ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

UNIVERSIDAD POLITÉCNICA DE MADRID

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL

REDES NEURONALES CONVOLUCIONALES PROFUNDAS PARA EL RECONOCIMIENTO DE EMOCIONES EN IMÁGENES

AUTOR: ÓSCAR PICAZO MONTOYA
TUTOR: LUIS BAUMELA MOLINA

JUNIO, 2018

A Alicia,
amor, amiga y compañera,
que me acompaña en este y otros proyectos

Y por supuesto a Daniel,
nuestro hijo,
con el que hemos vivido y viviremos
los mejores momentos.

AGRADECIMIENTOS

Antes de comenzar la exposición de esta memoria, me gustaría incluir una serie de agradecimientos hacia las personas que han hecho posible la realización de este trabajo.

En primer lugar, me gustaría agradecer a mi tutor, Luis Baumela, aceptar la propuesta de este trabajo fin de máster, así como el apoyo durante la realización del mismo.

A todos los grupos de investigación, entidades públicas y privadas que financian proyectos de investigación, crean utilidades, librerías, conjuntos de datos, que permiten a otras personas y entidades avanzar más rápidamente y crear nuevos proyectos que ayudan al avance de la ciencia, la ingeniería y a su repercusión en todas las personas.

Agradecer a mis hermanos y todos los compañeros en mi anterior proyecto de trabajo *Milenium-Nexium*, el tiempo y trabajo compartido sin el cual no habría sido posible el embarcarme en esta nueva etapa de estudios que concluye con este trabajo.

Por último, agradecer a mi familia y amigos todo su apoyo prestado día a día durante la duración del trabajo, sin el cual, habría sido imposible finalizarlo.

RESUMEN

Las redes neuronales convolucionales profundas (DCNNs, del inglés Deep Convolutional Neural Networks) han demostrado su capacidad para resolver problemas de clasificación de imágenes utilizando un modelo jerárquico, millones de parámetros, aprendiendo con grandes bases de datos. En este proyecto se ha estudiado el comportamiento de este tipo de redes con la búsqueda de emociones en imágenes de caras humanas. Se han usado y contrastado diferentes DCNNs que han funcionado anteriormente bien en otros ámbitos, y se han mejorado los resultados. Para ello se ha utilizado una base de datos de reciente aparición denominada RAF-DB, la cual está compuesta por 15.339 imágenes de caras descargadas de Internet totalmente clasificadas. También se han buscado otros conjuntos de datos como son CK+ y las usadas para el reconocimiento de expresiones en el reto ICML 2013 (FER2013) y, sin realizar ningún ajuste sobre las redes previamente entrenadas, se ha comprobado su comportamiento. Como último se ha realizado una aplicación móvil que puede usar cualquiera de las redes entrenadas como ejemplo de uso.

ABSTRACT

Deep Convolutional Neural Networks have demostrated their capacity to solve classification problems using a hierarchical model, millions of parameters, and learning with big databases. In this project, the behavior of these type of networks has been studied for estimating emotions in human faces images. Different DCNNs that have worked fine in different areas have been used and contrasted, and the results have been improved. For that purpose, a recent dataset, RAF-DB, has been used. It has 15.339 totally classified human face images downloaded from the Internet. Other databases have also been searched, CK+ and the database used in ICLM 2013 challenge (FER2013), without previous trained networks finetuning or training, checking their results. Finally, we have also developed a mobile application that may use any of the networks trained in this project.

TABLA DE CONTENIDOS

LISTADO DE FIGURAS

LISTADO DE TABLAS

1. INTRODUCCIÓN.

El estudio de las expresiones y emociones faciales tiene múltiples aplicaciones y se ha estudiado para seguridad [32], observando la respuesta de un individuo durante entrevistas o interrogatorios, o en el reconocimiento de caras [29]. También se ha utilizado en seguridad en la conducción [42], estudiando la frecuencia de parpadeo, cierre de ojos, bostezos. En el ámbito de la robótica, se ha utilizado el reconocimiento de expresiones faciales para la interacción entre humanos y robots [21][40]. En el campo del *Neuromarketing* [12], otras posibles aplicaciones serían las de estudiar la respuesta de los clientes ante nuevas imágenes de marca, nuevos empaquetados de los productos y nuevas campañas de *marketing*. Por último, en salud, se ha estudiado la detección automática del dolor a partir de las expresiones faciales [27].

Los algoritmos de reconocimiento de expresiones y emociones suelen organizarse en tres etapas. La primera consiste en la detección de las caras que se encuentran en la imagen. Una vez detectadas, se analizan una por una entrando en la segunda etapa. Su objetivo es la de extraer las características del rostro. Para ello se han utilizado varios métodos, el más antiguo son los métodos basados en apariencia [28], por ejemplo, mediante el uso de patrones binarios locales [33]. Posteriormente se ha trabajado en métodos basados en extraer características del rostro obteniendo las unidades de acción [43][3] (*action units* en inglés). Últimamente, al igual que el trabajo realizado en este proyecto, se están utilizando redes neuronales convolucionales profundas para extraer características de las caras, ya que en otras tareas relativas al reconocimiento en imágenes se han comportado de una forma bastante eficiente.

La última fase del reconocimiento consiste en la clasificación de esas características para obtener la emoción correspondiente. En esta fase se han utilizado diferentes métodos como SVM, vecinos cercanos, LDA o DBN.

Estudios más recientes han trabajado en la detección de las variaciones y tonalidades del color en las caras para poder reconocer emociones [2].

En relación las emociones, éstas se pueden dividir en básicas y compuestas. Las emociones básicas son: sorprendido, asustado, asqueado, feliz, triste, enfadado y neutral. En relación a las emociones compuestas, no está muy claro el número o definición de ellas, así en algunos casos definen hasta 12 diferentes [35], en otros casos 15 [11], o 17 [10]. Es por esto que en este proyecto se ha decidido trabajar únicamente con las emociones básicas.

Existen dos campos diferentes donde se han estudiado las emociones en imágenes, por un lado en laboratorios [26], donde las emociones se encuentran controladas por expertos, las imágenes son de calidad y por tanto, en principio, es más fácil encontrar características para cada una de las emociones que permitan diferenciarlas. Por otro lado, imágenes

descargadas de Internet, donde las emociones no están controladas y son imágenes de poca calidad. En este último caso en inglés es común encontrarlas con la denominación "*in the wild*"[35][23].

1.1. OBJETIVOS

El objetivo por tanto a cubrir en el proyecto consiste en estudiar el comportamiento de diferentes redes neuronales convolucionales profundas en el reconocimiento de emociones. Para ello se partirá del conjunto de datos RAF-DB [35], se comprobará cual es la mejor red que se comporta con esta base de datos y se intentará encontrar métodos que mejoren los resultados. Una vez se consiga el mejor comportamiento de la red, se aplicarán estos cambios en cada una de las redes diferentes y se compararán incluyendo el estado de arte actual de reconocimiento de este conjunto de datos.

Por último, se buscarán otros conjuntos de datos y, sin reentrenar o ajustar las redes (*finetuning*), se comprobará el comportamiento de cada una de ellas.

1.2. ESTADO DEL ARTE

Los sistemas automáticos de reconocimiento de emociones faciales se pueden dividir en convencionales y en los basados en aprendizaje profundo (tal y como se describen en [24]).

En relación con los sistemas convencionales, existen los basados en características geométricas, en características de apariencia y los híbridos de ambos.

Los sistemas basados en características geométricas buscan la relación entre componentes faciales, por ejemplo, de posición y ángulo de diferentes puntos de referencia de la cara. Como ejemplo, en [37], se utilizan los modelos estadísticos ASM (del inglés *Active Shape Models*) para medir las características geométricas de los puntos de referencia del rostro, y como clasificador se ha utilizado SVM. En la figura 1.1 se puede ver algunas características utilizadas para el reconocimiento.

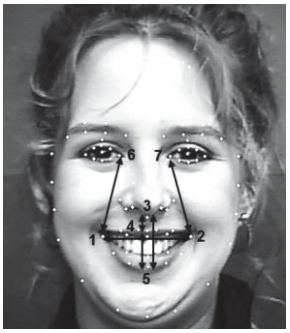

Figura 1.1. Ejemplo de puntos de referencia y algunas características geométricas usadas para el reconocimiento.

En cuanto a los sistemas basados en características de apariencia, se busca dividir el rostro en diferentes regiones, dándole diferente importancia a cada una de ellas y se comprueba la información que contiene. En [17] se ha utilizado un histograma de LBP (del inglés Local Binary Pattern) por cada bloque de diferente tamaño en el que se divide la cara bajo estudio. Como clasificador se ha utilizado PCA (del inglés *Principal Component Analysis*). Un ejemplo de esta técnica se puede ver en la figura 1.2.

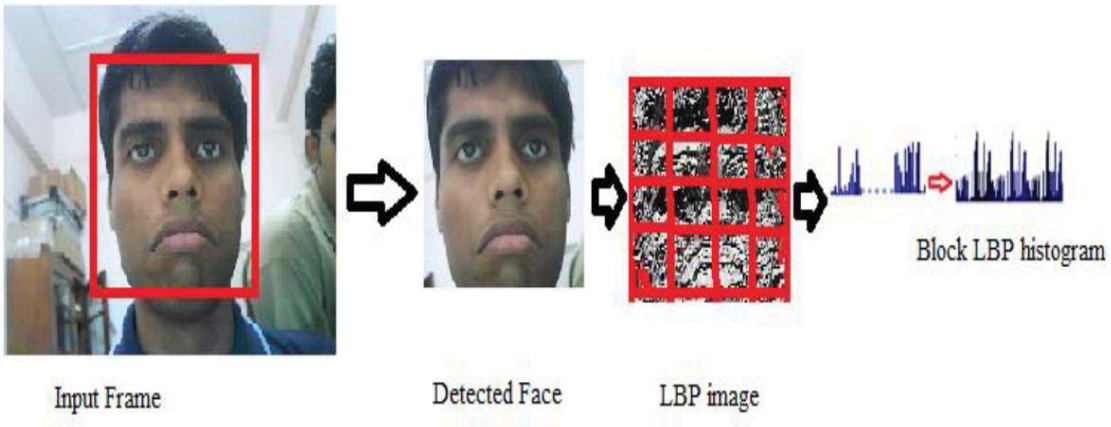

Figura 1.2. Cálculo del histograma LBP por bloque.

Ejemplos de sistemas híbridos combinan los métodos usados en los dos sistemas anteriores, así, por ejemplo, en [4] se utiliza la distancia euclídea entre los puntos de referencia normalizados, los ángulos formados por los triángulos generados a partir de estos puntos y por último un filtro *Garbor*. Como método de clasificación se utiliza KSDA (del inglés *Kernel Subclass Discriminant Analysis*).

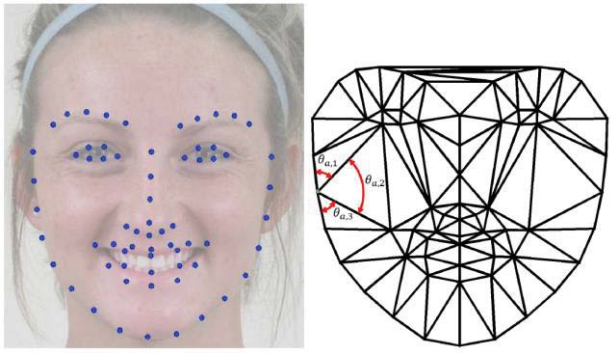

Figura 1.3. Puntos de referencia normalizandos y triangulación utilizados en [4].

Todos estos sistemas se pueden utilizar para estudiar imágenes estáticas o bien secuencias de vídeo, para estos últimos, se generan características dinámicas de desplazamiento entre el fotograma actual y los anteriores.

También se han utilizado imágenes de infrarrojos con el objetivo de evitar los problemas de iluminación de las imágenes convencionales. Así, en [34] se realiza una segmentación de la imagen en regiones comprobando la diferencia de temperatura entre secuencias. Mediante el algoritmo Adaboost y KNN (del inglés k-Nearest Neighbor) se han clasificado las diferentes expresiones faciales.

Otros estudios han incorporado el sensor de profundidad del sensor Kinect de Microsoft. En [39] se ha utilizado un sistema basado en características de apariencia mediante el estudio de regiones locales. Como clasificador se ha usado SVM. No se ha utilizado información de imágenes de la cámara, solamente la información de profundidad proporcionada por el sensor.

Los sistemas basados en aprendizaje profundo son los otros sistemas automáticos de reconocimiento que se están utilizando en los últimos años. En primer lugar, los basados en redes neuronales convolucionales, los cuales toman como entrada una imagen o un mapa de características, convolucionan esas entradas con un conjunto de filtros cuyos pesos se encuentran previamente entrenados y se obtienen unos mapas de características que representan su disposición espacial en las imágenes. Como segundo elemento, se tienen capas de *pooling* que se encargan de reducir las dimensiones de los datos obtenidos en la capa de convolución, tratando de ignorar variaciones pequeñas y distorsiones geométricas. El último elemento que se utiliza son las capas neuronales totalmente conectadas que se encargan de calcular los resultados por cada una de las clases que se desean reconocer. La mayoría de estos métodos se han utilizado para detectar las unidades de acción o AU (del inglés *Action Unit*). En [6] se emplean este tipo de redes para comprobar su comportamiento con diferentes bases de datos.

Uno de los problemas que se ha encontrado en el uso de las CNN, es que no son capaces de reflejar las variaciones temporales en los componentes faciales. Para solucionarlo, se han utilizado las redes LSTM (del inglés *Long Short-Term Memory*), un tipo de redes neuronales recurrentes (RNN del inglés *Recurrent Neural Network*), las cuales permiten el análisis y reconocimiento de secuencias de vídeo. Como ejemplo de combinación de CNN con LSTM para el reconocimiento de emociones en secuencias de vídeo se puede ver el resultado de [8], donde la representación espacial obtenida mediante CNN se combina con la salida de una LTSM para obtener la predicción de 12 AUs.

El principal problema de los sistemas automáticos basado en aprendizaje profundo es la cantidad de datos necesarios para el entrenamiento.

2. DESARROLLO.

2.1. ENTORNOS DE TRABAJO PARA REDES NEURONALES CONVOLUCIONALES.

En la actualidad existen muchos entornos o herramientas de desarrollo que permiten trabajar con redes neuronales convolucionales (además de otros modelos de aprendizaje automático y/o aprendizaje profundo). Como ejemplos se pueden enumerar:

- Caffe: Desarrollado por la Universidad de Berkeley.
- Caffe2: Sucesor de Caffe soportado por Facebook.
- TensorFlow: Soportado por Google.
- Torch: Soportado por ingenieros de Facebook, Twitter y Google. Desarrollado en lenguaje *Lua*.
- PyTorch: Una versión de Torch para Python soportada por Facebook.
- CNTK: Soportado por Microsoft.
- DSSTNE: Soportado por Amazon.

Se puede encontrar en [9] más información y una comparativa exhaustiva de estos y otros entornos de trabajo existentes en la actualidad.

Para este trabajo se descartaron todos excepto *caffe* [22] y *TensorFlow* [1], y dentro de estos dos el elegido fue *caffe*. La elección fue debido a la facilidad de encontrar modelos ya entrenados de redes neuronales convolucionales existentes, facilidad para entrenar únicamente algunas capas en modelos ya entrenados, se permite el entrenamiento de redes sin escribir código fuente, y por último la facilidad de utilizar los modelos con la librería de desarrollo OpenCV [5], la cual es simple de integrar en una aplicación móvil tanto en dispositivos Android como en dispositivos iOS.

2.2. ENTORNO DE TRABAJO *CAFFE*.

Lo primero para poder utilizar este entorno de trabajo es, o bien obtener los binarios genéricos compatibles con el equipo en el que se vaya a usar y utilizarlo directamente, o bien obtener el código fuente, compilarlo y posteriormente utilizarlo. La última opción es la mejor para aprovechar al máximo las características del equipo o equipos que se van a utilizar para la ejecución de todas las pruebas y es la que se utilizó en este caso.

Los ficheros fuente se obtuvieron utilizando la función de clonación de Git que se pueden ver en [13]. Se eligió el sistema operativo Ubuntu LTS 16.04 para la compilación completa de c*affe*. La compilación requiere la instalación de diferentes librerías utilizadas

por el entorno, que una vez instaladas permite tener el entorno operativo para poder trabajar.

2.3. ENTRENAMIENTO DE REDES NEURONALES CON *CAFFE*.

Para el entrenamiento de las redes neuronales se necesitan dos ficheros, uno de configuración del entrenamiento y otro que define la red neuronal, sus capas y los datos de entrenamiento y chequeo. Los nombres de estos ficheros de configuración no son importantes, se pueden nombrar de cualquier forma.

2.3.1. Fichero de configuración de entrenamiento.

Este fichero normalmente se nombra como "solver.prototxt". Se trata de un fichero de texto donde se especifican diferentes valores de configuración. Algunos de los valores más importantes son:

- *base_lr*: Especifica la tasa de aprendizaje inicial de entrenamiento para la red. Es un número real y dependiendo de la red este valor puede ser de 0,001 o menor.

- *display*: Cada cuántas iteraciones de entrenamiento la utilidad muestra el resultado por pantalla (también lo utiliza para escribir en los ficheros de traza).

- *gamma*: Este parámetro indica cuánto debe cambiar la tasa de aprendizaje cuando se llega al siguiente paso. Se trata de un número real.

- *lr_policy*: Este parámetro indica cómo debe cambiar la tasa de aprendizaje en el tiempo. Puede tomar los siguientes valores:

  · "*step*": Reduce la tasa de aprendizaje indicado por el valor "*gamma*".

  · "*multistep*": Reduce la tasa de aprendizaje indicado por el valor "*gamma*" cada cierto número de iteraciones indicadas en "*stepvalue*".

  · "*fixed*": La tasa de aprendizaje permanece constante.

  · "*poly*": La tasa de aprendizaje sigue un decremento polinomial hasta llegar a 0 al iterar "*max_iter*" veces siguiendo la siguiente fórmula:

$$base\_lr \cdot \left(1 - \frac{iter}{\max\_iter}\right)^{power}$$

· "*Sigmoid*": La tasa de aprendizaje sigue un decremento sigmoidal siguiendo la siguiente fórmula:

$$base\_lr \cdot \frac{1}{1 + e^{-gamma \cdot (iter - stepsize)}}$$

Los valores *base_lr, max_iter, power, gamma* y *stepsize* son valores dados en este fichero de configuración (unos se han explicado ya, y otros se explicarán más adelante). *iter* es el número de iteración actual en el algoritmo.

- *max_iter*: Este valor indica cuándo el entrenamiento de la red debe detenerse. El valor es un número entero.

- *momentum*: Este valor indica cuánto del peso previo será retenido en el nuevo cálculo. Este valor es un número real.

- *net*: Especifica el nombre completo del fichero de configuración de la red neuronal (este fichero se verá más adelante).

- *snapshot*: Este valor indica cada cuantas iteraciones *caffe* debe guardar un modelo y su estado. Es un número entero.

- *snapshot_prefix*: Este valor es una cadena de caracteres que indica el prefijo de los nombres de los modelos y estados almacenados.

- *solver_mode*: Este valor puede tomar el valor "GPU" o "CPU". En el primer caso utilizará la GPU para realizar el entrenamiento y en el segundo la CPU. Siempre que sea posible y se disponga de una GPU con memoria suficiente se intentará elegir esta opción ya que es más rápida que si se utiliza únicamente la CPU de la máquina.

- *step_value*: Este valor indica el número de iteraciones que se deben producir para pasar al siguiente paso (reducción de la tasa de aprendizaje). Es un número entero. Puede aparecer varias veces indicando diferentes pasos en el aprendizaje.

- *test_iter*: Este valor indica cuántas iteraciones de test deben ocurrir por intervalo de test (parámetro *test_interval*). Es un número entero.

- *test_interval*: Este valor indica cada cuánto se ejecuta la fase de test de la red.

- *weight_decay*: Este valor especifica el factor de penalización (regulación) para los pesos grandes.

2.3.2. Fichero de configuración de la red neuronal.

En este fichero se define cada una de las capas que componen la red neuronal. Lo primero que aparece en el fichero es un parámetro "name:" y después, en la misma línea, el nombre de la red entrecomillado.

A continuación, se enumerará cada tipo de capa, su configuración y parámetros principales en *caffe* utilizados en este proyecto, así como una breve descripción teórica de la misma.

2.3.2.1. Capa de entrada para el entrenamiento.

Esta capa es usada para la especificación de los datos de entrenamiento. Para la explicación de los parámetros se muestra un ejemplo a continuación:

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  data_param {
    source: "train_db"
    batch_size: 8
    backend: LMDB
  }
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: Con el valor “Data” se indica que es una capa de datos para la red.

- *top*: Indica el nombre de los datos de salida, los nombres “data” y “label” se indican siempre por convención en la capa de datos en modelos de clasificación e indican las imágenes de entrada y su etiqueta correspondiente.

- *include*: Grupo de parámetros donde:

  · *phase*: Con el valor “TRAIN” se indica que los datos son de entrenamiento en la red.

- data_param: Grupo de parámetros, donde:

  · *source*: Especifica la ubicación de los datos.

  · *batch_size*: Indica el número de imágenes que se usarán de forma simultánea. Contra mayor es este valor, menos iteraciones se necesitan para completar un *epoc* o iteraciones completas hasta completar todo el conjunto de entrenamiento. Igualmente, al hacer crecer este valor, se necesita más memoria RAM en el equipo

(CPU) o en la GPU (dependiendo de qué se esté usando para el cálculo). Es un valor importante de ajustar cuando se tiene una tarjeta gráfica con memoria normalmente más limitada aunque preferible para acelerar el entrenamiento global de la red.

· *backend*: Especifica el formato de datos de entrada, el cómo se encuentran almacenadas las imágenes. En este proyecto se ha utilizado siempre el formato "LMDB". Más adelante se explicará la utilidad necesaria para generar este tipo de base de datos.

2.3.2.2. Capa de entrada para el test.

Esta capa de datos de entrada se define exactamente igual que la anterior, solamente que se especifica que la fase es de "*TEST*" en el parámetro "*phase*" del grupo "*include*". A continuación, se muestra un ejemplo de la misma:

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  data_param {
    source: "test_db"
    batch_size: 8
    backend: LMDB
  }
}
```

2.3.2.3. Capa de convolución.

Esta capa realiza el cálculo de convolución de la imagen de entrada con un conjunto de filtros cuyos parámetros se aprenderán. Cada uno de esos filtros generan un mapa de características en la imagen de salida.

Para la explicación de los parámetros se muestra un ejemplo a continuación:

```
layer {
  name: "conv1_1"
  type: "Convolution"
  bottom: "data"
```

```
  top: "conv1_1"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 64
    pad: 1
    kernel_size: 3
    stride: 2
    weight_filler {
        type: "gaussian"
        std: 0.01
      }
      bias_filler {
        type: "constant" # initialize the biases to zero (0)
        value: 0
      }
  }
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: “Convolution”, especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada.

- *top*: Indica el nombre de los datos de salida. Normalmente se suele usar el mismo valor que el nombre de la capa *name*.

- *param*: Grupo de parámetros que afectan a los filtros (este parámetro es opcional e indica cómo afecta el valor de la tasa de aprendizaje):

  · *lr_mult*: multiplicador de la tasa de aprendizaje, si este valor es cero no se tocarán los valores de los pesos y es útil cuando no se quiere entrenar una capa y se desean mantener los pesos iniciales.

  · *decay_mult*: multiplicador del descenso de los pesos.

- *param*: Grupo de parámetros que afectan a los parámetros de *bias*:

  · *lr_mult*: multiplicador de la tasa de aprendizaje, si este valor es cero no se tocarán los valores de los pesos y es útil cuando no se quiere entrenar una capa y se desean mantener los pesos iniciales.

  · *decay_mult*: multiplicador del descenso de los pesos.

- *convolution_param*: Grupo de parámetros, donde:

  · *num_output*: Número de filtros.

  · *kernel_size*: Especifica el alto y ancho de cada filtro. Este parámetro aplicaría el mismo valor al ancho y alto, si se desean dar por separado se debe usar *kernel_h* y *kernel_w* para el alto y ancho respectivamente.

· *stride*: Especifica el intervalo en el que se aplican los filtros a la entrada. Si no se da este valor por defecto es 1.

· weight_filler: Grupo de parámetros que indica como inicializar los parámetros de los filtros. Este parámetro puede no indicarse si la red se ha inicializado con un modelo ya calculado. En caso de no darse ni tener modelo inicial se cargarán con un valor constante de cero. Donde:

- *type*: Indica el tipo de inicialización, puede ser por ejemplo "gaussian" para utilizar una distribución de tipo Gaussiana.

- *std*: Indica el valor de la desviación estándar cuando el *type* es "gaussian". La media por defecto es cero.

· bias_filler: Grupo de parámetros que indica como inicializar los parámetros de los parámetros *bias*. Este parámetro puede no indicarse si la red se ha inicializado con un modelo ya calculado. En caso de no darse ni tener modelo inicial se cargarán con un valor constante de cero. Donde:

- *type*: Indica el tipo de inicialización, puede ser por ejemplo "constant" para utilizar un valor constante.

- *value*: Indica el valor de los parámetros *bias* cuando el *type* es "constant".

2.3.2.4. Capa *ReLu*.

Esta capa siempre se suele colocar inmediatamente después de una de convolución para añadir no linealidad a los datos que provienen de la capa convolucional (ya que ésta realiza una operación lineal). Normalmente se suele calcular como:

$$y = \max(0, x)$$

Aunque en otras ocasiones se puede utilizar un peso en la parte negativa de los valores.

A continuación, se muestra un ejemplo de definición de una capa de este tipo:

```
layer {
  bottom: "conv1_1"
  top: "conv1_1"
  name: "relu1_1"
  type: "ReLU"
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: "ReLU", especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada.

- *top*: Indica el nombre de los datos de salida. Normalmente se suele usar el mismo valor que el de los datos de entrada *bottom*, permitiendo ahorrar en el consumo de memoria.

2.3.2.5. Capa de *pooling*.

Esta capa se suele utilizar para reducir información espacial, consiguiendo reducir sobreajuste (*overfitting*) en la red y algo de invarianza a la traslación.

A continuación, se muestra un ejemplo de definición de una capa de este tipo:

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1_2"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: “Pooling”, especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada.

- *top*: Indica el nombre de los datos de salida.

- *pooling_param*: Grupo de parámetros, donde:

· *pool*: Especifica la operación a realizar entre MAX (máximo valor), AVE (valor medio) y STOCHASTIC.

· *kernel_size*: Especifica el alto y ancho de cada filtro..

· *stride*: Especifica el intervalo en el que se aplican los filtros a la entrada. Si no se da este valor por defecto es 1.

2.3.2.6. Capa totalmente conectada.

Esta capa se suele utilizar al final de las redes convolucionales profundas. A continuación, se muestra un ejemplo de definición de una capa de este tipo:

```
layer {
  name: "fc6"
  type: "InnerProduct"
  bottom: "pool5"
  top: "fc6"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  inner_product_param {
    num_output: 4096
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: “InnerProduct”, especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada.

- *top*: Indica el nombre de los datos de salida.

- *inner_product_param*: Grupo de parámetros, donde:

  · *num_output*: Número de salidas.

Los parámetros *param*, *weight_filler* y *bias_filler* tienen la misma definición que en la capa de convolución.

2.3.2.7. Capa *dropout*.

Esta capa se suele utilizar después de una capa totalmente conectada y se usa para reducir el sobreajuste (*overfitting*). Para ello elimina algunas conexiones ocultas de la red con una probabilidad configurable (normalmente 0,5). Puede encontrarse una definición

completa de esta técnica en [20]. A continuación, se muestra un ejemplo de definición de una capa de este tipo:

```
layer {
  name: "drop6"
  type: "Dropout"
  bottom: "fc6"
  top: "fc6"
  dropout_param {
    dropout_ratio: 0.5
  }
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: “Dropout”, especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada.

- *top*: Indica el nombre de los datos de salida. Normalmente se suele usar el mismo valor que el de los datos de entrada *bottom*, permitiendo ahorrar en el consumo de memoria.

- *dropout_param*: Grupo de parámetros, donde:

  · *dropout_ratio*: Valor de probabilidad para eliminar una conexión oculta.

Los parámetros *param*, *weight_filler* y *bias_filler* tienen la misma definición que en la capa de convolución.

2.3.2.8. Capa de normalización de la respuesta local (LRN).

Esta capa se puede utilizar después de una capa *ReLU*. Conceptualmente trata de emular el concepto de inhibición lateral usado en neurobiología y se refiere a la capacidad de que una neurona excitada domine a sus vecinas.

A continuación, se muestra un ejemplo de definición de una capa de este tipo:

```
layer {
  name: "norm1"
  type: "LRN"
  bottom: "conv1"
  top: "norm1"
```

```
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: “LRN”, especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada.

- *top*: Indica el nombre de los datos de salida.

- *lrn_param*: Grupo de parámetros, donde:

· *local_size*: Número de canales a utilizar (cuando se utiliza a través de los canales) o la longitud de la región cuadrada a utilizar (en caso contrario). El valor por defecto es 5.

· *alpha*: Parámetro de escala (valor por defecto 1).

· *beta*: Parámetro exponente (valor por defecto 5).

· *norm_region*: Con el valor “ACROSS_CHANNELS” (por defecto si no se indica), el cálculo se realiza a través de los canales, “WITHIN_CHANNEL” en caso contrario.

2.3.2.9. Capa de resultados de la fase de test.

Esta capa se utiliza para medir la precisión o exactitud de la red con la base de datos de test. A continuación, se muestra un ejemplo de definición de una capa de este tipo:

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc8_expression"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: “Accuracy”, especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada. Se toma como entrada los datos de las clases en la red (“label” normalmente por convención) y los obtenidos en la última capa.

- *top*: Indica el nombre de los datos de salida.

- *include*: Grupo de parámetros. Donde:

  · *phase*: “TEST” indicando que solamente se usa en esta fase del proceso.

2.3.2.10. Capa *SoftMaxWithLoss*.

Esta es la última capa de la red. Convierte los valores obtenidos como resultado en la red a valores reales en el rango [0, 1]. Resulta en un valor de gradiente numéricamente más estable. A continuación, se muestra un ejemplo de definición de una capa de este tipo:

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8_expression"
  bottom: "label"
  top: "loss"
}
```

Donde:

- *name*: Nombre de la capa entre comillas. Debe ser único en cada capa.

- *type*: “SoftmaxWithLoss”, especificando que es una capa de este tipo.

- *bottom*: Indica el nombre de los datos de entrada. Se toma como entrada los datos de las clases en la red (“label” normalmente por convención) y los obtenidos en la última capa (los mismos que se meten como entrada a la capa de cresultados de la fase de test).

- *top*: Indica el nombre de los datos de salida.

Estos dos ficheros (el último visto de configuración de la red y el anterior de configuración de entrenamiento), son necesarios para el entrenamiento de la red, se necesitará un fichero adicional para usar la red neuronal. Básicamente resume el fichero de configuración de la red neuronal manteniendo la definición de las capas y eliminando la configuración de entrenamiento de las mismas, así como los parámetros de los datos de entrenamiento y chequeo.

2.3.3. Entrenamiento de una red neuronal desde cero.

Esta es la opción que se debe elegir cuando no se tiene ningún modelo previamente entrenado de la misma red. Es especialmente importante en este caso elegir el método de inicialización de pesos ya que es una de las cosas que ayudarán a que la red acabe aprendiendo de forma satisfactoria. El comando a utilizar sería el siguiente:

```
[dir.caffe.build.tools]/caffe train -solver [fichero de configuración de entrenamiento]
```

2.3.4. Entrenamiento de una red neuronal inicializando los pesos.

Cuando se tiene un modelo de red previamente entrenado es una buena práctica utilizar estos pesos para evitar el problema de inicialización de la red. El comando a utilizar sería el siguiente:

```
[dir.caffe.build.tools]/caffe train -solver [fichero de configuración de entrenamiento] -weights [fichero con el modelo precalculado]
```

2.3.5. Entrenamiento de algunas capas de la red neuronal o ajuste (*finetuning*).

Cuando se quiere cambiar alguna capa de la red, se suele modificar el fichero de configuración y cambiar el nombre de la capa o capas (y sus referencias en otras capas) de tal forma que ya no se utilizarán esos pesos del modelo introducido.

Este es la forma que normalmente se utiliza cuando, por ejemplo, el número de clases a utilizar en la capa final "*softmax*" es distinto del que se usó cuando se entrenó la red originalmente. El comando a utilizar sería el siguiente:

```
[dir.caffe.build.tools]/caffe train -solver [fichero de configuración de entrenamiento] -weights [fichero con el modelo precalculado]
```

2.3.6. Continuar entrenamiento si este se interrumpe .

Los entrenamientos de redes neuronales pueden llevar días o incluso varias semanas dependiendo del tamaño y el equipo o equipos donde se están entrenando. El permitir continuar con un entrenamiento detenido, ya sea de forma controlada o bien por diferentes causas no previsibles como cortes de luz, fallos del equipo, o cualquier otro problema, ayuda a ahorrar el tiempo que previamente se ha utilizado en el entrenamiento. El comando que permitiría hacer esto es el siguiente:

```
[dir.caffe.build.tools]/caffe train -solver [fichero de configuración de entrenamiento] -snapshot [fichero de estado previo]
```

2.3.7. Generación de bases de datos de imágenes de entrenamiento y test para la red .

La utilidad de entrenamiento de Caffe utiliza una base de datos de imágenes optimizada para el cálculo. Existe una utilidad que permite crear esta base de datos de imágenes a partir de los ficheros de imagen por separado y un fichero que asocia cada fichero de imagen con la clase correspondiente. El comando es el siguiente:

```
[dir.caffe.build.tools]/convert_imageset -resize_height=[alto] -resize_width=[ancho] [dir.imagenes] [fichero con nombre de imágenes y clases asociadas] [directorio de destino]
```

Los parámetros “resize_height” y “resize_width” son opcionales y permiten reescalar el tamaño de las imágenes de entrada al tamaño deseado. “[dir.imagenes]” representa el directorio donde se encuentran las imágenes. “[fichero con nombre de imágenes y clases asociadas]“ es la ubicación del fichero que contiene cada nombre de fichero de la imagen, uno o varios espacios y el número que especifica la clase a la que corresponde esa imagen. Por último “[directorio de destino]” especifica dónde se almacenarán los ficheros que componen la base de datos de imágenes final.

2.4. BASE DE DATOS DE APRENDIZAJE Y TEST.

Como base de datos de aprendizaje y test se ha elegido RAF-DB, que ha sido creada y utilizada en [35] debido a la cantidad de imágenes, clasificación y la reciente creación. La resolución de las imágenes es de 100x100 píxeles en color y blanco y negro, con una distribución de imágenes por clase que se puede ver en la tabla 2.1.

	Sorprendido	**Asustado**	**Asqueado**	**Feliz**	**Triste**	**Enfadado**	**Neutral**
Entrenamiento	1.290	281	717	4.772	1.982	705	2.524
Test	329	74	160	1.185	478	162	680

Tabla 2.1. Distribución de imágenes por clase de la base de datos de aprendizaje y test.

En la Figura 2.1 se puede ver un ejemplo de las imágenes de entrenamiento de este conjunto de datos.

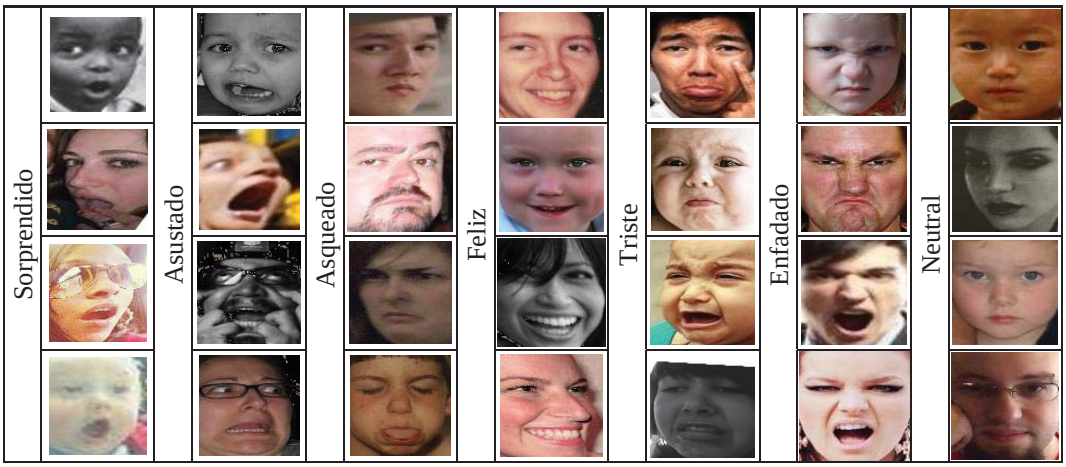

Figura 2.1. Muestra de imágenes de entrenamiento de la base de datos RAF-DB.

2.5. REDES NEURONALES CONVOLUCIONALES PROFUNDAS.

El diseño de una red neuronal convolucional profunda no es una tarea sencilla, la elección del número de capas, tipos de las mismas, conexiones, no sigue un patrón definido, ni existe un proceso que ayude en su definición. Lo que se hace actualmente es utilizar aquellas redes que han demostrado la capacidad de aprendizaje en el reconocimiento de imágenes con una tasa de acierto buena, llegando incluso a mejorar el reconocimiento de los seres humanos. A continuación, se van a describir algunas de las existentes actualmente y que se han utilizado en este proyecto.

2.5.1. AlexNet.

Esta red utilizada es una réplica de la original definida en [25] y se ha obtenido el modelo de la red entrenada para Caffe de [14].

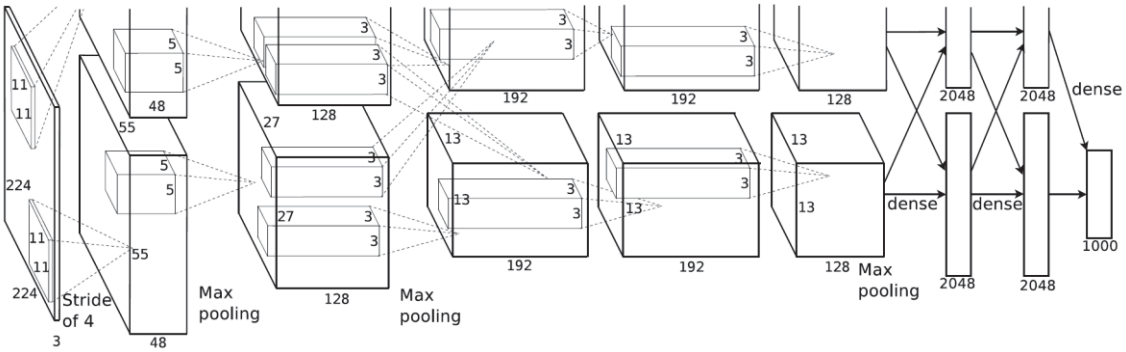

Figura 2.2. Arquitectura de la red neuronal AlexNet.

Tal y como se muestra en la Figura 2.2, se ha realizado el diseño de la red en dos grupos de bloques para aprovechar el uso de dos GPU's en paralelo que se utilizaron originalmente en su entrenamiento. La capa de entrada tiene una dimensión de 224x224x3 (ancho x alto x RGB), cinco capas convolucionales (tres con max. pooling) y tres redes totalmente conectadas. La salida de la última capa totalmente conectada termina en una capa softmax de 1.000 salidas correspondientes a las 1.000 clases para el que se ha preparado. Esta red se utilizó en la competición ILSVRC (*Imagenet Large Scale Visual Recognition Challenge*) [31] que ganó en 2012.

Para reducir el sobreaprendizaje (*overfitting*) se utilizan dos técnicas, por un lado se aumenta el número de imágenes de entrenamiento de la red copiando y transformando las imágenes (mediante la técnica de espejo y mediante las translaciones) y por otro lado utilizando la técnica de *Dropout*. Esta técnica puede ampliarse en [20] y consiste en poner

a 0 con una determinada probabilidad los pesos en algunas neuronas de las capas totalmente conectadas (en concreto, en esta red, se ha utilizado esta técnica en las primeras dos capas totalmente conectadas). El número de parámetros utilizados en esta red es de unos 60 millones.

2.5.2. GoogLeNet / Inception.

Originalmente esta red está definida en [38] y el modelo Caffe usado en el proyecto se ha obtenido de [15].

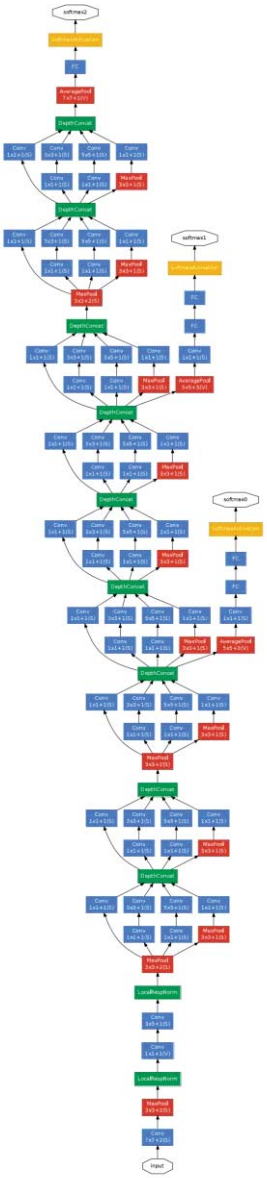

Figura 2.3. Arquitectura de la red GoogLeNet.

En la figura 2.3 puede verse la arquitectura de esta red, donde los bloques de color azul son capas de convolución, los rojos de *pooling*, los amarillos de *SoftMax* y las verdes serían capas de otras clases (normalmente de concatenación de resultados entre varias capas).

Esta red se diseñó teniendo en cuenta la eficiencia computacional, especialmente buscando la cantidad mínima de RAM para su entrenamiento y uso. Introducen los módulos *Inception*, los cuales sustituyen a una capa de convolución por varias capas de convoluciones más pequeñas y una última capa de filtro que se encarga de agrupar los resultados (un ejemplo de este tipo de módulo puede verse en la figura 2.4). Estos módulos producen mejores resultados que las convoluciones más grandes, adicionalmente mejoran la velocidad en el cálculo.

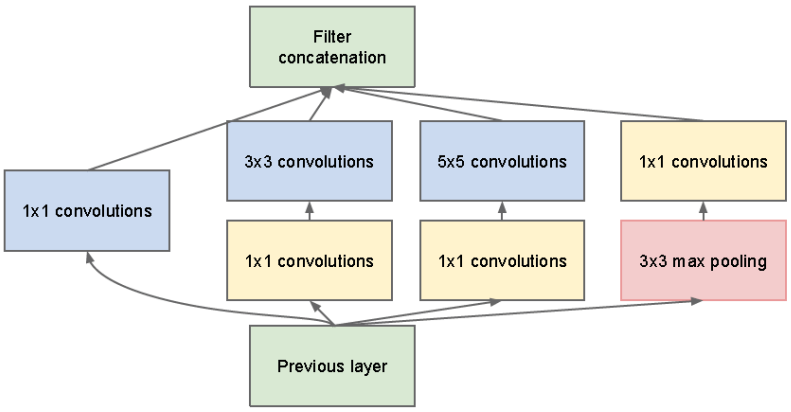

Figura 2.4. Ejemplo de un módulo *Inception* introducidos en la red GoogLeNet.

Esta red es la ganadora de la competición ILSVRC en el año 2014. El número total de parámetros es de unos 4 millones.

2.5.3. VGG.

La red está completamente definida en [36] y el modelo se ha obtenido de [30].

La entrada está preparada para imágenes de 224x224x3 (ancho x alto x RGB), tiene 13 capas de convolución, 5 de ellas con *maxpool*, y tres capas de redes profundas, dos de ellas de 4.096 y la última de 1.000. Por último, se tiene una capa de *softmax* para obtener los resultados de las clases.

Esta red quedó en segunda posición en la competición ILSVRC del año 2014, detrás de GoogLeNet. No obstante, se ha incluido en el proyecto debido a que es considerada la mejor red actual en cuanto a extracción de características en imágenes. El número de parámetros aproximado es de 138 millones.

El modelo que se ha usado en el proyecto está entrenado para el reconocimiento de caras y se puede ver más en profundidad en [29].

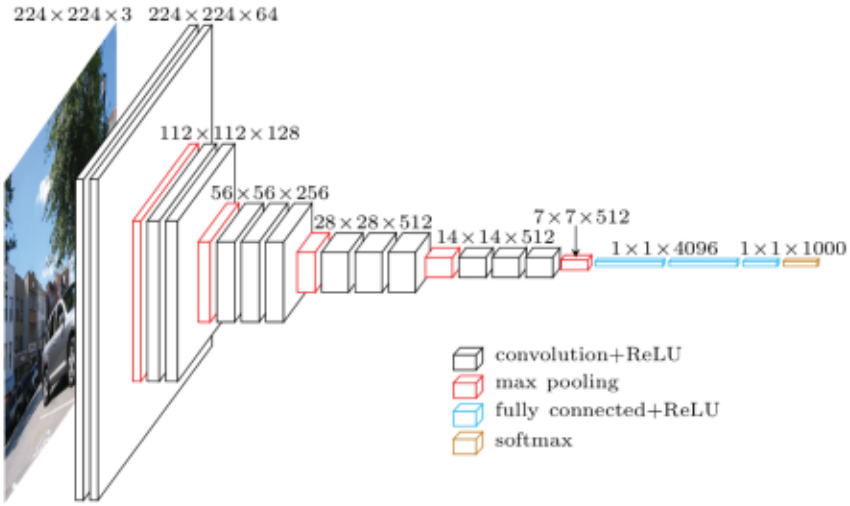

Figura 2.5. Arquitectura de la red VGG [19].

2.5.4. ResNet.

Se puede ver la definición completa de esta red en [18], en cuanto al modelo utilizado se ha obtenido de [16].

La principal característica de este tipo de red es que permite diseñar redes mucho más profundas (con más capas) que las diseñadas hasta el momento en que se definió. Como ejemplo, la red propuesta de 152 capas en este estudio tiene 8 veces más capas que la VGG vista previamente.

El objetivo de la técnica utilizada por este tipo de redes consiste en rediseñar las capas utilizando funciones residuales de aprendizaje con referencia a las capas de entrada. Este nuevo diseño facilita la optimización y ganan precisión incrementando la profundidad.

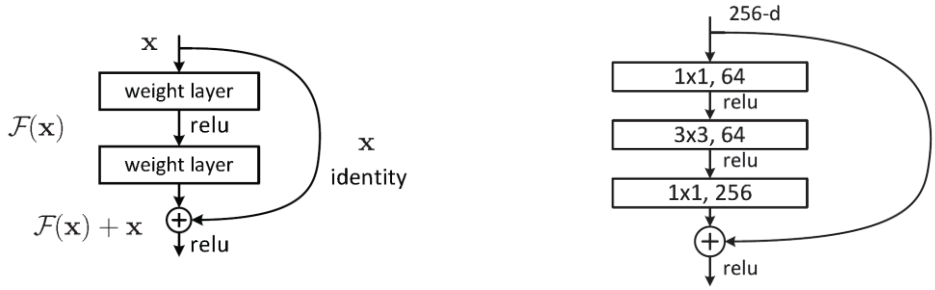

Figura 2.6. Bloque de aprendizaje residual.

Los resultados obtenidos se comparan entre redes de 50, 101 y 152 capas de profundidad. Se demuestra que con más número de capas se obtienen mejores resultados, no obstante, la complejidad añadida en la red de 152 capas no mejora sustancialmente las tasas de error que se obtiene con la red de 101 capas. Al contrario, la red de 101 capas sí que mejora bastante en relación con la de 50. En este proyecto se ha decidido utilizar el modelo de 101 capas.

Esta red es la ganadora de la competición ILSVRC en 2015.

2.6. METODOLOGÍA.

2.6.1. Búsqueda de la mejor red.

Lo primero que se realizó es la búsqueda de la red que mejor se comportaba con las imágenes que se consideraron para el entrenamiento de las mismas. El entrenamiento y evaluación se ha realizado con la base de datos RAF-DB, la cual dispone de 2.524 y 680 imágenes para entrenamiento y test respectivamente. En todas las redes se ha realizado el mismo número de *epochs* para poder comparar el comportamiento de cada una de ellas. Un *epoch* es una pasada completa sobre el conjunto de datos de entrenamiento. En este caso se han obtenido los resultados que se pueden ver en la tabla 2.5.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**146**	2	0	9	30	9	133
	A	11	**19**	0	11	9	10	14
	Q	2	0	**0**	10	50	7	91
	F	13	3	0	**784**	130	31	224
	T	6	3	0	35	**289**	19	126
	E	6	2	0	8	29	**83**	34
	N	30	0	1	16	109	5	**519**

Tabla 2.2. Matriz de confusión para la red AlexNet.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**220**	26	2	10	3	7	61
	A	13	**37**	0	6	7	1	10
	Q	0	0	**39**	24	27	15	55
	F	4	10	3	**1090**	21	4	53
	T	0	11	9	31	**366**	5	56
	E	2	5	13	10	5	**106**	21
	N	12	0	4	56	100	4	**504**

Tabla 2.3. Matriz de confusión para la red VGG.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**239**	26	1	18	23	5	17
	A	16	**34**	1	7	11	1	4
	Q	5	1	**8**	23	68	20	35
	F	15	15	13	**1050**	46	8	38
	T	3	5	10	22	**407**	9	22
	E	5	9	3	15	22	**92**	16
	N	79	1	2	93	226	1	**278**

Tabla 2.4. Matriz de confusión para la red GoogLeNet.

Red	Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
AlexNet	44,38	25,68	0,00	66,16	60,46	51,23	76,32	46,32
VGG	66,87	50,00	24,38	91,98	76,57	65,43	74,12	64,19
GoogLeNet	72,64	45,95	5,00	88,61	85,15	56,79	40,88	56,43

Tabla 2.5. Comportamiento de las redes con las bases de datos de test una vez entrenadas con el conjunto de entrenamiento. Los datos están obtenidos de la diagonal principal de la matriz de confusión.

Como se puede observar, la mejor red con este conjunto de datos es VGG. A partir de este punto, esta red es la elegida para intentar mejorar los resultados.

2.6.2. Clasificación binaria con múltiples redes.

El primer intento de mejora que se ha realizado en el proyecto es, dado que el número de clases a reconocer es pequeño, entrenar siete redes neuronales binarias, una por cada clase, de tal forma que cada red dirá si es de esa clase concreta o no lo es y al final se toma una decisión en función de los resultados de cada una de las redes.

En este proceso, inicialmente también se han usado las imágenes de la base de datos RAF-DB (con 2.525 y 680 imágenes de entrenamiento y test respectivamente). El número de *epochs* usados para el entrenamiento ha sido el mismo que en el experimento anterior. Los resultados obtenidos fueron valores muy bajos, por ejemplo, las redes que se han entrenado para las clases de “Sorprendido” y “Asustado”, resultaron en valores de 36,47 y 2,70 respectivamente (valor de la diagonal principal de la matriz de confusión). Después de analizar el proceso se ha visto que es debido al desequilibrio en el número de imágenes de entrenamiento para cada una de las clases. Por ejemplo, para la de “Sorprendido” se tienen 1.290 imágenes, frente al resto de imágenes pertenecientes a otras clases de 10.981.

Para corregir este desequilibrio se acudió al método de eliminación aleatoria del resto de imágenes no pertenecientes a la clase bajo estudio (*undersampling* en inglés), tratando de dejar el máximo número de imágenes de cada una del resto de clases, es decir, por ejemplo, cuando se entrena la red de la clase "Sorprendido", se tienen 1.290 imágenes de esa clase y se obtienen 1.290 imágenes de las otras clases, de forma equitativa, quedando 215 imágenes de "Asustado", 215 imágenes de "Asqueado", 215 imágenes de "Feliz", 215 imágenes de "Enfadado" y 215 imágenes de "Neutral". La selección de esas imágenes, como se ha indicado, es aleatoria entre las disponibles para entrenamiento.

Con este proceso se obtienen los resultados mostrados en la tabla 2.6.

Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral
79,64	83,78	63,75	86,67	89,96	89,50	91,02

Tabla 2.6. Porcentaje de acierto para cada una de las redes binarias entrenadas.

La base de test utilizada no se ha retocado con respecto a la original, lo único que ha cambiado son las imágenes usadas para el entrenamiento.

Si se analizan estos resultados, a primera vista se puede creer que se mejora de forma sustancial comparando con los obtenidos anteriormente con clasificación múltiple (una única red con múltiples clases), no obstante, como se ve en la tabla 2.8, al tomar una decisión conjunta sobre la salida de todas las redes, no resulta en la mejora esperada.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**219**	57	9	10	9	12	13
	A	6	**53**	3	2	7	2	1
	Q	4	15	**73**	9	22	22	15
	F	22	213	53	**595**	182	51	69
	T	8	56	24	0	**351**	21	18
	E	1	15	16	3	4	**121**	2
	N	19	3	9	55	86	7	**501**

Tabla 2.7. Matriz de confusión al elegir el mejor resultado entre todas las redes binarias.

Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
66,56	71,62	45,62	50,21	73,43	74,69	73,67	65,11

Tabla 2.8. Porcentaje de acierto al elegir el mejor resultado entre todas las redes binarias.

Para decidir si la imagen pertenece a una clase determinada, se ha obtenido el valor resultante de cada una de las redes, se ha calculado el cociente entre el valor de pertenencia a la clase y el acumulado de ese mismo valor con el de no pertenencia a la clase. Por último, se ha cogido el mayor valor entre todos los resultados.

De esta forma se ha podido constatar que, para estos datos concretos bajo estudio, el utilizar varios clasificadores binarios para cada una de las clases mejora el clasificador múltiple en menos de un punto en la media tal y como se muestra en la tabla 2.9.

Red	Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
VGG multiclase	66,87	50,00	24,38	91,98	76,57	65,43	74,12	64,19
VGG binaria por clase	66,56	71,62	45,62	50,21	73,43	74,69	73,67	65,11

Tabla 2.9. Comparativa de porcentaje de acierto entre la red multiclase y la red binaria por clase.

Si se analizan los resultados por clases, se puede observar que se empeora bastante en la clase “Feliz”, que a priori es la más sencilla normalmente de detectar, se mejoran en las clases de “Asustado”, “Asqueado” y “Enfadado” y en las restantes 3 clases prácticamente se tiene el mismo porcentaje (algo menor en la clase “Triste”).

2.6.3. Balanceo del conjunto de datos de entrenamiento.

Mediante el estudio de los clasificadores binarios, se ha podido observar la importancia del balanceo de los datos de entrenamiento para obtener buenos resultados en la red. Observando el número de imágenes para cada clase (vistos anteriormente en la tabla 2.1), se ve un claro desequilibrio en el número de imágenes de cada clase en el entrenamiento, por lo que se ha buscado la forma de corregir esto, intentando así obtener una mejora en la red resultante.

Tal como se ha estudiado en [7], existen varias maneras de corregir el balanceo en el número de datos de entrenamiento. Dependiendo de qué se modifica, existen tres caminos diferentes, modificar los datos de entrenamiento, modificar el proceso de entrenamiento de la red y un mixto, que realiza cambios en ambos procesos. Como método mixto, se propone el entrenamiento de la red con la base de datos modificada (balanceada) y posteriormente realizar un ajuste fino o *finetuning* con los datos originales (desbalanceados).

Dentro de la modificación en los datos de entrenamiento, se distingue entre reducir el número de datos hasta conseguir el mismo número en cada una de las clases (análogo al método usado anteriormente en el estudio de la clasificación binario), o en ampliar el

número de datos buscando el mismo objetivo de tener el mismo número de datos para cada clase. En este artículo se llega a la conclusión, con los datos utilizados en el estudio, que el método que mejor funciona es el de ampliar el número de datos.

Ampliando el número de imágenes de entrenamiento por clase hasta 4.772, correspondientes a la clase "Feliz" que es la que más imágenes tiene, simplemente duplicando aleatoriamente imágenes de cada clase correspondiente se obtienen los resultados que se muestran en la tabla 2.11.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**265**	4	2	27	5	5	21
	A	17	**36**	0	5	10	1	5
	Q	1	2	**45**	35	28	17	32
	F	4	1	2	**1118**	26	4	30
	T	2	1	7	50	**368**	8	42
	E	1	2	2	19	5	**118**	15
	N	22	1	3	92	48	1	**513**

Tabla 2.10. Matriz de confusión con base de datos ampliada.

Sorprendido	**Asustado**	**Asqueado**	**Feliz**	**Triste**	**Enfadado**	**Neutral**	**Media**
80,55	48,65	28,13	94,35	76,99	72,84	75,44	68,14

Tabla 2.11. Resultados obtenidos a partir de la matriz de confusión con base de datos ampliada.

En la Tabla 2.12 puede verse la comparativa entre los resultados obtenidos mediante el balanceo en el número de imágenes de entrenamiento por clase y los obtenidos con la original.

Red	**Sorprendido**	**Asustado**	**Asqueado**	**Feliz**	**Triste**	**Enfadado**	**Neutral**	**Media**
VGG base entrenamiento original	66,87	50,00	24,38	91,98	76,57	65,43	74,12	64,19
VGG base entrenamiento aumentada	80,55	48,65	28,13	94,35	76,99	72,84	75,44	68,14

Tabla 2.12. Comparativa de porcentaje de acierto entre la red VGG entrenada con la base de entrenamiento aumentada y la original.

Utilizando este método se obtiene una mejora de casi cuatro puntos porcentuales en la media, mejorando en todas las clases excepto en “Asustado” que se empeora en algo más de un punto.

En el siguiente paso del proyecto, se ha decidido aumentar el número de imágenes por clase, pero en vez de duplicarlas directamente se ha aplicado una transformación. La transformación consiste en coger una imagen de forma aleatoria de esa clase y aplicar una pequeña traslación, rotación y zoom de forma simultánea. Los valores de esa transformación también se eligen de forma aleatoria. Al igual que en el caso anterior, se ha buscado tener el mismo número de imágenes por cada clase, aumentando las que menos tienen y dejando las originales para el caso de la clase con mayor número. En la tabla 2.14 se pueden ver los resultados.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**283**	7	0	6	5	9	19
	A	19	**35**	0	5	8	3	4
	Q	7	5	**64**	20	20	14	30
	F	15	2	8	**1066**	31	7	56
	T	2	3	5	22	**406**	4	36
	E	5	2	2	15	2	**123**	13
	N	23	1	3	39	56	2	**556**

Tabla 2.13. Matriz de confusión con base de datos ampliada aplicando transformación.

Sorprendido	**Asustado**	**Asqueado**	**Feliz**	**Triste**	**Enfadado**	**Neutral**	**Media**
86,02	47,30	40,00	89,96	84,94	75,93	81,76	72,27

Tabla 2.14. Resultados obtenidos a partir de la matriz de confusión con base de datos ampliada aplicando transformación.

Comparando estos resultados con los obtenidos originalmente sin aumentar la base de datos de entrenamiento, como se puede ver en la tabla 2.15, en cinco de las clases se obtiene una mejora de varios puntos y se reduce en dos de ellas (en “Feliz” y “Asustado”), de media la mejora es de algo más de ocho puntos, mejorando incluso la anterior en la que no se realizaba ninguna transformación (en concreto cuatro puntos adicionales).

Red	Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
VGG base entrenamiento original	66,87	50,00	24,38	91,98	76,57	65,43	74,12	64,19
VGG base entrenamiento aumentada con transformación	86,02	47,30	40,00	89,96	84,94	75,93	81,76	72,27

Tabla 2.15. Comparativa de porcentaje de acierto entre la red VGG entrenada con la base de entrenamiento aumentada y la original.

2.6.4. Uso de imágenes en escala de grises.

Continuando con la búsqueda de mejoras en los resultados, esta vez se estudia el comportamiento de la red cuando se entrena con las imágenes en escala de grises. A priori parece que el color no debería contener mucha información en cuanto a las características necesarias para el reconocimiento de emociones en las caras.

En la tabla 2.17 pueden observarse los resultados obtenidos con la base de entrenamiento aumentada con transformaciones e imágenes convertidas a escala de grises (al igual de las imágenes de test).

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**282**	4	4	12	5	3	19
	A	18	**37**	0	5	8	3	3
	Q	2	1	**86**	18	20	10	23
	F	10	4	6	**1115**	15	3	32
	T	3	1	6	22	**407**	3	36
	E	5	2	6	13	1	**125**	10
	N	21	0	3	36	47	1	**572**

Tabla 2.16. Matriz de confusión con base de datos ampliada aplicando transformación y usando imágenes en escala de grises.

Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
85,71	50,00	53,75	94,09	85,15	77,16	84,12	75,71

Tabla 2.17. Resultados obtenidos a partir de la matriz de confusión con base de datos ampliada aplicando transformación y usando imágenes en escala de grises.

En la tabla 2.18 se puede comprobar que también se mejoran los resultados si se utilizan imágenes en escala de grises, obteniendo más de 11 puntos de mejora en la media y mejorando también en todas las clases bajo estudio.

Red	Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media	*Accuracy*
VGG base entrenamiento original	66,87	50,00	24,38	91,98	76,57	65,43	74,12	64,19	76,99
VGG base entrenamiento aumentada	80,55	48,65	28,13	94,35	76,99	72,84	75,44	68,14	80,28
VGG base entrenamiento aumentada con transformación	86,02	47,30	40,00	89,96	84,94	75,93	81,76	72,27	82,56
VGG base entrenamiento aumentada con transformación con escala de grises	85,71	50,00	53,75	94,09	85,15	77,16	84,12	75,71	85,53

Tabla 2.18. Comparativa completa de porcentaje de acierto para la red VGG en cada uno de los pasos usados para la mejora de resultados.

También se ha intentado mejorar los resultados utilizando la técnica de reentrenar la última capa de clasificación (*finetuning*) con la base de datos original desbalanceada tal como se indica en [7], no obstante, en este estudio y con los datos usados, no se han obtenido mejoras al contrario de lo indicado en la referencia anterior.

Posteriormente, en el capítulo de resultados, se aplicarán estos procesos de mejora obtenidos con la red VGG y se compararán las diferentes redes propuestas, así como el estado del arte actual de reconocimiento de emociones con la base de datos RAF-DB.

2.7. APLICACIÓN MÓVIL ANDROID.

Como ejemplo de uso de estas redes entrenadas, se ha implementado una aplicación móvil en Android, la cual permite, a partir de cualquier modelo entrenado de Caffe, comprobar los resultados de aplicar la red sobre imágenes obtenidas de la cámara del dispositivo, o bien de su memoria interna.

El proceso que sigue la aplicación móvil para obtener los resultados a partir de una imagen se puede ver en la figura 2.7.

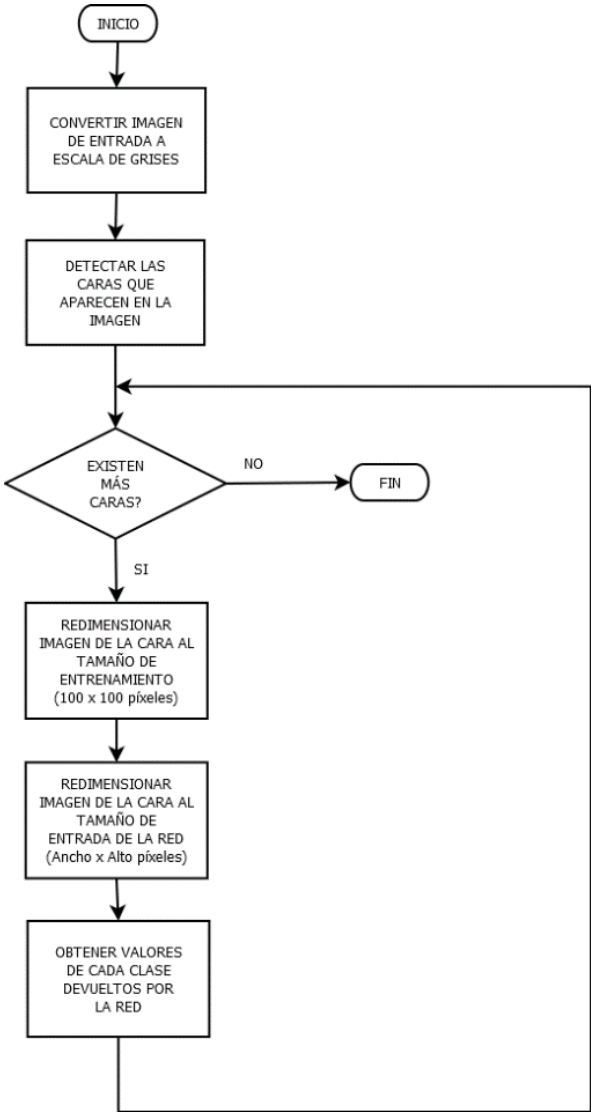

Figura 2.7. Diagrama de flujo para la obtención de los resultados de la red.

Para el proceso de detección de las caras en una imagen se ha utilizado el algoritmo en cascada Haar [41], el cual se puede usar fácilmente en OpenCV ya que proporciona por defecto el fichero de clasificación ya entrenado, así como las funciones de la API necesarias para su uso.

Para que la aplicación funcione es necesario proporcionar un modelo *caffe* de una red neuronal entrenado, las dimensiones de la imagen de entrada a la red en píxeles, las clases de salida de la red con su nombre y los parámetros del proceso de detección de caras: factor de escala y mínimo número de vecinos.

En la figura 2.8 pueden verse algunas capturas de pantalla correspondientes a la aplicación móvil desarrollada.

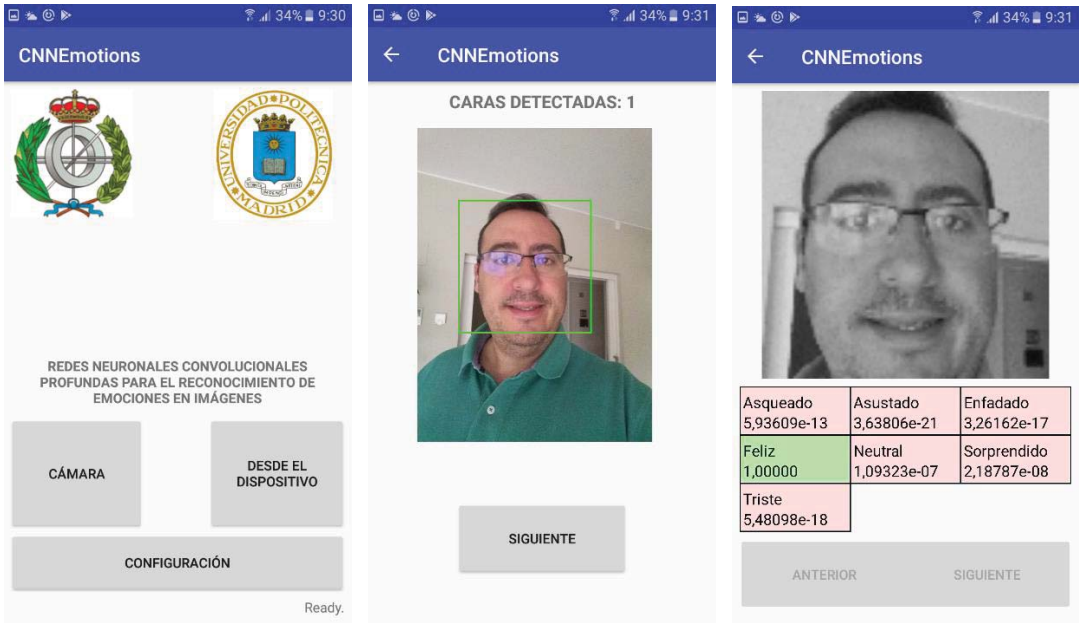

Figura 2.8. Capturas de la aplicación Android desarrollada.

En la ventana principal de la aplicación se elige la fuente de la imagen, o bien de la cámara para tomar una foto o selección de la memoria interna del dispositivo. En la segunda captura se muestra la foto con todas las caras detectadas en una imagen de ejemplo. Por último, se muestra cada una de las caras con el valor de cada clase.

3. RESULTADOS.

Para la base de datos RAF-DB se han obtenido los resultados que se muestran en la tabla 3.1 para la red AlexNet, en la tabla 3.2 para la red GoogLeNet, en la tabla 3.3 para la red VGG, y en la tabla 3.4 para la red ResNet-101. Para todas ellas se ha utilizado el mismo conjunto de entrenamiento aumentado con transformación y con las imágenes convertidas en escalas de grises. El conjunto de datos de test es el original de RAF-DB convertido a escala de grises.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**258**	2	1	16	9	5	38
	A	14	**34**	2	9	6	3	6
	Q	1	1	**54**	24	23	9	48
	F	10	2	4	**1092**	11	5	61
	T	1	0	2	39	**365**	1	70
	E	11	0	6	16	8	**104**	17
	N	12	0	1	36	27	1	**603**

Tabla 3.1. Matriz de confusión para la red AlexNet.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**255**	2	2	21	7	1	41
	A	11	**36**	0	16	6	2	3
	Q	0	0	**50**	41	21	9	39
	F	4	1	2	**1144**	5	2	27
	T	0	0	5	70	**338**	3	62
	E	9	0	4	31	3	**105**	10
	N	10	0	4	100	26	2	**538**

Tabla 3.2. Matriz de confusión para la red GoogLeNet.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**282**	4	4	12	5	3	19
	A	18	**37**	0	5	8	3	3
	Q	2	1	**86**	18	20	10	23
	F	10	4	6	**1115**	15	3	32
	T	3	1	6	22	**407**	3	36
	E	5	2	6	13	1	**125**	10
	N	21	0	3	36	47	1	**572**

Tabla 3.3. Matriz de confusión para la red VGG.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**287**	3	2	13	3	3	18
	A	14	**39**	2	6	6	5	2
	Q	4	0	**81**	36	12	6	21
	F	10	5	5	**1112**	10	6	37
	T	0	0	9	37	**395**	2	35
	E	6	2	10	25	4	**108**	7
	N	22	0	13	70	38	5	**532**

Tabla 3.4. Matriz de confusión para la red ResNet-101.

Si se agrupan los resultados de positivos reales (basándose en los resultados de la diagonal principal de las matrices de confusión) se obtiene la tabla 3.5 como comparativa de cada una de las redes utilizadas. Adicionalmente se ha incluido el estado del arte actual de reconocimiento sobre el conjunto de datos RAF-DB original [35].

Red	Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
AlexNet	78,42	45,95	33,75	92,15	76,36	64,20	88,68	68,50
GoogLeNet	77,51	48,65	31,25	96,54	70,71	64,81	79,12	66,94
VGG	85,71	50,00	53,75	94,09	85,15	77,16	84,12	75,71
ResNet-101	87,23	52,70	50,63	93,84	82,64	66,67	78,24	73,13
RAF-DB [35]	81,16	62,16	52,15	92,83	80,13	71,60	80,29	74,33

Tabla 3.5. Comparativa de porcentaje de acierto de cada una de las redes entrenadas y el mejor resultado obtenido sobre RAF-DB [35].

Analizando los resultados se puede constatar que, observando la media, el mejor comportamiento corresponde a la red VGG con los datos de entrenamiento tratados. Incluso se mejora el estado del arte actual para el conjunto de datos RAF-DB. La mejora es algo inferior a un punto y medio. El resto de redes están por debajo, no obstante ResNet-101 solamente obtiene algo más de un punto porcentual de desventaja frente al estado del arte. El peor resultado corresponde a la red GoogLeNet.

Clase por clase, con la red VGG, se mejora el porcentaje en todas excepto para la clase "Asustado" (comparando con el mejor resultado en el estado del arte actual), correspondiente a la clase con menor número de imágenes de entrenamiento.

La red ResNet-101 es la que mejor se comporta en la clase "Sorprendido".

En cuanto a la red GoogLeNet, obtiene el mejor porcentaje en la clase "Feliz".

Para comprobar cómo se comportan estas redes entrenadas con imágenes que provienen de conjuntos de datos diferentes a RAF-DB, se han buscado otras bases de datos y se han comparado todas las imágenes disponibles con su clase proporcionada. Cabe destacar que no se ha realizado ningún proceso de ajuste o *finetuning* en ninguna de las redes, se ha usado directamente el modelo de cada una ya entrenada.

El primer conjunto de datos utilizado para comprobar los resultados de las redes corresponden a CK+ [26], las cuales son fotografías de laboratorio con las emociones de los usuarios controladas, con resoluciones mayores a los 100x100 píxeles de las imágenes usadas para el entrenamiento de las redes (por lo que se ajustarán antes a esta resolución). El número de imágenes clasificadas es de 327.

En la tabla 3.6 se muestra la matriz de confusión para la red VGG, en la tabla 3.7 para la red AlexNet, en la tabla 3.8 para la red GoogLeNet y en la tabla 3.9 para la red ResNet-101.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**82**	0	0	0	0	0	1
	A	6	**6**	4	1	7	0	1
	Q	2	0	**32**	0	0	20	5
	F	0	0	0	**69**	0	0	0
	T	2	0	1	0	**17**	0	8
	E	1	0	6	0	9	**5**	24
	N	0	0	0	3	1	0	**14**

Tabla 3.6. Matriz de confusión para la red AlexNet con los datos de CK+.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**81**	0	0	2	0	0	0
	A	5	**3**	0	4	12	0	1
	Q	0	0	**29**	1	1	20	8
	F	0	0	0	**68**	0	1	0
	T	0	0	0	0	**19**	0	9
	E	0	0	8	5	13	**3**	16
	N	0	0	0	6	1	0	**11**

Tabla 3.7. Matriz de confusión para la red GoogLeNet con los datos de CK+.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**79**	0	0	1	2	0	1
	A	4	**8**	0	0	13	0	0
	Q	0	0	**41**	1	0	15	2
	F	0	0	0	**69**	0	0	0
	T	0	0	0	0	**25**	0	3
	E	0	1	14	0	19	**3**	8
	N	0	0	0	8	2	0	**8**

Tabla 3.8. Matriz de confusión para la red VGG con los datos de CK+.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**81**	1	0	1	0	0	0
	A	5	**10**	1	2	7	0	0
	Q	0	0	**45**	1	1	11	1
	F	0	0	0	**69**	0	0	0
	T	0	0	2	0	**24**	0	2
	E	1	0	12	3	12	**7**	10
	N	0	0	0	6	2	0	**10**

Tabla 3.9. Matriz de confusión para la red ResNet-101 con los datos de CK+.

Con los datos de la diagonal principal de las matrices de confusión de cada una de las redes se pueden comparar los resultados mostrados en la tabla 3.10.

Red	Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
AlexNet	98,80	24,00	54,24	100,00	60,71	11,11	77,78	60,95
GoogLeNet	97,59	12,00	49,15	98,55	67,86	6,67	61,11	56,13
VGG	95,18	32,00	69,49	100,00	89,29	6,67	44,44	62,44
ResNet-101	97,59	40,00	76,27	100,00	85,71	15,56	55,56	67,24

Tabla 3.10. Porcentaje de acierto de cada una de las redes con los datos de CK+.

Si se analizan los datos, la red que mejor se comporta de media con estos datos es la ResNet-101. Las clases “Sorprendido” y “Feliz” tienen un porcentaje de acierto bastante elevado en todas las redes. Las clases peores detectadas son “Asustado” y “Enfadado”, especialmente baja esta última.

En este caso, para la red ResNet-101 que es la que mejor se comporta, se tendría un *accuracy* del 75,23%.

En la Figura 3.1 se puede ver un ejemplo de las imágenes de entrenamiento de este conjunto de datos.

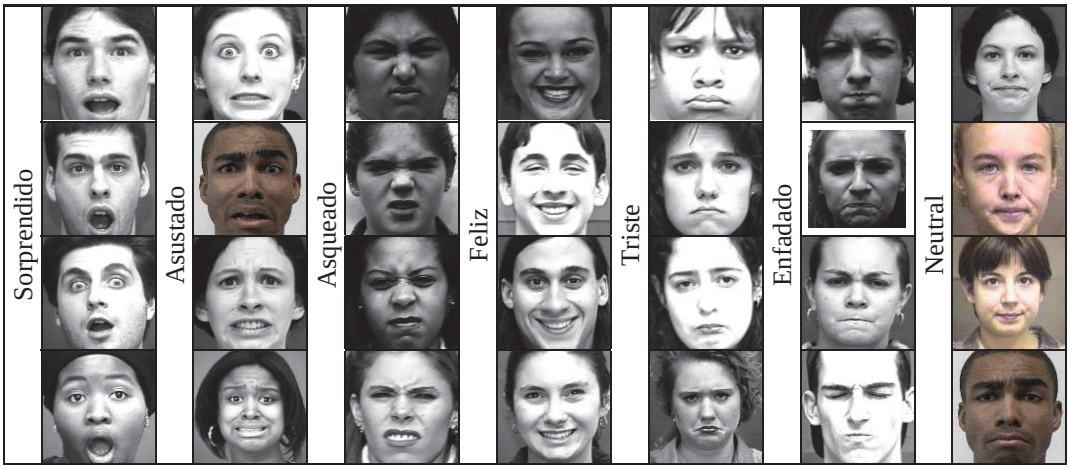

Figura 3.1. Muestra de imágenes de la base de datos CK+.

El otro conjunto de datos utilizado para comprobar los resultados obtenidos con las redes ya entrenadas es el utilizado por la competición de reconocimiento de expresiones faciales de 2013 (ICML 2013 / FER2013 [23]). Las imágenes tienen una resolución de 48x48 píxeles (por lo que se ha aumentado hasta los 100x100 píxeles para igualar las imágenes de entrenamiento de las redes). Todas las imágenes están en escala de grises. El número de imágenes es de 35.886 y tiene un *accuracy* de 68±5% al ser resuelto por humanos. El mejor *accuracy* automático obtenido para este conjunto de datos ha sido de 71,16%.

En este caso las matrices de confusión obtenidas por red pueden verse en las siguientes tablas.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**3263**	73	8	345	47	36	230
	A	1577	**313**	211	837	586	312	1285
	Q	59	13	**200**	80	63	67	65
	F	1065	94	254	**6293**	290	272	721
	T	775	68	211	858	**1466**	177	2522
	E	1035	128	393	521	380	**1174**	1322
	N	871	22	140	975	335	112	**3743**

Tabla 3.11. Matriz de confusión para la red AlexNet con los datos FER2013.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**3156**	57	15	474	72	18	210
	A	1246	**271**	182	1246	776	204	1196
	Q	49	11	**200**	115	97	46	38
	F	585	39	29	**7743**	118	54	421
	T	411	41	212	1345	**1832**	84	2152
	E	827	72	445	995	502	**933**	1179
	N	540	9	88	1574	433	34	**3520**

Tabla 3.12. Matriz de confusión para la red GoogLeNet con los datos FER2013.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**3194**	67	8	416	148	42	127
	A	1202	**351**	238	858	1216	321	935
	Q	50	9	**227**	54	121	71	15
	F	569	106	114	**7159**	505	131	405
	T	516	29	211	851	**2524**	106	1839
	E	661	57	517	611	752	**1358**	997
	N	493	9	134	1280	708	57	**3517**

Tabla 3.13. Matriz de confusión para la red VGG con los datos FER2013.

		Clase Resultado						
		S	**A**	**Q**	**F**	**T**	**E**	**N**
Clase Real	**S**	**3198**	163	10	442	62	37	91
	A	1326	**489**	205	1137	905	306	750
	Q	51	11	**230**	107	68	58	22
	F	576	98	102	**7686**	121	112	293
	T	611	73	288	1301	**1942**	107	1754
	E	727	98	612	964	478	**1257**	817
	N	696	23	167	1520	484	52	**3256**

Tabla 3.14. Matriz de confusión para la red ResNet-101 con los datos de FER2013.

Con los datos de la diagonal principal de las matrices de confusión de cada una de las redes se pueden comparar los resultados mostrados en la tabla 3.15.

Red	Sorprendido	Asustado	Asqueado	Feliz	Triste	Enfadado	Neutral	Media
AlexNet	81,53	6,11	36,56	70,01	24,12	23,70	60,39	43,20
GoogLeNet	78,86	5,29	36,56	86,14	30,15	18,84	56,79	44,66
VGG	79,81	6,85	41,50	79,64	41,54	27,42	56,74	47,64
ResNet-101	79,89	9,55	42,05	85,51	31,96	25,38	52,53	46,70

Tabla 3.15. Porcentaje de acierto de cada una de las redes con los datos de FER2013.

En este caso, y con las imágenes de este conjunto de datos, la mejor red es VGG teniendo en cuenta el valor medio de las clases.

En cuanto a las clases “Sorprendido” y “Neutral”, la mejor red ha sido AlexNet. Al fijarse en las clases “Asustado” y “Asqueado” sería la ResNet-101. La red VGG obtiene los mejores resultados para las clases “Enfadado” y “Triste”. En relación con la red GoogLeNet, se comprueba que es la que mejor se ha comportado en la clase “Feliz”.

El *accuracy* de la mejor red, que en este caso ha sido VGG, es de 51,08%. Este bajo porcentaje es debido a la baja resolución de las imágenes de entrada, cabe recordar que las redes bajo estudio se han entrenado con una resolución de 100x100 píxeles, mientras la resolución de este conjunto de datos bajo estudio es de 48x48 píxeles.

En la figura 3.2 se puede ver un ejemplo de las imágenes de entrenamiento de este conjunto de datos.

Figura 3.2. Muestra de imágenes de la base de datos FER2013.

Todos los estudios y pruebas se han realizado en un equipo HP Pavilion 500-325ns, con un microprocesador CPU Intel Core i7-4790S a 3.20GHz con 4 núcleos físicos y 8 lógicos. La cantidad de memoria RAM utilizada es de 16GB DDR3 SDRAM con una tarjeta gráfica NVIDIA GeForce GTX 745 [HP] con 4GB de memoria DDR3. El sistema

operativo que se ha usado para el entrenamiento de las redes ha sido un Ubuntu 16.04 de 64 bits.

Todas las redes se han entrenado con la GPU, excepto la ResNet 101, la cual requería más memoria RAM en la tarjeta gráfica incluso ajustando al máximo los parámetros de configuración en el entrenamiento.

Con este equipo, en la tabla 3.16, se pueden ver los tiempos de requeridos para el entrenamiento de cada una de las redes. Por cada red, se ha dado el modo de entrenamiento (GPU o CPU), el número de iteraciones utilizado en el entrenamiento y el tiempo requerido en completar el mismo.

Red	Modo	Iteraciones	Tiempo
AlexNet	GPU	15.000	7 horas y 16 minutos
GoogLeNet	GPU	30.000	19 horas y 42 minutos
VGG	GPU	120.000	4 días y 5 horas
ResNet-101	CPU	60.000	7 días

Tabla 3.16. Tiempo de entrenamiento de las redes.

Como se puede ver en la tabla, el número de iteraciones varía, esto es debido al parámetro *batch_size* del fichero de configuración de la red en la parte de definición de datos de entrenamiento, si se puede introducir un valor mayor, se reducen las iteraciones, pero se consume más memoria RAM. En todas las redes se ha mantenido constante el número de *epochs* usados para el entrenamiento.

La red ResNet-101, debido a su consumo de memoria por la cantidad de capas y parámetros internos, no se ha podido entrenar con la GPU y se ha tenido que utilizar la CPU de la máquina. Es la que más tiempo de entrenamiento ha requerido, a pesar de configurar la mitad de iteraciones que con la red VGG.

El programa de comprobación de los modelos entrenados de las redes se ha realizado en Python 3, con la librería OpenCV. En este caso no se ha utilizado directamente *caffe* con el fin de facilitar la implementación posterior de la aplicación en Android.

4. CONCLUSIONES.

Como se ha visto a través de las pruebas realizadas, y siempre teniendo en cuenta que se ha realizado sobre el conjunto de datos bajo estudio, el desbalanceo en la base de datos de entrenamiento afecta bastante a los resultados obtenidos en redes neuronales convolucionales profundas. El simple balanceo copiando aleatoriamente las imágenes y equiparando el número de imágenes entre las diferentes clases de entrenamiento mejora bastante los resultados, y si además, no se hace una simple copia, sino que se realizan transformaciones aleatorias de traslación, rotación y zoom, se mejoran todavía más los resultados.

Se podría haber optado por cambiar internamente el proceso de entrenamiento modificando el código de *caffe* buscando corregir ese desbalanceo y, al igual que por ejemplo los creadores de la base de datos de RAF-DB, crear un nuevo tipo de capa donde mejorar la clasificación obtenida mediante los métodos estándar soportados por *caffe*. De esta forma también se mejoran los resultados, no obstante, este sistema complica el posterior uso de la red entrenada, ya que no se pueden usar librerías estándar como OpenCV. En este caso se tendría que portar esa modificación de *caffe* y utilizarlo directamente sobre la plataforma elegida como aplicación final de uso de la red. Aunque *caffe* está escrito en C++ y el código en este lenguaje es fácilmente portable, el problema viene por la cantidad de dependencias de librerías de terceros que también tendrían que estar disponibles en la plataforma de uso.

El transformar las imágenes en escala de grises, tanto para el entrenamiento como para el test y uso final, también consigue mejorar los resultados finales, lo que parece indicar la independencia de las emociones con el color, o que la arquitectura de la red no consigue sacar las características del color que de información para mejorarlos. Tal y como se comentó en la introducción, otros estudios han utilizado el color y la variación de este para el reconocimiento de emociones [2].

El uso de escala de grises además facilita el uso de la red entrenada con conjuntos de datos que ya se encuentran en escala de grises y, en caso contrario, no podrían haberse utilizado en las pruebas.

Se ha comprobado que el uso de la red entrenada con otros conjuntos de datos que nada tiene que ver con la base de datos utilizada para entrenarlos obtiene unos resultados buenos en algunas de las clases si las imágenes de entrada tienen superior resolución a las usadas para el entrenamiento (100 x 100 píxeles), no obstante, en el caso de utilizar imágenes de test inferiores a la resolución de entrenamiento (48 x 48 píxeles en el caso de las pruebas) reduce bastante los resultados de la red en todas las clases.

5. LÍNEAS FUTURAS.

Aunque el balanceo con transformación de los datos de entrada ha conseguido mejorar bastante el resultado inicial de reconocimiento de la red, se debería intentar ampliar las bases de datos con imágenes reales clasificadas sobre todo en las clases de asustado, asqueado y enfadado que tienen un número de imágenes de hasta 20 veces inferior que la clase con mayor número (que en este caso es la clase feliz).

Igualmente, para tratar de aprovechar al máximo la extracción de características de las imágenes en estas redes cuyas imágenes de entrada son de 224x224 píxeles, se tendrían que usar imágenes de entrenamiento, test y uso de esa resolución (o mayores disminuyéndolas).

Se puede trabajar en realizar un modelo de red equivalente a VGG (o a cualquiera de las otras redes usadas en el proyecto) pero optimizada para imágenes en escala de grises, disminuyendo la dimensionalidad y obteniendo modelos más rápidos y con menos uso de recursos en la aplicación final.

.

6. REFERENCIAS.

[1] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado G, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mane D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viegas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, y Zheng X. 2016. TensorFlow: Large-scale machine learning on heterogeneous systems. *arXiv:1603.04467.*

[2] Benitez-Quiroz CF, Srinivasan R, y Martinez AM. 2018. Facial color is an efficient mechanism to visually transmit emotion. *Proceedings of the National Academy of Sciences April 3, 2018. 115 (14) 3581-3586.*

[3] Benitez-Quiroz CF, Wang Y, y Martinez AM. 2017. Recognition of Action Units in the Wild With Deep Nets and a New Global-Local Loss. *IEEE International Conference on Computer Vision (ICCV). 2380-7504.*

[4] Benitez-Quiroz CF, Srinivasan, R, y Martinez, AM. 2016. EmotioNet: An accurate, real-time algorithm for the automatic annotation of a million facial expressions in the wild. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 5562–5570.*

[5] Bradski G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[6] Breuer R, y Kimmel R. 2017. A Deep Learning Perspective on the Origin of Facial Expressions. *arXiv:1705.01842.*

[7] Buda M, Maki A y Mazurowski MA. 2017. A systematic study of the class imbalance problem in convolutional neural networks. *arXiv:1710.05381*.

[8] Chu WS, De la Torre F, y Cohn JF. 2017. Learning Spatial and Temporal Cues for Multi-label Facial Action Unit Detection. *12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017).*

[9] deeplearning4j.org. Recuperado el 23 de abril de 2018 de https://deeplearning4j.org/compare-dl4j-tensorflow-pytorch

[10] Du S, y Martinez AM. 2015. Compound Facial Expressions of Emotion: From basic research to clinical applications. *Dialogues in Clinical Neuroscience, Vol. 17, No. 4, 2015.*

[11] Du S, Taou Y, y Martinez AM. 2014. Compound Facial Expressions of Emotion. *Proceedings of the National Academy of Sciences 111 (15) E1454-E1462, 2014.*

[12] García JR, Saad, G. 2018. Evolutionary neuromarketing: darwinizing the neuroimaging paradigm for consumer behavior. *Journal of Consumer Behaviour. An International Research Review. Pages 397-414. July - October 2008. Volume7, Issue: 4-5. Special Issue: Neuromarketing*.

[13] github.com. Recuperado el 19 de diciembre de 2017 de https://github.com/BVLC/caffe/.

[14] github.com. Recuperado el 11 de enero de 2018 de https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet.

[15] github.com. Recuperado el 23 de enero de 2018 de https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet.

[16] github.com. Recuperado el 3 de febrero de 2018 de https://github.com/KaimingHe/deep-residual-networks#models.

[17] Happy SL, George A, y Routray A. 2012. A Real Time Facial Expression Classification System Using Local Binary Patterns. *4th International Conference on Intelligent Human Computer Interaction (IHCI), 2012*.

[18] He K, Zhang X, Ren S, y Sun J. 2015. Deep Residual Learning for Image Recognition. *arXiv:1512.03385*.

[19] heuritech.com. Recuperado el 5 de abril de 2018 de https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/.

[20] Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, y Salakhutdinov RR. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580.*

[21] Hsu, SC, Huang HH, y Huang CL. 2017. Facial Expression Recognition for Human-Robot Interaction. *IEEE International Conference on Robotic Computing (IRC), 2017.*

[22] Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick, R, Guadarrama S y Darrell T. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv:1408.5093*.

[23] kaggle.com. Recuperado el 19 de marzo de 2018 de https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge.

[24] Ko BC. 2018. A Brief Review of Facial Emotion Recognition Based on Visual Information. *Sensors 2018, 18, 401*.

[25] Krizhevsky A, Sutskever I, y Hinton GE. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS 2012).*

[26] Lucey P, Cohn J, Kanade T, Saragih J y Ambadar Z. 2010. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*.

[27] Lucey P, Cohn J, Lucey S., Matthews I, Sridharan S, y Prkachin K. 2009. Automatically Detecting Pain Using Facial Actions. *Proceedings of the International Conference on Affective Computing and Intelligent Interaction, pages 1–8, 2009. 1*

[28] Lyons MJ, Budynek J, y Akamatsu S. 1999. Automatic classification of single facial images. *IEEE Transactions on Pattern Analysis & Machine Intelligence, (12):1357–1362, 1999.*

[29] Parkhi OM, Vedaldi A y Zisserman A. 2015. Deep Face Recognition. *British Machine Vision Conference, 2015.*

[30] robots.ox.ac.uk. Recuperado el 11 de marzo de 2018 de http://www.robots.ox.ac.uk/~vgg/software/vgg_face/.

[31] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, y Fei-Fei L. 2015. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575.*

[32] Ryan A, Cohn JF, Lucey S, Saragih J, Luecy P, De la Torre F, y Rossi Adam. 2009. Automated Facial Expression Recognition System. *43rd Annual 2009 International Carnahan Conference on Security Technology. Páginas 172–177.*

[33] Shan C, Gong S, y McOwan PW. 2009. Facial expression recognition based on local binary patterns: *A comprehensive study. Image and Vision Computing, 27(6):803–816, 2009.*

[34] Shen P, Wang S, y Liu Z. 2013. Facial Expression Recognition from Infrared Thermal Videos. *Intelligent Autonomous Systems 12 pp 323-333.*

[35] Shan L, Weihong D, y JunPing D. 2017. Reliable Crowdsourcing and Deep Locality-Preserving Learning for Expression Recognition in the Wild. *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*.

[36] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556.*

[37] Suk M, y Prabhakaran B. 2014. Real-time Mobile Facial Expression Recognition System – A Case Study. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).*

[38] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, y Rabinovich A. 2014. Going Deeper with Convolutions. *arXiv:1409.4842*.

[39] Szwoch M, y Pieniążek P. 2015. Facial emotion recognition using depth data. *8th International Conference on Human System Interactions (HSI), 2015.*

[40] Vinciarelli A, Pantic M, y Bourlard H. 2009. Social Signal Processing: Survey of an Emerging Domain. *Image and Vision Computing, 31(1):1743–1759, 2009. 1*

[41] Viola P, y Jones MJ. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, volume 1, pages I–511. IEEE, 2001.*

[42] Vural E, Cetin M, Ercil A, Littlewort G, Barlett M, y Movellan J. 2008. Automated Drowsiness Detection For Improved Driving Safety. *Proceedings of the International Conference on Automotive Technologies, 2008.*

[43] Whitehill J, y Omlin CW. 2006. Haar features for facs AU recognition. *Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on, page 5–pp. IEEE, IEEE, 2006.*