo entrenado acertó en la categoría del animal que aparece en la imagen.

cnn.py > ...

```
# Etapa 5 - Predicción con datos nuevos

import numpy as np
from keras.preprocessing import image

img_prueba = image.load_img('new/10000.jpg', target_size = (64, 64))
img_prueba = image.img_to_array(img_prueba)
img_prueba = np.expand_dims(img_prueba, axis = 0)
res = clasificador.predict(img_prueba)
datos_entrenamiento.class_indices

if res[0][0] == 1:
    prediccion = 'perro'
else:
    prediccion = 'gato'

img_prueba = image.load_img('new/10005.jpg', target_size = (64, 64))
img_prueba = image.img_to_array(img_prueba)
img_prueba = np.expand_dims(img_prueba, axis = 0)
res = clasificador.predict(img_prueba)
datos_entrenamiento.class_indices

if res[0][0] == 1:
    prediccion = 'perro'
else:
    prediccion = 'gato'
```

Estas pantallas se obtuvieron directamente del software que se está explicando en la computadora, para fines educativos.

Explicación

Cómo detectar objetos en imágenes

La detección de objetos es un proceso en el que a partir de una imagen como entrada de un modelo se detecta cualquier tipo de objeto en ella, ubicándolo con un recuadro, una categoría y un valor de probabilidad.

La solución más viable es utilizando el **aprendizaje por transferencia (AT)**, donde existen varios resultados importantes en tareas de procesamiento de imágenes (Zhuang, Qi, Duan, Xi, Zhu, Zhu, Xiong y He, 2020). El AT es una alternativa para evitar estos largos tiempos de entrenamiento cuando se construye una red neuronal. El AT es una técnica que permite reutilizar modelos que se obtuvieron a partir del entrenamiento con un conjunto de datos de cierto problema para resolver otro problema similar.

En este proyecto se utilizará Caffe que es un framework de desarrollo para aprendizaje profundo, desarrollado por Berkeley AI Research (BAIR) y OpenCV.

Explicación

Proyecto de detección de objetos dentro de una imagen

Antes de comenzar, hay que asegurarse de que se tienen instaladas las librerías requeridas en Python para este proyecto: **OpenCV, Numpy y Matplotlib**. La detección de objetos se realizará utilizando OpenCV y un modelo preentrenado en Caffe. Se utilizará la librería DNN de OpenCV para realizar el proceso de detección de objetos. **Caffe** es utilizado para resolver una gran variedad de tareas que van desde una simple regresión hasta aplicación.

Para comenzar, se deben descargar los archivos MobileNetSSD_deploy.caffemodel y MobileNetSSD_deploy.prototxt.txt. El primero contiene al modelo generado por Caffe e incluye la configuración de una red neuronal ya entrenada con sus pesos. El segundo archivo contiene, en un formato legible, información para entrenar modelos y definir los parámetros de la CNN.

Además, Se recomienda crear dos carpetas, una llamada modelo y otra llamada imágenes. La primera es para almacenar los modelos descargados previamente. En la segunda carpeta se puede almacenar cualquier cantidad de imágenes sobre las que se desee realizar la detección de objetos.

dnn.py

```python
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import time

# Modelos de Caffe descargados
caffe_model = 'modelos/MobileNetSSD_deploy.caffemodel'
caffe_proto = 'modelos/MobileNetSSD_deploy.prototxt.txt'

# Las etiquetas de los objetos del modelo caffe estan predefinidos para MobileNetSSD y protbuf
# Los modelos han sido entrenados para identificar los objetos siguientes.
labels = ("background", "aeroplane", "bicycle",
          "bird", "boat", "bottle", "bus",
          "car", "cat", "chair", "cow",
          "diningtable", "dog", "horse",
          "motorbike", "person", "pottedplant",
          "sheep", "sofa", "train", "tvmonitor")

# Arreglo aleatorio de colores para identificar diferentes objetos
label_colors = np.random.uniform(0, 255, (len(labels),3))

# Imagen a utilizar en el proceso de identificación de objetos
source_image = 'imagenes/bici.jpg'
img = cv.imread(source_image)
```

Explicación

Adicionalmente se crean variables con las etiquetas de los objetos para los cuales ya se entrenó el modelo en Caffe y una selección aleatoria de colores para identificar a cada objeto en su categoría o clase. Por último, se define una imagen en la que se identificarán los objetos. La imagen bici.jpg se muestra a continuación:

El siguiente paso es **extraer el BLOB** (Binary Large Object), es decir, la extracción de regiones conectadas en una imagen binaria digital, básicamente contornos con características como orientación, centroide, color, área, media y desviación estándar de los pixeles en la región cubierta. Además, **se preprocesa la imagen** para facilitar su tratamiento en la red neuronal normalizando los cambios de iluminación (**mean substraction**) y aplicando ajustes en su escala (**scaling**).

La imagen ya normalizada se pasa como entrada a la red neuronal y se ejecuta la **propagación** hacia adelante para la detección de objetos, utilizando el modelo previamente entrenado. En la figura siguiente, se muestran las instrucciones para realizar lo descrito.

dnn.py > ...

```
46
47  # Se crea la red con los modelos pre entrenados para utilizarse en el proceso de identificación
48  net = cv.dnn.readNetFromCaffe(caffe_proto, caffe_model)
49
50  # Se obtienen las dimensiones de la imagen
51  orig_rows = img.shape[0]
52  orig_cols = img.shape[1]
53
54  rows = 300
55  cols = 300
56
57  # Redimensionamiento de la imagen a 300x300 pixeles por requerimientos del modelo
58  resized_img = cv.resize(img,(rows,cols))
59
60  # Extracción del blob y preprocesamiento (normalización)
61  blob = cv.dnn.blobFromImage(resized_img, 0.00784, (rows, cols), (127.5, 127.5, 127.5), swapRB=True, crop=False)
62
63  # La red neuronal recibe el blob de entrada
64  net.setInput(blob)
65
66  # Propagación hacia adelante utilizada para obtener todos los objetos detectados
67  start_time = time.time()
68  out = net.forward()
69  end_time = time.time()
70
71  print('Tiempo transcurrido en la inferencia: {} milisegundos'.format(round((end_time - start_time)*1000.0 , 3)))
```

Explicación

Se fija un umbral de nivel de confianza en un rango de 0 a 1, esto es, un valor que cuantifica qué tan bien se ajustan los objetos identificados a los que el modelo predice, donde 1 indica que se tiene un 100% de confianza de que el objeto identificado corresponde a la categoría mostrada. En la figura a continuación, se muestran las instrucciones para visualizar los resultados de esta tarea.

dnn.py

```
72
73  # Umbral de confianza para los objetos detectados
74  confidence = 0.7
75
76  print('Objetos detectados: ')
77  ## Datos de cada objeto detectado en el blob
78  for detection in out[0,0,:,:]:
79      #nivel de confianza
80      score = float(detection[2])
81
82      label_index = int(detection[1])
83
84      # Rectangulo y nombre del objeto detectado con el nivel de confianza obtenido
85      if score > confidence:
86          left = detection[3] * cols
87          top = detection[4] * rows
88          right = detection[5] * cols
89          bottom = detection[6] * rows
90
91          label_text = labels[label_index] +  " " + str(round(score, 4))
92          print(label_text)
93          cv.putText(resized_img, label_text, (int(left), int(top)), cv.FONT_HERSHEY_SIMPLEX, 0.5, label_colors[label_index], 2)
94          cv.rectangle(resized_img, (int(left), int(top)), (int(right), int(bottom)), label_colors[label_index], thickness=3)
95
96          #imagen original
97          row_factor = orig_rows/300.0
98          col_factor = orig_cols/300.0
99
100         # Se escala la detección de objetos a la imagen original
101         left = int(col_factor * left)
102         top = int(row_factor * top)
103         right   = int(col_factor * right)
104         bottom   = int(row_factor * bottom)
105
106         cv.putText(img, label_text, (int(left), int(top)), cv.FONT_HERSHEY_SIMPLEX, 0.5, label_colors[label_index], 2)
107         cv.rectangle(img, (int(left), int(top)), (int(right), int(bottom)), label_colors[label_index], thickness=3)
108
109 plt.figure(figsize=(15, 15))
110 plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
111 plt.show()
```

Explicación

Como se observa en la figura 5, se detectaron cinco objetos con un nivel de confianza superior a 0.6. En la imagen se identifican los recuadros alrededor de cada objeto, la categoría del objeto y su nivel de confianza.

Actividad

Realiza lo siguiente:

1. Implementa los cuatro pasos en la etapa de construcción de una red neuronal convolucional.

2. Practica diferentes escenarios de clasificación utilizando imágenes de diferentes dominios.

3. Usa técnicas de regularización con el objetivo de mejorar el desempeño de la CNN y agilizar su entrenamiento.

4. Haz propuestas de alternativas de uso del aprendizaje por transferencia en la industria.

5. Realiza ejercicios de aprendizaje por transferencia, resolviendo problemas de clasificación en otros dominios.

Cierre

En este proyecto se ha utilizado una CNN para la clasificación de imágenes de forma automática. El proyecto se ha dividido en cinco etapas que incluyen la creación y el entrenamiento de la red neuronal, así como la validación del modelo obtenido.

Por otro lado, el aprendizaje por transferencia es una situación en donde lo que se ha aprendido en un entorno se explota para mejorar la generalización en otro. Varios científicos coinciden en que es el siguiente paso para alcanzar la inteligencia artificial general (AGI, por sus siglas en inglés).

Bibliografía

- Gulli, A., y Pal, S. (2017). Deep learning with Keras. Inglaterra: Packt Publishing Ltd.
- Kaggle. (s.f.). Dogs vs. Cats. Recuperado de https://www.kaggle.com/c/dogs-vs-cats/data
- Keras. (s.f.). Keras: the Python deep learning API. Recuperado de https://keras.io/
- TensorFlow. (s.f.). Guide. Recuperado de https://www.tensorflow.org/guide
- Theano. (s.f.). Documentation. Recuperado de https://theano-pymc.readthedocs.io/en/latest/