# Descripción de los colores de una imagen mediante técnicas de Deep learning

Oscar Prades Palacios

**Resumen–** En este proyecto proponemos una solución para la descripción de los colores de una imagen, basada en el modelo propuesto por Wang *et al.* [10]. En una primera fase, se implementa un modelo auto-supervisado en el cual se utilizan como supervisión los histogramas de color generados por las mismas imágenes utilizadas durante el aprendizaje. Este modelo refleja las características del color de las imágenes sin la influencia de muestras erróneas. Para lidiar con las imágenes mal etiquetadas, implementamos un método de selección de muestras que filtra dichas imágenes y las descarta de la fase de entrenamiento del modelo. En una segunda fase, procedemos a refinar el modelo para que aprenda a etiquetar los colores con su respectivo nombre. Se propone una mejora respecto al modelo previo utilizando imágenes distintas durante el entrenamiento, obteniendo así un modelo con mayor rendimiento. Para finalizar se hacen varios experimentos con todos los modelos implementados para poder analizar y comparar su rendimiento.

**Palabras Clave–** Redes neuronales, CNN, Etiquetado de colores, Tensorflow, Aprendizaje auto-supervisado

**Abstract–** The aim of this project is to propose a solution for the description of the colors of an image, based on the model proposed by Wang *et al.* [10]. In a first stage, a self-supervised model is implemented and trained with the color histograms generated by the images used in the learning process. This model reflects the color characteristics of images without the influence of noisy samples. To deal with poorly labeled images, we implemented a sample selection method that filters these images and excludes them from the training phase of the model. In a second stage, we proceed to refine the model so it learns to label the colors with their appropriate name. It is proposed an improvement over the previous model using different images in the process training, obtaining a model with higher performance. Finally, several experiments are carried out with all models implemented in order to analyze and compare their performance.

**Keywords–** Neural networks, CNN, Color naming, Tensorflow, Self-supervised learning

## 1 Introducción

Los colores que percibimos, se pueden describir por varios aspectos como matiz, brillo, saturación, etc. Sin embargo, aunque el sistema visual humano funciona aproximadamente para todos por igual, las interpretaciones que hace nuestro cerebro de estos estímulos de entrada pueden ser diferentes según el observador.

El sistema visual es un conjunto de órganos, vías y centros nerviosos que permiten la captación, procesamiento y aprovechamiento de la información visual que nos llega a través del globo ocular, y que mediante un proceso de transducción de la información visual recibida desde nuestro campo de visión, hace que seamos capaces de alcanzar una percepción muy precisa del mundo físico que nos rodea. Las células o mecanismos encargados de captar la luz en nuestro sistema visual son llamados fotorreceptores. Éstos se localizan en el interior del ojo y existen dos tipos: conos y bastones. Mientras que los conos forman una especie de mosaico hexagonal, los bastones están distribuidos de forma más desorganizada que los conos. Existe una zona del globo ocular donde no hay ningún tipo de fotoreceptor, esta zona es denominada el punto ciego.

Los bastones son los responsables de la visión escotópica (condiciones de baja luminosidad), los conos por su parte, contienen tres tipos de opsinas y cada una de ellas tiene una mayor sensibilidad a diferentes longitudes de onda. Por ello los conos son los responsables de la percepción del color y dan lugar a la visión tricromática. De aquí proviene el co-

• E-mail de contacto: oskarpras@gmail.com
• Mención realizada: Computación
• Trabajo tutorizado por: Robert Benavente Vidal (Departamento de ciencias de la computación)
• Curso 2016/17

nocido espacio de color RGB. La interpretación final de los colores que percibimos se hace hace en base a unas categorías de color que tienen un nombre asociado [1, 2].

Según Correa *et al*. [3], la percepción del color puede variar entre personas o incluso puede deberse a factores no fisiológicos. Por ejemplo la forma de los objetos, la distancia de la imagen respecto al observador, la luminosidad del objeto observado, e incluso puede deberse a factores culturales y/o sociales. Todas estas diferencias hacen que el problema de la identificación o clasificación de colores de forma computacional, sea un problema complejo y en ocasiones muy costoso.

En este proyecto se propone abordar el problema de la clasificación automática de los colores de una imagen mediante el uso de redes neuronales. El objetivo es crear un sistema automático capaz de identificar cada color de una imagen y etiquetarlo con su respectivo nombre. Nos basaremos en la definición de los 11 colores básicos definidos por Berlin y Kay [4] [1], para así poder etiquetar cualquier conjunto de colores de una imagen en un grupo reducido de ellos.

Las aplicaciones practicas que se le puede dar a un sistema automático de este tipo son muy variadas. Desde sistemas autónomos que basan sus decisiones en imágenes que requieren información del color, etiquetado automático de imágenes para su posterior tratamiento, sistemas de ayuda y/o transcripción de colores para personas con problemas de visión, etc.

## 2 OBJETIVOS

El objetivo general del proyecto consiste en dar una solución al problema del etiquetado del color de una imagen. Para ello se pretende desarrollar y documentar una arquitectura de red neuronal convolucional capaz de proporcionar una resolución competente del problema, basándonos en la descripción de los 11 colores básicos definidos por Berlin y Kay [4]. Más específicamente los objetivos son:

- Estudiar e investigar las redes neuronales existentes que abordan el problema de la clasificación de colores y su etiquetado (*color naming*).
- Análisis de las herramientas disponibles para trabajar con redes neuronales.
- Desarrollar, implementar y entrenar un modelo de red neuronal convolucional a partir de las estudiadas en el punto anterior.
- Estudiar la posibilidad de introducir mejoras en la red implementada.
- Documentar, analizar y comparar el rendimiento de todas las arquitecturas de red implementadas, para su posterior conclusión.

## 3 ESTADO DEL ARTE

El problema de la asignación de nombres a los colores, se ha estudiado principalmente en campos como la psicología y la lingüística. Sin embargo, a partir del año 2000, gracias a la evolución de los sistemas computacionales, surge el interés por la automatización de esta tarea. Existen varias investigaciones con resultados interesantes resolviendo el problema de la clasificación de color. Como por ejemplo, el trabajo de Menegaz *et al.* [5], en el que se propone modelar cada color como un conjunto difuso, cuya función de pertenencia se define ajustando el modelo a los resultados de un experimento psicofísico. O los trabajos de Benavente *et al.* [6, 7]. En el primero se presenta un conjunto de datos obtenido a partir de un experimento de etiquetado de colores y se ofrece un amplio análisis mediante conjuntos estadísticos. En el segundo, se presenta un modelo paramétrico, donde cada categoría de color es modelada como un conjunto difuso con una función de pertenencia paramétrica. También podemos destacar el trabajo realizado por van de Weijer *et al.* [8] donde se utiliza el método del análisis sintáctico probabilístico latente (PLSA [9]), para aprender la información del color de imágenes extraídas de Google Imágenes.

En cuánto a la clasificación de color mediante redes neuronales convolucionales, destacamos el trabajo realizado por Wang *et al.* [10] en el cual se basa este proyecto. En él, se aborda el problema de la clasificación del color de una imagen mediante una red, bajo la implementación de un framework definido en dos fases. También es interesante el estudio realizado por Rafegas y Vanrell [11], donde se profundiza en como el color es codificado en una red neuronal convolucional. Uno de los trabajos más recientes a destacar es el llevado a cabo por Cheng *et al.* [12], aparte de contribuir con un conjunto de datos extenso para la clasificación de color, se aborda el problema con una red neuronal convolucional para la clasificación píxel a píxel.

La mayoría de los trabajos revisados actualmente sobre redes neuronales convolucionales se enfocan en detección de bordes, segmentación de formas, detección de movimiento y sobretodo en el reconocimiento y clasificación de imágenes o de características asociadas a ellas [13].

## 4 METODOLOGÍA

Para dar una solución al problema de la clasificación de colores, se pretende hacer uso de redes neuronales convolucionales (*Convolutional Neural Networks*, CNN). Estas redes están inspiradas en el comportamiento biológico de nuestro cerebro. Las entradas equivalen a las señales recibidas por los fotorreceptores del sistema visual, y las salidas a la respectiva información que recibimos de nuestro cerebro una vez que la información ha sido transferida e interpretada. En la figura (1) se muestra la equivalencia entre una neurona biológica y la adaptación artificial de ésta.

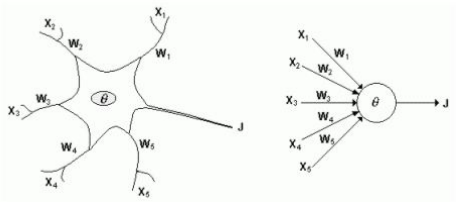

Fig. 1: Representación computacional de una neurona.

Las redes neuronales son sistemas de aprendizaje computacional capaces de aproximar un modelo matemático a ba-

[1] negro, azul, marrón, verde, gris, naranja, rosa, lila, rojo, blanco y amarillo.

se de recibir datos de entrada, proporcionar datos de salida y de comparar estos últimos con datos correctos verificados. A partir del modelo de error resultante, las redes son capaces de modificar valores en su estructura para aproximarse al resultado correcto esperado. El proceso de proporcionar datos de entrada a una red para que reajuste su estructura se llama entrenamiento. Cuando una red ha recibido suficiente información, llega al estado deseado y decimos que la red ha sido entrenada.

## 4.1. Propuesta de la arquitectura de la red

Después de la revisión del estado del arte y del análisis de las posibles soluciones existentes para el problema del etiquetado de colores, se decide basar nuestra solución en el modelo propuesto por Wang *et al.* [10]. En su trabajo proponen una arquitectura de red que consta de dos partes bien definidas:

- En una primera fase se entrena un modelo de CNN con capas de extracción de características generalizadas de manera que se puedan preservar y concentrar las características del color de las imágenes, sin la influencia de un etiquetado previo de los colores. Para ello se propone un modelo auto-supervisado de CNN (SS-CNN) entrenado con recortes de imágenes seleccionadas aleatoriamente del conjunto de entrenamiento y se utilizan los histogramas del color de esos recortes como su propia supervisión para el entrenamiento del modelo.
- La segunda fase trata de afinar la SS-CNN obtenida en la primera fase y re-entrenar solo la última capa de la CNN con imágenes etiquetadas con su nombre de color correspondiente. El modelo obtenido está claramente afectado por las muestras mal etiquetadas. Debido a esto se aplica un proceso de refinamiento sobre las muestras del conjunto de aprendizaje, para desechar aquellas que añaden ruido al proceso de entrenamiento de la CNN. La idea es predecir la etiqueta de todos los recortes de las imágenes de entrenamiento y para cada color, seleccionar las N muestras etiquetadas correctamente con la predicción más alta y asignarlas como "semillas". A continuación, mediante las características extraídas de la tercera capa del modelo, se calcula la media de estas "semillas" en cada uno de los 11 colores y se asigna un centro de clase para cada color. Finalmente, las muestras que quedan demasiado alejadas de estos centros, son desechadas. Una vez se obtienen las muestras de entrenamiento "purificadas", se procede con otra fase de entrenamiento de la red con las muestras filtradas. La selección de muestras y el proceso de entrenamiento se aplican N veces sobre el modelo.

## 4.2. Herramientas (APIS)

Existe una gran variedad de librerías para implementar CNN's. En esta sección vamos a revisar algunas de ellas y analizar qué ventajas nos pueden ofrecer.

- Caffe [14]: Librería desarrollada por el Berkeley Vision and Learning Center (BVLC) y por contribuciones de la comunidad. Tiene muy buena acogida por la comunidad y un gran soporte por parte del equipo. También ofrece una gran velocidad a la hora de procesar imágenes, alrededor de 60 millones de imágenes por día con una única NVIDIA K40 GPU.
- Torch [15]: Framework de computación científica para LuaJIT [16] con un amplio soporte para machine learning, especializado en algoritmos destinados para GPU. Desarrollada bajo licencia open source.
- Tensorflow [17]: Es una librería open source creada por el equipo de Google Brain Team [18], que permite hacer cálculos numéricos usando diagramas de flujo de datos. Debido a su arquitectura flexible en forma de grafo permite su ejecución en varias CPU's o GPU's y en diferentes dispositivos. La API de Tensorflow está disponible para varios lenguajes de programación, aunque el único que ofrece una estabilidad total de la API por el momento es Python.
- MatConvNet [19]: Es una toolbox (librería) de Matlab desarrollada por el equipo de Andrea Vedaldi de la Universidad de Oxford. Implementa CNN's, esencialmente destinadas para aplicaciones de visión por computador. Incluye algunos modelos de CNN's ya entrenadas para clasificación de imágenes, reconocimiento facial, segmentación, etc. También incluye las herramientas necesarias para la ejecución sobre GPU.
- Keras [20]: Es una API de alto nivel para desarrollar redes neuronales, escrita en Python y capaz de correr por encima de Tensorflow o Theano, actuando como *wrapper* de dichas librerías. De este modo permite hacer prototipos de redes neuronales de una forma rápida y sencilla. Ofrece soporte tanto para redes convolucionales como para redes recurrentes o la combinación de ambas.
- Theano [21]: Es una librería de Python que permite definir, optimizar y evaluar expresiones matemáticas eficientemente. Su uso para trabajar con redes neuronales es bastante extendido.

Después de haber analizado varias librerías como base para nuestro proyecto, se decide utilizar Tensorflow sobre Python v3.5 para implementar nuestro modelo de CNN.

Tensorflow es una herramienta muy utilizada y con un gran soporte (Apéndice A.3). La implementación que hace de modelos de clasificación es muy atractiva. El modelo construido es representado como un grafo en el que los nodos son las operaciones y las aristas las transmisiones de datos entre estos nodos. Este grafo generado, se inicializa y se lanza para su procesamiento en la computadora mediante el uso de sesiones predefinidas en el programa, obteniendo así una mejora en el rendimiento del modelo respecto a otras implementaciones. Estas sesiones son capaces de mantener un registro de todas las operaciones, variables y resultados que forman nuestro modelo, para así poder recuperar un estado concreto.

## 4.3. Conjuntos de imágenes (*Datasets*)

Para llevar a cabo el proyecto se van a utilizar dos conjuntos de imágenes distintos [22], creados en los trabajos de

van de Weijer *et al.* [8, 23] sobre el etiquetado de los colores en imágenes.

- Ebay colors: Este conjunto de imágenes ha sido recolectado del sitio de subastas Ebay (www.ebay.com). El conjunto de imágenes contiene 4 clases distintas: coches, zapatos, vestidos y alfarería. Cada clase contiene 10 imágenes dedicadas a testing y 2 de evaluación para cada uno de los 11 colores básicos. El etiquetado de los colores de las imágenes ha sido realizado por usuarios de Ebay. Cada imagen también tiene su correspondiente máscara de la región que ocupa el objeto en la imagen (Figura 2).
- Google colors: Este conjunto de imágenes contiene 100 imágenes para cada uno de los 11 colores básicos. Las imágenes han sido recolectadas mediante el buscador de Google imágenes, especificando en cada búsqueda el color esperado.

Fig. 2: Imágenes de Ebay y sus respectivas máscaras.

## 5 Implementación de una CNN para el etiquetado de colores

La CNN implementada en este trabajo consta principalmente de dos fases explicadas en la sección 4.1. Primero entrenamos el modelo SS-CNN para aprender características básicas de los colores. Seguidamente, procedemos a reentrenar la última capa del primer modelo para obtener el etiquetado de la imagen con el nombre del color al cual pertenece. Los pasos seguidos son:

- Implementar el algoritmo k-means y etiquetar las imágenes de entrenamiento con su respectivo histograma de color (Sección 5.1).
- Implementar la arquitectura de la SS-CNN (Sección 5.2).
- Refinamiento de la red (Sección 5.3).

### 5.1. Generación de los histogramas

El modelo SS-CNN es entrenado con los histogramas de color de las imágenes de entrenamiento como supervisión, ya que se considera que estos histogramas describen la distribución de probabilidad de diferentes colores en cada una de las imágenes utilizadas como entrenamiento. No se utiliza la implementación habitual del histograma que divide el espacio de color de una forma regular, porque la distribución de los colores en el espacio RGB es desigual. Algunos colores, ocupan áreas muy pequeñas y pueden ser difíciles de separar con una división regular del espacio de color. Por esta razón, se utiliza el algoritmo k-means (ver sección A.2) para clasificar todos los píxeles que provienen del conjunto de entrenamiento en 128 clústers representados en el espacio RGB. Estos clústers obtenidos como resultado del k-means, se utilizan como centros de clase para construir cada uno de los histogramas de cada imagen, clasificando cada uno de los píxeles de dicha imagen, en la partición (clúster) de su representante más cercano en el espacio RGB. Estos histogramas son los que posteriormente se utilizarán como predicción para dichas imágenes.

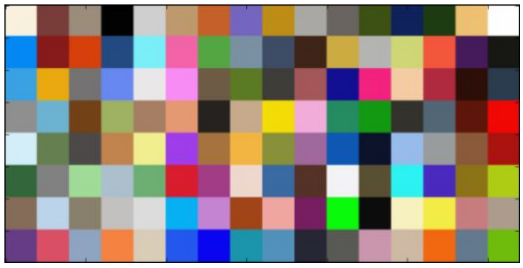

Fig. 3: Representación de los 128 centros en el espacio RGB.

Utilizamos el conjunto de imágenes de Google colors para el entrenamiento. Si asumimos que cada imagen consta de 90.000 píxeles de media (300 x 300), clasificar todo el conjunto de imágenes en 128 clústers podría ser muy costoso, más de 99 millones de píxeles por clasificar.

Limitándonos al hardware del que disponemos (CPU AMD A10-7700K, 3.4 Mhz, 16 Gb RAM), en una primera aproximación clasificamos una sola imagen con todos sus píxeles. El k-means tarda aproximadamente 700 segundos en completarse. Si procesáramos el conjunto de entrenamiento al completo, el algoritmo tardaría días en encontrar una solución. Para lidiar con este problema, se realiza un muestreo aleatorio de 500 píxeles por imagen. En la figura (3) se muestran los 128 centroides obtenidos con el algoritmo k-means. Se puede ver como cubren una gama de colores potencialmente representativa de todo el espacio de color RGB.

Con estos centroides como base, para cada imagen obtenemos un histograma representado en un vector de 128 posiciones, dónde cada una de estas posiciones representa la cantidad de píxeles de esa imagen que están más cercanos al centroide que ocupa esa posición en el vector. Normalizando cada uno de los histogramas, obtenemos un vector de probabilidades. La suma de todos los elementos de un mismo vector siempre será 1.

### 5.2. Implementación del modelo SS-CNN

Siguiendo el trabajo realizado por Wang *et al.* [10], nuestro modelo consta de 3 capas convolucionales y una última capa *fully connected* de 32, 96, 96 y 128 nodos para cada una de ellas respectivamente. Las neuronas de la capa *fully connected*, están completamente conectadas con cada una de las neuronas de la capa previa. Aparte de estas capas, existen un seguido de capas intermedias:

- *Rectifier Linear Units* (ReLU): Estas capas implementan la función de activación de las neuronas: f(x) =

max(0,x). El objetivo es incrementar las propiedades no lineales de la función de decisión.

- *Max pooling*: Estas capas aplican un proceso de muestreo sobre los datos que reciben (Figura 4). El objetivo es reducir la dimensionalidad de la representación de los datos de entrada aplicando un filtro de tamaño N x N.

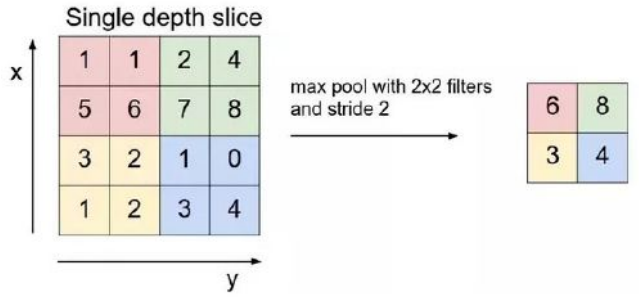


Fig. 4: Funcionamiento de una capa max pooling.

- *Local response normalization* (LRN) : Estas capas básicamente normalizan la salida de las capas ReLU, ya que las activaciones de estas suelen generar datos en un rango arbitrario.

Mientras que la primera y la segunda capa convolucionales son seguidas por capas ReLU, *max pooling* y LRN, la tercera capa convolucional no incluye una capa LRN. Los tamaños de los filtros convolucionales son de 3 x 3 y desplazamiento (*strides*) de 1, 1 y 2 respectivamente. Para todas las capas *max pooling*, se utilizan filtros de 3 x 3 y *strides* de 2. El tamaño de los recortes extraídos de las imágenes utilizadas para el entrenamiento son de 37 x 37 píxeles.

Para entrenar la red se utiliza el algoritmo de propagación hacia atrás de errores o retropropagación (*Backpropagation*), que se utiliza conjuntamente con un método de optimización como por ejemplo el descenso del gradiente en nuestro modelo (ver sección A.1). Definimos la *loss function* que calcula la entropía cruzada (*cross entropy*) para minimizar la divergencia entre el vector que utilizamos como supervisión y las predicciones hechas por la red neuronal:

$$Loss = -\frac{-1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c}\hat{p_{ij}}\log(p_{ij}) \qquad (1)$$

donde *n* es el tamaño del conjunto de imágenes (*batch*) utilizado cuando entrenamos el modelo SS-CNN, *c* el número de centros de clase (128 en nuestro modelo), $p_{ij}$ es la probabilidad de la imagen *i* de pertenecer a la clase *j* dada por la red y $\hat{p_{ij}}$, la probabilidad correspondiente en el vector de supervisión.

En el trabajo de Wang *et al.* [10], no se especifica la configuración de los parámetros utilizados en el entrenamiento de su modelo. Hemos hecho varias pruebas durante el entrenamiento y hemos decidido utilizar los valores siguientes:

- Iteraciones (*steps*) = 2000. Número de iteraciones durante el entrenamiento que se aplican al modelo.
- Conjunto de muestras en cada iteración (*batch size*) = 1100. Definimos el *batch size* con el mismo tamaño del grupo de recortes que usamos para entrenar el modelo. De esta manera el modelo será entrenado con todos los recortes de 37 x 37 disponibles en cada una de las iteraciones.
- Épocas (*epochs*) = 100. Asumimos que tenemos 20.000 ejemplos de entrenamiento. Por lo tanto, 20.000 / *batch size* = 19 iteraciones para completar un *epoch*. Como tenemos 2000 iteraciones, redondeando obtenemos que cada 100 iteraciones será aplicado un paso de *forward* y uno de *backward* en el algoritmo de *backpropagation* que reduce el error durante la fase de entrenamiento.
- Ratio de aprendizaje (*learning rate*) = 0.1. Se usa en el algoritmo del descenso del gradiente, y afecta en la velocidad a la que los pesos son ajustados durante el entrenamiento de la red.

## 5.3. Refinación del modelo SS-CNN (*finetuning*)

A partir del modelo inicial de SS-CNN (en adelante Net1), se aplica un proceso de refinamiento (*finetuning*). La última capa de la Net1 es reentrenada con las mismas imágenes de entrenamiento utilizadas previamente. Durante el *finetuning*, el modelo es entrenado con todas las capas fijadas y solo la última capa *fully connected* es reentrenada siguiendo las estructuras clásicas de los modelos CNN. Sin embargo, la salida de la última capa se modifica para adaptar el modelo a la clasificación de colores en 11 clases distintas. Se etiquetan los recortes con el color de sus imágenes de origen, para hacer la predicción sobre los 11 colores definidos por Berlin y Kay[4]. El nuevo modelo obtenido (en adelante Net2), se verá afectado claramente por las imágenes mal etiquetadas y estará muy lejos de ser preciso.

Las imágenes utilizadas para el entrenamiento del modelo están etiquetadas globalmente, pero en muchos casos, ese color solo aparece en una pequeña región de la imagen o corresponden a un objeto en concreto pero la imagen consta de más colores. Incluso algunas imágenes están incorrectamente etiquetadas por completo, pudiéndose dar el caso de que una imagen etiquetada con un color, no tenga ni un solo píxel de ese color. Por ello en los siguientes pasos se aplica un proceso de selección de muestras sobre las imágenes de entrada para desechar las muestras etiquetadas erróneamente y obtener así un conjunto de muestras "purificadas" con las que el modelo será reentrenado.

### 5.3.1. Purificación del conjunto de entrenamiento

Se implementa un método de selección de muestras (*sample selection*) capaz de filtrar las imágenes que se utilizan en el entrenamiento de la red basándose en la extracción de las características de estas imágenes desde la tercera capa de la red. Recordemos que en la Net2, solo la última capa ha sido entrenada y las capas previas contienen los pesos resultantes del entrenamiento del modelo Net1. Como este modelo es auto-supervisado y ha sido entrenado con los histogramas de la distribución del color de una imagen, la tercera capa de la red tiene un alto potencial para extraer las características del color de una imagen. Usamos dichas características para filtrar las futuras imágenes de entrenamiento.

El proceso del algoritmo es el siguiente:

1. Con el modelo Net2, hacemos la predicción de los recortes utilizados para su entrenamiento y para cada color, seleccionamos las 10 mejores muestras correctamente clasificadas (las imágenes predichas correctamente con el mínimo error o *score* más alto).

2. Definimos estas 10 imágenes como representantes del color al que pertenecen y se extraen las características desde la tercera capa de la red para todos los recortes. A continuación se calcula la media de aquellos seleccionados como representantes.

3. La media de estas características se toma como centros de clase.

4. Seleccionamos las muestras (recortes) cuyas características extraídas de la tercera capa de la red, están a una distancia euclídea menor a un umbral respecto al centro de su clase, y desechamos las muestras que quedan demasiado alejadas.

5. Tras obtener estas nuevas muestras, procedemos a reentrenar de nuevo el modelo Net2. Este proceso de selección de muestras y entrenamiento se aplica de forma repetida durante N veces para mejorar los resultados de nuestro modelo.

Encontramos un problema con el método *sample selection*. Si partimos de un color en el cual existen más muestras mal etiquetadas que correctas, cuando seleccionamos el centro de referencia para ese color basándonos en la predicción de la red, el algoritmo filtrará las imágenes con características erróneas para dicho color y será incapaz de representar correctamente el centro de clase en su siguiente iteración. Por más iteraciones que apliquemos de *sample selection* y entrenamiento, el resultado nunca converge hacia la solución esperada.

Para solucionar este problema, proponemos un método alternativo. Seleccionamos manualmente 10 imágenes del conjunto de entrenamiento para cada color, dónde realmente estas imágenes estén bien etiquetadas y hacemos la media de sus características extraídas de la tercera capa de la red. De este modo, podemos comparar los valores obtenidos, con los valores de los centros de clase adquiridos mediante el método *sample selection*. Estos valores son similares en ambos casos excepto con el color negro. Esto se debe a que el conjunto de este color es el peor etiquetado. Para corregir esto, en la primera iteración del *sample selection*, substituimos el valor del centro de clase del color negro con el obtenido manualmente, de manera que en las siguientes iteraciones, las imágenes obtenidas, serán más cercanas al color pertinente.

# 6 Resultados

Para el entrenamiento y el proceso de *finetuning*, se ha utilizado el conjunto de imágenes de Google colors y para verificar la fiabilidad del modelo, el conjunto de Ebay colors. Dentro del *dataset* de Ebay colors, seleccionamos únicamente las 2 imágenes dedicadas a la parte de validación (ver subsección 4.3), en total 88 imágenes. Para valorar el rendimiento de nuestro modelo en la clasificación de colores, se llevan a cabo varios experimentos, tanto de predicción del color de un objeto, como a nivel de píxel.

## 6.1. Predicción a nivel de objeto

Para evaluar únicamente el color de los objetos y no tener en cuenta el resto de la imagen, aplicamos las máscaras incluidas en el *dataset* de Ebay colors, eliminando así toda la región de la imagen que no forma parte del objeto que queremos clasificar. De cada imagen resultante extraemos 100 recortes de tamaño aleatorio y los redimensionamos al tamaño en el cual ha sido entrenada nuestra red (37 x 37). La predicción del color del objeto, se obtiene mediante la media de las predicciones de los 100 recortes extraídos de dicho objeto. En la tabla 1 se muestra la *accuracy* de nuestro modelo por categorías de objetos y su respectiva media.

Los resultado de la Net2-NSS (no *sample selection*), pertenecen a la predicción hecha por el modelo entrenado sin aplicar el *sample selection*. Este modelo ha sido reentrenado una única vez con las mismas imágenes que la Net1. Por esta razón este modelo obtiene los peores resultados. Habiendo sido entrenado en su gran mayoría con imágenes mal etiquetadas su capacidad de clasificación es bastante baja.

La Net2–SS (*sample selection*), es el modelo obtenido a base de aplicar el proceso de *sample selection* 10 veces y reentrenando el modelo después de cada iteración con las nuevas muestras obtenidas. En este caso el modelo mejora algo más respecto al anterior debido precisamente a la "purificación" de estas muestras que se utilizan para entrenar la red.

Aun así, los resultados siguen siendo bajos, sobretodo en la categoría de coches y alfarería. Debido a la superficie de algunos de estos objetos, las imágenes se ven condicionadas por la iluminación en el momento en el que han sido tomadas. En nuestro caso esto ocurre principalmente con los colores acromáticos (gris, blanco y negro). En el caso del color blanco, también es fácil confundirlo con el gris en cualquier región ensombrecida del objeto.

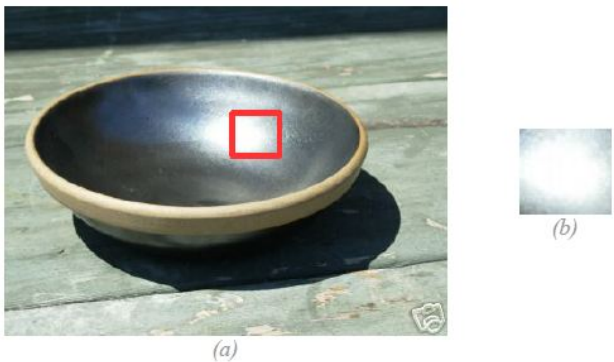


Fig. 5: (a): Imagen de la categoría alfarería, color negro. (b): Recorte extraído de la región marcada en rojo de la imagen (a).

En la figura (5), podemos ver un ejemplo de como afecta el reflejo de la luz en imágenes reales. En esta imagen en concreto, se debe a la reflexión especular de la luz al incidir en un determinado punto de la imagen.

Proponemos una variación en el modelo para mejorar los resultados de la clasificación. Igual que en el trabajo de van

| | COCHES | VESTIDOS | ALFARERÍA | CALZADO | MEDIA |
|---|---|---|---|---|---|
| Net2-NSS | 42.31 | 60.63 | 49.52 | 50.39 | 50.67 |
| Net2-SS | 53.76 | 70.99 | 57.05 | 69.91 | 62.87 |
| Net2-Ebay | 79.02 | 86.42 | 69.52 | 78.82 | 78.42 |
| CNN-Wang [10] | 73.18 | 91.82 | 83.36 | 91.18 | 84.89 |

Tabla 1: RESULTADOS EN PORCENTAJE DE LAS IMÁGENES BIEN CLASIFICADAS EN CADA MODELO.

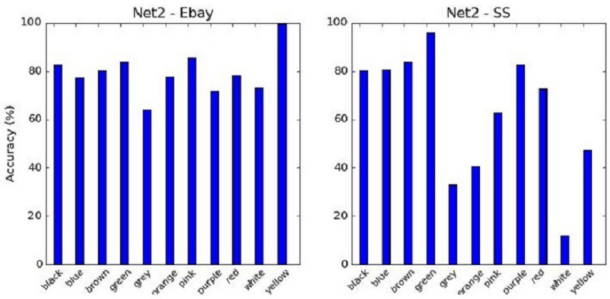


Fig. 6: Resultados de la clasificación por colores.

de Weijer *et al.* [8], decidimos utilizar imágenes del conjunto de Ebay colors para el aprendizaje. Utilizamos las 10 imágenes dedicadas a a la fase de *testing* para el aprendizaje de este nuevo modelo. Obtenemos de este modo el modelo Net2–Ebay. La diferencia respecto al modelo Net2-SS, es que en este caso durante la fase de *finetuning*, hemos utilizado las imágenes de Ebay en vez de las de Google. Tampoco hemos aplicado *sample selection* ya que consideramos que las muestras están lo suficientemente bien etiquetadas.

En la figura (6) se muestra la media de las muestras bien clasificadas (*accuracy*), obtenida por colores en los dos modelos: Net2-SS y Net2-Ebay. Estos resultados incluyen las 4 categorías de objetos distintos. Podemos ver una gran mejora en el modelo Net2–Ebay, sobretodo en los colores gris y blanco. Estos colores pertenecen al espectro acromático y son fáciles de confundir según su luminosidad. Aunque en algunos colores como el lila o el verde se reduce la *accuracy*, otros colores como el amarillo, el rosa o el naranja suben notablemente en los resultados. El nuevo modelo Net2-Ebay, obtiene mejores resultados y es capaz de clasificar cada uno de los 11 colores de una forma más regular.

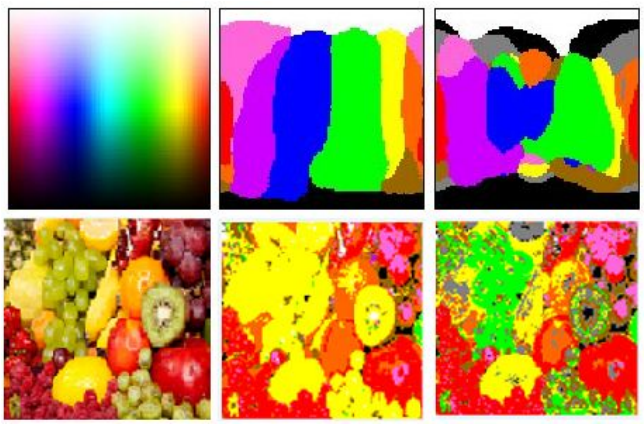

Fig. 7: A la izquierda la imagen original, en el centro la imagen resultado de la clasificación con el modelo Net2-Ebay y a la derecha por el modelo Net2-SS.

En la figura (7), se observa como la Net2-Ebay está por encima de la Net2-SS. Aparte de predecir los colores correctos de una forma más precisa, mejora el problema que teníamos debido a reflejos, degradados en los colores de la imagen, etc. En la figura (8) otro ejemplo comparativo entre los dos modelos, esta vez, con las imágenes de Ebay colors.

En la tabla 1, se incluyen los resultados obtenidos por el modelo de Wang *et al.* [10], en el cual esta basada la arquitectura de nuestro modelo. Este modelo ha sido entrenado únicamente con imágenes de Google colors al igual que nuestro modelo Net2-SS. Aunque los modelos sean similares y las imágenes utilizadas en el proceso de aprendizaje sean las mismas, el modelo de Wang, registra aproximadamente un 20 % más de *accuracy* que nuestro modelo. Esto puede deberse a varios factores:

- El submuestreo de píxeles que hemos aplicado en el algoritmo k-means. Al reducir el número de píxeles a clasificar por el algoritmo también reducimos la exactitud de los 128 clústers que obtenemos como resultado.
- El tiempo y los recursos de cómputo dedicados durante la fase de entrenamiento de la red. Estos modelos se suelen entrenar en más de una GPU, durante varios días e incluso semanas.
- Parámetros de la red. Diferentes valores asignados en parámetros como el *learning rate* o las *epochs* entre otros, producen diferentes resultados.

Además de estos motivos, pueden existir más detalles en la implementación que influyen en el resultado final del modelo obtenido.

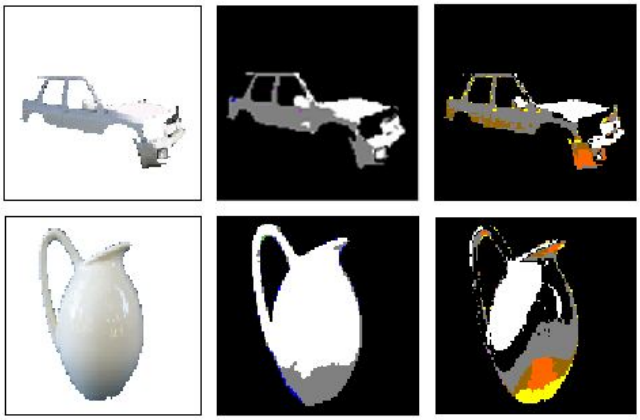

Fig. 8: A la izquierda la imagen original, una de la categoría coches y otra de alfarería, ambas de color blanco. En el centro, la imagen resultante de la clasificación con el modelo Net2-Ebay y a la derecha por el modelo Net2-SS.

## 6.2. Predicción a nivel de píxel

En este experimento se quiere comprobar el rendimiento de nuestro modelo clasificando los píxeles de una imagen determinada. Para estas pruebas utilizamos los modelos Net2–SS y Net2–Ebay. Comparamos nuestros resultados con los obtenidos con el método PLSA y PLSA-bg en el trabajo de van de Weijer *et al.* [8], y con el modelo de Wang *et al.* [10].

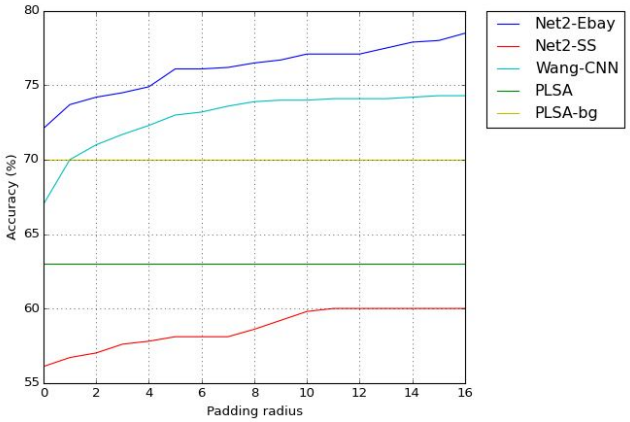


Fig. 9: Resultados de la clasificación a nivel de píxel, según el *padding radius* aplicado.

Para clasificar un único píxel, simplemente lo replicamos en una imagen de 37 x 37. También se ha llevado a cabo la predicción del color de un píxel teniendo en cuenta sus vecinos, ya que muchas veces los píxeles adyacentes de imágenes reales tienden a tener un color similar al píxel central y pueden ayudar a mejorar el rendimiento del modelo. El radio de selección de píxeles vecinos (*padding radius*), se establece entre 0 y 16, dónde 0 indica hacer la predicción para un único píxel. Por ejemplo, para *padding radius* = 1, escogeremos un píxel y todos sus vecinos a distancia 1, de manera que se obtiene un recorte de 3 x 3 el cual se acaba redimensionando a 37 x 37.

En la figura (9) se muestran los resultados de la clasificación a nivel de píxel según el *padding radius* utilizado. En ambos modelos de PLSA no se han tenido en cuenta los píxeles vecinos, únicamente el píxel central. En el caso de los modelos solo entrenados con imágenes de Google (PLSA y Net2–SS), el primero obtiene mejores resultados. Sin embargo, para los modelos Net2–Ebay y PLSA–bg, ambos entrenados con imágenes de Google y también con imágenes de Ebay, nuestro modelo registra mejores resultados aun sin tener en cuenta los píxeles vecinos. En cuánto al modelo de Wang, los resultados siguen el mismo patrón que los explicados en la sección 6.1.

En los dos modelos de Net2, se observa una mejoría en los resultados al aumentar el *padding radius*. Mientras que Net2-SS se estabiliza con *padding radius* igual a 10, la Net2–Ebay, sigue creciendo hasta el final. Los píxeles cercanos incrementan la robustez de nuestro modelo a la hora de predecir el color de un píxel. Nuestro modelo Net2-Ebay registra los mejores resultados en la clasificación de un píxel en concreto o de pequeñas regiones de una imagen.

Para demostrar la eficiencia del modelo Net2-Ebay en la clasificación del color a nivel de píxel, clasificamos un pequeño conjunto de las imágenes del *dataset* de Ebay colors. Predecimos el color de cada píxel y recreamos cada uno de los píxeles del objeto con el color etiquetado por nuestro modelo. En la figura (10) se muestran los resultados. Debido a que los objetos han sido extraídos de imágenes reales, estos se ven condicionados por factores que ya hemos explicado más arriba (ver sección 6.1), sin embargo, la predicción a nivel de píxel tiene un rendimiento aceptable, pudiendo representar estas variaciones del color de un objeto con un error muy bajo.

## 7 Conclusiones

En este trabajo se ha abordado el problema del etiquetado de colores (*color naming*), mediante técnicas de *deep learning*. Hemos implementado y entrenado nuestro propio modelo de CNN, basándonos en la arquitectura propuesta en el trabajo de Wang *et al.* [10], aportando una solución competente al problema del etiquetado de colores.

Hemos implementado el algoritmo k-means, para generar 128 clases distintas dentro del espacio de color RGB clasificando un porcentaje de todos los píxeles extraídos de un conjunto de imágenes. Con estas 128 clases, hemos generado los histogramas de la distribución del color de las imágenes. Hemos implementado un modelo de red neuronal auto-supervisada, que aprende a clasificar imágenes en estas 128 clases. Hemos creado otro nuevo modelo de red neuronal a partir de la red anterior ya entrenada, cambiando la última capa de la red para substituir la clasificación de 128 clases a los 11 colores descritos por Berlin y Kay [4]. Durante la implementación de la red, se han tenido que definir y probar parámetros relacionados con el aprendizaje del modelo. Hemos implementado un método llamado *sample selection*, capaz de extraer características del color de las imágenes desde la tercera capa de la red, y mediante estas características filtrar las imágenes que se utilizan en el entrenamiento del nuevo modelo, evitando así, muestras mal etiquetadas y por consiguiente aumentando la capacidad de predicción de nuestro modelo. Finalmente, hemos propuesto una mejora consistente dando buenos resultados y demostrando como esta nueva mejora ayuda a incrementar la fiabilidad de la CNN que se propone en el trabajo de Wang *et al.* [10]. Con todos los modelos implementados hemos llevado a cabo una serie de experimentos pudiendo comparar la capacidad de clasificación de nuestros modelos en varias situaciones y la diferencia que ofrecen respecto a otros modelos ya existentes.

Como resultado de la solución llevada a cabo, se ha obtenido una red neuronal capaz de clasificar los colores de una imágen, pequeñas regiones de la misma o los píxeles de estas, con un rendimiento considerable. Este modelo puede ser utilizado en tareas de detección de colores o como soporte para otros sistemas en los cuáles se deba obtener la información de los colores de una imagen. Por lo tanto, se considera que se han conseguido los objetivos previstos para este trabajo en el tiempo planificado.

Como mejoras a corto plazo, se podría ejecutar el algoritmo k-means en un computador más potente, incluso en un clúster, para realizar la clasificación en 128 clases, con más píxeles de los actuales. Esto podría mejorar en la especificidad de los centros generados por el algoritmo. También se podría realizar el entrenamiento de la CNN sobre una o varias GPUs, aumentando la velocidad en el aprendizaje y

Fig. 10: Algunos ejemplos de objetos etiquetados con el modelo Net2-Ebay. A la izquierda de cada columna la imagen original, a la derecha la representación de los píxeles etiquetados con los 11 colores básicos. Los colores por filas son los siguientes: negro, azul, marrón, verde, gris, naranja, rosa, lila, rojo, blanco y amarillo. Por columnas, las categorías de los objetos: coches, vestidos, alfarería y calzado.

pudiendo aumentar la duración de la fase de entrenamiento de la red. Otra mejora, sería incluir variaciones en los datos utilizados, por ejemplo, aumentar el número de muestras o utilizar otros *datasets*, tanto en la fase de aprendizaje como en la validación.

En cuánto a vías de continuación, la CNN resultante de nuestra solución propuesta, puede asistir a otros sistemas automáticos. Por ejemplo, en sistemas de detección de objetos, segmentación de regiones de una imagen por colores o sistemas que necesitan conocer el color de los píxeles de una imagen para su posterior tratamiento. Tensorflow aporta la posibilidad de ejecutar sus modelos en un dispositivo móvil, así que se podría crear una aplicación para smartphones y tablets. A través de la cámara, se podrían tomar las imágenes y nuestro modelo que previamente ha sido entrenado, se encargaría de etiquetar los colores de la imagen tomada. Del mismo modo, se podría crear una aplicación de escritorio, para que un usuario cualquiera pueda introducir cualquier imagen y obtener el etiquetado de los colores que la conforman.

## AGRADECIMIENTOS

En primer lugar, quiero dar las gracias a mi tutor de este TFG, Robert Benavente Vidal. Gracias por su implicación conmigo en el proyecto, por sus consejos y por su guía en todo momento. También quiero dar las gracias al equipo docente de la facultad de ingeniería de la UAB. Durante estos años he tenido la oportunidad de conocer a grandes profesionales y grandes personas encargadas de formar a las generaciones futuras. Y por último, pero no por eso menos importante, quiero dar las gracias a mi familia en general, sobretodo a mi abuelo, la persona que más apoyo me ha dado desde siempre y que en todo momento me ha animado y empujado a hacer lo que yo quería.

## REFERENCIAS

[1] E. B. Zuleta, "La percepción" in *El sistema nervioso: desde las neuronas hasta el cerebro humano*, 1st ed. Medellín, Colombia: Universidad de Antioquia, Ed. 2007, ch. 17, pp. 237 – 259.

[2] R. Nieuwenhuys, *El sistema nervioso central humano*, Vol 2, 4t ed. Buenos Aires, Madrid: Panamericana, D. L., Ed. 2009, pp. 751 – 784.

[3] V. Correa, L. Estupiñán, Z. Garcia, O. Jiménez, L. F. Prada, A. Rojas, S. Rojas and E. Cristancho, "Percepción visual del rango de color: diferencias entre género y edad", Universidad Militar de Nueva Granada, Revista Med, 2007, Vol 15, No 1.

[4] B. Berlin and P. Kay, "Basic color terms: their universality and evolution", Berkeley: University of California, 1969.

[5] G. Menegaz, A. Le Troter, J. Sequeira and J. M. Boi, "A discrete model for color naming" in *EURASIP Journal on Applied Signal Processing*, Hindawi Publishing Corp. New York, NY, United States, Ed. 2007, pp. 113-113

[6] R. Benavente, M. Vanrell and R. Baldrich, "A data set for fuzzy colour naming" in *Color Research and Application*, Vol 31, No 1, Wiley Subscription Services, Inc., A Wiley Company, Ed. 2006, pp. 48-56

[7] R. Benavente, M. Vanrell and R. Baldrich, "Parametric fuzzy sets for automatic color naming" in *J. Opt. Soc. Am. A*, Vol 25, No 10, OSA, Ed. 2008, pp. 2582-2593

[8] J. van De Weijer, C. Schmid, J. Verbeek, and D. Larlus, "Learning color names for real world applications", IEEE Trans. Image Process, pp. 1512-1523. 2009, Vol 18.

[9] T. Hofmann, "Probabilistic latent semantic indexing" in Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '99, New Work, USA, pp. 50-57, 1999.

[10] Y. Wang, J. Liu, J. Wang, Y. Li, H. Lu, "Color names learning using convolutional neural networks", The National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China, 2015

[11] I. Rafegas and M. Vanrell, "Color spaces emerging from deep convolutional networks" in *Color and Imaging Conference*, No 1, 2016, pp. 225-230

[12] Z. Cheng, X. Li and C. C. Loy, "Pedestrian Color Naming via Convolutional Neural Network" in *Computer Vision – ACCV 2016. Lecture Notes in Computer Science*, 2017

[13] R. Q. Juan A. and C. M. Mario, "Redes neuronales artificiales para el procesamiento de imágenes, una revisión de la última década" in Revista de ingeniería eléctrica, electrónica y computación, 2011, Vol 9, No 1.

[14] Caffe: Deep learning Framework. [Fecha de consulta: 03 Marzo 2017] Disponible en: http://caffe.berkeleyvision.org/

[15] Torch: A scientific computing framework for LuaJIT. [Fecha de consulta: 11 Febrero 2017] Disponible en: http://torch.ch/

[16] The LuaJIT project. [Fecha de consulta: 01 Abril 2017] Disponible en: http://luajit.org/

[17] An open-source software library for Machine Intelligence. [Fecha de consulta: 11 Febrero 2017] Disponible en: https://www.tensorflow.org/

[18] Google Brain Team. [Fecha de consulta: 11 Febrero 2017] Disponible en: https://research.google.com/teams/brain/

[19] MatConvNet: CNNs for MATLAB. [Fecha de consulta: 11 Febrero 2017] Disponible en: http://www.vlfeat.org/matconvnet/

[20] Keras: Deep learning library for Theano and Tensorflow. [Fecha de consulta: 13 Junio 2017] Disponible en: https://keras.io/

[21] Theano: Python library for neural networks. [Fecha de consulta: 26 Junio 2017] Disponible en: http://deeplearning.net/software/theano/

[22] Image Data Sets. [Fecha de consulta: 13 Febrero 2017] Disponible en: http://lear.inrialpes.fr/people/vandeweijer/data.html

[23] J. van de Weijer and C. Schmid, "Applying color names to image description", LEAR TEAM, Inria Rhone Alps, Monbonnet, France. Biblioteca Inria-Alpes, 2007.

# APÉNDICE

## A.1. Backpropagation

El algoritmo de *backpropagation*, repite un ciclo de dos fases: la propagación del error y la actualización de los pesos. Una vez la red ha recibido un estímulo como entrada, se propaga por toda la red, capa a capa hasta llegar a la capa de salida, dónde mediante una función (*loss function*) se compara la salida de la red con la salida deseada y se calcula el error para cada una de las neuronas de la capa de salida. Estos valores de error son propagados hacia atrás empezando desde la capa de salida y a cada neurona de la red se le asocia uno de estos valores que representan la contribución de dicha neurona a la salida de la red. El algoritmo utiliza estos valores de error para calcular el gradiente de la *loss function* y el gradiente, es el encargado de actualizar los pesos de la red con la intención de minimizar dicha función.

## A.2. K-means

El algoritmo k-means es un método de agrupamiento, que tiene como objetivo la partición de un conjunto de *n* observaciones en *k* grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. Dado un conjunto inicial de K centroides (grupos) $m_1, ..., m_k$ y después de una inicialización de los mismos, el algoritmo alterna entre dos pasos:

- Asignación: Asigna cada muestra al grupo con la media más cercana.

$$S_i^{(t)} = \left\{ x_p : \|x_p - m_i^{(t)}\| \leq \|x_p - m_j^{(t)}\| \forall 1 \leq j \leq k \right\} \tag{2}$$

  Donde cada $x_p$ va exactamente dentro de un $S_i^{(t)}$, incluso aunque pudiera ir en dos de ellos.

- Actualización: Calcula los nuevos centroides y se asigna como el centroide de las muestras en el grupo.

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{3}$$

El algoritmo se considera que ha convergido cuando las asignaciones ya no cambian.

## A.3. Datos de interés sobre Tensorflow

A continuación, algunos datos de interés extraídos del repositorio donde se encuentra el proyecto de este framework y que muestran la importancia y el potencial que está adquiriendo en la comunidad.

- Más de 15.000 *commits* desde Noviembre de 2015.
- Más de 700 contribuidores.
- Más de 1 millón de descargas de sus archivos.
- Más de 20.000 *forks* hechos a sus repositorios.
- Siempre está entre los 15 repositorios más populares en github, entre todas las categorías.
- Hoy en día se utiliza en las aulas de muchas universidades como en Toronto, Berkeley y Stanford.

## A.4. Planificación del proyecto

A continuación se muestra el diagrama de Gantt de la planificación del proyecto.