UNIVERSIDAD
DE GRANADA

Facultad de Ciencias
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicaciones

Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Localización de Regiones de Interés Utilizando Aprendizaje Profundo

Presentado por:
Laura Gómez Garrido

Tutor:
Jesús Chamorro Martínez
Ciencias de la Computación e Inteligencia Artificial

Curso académico 2019-2020

Localización de Regiones de Interés Utilizando Aprendizaje Profundo

Laura Gómez Garrido

Laura Gómez Garrido *Localización de Regiones de Interés Utilizando Aprendizaje Profundo*.
Trabajo de fin de Grado. Curso académico 2019-2020.

Responsable de tutorización	Jesús Chamorro Martínez *Ciencias de la Computación e Inteligencia Artificial*	Grado en Ingeniería Informática y Matemáticas Facultad de Ciencias Escuela Técnica Superior de Ingenierías Informática y de Telecomunicaciones Universidad de Granada

Declaración de originalidad

D./Dña. Laura Gómez Garrido

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2019-2020, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 15 de julio de 2020

Fdo: Laura Gómez Garrido

Dedicatoria (opcional)
Ver archivo `preliminares/dedicatoria.tex`

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Agradecimientos del libro (opcional, ver archivo `preliminares/agradecimiento.tex`).

Summary

An english summary of the project (around 800 and 1500 words are recommended).
File: `preliminares/summary.tex`

Introducción

De acuerdo con la comisión de grado, el TFG debe incluir una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas.

Ver archivo `preliminares/introduccion.tex`

Parte I.

Estado del arte

A continuación, explicaremos de forma concisa lo que es una *Red Neuronal* y una *Red Neuronal Convolucionada* para, seguidamente, hablar sobre los últimos avances en la localización de regiones de interés en imágenes.

1. Red Neuronal

1.1. El problema de clasificar una imagen.

Cuando observamos una imagen, podemos localizar varios elementos a partir de los cuáles esta se encuentra compuesta con tan sólo un vistazo. Sin embargo, para un ordenador no se trata de algo tan sencillo puesto que para él se trata de un gran conjunto de números que no tienen por qué tener relación alguna entre sí.

Figura 1.1.: Cómo un ordenador ve una imagen

Idealmente, en esta imagen desearíamos que fuera capaz de identificar que se trata de un perro, en concreto de un chiguagua, de pelaje blanco y manchas color café que se encuentra sobre un césped. Estos poquitos datos que para nosotros parecen tan triviales necesitan de horas y horas de computación para ser obtenidos a partir de una imagen cualquiera.

Dejamos todos estos detalles, a los cuales esperamos poder llegar en un futuro no muy lejano, y simplificamos el problema a tener un conjunto de etiquetas y buscar con cuál de

todas ellas tiene mayor relación nuestra imagen.

Una primera idea, sería utilizar utilizar *k-Nearest Neighbor Classifier*, en adelante *k*-NN. Este clasificador consiste en que para cada etiqueta se correspondan k imágenes y consideraremos como etiqueta idónea para nuestra clasificación aquella cuya distancia, siendo Manhattan y Euclídea las más comunes, entre los píxeles con nuestra imagen sea menor. Nos encontramos con que el tiempo de entrenamiento de nuestro clasificador sería ínfimo en comparación con el tiempo de clasificación, si bien se pueden hacer diversas mejoras que cambien estos hechos.

De esta sencilla propuesta, surgen diversos problemas. Al darle mayor importancia al valor concreto de los píxeles, en lugar de a las formas que de las que está compuesta la figura, nos encontramos con que valores como los colores de fondo pueden influir más en la clasificación que los propios píxeles de la figura que queremos clasificar. En el ejemplo de la imagen del chiguagua, si considerásemos las etiquetas verde y perro, tendríamos que por lo general como respuesta el color verde, pese a que estamos más interesados por la mascota en sí. Otro gran problema, sería la baja escalabilidad que nos proporciona esta solución, al incrementarse enormemente el costo computacional de clasificación conforme aumenta el número de etiquetas.

El siguiente paso, es buscar una forma de "memorizar" los datos de entrenamiento de forma que no tengamos que estar comparándolos con todos ellos cuando queramos clasificar una imagen. Lo que buscamos es poder valorar de alguna forma la imagen completa y a partir de esta "puntuación" conocer qué etiqueta le corresponde mejor a nuestra imagen. De esta forma, veríamos drásticamente reducido el tiempo de clasificación, sacrificando para ello el tiempo de entrenamiento, y podríamos ampliar en varias unidades la cantidad de datos de entrenamiento utilizados, aumentando así la precisión de nuestro clasificador.

1.2. Clasificadores Lineales

Antes de nada, vamos a comenzar contextualizando matemáticamente el entorno en el que nos encontramos. Una vez realizado esto podremos hablar correctamente de los clasificadores lineales y así poder extenderlos naturalmente a los conceptos de Red Neuronal y de Red Neuronal Convolucionada.

Sea $D \in \mathbb{N}$ la dimensión de nuestras imágenes, por lo general el número de píxeles que estas poseen, y $K \in \mathbb{N}$ la cantidad de etiquetas o categorías bajo las cuales pueden ser clasificadas. Siendo $N \in \mathbb{N}$ el número de ejemplos que utilizaremos para entrenar nuestro clasificador, tendremos que para cada dato de entrenamiento $x_i \in \mathbb{R}^D$ $i = 1, ..., N$ le corresponde una etiqueta $y_i \in 1, ..., K$ tal que juntos conforman el par (x_i, y_i) de imagen y categoría a la que pertenece.

Para que no sea más fácil de entender este contexto, tomaremos como ejemplo el conjunto de datos CIFAR-10 que consiste en 60000 imágenes RGB de dimensión 32x32 y 10 categorías. De esta forma, tendríamos que $D = 32 \cdot 32 \cdot 3 = 3072$, $K = 10$ y $N = 60000$.

Definición 1.1. Definiremos la *función de puntuación* o *score function* como una función $f : \mathbb{R}^D \to \mathbb{R}^K$ que asigna los píxeles de la imagen sin procesar una serie de puntuaciones para cada etiqueta.

La función de puntuación nos revela la probabilidad que tiene cada imagen de pertenecer a cada una de la distintas etiquetas de las que disponemos. Dicho esto, podemos definir un *clasificador lineal* o *linear classifier* como aquel que utiliza una función de puntuación lineal, es decir, de la forma:

$$f(x) = W \cdot x + b \quad W \in M_{K,D}(\mathbb{R}) \;\; b \in \mathbb{R}^K \;\; \forall x \in \mathbb{R}^D$$

Dicho esto, existen diferentes tipos de clasificadores lineales y la principal diferencia entre ellos reside en la función de pérdida que utilicen a la hora de entrenar la red.

Definición 1.2. Una *función de pérdida* o *loss function* es aquella que durante el entrenamiento de un clasificador se encarga de penalizar las etiquetas incorrectas.

FIXME: ¿Hablar de SVM, SoftMax y Cross-Entropy?

Llegados a este punto, debemos de explicar cómo funcionan los clasificadores lineales, es decir, cómo estos son entrenados y para qué se emplean las funciones de puntuación y de pérdida. Dado poseemos un conjunto de ejemplos con su correspondiente etiqueta pero desconocemos el valor de los parámetros W y b, nuestro objetivo es utilizar dichos datos de entrenamiento para estimar estos valores.

Para ello, comenzamos haciendo una conjetura sobre un posible valor para nuestros parámetros, evaluamos nuestra función de puntuación con todos nuestros datos de entrenamiento y seguidamente utilizamos la función de pérdida para estimar cómo de bien funciona nuestra conjetura. Analizamos los resultados y hacemos una nueva conjetura que los mejore, repitiendo el proceso un número lo suficientemente grande de veces.

Llegados a este punto, podemos preguntarnos cómo realizamos la nueva conjetura de forma que nos aseguremos tener unos resultados mejores. Hacerlo de forma totalmente aleatoria, repitiéndolo hasta obtener una pérdida lo suficientemente pequeña, no parece una buena idea puesto que es difícil saber cuándo encontraremos una buena respuesta. Entre las distintas técnicas, nos encontramos con las basadas en la *búsqueda aleatoria local* y las que se basan en el *Gradiente Descendiente*, entre otras muchas. Como ejemplo, tomaremos el algoritmo del gradiente descendiente y minimizaremos la función de pérdida para llegar a aquellos pesos que menor error nos den.

FIXME: Enlazar pdfs que expliquen bien cómo funciona el gradiente descendiente y la búsqueda aleatoria local

```
while condicion_de_parada :
  weight_grad=evaluate_grad(loss_fun,x,y)
  weight = -step_size*weight_grad
```

El código mostrado más arriba se trata de una versión muy simplificada de cómo funcionaría de cómo podríamos implementar el algoritmo, teniendo en cuenta que nosotros mismos podremos modificar la condición de parada de acuerdo a nuestras necesidades y que el valor *step_size* es algo que podremos fijar de la misma forma, sabiendo que este nos indica cuánto queremos avanzar en la dirección que nos indica el gradiente. FIXME: Enlazar

pdfs que expliquen la importancia de estos valores y cómo fijarlos.

FIXME: Añadir implementación en Tensorflow de un clasificador lineal.

Ejemplo 1.1. Fijemos una etiqueta, $y = 0$ por lo que $K = 1$. Tendríamos que nuestra función de puntuación sería $f : \mathbb{R}^D \to \mathbb{R}$ donde $W = (w_1, ...w_D) \in \mathbb{R}^D$ y $b \in \mathbb{R}$ luego

$$f(x) = W \cdot x + b = \sum_{j=1}^{D} w_j x_j + b.$$

Así mismo, tomamos como función de pérdida

$$L(x, f) = -\ln \frac{1}{\sum_{j=1}^{D} e^{f_j}} = -\ln \frac{1}{\sum_{j=1}^{D} e^{w_j x_j + b}} = \sum_{j=1}^{D} e^{w_j x_j + b}$$

que es un caso particular de *Entropía Cruzada* o *Cross-entropy*.

Este clasificador recibe el nombre de *Binary Softmax classifier* o *Binary Logistic Regression classifier*. De esta forma, si consideramos la función probabilística *sigmoide* tendríamos que el minimizar la función de pérdida estaríamos aumentando la probabilidad clasificar correctamente nuestros datos puesto que:

$$P(y = 0|x; W) = \sigma(f(x)) = \frac{1}{1 + e^{-f(x)}}.$$

Observación 1.1. Estamos calculando la probabilidad de coincidir o no con una determinada etiqueta, recibe el nombre de binario porque también podríamos considerar que tenemos dos etiquetas y que si no perteneces a una forzosamente perteneces a la otra. Este modelo puede por tanto construirse también utilizando ambas etiquetas y con mejores resultados a la hora de clasificar puesto que durante el entrenamiento la función de pérdida es ligeramente modificada para tener en cuenta la otra etiqueta. En cualquier caso, ambas construcciones siguen la misma interpretación probabilística y, además, tenemos que $P(y = 1|x; W) = 1 - P(y = 0|x; W)$.

Ejemplo 1.2. Tomamos como función de pérdida $L(x, f) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f_j - f_{yi} + \Delta)$ donde $\Delta \in \mathbb{R}$ es un parámetro que se suele ajustar utilizando una técnica llamada *validación cruzada* o *cross validation*. Esta función suele recibir el nombre de *hinge loss* y el clasificador lineal es conocido como *Multiclass Suport Vector Machine (SVM)*. Este clasificador "quiere"que la etiqueta correcta para cada imagen tenga una puntuación mayor que las incorrectas por un margen fijo Δ.

1.3. El modelo de una neurona

Cuando comenzamos planteando nuestro problema buscábamos conseguir clasificar una imagen con una probabilidad de acertar similar o superior a cómo lo haríamos nosotros mismos. Teníamos el problema de que un ordenador no era capaz de pensar o razonar de la misma forma que un ser humano y buscábamos una forma de clasificar esta información a pesar de ello. Es aquí donde nacen los modelos de redes neuronales, en un intento por ser capaces de simular cómo funciona nuestro propio cerebro en nuestros clasificadores y que

estos sean capaces de aprender cómo reconocer los distintos elementos de la misma forma que nosotros lo hemos ido haciendo a lo largo de nuestra vida.

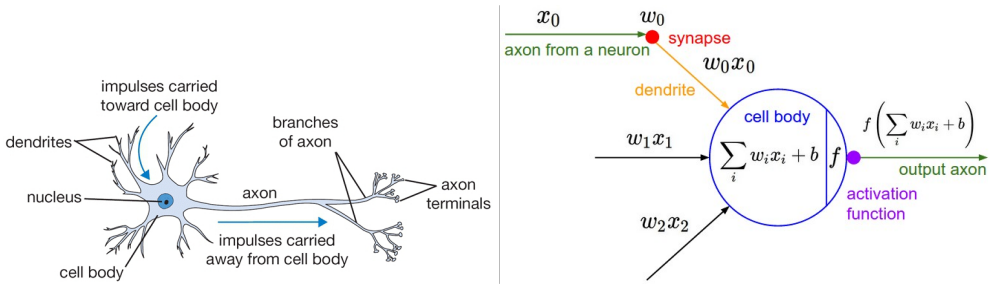

Figura 1.2.: Comparación entre una neurona biológica (izquierda) y el modelo matemático (derecha)

Nuestro cerebro está formado por múltiples neuronas interconectadas entre sí que están constantemente transmitiéndose información y aprendiendo a través de todos los datos que reciben. Si nos fijamos en 1.2 podemos ver que una neurona real esta formada por dentritas que son quienes, a través del proceso de sinapsis, reciben la entrada de información, que es asimilada y transformada por su núcleo antes de ser transmitida a la siguientes neuronas a través de las divisiones de su axón en caso de que supere un cierto umbral y se active.

De esta forma, si consideramos que tenemos D dentritas y que por la i-dentrita recibimos el la información x_i con un peso o fuerza de sinapsis w_i, tendríamos que nuestro núcleo trabaja con el vector de información $(w_1x_1, ..., w_Dx_D)$ que podemos condensar utilizando la norma $||\cdot||_1$ como $\sum_i w_ix_i$ y sumarle una determinada constante b propia de la neurona. El umbral de activación y la señal enviada al resto de neuronas la representaremos con la *función de activación* que será una función $f : \mathbb{R} \to \mathbb{R}$.

A continuación, mencionaremos los ejemplos más representativos utilizados como funciones de activación, enlazándolos con lo ya visto anteriormente:

- *Función sigmoide* $\sigma : \mathbb{R} \to [0,1]$ definida como $\sigma(x) = \frac{1}{1+e^{-x}}$. En este caso, nos encontramos con el ejemplo 1.1 transformado en una neurona cuya función de pérdida utilizada durante el entrenamiento suele ser la misma que en el ejemplo mencionado. Se suele utilizar en la última capa de nuestra red, cuando nuestras imágenes pueden pertenecer a varias clases o etiquetas al mismo tiempo.

- *Función Softmax* utilizada normalmente en la última capa donde hay tantas neuronas como etiquetas y en el problema de clasificación donde una imagen puede pertenecer únicamente a una sola etiqueta o caso. Se trata de una modificación de función la sigmoide que regulariza la salida para obtener la probabilidad de pertenecer a cada clase como sucesos independientes obteniendo así que la suma de todas las salidas de esta capa sería 1. Aquí, la i-neurona tendría función de activación $\sigma_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ y utilizaría $L(x) = \frac{1}{N}\sum_{i=1}^{N} -\ln \frac{e^{fy_i}}{\sum_{j=1}^{D} e^{f_j}} = \frac{1}{N}\sum_{i=1}^{N}(-f_{yi} + \ln \sum_{j=1}^{D} e_j^f)$ como función de pérdida para el entrenamiento.

- *Función ReLU*, cuyo nombre completo sería *Rectified Lineal Unit*. Se suele utilizar en las capas intermedias de nuestra red y utiliza como función de activación $f(x) = \max(0, x)$ que nos recuerda al ejemplo 1.3.
- *Función Tanh* se trata de de una centralización de la función sigmoide. $Tanh(x) = 2\sigma(x) - 1$.

1.4. La red completa

A. Primer apéndice

Los apéndices son opcionales.
Archivo: `apendices/apendice01.tex`

Glosario

La inclusión de un glosario es opcional.
Archivo: `glosario.tex`

$\mathbb{R}$ Conjunto de números reales.

$\mathbb{C}$ Conjunto de números complejos.

$\mathbb{Z}$ Conjunto de números enteros.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.