UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Emotions Recognition using Deep Learning

Trabajo Final de Máster

Máster Universitario en Inteligencia Artificial,
Reconocimiento de Formas e Imagen Digital

Autor: Carlos Pérez Estruch

Director: Roberto Paredes Palacios

15 de septiembre de 2016

Agradecimientos

Me gustaría agradecer y dedicar este trabajo a mis padres, sin ellos esto no sería posible.

También agradecer de forma especial a Roberto Paredes, por aceptar la tutoría de este TFM, la ayuda prestada y en general por la motivación y apoyo durante la realización del mismo.

Resumen

En este TFM se pretenden desarrollar diferentes redes convolucionales profundas para resolver el problema de reconocer emociones en rostros. En él se aplicarán las técnicas más novedosas de redes profundas y posteriormente se evaluará el efecto de cada uno de los parámetros que intervienen, así como aspectos de coste computacional.

Palabras clave

Aprendizaje profundo, Reconocimiento de Emociones, Redes Neuronales Convolucionales, Redes Profundas, Theano.

Abstract

In this TFM we are going to develop different deep convolutional networks to solve the problem of recognizing emotions in faces. We are going to apply the latest techniques of deep networks and then evaluate the effect of each one of the parameters involved, as well as aspects of computational cost.

Keywords

Deep Learning, Emotions Recognition, Convolutional Neural Networks, Deep Networks, Theano.

ÍNDICE

1. INTRODUCCIÓN.

1.1. Introducción.

En la actualidad el desarrollo y la innovación de la tecnología y la ciencia ha permitido el auge de sistemas de Inteligencia Artificial capaces de realizar tareas humanas complicadas. Motivados por la capacidad de computación y la gran cantidad de información almacenada digitalmente se precisan sistemas de procesamiento y análisis automático de datos. No obstante se ha llegado a un límite en la programación tradicional, en cuanto al uso de reglas pre-programadas, por la gran variabilidad de los datos de entrada y por tanto se tiende a buscar sistemas capaces de auto-programarse o, expresado de otro modo, sistemas que aprendan de los datos de entrada.

En este contexto se enmarca el «machine learning» (aprendizaje automático) cuya base se fundamenta en el aprendizaje humano. La idea principal se basa en crear programas capaces de aprender de información no estructurada en forma de ejemplos, es decir, crear modelos analíticos de forma automática. Uno de los enfoques principales del «machine learning» se conoce con el nombre de «Deep Learning» (aprendizaje profundo). En el «Deep Learning» se pretende realizar una abstracción de los datos originales de entrada aplicando un gran número de transformaciones no-lineales donde, por ejemplo, dada una matriz de píxeles de una imagen el sistema pueda interpretar elementos y características de la escena más allá del valor numérico de cada píxel.

En el presente trabajo se persigue el objetivo de aplicar uno de los métodos más importantes de «Deep Learning» en la actualidad, las redes neuronales convolucionales, al problema de reconocimiento de emociones en rostros de personas para observar su eficacia y obtener un buen modelo de representación.

En este documento se encontrará en el apartado 2 un resumen general a modo de estado del arte en cuanto al reconocimiento de emociones se refiere. Una vez expuesta la base del objeto de trabajo, en el apartado 3 se realizará una explicación teórica de las técnicas y métodos de «Deep Learning» que se utilizarán. En el apartado 4 se detallarán las herramientas y medios utilizados (software y hardware), además del origen de los datos. En el apartado 5 se expondrán los experimentos realizados y los resultados de los mismos. Finalmente en el apartado 6 se relatarán las conclusiones y trabajo futuro.

2. RECONOCIMIENTO AUTOMÁTICO DE EMOCIONES.

2.1. Introducción

La interacción hombre-máquina es un campo de estudio en el cual se busca como objetivo hacer más eficiente el intercambio de información: minimizar errores, hacerla más «amigable», etc. En definitiva se busca mejorar esta interacción pero para ello la máquina debería ser capaz de adaptarse a la situación en función del desarrollo de la comunicación, es decir, obtener la capacidad de interacción inteligente humana. En concreto, el reconocimiento de emociones, por ejemplo, a través de las expresiones del rostro (una de las habilidades principales de los humanos) y gracias a la gran capacidad de procesamiento (vídeo, sonido, etc) actual se han desarrollado sistemas automáticos de reconocimiento para tareas concretas. En este apartado se estudiarán los principales sistemas de reconocimiento de emociones que existen.

2.2. Tipos de reconocimiento de emociones.

Como es sabido, los humanos somos capaces de reconocer el estado emocional de otra persona a través de diversos estímulos como la voz, gestos, expresiones faciales, etc. A continuación se expondrán los diversos tipos de sistemas desarrollados para el reconocimiento de emociones por parte de las máquinas:

- Emociones en **habla**: La voz transmite una gran cantidad de información que puede ser utilizada para encontrar el estado emocional de una persona. Algunas de las variables que se utilizan en este tipo de detección son la velocidad de habla, la intensidad, la calidad de la voz, etc. En general se observan tres vertientes de investigación: una en la que se procesa la propia señal digital grabada, para encontrar algunas características como las mencionadas anteriormente, más relacionadas con la psicología y biología; otra en la que se entrenan algoritmos de «machine learning» directamente sobre la señal de voz; por último una fusión de ambas.

  Algunos ejemplos de aplicaciones son:

  - Good Vibrations: Se basa en la adquisición de señales biológicas de la voz, el corazón y la respiración para posteriormente realizar un feedback del estado emocional, la salud, etc, ya sea de una persona individual o de un colectivo. Clasifica las emociones entre placer, estrés y trastorno.

  - BeyondVerbal: Detecta emociones según la entonación en tiempo real.

  - Emovoice: Se basa en la detección de propiedades acústicas del habla, sin usar información de palabras, y posteriormente se entrena un clasificador «Naive Bayes» o «Support Vector Machines» a elección.

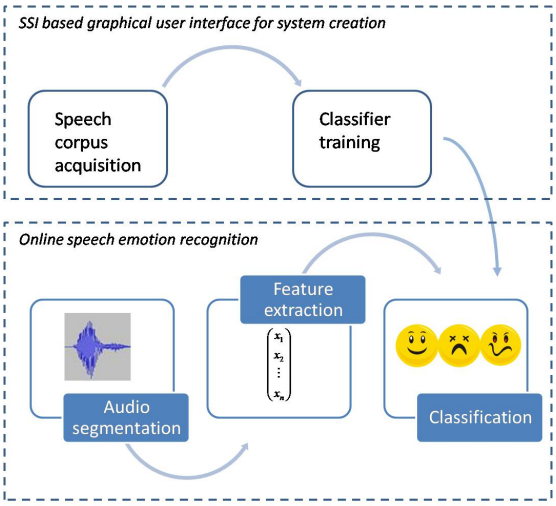

Figura 1: Recorrido de señal en Emovoice.
Fuente: [26]

- Emociones en **texto plano**: Este tipo de reconocimiento se basa en el procesado del lenguaje natural sobre texto basándose en la estructura de frases, vocabulario utilizado, relación entre palabras clave, etc. Un ejemplo típico de uso es el de procesar opiniones de consumidores de un determinado producto para valorar su experiencia.

  Algunas de las «API» más conocidas son:

  - The Tone Analizer: Desarrollada por IBM, detecta inclinación en opinión, tonos y estilos de escrituras distintos. En la web[1] se encuentra una demo donde se puede testear el funcionamiento del API.

  - Receptiviti: Usa palabras objetivo y categorías de emoción para deducir emociones y personalidad de un texto.

  - Bitext: Analiza relaciones entre palabras, frases, etc, para realizar una puntuación y posteriormente una clasificación en colores de las opiniones.

Figura 2: Ejemplo reconocimiento de emociones en texto con el API Bitext.
Fuente: [27]

[1] [https://tone-analyzer-demo.mybluemix.net]

- Emociones en **rostros**: La detección facial de emociones es posiblemente el tipo más extendido, ya que se considera uno de los indicadores más claros del estado emocional de una persona. Las características más importantes están relacionadas con la dirección de la mirada, posición de los pómulos, apertura de la boca (dientes expuestos), aparición de arrugas, en resumen la formación que adoptan los músculos de la cara.

  El uso principal de este tipo de detección se centra en mejorar la experiencia real en ciertas aplicaciones o situaciones, como por ejemplo la activación de cajeros automáticos en los cuáles no se entregaría dinero si se detecta la emoción de miedo. Otro uso importante está relacionado con el mundo de la publicidad, donde se adaptan los anuncios según el impacto emocional y la atención prestada, ya que las emociones se consideran un factor clave a la hora de comprar un producto.

  Las aplicaciones más destacadas son:

  - «**Emotion API**» de Microsoft: Trabaja con imágenes estáticas y se realiza en primer lugar un barrido de la imagen para detectar todas las posibles caras. Posteriormente se calculará la emoción y se clasificará entre enfado, desprecio, asco, miedo, felicidad, neutral, tristeza y sorpresa asignando un «score» de posibilidad de cada una. También existe la versión para reconocimiento en vídeo.

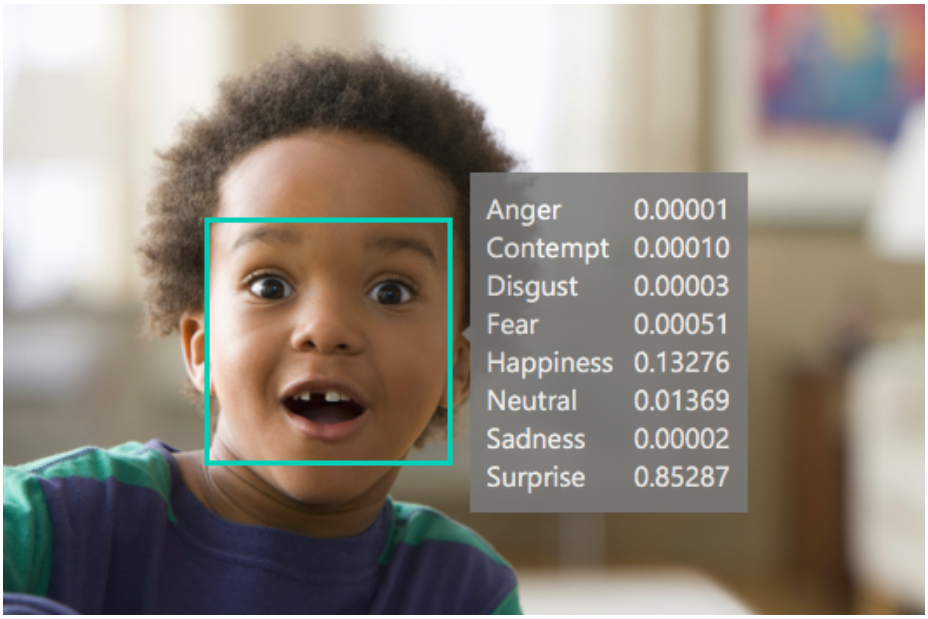

Figura 3: Ejemplo detección Emotion API de Microsoft. Fuente: [24]

  - Emovu: Desarrollada por el grupo Eyeris es un API lista para ser integrada en un sistema embebido. Está basada en el uso de redes convolucionales y tiene incorporado un complejo sistema de filtrado de decisiones y etapas de verificación.

  - Emotient: Basada en sistemas de Inteligencia Artificial esta API está especialmente diseñada para ser utilizada en campañas publicitarias ayudando a marcas y

anunciantes a mejorar su estrategia de venta cuantificando la respuesta emocional de una persona al ver un anuncio.

En este trabajo se trabajará exclusivamente en reconocer emociones en rostros de humanos.

2.3. Estado del Arte: Técnicas de reconocimiento de emociones en rostros.

Como se ha visto en el apartado anterior, existen varios tipos de reconocimiento de emociones, pero todos se fundamentan en lo mismo: adquirir unas señales (imágenes, texto, sonido, etc) de una persona, extraer unas características determinadas y posteriormente realizar una clasificación.

En este apartado se mostrará exclusivamente la parte del estado del arte, relacionada con las técnicas de extracción de características y la clasificación de las emociones en rostros, ya que es el objeto principal de investigación del mismo (se obviará la parte de adquisición de datos). En este cometido se destacan dos líneas de estudio:

- En primer lugar, la utilización de software de reconocimiento facial para la extracción de características y la posterior clasificación. Se asentan en el «tracking» facial y la formación de un modelos superpuestos (3D o 2D), en imitación a la estructura muscular facial.

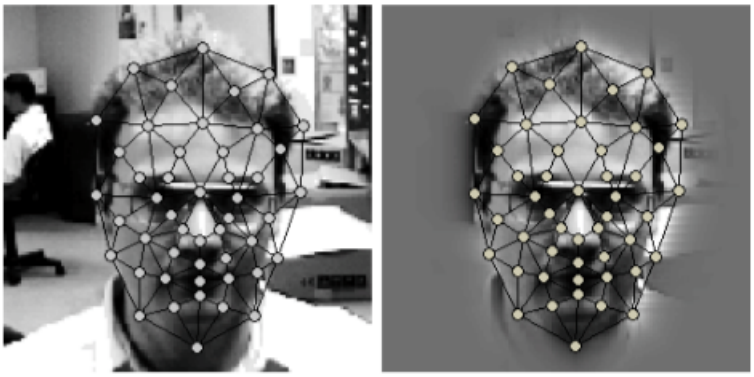

Figura 4: Grafo de PersonSpotter's y supresión de fondo. Fuente: [15]

  Una vez se ha extraído un modelo facial ya es posible extraer los parámetros pertinentes utilizando, por ejemplo, el sistema de codificación facial FACS (Facial Action Coding System) que declara unas unidades de movimiento muscular para distintas partes de la cara. Estos parámetros de movimiento como la posición, dirección, etc, de músculos se pasarían a un clasificador, siendo los más comunes según [15]: «Naïve Bayes» (NB), «Tree Augmented Naïve Bayes» (TAN), «Stochastic Structure Search» (SSS), «Modelos Ocultos de Markov» (HMM) y «Redes Neuronales» (RNA).

- En segundo lugar, aparecen algoritmos de «Deep Learning» en concreto las «Redes Neuronales Convolucionales» (como en el caso de este proyecto) donde se aprende directamente de los píxeles de las imágenes suministradas. En el siguiente apartado se entrará más en detalle sobre el funcionamiento de esta técnica.

3. REDES NEURONALES CONVOLUCIONALES.

3.1. Introducción.

Las redes neuronales son una de las técnicas más populares de «Deep Learning». Al igual que otras técnicas de IA se basa en copiar el modelo biológico humano, en este caso, del funcionamiento del cerebro, formado por la interconexión de neuronas. La idea principal se apoya en que la interconexión una gran cantidad de elementos simples pueden formar un modelo complejo y, por tanto, con operaciones entre elementos simples se pueden resolver problemas complejos.

En esta sección se tratará la base teórica de las redes neuronales y las redes convolucionales para el paradigma del aprendizaje supervisado.

3.2. Redes Neuronales Artificiales.

Una Red Neuronal Artificial es un modelo matemático formado por una serie de operaciones que, para un vector de entrada x ofrece un vector de salida distinta $o(x)$. En concreto, la tipología de RNA conocida como Perceptrón Multicapa, está constituida por 3 tipos de capas completamente conectadas:

- Capa de entrada: Se puede entender como el vector de datos de entrada antes de realizarse ninguna operación. Si se tiene un vector x de datos, cada valor desde ($x_1 ... x_n$) constituye una neurona de entrada. En el caso de una imagen se tiene tantas entradas como píxeles tiene la imagen.

- Capa Oculta: Está formado por las neuronas ocultas y contiene todos los cálculos intermedios de la red. Cada neurona oculta contiene un peso y bias que son los elementos que modificarán los datos de entrada a «algo distinto». El número y tamaño de capas ocultas es variable y no determinado.

- Capa de salida: Es la capa en la que se realiza la clasificación y tiene tantas neuronas como posibles clases que presenta el problema asociado. Al igual que la capa anterior cada neurona tiene pesos y bias.

Una vez se conoce la estructura y la división en capas, es importante conocer como funcionan internamente (se deja lado la capa de entrada, la cuál se utiliza exclusivamente como ejemplo para la introducción del vector de datos).

La transformación que se aplica al vector de datos, en cada capa oculta, sigue la siguiente fórmula:

$$h(x) = s(Wx + b)$$

donde b es el vector de bias, W es la matriz de pesos y s la función de activación.

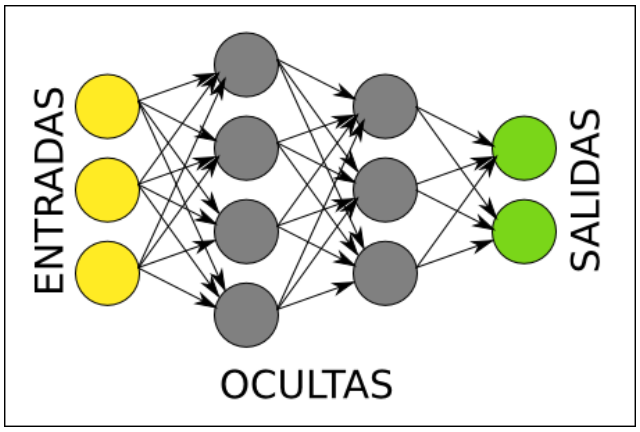

Figura 5: Topología Multi-Layer Perceptrón.
Fuente: URL.

3.2.1. Funciones de activación.

Una función de activación s, se utiliza para dar una no linealidad al problema y por tanto que la red sea capaz de resolver problemas más complejos. Existen distintas funciones de activación, siendo las más conocidas y usadas:

$$tanh(a) = \frac{\exp^{a} - \exp^{-a}}{\exp^{a} + \exp^{-a}} \tag{1}$$

$$sigmoid(a) = \frac{1}{1 + \exp^{-a}} \tag{2}$$

$$ReLU(a) = max(0, a) \tag{3}$$

Se debe elegir la función de activación, al igual que otros parámetros de la red neuronal, de forma razonada ya que cada una contiene sus ventajas y desventajas. En el caso de (1) y (2) computacionalmente son mucho más costosas, ya que se calcula la exponencial, al contrario que (3) donde simplemente se hace una operación de máximo.

También, según el tipo de función se han de inicializar los pesos de forma distinta. Esto se debe a que se tiene que lograr romper la simetría entre diferentes neuronas de una misma capa oculta y evitar valores que provoquen una sobresaturación (donde los gradientes no se propagarán bien) o una activación excesivamente lineal (donde no se logrará llegar a un resultado bueno o se tardaría demasiado).

Glorot y Bengio en [2], entre otras cosas, proponen unos métodos de **inicialización de pesos** basados la longitud del vector de entrada (n_{in}) y el número de neuronas en una capa oculta (n_{out}). Así pues, para las funciones de activación propuestas:

- Función **tangente hiperbólica** (tanh): Se inicializan los pesos de forma aleatoria siguiendo una distribución uniforme entre el intervalo $(-r,\, r)$, donde $r = \sqrt{\frac{6}{n_{in} + n_{out}}}$
- Función **sigmoide**: Se inicializan los pesos de forma aleatoria siguiendo una distribución uniforme entre el intervalo $(-r,\, r)$, donde $r = 4\sqrt{\frac{6}{n_{in} + n_{out}}}$
- Función **ReLU**: En la literatura no se establece una inicialización fija, ya que «ReLU» resuelve este problema. No obstante, se considera para este caso el mismo tipo de inicialización que para la tangente hiperbólica.

Los parámetros «bias» se puede inicializar a 0 en todos los casos.

Además de este tipo de funciones, en la capa de salida (también llamada de regresión logística) existe una función conocida como «softmax», que básicamente se encarga de normalizar la salida lineal, a valores entre [0, 1] (cuya suma es 1), el resultado de la cuál es la probabilidad a posteriori de clase. Este resultado se utiliza para calcular la función de coste y, si se realiza la operación $argmax$, se predice la etiqueta o clase a la que debería formar parte el conjunto de datos de entrada.

3.2.2. Aprendizaje por back-propagation.

Tanto W como b son parámetros que se aprenden, y por tanto se modifican a lo largo del entrenamiento de la red. El entrenamiento de la red se divide en dos partes, el paso «forward» donde se realiza la clasificación (para predecir la clase o resultado destino) y el paso de realimentación o «backward» donde la red aprende según el resultado anterior.

En general el algoritmo «backpropagation» se basa en minimizar una función de coste, teniendo en cuenta la o probabilidad obtenida por la función «softmax», asociada a la clase correcta[2]. Generalmente este coste, ℓ, se calcula como el opuesto de la log-verosimilitud:

$$\mathcal{L}(\theta = \{W, b\}, \mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} \log\left(P(Y = y^{(i)}|x^{(i)}, W, b)\right) \tag{4}$$

$$\ell(\theta = \{W, b\}, \mathcal{D}) = -\mathcal{L}(\theta = \{W, b\}, \mathcal{D}) \tag{5}$$

Como se puede intuir, al aplicar un logaritmo, si el valor de entrada es bajo (cercano a 0) se obtendrá un coste muy alto lo que indicará que probablemente la muestra se ha clasificado de forma errónea y por tanto implicará una mayor modificación de los parámetros (que han contribuido a este resultado).

Para conocer qué parámetros hay que modificar se aplica una retropropagación del error, donde se calcula las derivadas parciales de las capas en función de los parámetros. Si se evalúan esas funciones, es sencillo obtener el resultado para minimizar el coste de la

[2]Se recuerda que el aprendizaje supervisado se tienen a priori las etiquetas correctas de las clases a las que pertenecen los datos.

función objetivo.

Finalmente la modificación de parámetros como tal se realiza con un algoritmo conocido por el nombre de «descenso por gradiente» el cual calcula el coste medio de entre conjunto pequeño de muestras respecto al total. La modificación de pesos que se ha utilizado a lo largo de los experimentos de este trabajo se conoce, en inglés, con el nombre de «Minibatch Stochastic Gradient Descent»:

$$\text{new params} = \text{params} - \text{tasa de aprendizaje} * \text{Gradiente(error, params)} \quad (6)$$

donde «new params» son los nuevos parámetros de pesos modificados; «params» son los parámetros antiguos y «tasa de aprendizaje» («learning rate»), que define la porción de modificación que se debe aplicar.

3.3. Redes Neuronales Convolucionales.

Conocido el funcionamiento básico de las redes neuronales, se explicará de forma breve que son las redes convolucionales y como funcionan. En general una red convolucional no es más que una modificación de una red neuronal básica donde la principal diferencia radica en que los pesos, W, son filtros (de un tamaño deseado, 3x3 por ejemplo) que se convolucionarán[3] directamente con la imagen.

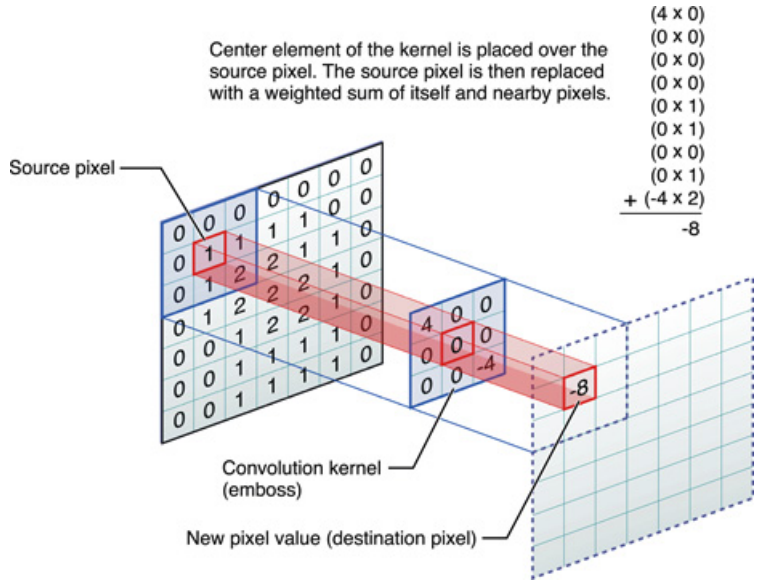

Figura 6: Ejemplo convolución de un filtro con una imagen.
Fuente: URL.

3.3.1. Capas.

Una red convolucional está formada por las siguientes capas:

[3]https://es.wikipedia.org/wiki/Convolución

- **Capa convolucional**: En extrapolación a un perceptrón multicapa, se puede decir que una capa convolucional es una capa oculta, donde se definen la neuronas con los filtros (W) (para convolucionarlos con los datos, generalmente imágenes), bias (b) y la función de activación escogida.

- **Capa de «pooling» o sub-muestreo**: Esta capa opcional se conecta entre la salida lineal de la capa convolucional y la función de activación. Se utiliza para reducir la dimensionalidad de la imagen, lo que reduce el coste computacional de las siguientes convoluciones, número de parámetros e incluso controlar el sobre-aprendizaje de la red.

  Se puede realizar «pooling» de varios tipos, siendo lo más común «max-pooling» y «average-pooling». En el primer caso se elige el valor máximo entre un conjunto de píxeles y en el segundo caso se calcula la media. La dimensionalidad es variable, aunque lo más común es 2x2, 3x3 o 4x4.

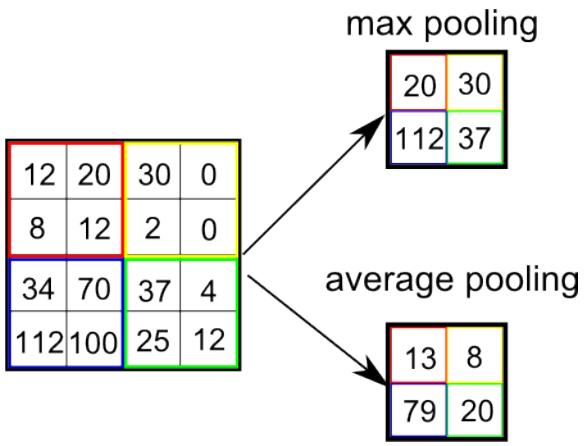

Figura 7: Ejemplo «max-pooling» y «average-pooling». Dim: 2x2.
Fuente: URL.

- **Capa fully-connected**: Está compuesta por una red neuronal completamente conectada (de ahí el nombre), por ejemplo un perceptrón multicapa, que se conecta a la salida de la última capa convolucional de una red. La capa de conexión se conoce como capa de «reshape», ya que convierte la matriz de datos en un vector plano.

Al igual que en las redes neuronales anteriormente explicadas, no existe un número fijo de capas que se deben utilizar, esto es una elección que se considera de acuerdo al problema a resolver. Se debe tener en cuenta que una red excesivamente profunda y con pocos datos puede no llegar a aprender nada y de la misma forma cuando se tiene gran cantidad de datos suele venir bien ir más profundo (mayor número de filtros, capas, etc).

3.3.2. Topología Básica.

Una vez se conocen las capas que constituyen una red convolucional se expone la red convolucional más conocida:

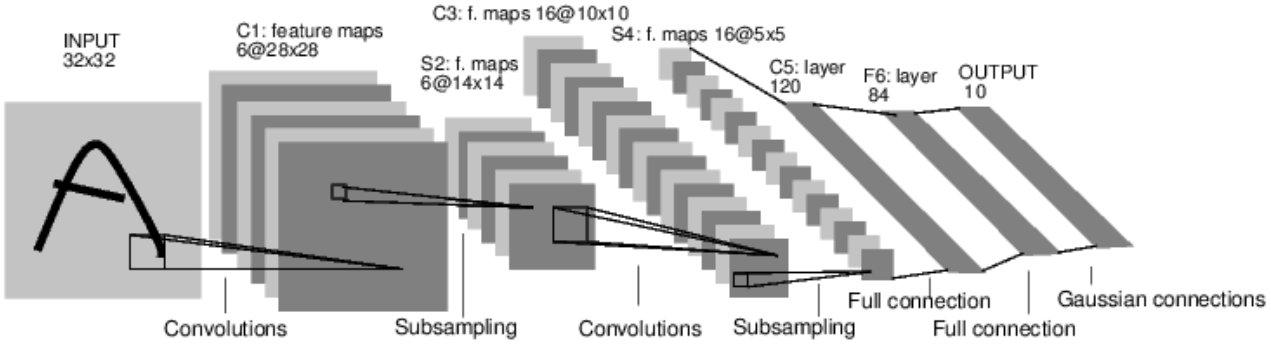

Figura 8: Topología LeNet-5.
Fuente: [3].

La topología LeNet-5 se considera la base de las redes convolucionales. Se diseñó con 2 capas convolucionales con sus respectivas capas «pooling» y «fully-connected» para realizar la clasificación de texto manuscrito sin realizar «demasiado» preproceso a una imagen.

La problemática principal de este tipo de red, y por lo que no desató una revolución del «Deep Learning» en 1998 se debe al coste computacional de realizar la operación de convolución, ya que entrenar una red completa y servible llevaría «demasiado» tiempo[4].

A partir de este momento se empezó a trabajar con otro tipo de hardware que iba apareciendo, como GPGPU, sin embargo no fue hasta el 2012 cuando apareció la conocida como «Alexnet» ([4]) que sentó las bases en cuanto a ejecución de redes convolucionales en GPU y demostró la potencia de las mismas clasificando el conjunto de datos «Imagenet» y ganando la competición «ILSVRC2012», con una gran bajada de error respecto a lo que se había sacado hasta el momento.

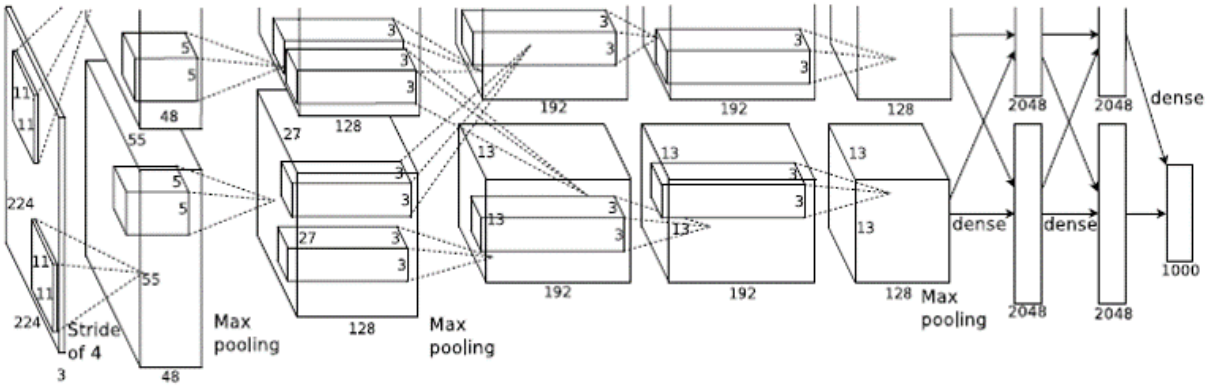

Figura 9: Topología AlexNet.
Fuente: [4].

[4]En la época, con redes neuronales básicas se podía obtener buenos resultados con mucho menos tiempo de entrenamiento.

3.3.3. Estado del Arte.

A partir de la aparición de [4], se ha llevado a cabo un cambio en la investigación de «Deep Learning», comúnmente se suele decir «*antes de 2012 nadie usaba las redes convolucionales, ahora todo el mundo las usa*» y debido a esto han aparecido diversas topologías y tipos de redes que presentan unos resultados mejores que la original «Alexnet» (Figura: 9).

En general, se considera que las redes convolucionales más importantes, son aquellas que marcan el estado del arte cada año ganando la competición «ILSVRC»[5], que desde 2012 han sido: «ZFNet» ([5]) en 2013, «GoogleNet»([6]) y «VGGNet»([7]) en 2014 (distintas categorías) y «ResidualNet» ([8]) en 2015.

Del estado del arte citado, «ZFNet» se considera una forma de optimizar la «Lenet-5» (citando [5], *el propósito es entender porque funcionan bien las redes convolucionales*) y «Googlenet» se presenta como una red «demasiado» profunda, como para que se pueda llegar a obtener unos resultados concluyentes por el tiempo que supondría implementarla y entrenarla, por lo que no se evaluarán en este trabajo. De las dos redes restantes:

En primer lugar se presenta la «**VGGNet**», que aunque no logró el premio en clasificación, se considera la red preferida para la extracción de características en imágenes por los usuarios, ya que presenta una arquitectura muy uniforme, con filtros de 3x3 y «pooling» de 2x2.

El principal concepto que se presenta es el de utilizar dos capas convolucionales seguidas, la primera sin «pooling», con una tamaño menor de filtro, para lograr una optimización en el coste computacional. Ejemplo: Para un mismo tamaño de imagen, una capa convolucional de 32 filtros de 5x5 (32@5x5) computa un coste computacional de, $32 \cdot 5 \cdot 5 = 800$ operaciones. En cambio si se realiza la operación equivalente, pasar dos veces un filtro 3x3, se tiene $32 \cdot 3 \cdot 3 \cdot 2 = 576$ operaciones.

Más allá de esto, los buenos resultados en la competición de clasificación se deben a la profundidad de la red y al uso de diferentes técnicas de RNA que se verán en el apartado siguiente.

En segundo lugar, se tiene la «**ResidualNet**» que parte de las dos ideas fundamentales de la competición del año anterior, ir mucho más profundo y utilizar filtros pequeños.

Como se dice en [8], conseguir mejores resultados de entrenamiento no es tan fácil como apilar un gran número de capas, ya que la profundidad satura la precisión de clasificación y la degrada en el momento que la red empieza a converger. Si por el contrario se analiza una red menos profunda, se observa que este fenómeno no aparece, por lo que se propone hacer una fusión entre red poco profunda y una más profunda. La mejor forma de realizarlo es sumando la salida, a modo de mapa residual, de las capas menos profundas a las siguientes (operación elemento-a-elemento) y se demuestra que esto funciona ya que no se

[5] http://www.image-net.org/challenges/LSVRC/

pierde nunca información en una capa, sino que se gana.

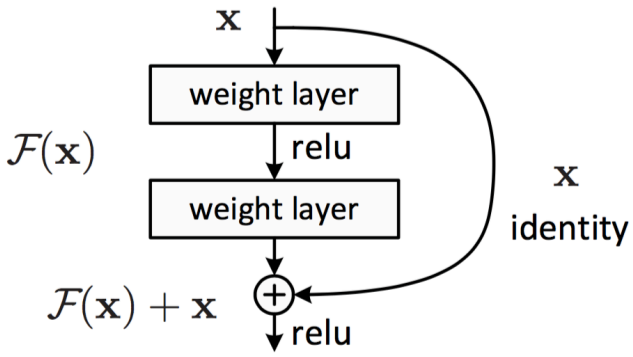

Figura 10: Capa Residual.
Fuente: [8].

En la imagen anterior se ve la formación de la capa residual básica. La conexión directa con la entrada permite al gradiente encontrar, de forma óptima, que parámetros se deben modificar en el entrenamiento y por tanto se deja de lado el de tener que extenderlo por toda la red.

3.4. Técnicas de RNA.

Una vez se conoce el estado del arte, en este apartado se expondrán las técnicas más importantes de la actualidad en relación a las redes neuronales artificiales. En general la aparición de estas técnicas se debe a la necesidad de combatir ciertos problemas específicos en entrenamiento de modelos. Todas las técnicas que se tratarán a continuación se pueden aplicar tanto a redes neuronales básicas como a convolucionales.

3.4.1. Dropout.

Uno de los problemas de las redes neuronales profundas es el sobre-aprendizaje. «Dropout» es una técnica que trata de corregir esto y se basa en la idea de que, para un mismo problema, se consiguen mejores resultados entrenando varias redes con porciones de los datos originales y combinando los resultados en fase de test.

Hinton et. al, en [9], advierten que la premisa anterior no puede ser factible para un sistema en el que se requiera de respuestas rápidas y por tanto lo abordan de forma distinta. Se propone «dejar de lado» un numero determinado de neuronas (tanto ocultas como visibles).

Si se desactiva una neurona temporalmente, se deja fuera de la propia red y por tanto se pierden las conexiones entrantes y salientes que ofrecía. Esto favorece a un entrenamiento más uniforme (y eficiente) del modelo ya que se simula el entrenamiento de varias subredes combinadas en una sola. Todo esto se realiza en base a una probabilidad se retención de neurona y a una máscara que se calcula de forma aleatoria según esta probabilidad.

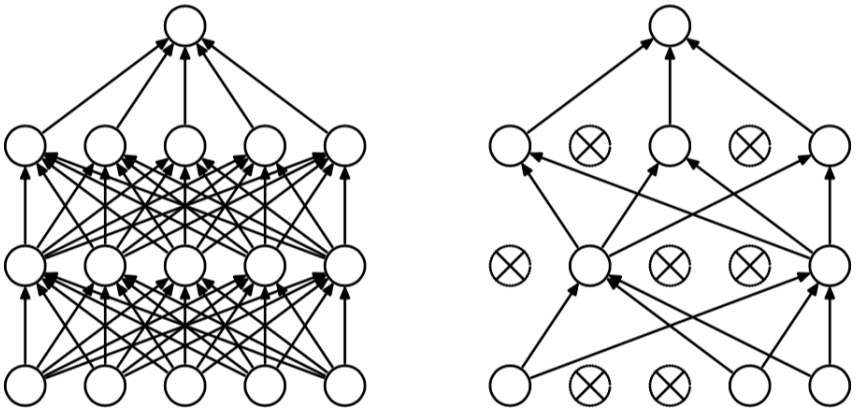

Figura 11: Funcionamiento Dropout. A la izquierda se observa la red normal. A la derecha la red con Dropout. Fuente: [9].

Se tiene que destacar que, aunque en fase de entrenamiento se desactivan las neuronas como tal, en fase de test no tiene sentido, porque se estaría testeando un número aleatorio de subredes y no la completa. La forma de realizarlo correctamente es, en fase de test, multiplicar las neuronas por la probabilidad de retención (que tenga cada capa asignada).

Si se entra en el detalle de los valores recomendados como probabilidad de retención, en el propio papel se define un valor de $0{,}5$ para redes neuronales básicas.

Para el caso de las redes convolucionales no se especifica un valor típico, no obstante existe una tendencia a utilizar un «dropout» ascendente según el número de capas convolucionales (lo más común es utilizar en la capa convolucional, anterior al «reshape», la misma probabilidad que en la capa «fully-connected»). Esto se hace ya que la primera capa convolucional contiene muy pocos parámetros e información vital para el correcto aprendizaje de la red, si se aplica una probabilidad muy elevada se corre el riesgo de cancelar el aprendizaje. A medida que se aumenta el número de filtros en las siguientes capas convolucionales se puede ir aumentando esta probabilidad de dropeo sin problemas.

Con el objetivo de mejorar «dropout», ha aparecido en la actualidad una técnica conocida como «DropConnect» cuya diferencia con el original radica en qué conexión se desactiva en fase de entrenamiento. Con este cambio se «dropean» de forma aleatoria algunos valores del vector de pesos de una neurona y no la propia neurona. Hasta el momento sólo se ha conseguido demostrar su eficacia en el conjunto «MNIST», por lo que no se abordará en este trabajo.

3.4.2. Batch Norm.

«Batch Norm» o «Mini-Batch Normalization» es una técnica que se centra especialmente en optimizar y acelerar el entrenamiento de una red. Se basa en reducir la covarianza interna de los datos, o más simplificado, reducir la frecuente variabilidad que hay entre los datos de un mismo conjunto (por ejemplo en imágenes cambios en la normalización de luz).

Para la transformación que se aplica se requieren 4 parámetros, por un lado la media y varianza del «mini-batch», con los que se normaliza, y por otro lado dos parámetros que se aprenden (de la misma forma que los pesos y bias), γ y β[6], que sirven para escalar y centrar (respectivamente) de forma correcta los datos:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figura 12: «Batch Normalization Transformation».
Fuente: [10].

Al igual que «dropout» funciona de forma distinta según la fase, entrenamiento o test. En fase de entrenamiento se utiliza, para el calculo de media y varianza, el resultado que ofrece cada «minibatch» (en cada epoch y capa). En cambio en fase de test, para un resultado correcto se debe pre-calcular los resultados de media y varianza según un conjunto «suficientemente» grande de datos de entrenamiento (cuantos más datos se obtiene un resultado más robusto).

También se puede testear, utilizando el cálculo como se realiza en entrenamiento, no obstante se considera que esto es poco útil en cuanto a una utilización real fuera del ámbito de investigación. Si se requiere que en fase de test el «minibatch» esté compuesto por un dato, el error en clasificación conseguido se dispararía, ya que el cálculo de media y varianza para solamente un dato (o pocos datos) no tiene sentido.

[6]En [10] se indica que el parámetro beta, β, sustituye al bias, por lo que se puede eliminar sin problemas.

3.4.3. Gaussian Noise.

Otra de las técnicas más conocidas y usadas en el entrenamiento de redes es la de añadir ruido gaussiano para distorsionar los datos ligeramente y reducir la aparición de sobre-aprendizaje. En general se trata de añadir ruido en la salida lineal de una capa oculta antes de la función de activación.

Los estudios realizados, como se puede comprobar en [13], muestran que funciona bastante bien, en concreto en redes neuronales como el perceptrón. No obstante el uso de ruido debe tratarse con cautela, ya que encontrar la cantidad de desviación necesaria suele ser una tarea complicada.

3.4.4. Data Augmentation.

Aunque no se trata de una técnica exclusivamente utilizada en «Deep Learning» y se ve en multitud de escenarios, «data augmentation» es una técnica muy usada en el entrenamiento de clasificadores automáticos.

Se fundamenta en crear nuevos datos a partir de los datos existentes, cosa que aporta robustez al modelo creado y se reduce el posible sobreaprendizaje. Por esta razón se suele utilizar en conjuntos de datos pequeños, aunque también funciona adecuadamente en conjuntos grandes. Existen dos modos comunes de utilizar «data augmentation»:

- **Offline**: Recibe este nombre debido a que todas las transformaciones se aplican como un preproceso de los datos, antes de lanzar el entrenamiento de un modelo. Por ejemplo, en un conjunto de 1000 imágenes, si se aplican 7 transformaciones, se tendrían 8000 imágenes[7] durante todo el entrenamiento. La ventaja principal de este modo radica en la robustez que añade al modelo, ya que se sabe que se utilizarán todas las transformaciones aplicadas y la rapidez de convergencia que aporta. Por el contrario la principal desventaja se observa en el tiempo de entrenamiento por «epoch» que añade, en el caso del ejemplo con un factor de x8 (aproximadamente).

- **Online**: Al contrario que el modo anterior, se aplican las transformaciones durante la ejecución del entrenamiento, sobre cada «mini-batch» y de forma aleatoria. Esto hace que se reduzca la necesidad de «dropout» ya que, por el numero de transformaciones que se pueden crear (casi infinitas) nunca se debería llegar a observar un sobreaprendizaje[8].

Vistos lo dos modos de funcionamiento, algunas de las transformaciones más utilizadas (entre otras) son: imagen espejo (eje vertical), traslaciones para simular movimiento, cambios de escala (reducción y aumento), giros (hasta un máximo razonable, como 45º), etc.

Por último, se aconseja aplicar todas las transformaciones utilizadas al conjunto de test (en modo «offline»), independientemente del modo en que se ha aplicado al conjunto de

[7] Cálculo: 7 * 1000 + 1000 originales.

[8] Se debe utilizar un número suficientemente grande de transformaciones.

entrenamiento, para sumar las probabilidades a priori de la salida lineal de la capa de regresión logística (en el caso de las redes convolucionales la salida lineal de la «fully connected»), antes de ejecutar el cálculo de la función de activación «softmax». Esto funciona realmente bien, ya que una red entrenada con transformaciones es robusta frente a transformaciones y en muchas ocasiones una imagen transformada posee más probabilidad de pertenecer a la clase correcta.

4. ANÁLISIS DE REQUERIMIENTOS.

4.1. Introducción.

La creciente utilización de las redes neuronales y en concreto las redes neuronales convolucionales han propiciado la aparición de herramientas (toolkits) para poder definir modelos y entrenarlos de forma sencilla. Una de las características del aprendizaje automático, es que la programación o las matemáticas no deben ser un obstáculo en su implantación, por ello normalmente este tipo de «toolkits» se diseñan con la idea de llegar al máximo número de personas, que puedan convertirse en usuarios de forma sencilla, con conocimientos muy básicos en «Reconocimiento de Formas» o «Inteligencia Artificial».

En esta sección se va a proceder a realizar una descripción de la tecnología y herramientas (software y hardware) necesarias para la elaboración de este proyecto, así como el origen de los datos que se han utilizado.

4.2. Lenguaje de programación y librerías.

El lenguaje que se ha escogido para la implementación de todo el código es **Python**.

Python es un lenguaje de programación interpretado, que nace con la idea de proporcionar un código fácilmente legible y entendible. Entre otras cosas permite varios estilos de programación como la programación orientada a objetos o la programación funcional.

En general con Python se simplifica de forma importante la creación de código, pero al mismo tiempo tiene capacidad de extensión en módulos de otro lenguaje (C, C++, ...) cuando se requiere algo de programación avanzada o hardware. Estas características hacen de Python un lenguaje muy valorado en aplicaciones de «Deep Learning».

Python también es un lenguaje muy extendido por lo que existe una gran cantidad de librerías que añaden una funcionalidad muy variada con sólo importarlas desde el código. Algunas de las librerías necesarias para el desarrollo de este trabajo son:

- **Numpy**: Extensión de python que agrega soporte para vectores y matrices, así como funciones matemáticas de alto nivel. Numpy permite, entre otras cosas, definir la precisión de los tipos de datos con los que se trabaja (int64, float32, etc).

- **Pickle**: Modulo que permite serializar y des-serializar objetos de python y convertir la jerarquía que tenga en un stream de bytes. Posteriormente permite cargar el stream de bytes y reconvertirlo al mismo tipo de objeto origen. Por ejemplo, permite guardar vectores y matrices Numpy en disco y posteriormente recuperarlos sin perder su estructura.

  Tiene dos variantes, «cpickle» que es una versión acelerada que trabaja sobre C y el original, además de ofrecer la posibilidad de utilizar protocolos especiales de guardado. Se ha de tener en cuenta que para grandes cantidades de datos se satura, haciendo

imposible la obtención de un guardado óptimo (problema aparentemente solucionado en python 3).

- **h5py**: Versión mejorada de pickle que trabaja con el formato binario HDF5. Está especialmente diseñado para trabajar con matrices Numpy de gran tamaño, lo que propicia su uso en el almacenamiento de grandes conjuntos de datos (del orden de terabytes de memoria).

- **cv2**: Nombre que recibe la API OpenCV a modo de «wrapper» para python, diseñada para resolver problemas de visión por computador o visión artificial. Utiliza parte de la funcionalidad del lenguaje C++ (para el cuál está diseñado OpenCV de forma nativa) en combinación con los vectores y matrices Numpy.

- **gzip**: Librería utilizada para realizar tareas de compresión de datos en disco duro.

- **timeit**: Modulo de control de tiempo de ejecución de un algoritmo. Es especialmente útil para controlar tareas de «Deep Learning».

Entre las librerías utilizadas también se encuentra el toolkit de redes neuronales, no obstante por la importancia del mismo se desarrollará en el siguiente apartado.

4.3. Toolkit de RNA.

La elección del toolkit de RNA es una elección importante, ya que ayuda al usuario a definir expresiones matemáticas que formaran parte de grafo de computación final al crear, por ejemplo, una red neuronal.

Existen una gran variedad de toolkits como: Theano, Torch, Caffe, TensorFlow, ..., por lo que, antes de la elección, se considera conveniente tener una idea preconcebida de la funcionalidad que se requiere (aunque básicamente todos hacen lo mismo). Algunas de las condiciones para la elección del toolkit de RNA en este trabajo han sido:

- Tiene que trabajar sobre Python por las ventajas y facilidad de programación que ofrece.

- Debe soportar la computación en GPU, de lo contrario sería complicado entrenar una red convolucional por el tiempo requerido.

- Debe poseer una documentación amplia.

Entre todas las opciones, los toolkits que cumplen estas características son Theano y TensorFlow. Ambos funcionan de la misma forma, no obstante TensorFlow es una librería recientemente liberada por Google, por lo que no se espera que haya una comunidad lo suficientemente grande como para resolver dudas y problemas de forma eficaz. Al contrario, Theano lleva mucho tiempo siendo una de las librerías más utilizadas por lo que la elección ha sido esta.

4.3.1. Theano.

Theano, [16], es una librería de Python que permite definir, optimizar y evaluar expresiones matemáticas que implican matrices multidimensionales de manera eficiente. En Theano se trabaja con programación funcional (o simbólica), en la que se define en primer lugar las operaciones que se deben realizar y posteriormente se crea una función a modo de modulo con «inputs» y «outputs» correspondientes para evaluar esas expresiones.

Algunas de las características de Theano son:

- Integración con Numpy: Theano admite como entrada a sus funciones cualquier tipo de array Numpy.

- Uso transparente de GPU: Theano trabaja con el compilador de C para GPU Cuda de nVidia, que permite evaluar funciones y realizar operaciones de forma ultra-rápida simplemente activando el uso de la tarjeta gráfica.

- Detección de errores: Theano posee una gran unidad de testeo y verificación que ofrece información detallada al usuario en caso de haber definido alguna expresión de forma incorrecta (y otros problemas).

La funcionalidad principal que aporta Theano, a diferencia de Numpy, es la creación automática del grafo de computación para computar gradientes.

Cabe destacar que existen un gran número de «capas intermedias» Theano-usuario a modo de librerías de redes neuronales, que facilitan el entrenamiento de modelos, su evaluación y la presentación de resultados. Algunas de estas librerías son: **Keras**, que puede trabajar sobre TensorFlow y Theano; **Blocks**, que se presenta como un framework para ayuda y control de redes neuronales sobre Theano; **Lassagne**, que está pensada para reducir la complejidad de Theano y **Pylearn2**, que es una librería de aprendizaje automático general también diseñada sobre Theano. En este trabajo no se ha utilizado ninguna de estas librerías, ya que se quería tener un control completo del grafo de computación sin limitaciones, por lo que **todo el código se ha diseñado sobre Theano puro**.

4.4. GPU.

Como se ha comentado en apartados anteriores, uno de los principales requerimientos es poder utilizar la computación sobre GPU. NVidia, como fabricante, ofrece la arquitectura CUDA (Arquitectura Unificada de Dispositivos de Cómputo) para aprovechar la potencia de las tarjetas gráficas y ganar un notable rendimiento en este tipo de aplicaciones de «Deep Learning». CUDA sólo funciona con algunas de las tarjetas gráficas del propio fabricante: GeForce, ION Quadro y Tesla GPUs.

En este trabajo se ha utilizado la GPU: **NVIDIA GeForce GTX 970**:

Figura 13: Gygabite GeForce GTX 970.

Las principales características de las tarjeta mencionada son las siguientes:

- **VRAM** (Cantidad de memoria): 4096 MB.
- Tipo de memoria: GDDR5.
- **Ancho de banda** GPU-Placa base: 225 GB/s.
- Frecuencia de reloj GPU: 1178 MHz.
- Frecuencia memoria vram: 1753 MHz.

4.5. Conjunto de datos.

El conjunto de datos que se ha utilizado se ha obtenido de una competición de kaggle, [1], titulada «Competición de Reconocimiento de Expresiones Faciales». Este conjunto de datos completamente nuevo está preparado por Pierre-Luc Carrier y Aaron Courville como parte de una investigación y no es un conjunto famoso, más allá de esta competición.

Los datos consisten en 35887 imágenes de tamaño 48x48 en escala de grises. De las 35887 imágenes, las primeras 28709 forman parte del conjunto de entrenamiento y las restantes 7178 del de test[9].

[9]En la competición se dividió el test en dos (3589 x 2), siendo la primera parte test público y la segunda privado para determinar al ganador entre los participantes. En este trabajo se utiliza siempre el test completo para valorar los modelos.

La tarea llevada a cabo en la competición es la de clasificar las emociones entre 7 categorías: 0 = Enfado, 1 = Asco, 2 = Miedo, 3 = Felicidad / Alegría, 4 = Tristeza, 5 = Sorpresa y 6 = Neutro.

(a) Enfado (b) Asco (c) Miedo (d) Felicidad (e) Tristeza (f) Sorpresa (g) Neutro

Figura 14: Ejemplos de imágenes de cada una de las clases.

Como datos importantes a conocer: los organizadores realizaron un pequeño estudio de la precisión de clasificación humana para conjunto y se estableció en $65\,\% \pm 5\,\%$; se avisa en [1] que debe haber errores en etiquetado y por último, el ganador de la competición obtuvo un resultado de error de test privado de $28{,}838\,\%$ y en el el test público de $30{,}23\,\%$. Teniendo en cuenta los datos anteriores, se ha establecido, como meta u objetivo, **obtener modelos que mejoren la precisión humana**[10].

Los resultados de los 5 primeros clasificados son:

Ranking	Nombre Red / Equipo	Error (%)
#1	RBM	$29{,}53$
#2	Unsupervised	$30{,}83$
#3	Maxim Milakov	$31{,}15$
#4	Radu+Marius+Kristi	$32{,}60$
#5	Lor.Voldy	$35{,}09$

Tabla 1: Resultados competición con la media de test público y privado.

[10]Cuando se rebasa la precisión humana, se puede afirmar que un modelo es buen representante de la tarea que está llevando a cabo.

5. EXPERIMENTACIÓN Y RESULTADOS.

5.1. Introducción.

Una gran cantidad de tiempo dedicado a este trabajo se ha empleado en programación y ejecución de experimentos, sin duda partes fundamentales en investigación para encontrar unas conclusiones factibles y reales.

En este apartado se ofrecerá una perspectiva amplia de la experimentación que se ha llevado a cabo en este TFM, desde la preparación de los datos del conjunto utilizado, pasando por la descripción de la capa intermedia de Theano creada, así como la evaluación de las distintas redes convolucionales utilizadas.

5.2. Preparación de datos.

Los datos, descritos en el apartado 4.5, los podemos encontrar en «URL».

Se da un fichero comprimido (.gz) de 91.97 MB. Al descomprimir el fichero se obtienen varios archivos, de los cuales interesa el llamado «fer2013.csv». Un archivo «CSV» es un tipo de documento en formato tabla, donde las columnas están separadas por comas y las filas por saltos de línea.

El archivo en cuestión contiene más de 30000 filas de 3 columnas cada una, formada por etiqueta de clase (de 0 a 6), píxeles de la imagen (con valores entre 0-255) separados por espacio en blanco y conjunto al que pertenece (entrenamiento, test público, test privado).

Una vez analizados cabe destacar que se han creado 4 conjuntos diferentes que se han utilizado a lo largo de toda la experimentación:

1. El primer conjunto está formado por los datos base sacados del fichero anterior, eso si convertidos a un formato más amigable para su uso a imitación del conjunto «MNIST» proporcionado en la página web de la librería Theano. Los datos de los píxeles, no normalizados, se convierten en un vector de vectores planos numpy de tipo «float32» y las etiquetas[11] de clase en un vector numpy de tipo «int32».

   En total este conjunto ocupa un espacio en disco de 300 MB y está formado por 28709 imágenes de entrenamiento y 7178 de test.

2. El segundo conjunto, ya en el campo de «Data Augmentation Offline», está constituido por las imágenes originales más la incorporación de la operación espejo de cada imagen. Esta operación se realiza con OpenCV utilizando la función:

```
img_flip = cv2.flip(img, flipCode = 1) # flipCode = 1 indica eje de rotación vertical.
```

[11] Algunos toolkits requieren las etiquetas como un vector en formato bit, es decir, en el caso de estudio la etiqueta 3 sería (0, 0, 0, 1, 0, 0, 0). Esto no es un problema en Theano, ya que se admite como número entero.

En este conjunto se almacenan dos versiones del test, una con los datos originales y otra con datos aumentados para, como se comentó en 3.4.4, realizar un testeo especial. También se guarda un conjunto de validación preparado con sus respectivas etiquetas y sin aumento de datos (sacado del conjunto de entrenamiento), para validar correctamente los modelos (de igual forma se guarda un conjunto de entrenamiento reducido con son los datos de validación quitados). La idea perseguida es la de dotar al conjunto de datos la mayor versatilidad posible en ejecución.

Figura 15: Imagen original y espejo. Etiquetada como alegría.

Este conjunto ocupa un espacio de 1.17 GB en disco. Está formado por 57418 imágenes de entrenamiento básicas y 7178 de test, más los conjuntos secundarios que contienen 7418 imágenes de validación, 50000 de entrenamiento y 14356 de test especial. Dimensionalidad: 48 x 48.

3. El tercer conjunto es una modificación del anterior, buscando la idea de conseguir aumentar el número de datos de forma considerable. En esta primera gran modificación se aplica la técnica de «cropping» o recorte para simular desplazamiento en las imágenes. Esto se puede conseguir de distintas formas, pero la escogida se queda con 5 recortes, las 4 esquina de la imagen más el recorte central.

Figura 16: Procesado 5 crop con imágenes espejo. Etiquetada como alegría.

Lo más común en este tipo de modificación es escoger un número píxeles a desplazar y quedarse con todas las imágenes posibles con saltos de píxel 1 a 1. Por ejemplo si se desplaza 4 píxeles por fila y columna, el resultado son 16 imágenes. En un primer momento se decidió utilizar esta modificación, pero por previsión del tiempo que tardaría en entrenarse un modelo se desestimó a favor de testear el «5 crop». Se ha aplicado una variación de 4 píxeles, modificando directamente el array numpy (sin opencv); las imágenes resultantes son de 44x44.

Este conjunto está formado por 287090 imágenes de entrenamiento y 7178 de test con el recorte central. De los subconjuntos se tiene: 30000 imágenes para validación y 257090 para entrenamiento, con la transformación completa; 71780 con las transformaciones para test especial. En total ocupa 4.71 GB.

4. El cuarto, y último, conjunto preparado también es una modificación del 2. En este caso a diferencia del anterior se ha querido realizar varias modificaciones que refuercen el entrenamiento de la red más allá de solamente movimientos de píxeles. Es una modificación propuesta y usada por el ganador de la competición con el conjunto de uso, así que al menos, en teoría, debe ser una modificación fiable.

   Se aplica pues 7 modificaciones (sin contar con el espejo): Desplazamiento 3 píxeles a derecha y 3 a izquierda desde el recorte central, recorte central, giro a 45º y -45º y por último dos escalados, aumentando a 1.2 (con recorte central) y reducción según la dimensión final deseada. Las imágenes se han dejado con una dimensión de 42x42 (reducción de 6 píxeles). Un ejemplo que ilustra las modificaciones es el siguiente:

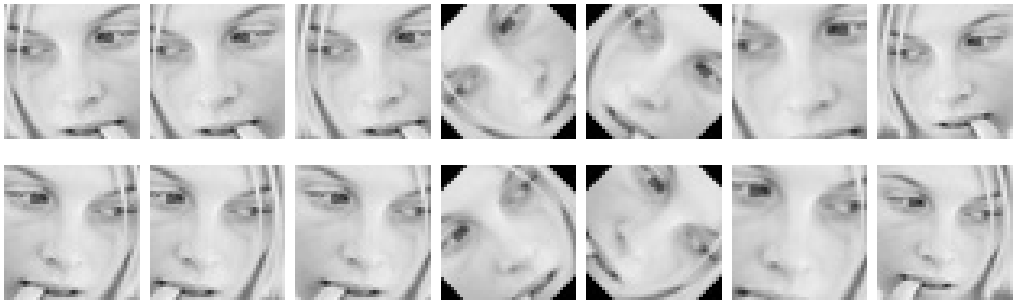

Figura 17: Procesado variado. Etiquetada como miedo.

De entre las modificaciones realizadas los desplazamientos se modifican directamente con la matriz numpy, no obstante las 2 restantes (sin contar la operación espejo) requieren del uso de opencv:

- El giro de las imágenes requiere una matriz de rotación (una por giro) y su correspondiente función:

```
M = cv2.getRotationMatrix2D((dim_h/2, dim_v/2), angle, 1)
img_rot = cv2.warpAffine(img,M,(dim_h,dim_v))
```

- El escalado requiere una función (por escalado):

```
img_res = cv2.resize(img,None,fx=1.2, fy=1.2, interpolation =
cv2.INTER_CUBIC)
```

En total el conjunto queda con 401926 imágenes de train (x14 respecto al original), con 7178 imágenes con recorte central para test. Los subconjuntos se quedan con, 40180

imágenes para validación, 361746 para entrenamiento (cuando se activa validación) y 100492 para test especial. Espacio en disco: 6.28 GB.

Los 4 conjuntos, inicialmente se intentaron preparar con la herramienta base «pickle» de python. No obstante, no se sabe si por el uso de python 2.7 o por el tamaño, con los conjuntos 3 y 4 pickle dejaba de funcionar. La solución más eficaz fue cambiar de «pickle» a «h5py», el cual almacena todos los conjuntos sin problemas (4 o 5 segundos para el más grande), además de proporcionar una carga de datos ligeramente más rápida.

5.3. Detalle de la programación.

Conocidas las transformaciones aplicadas a las imágenes y los diferentes conjuntos que se han utilizado, se va a realizar una descripción de la programación realizada con Theano.

El objetivo principal buscado durante el transcurso de la programación ha sido utilizar Theano como base y crear una capa intermedia para lanzar experimentos de forma óptima. Esto, aunque bien hecho es una gran ventaja, en cuanto al control de errores y la experiencia que se adquiere, al mismo tiempo requiere de una gran inversión de tiempo.

Principalmente la idea de montar una capa intermedia propia, nace de la cantidad de técnicas que salen a la luz y las pocas actualizaciones que reciben las capas intermedias existentes (en general). Se pensaba que el tiempo invertido en entender una librería ajena, en caso de necesitar una modificación importante era más elevado que entender y usar Theano como tal.

Conocidos los principios, en cuanto a la parte práctica se ha creado un proyecto python exclusivo que contiene todos los scripts. La estructura básica que sigue el proyecto es la siguiente:

- Dos carpetas de almacenamiento de datos, una para los conjuntos de datos y otra para el almacenamiento de los modelos entrenados.
- Una carpeta para todos los sripts relacionados con la carga de datos.
- Dos carpetas para los scripts de redes neuronales, una para las redes convolucionales y otra para redes neuronales básicas.
- Finalmente una carpeta para la inclusión de scripts de utilidades, tanto para redes neuronales como utilidades generales.
- En la carpeta principal del proyecto se encuentra el lanzador de experimentos.

La separación en carpetas se pensó para facilitar la comprensión general del proyecto. Si se requiere modificar ficheros, no hay problema en romper el código ya que todo está estructurado en clases y sin un nivel complicado de programación python. El siguiente código

hace referencia al lanzamiento básico de un experimento para el entrenamiento desde 0 de un modelo:

```
from testModel import test
test(
        learning_rate=0.1,
        L1_reg=0.,
        L2_reg=0.,
        n_epochs=10,
        dataset='data/dataset.h5',
        batch_size=100,
        n_hidden=[1024],
        activation=rna.relu(),
        batch_norm = True,
        p_drop = 0.5,
        p_dropConv = [0.2, 0.3, 0.4, 0.5],
        gaussian_std=0.,
        val=True,
        trainShared=False,
        im_dim=[48, 48],
        savename='TrainedModels/save.pkl'
)
```

Del código anterior, más allá de los parámetros básicos, cabe destacar que:

- **n_hidden** controla el número de capas ocultas de la «fully connected» (si se requieren dos la sentencia seria [1024, 1024]);
- **activation** (función de activación) está compuesta por una variable theano. Se ha facilitado su llamada utilizando funciones de un script de utilidades donde, además de la presente, para la activación sigmoid se usa **rna.sigmoid()** y para la tangente hiperbólica **rna.tanh()**.
- **p_drop** hace referencia a la probabilidad de desactivación de neurona de todas las capas ocultas de la «fully connected» (se comparte).
- **p_drop_conv** es el mismo parámetro que el anterior, pero en este caso una lista de probabilidades, donde cada una afecta a cada capa convolucional por separado. Se deben poner tantos valores como capas convolucionales se tengan en el modelo que se pretende lanzar.
- **val** es un parámetro de tipo booleano que activa la carga y uso del conjunto de validación.
- **trainShared** es un parámetro de tipo booleano que activa o desactiva la carga del conjunto de entrenamiento en la memoria GPU.

Es un parámetro realmente necesario ya que Theano corta el proceso con una excepción si se supera la capacidad de memoria ram gráfica. Por ejemplo, el conjunto 4 contiene más de 400000 imágenes de 42x42 de tipo float32, lo que suma un total aproximado de 2.62 GB que si se junta con una red de muchos parámetros imposibilitaría su entrenamiento.

El método test contiene todas las llamadas necesarias para cargar los datos del conjunto seleccionado, crear los modelos, ejecutar el entrenamiento y guardar los parámetros aprendidos (los parámetros que se guardan son aquellos que se aprenden, por ejemplo en una ejecución sin «batch norm» se guardarían todos los pesos y bias).

Para cambiar la arquitectura de la red convolucional a utilizar se tiene que modificar el script CNN de la carpeta de redes convolucionales. Dentro de ese script se encuentran programadas, en forma de clases con nombre, todas las redes utilizadas en este proyecto.

En cuanto a otra funcionalidad presente se destaca:

- Posibilidad de seguir entrenando un modelo anterior.
- Realización un test especial con la suma de las probabilidades a priori (ver 3.4.4).
- Durante el entrenamiento o test se realizan constantes «print» en la consola python para conocer la información de como va el entrenamiento, como por ejemplo:

  epoch x, validation error x %, test error x %

  si se tiene activado el conjunto de validación.

  epoch x, train cost x, test error x %

  si no se tiene activado el conjunto de validación.

  Entre otra información que se muestra destacar el control de **tiempo de entrenamiento** (al final de todos los epoch) y un **gráfico con curvas de coste de entrenamiento y validación** necesario para aplicar correctamente la metodología empleada.

Entre las modificaciones previstas a corto plazo, para la capa intermedia programada se encuentra, añadir la técnica «DropConnect», añadir la capacidad utilizar la técnica «data augmentation online», incorporar un sistema de almacenamiento de parámetros aprendidos cuando se lanza una excepción y se corta el aprendizaje y por último unificar/limpiar código duplicado.

Todo el código se ha programado con la ayuda de los tutoriales de [16]. Destacar las librerías Keras ([17]), lassagne ([18]) y el canal de Youtube ([19]) que son de gran ayuda, especialmente cuando se realiza el primer contacto con Theano.

5.4. Configuración de redes.

Una vez se ha tratado de que forma se han lanzado los experimentos se procede a conocer la topología de las diferentes redes que se han entrenado y testeado.

En primer lugar, se muestran las redes modificadas extraidas de [7]:

Convolutional VGG Mod Configurations	
A	B
5 weight layer	7 weight Layer
input image	
conv 64@3x3	conv 64@3x3
maxpool 2x2	
conv 128@3x3	conv 128@3x3
maxpool 2x2	
conv 256@3x3 conv 256@3x3	conv 256@3x3 conv 256@3x3
maxpool 2x2	
	conv 512@3x3 conv 512@3x3
	maxpool 2x2
FC-1024	
soft-max	

Tabla 2: Redes ConvNet 11 Layer reducida.

Red	A	B
Número de parámetros	7040	17280

Tabla 3: Número de parámetros de la parte convolucional.

No se ha computado el número de parámetros de la parte «fully connected» ya que el número de neuronas es un valor que se ha modificado a lo largo de la investigación. En el caso mostrado, con la red A solamente una capa oculta de 1024 neuronas aportaría (con un input de imagen de 48x48) 9.44 millones de parámetros (las capas ocultas poseen muchos más parámetros que las convolucionales).

Tanto la red A como B son una modificación de la red «11 layer weights» entre las VGGNet. La principal modificación radica en la reducción de profundidad ya que originalmente están pensadas para una imagen de entrada de unas dimensiones muy superiores (224x224) al de las usadas en este trabajo (48x48).

Se debe tener en cuenta que una reducción «pooling» de más, cuando no se necesita puede empeorar el entrenamiento. En la red B se tendrá salidas de dimensiones entre 2x2 y 3x3, lo que a priori se considera muy bajas con respecto a lo normal (5x5, 6x6) y por tanto

se corre el riesgo de perder información. Se deja a modo de comprobación.

En segundo lugar las redes que contienen partes residuales, en referencia a [8], son:

ResNet Mod Configurations	
C	D
14 weight layers	5 weight layers
input image	
conv 64@5x5	conv 64@3x3
maxpool 2x2	
$\begin{bmatrix}\text{conv } 64@3x3 \\ \text{conv } 64@3x3\end{bmatrix}$ x 2	conv 128@3x3
maxpool 2x2	
$\begin{bmatrix}\text{conv } 128@3x3 \\ \text{conv } 128@3x3\end{bmatrix}$ x 2	$\begin{bmatrix}\text{conv } 128@3x3 \\ \text{conv } 128@3x3\end{bmatrix}$
maxpool 2x2	
$\begin{bmatrix}\text{conv } 256@3x3 \\ \text{conv } 256@3x3\end{bmatrix}$ x 2	
FC-1024	
soft-max	

Tabla 4: Arquitectura redes residuales utilizadas.

Red	C	D
Número de parámetros	21440	4928

Tabla 5: Número de parámetros de la parte convolucional.

Cabe destacar que los cálculos de parámetros en esta ocasión se realizan añadiendo los parámetros de la técnica «batch normalization» ya que las redes residuales lo llevan activado por defecto. El número de parámetros que añade esta técnica es variable, dependiendo de la implementación. En este trabajo se ha empleado 2 parámetros por neurona (1 gamma + 1 beta).

Tanto **C** como **D** son topologías que parten de la idea de la «Residual 18-Layer», no obstante, al igual que el caso anterior, se ha modificado conforme a las necesidades existentes. El principal cambio que se ha realizado ha sido la eliminación de el sub-muestreo por salto de filtro (de 2 en 2 en el papel) a la capa de «max-pooling». Por número de capas convolucionales, se considera a C como la red más profunda entrenada, cosa que puede afectar al rendimiento, tanto por tiempo de computo como por memoria almacenada en la

GPU (se discutirá en los apartados siguientes).

5.5. Experimentación.

Durante el desarrollo del trabajo, se ha ido experimentando con las diferentes técnicas, redes y conjuntos que se han visto a lo largo de este documento. La experimentación es una parte especialmente larga ya que lanzar los suficientes modelos para llegar a una conclusión factible requiere de mucho tiempo.

La experimentación que se ha seguido ha ido evolucionando en función de las necesidades y nuevas técnicas incorporadas a la capa intermedia de Theano, ya que ha sido un proceso paralelo, donde aparecen puntos de inflexión en forma de grandes mejoras en los resultados.

Se debe conocer que se ha empleado en todo momento una metodología en cuanto a la búsqueda de los mejores modelos. Esta metodología está relacionada en como se debe considerar que un modelo ha estado correctamente entrenado y con el uso de un **conjunto de validación**. Se ha considerado de dos formas:

- Se entrena un modelo hasta la iteración que consiga el **menor error en validación** (se guardan los parámetros entrenados en ese punto). Si el conjunto de datos está bien preparado no debe haber grandes diferencias en el resultado de error entre validación y test.

- Se entrenan los modelos **controlando la cross-entropía** del conjunto de validación, donde el mínimo valor se considera el punto de entrenamiento óptimo. Este punto indica el momento en que se ha empezado a sobre-aprender los datos de entrenamiento y por tanto donde los parámetros consiguen realizar una mayor distinción entre clases.

  Si se conoce esta iteración, se puede volver a entrenar el mismo modelo, añadiendo el conjunto de validación al de entrenamiento, con el mismo número de iteraciones hasta el punto óptimo. Esto funciona, ya que la red se habrá entrenado hasta el mismo coste de entrenamiento, pero con unas cuantas imágenes de más que aportan robustez al modelo, muy necesaria en conjuntos pequeños como el utilizado.

Conocida esta premisa, se describirá brevemente el desarrollo de las pruebas realizadas:

1. Las primeras prueba realizadas se llevaron a cabo con el **conjunto de datos 1** y la **red A** con dos capas ocultas de 1024 neuronas. Estas pruebas demuestran lo que se pensaba desde un principio, y es que el conjunto de datos no es lo suficientemente grande para permitir la creación un modelo robusto y eficiente. Los resultados estaban entorno a $40\,\%$ y $42\,\%$ de error de test.

   Otra conclusión que se saca es que la red sufre, en pocas iteraciones, de un sobre-aprendizaje importante. En este punto se decidió programar la técnica «dropout» y dar los primeros pasos en el campo de la «data augmentation».

2. Una vez implementado «dropout», se realizó el mismo experimento que el anterior pero añadiendo una probabilidad de dropeo para las capas ocultas de $0{,}5$ (por convenio) y un «dropout» escalado para las capas convolucionales, $[0{,}2, 0{,}3, 0{,}4, 0{,}5]$[12] respectivamente. Estos resultados reflejan un error de test mejorado, siendo $36{,}66\,\%$ hasta el punto de entropía. Aunque es un buen resultado en comparación con la meta buscada sigue lejos de la capacidad que poseen la redes convolucionales actuales.

   Si se visualiza el resultado en la tabla de la competición:

Ranking	Nombre Red / Equipo	Error (%)
#1	RBM	$29{,}53$
#2	Unsupervised	$30{,}83$
#3	Maxim Milakov	$31{,}15$
#4	Radu+Marius+Kristi	$32{,}60$
#5	Lor.Voldy	$35{,}09$
#6	**A dropout (conjunto 1)**	$\mathbf{36{,}66}$

Tabla 6: Resultados competición con la media de test público y privado.

   A partir de este punto se desestimó el uso del conjunto original a favor del conjunto 2 con la modificación espejo.

3. Las pruebas que siguieron a este acontecimiento confirmaron la potencia del uso de «Data Augmentation offline» junto con «dropout». El principal resultado se obtiene con la **red A** con una capa oculta de 1024 neuronas, buscando en primer lugar el punto de entropía:

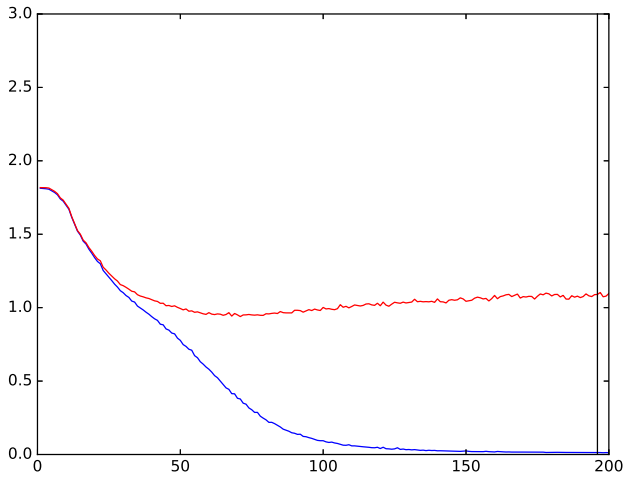

Figura 18: Coste / Cross-Entropía de entrenamiento (línea azul) y validación (línea roja).

[12]Se recuerda que se ha programado pasando al método la probabilidad de drop y no de retención (internamente se aplica la operación 1 - p). Esto es sólo una opción personal.

De la gráfica anterior se aprecia como el conjunto de validación aumenta su coste con el sobre-aprendizaje siendo el punto óptimo el «epoch» 71, con una entropía de entrenamiento de 0.37 y validación de 0.94. Como se ha explicado anteriormente, se entrena a continuación la red sin validación hasta ese mismo número de «epoch».

Para mejorar el resultado ,se aplican unas pocas iteraciones (10) con el factor de aprendizaje inferior en un factor de 10. Esto se conoce como **«learning rate annealing»** y se realiza en todos los experimentos.

El resultado para este experimento es de un $33{,}7\,\%$ de error. No obstante, realizando un test con la suma de las probabilidades a posteriori del test aumentado, el error final es de $32{,}52\,\%$. Se considera este resultado el mejor en cuanto a sencillez de implementación de técnicas. Con respecto a la competición de **Kaggle** estaría situado en 4ª posición:

Ranking	Nombre Red / Equipo	Error (%)
#1	RBM	$29{,}53$
#2	Unsupervised	$30{,}83$
#3	Maxim Milakov	$31{,}15$
#4	**A drop (conjunto 2)**	$\mathbf{32{,}52}$
#5	Radu+Marius+Kristi	$32{,}60$
#6	Lor.Voldy	$35{,}09$
#7	A dropout (conjunto 1)	$36{,}66$

Tabla 7: Resultados competición.

Se obtienen varias conclusiones de estas prueba:

- Utilizar la metodología del aprendizaje, hasta el punto de entropía, funciona realmente bien.
- El test aumentado consigue mejores resultados que el test convencional.
- Siendo el mejor resultado obtenido hasta el momento se requiere de la aplicación de otros métodos para rebajar el error e intentar «ganar la competición».

4. La primera prueba para intentar lograr una mejor clasificación fue aumentar la profundidad de la red. En este punto se usa la **red B** con distintas variaciones del número de neuronas de la capa oculta, además de comprobar la desactivación de la última capa de «pooling». Otro cambio con respecto a A, está relacionado con el «dropout» de las capas convolucionales, ya que se advirtió que un «dropout» grande en las últimas capas repercutía en un empeoramiento del aprendizaje. Así pues el vector de «dropout» óptimo queda de la siguiente forma: $[0{,}2,\ 0{,}3,\ 0{,}4,\ 0{,}4,\ 0{,}2,\ 0{,}2]$.

   En general, estos problemas con el «dropout» nacen de la propia profundidad de la red y del tamaño de las dimensiones de las imágenes (en la capa de reshape). Esto

es precisamente lo que desencadenó la necesidad del testeo de las redes residuales, ya que el principal motivo para usarlas es, precisamente, ir más profundo (ver [8]) sin saturar la precisión en clasificación.

Igualmente, con la máxima optimización posible para B, se consiguió un resultado en error de test de $33{,}27\,\%$ (con dos capas ocultas de 1024 y 512 respectivamente):

Ranking	Nombre Red / Equipo	Error (%)
#1	RBM	$29{,}53$
#2	Unsupervised	$30{,}83$
#3	Maxim Milakov	$31{,}15$
#4	A drop (conjunto 2)	$32{,}52$
#5	Radu+Marius+Kristi	$32{,}60$
#6	**B Drop (conjunto 2)**	$\mathbf{33{,}27}$
#7	Lor.Voldy	$35{,}09$
#8	A dropout (conjunto 1)	$36{,}66$

Tabla 8: Resultados competición.

5. En este momento, se realizó la implementación conjunta de la técnica «batch normalization» y las redes residuales, además de la creación del conjunto 3. Una vez programado se construyeron las redes C y D.

   En primera instancia se programó el batch norm con demasiados parámetros para gamma y beta. Se realizaron experimentos con ello, pero por constantes desbordamientos de memoria de la GPU se decidió rebajarlos a 2 por neurona (se cuenta cada filtro como una neurona).

   Una vez corregido, se testeo la **red A** con la misma configuración del mejor resultado (dos capas ocultas de 1024 y dropout), pero esta vez con batch norm. El resultado, aunque bueno, $32{,}76\,\%$, no superó al anterior error obtenido, pero se consiguió con la mitad de iteraciones. Una de las características de «batch normalization» es que se reduce en gran medida las iteraciones necesarias para que una red converja, sin degradar el entrenamiento.

   Otra prueba, con un resultado parecido $32{,}65\,\%$, se obtuvo reduciendo el número de capas ocultas a dos de 512. Esta es la última prueba que se realizó con el conjunto de datos 2, ya que quedaba demostrado que se había que aumentar el número de datos para mejorar la tasa de error.

Ranking	Nombre Red / Equipo	Error (%)
#1	RBM	$29{,}53$
#2	Unsupervised	$30{,}83$
#3	Maxim Milakov	$31{,}15$
#4	A drop (conjunto 2)	$32{,}52$
#5	Radu+Marius+Kristi	$32{,}60$
#6	**A BN (conjunto 2) (512, 512)**	$\mathbf{32{,}65}$
#7	**A BN (conjunto 2) (1024, 1024)**	$\mathbf{32{,}76}$
#8	B Drop (conjunto 2)	$33{,}27$
#9	Lor.Voldy	$35{,}09$
#10	A dropout (conjunto 1)	$36{,}66$

Tabla 9: Resultados competición.

6. A continuación las pruebas fueron centradas en probar el conjunto de datos 3. Por el número de imágenes de entrenamiento el tiempo aumentó considerablemente. Si en el conjunto 2 se realizaba una iteración completa (epoch) en 1,3 minutos, con este conjunto eran 6 minutos.

   Este conjunto se utilizó con todas las redes (A, B, D) y con batch norm (para acelerar el entrenamiento), no obstante los resultados no fueron los esperados, siendo en el mejor caso de $32,54\,\%$ con la red A (2 capas ocultas de 1024 neuronas a imitación del mejor resultado obtenido).

Ranking	Nombre Red / Equipo	Error (%)
#1	RBM	$29{,}53$
#2	Unsupervised	$30{,}83$
#3	Maxim Milakov	$31{,}15$
#4	A Dropout (conjunto 2)	$32{,}52$
#5	**A BN-Drop (conjunto 3)**	$\mathbf{32{,}54}$
#6	Radu+Marius+Kristi	$32{,}60$
#7	A BN (conjunto 2) (512, 512)	$32{,}65$
#8	A BN (conjunto 2) (1024, 1024)	$32{,}76$
#9	B Drop (conjunto 2)	$33{,}27$
#10	Lor.Voldy	$35{,}09$
#11	A drop(conjunto 1)	$36{,}66$

Tabla 10: Resultados competición.

   La principal conclusión de estos experimentos se basan en que el conjunto creado no ofrece realmente una mejora en cuanto a error final y por tanto se descartó por la cantidad de tiempo que requería con respecto al conjunto 2, no obstante utilizar un conjunto mayor seguía siendo la única mejora que se podía aportar al entrenamiento. Casualmente, en un foro de discusión iniciado por el ganador de la competición y el tercer clasificado comentaban la forma en que aplicaron «data augmentation», de esta forma se creó el conjunto 4.

7. Finalmente, con el conjunto 4 se testaron con las redes A, C y D (se descartó B ya que

no ofrecía nada en comparación con C), y se consiguieron los mejores resultados de toda la experimentación.

En todas las pruebas se ha utliizado el mismo DropOut escalado, «batch normalization», factor de aprendizaje (0.1), metodología de la entropía, test aumentado, «learning rate annealing» de 10 iteraciones y número de neuronas ocultas (1024) excepto en la red C (3048).

En el siguiente apartado se puede visualizar una tabla con los mejores resultados obtenidos.

5.6. Resultados de los experimentos.

La siguiente tabla muestra los mejores resultados obtenidos de toda la experimentación realizada:

Red	Error de test (%)
A	29,92
D	30,52
C	31,5

Tabla 11: Mejores resultados de clasificación.

Ranking	**Nombre Red / Equipo**	**Error (%)**
#1	RBM	$29{,}53$
#2	**A BN-Drop (conjunto 4) (1024)**	$\mathbf{29{,}92}$
#3	**D BN-Drop (conjunto 4) (1024)**	$\mathbf{30{,}52}$
#4	Unsupervised	$30{,}83$
#5	Maxim Milakov	$31{,}15$
#6	**C BN-Drop (conjunto 4) (3096)**	$\mathbf{31{,}50}$
#7	A Dropout (conjunto 2)	$32{,}52$
#8	A BN-Drop (conjunto 3)	$32{,}54$
#9	Radu+Marius+Kristi	$32{,}60$
#10	A BN-Drop (conjunto 2) (512, 512)	$32{,}65$
#11	A BN-Drop (conjunto 2) (1024, 1024)	$32{,}76$
#12	B Drop (conjunto 2)	$33{,}27$
#13	Lor.Voldy	$35{,}09$
#14	A drop(conjunto 1)	$36{,}66$

Tabla 12: Resultados finales de la competición.

La configuración de estas redes está comentada en 7.

De los resultados se puede atisbar que la **red A** (2) es la que mejor ha funcionado de toda la experimentación, quedando (de forma ficticia) en el **segundo puesto de la competición**.

Cabe destacar que la competición la ganaba la red que obtuviera el mejor resultado en el test privado, si se considera de esta forma, la red creada saca mejor resultado en el test privado.

Con **D** se logra igualmente un magnifico resultado, siendo una red ligeramente menos profunda que A, lo que sienta las bases de la potencia de las capas residuales. Este resultado conseguiría igualmente la segunda posición en la competición de Kaggle.

Por último, la **red C** tiene mucho potencial, no obstante por el tamaño de la misma resulta difícil de testear correctamente con la capa intermedia propia creada, ya que, al realizar la suma Theano guarda por duplicado la información en la GPU, lo que forma una especie de bucle de guardado aumentado de forma significativa la memoria almacenada.

Por la forma en que se ha programado, para calcular los parámetros de media y varianza que usa «batch normalization» se debe utilizar un tamaño de batch de entrenamiento lo «suficientemente grande» cosa que desborda la memoria y Theano corta el proceso.

Igualmente, el resultado para C no es malo y conseguiría un 4º puesto en la competición.

6. CONCLUSIONES Y TRABAJO FUTURO.

6.1. Conclusiones.

Se ha dedicado este trabajo a la prueba e investigación de algoritmos de «Deep Learning», en concreto de las redes neuronales y las redes convolucionales, para el reconocimiento de emociones en rostros y el resultado es plenamente satisfactorio. Si bien el conjunto de datos no es el mejor posible, ha servido para el objetivo demostrando la potencia de las técnicas actuales, logrando el objetivo de mejorar el error de clasificación humano.

Entre los resultados finales, se puede comprobar que los mejores se han obtenido con un conjunto al cual se le ha aplicado la técnica **«Data Augmentation»**. Se considera esta técnica fundamental en el reconocimiento de emociones y en concreto para el caso de estudio ya que aporta una gran robustez al modelo que se está creando. La propia variabilidad de las caras humanas ligada a la dificultad de captura de imágenes correctamente centradas, hacen necesario la creación de un sistema invariable a las transformaciones al máximo nivel posible.

Entrando en la parte más relacionada con el «Deep Learning», se observa claramente que las redes convolucionales funcionan de forma sorprendente. Aunque no se tiene una comparación con el mismo sistema en la contraparte de la visión artificial, la propia funcionalidad de las redes convolucionales, ligadas a la «relativa» facilidad de implementación las convierten en una gran alternativa a los sistemas tradicionales.

Tratando el tema de las topologías con las que se ha experimentado, «VGGNet» y «ResidualNet», se entiende rápidamente porqué son el estado del arte en la actualidad, reiterando que con el conjunto de datos utilizado no se puede lograr una mejoría muy exagerada.

Gracias a los experimentos se ha podido comprobar que no se debe reducir en demasía el tamaño de la imagen antes de pasarlo a la capa «fully-connected», ya que no se logra aprender lo suficiente. Aunque en este trabajo se ha visto que las red VGG reducida creada, A, funciona mejor que el resto, se ha de mencionar que la red residual, D, contiene menos parámetros de aprendizaje logrando un resultado similar, lo que la convierte en un gran objetivo de estudio.

De las técnicas desarrolladas se ha visto que cada una tiene su espacio y su uso en el entrenamiento de una red, es decir, cumplen lo que se promete. En concreto, además de «Data Augmentation», tanto «dropout» como «Batch Normalization» han servido correctamente al objetivo por el cuál se introdujeron, reducir el sobreaprendizaje y reducir el número de iteraciones necesarias para entrenar un modelo.

Con respecto a los resultados, es cierto que no se ha conseguido superar a Yichuan Tang, ganador de la competición de Kaggle, ya que este utilizaba unas técnicas de clasificación binaria SVM que, según [14] (un papel escrito por el mismo), parece ser que funcionan en todos los casos mejor que la clasificación por soft-max.

Además de todo esto se ha de recordar que todas las pruebas realizadas se han llevado a cabo con una capa intermedia especialmente programada para este trabajo. Esto, teniendo en cuenta los resultados, proporciona una gran satisfacción personal, ya que al inicio del proyecto no se conocía Theano, nunca se había realizado una programación funcional (o simbólica), no se conocía Python (lo suficiente) y mucho menos sus librerías. En este punto se puede decir que se ha logrado una comprensión profunda del funcionamiento de este toolkit y de python, resultando ser muy útiles en comparación con otros lenguajes históricos más complicados, habiendo creado un pequeño herramienta que entrena redes convolucionales o redes neuronales de forma eficiente y sobretodo muy sencilla.

Se ha de recalcar que el tiempo dedicado a programación no se considera desaprovechado en absoluto. Se entiende que, una de las mejores formas de conocer como funciona realmente, tanto a nivel matemático como informático un sistema de «Inteligencia Artificial», es programarlo, ya que con la aparición de problemas y la búsqueda de soluciones es donde se aprende realmente. Al mismo tiempo, se debe que tener en cuenta que la mayoría de técnicas implementadas, en su mayoría, ofrecen una pequeña parte de interpretación personal de su programación, por lo que, aun funcionando como se esperan, pueden suscitar a error.

Por último en referente a las conclusiones, se debe tener en cuenta que es «casi» imposible realizar una tarea de entrenamiento de redes convolucionales sin el computo con GPU, ya que por ejemplo el tiempo de entrenamiento completo de una red como la A, con el conjunto D se dispara alrededor de 8-9 minutos por «epoch». Si este trabajo se hubiera realizado solamente con CPU no se hubiera podido llegar, en absoluto, a las conclusiones descritas.

6.2. Trabajo Futuro.

Pasando ya en el **trabajo futuro**, el desarrollo de este proyecto ha desencadenado una gran cantidad de ideas que se podrías aplicar en el ámbito del «Deep Learning», además del número de técnicas que existen no testeadas.

Como se ha dicho las **redes residuales** son el estado del arte más inmediato, pero sólo destacan aquellas que se utilizan en proyectos de gran envergadura y no han dado mucho de que hablar para proyectos pequeños como el presente. Se ve una clara línea de investigación en la prueba y comparación del contexto en que una red residual funciona mejor y donde está el punto de separación entre usarlas o no. También decir que la propia idea base de funcionamiento abre todo un espectro de posibilidades, por lo que se intuye un gran desarrollo en este sentido.

Siguiendo con esta temática, hay un gran número grande de técnicas que a priori parecen interesantes, ente las que destacan «Drop-Connect», Max-Out, aprendizaje no-supervisado, etc. Se prevé su prueba en futuros proyectos.

En cuanto a la inclinación más personal, por la gran capacidad de computación de las GPU, se ha pensado en la idea de empezar a desarrollar un «dropout» real, es decir, investigar temas de entrenamiento paralelo con una ampliación de uso de «Data

Augmentation offline».

En cuanto a la programación de la **capa intermedia**, se desea seguir aumentando su funcionalidad para futuros proyectos. Entre los principales cambios que se requieren está la limpieza de código duplicado, así como la incorporación de nueva funcionalidad.

Destacar que, como se ha comentado a lo largo del trabajo, Theano realmente no realiza una gran gestión de la memoria GPU, apareciendo problemas de corte de experimentos y no liberación de memoria. Una de las principales mejoras va relacionada con la inclusión de un sistema de guardado automático de parámetros al disparo de una excepción incontrolada.

Para finalizar, otro punto fuerte que aportaría mucha funcionalidad, es la de utilizar documentos CSV para el almacenamiento automático de toda la información con respecto a los experimentos lanzados (variables, parámetros, etc), de esta forma se aseguraría de tener un registro automático personal de la actividad realizada, con la seguridad de no perder información.

7. BIBLIOGRAFÍA

[1] Goodfellow, I. J., Erhan, D., Carrier, P. L., ... y Bengio, Y. (2013). *«Challenges in Representation Learning: A report on three machine learning contests»*, en *Neural Information Processing*, págs. 117–124, 2013. DOI: 10.1007/978-3-642-42051-1_16.

[2] Glorot, X. y Bengio, Y. (2010). *«Understanding the difficulty of training deep feedforward neural networks»*, en *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, 2010.

[3] LeCun, Y., Bottou, L. y Haffner, P. (1997). *«Gradient-Based Learning Applied to Document Recognition»*, en *Proceedings of the IEEE*, págs. 2278–2324, 1998.

[4] Krizhevsky, A., Sutskever, I. y Hinton, G. (2012). *«ImageNet Classification with Deep Convolutional Neural Networks»*, en *Advances in Neural Information Processing Systems 25*, págs. 1097–1105, 2012.

[5] Zeiler, M.y Fergus, R. (2013). *«Visualizing and Understanding Convolutional Networks»*, en arXiv:1311.2901, 2013.

[6] Szegedy, C., Liu, W. ... y Rabinovich, A. (2014). *«Going Deeper with Convolutions»*, en *arXiv:1409.4842*, 2015.

[7] Simonyan, K. y Zisserman, A. (2014). *«Very Deep Convolutional Networks for Large-Scale Image Recognition»*, en *CoRR*, arXiv:1409.1556, 2014.

[8] Kaiming, Xiangyu, Shaoqing, y Jian (2015). *«Deep Residual Learning for Image Recognition»*, en arXiv preprint arXiv:1512.03385, 2015.

[9] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. y Salakhutdinov, R. (2013). *«Dropout: A Simple Way to Prevent Neural Networks from Overfitting»*, en *Journal of Machine Learning Research 15*, págs. 1929–1958, 2014.

[10] Ioffe, S. y Szegedy, C. (2015). *«Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift»*, en arXiv:1502.03167, 2015.

[11] Raiko, T., Valpola, H., y LeCun, Y. (2012). *«Deep Learning Made Easier by Linear Transformations in Perceptron»*, en *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, págs. 924–932, 2012.

[12] Rifai, S., Glorot, X., Bengio, Y. y Pascal, V. (2011). *«Adding noise to the input of a model trained with a regularized objective»*, en arXiv:1104.3250.

[13] Zur, R., Jiang, Y., Pesce, L. y Drukker, K. (2009). *«Noise injection for training artificial neural networks: A comparison with weight decay and early stopping»*, en Medical

Physics, Vol. 36, No. 10, págs. 4810–4818, 2009. Doi: 10.1118/1.3213517

[14] Tang, Y. (2013). *«Deep Learning using Linear Support Vector Machines»*, en International Conference on Machine Learning, 2013.

[15] Bettadapura, V (2012). *«Face Expression Recognition and Analysis: The State of the Art»*, en arXiv:1203.6722, 2012.

[16] Página web oficial del toolkit Theano.<http://deeplearning.net/software/theano/>

[17] Página web oficial de la librería Keras.<https://keras.io>

[18] Página web oficial de la librería lassagne.<https://github.com/Lasagne/Lasagne>

[19] Canal de Youtube enfocado a Deep Learning. <https://www.youtube.com/user/dvbuntu>

[20] Página web oficial de la API BeyondVerbal.<http://www.beyondverbal.com>

[21] Página web oficial de la API Good Vibrations.<http://www.good-vibrations.nl>

[22] Página web oficial de la API The Tone Analizer. <https://www.ibm.com/watson/developercloud/tone-analyzer.html>

[23] Página web oficial de la API EmoVu. <http://emovu.com/e/>

[24] Página web oficial del API de Microsoft. <https://www.microsoft.com/cognitive-services/en-us/emotion-api>

[25] Página web con información teórica sobre Deep Learning en español. <https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/>

[26] Página web oficial del API de EmoVoice. <https://www.informatik.uni-augsburg.de/lehrstuehle/hcm/projects/tools/emovoice/i>

[27] Página web oficial del API de BiText. <https://www.bitext.com>