UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

RECONOCIMIENTO MULTIMODAL DE EMOCIONES MEDIANTE EL USO DE REDES NEURONALES ARTIFICIALES

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL, RECONOCIMIENTO DE FORMAS E IMAGEN DIGITAL

Presentado por:
José Manuel Fuentes López

Dirigido por:
Vicente Juan Botti Navarro

Valencia, julio de 2019

Curso 2018/2019

Agradecimientos

A mi pareja por apoyarme y animarme cada día, especialmente en los momentos difíciles y a mis amigos y familia por alentarme a seguir formándome y trabajando duro.

A mi tutor Vicente Juan Botti Navarro por su guía y ayuda y a Joaquín Taverner Aparicio y Jaime Rincón Arango por su asesoramiento gracias a los cuales he terminado un trabajo del que sentirme orgulloso.

Resum

En aquest treball es desenvolupa un reconeixedor multimodal d'emocions humanes mitjançant l'ús de xarxes neuronals artificials. Per a això, es disenyen i entrenen tres models capaços de reconèixer emocions a partir d'imatges de la cara, audios i text, per separat. Aleshores, es combinen aquestos sistemes per a crear uno multimodal, més robust i precís que les seues parts. Tot i això, hi ha que tenir en compte que l'expressió d'emocions es, en certa manera, dependent de la cultura i l'idioma, per la qual cosa es proposa especialitzar el nostre sistema en l'idioma espanyol. A més, per a completar el projecte, es desenvolupa una aplicación web en *Django* capaç d'incorporar aquest sistema multimodal. Aquesta aplicació pot detectar emocions en directe a partir de les imatges obtingudes de la *webcam* i del audio del micròfon. Així mateix, l'aplicació permiteix gravar i descarregar una sessió anotada de detecció en directe. Finalment, cal destacar que la utilitat d'aquest projecte es molt variada, desde sessions de terapia psicológica, fins a recomendacions musicals personalitzades, passant per la educació o el màrqueting.

Paraules clau: emocions, xarxes neuronals, multimodal, web

Resumen

En este trabajo se desarrolla un reconocedor multimodal de emociones humanas mediante el uso de redes neuronales artificiales. Para ello, se diseñan y entrenan tres modelos capaces de reconocer emociones a partir de imágenes de la cara, audios y texto, por separado. Entonces, se combinan dichos sistemas para crear uno multimodal, más robusto y preciso que sus partes. Sin embargo, hay que tener en cuenta que la expresión de emociones es, en cierto grado, dependiente de la cultura y el idioma, por lo que se propone especializar nuestro sistema en el idioma español. Además, para completar este proyecto, se desarrolla una aplicación web en *Django* capaz de incorporar dicho sistema multimodal. Está aplicación es capaz de detectar emociones en directo a partir de las imágenes obtenidas de la *webcam* y del audio del micrófono. Asimismo, la aplicación permite grabar y descargar una sesión anotada de detección en directo. Por último, cabe destacar que la utilidad de este proyecto es muy variada, desde sesiones de terapia psicológica, hasta recomendaciones musicales personalizadas, pasando por la educación o el márquetin.

Palabras clave: emociones, redes neuronales, multimodal, web

Abstract

In this project we develop a multimodal recognizer of human emotions using artificial neural networks. In order to accomplish it, we design and train three models, which are able to recognize emotions from facial images, audios and text, separately. Then we combine those systems to create one which is multimodal and more robust and precise than its parts. Despite this, we have to take into account that the expression of emotions is, to some extent, dependent on culture and language, and that is why we propose to specialize our system in the Spanish language. Moreover, to complete this Project, a web application in Django is developed, which is able to incorporate this multimodal system. This app can detect emotions either live from the

images obtained from the webcam and the audio from the microphone, or from a video uploaded by the user, which is analyzed and returned to the user annotated via download. Likewise, the app allows the user to record and download an annotated session of live detection. Lastly, it should be noted that the applicability of this project is very diverse, from sessions of phycological therapy, to personalized musical recommendations, through education or marketing.

Key words: emotions, neural networks, multimodal, web

Índice

Índice de figuras

Índice de gráficas

Índice de tablas

Índice de ecuaciones

CAPÍTULO 1

Introducción

1.1 Motivación

Cuando hablamos de informática siempre pensamos en procesos meramente racionales. Sin embargo, el ser humano no suele actuar de esta forma. Las decisiones que tomamos los seres humanos están, generalmente, influenciadas por nuestro estado emocional y afectivo [1]. Es por ello que a la hora de crear agentes *software* que deban interactuar con personas de forma humana, se hace imprescindible que sean capaces de interpretar y utilizar las emociones.

De hecho, diversas investigaciones que analizan el potencial de las emociones en la interacción entre el usuario y el ente artificial, demuestran que, en presencia de un agente realista, las personas son más educadas, tienden a tomar decisiones socialmente deseables, están menos nerviosas [2] y muestran una mayor confianza en las recomendaciones que les proporciona el agente [3] [4].

Existen varios aspectos a considerar en la simulación artificial de interacciones sociales que consideran el papel de las emociones, siendo algunos ellos, la expresión física de emociones, la detección de emociones o estados emocionales y la simulación del comportamiento afectivo. La computación emocional es el área de investigación que se encarga de estudiar estos fenómenos, la cual incluye, entre otros, el reconocimiento de emociones en textos, audio e imágenes, la expresión de emociones a través de la expresión facial o gestual, la creación de modelos que simulan el surgimiento de las emociones o el efecto de las emociones en la toma de decisiones, entre otras [5].

En este trabajo concretamente nos centramos en lo que consideramos el primer paso a la hora de crear un agente que sea capaz de interactuar emocionalmente con un humano. Esto es, crear un reconocedor de emociones humanas en tiempo real lo más robusto y preciso posible. En este sentido, ya que los humanos expresamos emociones a través de más de un canal simultáneamente, lo ideal sería considerar más de una entrada para hacer una detección lo más fiable posible. Es por eso que diseñamos un reconocedor que utiliza las entradas más obvias, desde nuestro punto de vista, en la expresión humana de emociones, esto es, sus expresiones faciales (imagen), qué está diciendo (texto) y cómo lo está diciendo (audio).

Independientemente de que en proyectos futuros se pueda integrar en un sistema más complejo, como por ejemplo en un agente capaz de reconocer y expresar emociones para interactuar con usuarios, el reconocedor multimodal que en este trabajo se desarrolla ya tiene una gran aplicabilidad práctica, por ejemplo para niños con trastorno del espectro autista, que a veces tienen dificultades para expresar y detectar emociones; para análisis de estrategias pedagógicas, analizando las emociones de los estudiantes a lo largo de una clase; en terapias psicológicas, donde se puede registrar el progreso emocional del paciente a lo largo de la sesión; e incluso en ámbitos más lúdicos, como en reproductores automáticos de música, que podrían sugerir canciones tomando en cuenta, además de los gustos generales del usuario, su estado emocional actual.

1.2 Objetivos

Este proyecto se ha desarrollado siguiendo cuatro objetivos generales:

1. Desarrollar una red neuronal capaz de reconocer siete emociones básicas (enfado, asco, miedo, sorpresa, tristeza, felicidad y neutro) a partir de imágenes de expresiones faciales. Este objetivo puede descomponerse en tres subobjetivos:
    a. Buscar y seleccionar el/los corpus que se van a utilizar para entrenar el modelo. Estos datos deben ser suficientemente representativos para que la red pueda aprender invarianza al género, la edad, el origen y la iluminación.
    b. Buscar y seleccionar estudios similares que nos ofrezcan un punto de partida en cuanto a la arquitectura de la red neuronal.
    c. Afinar los parámetros de la red para obtener los mejores resultados posibles, hasta alcanzar un porcentaje de acierto, por lo menos, del 80%.
2. Desarrollar una red neuronal capaz de reconocer siete emociones básicas (enfado, asco, miedo, sorpresa, tristeza, felicidad y neutro) a partir del audio correspondiente a una persona hablando. Este objetivo puede descomponerse en tres subobjetivos:
    a. Buscar y seleccionar el/los corpus que se van a utilizar para entrenar el modelo. Estos datos deben corresponderse con hablantes del idioma español.
    b. Buscar y seleccionar estudios similares que nos ofrezcan un punto de partida en cuanto a la arquitectura y el tipo de la red neuronal.
    c. Afinar los parámetros de la red para obtener los mejores resultados posibles, hasta alcanzar un porcentaje de acierto, por lo menos, del 90%.
3. Desarrollar una red neuronal capaz de reconocer seis niveles de polaridad (positiva, muy positiva, negativa, muy negativa, neutra y ausencia) a partir de texto. Este objetivo puede descomponerse en tres subobjetivos:
    a. Buscar y seleccionar el/los corpus que se van a utilizar para entrenar el modelo. Estos datos deben corresponderse a textos escritos en español.
    b. Buscar y seleccionar estudios similares que nos ofrezcan un punto de partida en cuanto a la arquitectura y el tipo de la red neuronal.
    c. Afinar los parámetros de la red para obtener los mejores resultados posibles, hasta alcanzar un porcentaje de acierto, por lo menos, del 65%.
4. Diseñar una forma de combinar los tres modelos anteriores para crear un modelo multimodal más robusto y preciso que sus partes, que además pueda paliar las debilidades de cada una de las tres redes en un entorno realista.
5. Implementar una aplicación web que utilice el sistema de reconocimiento multimodal creado. Además, debe ser capaz de utilizarlo para detectar emociones en directo. También tiene que permitir al usuario grabar la sesión anotada y descargar el vídeo resultante.

1.3 Tecnología utilizada

1.3.1 Software

Para el entrenamiento de las redes neuronales se ha utilizado la librería *Keras* sobre *Tensorflow*. *Keras* es una librería de alto nivel escrita en *Python* para el desarrollo de redes neuronales cuyo objetivo es facilitar al máximo la especificación y uso de este tipo de sistemas. Soporta combinaciones de todo tipo de redes, incluyendo convolucionales y recurrentes. Y permite fácilmente lanzar ejecuciones tanto en CPU como en GPU.

Para el desarrollo de la aplicación web se ha utilizado *Django*. *Django* es un *framework* de desarrollo web de código abierto fundamentado en el patrón *model-template-view*. Está escrito en *Python* y permite crear aplicaciones web de forma rápida y sencilla.

Aunque *Django* esté escrito en *Python*, cualquier código que deba ejecutarse en el lado del cliente debe estar escrito, como es habitual, en *JavaScript*, ya que los navegadores no disponen de un intérprete de *Python*. Si bien el análisis de audio y texto se realiza con menor frecuencia, el reconocimiento de las expresiones faciales pretendemos hacerlo varias veces por segundo, por lo que es obligatorio que el modelo correspondiente se ejecute en el lado del cliente, ya que es impensable realizar varias llamadas por segundo al servidor. Por ello, no podemos utilizar *Keras* para para hacer predicciones con modelo. Todo esto hace necesario el uso de la librería *TensorFlow.js*, que permite lanzar modelos de *deep learning* en *JavaScript*, tanto en el navegador, como con *Node.js*. Además, proporciona herramientas para convertir modelos creados en *Keras* al formato de *TensorFlow.js*.

1.3.2 Hardware

Para el desarrollo de este proyecto se ha utilizado un ordenador portátil con un procesador Intel Core i7-7700HQ a 2.80 GHz y una memoria RAM de 16 GB.

Además, para el entrenamiento de las redes neuronales se han utilizado tres tarjetas gráficas: una NVIDIA GeForce GTX 1050 Ti, una NVIDIA GeForce GTX 1070 y una NVIDIA GeForce GTX Titan XP.

1.4 Estructura

La memoria se estructura en torno a 8 bloques principales:

1. Introducción: en el actual apartado se ha presentado un primer acercamiento al proyecto, exponiendo la motivación del mismo y planteando los objetivos principales. Se han listado los más importantes elementos *software* utilizados y el *hardware* sobre el que se ha realizado el proyecto, se ha expuesto brevemente la estructura de esta memoria y se han enumerado las convenciones que se han seguido a lo largo de la redacción de esta memoria. Además, se han explicado algunos conceptos básicos sobre redes neuronales artificiales, que se utilizan a lo largo del trabajo.
2. Estado del arte: en el segundo capítulo se analiza el estado actual de este tipo de tecnología, revisando artículos que se hayan enfrentado a tareas similares. Revisamos artículos que enfrenten el problema de detección de emociones tanto de forma unimodal, ya sea a partir de expresiones faciales, audio o texto, como de forma multimodal, combinando las mismas características que nosotros consideramos u otras. Se destacan fortalezas y debilidades de estos estudios y se discuten las aportaciones originales que conlleva nuestra propuesta.
3. Desarrollo del reconocedor multimodal: en el tercer apartado se desarrolla el reconocedor en cuestión. Este apartado se divide en cuatro subsecciones:
    a. En la primera de ellas se diseña y entrena la red encargada de reconocer emociones a partir de expresiones faciales. A esta tarea nos referimos comúnmente como el problema facial.
    b. En la segunda se diseña y entrena la red encargada de reconocer emociones a partir de audio. A esta tarea nos referimos comúnmente como el problema acústico.

    c. En la primera de ellas se diseña y entrena la red encargada de reconocer polaridad a partir de texto. A esta tarea nos referimos comúnmente como el problema textual.
    d. En la última nos encargamos de combinar esos tres modelos, formando así nuestro reconocedor multimodal.
4. Desarrollo de la aplicación: en la cuarta parte de este documento se explica cómo se ha desarrollado la aplicación web que utiliza el sistema multimodal y qué funcionalidades tiene.
5. Conclusiones: en el último capítulo se cierra el proyecto analizando el nivel de completitud alcanzado respecto a los objetivos planteados y planteando las posibles ampliaciones que podrían hacerse en el futuro sobre el producto final obtenido.

1.4 Convenciones

A lo largo de la memoria seguimos algunos criterios de marcado que conviene explicar a priori.

1. Las palabras extranjeras se remarcan en cursiva.
2. Se entrecomillan las citas textuales externas a la obra.
3. Los nombres de corpus, lenguajes de programación, métodos, módulos, librerías o *frameworks* también se escriben en cursiva.

Además, a lo largo de la memoria se muestran en figuras representaciones de las redes neuronales que vamos desarrollando. *Keras* dispone de un método en su módulo *utils* llamado *plot_model* que sirve para guardar una representación de las redes. Sin embargo, no hemos utilizado ese método porque representan todas las capas y entonces salían imágenes demasiado grandes para añadirlas a esta memoria. Por ello, las representaciones gráficas que aparecen se han hecho siguiendo unas convenciones y comprimiendo varias capas para conseguir imágenes más compactas y didácticas.

1. Colores: en general, las capas de entrada se dibujan en azul, las recurrentes en morado, las convolucionales en gris, las densas en verde y las intermedias (concatenaciones, sumas, *max pooling*) en amarillo.
2. Las de entrada se representan con el texto "Input *<dimensionalidad de la entrada>*"
3. Las recurrentes, en general *Long Short Term Memory* (LSTM), tienen el texto "LSTM*<número de unidades>*"
4. Las convolucionales las escribimos como "C*<número de filtros>*@*<dimensionalidad de los filtros>*"
5. Para las densas ponemos como texto "Dense*<número de neuronas>*".
6. El *max pooling* o *average pooling* lo escribimos como "MaxPool *<tamaño del pooling>*"
7. "*Concat*" representa una capa de concatenación y "*Add*" una de suma.
8. Abreviamos el *batch normalization* como "BN" y el ruido gaussiano como "GN".
9. En general, contraemos en una sola caja a todas las capas que van desde una capa "principal", es decir, una convolucional, una LSTM o una densa, hasta su activación o *dropout* si lo hay.
10. También de forma general comprimimos todas las capas que van entre una activación y la siguiente capa "principal".

Un ejemplo de este tipo de representaciones se puede ver en la Figura 1.

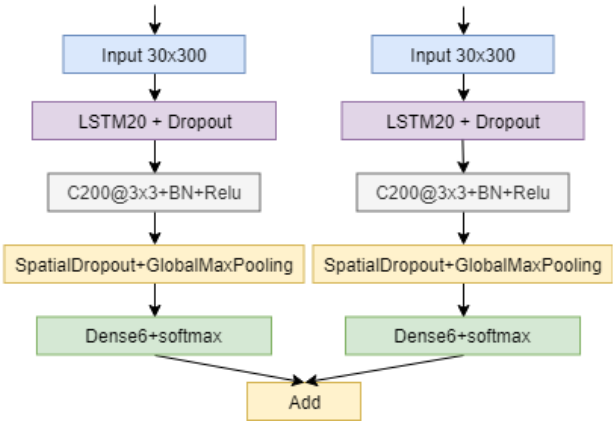


Figura 1: Ejemplo de representación de red neuronal

1.5 Conceptos previos

En este apartado explicamos brevemente qué es una red neuronal, cómo funcionan los diferentes tipos de redes neuronales utilizadas, el funcionamiento de algunos tipos de capas y una técnica frecuentemente utilizada en el entrenamiento de este tipo de modelos.

1.5.1 Redes neuronales artificiales

Las redes neuronales son un modelo computacional conexionista ligeramente inspirado en las conexiones neuronales biológicas. Están formados por un conjunto de nodos (neuronas), organizados en capas. De este modo, en las redes neuronales clásicas o *fully connected*, a las que a veces nos referiremos como "normales", todos los nodos de una capa se conectan mediante aristas dirigidas con pesos a todos los nodos de la capa siguiente. Los datos son introducidos por la primera capa y se propagan multiplicándose por los pesos correspondientes hasta una capa de salida. Además, en cada neurona tenemos una función de activación, generalmente no lineal, que proporciona al modelo la capacidad de resolver problemas no linealmente separables.

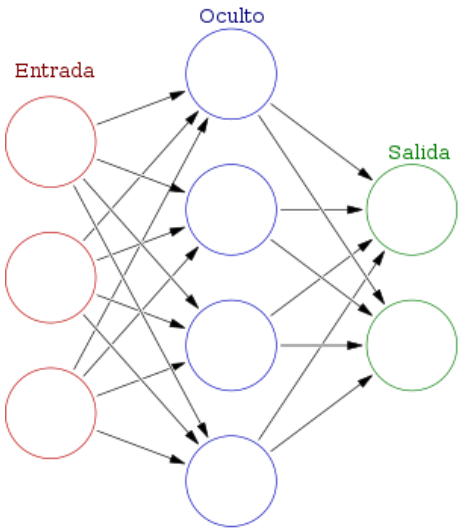


Figura 2: Esquema de ejemplo de una red neuronal normal

1.5.2 Redes convolucionales

Las redes convolucionales son un tipo de redes neuronales artificiales que simulan el córtex visual del ojo humano y que generalmente se utilizan para la clasificación de imágenes. En este

caso, las neuronas se corresponden con filtros de un tamaño concreto, que no son más que una ventana deslizante con pesos que va moviéndose a través de la imagen y generando una nueva imagen. Cuando el filtro está en una posición concreta de la imagen original, obtiene un píxel de la imagen salida al multiplicar el valor de cada píxel que cae dentro de la ventana por el peso de la posición correspondiente en el filtro. Como puede observarse, es un modelo que apenas requiere preproceso de los datos, ya que opera directamente con imágenes.

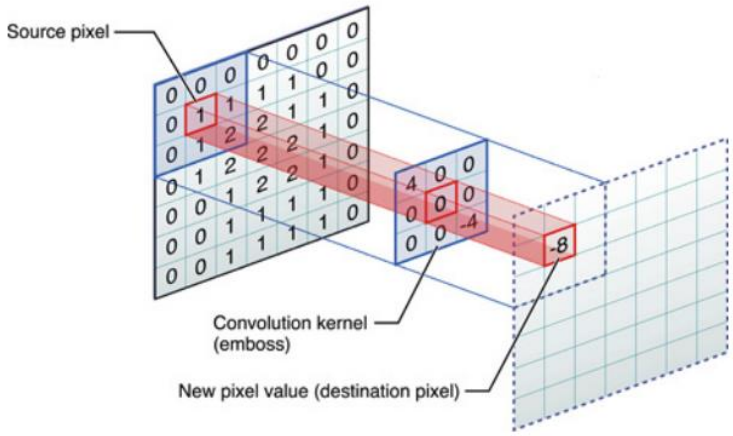


Figura 3: Esquema de funcionamiento de una capa convolucional

1.5.3 Redes recurrentes

Las redes recurrentes son un tipo de red neuronal artificial capaz de procesar secuencias de longitud variable. Son por tanto muy útiles en el procesamiento de texto o audio, ya que además consiguen capturar la relación temporal de los datos de entrada. Funcionan de forma que pasan el primer elemento de la entrada por una red normal (generalmente pequeña) y, al ir a procesar el segundo elemento de la secuencia tenemos en cuenta, no solo dicho elemento, sino el resultado del anterior. Además, existe un tipo concreto de redes recurrentes, las *Long Short-Term Memory* (LSTM) que resuelven algunos problemas que presentan las redes recurrentes habituales.

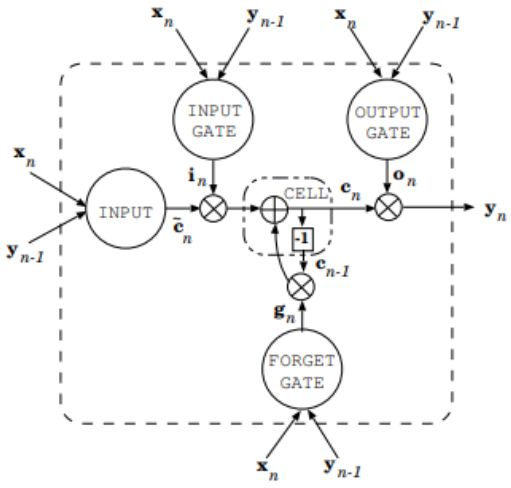


Figura 4: Esquema de una LSTM

1.5.4 Otras capas

Además de las ya descritas, utilizamos algunas capas especiales en nuestros modelos que describimos brevemente a continuación:

- *Max Pooling / Average Pooling*: Reduce el tamaño de las imágenes agrupando ventanas de dimensión variable y quedándose con el valor máximo o medio de cada ventana de píxeles. Por ejemplo, al aplicar un *max pooling* 2x2, la imagen resultante sería de la mitad de tamaño que la original, de forma que por cada cuadrado 2x2 de píxeles de la imagen original, obtendríamos un solo píxel con el valor máximo o medio de dicho cuadrado.
- *Global Max Pooling / Global Average Pooling*: De cada mapa de una imagen, se queda solo con un píxel, el de mayor valor o la media. Es equivalente a un *max pooling / average pooling* del tamaño de la imagen.
- *Batch Normalization*: Normaliza los datos de la capa anterior.
- *Gaussian noise:* Añaden ruido gaussiano para paliar el sobreentrenamiento.
- *Dropout*: Con una cierta probabilidad, fija a 0 el valor de una neurona.
- *Spatial Dropout*: Con una cierta probabilidad, fija a 0 el valor de un mapa entero en una red convolucional.

1.5.5 *Data augmentation*

Un método habitual y en general efectivo para evitar el sobreentrenamiento de una red neuronal es el *data augmentation*. Esta técnica consiste en, durante el entrenamiento, realizar modificaciones aleatorias de las muestras que van pasando por la red. Así, una imagen puede que en un *epoch* llegue sin alterar, en otro invertida horizontalmente, en otro girado 5°, etc. De esta forma, evitamos que la red memorice las imágenes, y le enseñamos a buscar los patrones que debería aprender. Las transformaciones que queremos que se apliquen y en qué magnitud (por ejemplo, el ángulo máximo de giro), son parámetros que hay que especificar.

CAPÍTULO 2

Estado del arte

En este apartado repasamos la literatura acerca del tema escogido para el presente trabajo. Empezaremos con un breve repaso sobre teoría de emociones desde un punto de vista más psicológico. Entonces repasaremos los trabajos sobre clasificadores unimodales de emociones, para después centrarnos en el estado actual de los clasificadores multimodales. Por último, haremos una crítica al estado del arte y presentaremos la aportación original que proporciona nuestro trabajo.

2.1 Emociones

La primera vez que se puso el foco de una forma científica sobre el estudio de las emociones humanas fue en el siglo XIX. En [6], Charles Darwin exponía su descubrimiento de que ciertas expresiones de emociones eran universales e incluso presentaban similitudes con expresiones homólogas en animales.

Mucho más tarde, Paul Ekman tomó como base estos estudios para investigar y defender que las emociones son discretas, medibles y psicológicamente distinguibles. En [7], concluye que existen seis emociones básicas universales: enfado, asco, miedo, felicidad, tristeza y sorpresa. Este es un modelo clásico muy utilizado y es el que nosotros elegimos. Más adelante, el propio Ekman extendió su lista para incluir más emociones, algunas de las cuales no se correspondían con expresiones faciales [8].

En 1982, Robert Plutchik planteaba un modelo similar al de Paul Ekman, pero representado como una "rueda de emociones" [9]. Así contrastaba cada emoción con su contraria, además de considerar diferentes intensidades para cada emoción, como se puede ver en la Figura 5.

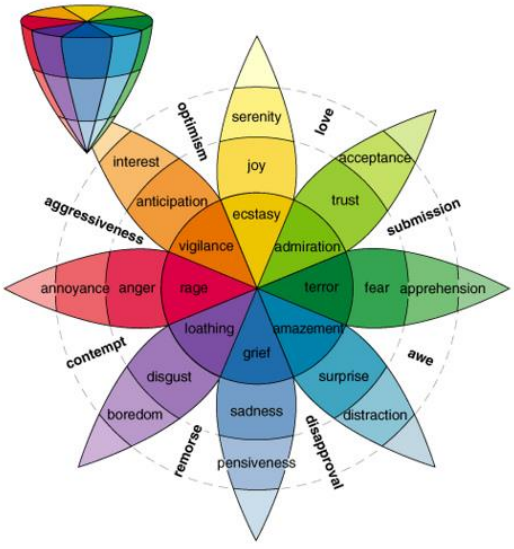


Figura 5: Rueda de emociones de Robert Plutchik

Otros estudios posteriores intentan de maneras diferentes seguir con esta filosofía y realizan diferentes clasificaciones discretas de emociones [10], [11].

Como contraparte, existen aproximaciones continuas de representación de emociones en varias dimensiones. El modelo más conocido en este sentido es el modelo circunflejo [12], que establece que cualquier emoción puede expresarse en un espacio de dos dimensiones: *valence* y *arousal*. La dimensión *valence* se corresponde con el placer, y diferencia emociones positivas (felicidad) y negativas (miedo). Por otra parte, el *arousal* separa las emociones activas (enfado) de las pasivas (tristeza). De esta forma, cada una de las emociones se representaría como un punto en ese espacio.

Posteriormente, Russell y Mehrabian extendieron esta teoría, creando el modelo PAD [13]. Este modelo introducía una nueva dimensión, *dominance*. De esta forma, se resolvían algunos problemas que planteaba el modelo bidimensional, como que el miedo y la ira no podían distinguirse sin una tercera dimensión. *Dominance* representa, para resolver estas deficiencias, el impulso de huir o luchar ante un estímulo de ese tipo.

Otro modelo bastante popular, es el expuesto en [14], PANAS. Este sistema se divide en veinte ítems, diez correspondientes a emociones positivas y diez a negativas. De esta forma, cada uno de los ítems se puntúa con un número del uno al cinco. Las dos dimensiones que define (positivo y negativo), tienen muy poca correlación entre ellas, lo cual significa que valores bajos del estado positivo no implican un valor significativo del estado negativo.

2.2 Expresiones faciales

Una de las herramientas más importantes que tenemos los humanos para expresar emociones son las expresiones faciales. Es por ello que el problema de reconocimiento automático de expresiones faciales está en el punto de mira desde más de 25 años. A lo largo de todo este tiempo, diferentes técnicas de aprendizaje automático se han ido desarrollando y probando sobre este problema.

Clásicamente, la clasificación automática de expresiones faciales supone un proceso en dos pasos: la extracción de características de las imágenes y posteriormente la clasificación. La parte de clasificación no merece especial mención, ya que, tradicionalmente, se utilizan algoritmos clásicos de aprendizaje automático, como LDA [15] o *Support Vector Machine* (SVM) [16]. En cuanto a la extracción de características, una de las primeras aproximaciones se basa en utilizar *Gabor wavelets* [17]. Estos son funciones que simulan la percepción del sistema visual humano [18]. También se ha utilizado la técnica de *Local Binary Patterns* (LBP) para la parte de extracción de características, como se puede ver en [19] (donde de hecho, se hace una comparación con los *Gabor wavelets*, demostrando una mejoría en la precisión del sistema). Esta es una técnica que fue diseñada para análisis de texturas y que se basa en utilizar un operador de vecindad circular para construir un histograma. Otra técnica podría ser la de *geometric deformation features*, que tiene en cuenta el desplazamiento de unos puntos de una malla sobre la cara a lo largo de una secuencia de imágenes que terminan en el pico de la emoción en cuestión, obteniendo resultados bastante buenos [20].

Estos modelos sin embargo presentan varios defectos. El primero de ellos es que, en algunos casos, el proceso es semi-automático, y puede requerir de un operador humano. Por ejemplo, con los *geometric deformation features*, a la hora de colocar la malla. El segundo problema que plantean es que consiste en un proceso en dos partes, que además de ser costoso, implica que la precisión del sistema depende de los dos modelos, haciendo más difícil su optimización.

Con la llegada de las redes neuronales artificiales (NN) y, en especial, de las redes neuronales convolucionales (CNN), estos modelos para este tipo de tareas quedaron obsoletos. Esto sucede

porque una CNN supone un único sistema totalmente automático, rápido, que no requiere de ningún tipo de extracción de características, y que además mejora la calidad de los resultados en general. Por ello, la mayoría de los sistemas actuales de reconocimiento de expresiones faciales utilizan este tipo de redes [21], [22].

2.3 Voz

Como ya se ha mencionado, cuando hablamos de voz nos referimos al cómo dice algo (prosodia) y no al qué se dice. Nos referimos a esta última categoría como "Texto". Aun así, es muy habitual en la literatura encontrar estudios que combinan características acústicas y lingüísticas para obtener mejores resultados [23], [24]. En general, para el análisis de datos acústicos, el primer paso consiste en extraer características globales relacionadas con el tono, la energía o la duración de las señales [25]. Otras características más complejas, como los *Mel Frequency Cepstral Coefficients* (MFCC) o *Spectral Rolloff* también son utilizadas [26]. Para la clasificación, de nuevo, se pueden utilizar métodos típicos de reconocimiento de formas, como modelos ocultos de Markov [25] o SVM [24].

Más recientes estudios aprovechan la potencia de las redes neuronales para la clasificación del audio. Por ejemplo, en [26] se hace un exhaustivo estudio en el que se compara el desempeño de los diferentes tipos de redes neuronales en la tarea de reconocimiento de emociones a partir de audio. Como las características que extraen son bidimensionales (MFCC), pueden aplicar redes densas normales aplanando las características, redes convolucionales, o redes recurrentes. Aunque la LSTM puede parecer la opción más acertada, ya que es muy hábil procesando secuencias con una relación temporal, la CNN ofrece unos resultados muy prometedores. No es el único estudio en considerar las redes convolucionales para el análisis de audio, ya que debido a su velocidad y eficacia son un tipo de redes muy utilizadas hoy en día. Por ejemplo, en [27] se utiliza un tipo de CNN para detección de emociones en secuencias de audio, tomando como entrada espectrogramas de las señales acústicas.

2.4 Texto

El procesado automático de texto y su clasificación se ha convertido en un problema muy popular desde el estallido de las redes sociales. El éxito de, por ejemplo, Twitter provoca, no solo que se disponga de una gran cantidad de datos públicos para este tipo de sistemas, sino también que sea interesante disponer de modelos para detectar automáticamente, por ejemplo, mensajes de odio [28]. Otras tareas del estilo pueden ser la detección de sarcasmo [29], o de la polaridad [30].

Para este tipo de tareas es habitual realizar una extracción de características como puede ser *n-grams* o *skip-grams*, y después utilizar algún clasificador clásico, como puede ser SVM [28]. Además, previamente a la extracción de características, se suele realizar algún tipo de preproceso, como *stemming* o lematización, sobre el lenguaje natural [30].

De nuevo, las más recientes aproximaciones demuestran que las redes neuronales artificiales son un muy valioso recurso para este tipo de tareas. Como sucede con el audio, el texto requiere de una extracción de características previa al uso de las redes neuronales. En este sentido, es habitual utilizar representaciones como *bag-of-words* o *word embeddings*. Esto se realiza en [31], donde hace una comparación sobre qué tipo de red funciona mejor para la tarea de clasificación de polaridad, entre CNN, LSTM y *Deep Averaging Networks*. Por otra parte, resulta muy interesante comprobar cómo, incluso en competiciones anuales, mejoran los resultados de los equipos que introducen este tipo de técnicas en las más recientes ediciones [31] [32]

respecto a los que obtuvieron en ediciones anteriores, en las que utilizaron otros clasificadores [33].

Otros trabajos[1] también realizan revisiones de artículos para obtener comparativas exhaustivas sobre el comportamiento de diferentes tipos de redes neuronales en esta tarea, demostrando que algunas variantes de estos modelos ofrecen resultados muy prometedores.

2.5 Multimodal

La literatura en este campo es menos abundante, ya que crear reconocedores multimodales es una tarea laboriosa que implica el desarrollo de varios sistemas y su correcta combinación.

Lo primero que habría que preguntarse es si un sistema multimodal ofrece mejoras reales y significativas respecto a sistemas unimodales. En [34] se hace un metaanálisis de 30 estudios que comparan el acierto de sistemas multimodales frente a sus componentes, concluyendo que, en general, los sistemas multimodales mejoran en un 8.12% los resultados del mejor de los clasificadores unimodales que los componen. Si bien es cierto que la mejora es considerablemente mayor en datos actuados que en datos espontáneos.

La mayoría de los sistemas desarrollados en la literatura combinan características faciales y acústicas, como en [35] o en [36]. Otros, como [37] utilizan únicamente el habla, combinando características acústicas junto con el propio mensaje hablado (texto).

Los más interesantes resultan los modelos trimodales. Por ejemplo, en [38] se entrena un clasificador que combina expresiones faciales, postura corporal y texto, utilizando un clasificador bayesiano para una tarea de clasificación en 8 emociones. El que más nos interesa, sin embargo, es [39] porque, igual que en nuestra propuesta, se intenta combinar imágenes faciales, audio y texto. Para ello, realizan una extracción de características de los rasgos faciales (distancia entre ojos, posición del labio superior, etc.), del audio (*mel frequency cepstral coefficient, spectral centroid,* etc.) y del texto (*word embeddings, part of speech*, etc.). En el artículo se prueba dicho sistema multimodal y se compara con las componentes unimodales para las tareas de clasificación de polaridad (negativo o positivo), así como en clasificación en cuatro emociones básicas (feliz, triste, furioso y neutro), demostrando que el sistema multimodal mejora en todos los casos a los clasificadores unimodales o bimodales.

2.6 Aportación original

Como ya hemos visto, hay algunos intentos de desarrollar reconocedores multimodales de emociones. Sin embargo, por una parte, existen algunas carencias en esos modelos que intentaremos cubrir en este trabajo. En primer lugar, los modelos bimodales no presentan una diferencia significativa con los unimodales, por eso nosotros intentamos diseñar un sistema más complejo, con tres tipos de entrada. En cuanto a los modelos trimodales, que son escasos, encontramos que, en algunos casos, tratan tareas de poco valor, como el clasificador que hemos visto en tan solo cuatro emociones. Además, por tener unos cuantos años, pocos de los sistemas encontrados utilizan redes neuronales para la clasificación, aunque, como hemos visto en los apartados anteriores, estas han demostrado (con la suficiente cantidad de datos de entrenamiento) mejorar en general los resultados obtenidos por otros clasificadores, como los SVM o las redes bayesianas.

[1] https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f

Por otra parte, nosotros proponemos desarrollar un sistema multimodal que pueda operar en una aplicación en directo e implementamos dicha aplicación. Si bien es cierto que la información sobre la velocidad de la extracción de características y de los clasificadores no suele proporcionarse en los artículos, así que no podemos asegurar que los sistemas revisados no pudieran utilizarse de esta manera.

Por último, algo que en general se obvia al diseñar este tipo de sistemas es que, si bien las expresiones faciales pueden considerarse universales, no sucede lo mismo con el modo en que hablamos. Tanto la prosodia al expresar emociones como, evidentemente, la forma en que construimos frases para hacerlo son locales, dependientes del lenguaje y la ubicación geográfica. Así, no entonamos igual para mostrar enfado que los ingleses o los mexicanos. Por ello, una importante aportación de nuestro proyecto es que desarrollamos un sistema de reconocimiento de emociones multimodal en español de España. Lo cual se diferencia de los demás trabajos, que suelen utilizar corpus de audio y texto en inglés.

CAPÍTULO 3

Desarrollo del reconocedor multimodal

3.1 Expresiones faciales

Como ya se ha dicho, se pretende desarrollar un clasificador en 7 emociones básicas a partir de expresiones faciales utilizando redes convolucionales. Las emociones que se van a tratar son: felicidad, enfado, asco, miedo, sorpresa, tristeza y neutro.

3.1.1 Corpus

Para la tarea se han recogido un compendio de corpus reconocidos y disponibles en internet. Los motivos para escoger más de uno son varios: tener una cantidad significativa de muestras y obligar a la red entrenada a aprender invarianza a diferentes aspectos como raza, género, calidad de la imagen o condiciones de iluminación. De esta forma obtendremos una CNN realmente fiable, robusta y completa. Los corpus elegidos son los siguientes:

- *Cohn-Kanade database* (CK+) [40] [41]

Este corpus facial consta de dos versiones.

La primera contiene 486 secuencias de 97 actores. Estas secuencias fotográficas empiezan con una expresión neutra y terminan en un pico de una emoción concreta. Todas las imágenes que contiene están en escala de grises y se corresponden a emociones posadas.

La segunda versión es una ampliación de la primera que añade 107 secuencias de 26 nuevos individuos. Además, las nuevas secuencias incluyen fotografías a color y emociones espontáneas (no posadas). Es por ello que la versión que hemos utilizado es esta última.

Los actores son hombres y mujeres de edad media y distinto origen (americanos, afroamericanos, asiáticos y latinos). Además, las emociones contenidas en este corpus son las 7 emociones básicas con las que trabajamos y una octava: desprecio. Esta última la hemos descartado porque no es muy habitual en los estudios de este estilo y, por ello, tampoco suele considerarse en otros corpus como este, por lo que tendríamos muy pocas muestras con esta emoción.

Tal y como se ha descrito, las muestras se componen de secuencias de fotográficas que van desde un estado neutro hasta una emoción. Es por ello que hemos escogido, de cada secuencia, la última imagen, quedándonos así con 593 imágenes etiquetadas con la correspondiente emoción.

- *The Japanese Female Facial Expression Database* (JAFFE) [42]

Como su nombre indica, este corpus cuenta únicamente con imágenes de mujeres japonesas de edad joven/media. En concreto son 213 fotografías en escala de grises de 10 mujeres posando las 7 emociones básicas que en este trabajo consideramos. Aunque pueda parecer limitada la utilidad de este corpus, ya que contiene solo mujeres de edad similar y de un mismo país, puede resultar útil para obligar a nuestro modelo a aprender invarianza al origen del sujeto (la mayoría de corpus contienen fotos de personas de raza caucásica).

- *Karolinska Directed Emotional Faces* (KDEF) [43]

Este corpus cuenta con un total de 4900 imágenes en color de 70 actores. Los actores son la mitad hombres y la mitad mujeres de entre 20 y 30 años, y todos ellos con la cara totalmente descubierta (sin barba, bigote, gafas, maquillaje ni pendientes). Las emociones utilizadas son las 7 básicas que nosotros hemos considerado. Además, para cada expresión se tienen 5 puntos de vista, es decir, para cada emoción interpretada por cada actor, se han tomado 5 fotografías desde diferentes ángulos (de frente, 45° a la derecha y a la izquierda, y los dos laterales). Cada actor interpreta cada emoción 2 veces, así que realmente tendríamos 10 imágenes para cada actor y cada emoción. Para adecuarnos al resto de nuestra información, evitar redundancia y ser capaces de detectar correctamente las caras, hemos omitido las fotografías de los dos perfiles, quedándonos por tanto con 42 (3 puntos de vista x 2 veces x 7 emociones) imágenes para cada actor, es decir, un total de 2940.

- *Emotion Detection From Facial Expressions*[2] (EDFFE)

Este corpus es el training set de una competición de *Kaggle* y se compone de imágenes de *Labeled Faces in the Wild*[3] y de *Indian Movie Face DataBase*[4] (IMFDB) etiquetadas. Se utiliza este en vez de los dos corpus mencionados porque *Labeled Faces in the Wild* no viene etiquetado con emociones y porque *Indian Movie Face DataBase* consta exclusivamente de capturas de películas de Bollywood y tener demasiadas muestras de este tipo podría sesgar la red. Son un total de 13691 imágenes en color de todo tipo (etnias y edades) clasificadas en nuestras 7 emociones básicas y "desprecio", la cual de nuevo descartamos.

- *The extended yale face database B* [44]

Este corpus consta de 16128 fotografías de hombres y mujeres con expresión neutral bajo diferentes condiciones de iluminación. Se tomaron 576 fotografías en escala de grises de cada actor en la misma posición y con la misma expresión facial neutra variando la iluminación. Como todas son con la misma expresión, este corpus no servirá para clasificar emociones, pero sí para aportar invarianza a las condiciones de iluminación. Es por ello también que no se ha utilizado todo el corpus, sino que se ha seleccionado un subconjunto de 1172 imágenes correspondientes a 2 de los actores.

- *Challenges in Representation Learning: Facial Expression Recognition Challenge* (FER)[5]

Este último corpus pertenece también a una competición, pero en este caso es original. Además, al constar de 35888 fotografías, constituye el grueso de nuestro set. Todas las imágenes están en escala de grises y tienen un tamaño de 48x48 píxeles. Se clasifican en las 7 emociones básicas ya mencionadas. Las imágenes fueron recolectadas de internet, así que se corresponden con emociones espontáneas, y pertenecen a hombres y mujeres de todas las edades.

3.1.2 Extracción de características

Como vamos a utilizar redes convolucionales, la extracción de características simplemente consiste en conseguir las caras de nuestro conjunto de corpus. Por eso, para construir nuestro *dataset*, hemos unido todas estas imágenes y las hemos pasado por un detector de caras en

[2] https://www.kaggle.com/c/emotion-detection-from-facial-expressions
[3] http://vis-www.cs.umass.edu/lfw/
[4] http://cvit.iiit.ac.in/projects/IMFDB/
[5] https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

Python: *face_recognition*[6]. Como algunos de los corpus constan de imágenes obtenidas de internet o de forma automática, algunas fotografías de mala calidad se descartaron (aquellas en las que el detector no encontraba la cara). Recortamos todas las imágenes utilizando el detector, añadimos bordes negros para que sea cuadrada, las redimensionamos todas a 48x48 y las guardamos en escala de grises. Así, conseguimos un *dataset* de un total de 41246 imágenes que dividimos aleatoriamente en un 70% de training, un 15% de validación y un 15% de test. Todas las pruebas se hicieron evaluando sobre el conjunto de validación y solo al final damos un resultado sobre el conjunto de test.

3.1.3 Modelo inicial

Para empezar, se ha decidido no escribir una red neuronal cualquiera, sino partir de 3 CNNs sencillas propuestas en 3 artículos que tratan este mismo problema de clasificación. En la Figura 6 se indica esquemáticamente la estructura de las redes propuestas, así como las referencias a los artículos originales.

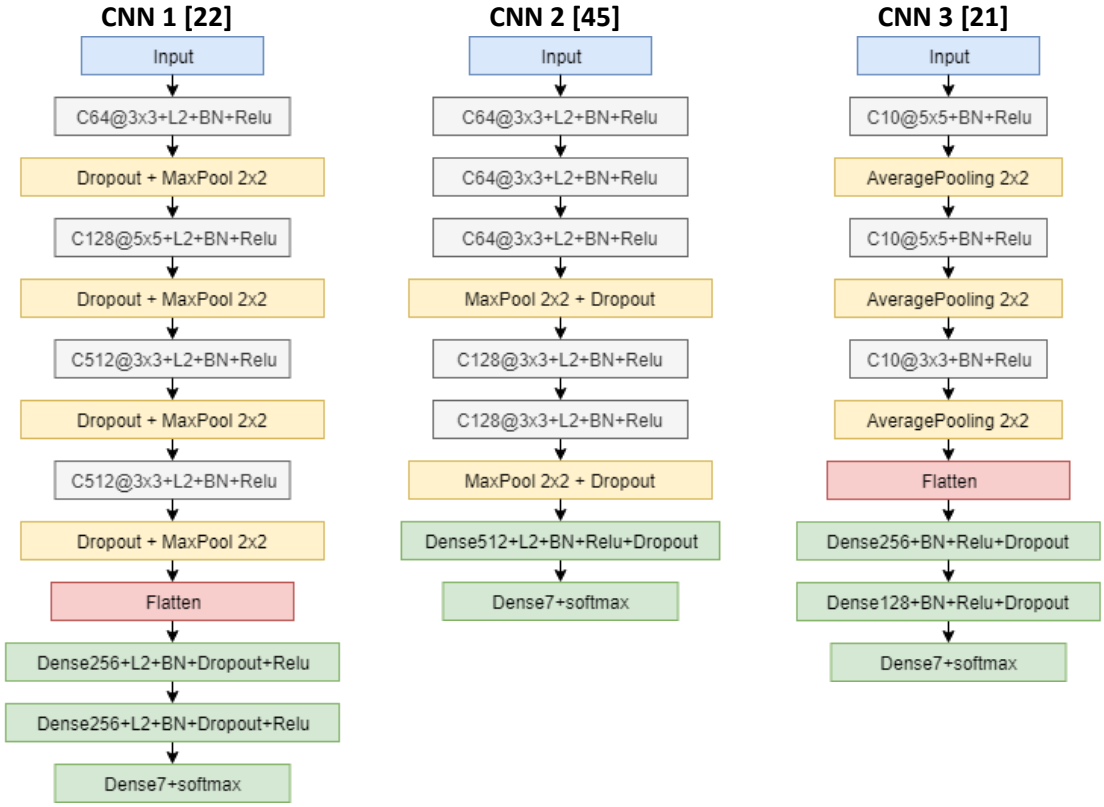


Figura 6: Propuestas de modelo inicial para el problema facial

Como función de pérdida se ha elegido el *categorical crossentropy* y se ha utilizado *adam* como el optimizador para todas las pruebas.

Entonces probamos estos 3 modelos sobre nuestro dataset con 300 *epochs* y un *batch size* de 50. Como resultado de *accuracy*, para que sea lo más realista posible, tomamos la media del *accuracy* obtenido en los últimos 30 *epochs*. De acuerdo con los resultados iniciales, que podemos ver en la Tabla 1, tomaremos como base la CNN 2 para estimar los mejores parámetros y la mejor topología para nuestro problema.

[6] https://github.com/ageitgey/face_recognition

	CNN 1	CNN 2	CNN 3
Validation accuracy (%)	64.20	71.55	67.22

Tabla 1: Resultados de los modelos iniciales para el problema facial

3.1.4 Preproceso

Respecto al tratamiento del corpus también se han perfeccionado dos aspectos: el tamaño óptimo de las imágenes tras el recortado (hasta ahora 48x48) y la forma de recortar las caras.

En cuanto al tamaño, hemos probado tanto aumentarlo como reducirlo.

	32x32	48x48	64x64	80x80
Validation accuracy (%)	71.39	71.55	70.30	69.88

Tabla 2: Efecto del tamaño de las imágenes en el problema facial

De acuerdo con los resultados de la Tabla 2, nos quedamos con el tamaño que ya estábamos considerando porque al aumentar van empeorando los resultados, posiblemente porque aparece más información inútil que hace de ruido para la red, y al reducirlas parece que empieza a perderse algo de información útil.

En cuanto al recortado de las imágenes se han considerado varias alternativas, teniendo en cuenta que el detector nos da, no una caja, sino los puntos característicos de la cara:

1. Obtener las coordenadas mínima y máxima en x e y para todos los puntos característicos, recortar un rectángulo con esas coordenadas y añadir márgenes negros para hacerla cuadrada antes de redimensionar. Este era el método que habíamos utilizado hasta ahora.
2. Recortar por los puntos más exteriores de la cara, esto es, los de barbilla y cejas, lo que queda fuera de esa superficie se rellena a negro para hacerla cuadrada.
3. Obtener las coordenadas mínima y máxima en x e y, y recortar un cuadrado utilizando como longitud del lado el máximo entre *(xmax – xmin)* y *(ymax - ymin)*. En caso de que ese cuadrado exceda los límites de la imagen, se rellena replicando la última fila o columna de píxeles.

En la Figura 7 vemos un ejemplo de una misma imagen recortada de las tres formas y en la Tabla 3, los resultados obtenidos utilizando cada tipo de recortado.

Figura 7: Diferentes opciones de recortado de caras

	Recortado 1	Recortado 2	Recortado 3
***Validation accuracy* (%)**	71.55	71.41	70.90

Tabla 3: Resultados en función del modo de recortado facial

Nos quedamos de nuevo con el método que estábamos utilizando, mientras que la opción 2 puede que pierda algo de información importante, como el ceño, el 3 puede que añada información inútil que pueda molestar al clasificador.

Por otra parte, como se indica en [46], nuestro corpus mayoritario (FER) podría contener un mal etiquetado, ya que este *dataset* fue creado automáticamente utilizando la API de búsqueda de Google, con la que se recogieron las imágenes buscándolas con palabras clave relacionadas con cada una de las emociones a considerar. Es por esto que Microsoft reetiquetó todo el corpus manualmente, creando una nueva versión del mismo, FER+ [47]. Cada una de las imágenes fue etiquetada por 10 humanos por separado en las 7 clases en cuestión, "desprecio" y "desconocido", por si tenían dudas. Con respecto a este reetiquetado podemos tomar varias decisiones:

1. La clase de cada imagen pasa a ser una distribución de probabilidad proporcional a los votos en lugar de una clase exclusiva.
2. La clase de cada imagen es la emoción que recibió más votos (en caso de empate se elige aleatoriamente entre las empatadas).
3. La clase de cada imagen se elige estocásticamente, donde cada clase tiene una probabilidad proporcional al número de votos recibidos.

Probamos esos tres reetiquetados y mostramos los resultados en la Tabla 4.

	Reetiquetado 1	Reetiquetado 2	Reetiquetado 3
***Validation accuracy* (%)**	80.70	82.52	74.68

Tabla 4: Resultados en función del criterio de reetiquetado del corpus facial FER

Como vemos, los resultados mejoran mucho, y la mejor opción es quizás la más sencilla y la más lógica, la elección de consenso es la más acertada.

3.1.5 Estimación de la red

A continuación, se ha han ido probando modificaciones sobre la topología de la red y sus parámetros para obtener un buen modelo para nuestro problema.

- *Spatial Dropout*

Empezamos sustituyendo el *Dropout* de la parte convolucional por *SpatialDropout2D* como se sugiere generalmente[7], probando además varios valores para él. Los resultados pueden verse en la Tabla 5.

[7] https://keras.rstudio.com/reference/layer_spatial_dropout_2d.html

	Dropout	*SpatialDropout2D*						
	0.25	**0.10**	**0.15**	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**
***Validation accuracy* (%)**	82.52	81.86	81.95	82.13	82.37	82.49	82.54	82.33
Training accuracy (%)	87.72	90.43	89.36	88.46	87.39	86.20	85.05	83.90

Tabla 5: Resultados variando el porcentaje de *spatial dropout* en el problema facial

En este caso hemos añadido también el acierto en *training* ya que es una técnica de regularización para ver, no solo que con 0.35 el *SpatialDropout2D* mejora ligeramente los resultados obtenidos con *Dropout*, sino también que, en general, esta técnica reduce más eficazmente el *overfitting* en redes convolucionales.

- *L2 regularization*

Como seguimos teniendo un ligero de problema de sobreentrenamiento, lo siguiente que afinamos es el valor de l2, que hasta ahora estábamos utilizando con 0.001. No solo hemos probado a variar el valor, sino que también hemos probado a dejarlo sólo en la parte convolucional, solo en la *fully connected* o quitarlo del todo.

En este caso, a la vista de los resultados de la Tabla 6, dejaremos la regularización l2 solo en las capas densas del final con un valor de 0.001 aunque aumente bastante el *overfitting*, teniendo en cuenta que aún no hemos aplicado otras técnicas, como el *data augmentation*.

	En todo		**Solo conv**	**Solo FC**			**Sin L2**
	0.001	**0.01**	**0.001**	**0.001**	**0.0005**	**0.005**	**-**
***Validation accuracy* (%)**	82.54	73.74	82.59	84.83	84.55	84.55	84.38
Training accuracy (%)	85.05	73.65	99.19	94.40	95.14	92.13	99.60

Tabla 6: Efecto de la regularización l2 en el problema facial

- *Data augmentation*

A continuación, cuando sí que tenemos un problema con el *overfitting*, aplicamos técnicas de *data augmentation*. En primer lugar, probamos el más obvio, que es el volteo horizontal, obteniendo los resultados de la Tabla 7.

	Sin *horizontal flip*	**Con *horizontal flip***
***Validation accuracy* (%)**	84.83	86.04
Training accuracy (%)	94.40	90.03

Tabla 7: Resultados obtenidos al realizar volteos horizontales aleatorios a las caras

Continuamos aplicando además rotaciones, con diferentes rangos. De acuerdo a los resultados que vemos en la Tabla 8, nos quedamos con el mejor, que son rotaciones de hasta 10°. Como ya hemos eliminado el problema del *overfitting*, continuamos estimando otros parámetros.

Rango de rotación (°)	**Sin rotación**	**5**	**10**	**15**	**20**
***Validation accuracy* (%)**	86.04	85.98	86.17	86.15	85.48
Training accuracy (%)	90.03	87.43	85.90	84.75	83.95

Tabla 8: Resultados aplicando rotaciones con diferentes rangos a las imágenes faciales

- Tamaño de *batch*

El siguiente parámetro que hemos testado es el *batch size*. A la vista de los resultados reflejados en la Tabla 9, de aquí en adelante utilizamos *batches* de 400 muestras.

Batch size	**10**	**25**	**50**	**100**	**200**	**300**	**400**	**500**
***Validation accuracy* (%)**	85.07	85.93	86.17	86.44	86.58	86.33	86.90	86.82

Tabla 9: Efecto del tamaño de *batch* sobre el entrenamiento en el problema facial

- *Dropout*

También estimamos el valor del *dropout* que queda en las capas densas del final, y que hasta ahora tenía una probabilidad del 25%. De nuevo, como es una técnica de regularización, añadiremos el porcentaje de aciertos en el conjunto de entrenamiento. Como se puede ver en la Tabla 10, ningún otro valor mejora la tasa de acierto, así que nos quedamos con el valor que utilizábamos.

***Dropout* (%)**	**10**	**20**	**25**	**30**	**40**	**50**
***Validation accuracy* (%)**	86.68	86.82	86.90	86.79	86.51	86.71
Training accuracy (%)	91.07	90.23	89.97	89.53	88.93	88.44

Tabla 10: Resultados variando el porcentaje de *dropout* en el problema facial

- Parte *fully connected*

Para perfeccionar la parte *fully connected* del final de nuestra red hemos probado, en primer lugar, a variar el número de neuronas de la única capa densa que tenemos (sin contar la de salida), obteniendo los resultados que se ven en la Tabla 11. De nuevo no conseguimos mejorar los resultados variando el tamaño.

Neuronas	**256**	**512**	**1024**
***Validation accuracy* (%)**	86.77	86.90	86.80

Tabla 11: Efecto del número de neuronas de la capa densa en el problema facial

Por otra parte, hemos probado a añadir más capas de diferentes formas. Las arquitecturas que se han probado para esa parte de la red (entre el *flatten* y la capa de salida) son las siguientes:

1. BloqueFC (256) + BloqueFC (512)
2. BloqueFC (512) + BloqueFC (1024)
3. BloqueFC (256) + BloqueFC (512) + BloqueFC (1024)
4. BloqueFC (128) + BloqueFC (256) + BloqueFC (512)
5. BloqueFC (512) + BloqueFC (1024) + BloqueFC (2048)

Donde "BloqueFC" implica una capa densa con el número de neuronas indicado y regularización l2, seguido de *batch normalization*, una activación *relu* y *dropout*. Como se puede ver en la Tabla 12, de nuevo no conseguimos mejorar los resultados, así que conservamos el modelo que veníamos utilizando.

	Actual	**Opción 1**	**Opción 2**	**Opción 3**	**Opción 4**	**Opción 5**
***Validation accuracy* (%)**	86.90	86.77	86.62	86.49	86.28	86.36

Tabla 12: Pruebas con distintas arquitecturas para la parte densa en el problema facial

- Parte convolucional

En la parte convolucional se han hecho pruebas similares. En primer lugar, recordemos que la parte convolucional consta de 3 bloques convolucionales de 64 filtros 3x3 seguidos de *maxpooling* 2x2 y *spatialdropout*, y después otros 2 bloques convolucionales de 128 filtros 3x3 seguidos de nuevo por *maxpooling* 2x2 y *spatialdropout*. Cada bloque convolucional está constituido por una capa convolucional, un *batchnorm* y una *relu*.

Entonces, primero hemos probado a variar el número de filtros de las capas convolucionales manteniendo que las tres primeras tienen el mismo número y las dos segundas también, además de que las dos segundas tienen el doble de filtros que las tres primeras. Aclarado esto, mostramos los resultados en la Tabla 13, donde se puede ver que los resultados mejoran si las tres primeras convolucionales tienen 128 filtros, y las dos segundas, 256.

Número de filtros de la primera capa	32	64	128
***Validation accuracy* (%)**	86.13	86.90	86.98

Tabla 13: Resultados obtenidos variando el número filtros de la CNN del problema facial

Para esquematizar esta parte se utiliza una notación como "(BC 64) x 4". Esto quiere decir que hay consecutivos 4 bloques compuestos por: una capa convolucional del número de filtros indicados, de tamaño siempre 3x3, *batchnorm* y una *relu*. Abreviaremos también los intermedios conformados por una capa *maxpooling* 2x2 seguido por *spatialdropout* como I. Como antes, también se ha probado a cambiar la arquitectura de la parte convolucional, las alternativas propuestas son las siguientes:

1. (BC 64) x 4 + I + (BC 128) x 3 + I + (BC 256) x 2 + I
2. (BC 128) x 3 + I + (BC 256) x 2 + I + (BC 512) + I
3. (BC 64) x 4 + I + (BC 128) x 3 + I + (BC 256) x 2 + I + (BC 512) + I
4. (BC 32) x 5 + I + (BC 64) x 4 + I + (BC 128) x 3 + I + (BC 256) x 2 + I + (BC 512) + I
5. (BC 64) x 4 + I + (BC 128) x 3 + I + (BC 256) x 2 + I + (BC 512) + I + (BC 1024) + I

Los resultados obtenidos se observan en la Tabla 14, por los cuales nos quedamos con la última arquitectura.

	Actual	Opción 1	Opción 2	Opción 3	Opción 4	Opción 5
***Validation accuracy* (%)**	86.98	87.76	87.78	88.15	87.65	88.50

Tabla 14: Pruebas con distintas estructuras para la parte convolucional en el problema facial

- *Overfitting*

De nuevo, al aumentar la capacidad de la red, nos enfrentamos a un problema de *overfitting*, ya que la tasa de aciertos sobre el conjunto de entrenamiento se ha elevado hasta el 97.47%. Por ello, se han aplicado técnicas adicionales de regularización.

En primer lugar, se ha probado a añadir al *data augmentation* actual, *shear*, ya que también tiene sentido, con diferentes valores para el ángulo máximo, ofreciendo los resultados de la Tabla 15. Añadimos por tanto *shear* con un rango de 10°, que incrementa ligeramente los resultados y reduce ligeramente el problema del sobreentrenamiento.

Rango (°)	Sin *shear*	10	15	20	25	30	35	40
***Validation accuracy* (%)**	88.50	88.52	88.25	88.34	88.34	88.30	88.03	88.17
Training accuracy (%)	97.47	97.20	96.72	96.19	95.64	95.12	94.38	93.64

Tabla 15: Resultados al aplicar *shear* con distintos rangos a los datos del problema facial

A continuación, se ha probado a utilizar la técnica de *cutout* o *random erasing*, que consiste en tapar partes de la imagen mediante "parches" de un color uniforme. Hay que tener en cuenta que a priori parece poco recomendable para la tarea en cuestión. No se he implementado, sino que se ha utilizado una librería[8]. Para ello, se ha fijado la mínima y máxima superficie de recortado al 1% y 10% respectivamente, y hemos probado a variar la probabilidad de recorte, obteniendo los resultados de la Tabla 16. A raíz de ello comprobamos, como parecía previsible, que esta técnica empeora los resultados, sin ofrecer una gran mejora con el problema del *overfitting*, así que la descartamos.

Probabilidad (%)	Sin *cutout*	10	20	30	40	50
***Validation accuracy* (%)**	88.52	87.75	87.94	88.00	87.81	87.77
Training accuracy (%)	97.20	95.86	95.63	96.00	93.74	94.76

Tabla 16: Pruebas realizadas añadiendo la técnica de *cutout* a los datos del problema facial

También se ha probado a añadir ruido gaussiano entre el *batchnorm* y la *relu*. Además de probar varios porcentajes, se ha probado a añadirlo en todas las capas, solo en las convolucionales, o solo antes de los *maxpooling*. A la vista de los resultados mostrados en la Tabla 17, lo descartamos, ya que no conseguimos ninguna mejoría significativa en uno ni en otro sentido

Probabilidad (%)		Sin ruido	10	20	30	40	50
Todas las capas	*Validation accuracy* (%)	88.52	88.42	88.47	87.71	88.10	88.12
	Training accuracy (%)	97.20	96.91	96.29	95.77	95.20	94.70
Convolucionales	*Validation accuracy* (%)	-	88.29	88.28	88.31	88.30	88.25
	Training accuracy (%)	-	96.92	96.36	95.92	95.34	94.74
Antes de *maxpooling*	*Validation accuracy* (%)	-	88.34	88.08	88.05	88.22	87.97
	Training accuracy (%)	-	97.10	96.88	96.64	96.50	96.34

Tabla 17: Resultados obtenidos al añadir ruido gaussiano en la red del problema facial

Por último, probamos la técnica de *mixup data augmentation* (MDA), variando el parámetro alpha de la función beta y utilizando para ello una librería de *Python*[9]. Esta técnica consiste en mezclar aleatoriamente pares de imágenes y sus etiquetas con unos determinados pesos, creando así muestras híbridas.

[8] https://github.com/yu4u/cutout-random-erasing
[9] https://github.com/yu4u/mixup-generator

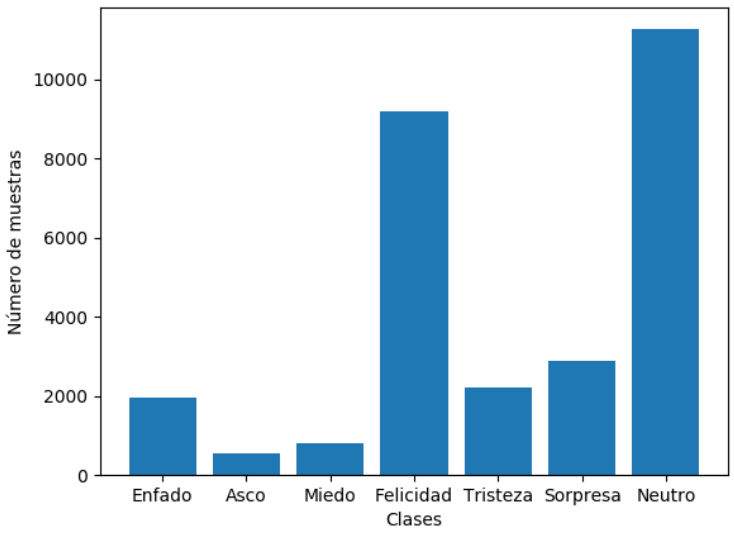
Alpha	Sin MDA	0.1	0.2	0.3	0.4	0.5
***Validation accuracy* (%)**	88.52	88.55	88.58	88.83	88.57	88.90
Training accuracy (%)	97.20	93.89	92.78	91.10	89.90	88.75
Alpha	**-**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
***Validation accuracy* (%)**	-	88.59	88.71	88.50	88.70	88.72
Training accuracy (%)	-	87.77	86.90	86.12	85.35	84.60

Tabla 18: Resultados al incorporar la técnica de *mixup data augmentation* al problema facial

En este caso sí que resulta una técnica eficaz, como podemos ver en la Tabla 18, ya que con un *alpha = 0.5* no solo eliminamos el *overfitting*, sino que también mejoramos los resultados. Así que incorporamos esta solución.

- Balanceado

Para terminar con las pruebas sobre este modelo secuencial, como el número de muestras por cada clase es muy dispar (como se puede ver en la Gráfica 1), se ha intentado balancear el peso de esas clases de forma inversamente proporcional al número de muestras en el conjunto de entrenamiento. No lo aplicamos ya que, como se ve en la Tabla 19, no aporta nada.



Gráfica 1: Número de muestras de entrenamiento por clase en el problema facial

	Sin balancear	Balanceado
***Validation accuracy* (%)**	88.90	88.89

Tabla 19: Impacto del balanceo de clases en el problema facial

3.1.6 Estructuras no secuenciales

A partir de la red final obtenida en el apartado anterior, se han planteado una serie de arquitecturas más complejas para intentar mejorar más los resultados.

- *Cat*

En primer lugar, se han planteado dos arquitecturas no secuenciales inspiradas en el modelo *Inception* de Google [48], que hacen varias convoluciones diferentes en cada paso para después

concatenar los resultados. Como varias convoluciones correlativas de igual tamaño se corresponden con una convolución más grande, se han utilizado siempre filtros 3x3 y lo que se ha variado es el número de convoluciones que se realizan en cada paso.

La primera propuesta la mostramos esquematizada en la Figura 8. Como se puede ver, lo que hacemos es, para cada filtro que se repetía *n* veces (por ejemplo, al principio se hacían 4 convoluciones de 64 filtros 3x3), probamos a repetirlo *n* veces, *n-1* veces, etc. Y concatenamos esos resultados.

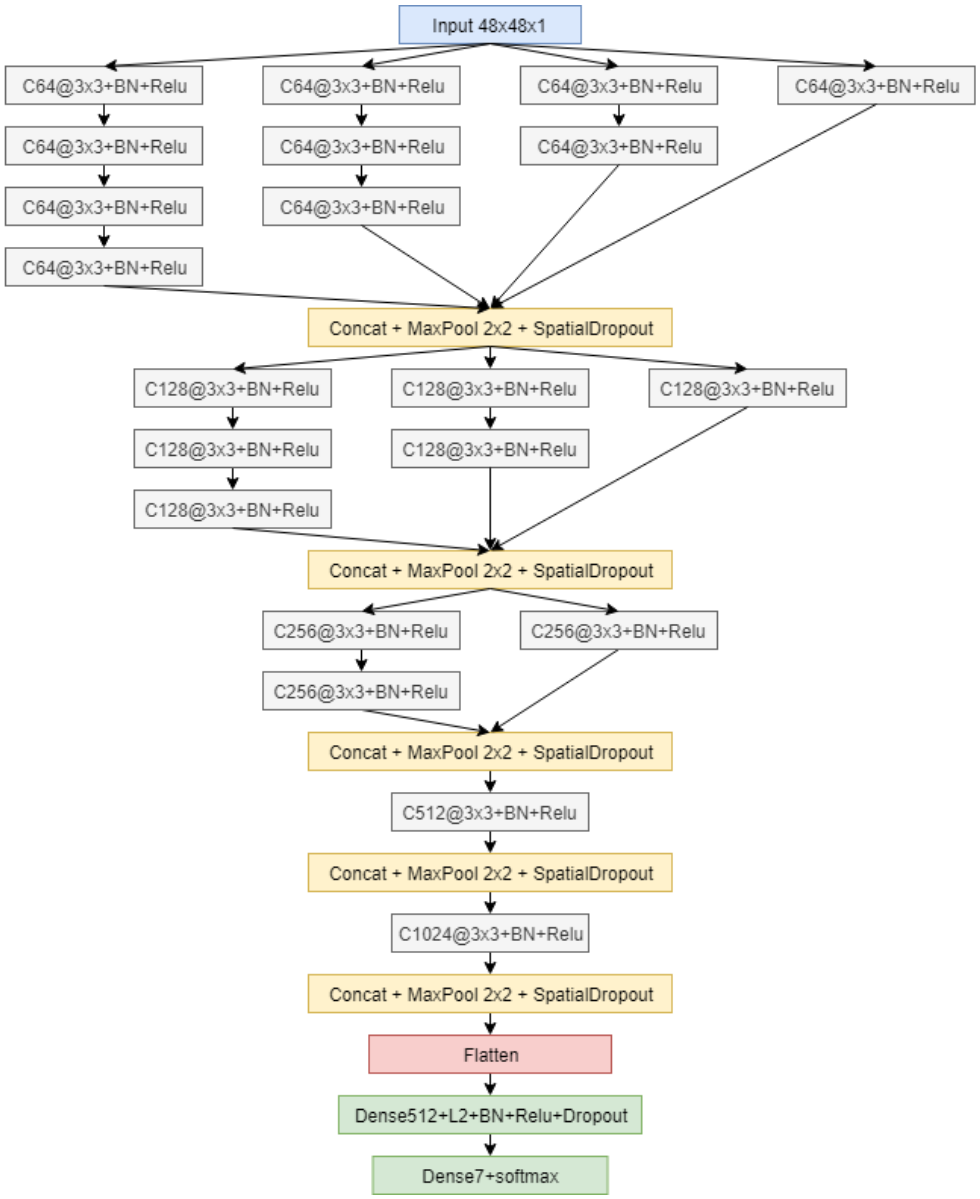


Figura 8: Primera propuesta de CNN tipo *Inception* para el problema facial

La segunda propuesta es muy similar, pero solo realizando la concatenación en la primera etapa (en las convoluciones de 64 filtros). De nuevo, mostramos la red en la Figura 9.

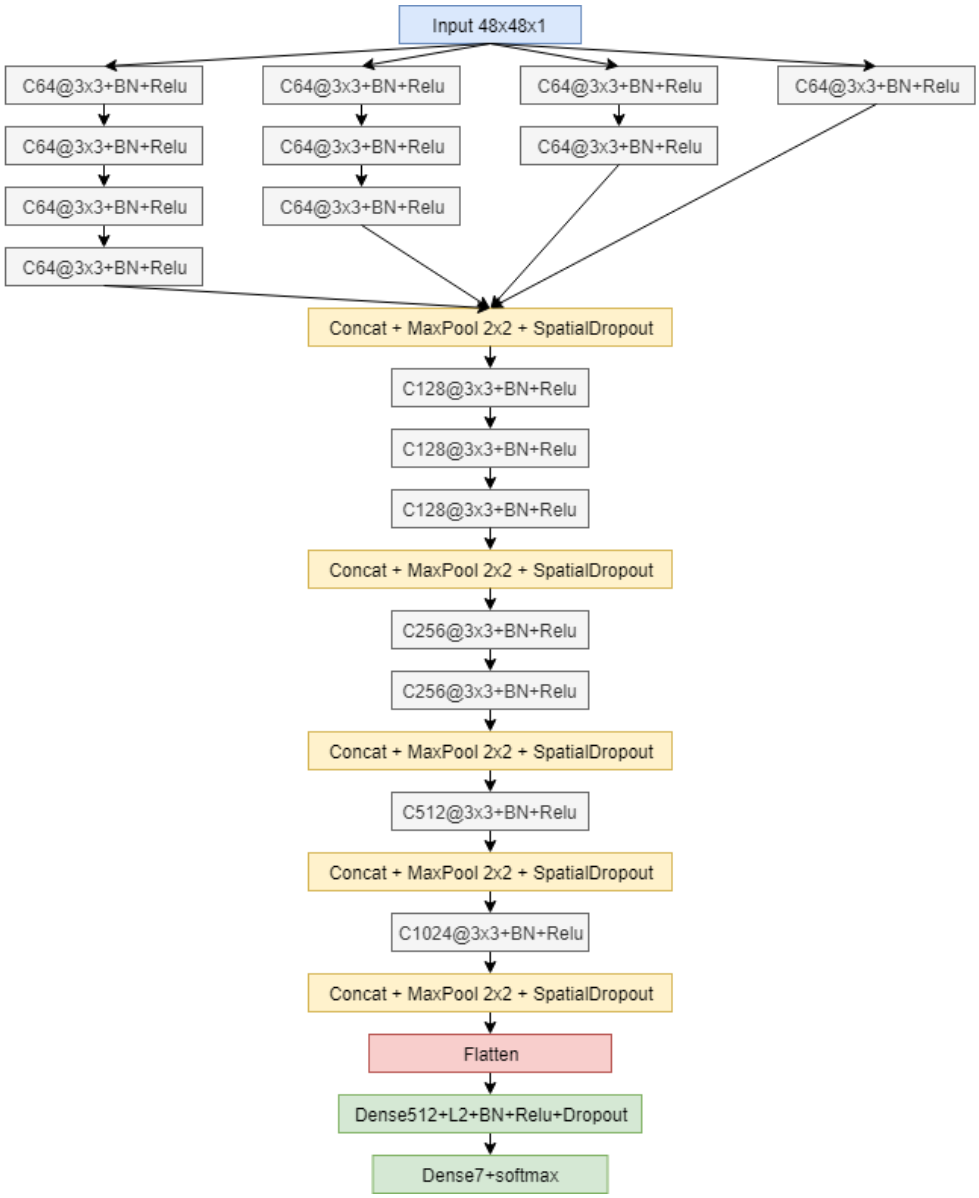


Figura 9: Segunda propuesta de CNN tipo *Inception* para el problema facial

Vemos en los resultados de la Tabla 20 que estas arquitecturas no presentan ninguna mejoría.

	Secuencial	Cat 1	Cat 2
***Validation accuracy* (%)**	88.90	88.71	88.76

Tabla 20: Resultados obtenidos con estructuras tipo *Inception* en el problema facial

- Bilineal

También se han planteado dos redes bilineales. En la primera, lo único que se ha hecho es duplicar la parte convolucional de la red, la entrada se proporciona a las dos primeras capas de 64 filtros 3x3, y al final se hace el producto externo de la salida de las dos capas de 1024. En esta propuesta se elimina la parte *fully connected*, de forma que ese producto se conecta directamente con la capa densa de 7 neuronas de salida. Lo vemos en la Figura 10.

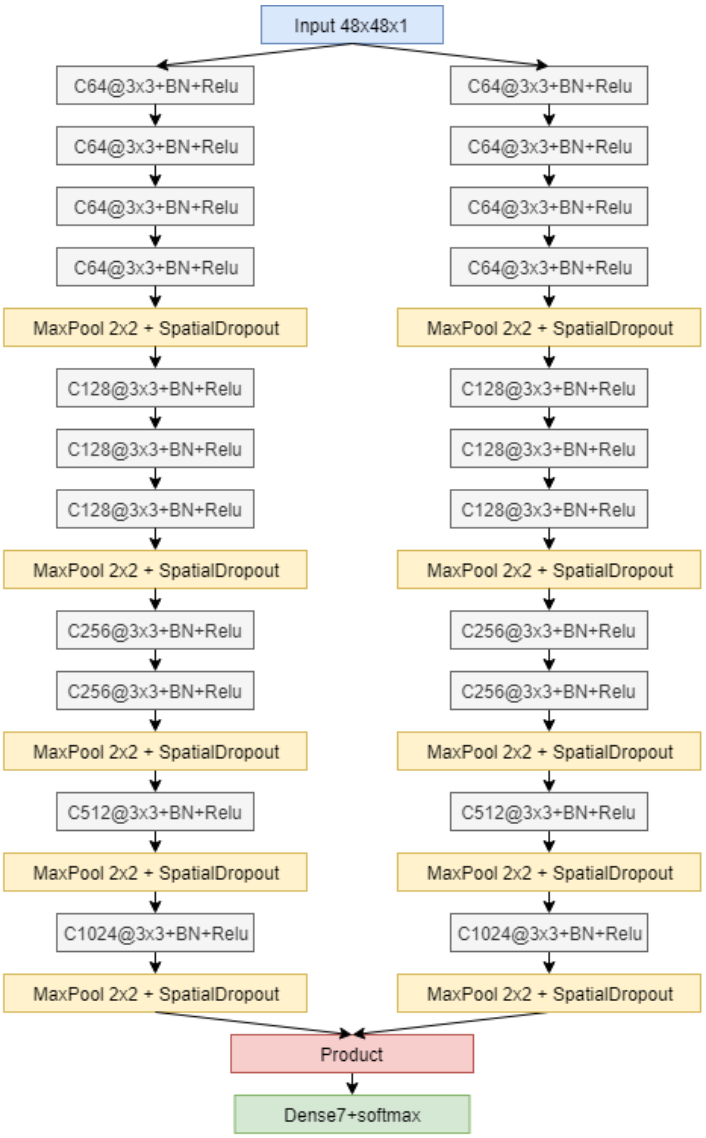


Figura 10: Primera propuesta de red bilineal para el problema facial

Cabe aclarar que el producto externo se ha añadido como una capa personalizada, y la implementación de la misma se ha extraído de un *GitHub* de Roberto Paredes Palacios en el que utiliza redes bilineales en una tarea de clasificación de modelos de coches[10].

La segunda bilineal considerada no tiene nada que ver con nuestro modelo secuencial de partida, sino que es más sencilla y es la red bilineal del *GitHub* de Roberto Paredes Palacios que antes se ha mencionado. La arquitectura se representa en la Figura 11.

[10] https://github.com/RParedesPalacios/ComputerVisionLab/blob/master/Exercises/cars1.py

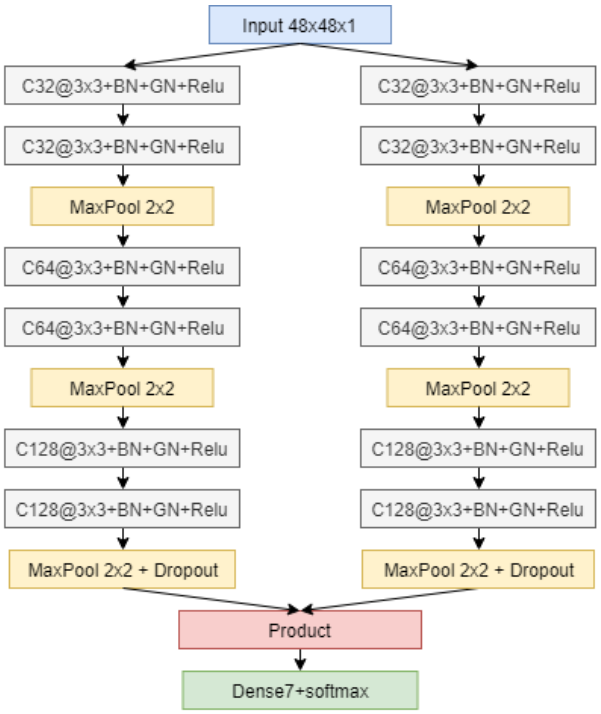


Figura 11: Segunda propuesta de red bilineal para el problema facial

De nuevo, de acuerdo a los datos mostrados en la Tabla 21, no conseguimos una mejora.

	Secuencial	Bilineal 1	Bilineal 2
***Validation accuracy* (%)**	88.90	88.65	86.52

Tabla 21: Resultados obtenidos con redes *bilinear* en el problema facial

- *ResNet*

En tercer lugar, se han planteado dos modelos de *Residual Networks*. El primero de ellos parte de nuestra red secuencial y, al inicio de cada bloque de capas convolucionales del mismo número de filtros, añade una conexión residual al final de dicho bloque utilizando como intermediario una capa convolucional del mismo número de filtros, para que coincidan, pero de tamaño 1x1. Esta arquitectura puede verse en la Figura 12.

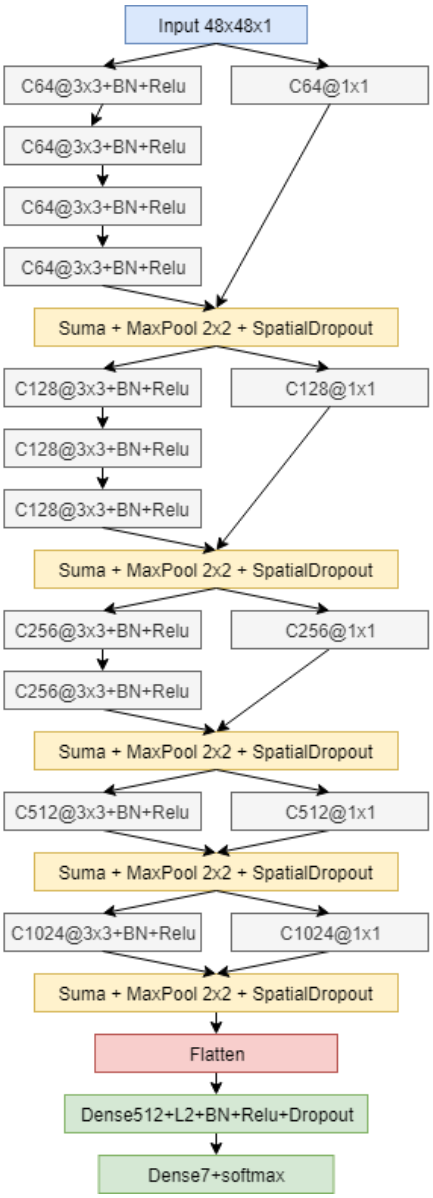


Figura 12: Primera propuesta de *ResNet* para el problema facial

Por otra parte, en una segunda arquitectura, probamos a adaptar el modelo de *ResNet* con *full pre-activation* propuesto en [49], quedando una red como la que se puede ver en la Figura 13.

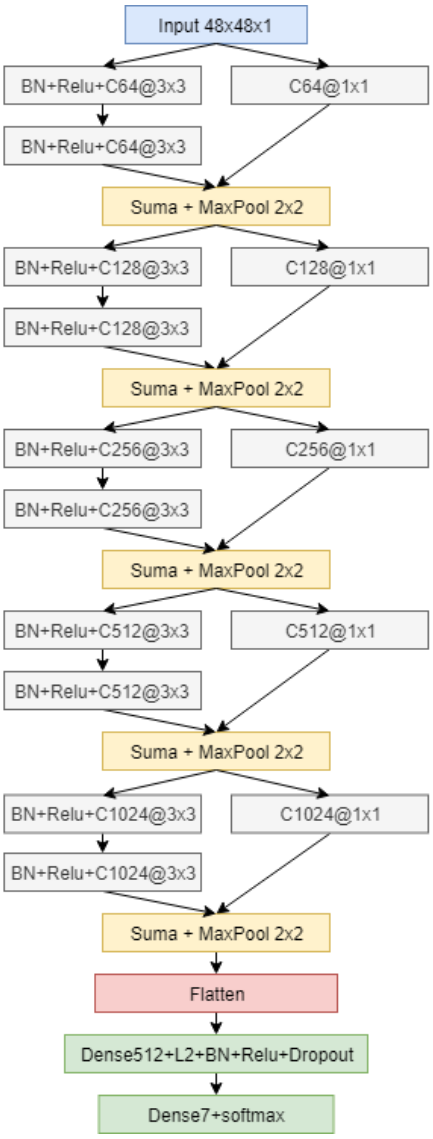


Figura 13: Segunda propuesta de *ResNet* para el problema facial

Los resultados obtenidos con estos dos modelos aparecen en la Tabla 22, donde vemos que así tampoco hemos conseguido superar al modelo secuencial.

	Secuencial	*ResNet* 1	*ResNet* 2
***Validation accuracy* (%)**	88.90	88.72	79.99

Tabla 22: Resultados obtenidos con *ResNets* en el problema facial

- *DenseNet*

Además de los ya vistos, se han planteado 2 modelos de *DenseNet* [50]. El primero consiste en adaptar el modelo *DenseNet* a nuestra red secuencial, sustituyendo cada bloque de capas convolucionales del mismo número de filtros por un bloque densamente conectado de 5 o 4 capas convolucionales del mismo número de filtros. Lo único que no modificamos es la capa final de 1024 filtros porque hacer un bloque denso con ese número de filtros alargaba demasiado el tiempo de cómputo y los requisitos de memoria (motivo por el que también los bloques de 256 y 512 filtros constan de 4 capas y no de 5). Podemos ver esta propuesta en la Figura 14.

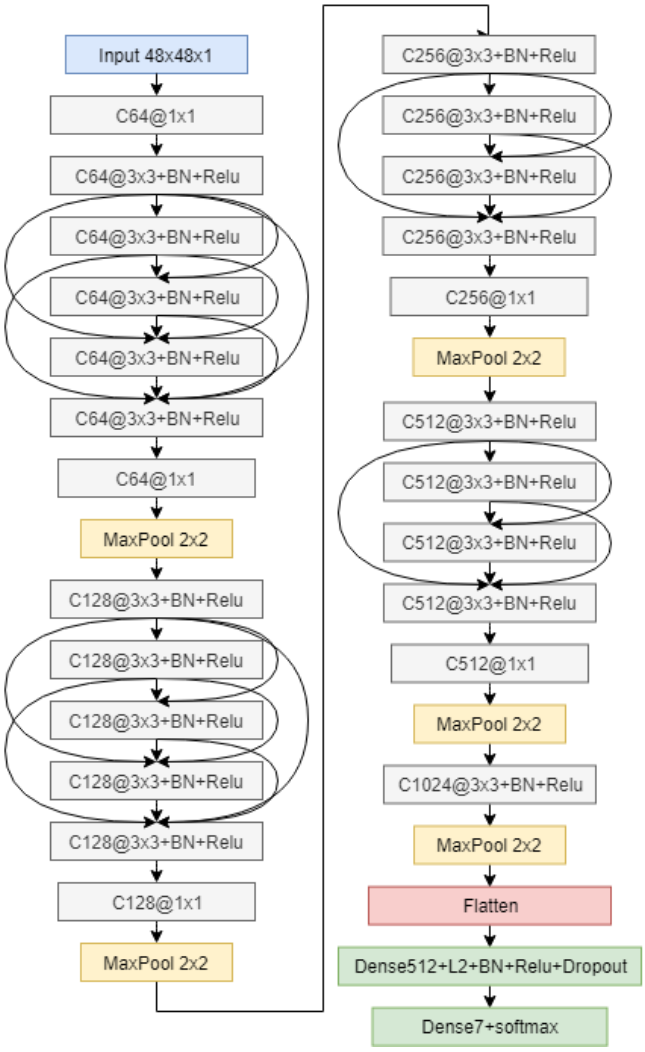


Figura 14: Primera propuesta de *DenseNet* para el problema facial

La segunda arquitectura planteada consiste, de forma similar, en adaptar la versión *bottleneck* de la *DenseNet* a nuestro modelo [50]. Como esta versión implica más capas convolucionales, se ha decidido prescindir del bloque inicial de 64 filtros y de la capa convolucional final de 1024. De esta forma nos queda la arquitectura representada en la Figura 15.

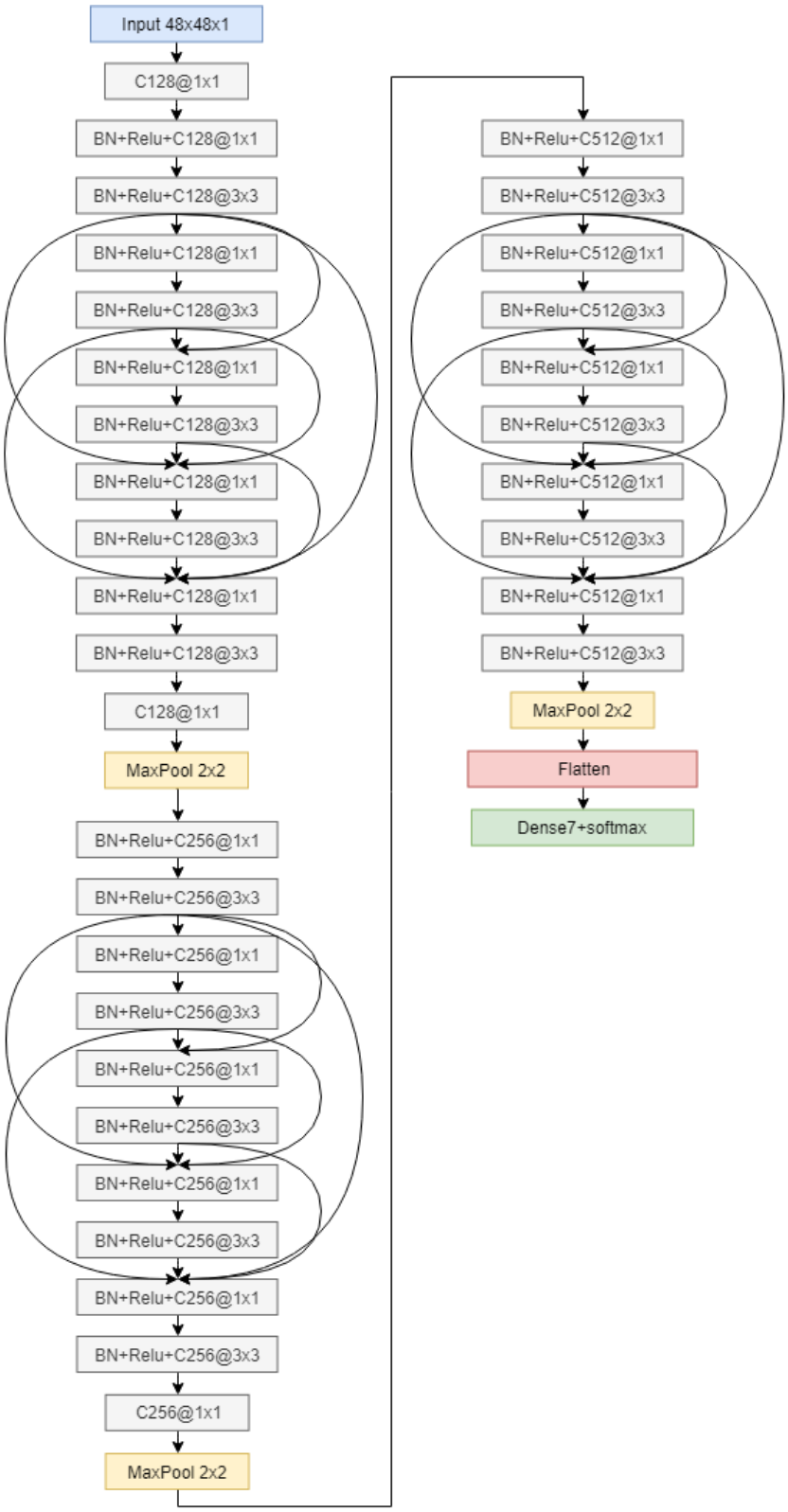


Figura 15: Segunda propuesta de *DenseNet* para el problema facial

	Secuencial	*DenseNet* 1	*DenseNet* 2
***Validation accuracy* (%)**	88.90	88.19	87.25

Tabla 23: Resultados obtenidos con *DenseNet* en el problema facial

De nuevo, vemos en la Tabla 23 que estas arquitecturas no superan al modelo secuencial.

En conclusión, ninguno de estos modelos más complejos nos ha dado mejores resultados que nuestra red secuencial. Esto puede deberse a que este tipo de arquitecturas están diseñadas para problemas más grandes, con muchas más clases que solo siete. Por este mismo motivo, no probaremos estructuras similares para los problemas acústico y textual.

3.1.7 Reducción de parámetros

Por último, se ha intentado reducir el número de parámetros de la red obtenida, con el fin de que sea más ligera y rápida, teniendo en cuenta que el objetivo es poder usarla en directo. Por ello, lo primero que se ha hecho es probar los dos modelos predefinidos de *MobileNet* disponible en *Keras*, obteniendo los porcentajes de acierto que aparecen en la Tabla 24.

	Secuencial	MobileNet 1	MobileNet 2
***Validation accuracy* (%)**	88.90	82.50	80.60

Tabla 24: Resultados obtenidos con *MobileNet* en el problema facial

Como estos resultados son muy malos, probamos a reducir el número de parámetros de nuestra red, para ello proponemos 7 modificaciones del modelo secuencial:

1. Reducir el número de filtros de la última capa convolucional de 1024 a 512.
2. Reducir el número de filtros de la última capa convolucional de 1024 a 512 y los de la penúltima capa de 512 a 256.
3. Reducir el número de filtros de la última capa convolucional de 1024 a 512 y el número de neuronas de la capa *fully connected* de 512 a 256.
4. Reducir el número de filtros de la última capa convolucional de 1024 a 512, los de la penúltima capa de 512 a 256 y el número de neuronas de la capa *fully connected* de 512 a 256.
5. Reducir el número de filtros de las capas de 256 a 128, los de la penúltima capa convolucional de 512 a 258 y el número de neuronas de la capa *fully connected* de 512 a 256.
6. Reducir el número de filtros de las capas de 256 a 128, los de la penúltima capa convolucional de 512 a 258, los de la última capa convolucional de 1024 a 512 y el número de neuronas de la capa *fully connected* de 512 a 256.
7. Reducir el número de filtros de las capas de 256 a 128, los de la penúltima capa convolucional de 512 a 258, los de la última capa convolucional de 1024 a 256 y el número de neuronas de la capa *fully connected* de 512 a 256.

Con estas alternativas obtenemos los resultados de la Tabla 25. Con la alternativa 4, no solo reducimos el número de parámetros de 7.806.663 a 5.048.519, sino que además obtenemos una ligera mejoría en los resultados. Con todo, nuestro modelo final sería como el que se representa en la Figura 16.

	Original	Alt. 1	Alt. 2	Alt. 3	Alt. 4	Alt. 5	Alt. 6	Alt. 7
***Validation accuracy* (%)**	88.90	88.75	88.67	89.04	88.81	88.80	88.77	88.54

Tabla 25: Resultados obtenidos al reducir los parámetros de la CNN del problema facial

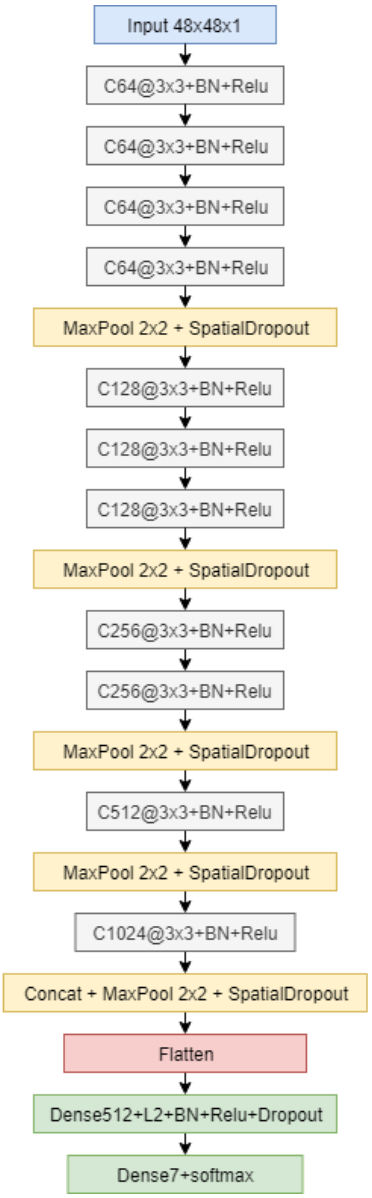


Figura 16: Modelo de CNN final para el problema facial

3.1.8 Test

Con el modelo final obtenido en el apartado anterior y representado en la Figura 16, entrenamos con los conjuntos de entrenamiento y validación, evaluando sobre el conjunto de test. Obtenemos así un porcentaje de aciertos del 87.65%, con un intervalo de confianza al 95% del 0.82%.

A continuación, desglosamos estos resultados ofreciendo el porcentaje de aciertos por corpus con el intervalo de confianza al 95% y la matriz de confusión.

Corpus	CK	FER	JAFFE	EDFFE	Yale	KDEF
***Test accuracy* (%)**	95.12±6.59	85.86±1.12	89.66±11.08	89.55±1.37	100.00	91.33±2.71

Tabla 26: Porcentaje de acierto por corpus en el problema facial

Como podemos ver en la Tabla 26, el corpus FER, que es el más extenso y que se compone exclusivamente de imágenes no posadas, es el que peor resultados da, como sería esperable.

Por otra parte, los corpus compuestos de imágenes posadas, como el Cohn-Kanade o el de Yale, ofrecen muy buenos resultados. Cabe resaltar que el resultado excepcional obtenido con el corpus de Yale se debe a que son fotografías posadas y a que todas son de emoción neutra, que es la emoción que cuenta con más muestras en el corpus (16105 de un total de 41246). También valdría la pena comentar que, aunque el JAFFE está formado por fotografías posadas, obtiene unos resultados comparables al EDFFE. Esto muy probablemente se debe a que el JAFFE contiene exclusivamente imágenes de mujeres japonesas, cuando la mayor parte de nuestro conjunto está formado por fotografías de gente caucásica.

Comparándolo con el estado del arte, vemos que, en corpus posados o sesgados, como el Cohn-Kanade o el JAFFE, es relativamente fácil encontrar artículos que obtienen mejores resultados [51]. Sin embargo, hay que tener en cuenta que esos sistemas son mucho más limitados, ya que no necesitan aprender invarianza a algunas características importantes para un clasificador robusto y universal, ni han aprendido a tratar emociones espontáneas, que siempre son más difíciles de captar. Por ello, esos resultados no son comparables y esos sistemas probablemente no sean extrapolables a un entorno realista. Resultados más relevantes son los que podamos encontrar para corpus más grandes y variados, como el FER, que recordemos que es el más grande de nuestro conjunto de *datasets* y que consta de imágenes recopiladas automáticamente de la red, sin sesgo ni selección. En este sentido sí que es significativo comprobar que obtenemos mejores resultados que otros estudios [47], que además utilizan también la versión del FER reetiquetada por Microsoft, FER+.

A continuación, en la Tabla 27, mostramos la matriz de confusión, donde las columnas se corresponden con la emoción predicha y las filas con la real.

Matriz de confusión	Enfado	Asco	Miedo	Felicidad	Tristeza	Sorpresa	Neutro
Enfado	77.54	1.89	1.65	3.78	0.24	3.07	11.82
Asco	9.70	72.39	1.49	2.24	2.24	2.24	9.70
Miedo	3.55	0.00	69.23	2.37	2.96	16.57	5.33
Felicidad	0.57	0.05	0.05	94.16	0.05	0.78	4.34
Tristeza	5.92	0.42	2.54	4.65	55.91	1.27	29.39
Sorpresa	2.06	0.00	1.58	3.32	0.32	86.71	6.01
Neutro	1.69	0.04	0.12	3.14	1.24	0.95	92.82

Tabla 27: Matriz de confusión para el test del problema facial

Llama la atención el mal resultado en la detección de la tristeza, aunque quizás pueda deberse a que podría ser la emoción más sutil y menos exagerada de las que estamos considerando, ya que de hecho se confunde muchísimo con neutro. El otro dato que llama la atención es el alto porcentaje de confusión de sorpresa con miedo, aunque esto resulta totalmente esperable, porque son dos emociones que pueden ir muy relacionadas y cuyas reacciones pueden ser muy similares.

3.2 Audio

3.2.1 Corpus

Hemos querido elaborar un sistema de reconocimiento multimodal de emociones para individuos españoles. Como ya se ha mencionado, es importante entender que estos sistemas deben diseñarse de forma local, ya que, si bien las expresiones faciales son por lo general universales, no sucede lo mismo con la forma en que expresamos emociones de forma oral. El idioma y la cultura condicionan nuestra forma de expresarnos, de forma que una tonalidad que puede sonarnos a felicidad, es posible que una persona de la India o de Polonia no la reconozca correctamente, ya que no es el tipo de entonación con la que ellos denotan esa emoción. Esto conlleva una limitación a los reconocedores de este estilo, pero también una oportunidad para nosotros de innovación, ya que la gran mayoría de estos trabajos utilizan *datasets* en inglés, que no serían tan fiables con individuos no angloparlantes.

El problema que esto acarrea es que necesitamos disponer de corpus de audio emocional en castellano, los cuales, igual que los estudios que planteen el problema desde esta perspectiva, son escasos. El único que hemos encontrado es el *INTER1SP Spanish Emotional Database*[11]. Este es un *dataset* de la Universitat Politècnica de Catalunya que contiene grabaciones en una habitación sin ruido de dos actores, un hombre y una mujer. Se dispone de 3036 oraciones pronunciadas por la mujer (grabaciones) y 3005 del hombre. Están clasificados en once categorías: enfado, asco, miedo, alegría, tristeza, sorpresa, neutral alto, neutral suave, neutral rápido, neutral lento y neutral normal. Es decir, nuestras categorías, pero subdividiendo la neutral en cinco, por lo que hemos unificado los audios de esas cinco categorías en una sola.

3.2.2 Extracción de características

A la hora de utilizar redes neuronales para la clasificación de audio o texto la opción más obvia quizás sea utilizar redes recurrentes, ya que están pensadas para trabajar con secuencias con una relación temporal. Sin embargo, este modelo tiene el inconveniente de que son bastante lentas, lo cual puede ser un problema tanto para su entrenamiento, como para su uso en una aplicación en tiempo real, que es el objetivo de este trabajo. Es por ello que en su lugar vamos a utilizar redes convolucionales, ya que dan buenos resultados a la hora de encontrar relaciones espaciales y sobre todo porque son muy rápidas.

Por supuesto, el problema que introduce el hecho de usar redes convolucionales es que hay que representar los archivos de audio como imágenes. Para ello, se ha decidido extraer las siguientes características de cada audio:

- Representación en el dominio temporal de la señal acústica.
- *Mel Frequency Cepstral Coefficients* (MFCC): es una característica muy habitual en el área de reconocimiento de audio que se basa en la percepción humana auditiva, de modo que elimina todos los elementos superfluos de la señal acústica, como el ruido de fondo.
- *Log Filterbank Energies*: es un espectrograma que se obtiene al aplicar bancos de filtros al periodograma de la señal. Son un paso intermedio en la obtención de los MFCC. Por eso, presentan una correlación mayor que los MFCC, pero en algunos casos retienen una mayor cantidad de información de la señal original[12]. [52]

[11] http://catalog.elra.info/en-us/repository/browse/ELRA-S0329/
[12] https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

- *Spectral Subband Centroid*: se obtiene dividiendo la banda de frecuencia en un número fijo de sub-bandas y calculando el centroide de cada una empleando el espectro de potencia de la señal. Se ha comprobado que su uso junto a otras características, como los MFCCs mejoran en general el desempeño de sistemas de reconocimiento del habla [53]. Además, suplen los malos resultados que se obtienen con los MFCC en condiciones de ruido aditivo [54].

Estas características corresponden a matrices que pueden representarse como imágenes. Con los parámetros por defecto de la librería *python_speech_features*[13], que es la que hemos utilizado para la extracción de estas características, estas matrices tienen un número de columnas común dependiente de la longitud del audio, y un número de filas independiente del audio, pero diferente para cada característica. En concreto, los MFCC tienen 13 filas, los *Log Filterbank Energies,* 26, y los *Spectral Subband Centroid,* 26 también.

Por ello, la extracción de características consiste en, para cada audio:

1. Obtener las matrices mencionadas.
2. Llevarlas todas al rango 0-1.
3. Obtener como imagen la representación en el dominio temporal de la señal acústica.
4. Redimensionar la anchura de dicha imagen al número de columnas de las matrices y su altura a 128.
5. Utilizar esas cuatro imágenes normalizadas como entrada a una red neuronal con múltiples entradas.

Así, obtenemos unas imágenes de entrada como las que se pueden observar en la Figura 17.

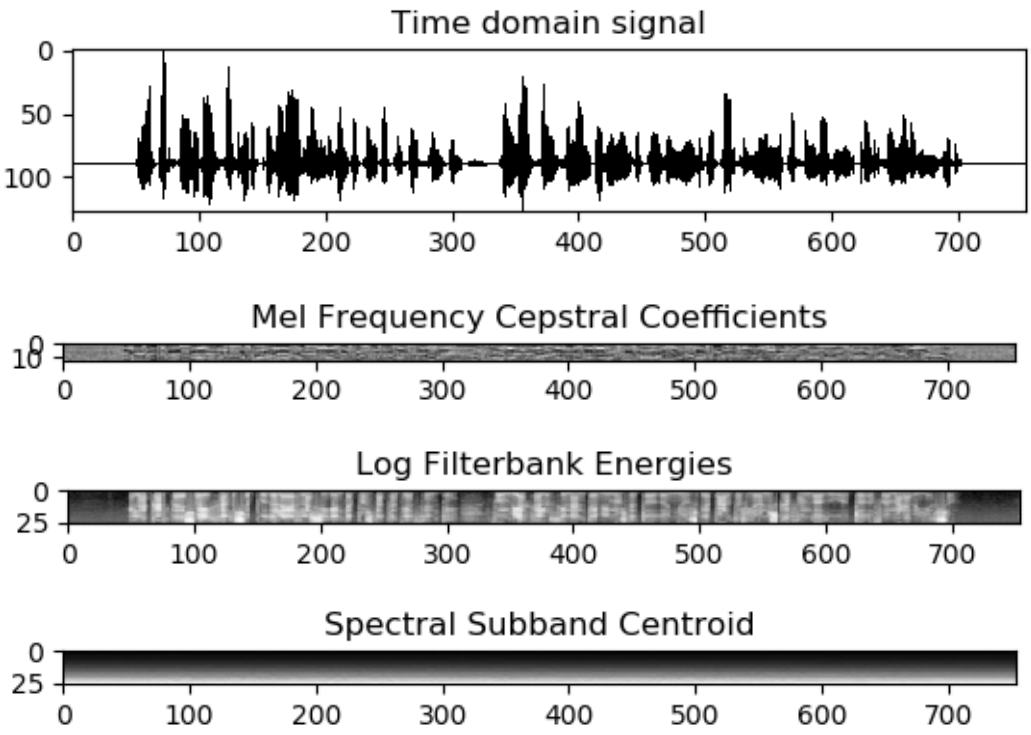
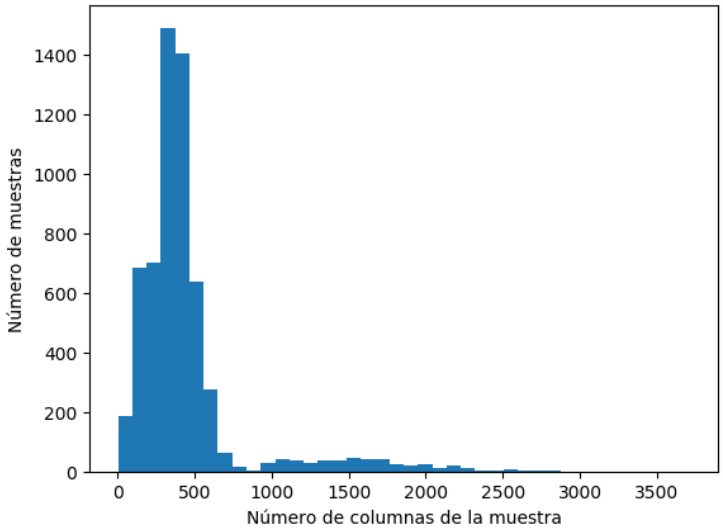


Figura 17: Ejemplo de las características extraídas de un archivo de audio

Un problema añadido al tratamiento del audio, que ya se ha esbozado, es que, para cada archivo de audio, la representación correspondiente tiene una dimensionalidad diferente (mismo alto

[13] https://python-speech-features.readthedocs.io/en/latest/

siempre, pero distinto ancho). Además, como se muestra en la Gráfica 2, esta variabilidad es demasiado alta como para considerar un *padding* básico, ya que hay muestras para las que obtenemos características con 50 columnas, y otras que tienen 2500. Por ello, la opción que se ha considerado es cortar estas imágenes en fragmentos de un tamaño fijo y a los sobrantes les aplicamos *padding*. Por ejemplo, si cortáramos por trozos de 200 píxeles, una imagen de 900 se descompondría en 5, la última de las cuales sería rellenada con ceros hasta alcanzar los 200 píxeles. El mejor tamaño de partición es uno de los parámetros a estimar.



Gráfica 2: Histograma de dimensionalidad de las muestras de audio

3.2.3 Modelo inicial

Para el modelo de partida, nos hemos inspirado en la CNN propuesta en [55]. En [55] se plantea un problema de reconocimiento de emociones a partir de los MFCC obtenidos del audio, por lo que el modelo utilizado debería sernos útil, al menos como primer planteamiento. Lo que se propone es una red secuencial sencilla que consta de una capa convolucional de 20 filtros 5x5 con la función de activación 'relu', seguida de *max pooling* y *flatten*, después añaden una capa densa de 1000 neuronas con una 'relu' y finalmente una capa de salida con 7 neuronas y 'softmax' como función de activación.

Esta red ha sido ligeramente adaptada para intentar mejorarla inicialmente y para adecuarse a nuestro planteamiento. Por ello, como se muestra en la representación de la Figura 18, se han hecho las siguientes modificaciones:

1. La red tiene cuatro entradas, como ya se ha descrito, cada una de ellas pasa por su propia capa convolucional y su capa densa, después se concatenan esas salidas y se conecta a una única capa densa de salida.
2. La capa convolucional tiene 32 filtros para poder capturar más información del audio de entrada.
3. La capa densa posterior a la convolucional tiene 32 neuronas para aligerar la red, ya que 1000 neuronas tras tan solo 32 filtros quizás sea demasiado (más teniendo cuatro de ellas, una para cada característica). Además, capas muy grandes permiten a las redes neuronales memorizar mucha información, por lo que tendríamos *overfitting* fácilmente.

4. Tras todas las capas, excepto la de salida, se han introducido capas de regularización. Esto es necesario porque, al no disponer de muchas muestras, es fácil tener *overfitting* rápidamente. Se ha añadido *batch normalization* y ruido gaussiano tras todas ellas, además de *spatial dropout* tras las convolucionales, dado su buen funcionamiento en la tarea facial.

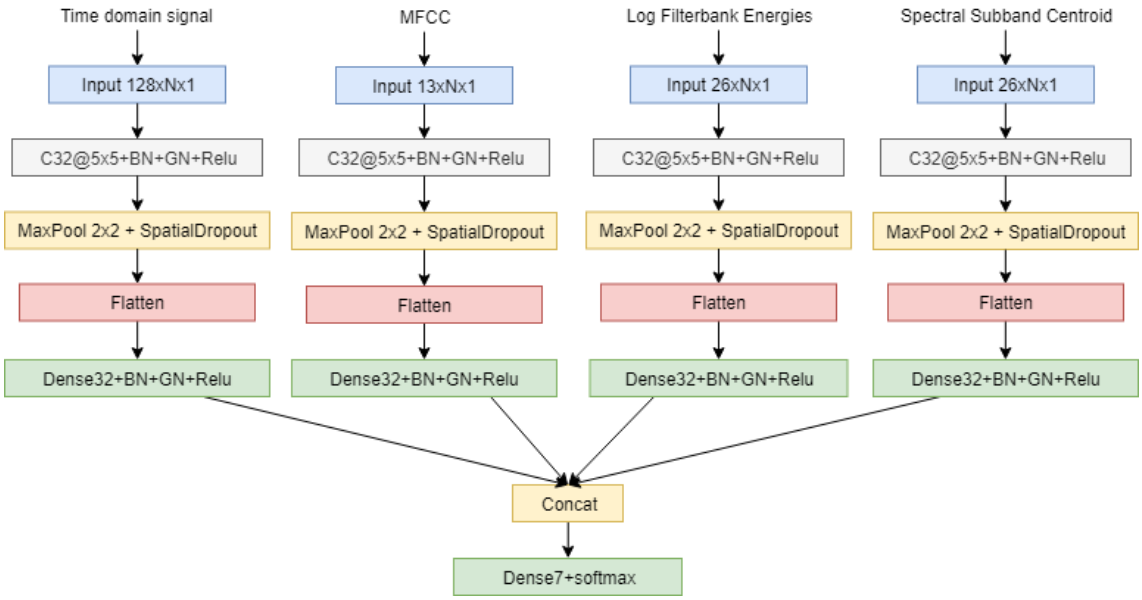


Figura 18: CNN inicial para la tarea acústica

Además, como en el problema anterior, la función de pérdida que se ha elegido es el *categorical crossentropy* y el optimizador utilizado es *adam*.

Por otra parte, para calcular el porcentaje de aciertos no podemos utilizar simplemente el *categorical accuracy*, ya que, como tenemos las muestras divididas en pedazos, no es una medida adecuada de acierto saber qué porcentaje de trozos acertamos, sino cuántos audios acertamos. Por ello, para determinar si hemos acertado o no con un audio del conjunto de validación o de test, lo que hacemos es evaluar todos sus fragmentos, sumar la salida que nos da la red para cada uno de ellos y quedarnos con el argumento máximo de la suma como clase predicha. Así es como comprobamos si una muestra está bien o mal clasificada y es como calculamos el *accuracy* de la CNN.

Además, empezamos con un tamaño de *batch* de 100 y 75 *epochs*. De forma similar a como hicimos con las caras, los resultados de *accuracy* que tenemos en cuenta (calculándolo como se ha descrito), se corresponden con la media de *accuracy* de los últimos 20 *epochs*, para eliminar el componente "suerte" (en el problema facial eran los últimos 30 porque se hacían muchos más *epochs*).

Como en el problema facial, hemos particionado el corpus de archivos de audio en un 70% para entrenamiento, un 15% para validación y el 15% restante para test.

3.2.4 Preproceso

Una vez tenemos el modelo inicial, hemos tenido que estimar un par de parámetros relacionados con los datos. Lo primero ha sido, como ya se ha adelantado, elegir el tamaño de troceado de los audios, es decir, el ancho que tienen las características que sirven como entrada para la red neuronal. Se han considerado tamaños de hasta 500 píxeles, ya que aproximadamente un 80% de los datos tienen un tamaño menor o igual, lo que lleva a que, con

particionados de mayor tamaño, más de un 32% de la información resultante serían ceros. Los resultados obtenidos con la red inicial y los diferentes tamaños de entrada son los que se ven en la Tabla 28.

Ancho (píxels)	100	150	200	250	300	350	400	450	500
***Validation accuracy* (%)**	54.96	53.29	65.83	63.40	51.85	64.65	61.93	51.42	64.21

Tabla 28: Efecto sobre el problema acústico del tamaño de partición de los datos

Escogemos por tanto dividir las características de cada audio en subimágenes de 200 píxeles de audio.

Al dividir las muestras de este modo, nos damos cuenta de que, a veces, las primeras y últimas divisiones se componen en gran medida de silencio del inicio o el final de la grabación, por lo que es imposible que la red pueda sacar algo de información y clasificarlo correctamente. Para resolver este problema, se han propuesto y evaluado dos soluciones:

1. La primera cosiste en, antes de trocear un audio, recortarlo, eliminando los silencios iniciales y finales, de forma que toda la información que quede sea sonido efectivo.
2. La segunda consiste en eliminar una división es enteramente silencio. Como el trozo inicial nunca se queda completamente en silencio, ya que no suele haber silencios tan largos, esto consiste en descartar el trozo sobrante final, que es de menor tamaño, si no contiene audio de verdad.

Como estas dos soluciones podrían estar relacionadas con el tamaño de división óptimo, hemos vuelto a probar con las particiones de la prueba de arriba. Primero mostramos los resultados de la primera solución en la Tabla 29.

Ancho (píxels)	100	150	200	250	300	350	400	450	500
***Validation accuracy* (%)**	54.15	56.55	53.66	52.86	58.03	54.53	54.55	60.15	57.79

Tabla 29: Resultados obtenidos al aplicar un preproceso a los datos del problema acústico

Vemos que, pese a plantearlo como una mejora de forma lógica, en la mayoría de los casos empeora los resultados. Esto se debe muy probablemente a que de esta forma se elimina información contextual sobre sonidos que se corresponden al inicio de una frase (sonido después de silencio) o al final de una frase (silencio después de un sonido). Lo cual es información que puede ser relevante. Por eso precisamente se propuso la segunda solución, más contenida, cuyos resultados se muestran en la Tabla 30.

Ancho (píxels)	100	150	200	250	300	350	400	450	500
***Validation accuracy* (%)**	55.65	55.33	68.36	68.12	53.08	65.82	63.20	54.27	64.62

Tabla 30: Efecto de un preproceso más refinado sobre los datos del problema acústico

En este caso sí que vemos una mejoría, proporcional a los resultados obtenidos inicialmente. Por tanto, optamos por esta solución y por 200 como tamaño de las imágenes de entrada.

3.2.5 Estimación de la red

- *Overfitting*

Con esas pruebas preliminares, nos damos cuenta de que tenemos un problema grave de *overfitting*, ya que en entrenamiento se alcanza casi el 100% de aciertos. Por ello, lo primero que se ha hecho es añadir técnicas de *data augmentation.*

Para ir comprobando cómo mejoramos este problema, se añadirá a las tablas el porcentaje de acierto en entrenamiento. Este porcentaje se calcula de la forma normal, es decir, es el porcentaje de trozos que la red clasifica correctamente. No como con el de validación, que como ya hemos explicado, nos interesa saber cuántos archivos de audio completos clasifica bien, sumando los resultados de los pedazos. Esto lo hacemos así porque a la hora de entrenar la red nos interesa que aprenda a clasificar bien incluso los trozos aislados. Por otra parte, la precisión real del clasificador (que se evalúa en validación) se corresponde con el número de audios bien clasificados.

En este caso es más difícil intuir qué transformaciones podrían beneficiarnos sin estropear las imágenes, ya que no son fotografías, sino representaciones. Por ello, probaremos bastantes de ellas para ver cuáles pueden ser eficaces con este tipo de datos. Lo primero que probamos es a añadir volteos horizontales y verticales, obteniendo los resultados de la Tabla 31.

	Sin volteo	Horizontal	Horizontal + Vertical
***Validation accuracy* (%)**	68.36	76.62	74.32
Training accuracy (%)	99.35	98.20	92.92

Tabla 31: Resultados obtenidos al aplicar volteos a los datos del problema acústico

Como vemos, realizar inversiones horizontales proporciona una gran mejora. Esto puede entenderse si pensamos que al reproducir un audio hacia atrás podríamos igualmente discernir la emoción en él. Añadir volteos verticales empeora los resultados, así que no lo incorporamos.

También probamos a añadir *shear* con diferentes ángulos, como con el problema facial. Pero, como vemos en la Tabla 32, empeora tanto los resultados que no hemos probado con más valores.

Rango (°)	Sin *shear*	5	10
***Validation accuracy* (%)**	76.62	59.51	58.07
Training accuracy (%)	98.20	75.02	75.02

Tabla 32: Resultados obtenidos al aplicar *shear* a los datos del problema acústico

Lo siguiente que se ha intentado es añadir desplazamientos horizontales. Vemos en la Tabla 33 una gran mejoría al incorporar desplazamientos de hasta un 15%, así que en adelante entrenamos con ello.

Porcentaje (%)	Sin desplazamiento	10	15	20	25	30
***Validation accuracy* (%)**	76.62	89.96	90.61	90.17	89.38	89.58
Training accuracy (%)	98.20	82.65	82.05	80.23	78.55	77.11

Tabla 33: Efecto al aplicar desplazamientos horizontales en el problema acústico

Por último, aunque ya no nos encontramos en una situación de *overfitting*, probamos a añadir *zooms* al *data augmentation*, obteniendo resultados negativos, así que lo descartamos, como se ve en la Tabla 34.

Porcentaje (%)	Sin *zoom*	10	20
***Validation accuracy* (%)**	90.61	81.24	78.00
Training accuracy (%)	82.05	69.56	67.78

Tabla 34: Resultados obtenidos al aplicar *zoom* a los datos del problema acústico

- Características

Hasta ahora hemos entrenado con las cuatro características extraídas de los audios, como se indica en el apartado "Preproceso" de este mismo capítulo. Sin embargo, no hemos comprobado a priori si todas estas características son relevantes en la clasificación. Es por ello que lo siguiente que se ha probado son todas las posibles combinaciones de características. Para hacer más sencilla la tabla de resultados vamos a considerar la representación en el dominio del tiempo la característica 1, los *mel frequency cepstral coefficients* la característica 2, los *log filterbank energies* la característica 3 y los *spectral subband centroid* la característica 4.

Características	1	2	3	4	1,2	1,3	1,4	2,3
***Validation accuracy* (%)**	42.52	74.97	88.82	66.47	78.39	89.67	63.21	91.05
Características	**2,4**	**3,4**	**1,2,3**	**1,2,4**	**1,3,4**	**2,3,4**	**1,2,3,4**	
***Validation accuracy* (%)**	76.93	89.67	89.26	77.60	89.46	90.58	90.61	

Tabla 35: Acierto en función de las características utilizadas en el problema auditivo

Como vemos en la Tabla 35, el mejor resultado viene de utilizar únicamente los *mel frequency cepstral coefficients* y los *log filterbank energies.* Esto mejora los resultados y aligera la red, lo cual es muy positivo a la hora de intentar hacer predicción en tiempo real. Lo más probable es que las otras dos características ofrezcan información redundante o no representen gráficamente la información emocional de una forma eficaz. Nos queda así una red reducida como la que se ve en la Figura 19.

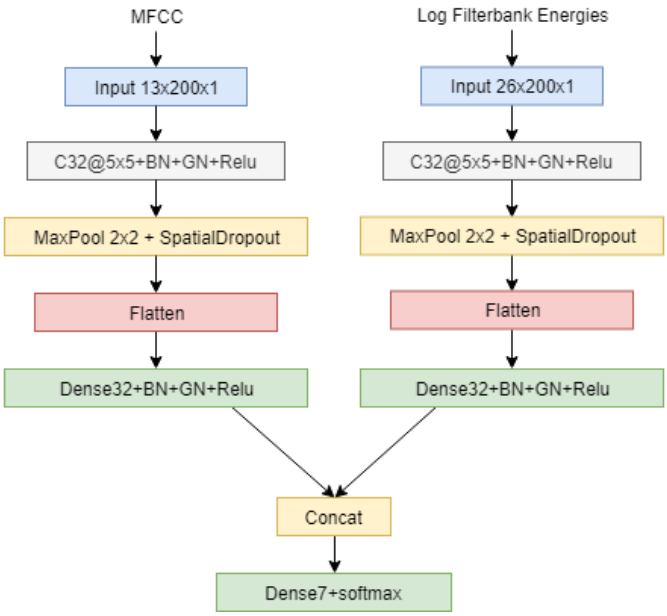


Figura 19: Red convolucional para audio con solo dos características de entrada

- Parte *fully connected*

Una vez resuelto el problema del *overfitting*, lo ideal es probar a ampliar la capacidad de la red. Es por ello que las siguientes pruebas que se han realizado tienen por objetivo estimar el mejor tamaño de las capas densas previas al concatenado. A raíz de los resultados, representados en la Tabla 36 dejaremos las capas densas con 256 neuronas cada una.

Número de neuronas	32	64	128	256	512
***Validation accuracy* (%)**	91.05	92.46	93.07	93.43	93.32

Tabla 36: Efecto del número de neuronas de la capa densa en el problema acústico

- Parte convolucional

Para seguir ampliando la red, se han probado diferentes números de filtros para las capas convolucionales, obteniendo los resultados de la Tabla 37. Por ello, nos quedamos con 128 filtros en cada una de las dos capas convolucionales.

Número de filtros	32	64	128
***Validation accuracy* (%)**	93.43	94.81	95.22

Tabla 37: Efecto del número de filtros de la capa convolucional en el problema acústico

Además, hemos probado a aumentar el tamaño de parte convolucional, duplicándola como se puede ver en la Figura 20. Para el número de filtros, se ha probado a tener 64 en las primeras capas convolucionales y 128 en las segundas, o 128 en las primeras y 256 en las segundas. Un esquema de esta estructura se puede observar en la Figura 20, y los resultados obtenidos en la Tabla 38.

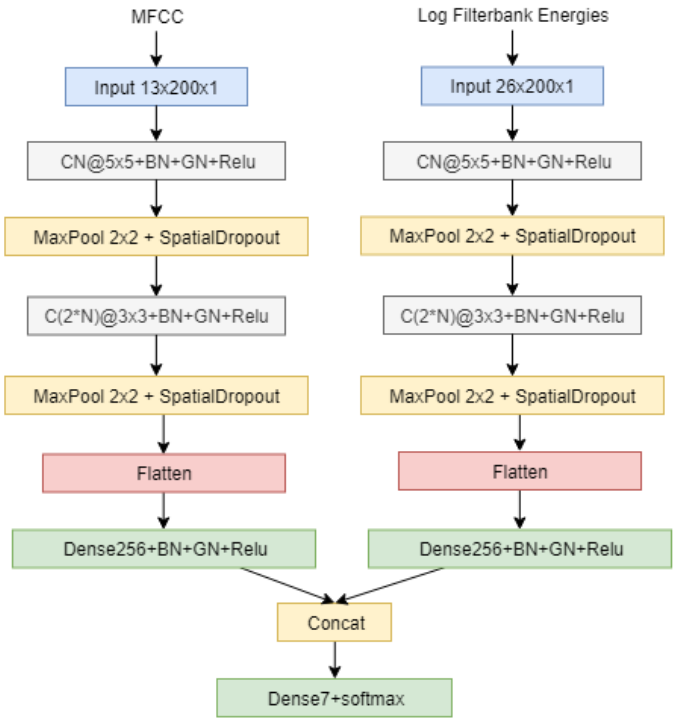


Figura 20: CNN ampliada duplicando la parte convolucional en el problema acústico

Esquema	Estructura simple	Nueva estructura	
Número de filtros	128	64+128	128+256
***Validation accuracy* (%)**	95.22	95.86	96.20

Tabla 38: Resultados tras probar la nueva CNN ampliada en el problema acústico

A la vista de los resultados, conservamos en adelante esta nueva estructura con 128 filtros en la primera capa convolucional y 256 en la segunda, para cada una de las dos entradas en cuestión.

- Ruido gaussiano

El ruido gaussiano lo introdujimos al modelo propuesto en [55] como una forma de regularizar ligeramente la red. Sin embargo, la probabilidad fue arbitrariamente establecida en 0,3. Es por ello que lo siguiente que se hizo fue estimar el mejor valor, así como probar a eliminarlo de la parte convolucional, de la parte *fully connected* y de toda la red, ya que puede que su efecto sea negativo llegados a este punto, en el que ya no tenemos problemas de *overfitting*.

	En toda la red			En convoluciones	En densas	-
Porcentaje	0.1	0.3	0.5	0.3	0.3	0.0
***Validation accuracy* (%)**	97.25	96.20	94.58	97.17	97.26	97.51

Tabla 39: Resultados obtenidos al aplicar ruido gaussiano en la CNN del problema acústico

A raíz de los resultados mostrados en la Tabla 39, se ha tomado la decisión de eliminar el ruido de la red. Además, esto no supone un problema ya que el *training accuracy* se mantiene aun así en un 95.65%, que sigue siendo inferior que el *validation accuracy.*

- Parte convolucional (II)

Para terminar, se ha probado de nuevo a ampliar la parte convolucional de la red, añadiendo un tercer grupo (Convolución + *Batch normalization* + 'Relu' + *Max pooling* + *Spatial dropout*). Como

antes, se han experimentado para estimar si es mejor que las tres capas convolucionales tengan 64, 128 y 256 filtros, respectivamente, o 128, 256 y 512.

Esquema	Estructura duplicada	Nueva estructura (triplicada)	
Número de filtros	128+256	64+128+256	128+256+512
***Validation accuracy* (%)**	97.51	97.99	98.33

Tabla 40: Resultados tras volver a ampliar la CNN del problema acústico

Los resultados de la Tabla 40 nos llevan a ampliar de nuevo la red, añadiendo una tercera capa convolucional de 512 filtros.

- Parte densa

Como ha dado buenos resultados ampliar la parte convolucional de la red, hemos probado a ampliar también la parte densa, añadiendo una capa *fully connected* de 256 neuronas, entre la concatenación de las dos ramas y la capa de salida.

Esquema	Estructura actual	Estructura ampliada
***Validation accuracy* (%)**	98.33	98.14

Tabla 41: Efecto de ampliar la parte densa de la red del problema auditivo

En este caso, a raíz de los resultados reflejados en la Tabla 41, optamos por no modificar la red que teníamos.

- Spatial Dropout

Al igual que con el ruido gaussiano, la probabilidad del *spatial dropout* es un valor que establecimos arbitrariamente. Es por ello que la última prueba tiene por objetivo estimar un valor adecuado para este parámetro. En este caso, fijándonos en los resultados mostrados en la Tabla 42, escogemos 0.35 como valor óptimo.

Porcentaje	0.20	0.35	0.50
***Validation accuracy* (%)**	98.34	98.50	98.33

Tabla 42: Resultados al variar el porcentaje de *spatial dropout* en el problema acústico

3.2.6 Test

Finalmente, con los parámetros estimados en las pruebas anteriores, nos queda una red como la que se representa en la Figura 21. Probamos esta red sobre el conjunto de test, entrenando con los conjuntos de entrenamiento y validación, y obtenemos así un resultado final del 98.90% de acierto, con un intervalo de confianza al 95% del 0.68%. En este caso, como la tasa de error es tan baja, no vale la pena mostrar la matriz de confusión.

En cuanto al estado del arte, aunque no hemos encontrado apenas estudios que utilicen este corpus, debido a que la mayoría utiliza corpus más grandes en inglés, podemos ver que nuestros resultados son muy superiores a los que obtienen en [56].

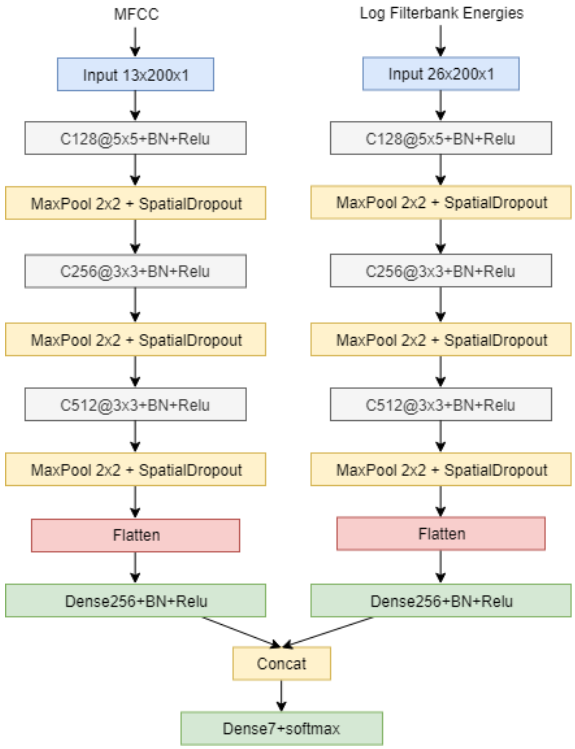


Figura 21: Modelo de red convolucional final para el problema acústico

3.3 Texto

El tercer elemento que hemos decidido tener en cuenta para nuestro clasificador multimodal de emociones es texto, de forma que no solo se tenga en cuenta cómo se dice algo, sino también qué se dice.

3.3.1 Corpus

Como en el caso anterior, esta característica sufre de una localidad. En el texto de una forma mucho más acentuada todavía, ya que cada idioma tiene unas construcciones y una gramática específicas que condiciona la forma en que expresamos nuestras emociones. Además, en el texto también es importante el registro, ya que existen diferencias emocionales entre un registro formal y uno coloquial. Como nosotros queremos construir un sistema para cualquier usuario, en el que además el texto va a provenir del habla transcrita automáticamente, lo más adecuado es que esté preparado para reconocer sentimientos en el ámbito coloquial, ya que es el tipo de texto que va a tener que analizar.

Por ello, para esta tarea hemos decidido utilizar el corpus general del Taller de Análisis de Sentimientos en la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN)[14]. Este corpus es una recopilación de 68017 tuits escritos entre noviembre de 2011 y marzo de 2012 por personas de todo tipo. Los tuits están etiquetados con una polaridad global, un nivel de agrado, la temática del texto y con polaridad a nivel de entidad. En este caso, cambiamos un etiquetado basado en emociones (enfado, alegría, tristeza, etc.), por un conjunto de clases basado en polaridad. Las clases que contempla este corpus son: muy negativo “N+”, negativo “N”, neutro “NEU”, positivo “P”, muy positivo “P+” y ausencia de emoción “NONE”. Esto se debe a que es muy difícil inferir emociones como por ejemplo el miedo a partir de texto plano. Lo que

[14] http://www.sepln.org/

haremos entonces es diseñar y afinar una red neuronal para estas clases, para después hacer algún tipo de conversión a distribuciones emocionales al combinar esta tarea con las anteriores.

Este corpus pertenece a una competición anual y está dividido para ese propósito en 60798 muestras de test y tan solo 7219 de *training*. Sin embargo, para nuestro objetivo, esta distribución no tiene ningún sentido, ya que se desperdician muchas muestras en el test que nos podrían ayudar a afinar mejor la red. Por esto, lo que hemos hecho es mezclar los dos conjuntos, barajarlos y, como en las anteriores tareas, dividir los datos en un 70% de entrenamiento, un 15% de validación y un 15% de test.

Además, la SEPLN también proporciona otros corpus más pequeños, concretamente uno con tuits políticos de la campaña de las elecciones de 2015, uno de tuits deportivos recogidos durante la final de la Copa del Rey de 2014, y otros de tuits latinoamericanos. Sin embargo, hemos descartado integrar también estas colecciones por tratar temas muy específicos y limitados que pudieran sesgar nuestro sistema (políticos o deportes) o por incluir otros dialectos del español que puedan expresar sentimientos de formas ligeramente diferentes, de forma que puedan confundir a nuestra red. De todas formas, estos corpus son de tamaño reducido y tampoco ofrecerían un interés real que merezca su consideración.

3.3.2 Extracción de características

Para la extracción de características de los tuits, no hemos hecho ningún procesado lingüístico como es habitual en las técnicas clásicas de calsificación de lenguaje natural. En su lugar, hemos utilizado directamente *word embeddings* para transformar las cadenas de palabras (tuits) en matrices de reales (cada palabra representada como un vector).

Para ello, hemos utilizado los *embeddings* disponibles en un *GitHub*[15]. Este sitio proporciona varios *embeddings* en español entrenados mediante distintas técnicas y utilizando diferentes corpus:

1. El primero se obtuvo utilizando la librería *fastText* [57] sobre el corpus *Spanish Billion Word Corpus*[16].
2. El segundo es resultado de utilizar *Global Vectors for Word Representation* [58] sobre *Spanish Billion Word Corpus.*
3. Para el tercero usaron de nuevo *fastText* [57], pero con el corpus *Wikipedia Spanish Dump*[17].
4. Por último, el cuarto está aprendido utilizando el módulo *Word2Vec* de la librería *gensim*[18] con el corpus *Spanish Billion Word Corpus*.

Como los diferentes *embeddings* están entrenados de formas distintas y podrían por ello contener información complementaria, se ha decidido utilizar los cuatro, creando así una matriz tridimensional que podría interpretarse como una imagen con cuatro canales. Esto es posible además ya que todos los *embeddings* utilizados mapean las palabras a una misma dimensionalidad, esto es, un vector de 300 dimensiones por palabra.

[15] https://github.com/dccuchile/spanish-word-embeddings
[16] http://crscardellino.github.io/SBWCE/
[17] https://archive.org/details/eswiki-20150105
[18] https://radimrehurek.com/gensim/models/word2vec.html

Además, en este caso nos encontramos con dos problemas. El primero consiste en que, como el corpus se compone de tuits, hay palabras mal escritas, como “largoooooo” por “largo” o “mio” por “mío”. Para intentar resolverlo, hemos tomado las siguientes decisiones:

- Pasamos todo el texto a minúsculas.
- Reemplazamos todas las tildes abiertas “`” y circunflejos “^” por tildes cerradas “´”, ya que son las únicas que se emplean en el castellano.
- Reemplazamos los guiones “-“ y guiones bajos “_” por espacios, ya que suelen utilizarse para formar palabras compuestas, que pueden entenderse como dos palabras separadas.
- Descartamos todos los símbolos de puntuación y almohadillas, dejando solo las palabras.
- Si alguna palabra contiene algún carácter no alfabético, la descartamos.
- Si alguna letra se repite más de tres veces consecutivas, nos quedamos solo con una, convirtiendo así palabras como “largoooooo” en la palabra válida “largo”.
- Si al buscar la palabra en el diccionario de *embeddings* no la encontramos, la buscamos en el diccionario de *embeddings* sin tildes. Porque en internet la gente suele escribir sin poner las tildes.
- Si aun así no la encontramos, buscamos la palabra quitándole las tildes en el diccionario de *embeddings*, para el caso de una palabra que no tenga tildes, pero el usuario haya puesto alguna por accidente.
- Por último, si aun así no encontramos el *embedding*, buscamos la palabra sin tildes en el diccionario de *embeddings* sin tildes, para resolver el caso de que el usuario haya escrito la palabra poniendo la tilde que lleva, en el lugar incorrecto.

Así resolvemos incorrecciones como las que hemos planteado a modo de ejemplo en el párrafo de arriba.

El segundo problema es el mismo que teníamos con el problema auditivo, que los tuits tienen longitudes diferentes. Sin embargo, en este caso es más fácil, porque el más largo tiene 30 palabras, es decir, la variabilidad no es mucha. Por ello, en este caso hemos decidido aplicar directamente *padding* a 30 palabras, añadiendo vectores de ceros a los tuits más cortos.

Así finalmente transformamos cada tuit en una matriz tridimensional de tamaño 30x300x4.

3.3.3 Modelo inicial

Como en el problema anterior, lo más lógico puede parecer utilizar redes recurrentes para el análisis de este tipo de datos secuenciales. Sin embargo, debido a la lentitud de este tipo de sistemas, hemos de nuevo intentado abordar el problema mediante redes convolucionales, que son más rápidas. Esto es posible ya que, como ya hemos dicho, cada palabra se representa mediante un vector utilizando los *embeddings*, de forma que una frase será una matriz, y si utilizamos varios *embeddings*, tenemos una imagen con cuatro canales, como si del *rgba* se tratara.

Existen precedentes en este planteamiento que avalan la idea y nos proporcionan un punto de partida. Concretamente, el modelo inicial planteado se basa en la propuesta de [59]. Este modelo presenta la idea de hacer convoluciones de forma que los filtros tengan por tamaño la dimensionalidad de las representaciones, por un número *n* de palabras, sin *padding*. Además, plantean realizar en paralelo varias convoluciones variando el tamaño *n* de los filtros, para después concatenar los resultados tras un *GlobalMaxPooling* y pasarlos a una última capa densa.

Un esquema de nuestro modelo inicial puede verse en la Figura 22. Nosotros hemos elegido hacer en paralelo 3 convoluciones de 3, 4 y 5 palabras, respectivamente.

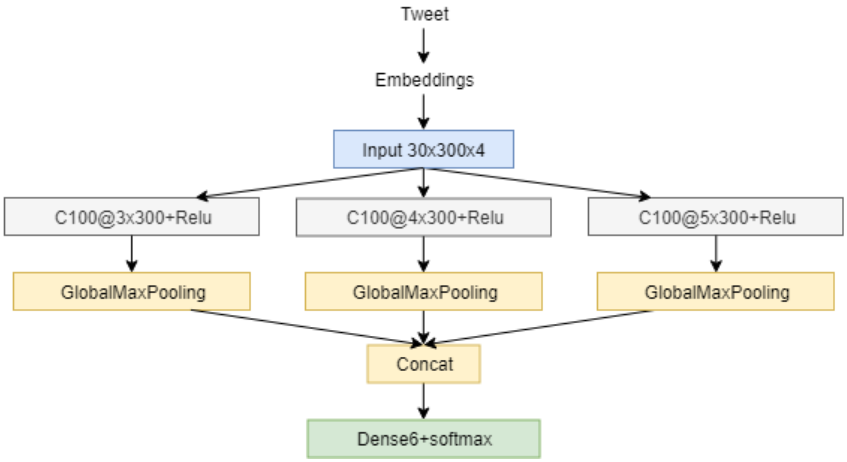


Figura 22: Modelo inicial para el problema textual

Como en los apartados anteriores, la función de pérdida que se ha elegido es el *categorical crossentropy* y el optimizador utilizado es *adam*.

Además, empezamos con un tamaño de *batch* de 100 y 150 *epochs*. De forma similar a como hicimos con los anteriores apartados, los resultados de *accuracy* que tenemos en cuenta se corresponden con la media de *accuracy* de los últimos 20 *epochs*.

3.3.4 Preproceso

Cada uno de los *embeddings* utilizados es distinto y está entrenado mediante diferentes algoritmos. Esto se traduce en que los vectores obtenidos a partir de ellos pueden encontrarse en escalas diferentes. Por ejemplo, los valores del segundo *embedding* oscilan más o menos entre -7 y 2, mientras que para el tercer *embedding* todas las componentes de los vectores se valen entre -0.3 y 0.3, aproximadamente. Por ello, se ha valorado normalizar cada uno de los *embeddings*, calculando, del conjunto de entrenamiento, la media y la desviación típica de cada uno de los canales (de cada uno de los *embeddings*). Una vez tenemos esos valores, le restamos a cada canal de cada imagen de los conjuntos de entrenamiento y validación la media correspondiente y lo dividimos entre la desviación típica que corresponda. Así los cuatro canales siguen una distribución normal de acuerdo al conjunto de entrenamiento y por tanto tienen valores similares. Sin embargo, como podemos observar en los resultados volcados en la Tabla 43, esta solución es contraproducente. Esto puede deberse a que el módulo de los vectores en que son convertidas las palabras pueda contener determinada información, lo cual lleva a empeorar los resultados tras la normalización. También se probó a normalizar cada uno de los vectores en que convertimos las palabras, de forma que todos fueran unitarios, pero dio un resultado también malo, similar al del normalizado por canales.

	Sin normalizar	Normalizado por canales
***Validation accuracy* (%)**	62.75	60.40

Tabla 43: Efecto de normalizar los datos en el problema textual

3.3.5 Estimación de la red

- *Overfitting* (I)

Como sucedía con el audio, el primer problema con el que nos topamos es con el sobreentrenamiento, ya que porcentaje de aciertos en el conjunto de entrenamiento alcanza un 99%. Por ello, empezamos probando a añadir algunas técnicas de regularización que puedan paliar esta circunstancia. Lo primero que probamos es a añadir *batch normalization* entre las capas convolucionales y sus activaciones, obteniendo los resultados reflejados en la Tabla 44.

	Sin *batchnorm*	Con *batchnorm*
***Validation accuracy* (%)**	62.75	64.35
Training accuracy (%)	99.19	99.32

Tabla 44: Resultados al aplicar *batch normalization* en la CNN del problema textual

Los resultados mejoran significativamente, por lo que, aunque no ayuda con el *overfitting*, lo incorporamos.

También probamos a añadir ruido gaussiano entre la capa *batch normalization* y la activación. A raíz de los resultados de la Tabla 45, decidimos no incluir el ruido gaussiano a la red.

Porcentaje	0.0	0.1	0.2	0.3	0.4	0.5
***Validation accuracy* (%)**	64.35	63.62	64.00	63.16	62.99	61.48
Training accuracy (%)	99.32	99.32	99.31	99.27	99.16	99.16

Tabla 45: Efecto al añadir ruido gaussiano a la red del problema textual

- Número de palabras

Antes de continuar tratando el sobreentrenamiento que tiene lugar, decidimos estimar parámetros potencialmente más importantes. El primero de ellos es el tamaño óptimo de los filtros que aplicamos a la entrada. Recordemos que esos filtros son de tamaño igual a la dimensionalidad de las representaciones por *n* palabras. Recordemos también que hasta ahora decidimos arbitrariamente realizar 3 convoluciones en paralelo cuyos filtros abarcan 3, 4 y 5 palabras, respectivamente. En la Tabla 46, se muestran los resultados tras probar con diferentes combinaciones de número de palabras / tamaño de los filtros.

Tamaños de los filtros	**1,2,3**	**2,3,4**	**3,4,5**	**4,5,6**	**5,6,7**	**6,7,8**
***Validation accuracy* (%)**	65.05	64.98	64.35	63.84	63.67	63.50
Tamaños de los filtros	**2,4,6**	**3,5,7**	**1,2,3,4**	**2,3,4,5**	**3,4,5,6**	**4,5,6,7**
***Validation accuracy* (%)**	64.47	63.84	65.73	64.80	64.48	63.80
Tamaños de los filtros	**5,6,7,8**	**6,7,8,9**	**1,3,5,7**	**2,4,6,8**	**3,5,7,9**	**1,2,3,4,5**
***Validation accuracy* (%)**	63.28	63.58	65.05	65.04	64.68	65.53
Tamaños de los filtros	**2,3,4,5,6**	**3,4,5,6,7**	**4,5,6,7,8**	**5,6,7,8,9**	**2,4,6,8,10**	**1,3,5,7,9**
***Validation accuracy* (%)**	64.72	64.15	63.76	63.34	64.71	65.12

Tabla 46: Resultados obtenidos al variar el tamaño de los filtros en el problema textual

Podemos ver que los mejores porcentajes se obtienen cuando los filtros son de tamaño 1x300, 2x300, 3x300 y 4x300 en paralelo. Nos queda pues una red como la que se representa en la Figura 23.

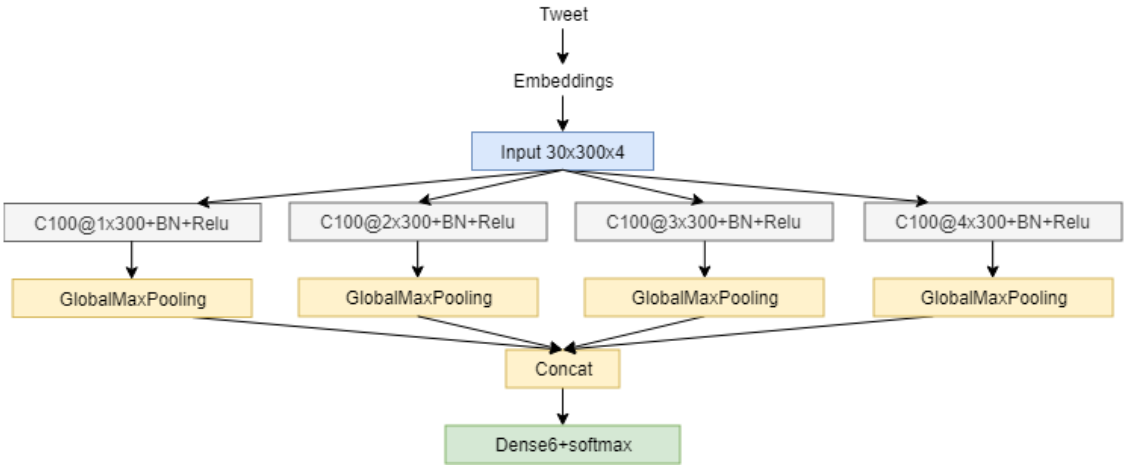


Figura 23: Modelo con tamaños de los filtros estimados para el problema textual

- Embeddings

Otro parámetro importante que debemos estimar, como con las características del problema auditivo, es si necesitamos o no todos los *embeddings* (los 4) que estamos utilizando. Hemos probado todas las combinaciones posibles, como se resume en la Tabla 47. Como se puede ver en los resultados mostrados (numerándolos del uno al cuatro), todos los *embeddings* utilizados aportan alguna información original, de forma que obtenemos un mayor porcentaje de acierto manteniéndolos todos.

Embeddings	**1**	**2**	**3**	**4**	**1,2**	**1,3**	**1,4**	**2,3**
***Validation accuracy* (%)**	64.71	62.59	59.56	64.11	64.72	65.29	65.03	63.45
Embeddings	**2,4**	**3,4**	**1,2,3**	**1,2,4**	**1,3,4**	**2,3,4**	**1,2,3,4**	
***Validation accuracy* (%)**	64.76	64.85	65.51	65.55	65.54	65.38	65.73	

Tabla 47: Efecto de los embeddings utilizados en el problema textual

- Overfitting (II)

Una vez estimados esos parámetros, volvemos a intentar resolver el problema de sobreentrenamiento que tenemos. Para ello probamos a añadir *spatial dropout* entre la activación de las capas convolucionales y el *global max pooling*. Los resultados obtenidos variando el porcentaje de *dropout* se muestran en la Tabla 48.

Porcentaje	**0.00**	**0.10**	**0.15**	**0.20**	**0.25**
***Validation accuracy* (%)**	65.73	66.14	66.07	65.78	65.82
Training accuracy (%)	99.31	99.19	99.07	98.93	98.77
Porcentaje	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
***Validation accuracy* (%)**	65.75	66.20	66.61	66.18	66.65
Training accuracy (%)	98.57	98.34	98.07	97.66	97.21

Tabla 48: Resultados obtenidos al añadir *spatial dropout* a la CNN del problema textual

Con un porcentaje del 50% de *dropout* conseguimos mejorar casi un punto los resultados, además de reducir en más de dos el *overfitting*.

Además de eso, probamos, como ya hemos hecho en los problemas anteriores, a realizar rotaciones horizontales y verticales a las "imágenes", consiguiendo así los resultados de la Tabla 49.

Volteos	Sin volteos	Horizontales	Verticales	Horizontales y verticales
***Validation accuracy* (%)**	66.65	66.54	66.83	66.91
Training accuracy (%)	97.21	94.23	94.40	88.40

Tabla 49: Resultados obtenidos al añadir volteos a los datos del problema textual

En esta ocasión, aplicar volteos tanto horizontales como verticales mejora nuestros resultados, reduciendo además el porcentaje de acierto en entrenamiento, así que lo añadimos.

Sin embargo, como aún existe una diferencia muy notoria entre los resultados en entrenamiento y validación, probamos a añadir también variaciones aleatorias de "color", es decir, alteraciones aleatorias en los diferentes canales (en nuestro caso, cada uno de los *embeddings*). Como se ve en la Tabla 50, volvemos a conseguir mejorías cuando hacemos variaciones con un rango de 0.4.

Rango de variaciones	0.00	0.10	0.15	0.20	0.25
***Validation accuracy* (%)**	66.91	67.67	67.80	68.19	68.41
Training accuracy (%)	88.40	86.47	84.93	82.83	81.10
Rango de variaciones	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
***Validation accuracy* (%)**	68.45	68.73	69.01	68.50	68.51
Training accuracy (%)	79.30	77.29	75.22	74.04	72.72

Tabla 50: Resultados al añadir variaciones en los canales de los datos en el problema textual

Aunque en este punto casi no quedaba sobreentrenamiento, aún probamos a añadir regularización l2 solo en la capa densa del final, ya que añadirla en las capas convolucionales estropeaba mucho los resultados. Aun así, como se refleja en la Tabla 51, no beneficia en nada este tipo de regularización, y la desechamos.

Valor de l2	Sin l2	0.1	1	2	3
***Validation accuracy* (%)**	69.01	68.32	66.33	66.01	65.91
Training accuracy (%)	75.22	69.42	65.04	64.05	64.02
Valor de l2	**4**	**5**	**6**	**7**	**8**
***Validation accuracy* (%)**	65.28	65.65	65.35	64.69	65.17
Training accuracy (%)	63.14	63.60	63.10	62.83	62.91

Tabla 51: Resultados obtenidos al añadir regularización l2 en la CNN del problema textual

- Parte convolucional

Desde el planteamiento del modelo inicial, las convoluciones utilizan 100 filtros de forma arbitraria. Así que el siguiente paso fue buscar el número ideal de filtros, cuyos resultados se pueden ver en la Tabla 52.

Número de filtros	50	100	150	200	250	300
***Validation accuracy* (%)**	68.28	69.01	69.08	69.13	68.91	68.99

Tabla 52: Efecto del número de filtros en la CNN del problema textual

A la vista de los resultados, establecemos en 200 el número de filtros de las capas convolucionales. Sin embargo, de esta manera el *training accuracy* sube hasta el 77.34%, casi diez puntos por encima de los resultados sobre el conjunto de validación, por lo que probamos unas últimas técnicas para reducir el *overfitting*.

- *Overfitting* (III)

Como ya hemos probado muchas de las opciones habituales que ofrece *Keras* para combatir el sobreentrenamiento, en este punto intentamos aplicar dos técnicas un poco más elaboradas.

La primera de ellas es el *cutout*. Las oclusiones que añade esta técnica pueden ser uniformes de un solo color, o cada píxel del parche de un color aleatorio. Probamos a variar el porcentaje máximo de área de la imagen que pueden ocupar los parches, así como si el color del parche es uniforme o no, como se ve en la Tabla 53.

Porcentaje de área con color uniforme	Sin CO	0.1	0.2	0.3	0.4	0.5
***Validation accuracy* (%)**	69.13	69.25	69.56	69.40	69.31	69.45
Training accuracy (%)	77.34	75.21	74.86	74.74	74.70	74.62
Porcentaje de área con color variable	-	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
***Validation accuracy* (%)**	-	69.52	69.34	69.40	69.48	69.19
Training accuracy (%)	-	74.56	74.07	73.83	73.85	73.92

Tabla 53: Resultados obtenidos al aplicar *cutout* en la CNN del problema textual

Con esta técnica, tapando hasta un 20% de la imagen con un color uniforme obtenemos los mejores resultados, que superan los obtenidos anteriormente, además de reducir el porcentaje de acierto en entrenamiento.

La segunda técnica ya la utilizamos en el problema facial, es el *mixup data augmentation (MDA)*. De nuevo, probamos variando el parámetro alpha de la función beta que utiliza para mezclar las imágenes.

Alpha	Sin MDA	0.1	0.2	0.3	0.4	0.5
***Validation accuracy* (%)**	69.56	69.35	69.33	69.25	69.09	69.30
Training accuracy (%)	74.86	71.71	70.21	68.62	68.08	67.17
Alpha	**-**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
***Validation accuracy* (%)**	-	69.01	68.91	68.78	68.75	68.67
Training accuracy (%)	-	66.74	66.14	65.72	65.32	64.98

Tabla 54: Resultados obtenidos al aplicar MDA en la CNN del problema textual

Como se puede ver en la Tabla 54, esta técnica no nos ayuda a mejorar los resultados, así que no la utilizamos en adelante.

- Ampliaciones de la red

Por último, se ha probado a ampliar la red de varias formas, añadiendo capas convolucionales o densas en distintos puntos del modelo. La primera propuesta consiste en, después de cada convolución, cuya salida es un vector de tamaño *n*x1, añadir una capa convolucional con filtros de tamaño 3x1, como se representa en la Figura 24. Sin embargo, con esta arquitectura los resultados bajan hasta un 68.68% de acierto en validación, así que la descartamos.

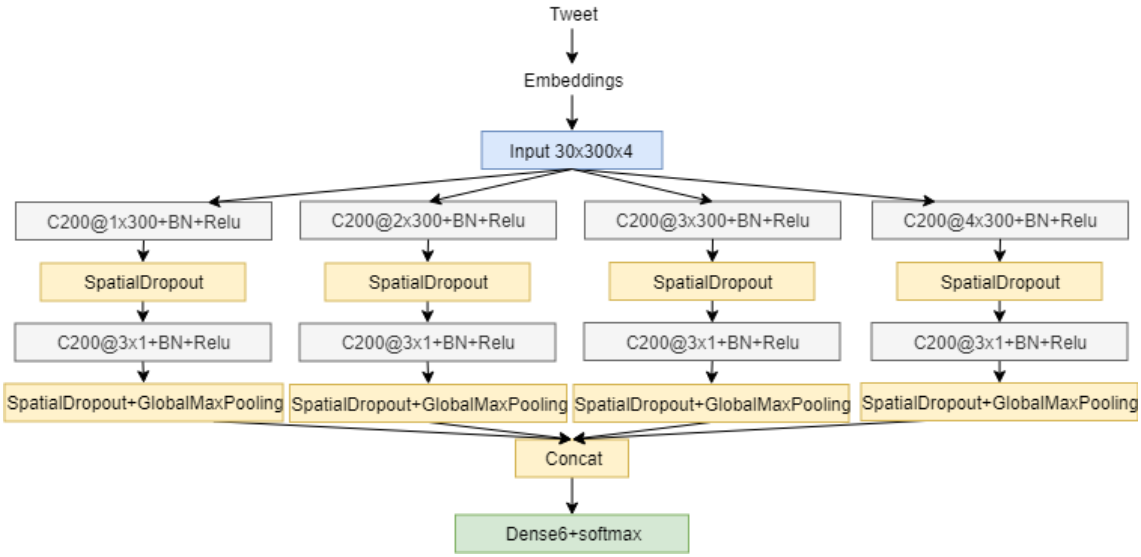


Figura 24: Primera propuesta de ampliación para la CNN del problema textual

La segunda propuesta consiste en añadir una capa densa de 100 neuronas entre la concatenación y la capa de salida, como se puede ver en la Figura 25. De nuevo, los resultados empeoran, aunque no mucho, bajando a un 69.50%, así que también desechamos esta idea.

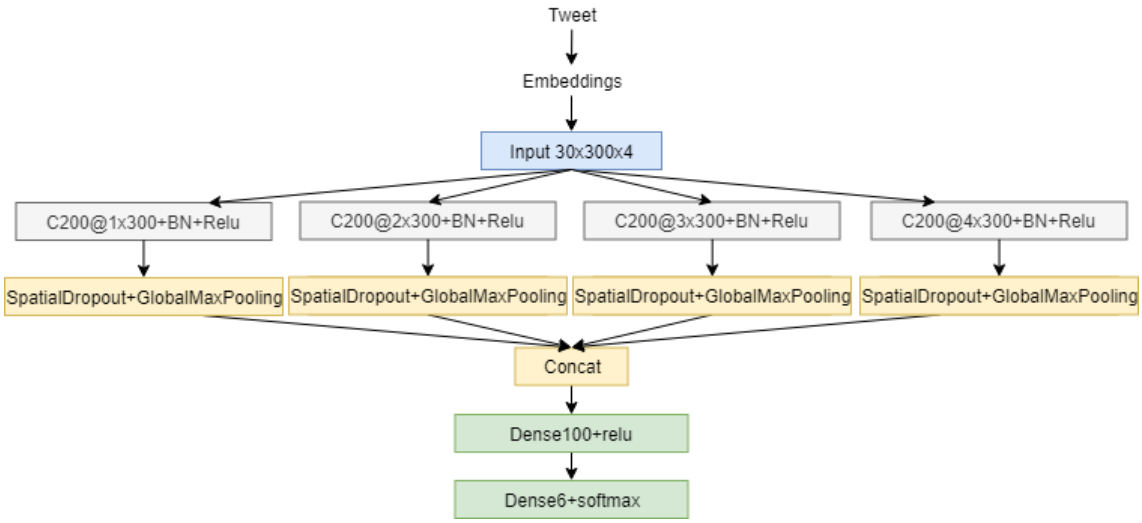


Figura 25: Primera propuesta de ampliación para la CNN del problema textual

Por último, también se ha propuesto añadir una capa convolucional inicial directamente conectada a la entrada con 20 filtros de tamaño 3x3 y *padding*, a modo de preproceso de la entrada, como se puede observar en la Figura 26. Esta última arquitectura planteada tampoco nos ofrece ninguna mejora, ya que los resultados bajan a un 67.62%.

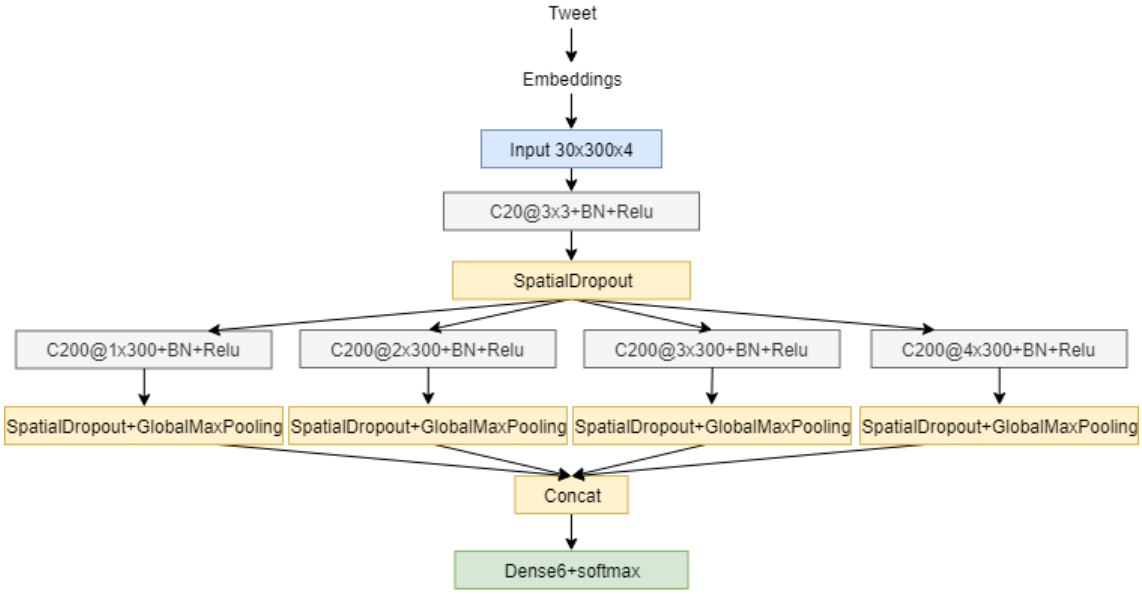


Figura 26: Tercera propuesta de ampliación para la CNN del problema textual

Como ninguna de las ampliaciones ha dado ya mejores resultados, tomamos como red final la que teníamos previamente, que podemos ver en la Figura 27.

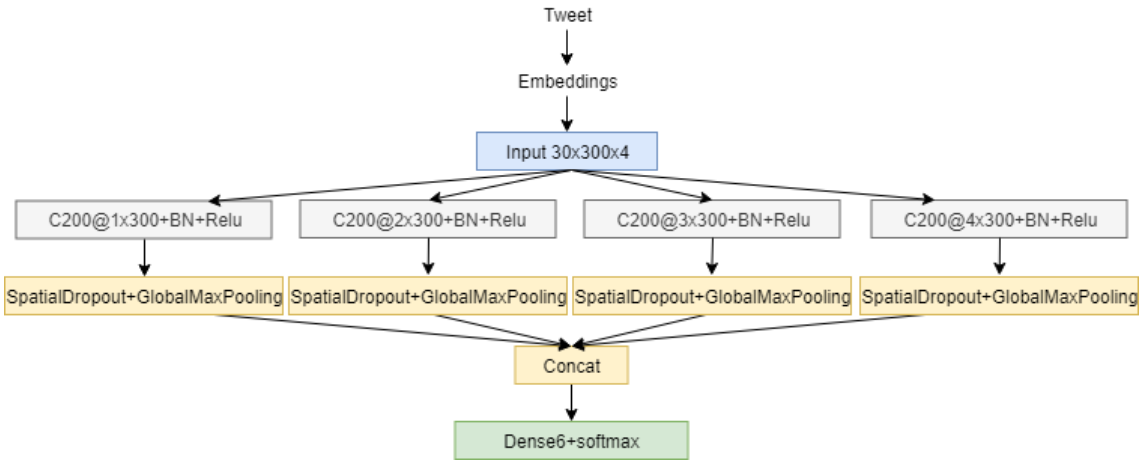


Figura 27: Red convolucional final para el problema textual

3.3.7 Modelo alternativo

Como los resultados no son muy altos (frente al casi 99% de aciertos del problema auditivo o el 88% del modelo facial) probamos un modelo alternativo. Recordemos que decidimos tratar este problema mediante una red convolucional en lugar de una recurrente, que es la solución más natural. Esta decisión fue tomada porque las convolucionales son mucho más rápidas y, en principio, había estudios previos que avalan la idea.

Es por ello que se ha intentado superar los resultados obtenidos con la red convolucional utilizando una recurrente. Para ello planteamos una *long short-term memory* (LSTM) muy sencilla, como la que se puede ver en la Figura 28 (la capa LSTM solo devuelve la última salida). En este caso, intentamos paliar la lentitud de este tipo de redes utilizando solo uno de los *embeddings* planteados en lugar de los cuatro.

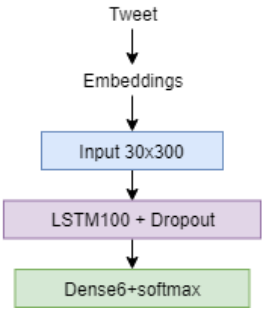


Figura 28: LSTM inicial para el problema textual

- Embedding

La primera prueba para esta red se trata de escoger el mejor de los *embeddings*. A la vista de los resultados de la Tabla 55, probamos a combinar los dos mejores *embeddings* (el 1 y el 4) usando dos redes en paralelo como las de la Figura 28 y sumando las salidas de las capas de salida. Otras formas de combinarlos se probaron, pero no dieron buenos resultados. Con esta nueva red, que se representa en la Figura 29, alcanzamos un porcentaje de acierto sobre el conjunto de validación de un 66.80%, por lo que lo escogimos para continuar. No se plantea utilizar tres o los cuatro *embeddings*, como ya se ha dicho, por velocidad. De todas formas, se probó y no ofrecía mejores resultados.

Embeddings	1	2	3	4
***Validation accuracy* (%)**	64.57	63.05	58.52	64.90

Tabla 55: Resultados para elegir qué *embedding* utilizar con la LSTM del problema textual

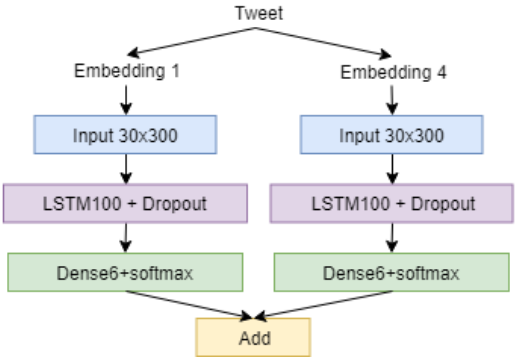


Figura 29: LSTM con dos *embeddings* para el problema textual

- Unidades

A continuación, estimamos el número de unidades ideal de las capas recurrentes, es decir, la dimensionalidad de la salida, que fue establecida inicialmente en 100.

Unidades	20	50	100	150	200	250
***Validation accuracy* (%)**	68.05	66.60	66.80	66.29	66.88	66.72

Tabla 56: Efecto del número de unidades en la LSTM del problema textual

Dados los resultados de la Tabla 56, establecemos en 20 el número de unidades de las capas LSTM.

- *Dropout*

Inicialmente se añadió *dropout* a las capas LSTM porque ayudaba a estabilizar la red durante el entrenamiento. Tras estimar el número adecuado de unidades, se decidió obtener el mejor porcentaje para el *dropout*, que hasta este momento ha sido de 0.2.

Porcentaje	0.1	0.2	0.3	0.4	0.5	0.6
***Validation accuracy* (%)**	66.23	68.05	68.26	68.27	68.93	68.42

Tabla 57: Efecto del valor de *dropout* en la LSTM del problema textual

Establecemos el *dropout* en un 50%, de acuerdo a los resultados mostrados en la Tabla 57.

- Ampliaciones de la red

Como en el caso de la red convolucional, se proponen algunas ampliaciones a la LSTM básica utilizada hasta ahora. La primera idea consiste en añadir, entre la entrada y la LSTM, una capa convolucional a modo de preproceso de la entrada. Sin embargo, así obtenemos unos resultados bastante peores, de un 67.49%. Un esquema de esta propuesta podemos verla en la Figura 30.

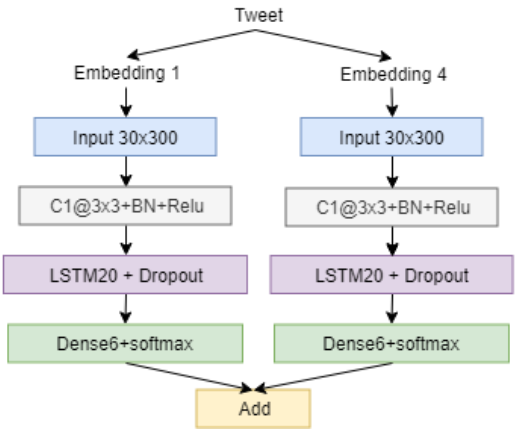


Figura 30: Primera propuesta de ampliación para la LSTM del problema textual

La segunda ampliación, representada en la Figura 31, consiste en agregar una capa densa entre la LSTM y la capa de salida. Los resultados en este caso son muy similares, de un 68.80%, así que también rechazamos esta arquitectura.

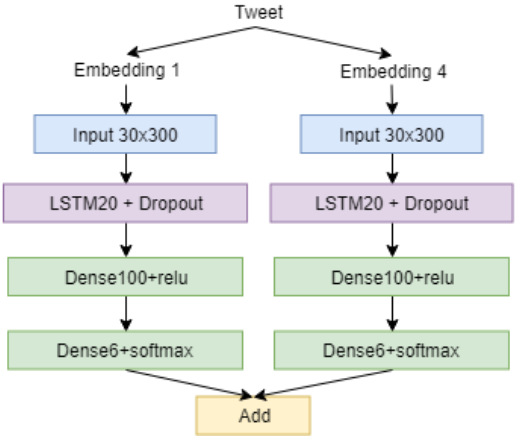


Figura 31: Segunda propuesta de ampliación para la LSTM del problema textual

La tercera propuesta consiste en modificar la LSTM para que devuelva todas las secuencias, de forma que la salida de la LSTM será una matriz de tamaño 30x20. Entonces tratamos esa matriz como una imagen y la pasamos por una capa convolucional que termina en un *global max pooling*, que conectamos a la capa de salida. Con esta estructura, representada en la Figura 32, sí que obtenemos una mejoría en los resultados, alcanzando un 69.45%.

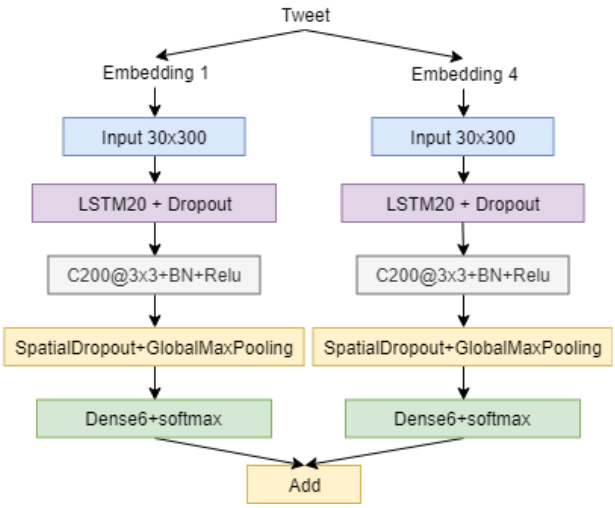


Figura 32: Tercera propuesta de ampliación para la LSTM del problema textual

Como alternativa a la arquitectura anterior, se propone, en lugar de una capa convolucional, añadir cuatro en paralelo, como si integráramos nuestra red convolucional del apartado anterior entre la LSTM y la salida. Este modelo está representado en la Figura 33, y nos ofrece los mejores resultados hasta el momento para este problema, con un 69.64% de acierto. También probamos a añadir el *data augmentation* que utilizamos en la convolucional, pero para la LSTM no funciona y empeora los resultados hasta un 67.25%. Tampoco se hace necesario, ya que no hay sobreentrenamiento en la LSTM, donde el acierto en entrenamiento es del 72.77%. Por tanto, la LSTM final para el problema textual es la que aparece en la Figura 33.

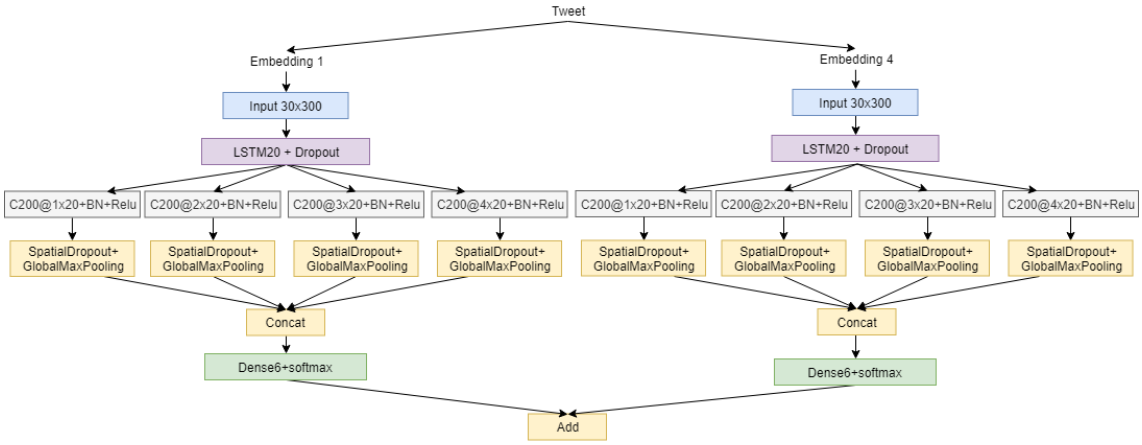
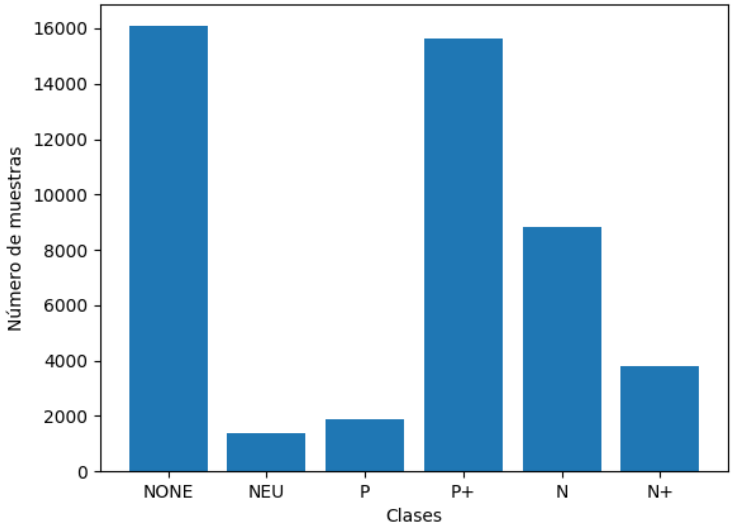


Figura 33: Cuarta propuesta de ampliación para la LSTM del problema textual

- Equilibrado de clases

Por último, como en el problema facial, en este problema existe un desbalanceo de clases, como puede verse en la Gráfica 3. Por ello, hemos probado a balancear las clases utilizando el método *compute_class_weight* del módulo *sklearn*. Sin embargo, con el balanceado los resultados bajaban a un 69.56%, así que descartamos la idea finalmente.



Gráfica 3: Número de muestras de entrenamiento por clase en el problema textual

3.3.8 *Ensemble*

Como los modelos han obtenido resultados similares, nuestra última propuesta fue utilizarlos conjuntamente. Para crear ese *ensemble* se han estudiado diferentes técnicas:

1. Añadir una capa que sume las salidas de las dos redes y a continuación poner una capa densa de salida con seis neuronas y una *softmax*. Esto empeoraba bastante los resultados de las dos redes, así que no se incluye en la tabla de resultados.
2. Añadir una capa que concatene las salidas de las dos redes y a continuación poner una capa densa de salida con seis neuronas y una *softmax*. Esta solución admite variaciones:
    a. Congelando o no los pesos de las dos redes ya entrenadas, de forma que entrenaríamos solo la última capa de fusión.
    b. Pasando a las dos redes los datos sin alterar, o pasarle a la convolucional las muestras con el *data augmentation* que estimamos.
3. Sencillamente asignar un peso a cada una de las dos redes y dar como resultado una suma ponderada de las dos salidas.

En la Tabla 58 se muestran los resultados de las diferentes opciones que admite la solución 2. Por otra parte, en la Tabla 59 aparecen los resultados de la solución 3 variando el peso de la LSTM, el peso de la CNN lo mantenemos en 1.

Validation accuracy (%)	Congelando pesos	Sin congelar ningún peso
Con *data augmentation*	68.07	69.47
Sin *data augmentation*	69.20	67.25

Tabla 58: Resultados al unir las redes del problema textual en una nueva red

Peso LSTM	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
***Validation accuracy* (%)**	69.56	70.00	70.22	70.24	70.37	70.42	70.58	70.63
Peso LSTM	**0.9**	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**
***Validation accuracy* (%)**	70.69	70.63	70.51	70.45	70.43	70.39	70.37	70.26
Peso LSTM	**1.7**	**1.8**	**1.9**					
***Validation accuracy* (%)**	70.27	70.18	70.17					

Tabla 59: Resultados al combinar las redes del problema textual mediante pesos

Como se puede ver en las tablas, los mejores resultados los obtenemos con una sencilla suma ponderada de las salidas de las dos redes, multiplicando por 0.9 la salida de la LSTM y sumándosela a la de la CNN. Por eso, nuestro modelo final es un *ensemble* de la CNN y la LSTM con esos pesos.

3.3.9 Test

En este caso la fase de test es un poco más extensa, ya que evaluamos la CNN, la LSTM y el *ensemble*. Como se ve en la Tabla 60, los resultados son muy similares a los obtenidos en validación. Se proporcionan los porcentajes de acierto con su intervalo de confianza al 95%.

Porcentaje	CNN	LSTM	*Ensemble*
***Test accuracy* (%)**	68.69±0.90	69.12±0.90	70.10±0.89

Tabla 60: Resultados en el test del problema textual con seis clases

Además, la matriz de confusión la podemos ver en la Tabla 61.

Matriz de confusión	None	Neu	P	P+	N	N+
None	78.80	0.15	0.27	8.87	10.26	1.66
Neu	16.61	5.90	0.74	29.15	40.22	7.38
P	33.70	0.44	17.94	38.07	8.10	1.75
P+	18.17	0.36	0.33	78.13	2.57	0.44
N	22.63	0.31	0.10	4.30	66.58	6.08
N+	11.68	0.50	0.12	4.47	23.85	59.38

Tabla 61: Matriz de confusión en el problema textual con seis clases

Llama la atención los pésimos resultados de la clase “Neu”, que solo acierta un 5.9% de las veces. Esto puede deberse a que es la clase que cuenta con menos muestras. También llama la atención que “None” causa muchos errores de las otras clases, lo cual puede ser (de forma paralela) porque es la clase más abundante. Sin embargo, ya hemos comprobado que balancear las clases empeora los resultados.

Por otra parte, en esta tarea convendría también comprobar el porcentaje de acierto reduciendo el problema a cuatro clases, uniendo P con P+ y N con N+. En ese caso, reentrenamos los dos modelos exactamente con los mismos parámetros, y obtenemos los resultados que aparecen en la Tabla 62.

Porcentaje	CNN	LSTM	*Ensemble*
***Test accuracy* (%)**	74.40±0.85	74.71±0.84	75.14±0.84

Tabla 62: Resultados en el test del problema textual con cuatro clases

De nuevo, comprobamos también la matriz de confusión, que aparece en la Tabla 63.

Matriz de confusión	None	Neu	P / P+	N / N+
None	77.59	0.09	10.97	11.35
Neu	16.61	5.17	31.37	46.86
P / P+	18.09	0.18	78.52	3.21
N / N+	19.90	0.26	5.53	74.32

Tabla 63: Matriz de confusión en el problema textual con cuatro clases

Se puede ver que se extraen las mismas conclusiones que con el problema en seis clases. La única diferencia es que conseguimos eliminar las confusiones por intensidad de la polaridad.

Podemos decir que son muy buenos resultados comparándolos con el estado del arte en esta tarea, y más si disponemos de artículos sobre el mismo corpus. Contrastando los nuestros con los obtenidos en [32] y [31], comprobamos que son muy buenos resultados. Aunque hay que tener en cuenta que los nuestros cuentan con la ventaja de que hicimos una partición libre y mejor distribuida de los datos que la que ellos podían utilizar en la competición, por lo que realmente no son del todo comparables, pero nos dan una medida para asegurar que hemos obtenido buenos resultados.

3.4 Multimodal

A la hora de combinar nuestros tres modelos unimodales existe un problema, y es que no encontramos ningún corpus de vídeo (trimodal) etiquetado con emociones en español. Como ya se ha comentado, la expresión de emociones mediante audio y texto es dependiente del idioma, de modo que no podemos utilizar un corpus en inglés.

De todas formas, esto no tiene por qué ser necesariamente malo, ya que el mejor modo de combinarlos puede depender del dominio. Por ejemplo, en un ambiente con mala iluminación posiblemente lo mejor sea darle menos peso al clasificador facial, mientras que, en un ambiente ruidoso, como un bar o una clase, probablemente habría que reducir la importancia del clasificador acústico.

Es por ello que hemos diseñado una arquitectura de combinación concreta y proporcionamos unos pesos iniciales fundamentados, pero permitimos al usuario ajustar a conveniencia unos parámetros que condicionan dicha combinación. El reconocedor de emociones multimodal se obtiene combinando linealmente el reconocedor facial y el acústico de acuerdo a un peso, siguiendo la Ecuación 1. Entonces, como el reconocedor textual no reconoce emociones, sino polaridad, lo utilizamos a modo de desempate en casos de confusión. Es decir, si la diferencia entre la emoción con más puntuación al combinar los modelos facial y acústico y otras emociones es menor a un umbral, consideramos que hay duda, y utilizamos la polaridad obtenida del texto para desempatar. El desempate se hace de la siguiente forma:

- Si la polaridad es “None”, se elige sencillamente la emoción que tenía más puntuación.

- Si la polaridad es “Neu”, se elige la emoción más neutra, siguiendo el orden: Neutro > Sorpresa > Asco > Enfado > Miedo > Felicidad > Tristeza.
- Si la polaridad es “P”, se elige la emoción más positiva, siguiendo el orden: Felicidad > Sorpresa > Neutro > Asco > Enfado > Miedo > Tristeza.
- Si la polaridad es “N”, se elige la emoción más negativa, siguiendo el orden: Tristeza > Miedo > Enfado > Asco > Neutro > Sorpresa> Felicidad.

$$Emoción = Emoción_{facial} * peso + Emoción_{acústica} * (1 - peso)$$

Ecuación 1: Combinación de los modelos facial y acústico

Como se puede observar, finalmente hemos decidido quedarnos con el clasificador en cuatro niveles de polaridad. Esto es porque tanto si da “P” como “P+”, escogemos la emoción más positiva (igual con el negativo), así que realmente no hay una diferencia, y el modelo de cuatro etiquetas es probablemente más preciso.

Para la ordenación de las emociones de acuerdo a su polaridad se ha utilizado el modelo de *pleasure-arousal* explicado en el “Capítulo 2: Estado del arte”, ya que podemos considerar que el *pleasure* puede corresponderse con la polaridad (positivo-negativo) de una emoción. Entonces, nos hemos basado en [60], donde se proporcionan experimentalmente valores de *pleasure-arousal* para algunas emociones básicas en español. De acuerdo a los valores de *pleasure*, calculados como medias de los datos del artículo para cada emoción, hemos ordenado las emociones de mayor a menor para la polaridad positiva, de menor a mayor para la negativa, y de menor a mayor en valor absoluto para la polaridad neutra.

Establecido esto, tenemos una arquitectura consistente para el clasificador multimodal, de forma que los parámetros que dejamos ajustables para el usuario son: el peso que se le da al modelo facial, y el umbral (%) de confusión a partir del cual se utiliza el modelo textual para desempatar. Por una parte, el umbral se establece arbitrariamente en un 20%, es decir, si la más votada y alguna otra se distancian en menos del 20%, hay que desempatar. Por otra parte, el peso del modelo textual se fija inicialmente en un 60%, adaptando de forma proporcional el trabajo de [61] [62], que estudian qué porcentajes aportan diferentes partes de nuestro acto comunicativo a la expresión de estados afectivos, incluso en situaciones de expresiones contradictorias. En esos artículos se establece que un 55% corresponde a la expresión corporal, que en nuestro caso puede interpretarse como la expresión facial, un 7% al lenguaje verbal, que en nuestro caso es el texto, y un 38% al lenguaje paraverbal, que para nosotros se corresponde al modelo acústico. Como nosotros extraemos el modelo textual de la combinación, ese 55% queda proporcionalmente en aproximadamente un 60%.

CAPÍTULO 4

Desarrollo de la aplicación

Una vez entrenamos con los corpus completos los tres modelos desarrollados, hemos querido incorporar el reconocedor multimodal a una aplicación web en Django que obtenga imágenes de la *webcam* y audio del micrófono y, en directo, prediga la emoción del usuario.

En cuanto a la temporalidad, hay que distinguir el modelo facial de los modelos textual y acústico. Por una parte, el primero es instantáneo y es interesante muestrear con una alta frecuencia, ya que las expresiones faciales pueden cambiar rápido. Por ello, realizamos la evaluación de la expresión facial cada 400 milisegundos. Como la aplicación es web, con esta velocidad de muestreo el modelo debe poder ejecutarse en el cliente mediante *Javascript* (los navegadores no tienen intérprete de Python y, evidentemente, es una mala práctica hacer peticiones al servidor cada 400 milisegundos). Por ello, lo primero que se ha hecho es convertir nuestro modelo facial guardado al formato de *Tensorflow.js*, que es una librería para poder utilizar todas las utilidades de *Tensorflow* y *Keras* en *Javascript*.

El problema que tiene *Tensorflow.js* es que es un proyecto muy joven, por lo que muchas utilidades aún no están implementadas. Por ejemplo, la capa *SpatialDropout* que nosotros utilizamos en nuestros modelos, no está disponible por el momento. Por eso, para que sea capaz de cargar los modelos de forma adecuada, hubo que implementar la capa en *Javascript* siguiendo sus recomendaciones para escribir capas personalizadas. Por suerte, es una capa de comportamiento sencillo que se define fácilmente, lo que hace es borrar (poner a 0) canales de la imagen con una probabilidad determinada. Una vez tenemos esto, la librería ya es capaz de leer correctamente el modelo y de cargarlo.

Por otra parte, no tiene sentido que los modelos textual y acústico sigan un muestreo tan frecuente, sino que sería lógico realizar una predicción cada 5 o 6 segundos aproximadamente. Estos modelos, al contrario que el facial, van a ejecutarse en el servidor por varios motivos: porque las predicciones son mucho más espaciadas y no es ningún problema lanzar peticiones periódicas al servidor cada 5 segundos; porque, aunque sean modelos no muy grandes, las redes neuronales suponen una carga computacional considerable, y sería una irresponsabilidad cargar todos los modelos para que los ejecute un cliente del que no conocemos las prestaciones de su hardware, de este modo repartimos la carga entre cliente y servidor; porque no hemos encontrado un módulo en *Javascript* capaz de calcular los *log mel-filterbank energies* de un audio, que es una de las características de entrada de nuestro reconocedor acústico.

Cabe aclarar que para el modelo textual ha habido que hacer un cribado de los archivos de *embeddings*. Recordemos que utilizamos cuatro de ellos, cada uno de los cuales almacena la representación vectorial de aproximadamente 900000 palabras, ocupando cada uno alrededor de 4.5 GB. Evidentemente, es imposible obtener en tiempo real los *embeddings* de cualquier frase a partir de esos archivos. De hecho, ya sería complicado el solo hecho de cargar los cuatro en memoria. Por ello, los hemos reducido seleccionando solo aquellas palabras que aparecen en el corpus general que se utilizó para el entrenamiento del problema textual. Esto debería ser suficientemente representativo del lenguaje coloquial, ya que estamos hablando de alrededor

de 68000 tuits. Así conseguimos reducir el tamaño de los diccionarios de *embeddings* a más o menos 260 MB, que es un tamaño fácilmente manejable.

Entonces lo que hace la web es, primero, pedir permisos de uso de cámara y micrófono y poner en marcha la adquisición y muestra de las imágenes y el audio que obtiene. En segundo lugar, iniciamos tres rutinas:

1. La primera 2.5 veces por segundo, obtiene un *tracking* facial de la imagen de la cámara utilizando la librería *clmtrackr*[19], recorta la cara utilizando esa información, la redimensiona a 48x48, la convierte a escala de grises y se la pasa como entrada a la red para obtener los resultados del modelo facial.
2. La segunda, va llenando un *buffer* conforme va llegando el audio del micrófono, de forma que cuando el buffer se llena (de 5 a 6 segundos) se lo envía al servidor. El servidor, por una parte, extrae las características MFCC y *log mel-filterbank energies* y las pasa como entrada al clasificador acústico. Por otra parte, transcribe el audio en texto utilizando la librería *speech_recognition*[20] con el transcriptor de Google. Una vez tenemos el texto, lo transforma utilizando los *embeddings* y los pasa al reconocedor textual. Finalmente, los dos resultados se devuelven como respuesta al cliente.
3. La tercera, constantemente calcula el resultado del clasificador multimodal combinando los últimos resultados obtenidos de los tres unimodales como se ha explicado en el capítulo anterior. Una vez calculado, muestra los resultados, tanto en un diagrama de barras, como en un panel superpuesto al marco en el que se ve lo que la cámara está captando.

Además, la web proporciona información adicional, mostrando en pequeños contenedores la entrada de la red convolucional (la cara localizada, recortada y en escala de grises) y la curva que describe la onda correspondiente al audio que se envía al servidor para ser analizado. Opcionalmente, también muestra superpuesto al vídeo de la cámara, el *tracking* facial proporcionado por *clmtrackr.* Opciones menores como poner la cámara a pantalla completa u ocultar el menú de opciones también se han añadido.

Por supuesto, la web permite seleccionar el dispositivo de vídeo y de audio a utilizar (por si se tiene más de una cámara o de un micrófono). También permite ajustar los dos parámetros del reconocedor multimodal (peso y umbral). Por último, la web ofrece la funcionalidad de grabar sesiones de detección en directo, ya que los resultados se superponen a la vista de la cámara. Así, por ejemplo, un psicólogo podría dejar abierta la aplicación grabando una sesión de terapia para, al acabar, descargar el vídeo con la evolución emocional del paciente a lo largo de la sesión anotada en él.

[19] https://github.com/auduno/clmtrackr
[20] https://github.com/Uberi/speech_recognition

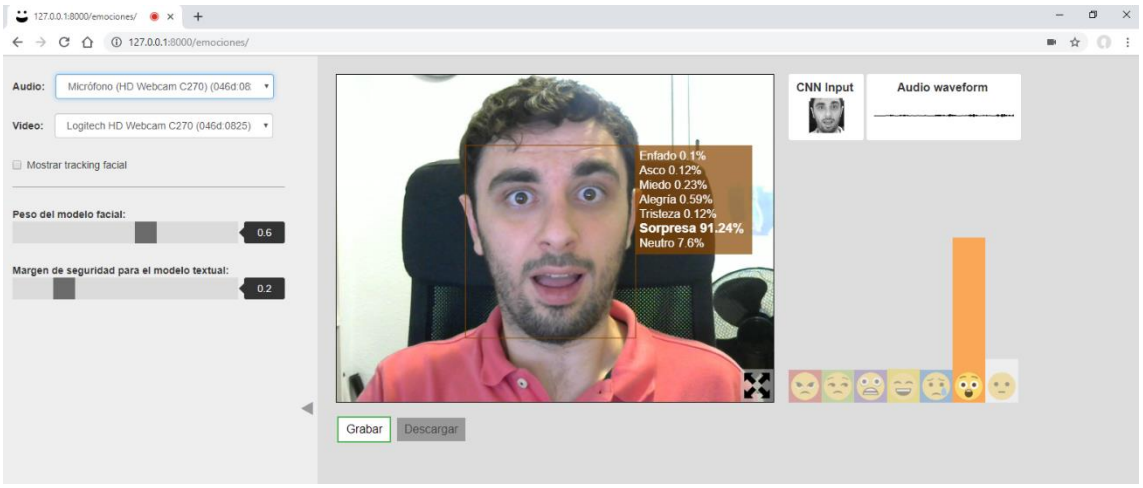


Figura 34: Interfaz de la aplicación

El código completo se encuentra disponible en mi *GitHub*[21]. Para poner en marcha la aplicación, basta con descargarla, tener instalados los módulos que utiliza y, desde una consola dentro de la carpeta principal, lanzar la orden "python manage.py runserver –noreload". La aplicación entonces se puede probar desde el navegador en la dirección http://127.0.0.1:8000/emociones/.

Por último, una versión *lite* de la aplicación se ha subido a la plataforma como servicio *Heroku* y está disponible al público[22] para poder usarla sin descargar ni instalar nada. Esta versión solo incluye el clasificador facial porque el plan gratuito de *Heroku* solo incluye 256 MB de memoria RAM, por lo que no es posible cargar en memoria y utilizar los modelos acústico y textual, ni los *embeddings*.

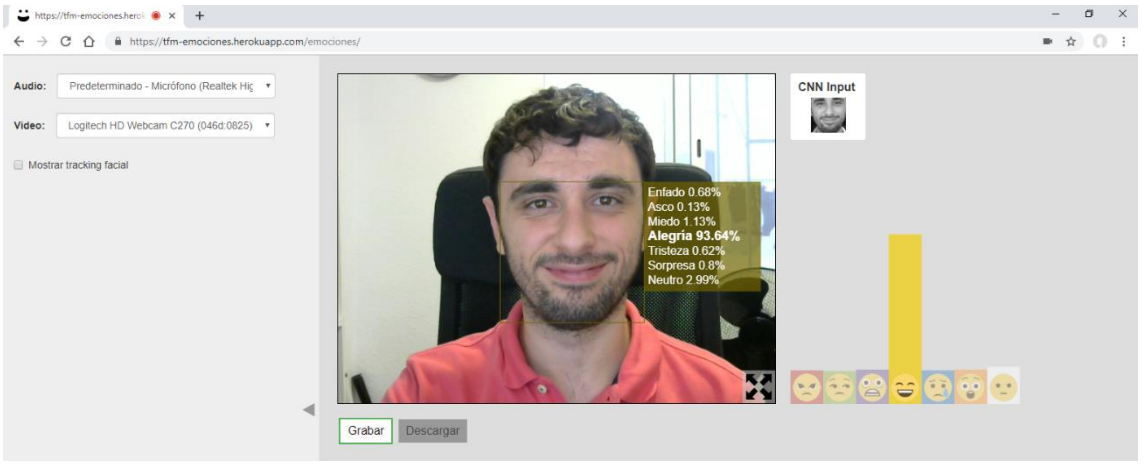


Figura 35: Interfaz de la versión *lite* disponible en *Heroku*

4.1 Análisis del marco legal y ético

Dado el potencial de esta herramienta para realizar experimentos con humanos, en los que se hacen grabaciones de los individuos, vale la pena aclarar algunas consideraciones legales y éticas al respecto. Aunque efectivamente se graba al usuario, las imágenes obtenidas para la detección de la emoción en ningún momento abandonan el cliente ni se almacenan de modo alguno, ya que la detección se hace en el propio navegador y no existe ninguna necesidad de guardar las

[21] https://github.com/jofuelo/reconocedor_multimodal_emociones
[22] https://tfm-emociones.herokuapp.com/emociones/

imágenes. Por otra parte, el audio sí es enviado al servidor y este se almacena brevemente en un formato específico para su transcripción. Sin embargo, este archivo se elimina en cuanto finaliza su transcripción, además de que no contiene ningún tipo de dato personal o información sobre la identidad del hablante más allá de la propia señal acústica. Ninguna información personal o dato adicional es almacenado en modo alguno y la aplicación no cuenta con una base de datos para tal propósito.

Por supuesto, si en trabajos futuros resultara deseable guardar información para obtener algún tipo de estadísticas o aprendizaje adicional personalizado, se informaría a todos los usuarios de esto, y se les daría la opción de aceptarlo o de solicitar que no se almacenen datos sobre ellos.

Teniendo en cuenta los datos con los que se trabaja en el proyecto, podemos decir que se cumple la normativa del Reglamento General de Protección de Datos (RGPD) [63].

CAPÍTULO 5

Conclusiones

Finalmente, se ha conseguido desarrollar un sistema multimodal de reconocimiento de emociones y se ha integrado en una aplicación en *Django* que puede usarse en directo, pero que también permite grabar sesiones de detección. Además, podemos considerar que es un reconocedor robusto, ya que hemos logrado un clasificador facial con una precisión en test de un 87.65±0.82% con invarianza a la iluminación, la edad o el origen. También hemos conseguido un reconocedor acústico en español con porcentaje de acierto del 98.90±0.68%. Y un reconocedor de texto como *ensemble* de dos redes neuronales con una precisión del 75.14±0.84%, teniendo en cuenta que está entrenado y probado con tuits que suelen tener palabras mal escritas, *hashtags*, etc. que no vamos a encontrar en el audio transcrito. Por todo ello podemos concluir que hemos cumplido todos los objetivos que nos planteamos al inicio ya que hemos desarrollado los sistemas que planteamos, superando en todos los casos los objetivos de precisión que establecimos al inicio del proyecto.

Como observación quizás subjetiva, cabría añadir que fuimos invitados a la 8ª feria de inventos que organiza la Universitat Politècnica de València, que se celebró el pasado domingo 16 de junio. Durante todo el día comprobamos un par de datos bastante alentadores sobre el reconocedor facial, que quizás sea el de más importancia de los tres: que incluso a contraluz se comportaba bastante bien (peor que con buena iluminación, pero bastante bien) y que funcionaba correctamente en un amplio rango de edades, desde niños pequeños (4 años en adelante) hasta gente bastante mayor.

Por último, destacar que el trabajo, de cualquier índole, es muy importante para el aprendizaje. En concreto, este proyecto me ha ayudado a reforzar y mantener mis conocimientos de programación y *deep learning* a varios niveles y, especialmente, en la biblioteca *Keras*, lo cual es de agradecer, ya que es una tecnología muy puntera hoy en día y sobre la que tenía poca experiencia. En menor medida, también he aprendido bastante sobre el *framework Django*, lo cual es muy útil debido a que *Python* es uno de los lenguajes de programación más populares actualmente.

5.1 Trabajo futuro

A partir de aquí, las posibilidades son muchísimas. En primer lugar, sería ideal poder recabar un corpus de vídeo etiquetado con emociones en español para poder evaluar nuestro sistema multimodal y comprobar que efectivamente supera las limitaciones de sus componentes unimodales.

Además, las tres entradas que hemos considerado no son, evidentemente, las únicas expresiones con carga emocional. Añadir más modalidades a nuestro clasificador, como la postura corporal, el ritmo cardíaco o la conductividad de la piel podría ser realmente interesante y abrirnos la puerta a reconocedores de emociones que pudieran rivalizar con el ser humano.

Por otra parte, sería valioso explorar también otros modelos de emociones ya que, aunque el utilizado es muy popular y está muy establecido en la literatura, es limitado. Quizás modelos

multidimensionales continuos como el PANAS podrían acercarse más al inmenso espectro que es la sensibilidad humana.

En otra dirección, también podríamos integrar este tipo de reconocedores junto con sistemas multiagentes, de forma que creemos agentes que sean capaces de reconocer nuestras emociones y actuar en consecuencia, teniéndolas en cuenta, de forma que les afecten. Este es un paso clave en la creación de agentes artificiales más humanos.

En conclusión, este es un campo con muchas posibilidades y, aunque se ha desarrollado un trabajo considerablemente complejo y completo, hay mucho que aún se puede hacer.

Bibliografía

[1] B. Alfonso, E. Vivancos and V. Botti, An Open Arquitecture for Affective Traits in a BDI Agent, Proceedings of the 6th ECTA 2014. Part of the 6th IJCCI 2014 (2014), págs. 320-325, 2014.

[2] N. Krämer, B. Tietz and G. Bente, Effects of Embodied Interface Agents and Their Gestural Activity, vol. 2792, 2003, pp. 292-300.

[3] J.-J. Cabibihan, H. Javed, M. Jr and S. Mariam Aljunied, Why Robots? A Survey on the Roles and Benefits of Social Robots in the Therapy of Children with Autism, vol. 5, 2013, p. .

[4] S. March and G. Smith, Design and Natural Science Research on Information Technology, vol. 15, 1995, pp. 251-266.

[5] R. W. Picard and R. Picard, "Affective computing. Vol. 252," *MIT press Cambridge,* vol. 29, no. 13-14, pp. 1729-1735, 1997.

[6] C. Darwin, The expression of the emotions in man and animals, [Culture et civilisation], Ed., 1969.

[7] P. Ekman, Universals and cultural differences in facial expressions of emotion, University of Nebraska Press, Ed., [Lincoln], 1971.

[8] P. Ekman, Basic Emotions, M. Power, Ed., Sussex, UK: John Wiley & Sons, 1999.

[9] R. Plutchik, "A psychoevolutionary theory of emotions," *Social Science Information,* vol. 21, no. 4-5, pp. 529-553, 1982.

[10] D. Cowen, A., Elfenbein, H., Laukka, P., & Keltner, "Mapping 24 emotions conveyed by brief human vocalization.," *American Psychologist,* 2018.

[11] W. Parrott, Emotions in social psychology, Psychology Press, Ed., Philadelphia, Pa., 2001.

[12] J. Russell, "A circumplex model of affect.," *Journal of Personality and Social Psychology,* vol. 39, no. 6, pp. 1161-1178, 1980.

[13] J. Mehrabian, A., & Russell, An approach to environmental psychology, M.I.T. Press, Ed., Cambridge, Mass., 1976.

[14] D. Watson, L. Clark and A. Tellegen, "Development and validation of brief measures of positive and negative affect: The PANAS scales.," *Journal of Personality and Social Psychology,* vol. 54, no. 6, pp. 1063-1070, 1988.

[15] J. Lyons, M., Budynek, "Automatic classification of single facial images," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 21, no. 12, pp. 1357-1362, 1999.

[16] M. Stewart Bartlett, G. Littlewort, I. Fasel and J. Movellan, Real Time Face Detection and Facial Expression Recognition: Development and Applications to Human Computer Interaction, vol. 5, 2003, p. 53.

[17] K. B. Vinay and B. S. Shreyas, Face Recognition Using Gabor Wavelets, 2006, pp. 593-597.

[18] C. Wei-lun, "Gabor wavelet transform and its application," 2010.

[19] C. Shan, S. Gong and P. Mcowan, Robust facial expression recognition using local binary patterns, vol. 2, 2005, pp. II-370.

[20] I. Kotsia, I., & Pitas, "Facial Expression Recognition in Image Sequences Using Geometric Deformation Features and Support Vector Machines," *IEEE Transactions on Image Processing,* vol. 16, no. 1, pp. 172-187, 2007.

[21] P. Raj Dachapally, Facial Emotion Detection Using Convolutional Neural Networks and Representational Autoencoder Units, 2017, p. .

[22] S. Alizadeh and A. Fazel, Convolutional Neural Networks for Facial Expression Recognition, 2017, p. .

[23] C. Min Lee, S. Narayanan and R. Pieraccini, Combining acoustic and language information for emotion recognition., 2002, p. .

[24] B. Schuller, R. Jiménez Villar, G. Rigoll and M. Lang, Meta-Classifiers in Acoustic and Linguistic Feature Fusion-Based Affect Recognition, vol. 1, 2005, pp. 325-328.

[25] B. Schuller, G. Rigoll and M. Lang, Hidden Markov Model-based Speech Emotion Recognition, vol. 2, 2003, pp. 401-404.

[26] A. Balakrishnan and A. Rege, "Reading Emotions from Speech using Deep Neural Networks," 2017.

[27] Y. Niu, D. Zou, Y. Niu, Z. He and H. Tan, A breakthrough in Speech emotion recognition using Deep Retinal Convolution Neural Networks, 2017, p. .

[28] M. Malmasi, S., & Zampieri, "Detecting Hate Speech in Social Media," *RANLP 2017 - Recent Advances in Natural Language Processing Meet Deep Learning,* 2017.

[29] S. Bharti, B. Vachha, R. Pradhan, K. Babu and S. Jena, "Sarcastic sentiment detection in tweets streamed in real time: a big data approach," *Digital Communications and Networks,* vol. 2, no. 3, pp. 108-121, 2016.

[30] S. Ahmed, S. Hina and R. Asif, "Detection of Sentiment Polarity of Unstructured Multi-Language Text from Social Media," *International Journal of Advanced Computer Science and Applications,* vol. 9, no. 7, 2018.

[31] J. González Barba, L. Hurtado Oliver and F. Pla, ELiRF-UPV at TASS 2018: Sentiment Analysis in Twitter based on Deep Learning, 2018, p. .

[32] L. Hurtado Oliver, F. Pla and J. González Barba, ELiRF-UPV at TASS 2017: Sentiment Analysis in Twitter based on Deep Learning, 2017, p. .

[33] L. Hurtado Oliver and F. Pla, ELiRF-UPV en TASS 2016: Análisis de Sentimientos en Twitter., 2016.

[34] S. D'Mello and J. Kory, Consistent but modest: A meta-analysis on unimodal and multimodal affect detection accuracies from 30 studies, 2012, p. .

[35] P. Tzirakis, G. Trigeorgis, M. Nicolaou, B. Schuller and S. Zafeiriou, End-to-End Multimodal Emotion Recognition Using Deep Neural Networks, vol. PP, 2017, p. .

[36] S. ul haq and P. Jackson, Multimodal Emotion Recognition, 2010, p. .

[37] Chung-Hsien Wu and Wei-Bin Liang, "Emotion Recognition of Affective Speech Based on Multiple Classifiers Using Acoustic-Prosodic Information and Semantic Labels," *IEEE Transactions on Affective Computing,* vol. 2, no. 1, pp. 10-21, 2011.

[38] G. Caridakis, G. Castellano, L. Kessous, A. Raouzaiou, L. Malatesta, S. Asteriadis and K. Karpouzis, Multimodal emotion recognition from expressive faces, body gestures, vol. 247, 2007, p. .

[39] S. Poria, H. Peng, A. Hussain, N. Howard and E. Cambria, "Ensemble application of convolutional neural networks and multiple kernel learning for multimodal sentiment analysis," *Neurocomputing,* vol. 261, pp. 217-230.

[40] T. Kanade, J. Cohn and Y. Tian, Comprehensive Database for Facial Expression Analysis, 1970, p. .

[41] P. Lucey, J. Cohn, T. Kanade, J. Saragih, Z. Ambadar and I. Matthews, The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression, 2010, pp. 94-101.

[42] M. Lyons, S. Akamatsu, M. Kamachi and J. Gyoba, Coding Facial Expressions with Gabor Wavelets, vol. 1998, 1998, pp. 200-205.

[43] D. Lundqvist, A. Flykt and A. Öhman, The Karolinska Directed Emotional Faces – KDEF, CD ROM from Department of Clinical Neuroscience, Psychology section, 1998, pp. 91-630.

[44] A. S. Georghiades, P. Belhumeur and D. Kriegman, From Few to Many: Illumination Cone Models for Face Recognition Under Variable Lighting and Pose, vol. 23, 2001, pp. 643-660.

[45] A. Raghuvanshi and V. Choksi, "Facial Expression Recognition with Convolutional Neural Networks," 2016.

[46] I. Goodfellow, D. Erhan, P. Luc Carrier, A. Courville, M. Mirza and B. Hamner, "Challenges in representation learning: A report on three machine learning contests," *Neural Networks,* vol. 64, pp. 59-63, 2015.

[47] E. Barsoum, C. Zhang, C. Canton Ferrer and Z. Zhang, Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution, 2016, pp. 279-283.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going Deeper with Convolutions, 2014, p. .

[49] K. He, X. Zhang, S. Ren and J. Sun, Identity Mappings in Deep Residual Networks, vol. 9908, 2016, pp. 630-645.

[50] G. Huang, Z. Liu, L. van der Maaten and K. Weinberger, Densely Connected Convolutional Networks, 2017, p. .

[51] G. Mahesh and N. Patel, Recognition of Facial Expressions using Local Mean Binary Pattern, vol. 16, 2017, p. 54.

[52] V. Tyagi and C. Wellekens, On desensitizing the Mel-Cepstrum to spurious spectral components for Robust Speech Recognition, vol. 1, 2005, pp. 529-532.

[53] K. K. Paliwal, Spectral subband centroid features for speech recognition, 1998, pp. 617-620 vol.2.

[54] J. Chen, Y. Arden Huang, Q. Li and K. K. Paliwal, Recognition of Noisy Speech Using Dynamic Spectral Subband Centroids, vol. 11, 2004, pp. 258-261.

[55] E. Franti, I. Ispas, V. Dragomir, M. Dascalu, E. Zoltan and I. C. Stoica, "Voice Based Emotion Recognition with Convolutional Neural Networks for Companion Robots," *ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY,* vol. 20, no. 3, pp. 222-240, 2017.

[56] L. Kerkeni, Y. Serrestou, M. Mbarki, K. Raoof and M. Mahjoub, Speech Emotion Recognition: Methods and Cases Study, 2018, pp. 175-182.

[57] P. Bojanowski, E. Grave, A. Joulin and T. Mikolov, Enriching Word Vectors with Subword Information, vol. 5, 2016, p. .

[58] J. Pennington, R. Socher and C. Manning, Glove: Global Vectors for Word Representation, vol. 14, 2014, pp. 1532-1543.

[59] Y. Zhang and B. Wallace, A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification, 2015, p. .

[60] J. Taverner, E. Vivancos and V. Botti, Towards a Computational Approach to Emotion Elicitation in Affective Agents, 2019, pp. 275-280.

[61] A. Mehrabian and S. Ferris, "Inference of attitudes from nonverbal communication in two channels.," *Journal of Consulting Psychology,* vol. 31, no. 3, pp. 248-252, 1967.

[62] M. Mehrabian, A., & Wiener, "Decoding of inconsistent communications.," *Journal of Personality and Social Psychology,* vol. 6, no. 1, pp. 109-114.

[63] Boletín Oficial del Estado, Ley orgánica 3/2018, de 5 de diciembre, de protección de datos personales y garantía de los derechos digitales, https://boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf, 12 2018.