ESCUELA TECNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Grado en Ingeniería en
Sistemas Audiovisuales y Multimedia

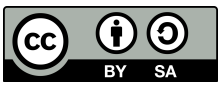
Trabajo Fin de Grado

Deep-learning para detección de vehículos

Autor: Nuria Oyaga de Frutos

Tutores: José María Cañas Plaza, Inmaculada Mora Jiménez

Curso académico 2016/2017



Índice general

Índice de figuras

Capítulo 1

Infraestructura

En este capítulo se expondrán los principales componentes software utilizados, centrados, principalmente, en la conexión con la cámara y el desarrollo, entrenamiento y test de la red neuronal. Además, se expone una descripción de las bases de datos de las que se partirá para realizar las distintas pruebas sobre la red neuronal. Estas bases de datos serán luego modificadas y adaptadas para el problema concreto que se plantee, permitiendo obtener diversas conclusiones acerca del comportamiento de la propia red y, así, emplear la más adecuada.

Software

JdeRobot

JdeRobot [1] es una plataforma de software libre que facilita la tarea de los desarrolladores del campo de la robótica, visión por computador y otras relacionadas, siendo este su principal fin.

Está escrito en su mayoría en el lenguaje C ++ y proporciona un entorno de programación basado en componentes distribuidas, de tal manera que una aplicación está formada por una colección de varios componentes asincrónos y concurrentes. Esta estructura permite la ejecución de los distintos componentes en diferentes equipos, estableciendo una conexión entre ellos mediante el middleware de comunicaciones ICE. Además, se obtiene

[1] http://jderobot.org

gran flexibiidad a la hora de desarrollar las aplicaciones, ya que estos componentes pueden escribirse en C ++, Python, Java ... y todos ellos interactúan a través de interfaces ICE explícitas.

A pesar de que esta plataforma incluye una gran variedad de herramientas y librerías para la programación de robots, y de una amplia gama de componentes previamente desarrollados para realizar tareas comunes en este ámbito, no es la verdadera finalidad del proyecto su uso, por lo que únicamente se centrará en la utilización de uno de sus componentes para facilitar la obtención de las imágenes.

Camera Server

Se trata de un componente que permite servir a un número determinado de cámaras, ya sean reales o simuladas a partir de un archivo de vídeo. Internamente gstreamer para el manejo y el procesamiento de las diferentes fuentes de vídeo.

Para su uso, es necesario editar su fichero de configuración, adaptándolo a las necesidades concretas que plantee la máquina. Dentro de este fichero se permite especificar los siguientes campos:

- Configuración de la red, donde se indica la dirección del servidor que va a recibir la petición.
- Número de cámaras que se servirán.
- Configuración de las cámaras. Se podrán modificar los siguientes campos para cada cámara:
  - Nombre y breve descripción
  - URI: string que define la fuente de vídeo
  - Numerador y denominador del frame rate
  - Altura y anchura de la imagen
  - Formato de la imagen
  - Invertir o no la imagen

Caffe

Caffe [2] es un framework de deep learnirng que permite el desarrollo, entrenamiento y evaluación de redes neuronales. Incluye, además, modelos y ejemplos previamente trabajados para un mejor entendimiento de las redes neuronales. Es una plataforma de software libre, escrito en C ++ , que utiliza la librería CUDA para el aprendizaje profundo y permite interfaces escritas en Python o Matlab.

Esta plataforma es interesante por múltiples factores. Además de incluir múltiples ejemplos y modelos ya entrenados, lo que ofrece mayor agilidad a la hora de empezar a entender el funcionamiento del aprendizaje profundo, es destacabla la velocidad que ésta ofrece para el entrenamiento de las redes y su posterior evaluación, ya que está prevista con varios indicadores que permiten evaluar la propia red y compararla con otras.

Su base se encuentra en las redes neuronales convolucionales explicadas en el Capítulo **??**, utilizando un entrenamiento por lotes. En concreto, su estructura y funcionamiento básico queda explicado en la Figura 1.1, donde se .

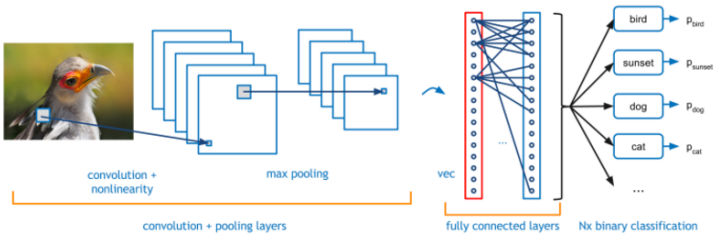


Figura 1.1: Estructura y funcionamiento básico de red en Caffe

La plataforma utiliza una serie de capas (*layers*), que, según su configuración y la distinta conexión entre ellas, permite la creación de diferentes redes neuronales. Estas etiquetas se dividen en varios grupos, en función del tipo de entrada, el tipo de salida o la función que realiza cada una de ellas. Este trabajo no utiliza todas las capas existentes en la plataforma, a continuación se explicarán cada una de las capas empleadas, clasificadas según al grupo que pertenecen.

[2] http://caffe.berkeleyvision.org/

Data Layers

Su uso se centra en la introducción de datos a la red neuronal, y estarán situadas siempre en la parte inferior de la misma. Estos datos pueden provenir de diferentes vías como bases de datos eficientes como LMDB, utilizada en este trabajo, diréctamente desde la memoria o desde archivos en disco en HDF5 o formatos de imagen comunes.

Dentro de esta capa es posible, además de especificar la ruta de los datos y el tamaño del lote (*batch*), indicar la fase en la que se utilizarán los datos, entrenamiento o test, así como algunos parámetros de transformación para el preprocesamiento de la imagen. En concreto, en este trabajo, se utilizarán datos de entrada para ambas fases y un factor de escala para establecer el rango de las imágenes en [0,1].

Vision Layers

Típicamente toman una imagen de entrada y producen otra de salida, de forma que, aplicando una operación particular a alguna región de la entrada, se obtiene la región correspondiente de la salida. Caffe dispone de varias capas de este estilo, a continuación se comentan las dos utilizadas en el trabajo.

Convolution Layer

Realiza la convolución de la imagen de entrada con un conjunto de filtros de aprendizaje, cada uno produciendo un mapa de características en la imagen de salida. Se deben especificar datos como el número de salidas, el tamaño del filtro, el desplazamiento entre cada paso del filtro, y la inicialización y relleno de los pesos y bias.

Pooling Layer

Combina la imagen de entrada tomando el máximo, el promedio, u otras operaciones dentro de las regiones, siendo su finalidad la reducción del muestreo. En esta capa se pueden especificar parámetros como el tipo de pooling a realizar, máximo, promedio o estocástico, el tamaño del filtro o el desplazamiento entre cada paso del filtro.

Common Layers

Inner Product

Calcula un producto escalar con un conjunto de pesos aprendidos, y, de manera opcional, añade sesgos. Trata la entrada como un simple vector y produce una salida en forma de otro, estableciendo la altura y el ancho de cada *bolb* en 1. Se establece el número de salidas, y la inicialización y relleno de los pesos y bias.

Dropout

Durante el entrenamiento, únicamente, establece una porción aleatoria del conjunto de entrada a 0, ajustando el resto de la magnitud del vector en consecuencia, evistando así el sobre ajuste. Se debe indicar el ration en un valor del 0 a 1, que indicará el porcentaje de muestras que se ignorarán.

Activation / Neuron Layers

En general, estas capas, son operadores de elementos, que toman un /textitbolb inferior y producen uno superior del mismo tamaño. Existen varias capas con este funcionamiento en la plataforma, en concreto se empleará la ReLu.

ReLu

Utiliza la función $y = max(0, x)$ cuya gráfica se define en la Figura 1.2

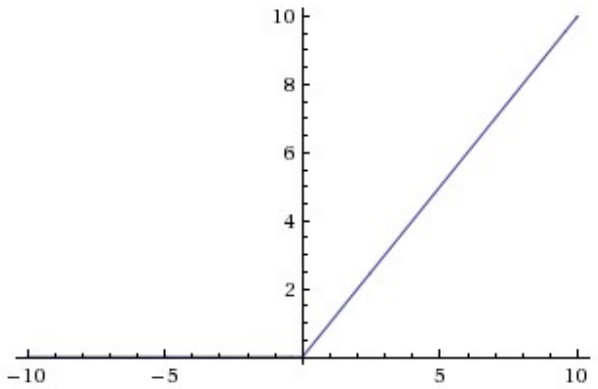


Figura 1.2: Función de activación ReLu

Loss Layers

El cálculo de la pérdida permite el aprendizaje mediante la comparación de la salida con un objetivo y la asignación de un coste para minimizarla. Se calcula mediante el paso hacia adelante. Existen diferentes medidas de las que se destacan dos.

Softmax with Loss

Se calcula como:

$$E = \frac{-1}{N} \sum_{n=1}^{N} \log(\hat{p}_{n,l_n})$$

Siendo N el número total de muestras y $\hat{p}$ las probabilidades de cada etiqueta para cada muestra.

Accuracy

Se calcula como:

$$\frac{1}{N} \sum_{n=1}^{N} \delta\{\hat{l}_n = l_n\}$$

donde $\delta\{\text{condición}\} = \begin{cases} 1 & \text{si condición} \\ 0 & \text{resto} \end{cases}$

Por último, además de las capas y parámetros definidos anteriormente, Caffe, permite el desarrollo de un *solver* en el que se podrán ajustar parámetros como el número de iteraciones totales que se ejecutarán, el de test que se van a realizar, cada cuantas iteraciones se realizarán esos test, así como se sacarán redes intermedias.
Para Caffe, el número de iteraciones no se corresponde con el número de veces que la red recorre la base de datos al completo, sino como las veces que se pasa por cada lote al completo. De esta manera, se define el número de épocas, es decir, el número de veces que se recorre de manera completa la base de datos, con la siguiente expresión:

$$\text{N.Epocas} = \frac{\text{Tamaño lote de entrenamiento} \times \text{Total iteraciones}}{\text{Muestras entrenamiento}}$$

En cuanto al número de iteraciones que se establecerán de test, se debe cumplir la siguiente igualdad:

$$\text{Iteraciones test} = \frac{\text{Muestras test}}{\text{Tamaño lote de test}}$$

Bases de datos

MNIST

MNIST [3] está formada por diferentes imágenes con números escritos a mano y consta de un conjunto de entrenamiento de 60.000 ejemplos y otro de prueba de 10.000 ejemplos. Es una buena base de datos para personas que quieren probar técnicas de aprendizaje y métodos de reconocimiento de patrones en datos del mundo real, mientras que dedican un mínimo esfuerzo a preprocesar y formatear.

Se trata de un subconjunto de una más grande, NIST, en la que las imágenes originales en blanco y negro (NIV) fueron normalizadas en el tamaño para encajar en un cuadro de 20x20 píxeles, preservando su relación de aspecto. Las imágenes obtenidas contienen niveles de gris como resultado de la técnica anti-aliasing utilizada por el algoritmo de normalización. Estas imágenes se centraron en una de 28x28 calculando el centro de masa de los píxeles y trasladando la imagen para situar este punto en el centro del campo 28x28.

Fue construida a partir de la Base de Datos Especial 3 y la Base de Datos Especial 1 del NIST, que contienen imágenes binarias de dígitos manuscritos. NIST originalmente designó SD-3 como su conjunto de entrenamiento y SD-1 como su conjunto de pruebas. Sin embargo, SD-3 es mucho más limpio y más fácil de reconocer que SD-1.Esto es debido a que SD-3 fue recogido entre los empleados de la Oficina del Censo, mientras que el SD-1 fue recogido entre los estudiantes de secundaria. Dado que para una buena extracción de conclusiones es necesario que el resultado sea independiente de la elección del conjunto de entrenamiento y de prueba entre el conjunto completo de muestras, fue necesaria la elaboración de un nuevo conjunto en el que ambas bases de datos estuviesen representadas de manera equitativa. Además, se aseguraron de que los conjuntos de escritores en el de entrenamiento y el de prueba son disjuntos.

[3] http://yann.lecun.com/exdb/mnist/

COCO

Microsoft COCO [4] es un gran conjunto de datos de imágenes diseñado para la detección de objetos, segmentación y generación de subtítulos. Alguna de las características principales de este conjunto de datos son:

- Múltiples objetos en cada imagen
- Más de 300.000 imágenes
- Más de 2 millones de instancias
- 80 categorías de objetos

Esta plataforma se ha desarrollado para varios retos, en concreto es de interés el reto de la detección, establecido en 2016. Se utilizan conjuntos de entrenamiento, prueba y validación con sus correspondientes anotaciones. COCO tiene tres tipos de anotaciones: instancias de objeto, puntos clave de objeto y leyendas de imagen, que se almacenan utilizando el formato de archivo JSON y comparten estructura de datos establecida en la Figura 1.3.

Para la detección son de interés las anotaciones de instancias de objetos, cuya estructura se muestra en la Figura 1.4.Cada anotación de instancia contiene una serie de campos, incluyendo el ID de categoría y la máscara de segmentación del objeto. El formato de segmentación depende de si la instancia representa un único objeto (iscrowd = 0), en cuyo caso se utilizan polígonos, o una colección de objetos (iscrowd = 1), en cuyo caso se utiliza RLE. Debe tenerse en cuenta que un único objeto puede requerir múltiples polígonos, y que las anotaciones de la multitud se utilizan para etiquetar grandes grupos de objetos. Además, se proporciona una caja delimitadora para cada objeto, cuyas coordenadas se miden desde la esquina superior izquierda de la imagen y están indexadas en 0. Finalmente, el campo de categorías almacena el mapeo del ID de categoría a los nombres de categoría y supercategoría.

[4] http://mscoco.org/

```
{
  "info"          :info,
  "images"        :[image],
  "annotations"   :[annotation],
  "licenses"      :[license],
}

info{
  "year"          :int,
  "version"       :str,
  "description"   :str,
  "contributor"   :str,
  "url"           :str,
  "date_created"  :datetime,
}

image{
  "id"            :int,
  "width"         :int,
  "height"        :int,
  "file_name"     :str,
  "license"       :int,
  "flickr_url"    :str,
  "coco_url"      :str,
  "date_captured" :datetime,
}

license{
  "id"            :int,
  "name"          :str,
  "url"           :str,
}
```

Figura 1.3: Estructura básica de anotaciones

```
annotation{
  "id"            :int,
  "image_id"      :int,
  "category_id"   :int,
  "segmentation"  :RLE or [polygon],
  "area"          :float,
  "bbox"          :[x,y,width,height],
  "iscrowd"       :0 or 1,
}

categories[{
  "id"            :int,
  "name"          :str,
  "supercategory" :str,
}]
```

Figura 1.4: Estructura de instancias de objetos

Capítulo 2

Desarrollo

En este capítulo se expondrá el trabajo realizado para el entendimiento del problema de clasificación, elaborando un amplio estudio sobre las variantes posibles sobre las redes entrenadas, y una primera aproximación a la detección,todo ello con redes neuronales desarrolladas sobre la plataforma Caffe.

Clasificador de dígitos

Para el estudio de la clasificación empleando redes neuronales desarrolladas con Caffe, se optó por la transformación, según determinados criterios, de la red básica proporcionada por la propia plataforma en este ámbito.

Red básica

Esta red está orientada a la clasificación de números utilizando en el entrenamiento la base de datos numérica MNIST, explicada en el Capítulo 1. Para su entrenamiento, Caffe proporciona tres archivos que se deberán editar para adaptar la red al problema que se abarque.

Definición de la red

Caffe utiliza el archivo *lenet_train_test.prototxt* para la especificación de todos los parámetros que son necesarios en el entrenamiento de la red, es decir, define las

imágenes que se emplearán, la propia estructura de la red y la forma en la que se analizarán las imágenes proporcionadas, todo ello empleando diferentes capas (*layers*).

La primera línea de este documento es utilizada para indicar el nombre que se le quiere dar a la red.

```
name: "LeNet"
```

En concreto, esta red recibe el nombre de LeNet, un tipo de red que es conocida por un buen funcionamiento en las tareas de clasificiación de dígitos y que, por lo general, consta de una capa convolucional seguida por una capa de agrupamiento (*pooling*), repetido dos veces y, finalmente, dos capas totalmente conectadas similares a las perceptrones multicapa convencionales. En el ejemplo de Caffe, la estructura habitual de la red LeNet se ve ligeramente modificada, ya que en lugar de emplear una función de activacion sigmoidal se utiliza una linear.

Tras la definición del nombre se definen dos capas de datos, una de ellas correspondiente a los datos de entrenamiento de la red y, la otra, correspondiente a los datos que se utilizarán para realizar el test durante el entrenamiento para obtener datos de *accuracy* y *loss*.

```
layer {
    name: "mnist"
    type: "Data"
    top: "data"
    top: "label"
    include {phase: TRAIN}
    transform_param {scale: 0.00390625}
    data_param {
        source: "examples/mnist/mnist_train_lmdb"
        batch_size: 64
        backend: LMDB
    }
}
```

Es importante que los parámetros de transformación, en este caso un factor de escala que establece el rango de la imagen en [0,1], sean los mismos en ambas fases, pues si se evaluase la red con una transformación de la imágen distinta a la aplicada en el entrenamiento los resultados obtenidos no serían reales.
Se utilizará, por tanto, dos capas de datos que difieren en la fase en la que se utilizarán los datos, entrenamiento o evaluación de la red, el tamaño del lote, siendo 64 muestras para el entrenamiento y 100 para el test, y la ruta de la que se cogen los datos.

A continuación, se comienzan a definir las capas del entrenamiento propiamente dicho. Se intercala una capa de convolución con una de agrupamiento y se repite dos veces.

```
layer {
    name: "conv1"
    type: "Convolution"
    bottom: "data"
    top: "conv1"
    param {lr_mult: 1}
    param {lr_mult: 2}
    convolution_param {
        num_output: 20
        kernel_size: 5
        stride: 1
        weight_filler {type: "xavier"}
        bias_filler {type: "constant"}
    }
}
```

En la capa de convolución, explicada en el Capítulo 1, se define que el tamaño del filtro será de 5x5 y que se obtendrán 20 salidas, en la segunda capa de convolución, sin embargo, se obtendrán 50 salidas. Además se define el algoritmo ”Xavier”para la inicialización de los pesos, que determina automáticamente la escala de inicialización basada en el número de entradas y de las neuronas de salida, y la inicialización del *bias* mediante una constante que por defecto es 0.

```
layer {
    name: "pool1"
    type: "Pooling"
    bottom: "conv1"
    top: "pool1"
    pooling_param {
        pool: MAX
        kernel_size: 2
        stride: 2
    }
}
```

La capa de agrupamiento, también explicada en el Capítulo 1, será alimentada por la capa de convolución anterior y alimentará a la siguiente en caso de que la haya. Se definen en ella un tamaño de filtro de 2x2, un intervalo de dos muestras entre cada aplicación del filtro, por lo que no hay solape, y el método del máximo para realizar el agrupamiento.

Definición del solucionador

Ejecución de la red