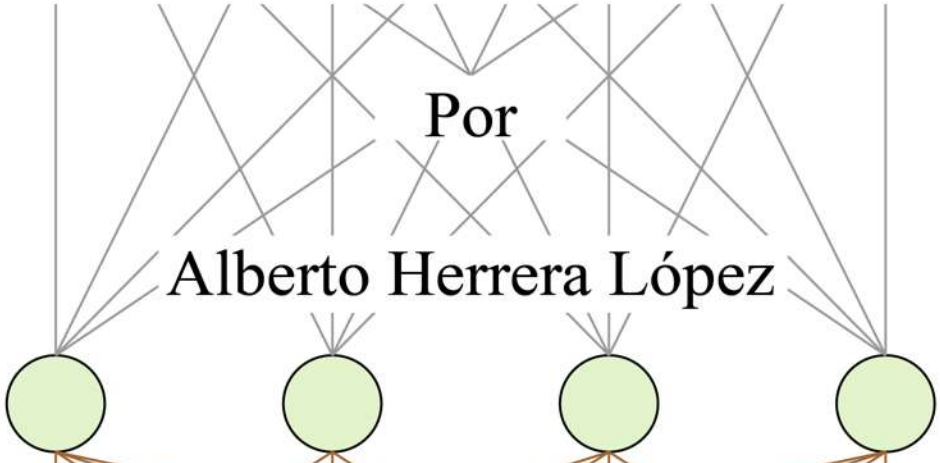
UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

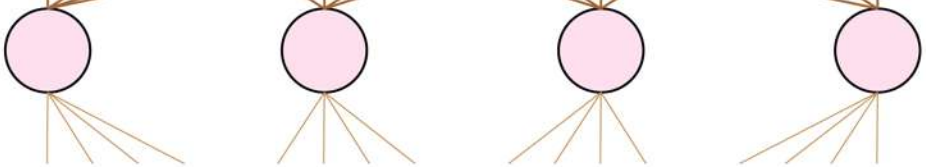
Reconocimiento Automático de Expresiones Faciales usando Aprendizaje Profundo

Por

Alberto Herrera López



Tesis para optar al grado de Lic. en Cs. Físico Matemáticas



Asesora: **Dra. Karina Mariela Figueroa Mora**

Sinodales:
Dr. Francisco Javier Dominguez Mota
Dr. José Antonio González Cervera
Dr. Juan Ahtziri González Lemus

Morelia, Michoacán, México.
Septiembre, 2019

Este trabajo fue financiado por la Universidad Michoacana de San Nicolás de Hidalgo

Agradecimientos

Quiero agradecer y dedicar este trabajo a cada una de las personas que me acompañaron durante la carrera. A mis papás Gabriel Herrera Zavala y Angélica López Nápoles por apoyarme en mi desición de estudiar algo tan poco común como la carrera en Ciencias Físico Matemáticas y proporcionarme todo lo necesario para ser alguien exitoso en la vida. A mi hermano Gabriel Herrera López porque desde que éramos pequeños siempre me ha motivado, orientado, enseñado y apoyado en cada proyecto que me propongo y por estar conmigo aún cuando nos encontrábamos lejos. Siempre estaré agradecido con ustedes y siempre estarán en mi corazón. También quiero agradecer a mi asesora, Karina Mariela Figueroa Mora, por orientarme en muchas ocasiones a lo largo de la carrera y por siempre haberse portado comprensiva demostrando que es una maravillosa persona y una increíble maestra.

Agradezco también a mis amigos quienes en su momento me supieron brindar su apoyo para cumplir este logro.

Índice

Lista de Figuras

Resumen

El reconocimiento de emociones tiene un gran impacto y aplicaciones en estos días, por ejemplo: en marketing, en comportamiento de personas en ciertos contextos, etc. En este trabajo se muestra cómo se puede abordar el problema de reconocimiento de emociones de una persona como un problema de clasificación de imagenes de acuerdo a la expresión mostrada en el rostro. Las emociones se expresan como categorías, éstas fueron 7: *neutral, felicidad, tristeza, enojo, miedo, disgusto y sorpresa*. Para lograr esta clasificación se utiliza un red neuronal convolucional artificial.

En este trabajo se detallan los conceptos básicos de una red neuronal convolucional artificial y se muestra su implementación con una librería conocida llamada *Keras*. Además se explica el preprocesamiento que se realizó sobre las imágenes suminstradas a la red neuronal convolucional propuesta. Por último, como un caso de estudio, se utilizó la red propuesta para evaluar las reacciones emocionales de un grupo de estudiantes en una clase académica.

Palabras clave: Inteligencia artificial, red neuronal, convolución, Keras, dinámica de clase.

Abstract

The emotions recognition has a great impact and applications these days, for example: in marketing, in behavior of people in certain contexts, etc. This paper shows how the problem of recognizing a person's emotions can be addressed as a problem of image classification according to the expression shown on the face. Emotions are expressed as categories, these were 7: *neutral, happiness, sadness, anger, fear, disgust and surprise*. To achieve this classification, an artificial convolutional neural network is used.

This paper details the basic concepts of an artificial convolutional neural network and shows its implementation with a known library called *Keras*. It is also explained the preprocessing that was performed on the images supplied to the proposed convolutional neural network. Finally, as a case study, the proposed network was used to evaluate the emotional reactions of a group of students in an academic class.

Capítulo 1

Introducción

La capacidad de reconocer la emoción de una persona es una tarea vital para la convivencia sana con compañeros de trabajo, en la escuela o dentro de nuestras familias además hay empresas interesadas en saber interpretar las reacciones, ya sea para determinar si un producto que se piensa lanzar al mercado será o no aceptado, o para medir el nivel de felicidad de los empleados al llegar al trabajo. Una forma de lograrlo es observar muchos rasgos de un gran número de personas a lo largo de un intervalo de tiempo (usulmente largo), con lo cual se justifica automatizar este proceso ya que una persona no podría realizarlo en tales magnitudes.

Para lograr interpretar las emociones de una persona hay varios caminos, ya sea la forma en la que escribe, el tono de voz que utiliza, la corporalidad que muestra o la expresión facial que genera ante una situación. Para este trabajo se decidió abordar el tema de reconocimiento de emociones a través de la expresión facial de una persona. Se decidió utilizar 7 categorías, a saber: *neutral, felicidad, tristeza, sorpresa, enojo, miedo y disgusto*. Esta consideración se basó en estudios que demuestran que éstas son las emociones básicas [33].

Para nosotros como seres humanos puede ser fácil interpretar la expresión de una persona viendo su rostro, es decir, si una persona tiene los ojos muy abiertos, cejas levantadas y boca abierta podríamos pensar que esa persona está sorprendida, sin embargo para una computadora la tarea puede ser muy complicada ya que por sí sola no puede interpretarla. El proceso de programar una computadora para interpretar imágenes es parte del área de visión computacional. Para llevar acabo este proceso, en este trabajo se decidió utilizar redes neuronales convolucionales las cuales son un tipo de algoritmo dentro del aprendizaje supervisado de una computadora.

Las redes neuronales convolucionales tienen diversas aplicaciones, algunas de éstas son: como localización y clasificación de objetos dentro de una imagen[1], aprendizaje de estilos artísticos [23], interpretación de escenas en imágenes y videos [35], reconocimiento de voz[6], etc. En nuestro caso, la red propuesta será usada como un clasificador, cabe mencionar que se

decidió utilizar este tipo de red sobre otros algoritmos de clasificación ya que muestran un excelente desempeño cuando la información está distribuída en forma de matriz [1] (las imágenes son una matriz de pixeles).

Este trabajo está organizado de la siguiente manera:

Capítulo 2: Antecedentes. En este capítulo se presenta el estado del arte en torno al reconocimiento de emociones con técnicas de aprendizaje profundo. Se muestran trabajos publicados además de software que ha desarrollado la industria.

Capítulo 3: Conceptos Básicos. En este capítulo se presentan los conceptos básicos acerca de las redes neuronales y el estado del arte entorno al reconocimiento de expresiones faciales. Además se explican los métodos utilizados para tratar las imágenes utilizadas previo a ser suministradas a la red neuronal convolucional propuesta.

Capítulo 4: Metodología En este capítulo se presenta la metodología empleada, la selección de las herramientas que se emplearán, i.e. el lenguaje de programación, el entorno de desarrollo, las librerías, etc., así como el Preprocesamiento de imágenes previo a que las imágenes fueran suministradas a la red neuronal convolucional artificial propuesta.

En este capítulo se presenta además en la sección de Experimentación la fase entrenamiento y validación de la red neuronal convolucional artificial propuesta, así como la justificación de las arquitecturas que llevaron a ésta.

Capítulo 5: Caso de Uso: Emociones Experimentadas por un Grupo de Alumnos en una Clase Académica La red neuronal propuesta fue empleada para analizar las emociones experimentadas por alumnos de una clase, esto con la finalidad de que pueda ser una herramienta para el análisis del desempeño de un profesor.

Capítulo 6: Conclusiones y Trabajo a Futuro. Finalmente, se mostrarán las conclusiones y trabajo a futuro como resultado de este trabajo.

Parte I

ANTECEDENTES Y CONCEPTOS BASICOS

Capítulo 2

Antecedentes

En esta sección mencionaremos el estado del arte respecto al uso de una red neuronal para la detección de emociones. En la literatura existen muchas publicaciones sobre este problema, y por supuesto, la industria ha desarrollado mucho software comercial para este mismo propósito.

2.1 Estado del Arte

En esta sección se presentan algunos trabajos publicados relacionados con el tema del reconocimiento facial usando redes neuronales.

- *Facial Emotion Recognition from Videos Using Deep Convolutional Neural Networks* [51]: Trabajo en el que se detalla el desarrollo de una red neuronal convolucional para la clasificación de expresiones en 10 categorías: *felicidad, tristeza, enojo, sorpresa, disgusto, miedo, neutral, orgullo, contemplación, vergüenza*. Las imágenes utilizadas para la etapa de entrenamiento de dicha red neuronal fueron tomadas de videos de la base de datos *Amsterdam Dynamic Facial Expression Set Bath Intensity Variations* y para la etapa de prueba usaron imágenes obtenidas de la base de datos *Warsar Set of Emotional Facial Expression Pictures*. Reportan una efectividad del 95.12 % de efectividad a la hora de predecir las categorías del conjunto de prueba.

- *Emotion Recognition Using Deep Learning Approach from Audio-Visual Emotional Big Data* [14]: En este artículo se propone un sistema para reconocer emociones utilizando técnicas de aprendizaje profundo en conjunto con máquinas de soporte vectorial (SVM) para crear una máquina de aprendizaje extremo (ELM). Los datos se obtuvieron a partir de Big Data y una base datos llamada eNTERFACE y comprenden audio y video. En ambos casos, para analizar audio y video, fueron utilizadas redes neuronales convolucionales. Este sistema al-

canza una efectividad del 99.9 % en el conjunto de prueba obtenido con Big Data y 86.4 % en el conjunto de prueba de eNTERFACE.

- *Group Emotion Recognition with Individual Facial Emotion CNNs and Global Image based CNNs* [46]: En este artículo se presenta el trabajo realizado para el Emotion Recognition in the Wild Challenge 2017. La meta era clasificar la emoción de un grupo de personas en una imagen en *positiva, neutral* y *negativa*. Para lograr esto utilizaron dos redes neuronales convolucionales, una para detectar la emoción individual de cada persona en la imagen y otra para detectar la emoción global de la imagen. Para entrenar ambas redes neuronales utilizaron imágenes que presentan algún tipo de rotación. Este trabajo ganó dicho concurso con una efectivadad del 80.9 % en el conjunto de validación.

- *A Deep Look into Group Happiness Prediction from Images* [7]: Trabajo realizado para el Emotion Recognition in the Wild Challenge 2016 en reconocimiento de felicidad en grupo. Las herramientas propuestas consistieron en una red neuronal convolucional para encontrar los rostros dentro de la imagen, una red neuronal artificial densa con propagación hacia delante para procesar las imágenes y un modelo creado que utiliza *Long Short-Term Memory (LSTM)* para determinar la intensidad de felicidad. Hicieron una comparación entre el sistema propuesto y el uso de una sola red neuronal convolucional para clasificar los rostros y llegar a un 33 % más de efectividad sobre la red neuronal convolucional.

- *Image based Static Facial Expression Recognition with Multiple Deep Network Learning* [53]: Trabajo realizado en el Emotion Recognition in the Wild Challenge (EmotiW) del 2015. Se utilizó una red neuronal convolucional para clasificar 7 emociones básicas en imágenes, a saber *enojo, disgusto, miedo, felicidad, neutral, tristeza* y *sorpresa*. Proponen el uso de 3 módulos de detección de rostros. Para generar imágenes nuevas a partir de las imágenes dadas fueron generadas perturbaciones por medio de rotaciones de manera aleatoria en las imágenes. La red neuronal propuesta alcanza una efectividad del 61.26 %.

2.2 Aplicaciones Comerciales

En esta sección presentaremos algunas de las principales aplicaciones comerciales. Como se podrá ver, el tema de reconomiento de emociones ha sido muy trabajado en la industria por la relevancia del tema.

- *Amazon Rekognition*[2]:

  Amazon ofrece un API llamada *Amazon Rekognition* dentro de sus productos de AWS. Esta API utiliza técnicas de aprendizaje profundo para identificar objetos, personas, texto, escenas y actividades, e incluso puede detectar contenido inapropiado. También es capaz de

reconocer y analizar el rostro para la verificación de identidad de usuarios, contar la cantidad de personas y utilidades en seguridad pública. Desde el lado del usuario, éste no requiere conocimiento previo en aprendizaje de máquina ya que sólo debe cargar las imágenes o videos a analizar a Amazon S3 (Amazon Simple Storage Service) o videos en straming, y la inteligencia artificial, ya entrenada, dará los resultados solicitados dentro de las siguientes opciones:

- *Amazon Rekognition Video*:
  - Las emociones que es capaz de detectar son *felicidad, tristeza, enojo, confusión, disgusto, sorpresa, calma y desconocido*.
  - Esta API está disponible para usarse en los lenguajes Java, Python, .Net, C++, Go, JavaScript, PHP V3, y Ruby V2.
  - Analiza videos en streaming en tiempo real: *Amazon Rekognition* es capaz de analizar en vivo videos para reconocer y detectar rostros. Para lograr esto se le suministra una transmisión de *Amazon Kinesis Video Stremas* como entrada, así logra realizar reconocimiento facial provistos por el usuario. También puede analizar videos almacenados con anterioridad en *Amazon S3*.
  - Identificación y reconocimiento de personas: *Amazon Rekognition*, por medio de la API *TrackPersons* puede detectar personas y la manera en la que éstas se mueven, ésto lo puede lograr aunque el rostro de la persona no sea visible o salga y entre del marco. Esta API regresa segmentos temporales y clasificaciones de confiabilidad.
  - Reconocimiento de rostros: *Amazon Rekognition Video* da la opción de buscar rostros en tiempo real por medio de comparaciones en conjuntos de rostros suministrados por el usuario. Por medio de la API *CreateCollection*, se crea de manera sencilla un conjunto de rostros de los cuáles se procesan vectores que representen sus características faciales. A partir de esto, *Rekognition* busca en el conjunto antes dado por el usuario para buscar similitudes dentro del video.
  - Análisis facial: *Amazon Rekognition* además de encontrar los rostros de un video, es capaz de analizar sus caraterísticas, algunas de éstas son: si la persona sonrie, si tiene ojos abiertos o algunas emociones. Con la API *DetectFaces, Rekognition Video* regresa ciertos puntos de referencia, a saber: ojo izquierdo, ojo derecho, nariz, comizura izquiera y derecha de la boca.
- *Amazon Rekognition Image*: Además de reconocer rostros y algunas características en ellos, también puede reconocer escenas, objetos, imágenes no seguras y texto en imágenes.
- Administración a través de la API, la consola o la línea de comandos:

    - Tanto para *Amazon Rekognition Image* como para *Amazon Rekognition Video* se puede acceder por medio de la API de *Amazon Rekognition*, la consola de administración AWS y la interfaz de la línea de comandos de AWS (CLI). Todas estas opciones proporcionan la capacidad de usar las API de *Amazon Rekognition*.
  - Los precios para utilizar esta API varían según la localización geográfica la cantidad de imágenes procesadas, en el caso de Amazon Rekognition Image, y según la cantidad de minutos de video, en el caso de *Amazon Rekognition Video*. Siendo gratis el procesamiento de hasta 5000 imágenes al mes, para el caso de *Amazon Rekognition Image* y 1000 minutos de video por mes, para el caso de Amazon Rekognition Video, tras haber realizado la inscripción y durante el primer año. Los costos rondan al rededor de .00001 USD por imagen y .0001 por minuto de video procesados.

- *Cloud Vision* API de Google[8]:

  Por medio de esta API, Google ofrece un servicio de detección de rostros con el cuál, además de detectar los rostros existentes dentro de una imagen, se pueden detectar las emociones de *alegría, tristeza, enojo y sorpresa* con un cierto grado de probabilidad determinado por las etiquetas: “desconocido”, “muy improbable”, “improbable”, “posible", “probable” y “muy probable”.

  Esta API está disponible para los lenguajes: C#, Go, Java, Node.Js, PHP, Python y Ruby.

  Los costos por utilizar esta API varían entre .60USD y 1.50USD por cada 1000 unidades, por uso mensual, y es gratis si las unidades a procesar van de 1 a 1000, al mes.

- *F.A.C.E* API de Sightcorp[41]:

  Por medio de esta API, con la ayuda de aprendizaje-profundo y visión-computacional, Sightcorp trata de brindar un servicio orientado a empresas o personas que ofrezcan algún servicio o producto, dándoles información en tiempo real de las reacciones que van teniendo los posibles clientes, además de información respecto edad, género, hombre o mujer, etnia, si es de procedencia africana, asiática, española o caucásica y colores de vestimenta. En cuanto a las emociones, presentan 6, siendo estas: *tristeza, disgusto, enojo, sorpresa, miedo* y *felicidad*.

  Para acceder a esta API se tiene que hacer una petición HTTP POST a la que se le da la información del video o imagen a procesar y esta API devuelve un archivo en formato JSON con la siguiente información:

  - error_code
  - description
  - img_size
  - Si logra encontrar un rostro:

    - age: Un número entero que tiene como valor mínimo 0
    - gender: Un número entero entre -100 y 100, negativo para hombre, positivo para mujer
    - mood: Un número entero entre 0 y 100 que cuenta que tanta sonrisa tiene el rostro
    - position: Punto en x, punto en y, ancho y alto del rostro localizado
    - rotation: Devuelve 3 pares de entors entre -180 y 180, coincidente con ángulos eulerianos.
    - landmarks: Posición de los ojos y maskpoints.
    - clothingcolors: Devuelve valores hexadecimales.
    - ethnicity: Regresa un valor numérico de entre 0 y 100 para cada una de las etnias correspondientes.
    - emotions: Regresa un valor numérico de entre 0 y 100 para cada una de las emociones correspondientes.

  Para utilizar esta API es necesario comprar una membresía dentro de las siguientes opciones:

  - Prueba: Esta membresía es gratuita, válida por un periodo de 2 semanas. Con esta membresía se pueden hacer 5000 llamadas a la API, en ese periodo, con una velocidad de 1 llamada por segundo.
  - Principiante: Esta membresía tiene un costo de 49 euros mensuales. Con esta membresía se pueden hacer 60,000 llamadas a la API mensualmente, con una velocidad de 5 llamadas por segundo.
  - Profesional: Esta membresía tiene un costo de 199 euros mensuales. Con esta membresía se pueden hacer 300,000 llamadas mensuales, con una velocidad de 10 llamadas por segundo.
  - Empresa: Esta membresía tiene un costo de 499 euros mensuales. Con esta membresía se pueden hacer 1,000,000 llamadas mensuales, con una velocidad de 20 llamadas por segundo.

- iMotions

  - *AFFECTIVA* [17]: Por medio de este software, iMotions ofrece a los usuarios la opción de analizar las emociones de sus posibles compradores o sujetos de prueba de algún producto. Ofrece la siguiente información:
    - Conjunto de sentimiento: Positivo, negativo y neutral.
    - Puntos de referencia faciales: Extrae el punto en coordenadas x, y de 34 puntos característicos.

    - Canal de emociones: *sonrisa* (disfrute), *ceño* (concentración, confusión, disgusto), *levantamiento de cejas* (sorpresa), *esquina de labios* (tristeza).
    - Enganche emocional (Valor numérico).
    - Orientación de la cabeza (Posición en un espacio 3D con ángulos Eulerianos).
    - Distancia interocular (Distancia entre los extremos exteriores de los ojos).
  - Emotient[18]: Software similar a *Affectiva* pero más orientado a la visualización de los datos exportables a Excel, MathLab, SPSS y R, para su posterior análisis.

  Para tener acceso a este software se debe hacer una solicitud a iMotions y pagar según los productos solicitados.

- *Eyeris EmoVu*[11]:

  SDK que consiste en 5 análisis faciales y módulos de reconocimiento:

  - Reconocimeinto de emoción: Este módulo produce una medida continua de intensidad para 7 expresiones universales: *alegría, enojo, tristeza, sorpresa, disgusto, miedo, neutral*. Además calcula 7 emociones que son automáticamente derivadas de las medidas de la intensidad de expresion e información de la posición de la cabeza. Estás medidas incluyen: *atención, aumento de emoción, expresividad*, indicadores positivos y negativos del estado de ánimo.
  - Reconocimiento de género: Este módulo produce una etiqueta de género y la correspondiente medida de fiabilidad
  - Reconocimiento de edad: Este módulo regresa un identificador para cada cara dentro de la imagen. El usuario simplemente agrega imágenes a un conjunto para que el módulo automáticamente genere una base de datos de reconocimiento. El modelo generado puede ser fácilmente aplicado a una nueva imagen para obtener un identificador de edad con la correspondiente fiabilidad de medida.
  - Rastreo facial: Este módulo provee un cuadrado señalando la región, una marca de localización facial e información de la posición de la cabeza para cada cara detectada en la imagen.
  - Medidas de estado de ánimo: Este módulo calcula las medidas del estado de ánimo con base en los resultados del reconocimiento de la emoción.

  Este SDK está disponible en los lenguajes C++ y C#.

- *Insights Develop* de NVISO[29]

  SDK que utiliza redes neuronales convolucionales para detectar el nivel de una cierta emocion. Permite la integracion a teléfonos inteligentes.

- *Kairos*[20]

  SDK que permite agregar reconocimiento facial al software. Las características que tiene son:

  - Detección facial: Encuentra y rastrea caras en cualquier video, foto o imagen.
  - Identificación facial: Busca coincidencias faciales.
  - Verificación facial: Verifica la cara de alguien.
  - Detección de edad: Detecta grupos de edades.
  - Detección de género: Detecta el género de cada cara encontrada, dando como resultado masculino o femenino
  - Detección múltiple de caras: Detecta las caras dentro de grupos de personas.
  - Características faciales: Detecta ojos, cejas, nariz, boca y otras marcas.

- *Face* API de Microsoft[27]

  API que permite detectar una o varias caras humanas en una imagen encuadrándolas, junto con atributos faciales que contienen predicciones de características faciales basadas en aprendizaje automático. Las características de atributos faciales disponibles son: *edad, emoción, sexo, postura, sonrisa y vello facial*, junto con 27 puntos de referencia para cada cara de la imagen. Dentro del reconocimiento de emociones, esta API regresa el grado de confianza de un conjunto de emociones para cada cara de la imagen, como enfado, desprecio, desagrado, miedo, felicidad, neutralidad, tristeza y sorpresa. API disponible en el lenguaje C#. La información la devuelve en formato JSON.

- *FaceReader de Noldus*[28] Software de reconocimiento que analiza 6 expresiones faciales básicas: *neutral, desprecio, aburrimiento, interés* y *confusión*. Además detecta la dirección de la mirada, orientación de la cabeza y características de la persona, modelando 500 puntos clave de la cara.

- *SkyBiometry*[42] API de reconocimiento y detección de rostros basado en la nube que permite detectar emociones en las fotos. Detecta las caras y el estado de ánimo de entre feliz, triste, enojado, sorprendido, disgustado, asustado y neutral. Determina si una persona sonríe o no. Disponible para: C#, Python, Ruby, PHP, Mashape, JavaScript, Java y iOs.

- *CrowdEmotion* API[9] API que utiliza el reconocimiento facial para detectar la serie temporal de seis emociones *felicidad, sorpresa, ira, disgusto, miedo* y *tristeza*.

Capítulo 3

Conceptos Básicos

En este capítulo se presentan los conceptos básicos a utilizar en este trabajo. Se abordan los conceptos de grafo, tensores, el aprendizaje de máquina y su clasificación, el concepto de neurona, la arquitectura de las redes neuronales, su proceso de entrenamiento y, finalmente, algunos conceptos para el preprocesamiento de imágenes.

3.1 Conceptos Básicos Matemáticos

Previo a introducir los conceptos propios de *Aprendizaje Profundo* es necesario mencionar algunos conceptos matemáticos ya que estos serán utilizados posteriormente:

3.1.1 Grafos

Sea V un conjunto finito no vacío, y sea $E \subseteq V \times V$. El par ordenado (V, E) es un grafo donde V es el conjunto de vértices, o nodos, y E es el conjunto de aristas. Se escribe $G = (V, E)$ para denotar al grafo G [13]. Cuando no importa la dirección a la que apuntan las aristas se dice que G es un grafo no dirigido, de otro modo es un grafo dirigido. Se puede representar un grafo de forma gráfica donde se suelen representar los vértices con círculos y las aristas con líneas, como se muestra en la figura 3.1.

Camino Sea x, y vértices de un grafo $G = (V, E)$. Denotaremos un camino por $x - y$, donde x y y son los vértices inicial y final, respectivamente. Un camino en G es una sucesión finita alternada $x = x_0, e_1, x_1, e_2, ..., e_{n-1}, x_{n-1}, e_n, x_n = y$ de vértices y aristas de G, donde $x = x_0$ denota el nodo inicial y $x_n = y$ denota el nodo final, y que contiene las aristas $e_i = \{x_{i-1}, x_i\}$ donde $1 \leq i \leq n$. Si $x = y$ con $n > 1$ entonces es un *camino cerrado*. Si ningún vértice del camino $x - y$ se repite entonces el camino es un camino simple. Dado lo anterior podemos definir un *ciclo* como un camino simple

cerrado $x - x$, es decir, el vértice inicial y final son el mismo. La imagen 3.1 b) muestra un ciclo.

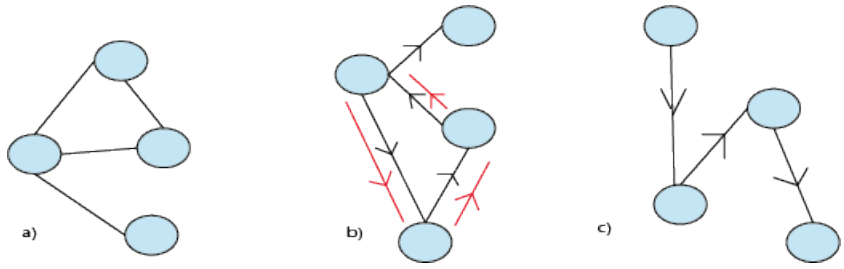


Figura 3.1: Tres grafos con 4 vértices donde los vértices son representados por círculos y las aristas por líneas. a) es un grafo no dirigido, b) y c) son grafos dirigidos, debido a la presencia de las flechas en las aristas. b) presenta un ciclo (marcado en rojo) mientras c) no.

3.1.2 Tensor

Los tensores son los bloques básicos que contienen la información utilizada en los algoritmos de *aprendizaje de máquina*. Se puede pensar un tensor como una generalización de los vectores. Un tensor de orden 0 es un escalar, un tensor de orden 1 es un vector, un tensor de orden 2 es una matriz, un tensor de orden 3 es un ortoedro (prisma rectangular). Podemos notar que un tensor de orden d se construye con tensores de orden $d-1$. La imagen 3.2 muestra una representación gráfica de los tensores de rango 0, 1, 2 y 3.

3.1.3 Regla de la Cadena para Derivadas de Funciones de Varias Variables

Del cálculo diferencial de una variable sabemos que, si tenemos una función compuesta $f(t) = g[r(t)]$ la *Regla de la Cadena para Derivadas de una Variable* sirve para calcular la derivada de $f(t)$ de la siguiente manera [5]:

$$f'(t) = g'[r(t)] \cdot r'(t) \tag{3.1}$$

Sin embargo, esta definición no es suficiente para los fines de este trabajo ya que no está claro cómo manipular las derivadas cuando hay más de una variable. Para el caso en el que tengamos que trabajar con más de una variable, supongamos que f es una función que recibe como argumento n funciones g_i para alguna $i \in \{1, 2, 3, ..., n\}$ y a su vez estas funciones g_i reciben como argumento la misma variable ω, entonces la función f se escribe como $f(g_i(\omega), ..., g_n(\omega))$. Para calcular las derivadas de dicha función f se utiliza la *Regla de la Cadena para Derivadas de Funciones de Varias Variables* que se escribe como [5, 44]:

$$\frac{\partial f(g_1(\omega), ..., g_t(\omega))}{\partial \omega} = \frac{\partial f(g_1(\omega), ..., g_n(\omega))}{\partial g_1(\omega)} \cdot \frac{dg_1(\omega)}{d\omega} + ... + \frac{\partial f(g_1(\omega), ..., g_n(\omega))}{\partial g_n(\omega)} \cdot \frac{dg_n(\omega)}{d\omega} \tag{3.2}$$

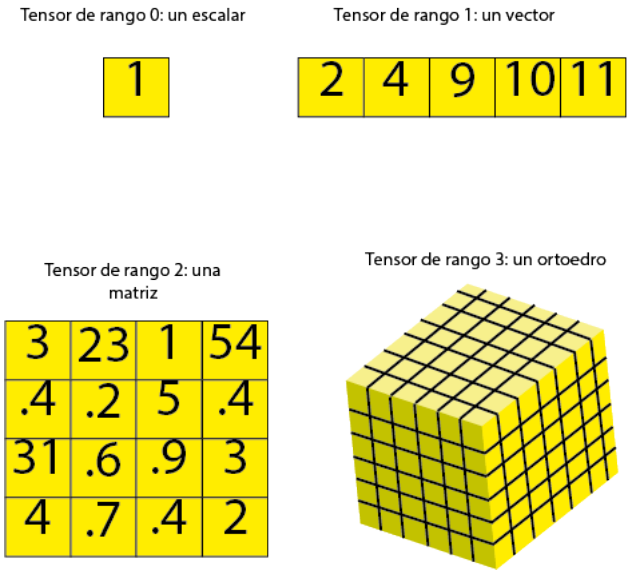


Figura 3.2: Representación gráfica de los tensores. Con esta representación resulta sencillo apreciar que un tensor de orden n está formado por tensores de orden $n-1$. Para un tensor de rango 3 resultaría tentador referirlo como un cubo, sin embargo no todas sus dimensiones son iguales entre sí. Para este trabajo se utilizan tensores de rango 2, que son las imágenes, y tensores de rango 3, que se forman al juntar las imágenes en bloques.

La cual se puede reescribir usando $\sum$:

$$\frac{\partial f(g_1(\omega),...,g_n(\omega))}{\partial \omega} = \sum_{i=1}^{n} \frac{\partial f(g_1(\omega),...,g_n(\omega))}{\partial g_i(\omega)} \cdot \frac{dg_i(\omega)}{d\omega} \tag{3.3}$$

Se utiliza $\frac{\partial}{\partial g_i}$ en lugar de $\frac{d}{dg_i}$ para hacer referencia a que es *derivada parcial*, es decir, la derivada de una función respecto una variable cuando la función tiene más de una variable. La ecuación 3.3 se utilizará en conjunto con *grafos* en la sección 3.4.2.

3.1.4 Regresión

La regresión es un proceso que cae en el campo de la estadística cuya finalidad es encontrar las relaciones entre una o varias variables independientes y una variable dependiente. En otras palabras, la regresión permite un mejor entendimiento acerca de cómo influye cada uno de los valores de las variables independientes sobre el valor de la variable dependiente. La regresión es bastante utilizada, por ejemplo, para construir un modelo que permita hacer predicciones, o proyecciones, en el tiempo. En la Figura 3.3 se muestran 2 ejemplos de regresiones.

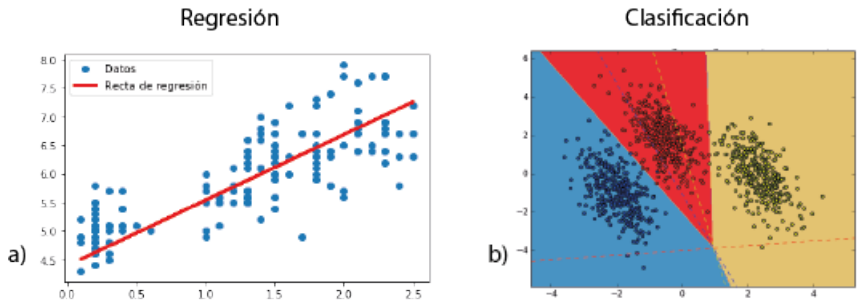


Figura 3.3: a) es un ejemplo de regresión lineal. La regresión es bastante útil para predecir el comportamiento de alguna variable independiente que desconozcamos. b) es un ejemplo de clasificación. La clasificación es útil cuando lo que se desea no es conocer el valor de la variable independiente, sino la categoría a la que pertenece, en este caso se muestran tres categorías, azul, rojo y amarillo.

3.1.5 Clasificación

Se puede entender la clasificación como el problema de asignar a un conjunto de datos una categoría que se haya definido con anterioridad, estas clasificaciones puede ser binarias, es decir, que sólo existan dos categorías, o multivariadas, que existan múltiples categorías. Como ejemplo de clasificación binaria podemos decir si una imagen muestra o no a una persona o si una persona es de una nacionalidad en específico o no lo es. Como ejemplo de clasificación múltiple podemos decir si la expresión de una persona corresponde a felicidad, tristeza, sorpresa, neutral, miedo, enojo o disgusto (como las categorías que se usan en este trabajo).

3.2 Aprendizaje de Máquina (Machine Learning)

3.2.1 Machine learning

Aprendizaje de máquina o *Machine learning* es un subcampo de la inteligencia artificial que contiene algoritmos por medio de los cuáles un programa es capaz de *aprender*. En este contexto, *aprender* significa la capacidad de mejorar el desempeño sobre una tarea por medio de la experiencia [37], esta tarea puede ser catalogar rostros por medio de patrones o lograr que un robot camine por sí solo, identificando la fuerza que debe haber en cada pierna.

3.2.2 Tipos de aprendizaje

De acuerdo a la metodología empleada para el procesamiento de la información, los algoritmos de aprendizaje pueden ser clasificados en:

Aprendizaje Supervisado

Un algoritmo de aprendizaje supervisado recibe un conjunto de datos etiquetados (la etiqueta representa una identificación o clase a la que pertence cada dato) y los procesa para establecer una estructura u orden que pueda consultar después. El *aprendizaje* consiste en un modelo de regresión o clasificación (estos conceptos son explicados en la sección 3.1). Dicho modelo es utilizado como base para predecir la clasificación de datos nuevos, sin etiquetas [40][45].

Aprendizaje no Supervisado

Consiste en determinar patrones de entradas de los datos recibidos [37], usualmente consiste en clasificar los datos agrupando los que tienen elementos en común [45]. En el aprendizaje no supervisado, a diferencia del aprendizaje supervisado, se usan datos sin etiquetar y no existe un conjunto de entrenamiento.

• Se puede decir que los algoritmos de aprendizaje supervisado crean un modelo predictivo, mientras que los algoritmos de aprendizaje no supervisado crean un modelo descriptivo [19].

Aprendizaje por Refuerzo

En este tipo de aprendizaje se tiene como objetivo lograr que un agente autónomo (entidad capaz de percibir su entorno, procesar la información que recibe de éste y ejecutar alguna acción) aprenda un comportamiento exitoso a través de la interacción con su entorno [3], como un programa capaz de hacer recomendaciones de productos con base en las páginas que visite una persona a lo largo del tiempo.

3.3 Aprendizaje Profundo (Deep Learning)

El Aprendizaje Profundo se puede definir como un conjunto de algoritmos que pertenecen a la clase de aprendizaje automático cuya meta es representar abstracciones de datos usando arquitecturas formadas por transformaciones no lineales múltiples [52], a estas arquitecturas se les llaman *redes neuronales artificiales*.

Como definición formal [10] se definen dichas arquitecturas de la siguiente manera: sea $S = ((x_1, c_1), ..., (x_m, c_m))$ un conjunto de datos etiquetados, un método de aprendizaje profundo es una familia de funciones no lineales $\mathcal{F}$, tal que:

$$\mathcal{F}(S) = \phi_n(\phi_{n-1}(\phi_{n-2}(...(\phi_0(S)...)))) \tag{3.4}$$

donde $\phi_t : C_t \to C_{t+1}$ y C_t es la capa t-ésima. Es decir, la función $\mathcal{F}$ es una composición de funciones. Utilizamos los puntos suspensivos (...) para denotar que no conocemos el número de funciones que forman la composición, razón por la cuál desconocemos también el número de paréntesis que abren y cierran. Es importante tomar en cuenta esta definición formal ya que es la que permite un mejor entendimiento de cómo se da el aprendizaje en este tipo de arquitecturas gracias al algoritmo de *Backpropagation*, el cuál se explica en la sección 3.4.2.

Neurona

Podemos pensar una neurona como un nodo en un grafo dirigido distribuido por capas. Las neuronas son la unidad básica de una red neuronal y en éstas se realizan operaciones que permiten procesar los datos que reciben. Si d es el número de datos de entrada, entonces $\overline{X} = [x_1, ..., x_d]$ es el conjunto de datos y $\overline{W} = [\omega_1, ..., \omega_d]$ el conjunto de pesos asociado a cada dato recibido, así la operación que realiza la neurona se define como $\overline{W} \cdot \overline{X} = \sum_{i=1}^{d} \omega_i x_i$, conocida como *producto punto* entre vectores. Una vez calculado $\overline{W} \cdot \overline{X}$ se pasa como argumento a una función ϕ. Esta función ϕ se conoce como *función de activación*, de la cual se hablará a continuación. La Figura 3.4 muestra la representación de una neurona.

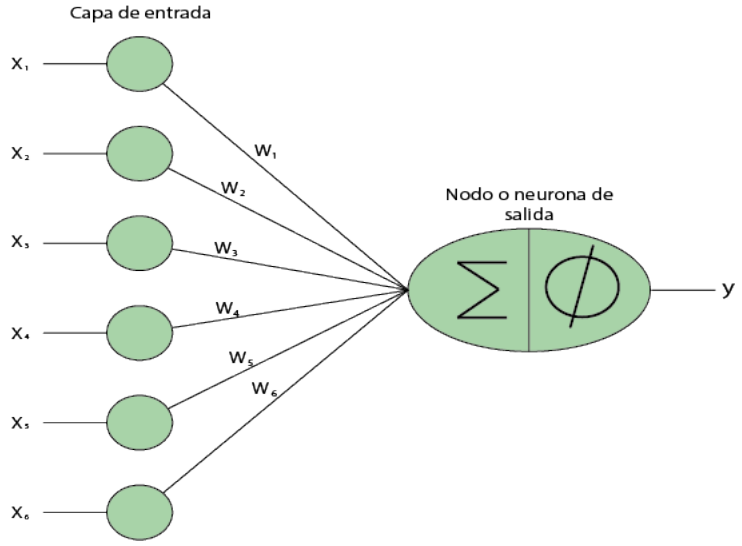


Figura 3.4: Arquitectura básica de una neurona donde $x_1, x_2, ..., x_6$ son los datos de entrada, $\omega_1, \omega_2, ..., \omega_6$ son los pesos correspondientes a cada dato, $\sum$ es el producto punto $\sum_{i=0}^{i=6} \omega_i x_i$, ϕ es la función de activación (ver 3.3) y $y = \phi(\sum)$ es el valor de salida .

Función de Activación

Una función de activación ϕ define el valor de salida de una neurona según los valores obtenidos al efectuar la operación $\overline{W} \cdot \overline{X}$ [1]. Las siguientes funciones son ejemplos de funciones de activación en las que $v = \overline{W} \cdot \overline{X}$

- Función identidad: $\phi(v) = v$
- Función signo: $\phi(v) = \text{sign}(v)$
- Función sigmoide: $\phi(v) = \frac{1}{1+e^{-v}}$
- Función tanh: $\phi(v) = \frac{e^{2v}-1}{e^{2v}+1}$
- ReLU (Rectified Linear Unit, Unidad Lineal Rectificada, en español): $\phi(v) = \max(v,0)$
- Función Hard Tanh: $\phi(v) = \max(\min[v,1],-1)$
- Función softmax: $\phi(\bar{v})_i = \frac{e^{v_i}}{\sum_{j=1}^{k} e^{v_j}}$

Función de Pérdida

Esta función es de vital importancia ya que es la función que se quiere minimizar, con el algoritmo *Backpropagation* (ver sección 3.4.2), para aumentar la tasa de aprendizaje de la red neuronal artificial. La elección de la función de pérdida es crítica ya que define la sensibilidad con la que se tratarán las salidas de la red neuronal. De ahora en adelante esta función se representará por L. A continuación se presentan dos de las funciones de pérdida más comunes:

- Regresión Logística o Clasificación Binaria: Aunque en su nombre esté la palabra 'regresión', esta función se utiliza para clasificar. En este caso, asumimos que el valor observado y es tomado del conjunto $\{-1, 1\}$, es decir que existen dos categorías que etiquetamos como -1 y 1, y la predicción $\hat{y}$ es un valor numérico arbitrario obtenido por el procesamiento de los datos. En tal caso, la función de pérdida para una sola instancia con un valor observado y y una predicción $\hat{y}$ se define cómo:

$$L = ln(1 + e^{-y \cdot \hat{y}}) \tag{3.5}$$

- Entropía Cruzada (Cross-entropy): Si $\hat{y}_1 \dots \hat{y}_k$ son las probabilidades de las k clases (o categorías), entonces la función de pérdida para una sola instancia está definida por:

$$L = -ln(\hat{y_r}) \tag{3.6}$$

  para la r-ésima clase. Este tipo de función implementa regresión logística multinomial.

Matriz de Pesos

La matriz de pesos es escencial en la arquitectura de una red neuronal ya que ésta contiene la información del peso asociado a cada una de las aristas que unen la capa C_t con la capa C_{t+1}. Dichos pesos se representan por $\overline{W_{(t,t+1)}}$. Se puede pensar en el peso de una arista como la importancia que tiene la neurona que conecta esa arista.

Capa Computacional Única: El Perceptrón

La red neuronal más sencilla se conoce como *Perceptrón*. Esta red neuronal contiene un única capa de entrada y una neurona de salida. En este caso la matriz de pesos se convierte en un tensor de rango 1, es decir un vector. El *Perceptrón* integra todos los conceptos vistos hasta ahora. Recibe un tensor que contiene la información de entrada, a estos datos se les aplica la operación *producto punto* mencionada anteriormente, y el resultado es procesado por la función de activación para crear una salida. Para permitir el aprendizaje en el *Perceptrón*, que viene dado por actualizar los pesos para crear un modelo eficiente, es decir crear un modelo capaz de catalogar correctamente los datos de entrada, se utiliza la siguiente función:

$$\overline{W} \leftarrow \overline{W}(1 - \alpha\lambda) + \alpha(y_i - \hat{y}_i)\overline{X_i} \tag{3.7}$$

Históricamente [1] el *Perceptrón* es de los pocos modelos de aprendizaje de máquina en los que la función de pérdida vino después del algoritmo que permite actualizar los pesos. Dicha función de pérdida se conoce como *criterio del perceptrón* y está definida como:

$$L_i = \text{máx}\{0, -y_i(\overline{W} \cdot \overline{X_i})\} \tag{3.8}$$

En la Figura 3.6 se muestra un ejemplo de matriz de pesos.

3.3.1 Arquitectura Básica de las Redes Neuronales Artificiales

Una red neuronal artificial es una arquitectura que toma el concepto del *Perceptrón* y lo extiende de una manera general. A diferencia del *Perceptrón* una red neuronal artificial se puede entender como un grafo con varias neuronas y donde cada arista dirigida conecta la salida de una neurona a la entrada de otra. La idea detrás de una red neuronal es que varias neuronas pueden ser unidas por conexiones entre sí para poder procesar cálculos complejos. Una red neuronal artificial está formada por capas, cada capa es un conjunto de neuronas. Las capas pueden ser descritas como la unión de conjuntos disjuntos $C = \cup_{t=0}^{t=T} C_t$, donde C_t es la t-ésima capa de la red. Cada arista de E, donde E es el conjunto de todas las aristas del grafo que forma la red neuronal, conecta alguna

neurona de la capa C_t con la capa C_{t+1} para alguna $t \in \{T\}$ donde T es el número de capas de la red neuronal, excluyendo C_0. A T también se le conoce como profundidad de la red [39]. Las capas $C_1, \ldots, C_{T-1}$ se llaman capas ocultas. Mientras que la capa C_0 se llama capa de entrada y la capa final C_T se llama capa de salida. La Figura 3.5 muestra una red neuronal con un esquema.

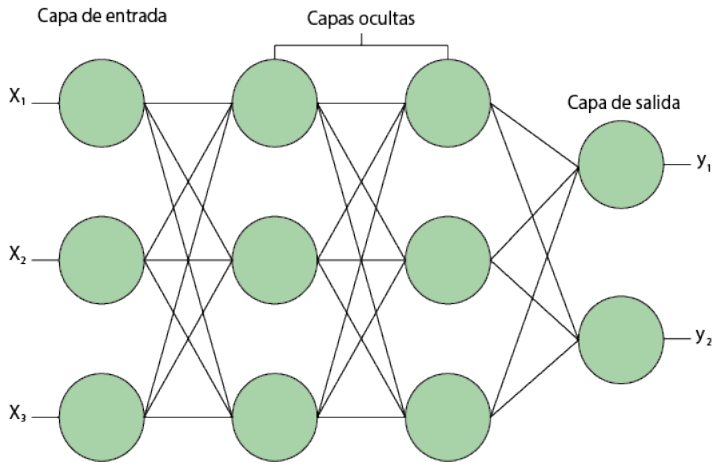


Figura 3.5: Arquitectura básica de una red neuronal con alimentación hacia delante. En esta red $|C| = 4$ y máx $|C_t| = 3$.

Como se mencionó anteriormente los valores de la capa de entrada son proporcionados por los datos sobre los cuales se quiere entrenar la red neuronal o sobre los que se desea utilizar un modelo creado previamente por una red reuronal ya entrenada (esto es posible debido a la naturaleza de un algoritmo de aprendizaje de máquina y se hablará sobre esto en la sección 3.4), y los valores de las neuronas en capas posteriores se obtienen al realizar operaciones entre los pesos y los valores de las neuronas en capas anteriores. Cabe mencionar que el proceso descrito no proporciona información sobre los valores iniciales que tienen los pesos, por lo que se inicializan con valores aleatorios. Esos valores iniciales serán posteriormente modificados de forma dinámica [1] por el algoritmo *Backpropagation* (ver página 24, sección 3.4.2).

La figura 3.6 muestra un ejemplo de una matriz de pesos.

Red Neuronal con Alimentación hacía Delante (Feedforward propagation neural network)

La característica principal de esta red es que cada neurona de la capa C_t está conectada con las neuronas de la siguiente capa C_{t+1}, creando un grafo dirigido acíclico.

El tamaño de este tipo de red es $|C|$, teniendo como ancho máx $|C_t|$ [39]. Además se agrega una neurona extra a cada capa llamada *neurona de sesgo b*, que proporciona un valor constante de 1 [1]. La figura 3.5 muestra un ejemplo de la arquitectura básica de una red neuronal con alimentación hacia delante.

	c_1^2	c_2^2	c_3^2	c_4^2	c_5^2	c_6^2
c_1^1	.65	.60	.34	.03	.2	.89
c_2^1	.65	.01	.61	.23	.34	.22
c_3^1	.5	.35	.55	.16	.78	.63
c_4^1	.08	.98	.9	.12	.09	.26
c_5^1	.56	.95	.65	.68	.53	.67
c_6^1	.04	.54	.78	.48	.32	.5

Figura 3.6: Ejemplo de matriz de pesos. Esta matriz contiene los pesos asociados a las neuronas de la primer capa conectadas a las neuronas de la segunda capa. El subíndice de c representa el número de neurona y el súper índice representa la capa. Por ejemplo, el peso que hay entre la neurona c_2^1 y la neurona c_4^2 es .23.

3.3.2 Red Neuronal Convolucional (Convolutional Neural Network CNN)

Las redes neuronales convolucionales son diseñadas especialmente para trabajar con datos distribuidos en forma de matriz en donde la posición de los datos es sumamente importante. El ejemplo más sencillo de ésto son las imágenes, donde los datos están contenidos en los pixeles que la componen. Debido a ésto la gran mayoría de aplicaciones de las redes neuronales convolucionales está enfocada al tratamiento de imágenes. Este tipo de redes también puede ser utilizado para procesar datos ligados al tiempo.

La primera motivación para desarrollar este tipo de redes neuronales se deriva de los experimentos de Hubel y Wiesel sobre la corteza visual de los gatos[15]. La corteza visual tiene pequeñas regiones de células que son sensibles a regiones específicas del campo visual, es decir, si ciertas regiones del campo visual son activadas entonces ciertas células específicas también serán activadas. Además las células activadas tambien dependen de la forma y de la orientación de los objetos en el campo visual. Por ejemplo, bordes verticales activan ciertas células, mientras que bordes horizontales activan otras. Estas células están conectadas por medio de una arquitectura de capas. Fue ésto lo que permitió descubrir que los mamíferos usamos diferentes capas para construir partes de las imágenes en diferentes niveles de abstracción.

Un factor que ha jugado un papel importante para que aumente la investigación y desarrollo de las redes neuronales convolucionales ha sido la competencia anual *ImageNet* [16] en la que las redes neuronales convolucionales han sido constantemente ganadoras desde el 2012, siendo la ganadora la red *AlexNet*[22] en dicho año.

Una red neuronal convolucional es similiar a una red con propagación hacia delante tradicional con la excepción de que las operaciones en las capas se realizan sobre cada elemento de la matriz que las compone. Tipicamente una red neuronal convolucional está formada por capas de convolución (ver página 21), capas pooling (ver página 23) y capas conectadas como en una red neuronal tradicional. A continuación se explica cómo funcionan las capas de convolución y pooling.

Capa de Convolución

Las capas de convolución son las que le dan el nombre a este tipo de red debido a las operaciones que en ella se efectúan. A continuación se describen las características que tiene una capa de convolución:

- Entrada: Los datos de entrada están distribuidos en forma de matrices y sea L el ancho y B el alto de dichas matrices. Típicamente estas matrices representan imágenes, por lo que es necesario tomar en cuenta el número de canales que tiene, por ejemplo la profundidad sería 1 si la imagen está en escala de grises o sería 3 si la imagen está en formato de 3 colores, al cuál nos referiremos como profundidad d. Por lo tanto los datos de entrada para la capa de convolución i son de tamaño $L_i \times B_i \times d_i$. En este trabajo la profundidad de los datos de entrada de la primer capa será 1 ya que corresponde con imágenes en escala de grises, sin embargo, en el desarrollo de este tema se seguirá usando d para mantener generalidad. Para $q > 1$ a los datos que se encuentran en las capas ocultas se les llama *mapas de características* o *mapas de activación*.

- En las redes neuronales convolucionales los parámetros están organizados en estructuras matriciales llamadas *filtros* o *kernels* de dimensiones $Fl_q \times Fb_q \times Fd_q$ para la capa q donde *Fl, Fb, Fd* son el largo, ancho y profundidad respectivamente.

- Convolución: Esta operación coloca un filtro en todas las posibles posiciones sobre la matriz de entrada a la capa q para realizar un producto punto en los $Fl_q \times Fb_q \times Fd_q$ parámetros del filtro y la región sobre la que está colocado dicho filtro. Así el filtro p en la capa q tiene los parámetros denotados por el tensor tridimensional $W^{p,q} = [\omega_{ijk}^{(p,q)}]$, donde los índices i, j, k indican las posiciones a lo largo de lo alto, lo ancho y lo profundo del filtro. Los mapas de características de la *q-ésima* capa se representan por el tensor tridimensional $C^{(q)} = [c_{ijk}^{(q)}]$.

Entonces la convolución de la capa q a la capa $q+1$ se defide de la siguiente manera:

$$c_{ijp}^{(q+1)} = \sum_{r=1}^{Fl_q} \sum_{s=1}^{Fb_q} \sum_{k=1}^{Fd_q} \omega_{rsk}^{(p,q)} c_{i+r-1,j+s-1,k}^{(q)} \tag{3.9}$$

La figura 3.7 muestra un ejemplo del proceso de convolución sobre una capa de entrada.

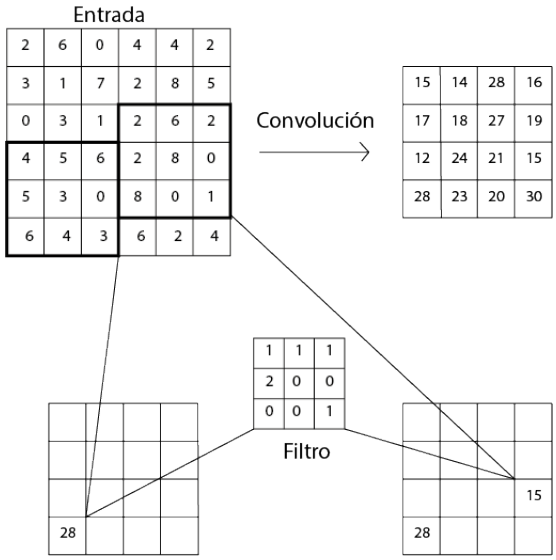


Figura 3.7: Convolución sobre una capa de $6 \times 6 \times 1$ con un filtro de 3×3. En la matriz de entrada, el cuadro resaltado en la esquina inferior izquierda al ser convolucionado con el filtro mostrado, tiene como resultado el número 28. El 15 corresponde al segundo cuadro resaltado.

Pooling

La función Pooling es una función bastante útil para procesar imágenes en una red convolucional ya que reduce el tamaño de la imágen mostrando sus caraterísticas principales. Esta función se agrega como una o más capas en la red neuronal convolucional. Las operaciones que realizan este tipo de capas dependen de una región de $m \times n$ y se construye una nueva imagen con los valores que resultan de esas operaciones, dicha imagen es más pequeña que la que recibió como parámetro. Formalmente, sea L el alto y sea B el ancho de la matriz de entrada y sea m el alto y n el ancho de la región sobre la que se aplicará la función Pooling, entonces, la matriz de salida será de $(L-m+1) \times (B-n+1)$.

Existen varios tipos de Pooling, mencionaremos 3:

- MaxPooling: La función MaxPooling calcula el valor máximo de la región sobre la que se coloca el filtro. La figura 3.8 muestra este proceso.

- AveragePooling: Esta función calcula el valor promedio de la región sobre la que se coloca el filtro.
- GlobalPooling: A diferencia de las operaciones anteriores, *GlobalPooling* no trabaja sobre varias regiones de tamaño fijo, en cambio, esta función devuelve el valor máximo o el valor promedio de toda la capa.

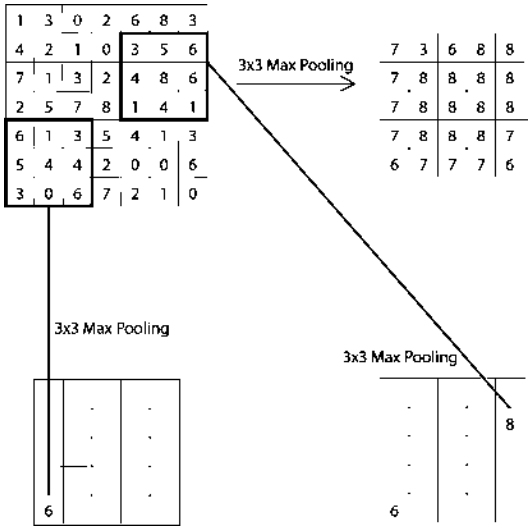


Figura 3.8: Operación MaxPooling sobre una región de 3×3. Esta operación consiste en tomar el valor máximo de la matriz original que está dentro de la región deseada, desplazar dicha región y crear una matriz nueva con los valores obtenidos. En este caso la operación toma una matriz de 7×7 y genera una matriz de 5×5.

3.4 Entrenamiento de una Red Neuronal

Una red neuronal artificial, al ser una técnica de aprendizaje supervizado [37] debe ser entrenada. Para llevar acabo este entrenamiento se requieren 2 conjuntos de datos y 2 conjuntos de etiquetas. El primer conjunto de datos, llamados *datos de entrenamiento*, corresponte a los datos de los cuales la red neuronal aprenderá los patrones necesarios para identificar nuevos datos. El segundo conjunto de datos, llamados *datos de prueba*, está oculto de la red neuronal y lo utilizará para verificar la efectividad que tiene al identificar datos nuevos, es decir, el porcentaje de aciertos sobre el conjunto de validación. Los dos conjuntos de etiquetas contienen la información respecto a cuál categoría pertenecen cada uno de los datos de entrenamiento y prueba. Además, son necesarios los algoritmos de *Forwardpropagation* y *Backpropagation*, que se presentan a continuación.

3.4.1 Forwardpropagation

El algoritmo de *Forwardpropagation* se encarga de transmitir la información de las neuronas en la capa C_r a las neuronas de la capa C_{t+1} de tal modo que si c^{t+1} es una neurona en la capa C_{t+1} entonces $c^{t+1} = \phi(a_{c_t})$ donde a_{c_t} es el valor de $\sum_{i=1}^{d} \omega_i x_i + b$ y ϕ es la función de activación.

3.4.2 Backpropagation

El algoritmo de *Backpropagation* es el responsable de lograr que la red neuronal aprenda por medio de variaciones realizadas a los pesos de las conexiones entre neuronas. Esto se consigue gracias a la optimización de las derivadas parciales de la función de pérdida E, como se muestra a continuación:

Se calculan las derivadas parciales de la función de pérdida respecto varios pesos. El primer paso es calcular la derivada de la fución de pérdida respecto la salida $\frac{\partial L}{\partial o}$. Si la red neuronal tiene múltiples salidas, entonces esta derivada se calcula respecto cada una de ellas. Esto inicializa el cálculo de las derivadas parciales. Posteriormente las derivadas parciales son propagadas hacia atrás según la regla de la cadena definida por la ecuación 3.3.

Ahora consideremos un camino formado por neuronas que estén dentro de las capas ocultas $C_1, C_2, ..., C_{T-1}$ que termina en la salida o.

Ya que el número de caminos entre alguna neurona y la salida puede crecer exponencialmente se utiliza el siguiente lema para calcular las derivadas parciales:

Teorema 1 (Lema de suma de caminos) *Consideremos un grafo acíclico dirigido en el que el i-ésimo nodo contiene la variable c_i. La derivada local $z(i, j)$ de la arista dirigida $e(i, j)$ en el grafo se define como $z(i, j) = \frac{\partial c_j}{\partial c_i}$. Sea $\mathcal{P}$ un conjunto no vacío de caminos que existen entre la variable ω en el grafo y el nodo de salida que contiene la variable o. Entonces el valor de $\frac{\partial o}{\partial \omega}$ está dada por el cálculo del producto de los gradientes locales a lo largo de cada camino en $\mathcal{P}$ sumados todos entre sí:*

$$\frac{\partial o}{\partial \omega} = \sum_{P \in \mathcal{P}} \prod_{(i,j) \in \mathcal{P}} z(i, j)$$

Entonces, si $\mathcal{P}$ es un conjunto de caminos entre la neurona c_r y la salida o, la derivada respecto la función de pérdida L es:

$$\frac{\partial L}{\partial \omega_{(c_{t-1}, c_t)}} = \frac{\partial L}{\partial o} \cdot \left(\sum_{[c_t, c_{t+1}, ..., c_{T-1}, o] \in \mathcal{P}} \frac{\partial o}{\partial c_{T-1}} \prod_{i=t}^{T-2} \frac{\partial c_{i+1}}{\partial c_i} \right) \frac{\partial c_t}{\partial \omega_{(c_{t-1}, c_t)}} \tag{3.10}$$

Nos referiremos a $\frac{\partial L}{\partial o} \cdot \left(\sum_{[c_t, c_{t+1}, ..., c_{T-1}, o] \in \mathcal{P}} \frac{\partial o}{\partial c_{T-1}} \prod_{i=t}^{T-2} \frac{\partial c_{i+1}}{\partial c_i} \right)$ como: $\Delta(c_T, o) = \frac{\partial L}{\partial c_t}$. La idea

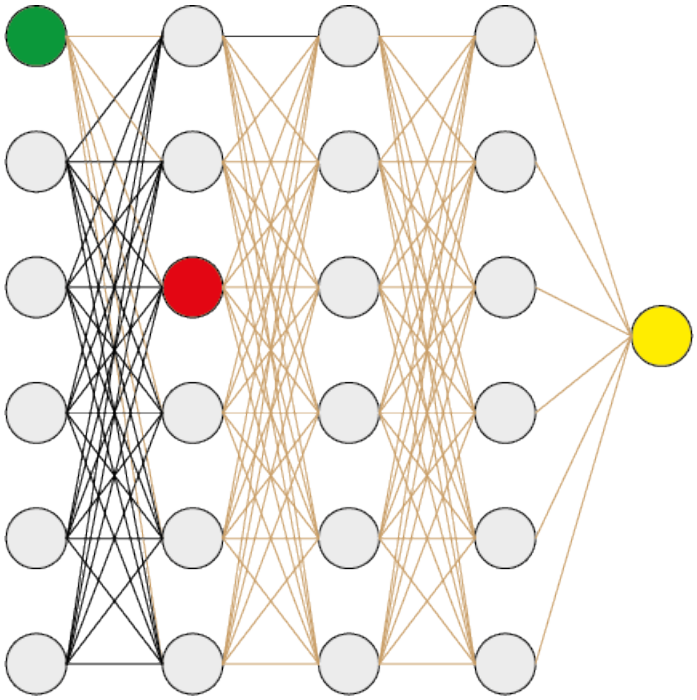


Figura 3.9: Ejemplo de red neuronal. Vemos que el número de caminos (líneas café claro) que tienen al vértice rojo como inicio y al amarillo como final es 216 mientras que para el vértice verde hay 1296 caminos, es decir, el número de caminos en una red neuronal podría crecer de forma exponencial.

es primero calcular $\Delta(c_{T-1})$ para las neuronas de la capa c_{T-1} para después calcular recursivamente los valores para las neuronas en las capas anteriores en términos de los valores de las capas posteriores.

El valor de $\Delta(o,o) = \frac{\partial L}{\partial o}$ es calculado como punto de partida para la recursión para posteriormente propagarse hacia atrás y actualizar los pesos de forma dinámica.

La regla de la cadena genera la recursión para $\Delta(c_t, o)$, siendo esta:

$$\Delta(c_t,o) = \frac{\partial L}{\partial c_t} = \sum_{c_{t+1}:c_t \to c_{t+1}} \frac{\partial L}{\partial c_{t+1}} \frac{\partial c_{t+1}}{\partial c_t} = \sum_{c_{t+1}:c_t \to c_{t+1}} \frac{\partial c_{t+1}}{\partial c_t} \Delta(c_{t+1},o) \tag{3.11}$$

Sea a_c el valor calculado en la neurona c_{t+1} antes de aplicar la función de activación ϕ, es decir $c_{t+1} = \phi(a_c)$ donde a_{t+1} es el producto punto calculado con las entradas a la neurona c_{t+1}.

Para poder calcular 3.11 aún necesitamos calcular $\frac{\partial c_{t+1}}{\partial c_t}$. Para lograr esto usamos la

regla de la cadena y se obtiene:

$$\frac{\partial c_{t+1}}{\partial c_t} = \frac{\partial c_{t+1}}{\partial a_c}\frac{\partial a_c}{\partial c_t} = \frac{\partial \phi(a_c)}{\partial a_c}\omega_{c_t,c_{t+1}} = \phi'(a_c)\omega_{c_t,c_{t+1}} \tag{3.12}$$

El valor obtenido en 3.12 se utiliza en 3.11 para obtener:

$$\Delta(h_t, o) = \sum_{c_{t+1}:c_t \to c_{t+1}} \phi'(a_c)\omega_{(c_t,c_{t+1})}\Delta(c, o) \tag{3.13}$$

Esta recursión es repetida hacia atrás comenzando por el nodo de salida o.

En resumen, el algoritmo de *Backpropagation* es:

1. Utilizar los valores generados por el algoritmo de *Forwardpropagation*.
2. Inicializar $\Delta(o, o)$ con $\frac{\partial L}{\partial o}$
3. Usar la recursión de la ecuación 3.13 para calcular cada $\Delta(c_t, o)$ en la dirección hacia atrás. Después de realizar dichos cálculos, calcular los gradientes con respecto los pesos incidentes a cada neurona de la siguiente forma:

$$\frac{\partial L}{\partial \omega_{c_{t-1},c_t}} = \Delta(c_t, o) \cdot c_{t-1} \cdot \phi'(a_c) \tag{3.14}$$

Ajuste de pesos de la red neuronal

Una vez obtenidas las derivadas parciales, como se mostró anteriormente, se utilizan para ajustar los pesos de toda la red neuronal. Para realizar esto hay varios métodos que incluyen *descenso del gradiente*, *AdaGrad* o *RMSProp*.

En lo siguiente α se conoce como *tasa de aprendizaje* y es un número real entre 0 y 1 sin incluirlos.

- *Descenso del gradiente*: Éste es el método más intuitivo para ajustar los pesos y está definido de la siguiente manera:

$$\overline{W} \leftarrow \overline{W} - \alpha\frac{\partial L_i}{\partial \overline{W}} \tag{3.15}$$

- *AdaGrad*: En este algoritmo se hace seguimiento del cuadrado de la magnitud de la derivada parcial. Entonces, para cada época (iteraciones sobre el conjunto total de datos) se hace lo siguiente:

1. Sea A_i el *valor agregado* de la neurona c_i para cualquier i. Se inicializa A_i con el valor de c_i y se actualiza de la siguiente manera:

$$A_i \leftarrow A_i + \left(\frac{\partial L}{\partial \omega_i}\right)^2 \tag{3.16}$$

2. Una vez actualizado el valor A_i se ajusta el valor del peso ω_i de la siguiente forma:

$$\omega_i \leftarrow \omega_i - \frac{\alpha}{\sqrt{A_i}}\left(\frac{\partial L}{\partial \omega_i}\right) \tag{3.17}$$

- *RMS Prop*:Similar a *AdaGrad*, este algoritmo tambien usa $\sqrt{A_i}$, sin embargo también es utilizado un factor de decaimiento ρ de la siguiente manera:

$$A_i \leftarrow \rho A_i + (1-\rho)\left(\frac{\partial L}{\partial \omega_i}\right)^2 \tag{3.18}$$

  Y ajusta el peso ω_i de igual forma que *AdaGrad*.

3.5 Conceptos para el preprocesamiento de imágenes

Para que las imágenes pudieran ser procesadas de forma correcta por la red neuronal utilizada, y con el objetivo de reconocer la emoción en un rostro, se utilizaron los siguientes métodos provistos por librerías de OpenCV[32]:

3.5.1 Clasificadores en cascada

Un clasificador en cascada es un árbol de decisión degenerado en el que cada estación un clasificador es entrenado para detectar casi todos los objetos de interés (vea figura 3.10).

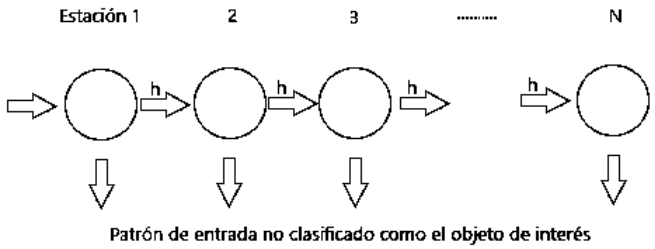


Figura 3.10: Clasificadores en cascada con N estaciones. En cada estación un clasificador es entrenado para alcanzar una tasa de éxito de h igual a h^N y una tasa de falsas alarmas de f igual a f^N.

3.5.2 Clasificador Haar

Este clasificador [30] se utilizó para encontrar la región en la que se encuentra un rostro en una imagen. Fue propuesto originalmente por Paul Viola y Michael Jones [50] y optimizado por Rainer Lienhart, Alexander Kuranov y Vadim Pisarevsky [36].

El algoritmo utilizado para este clasificador tiene un enfoque de aprendizaje de máquina donde una función en cascada es entrenada con un conjunto de imágenes positivas (imágenes que contienen aquello que se quiere detectar) y otro conjunto con imágenes negativas (imágenes que no contienen lo que se quiere detectar). En los inicios de este algoritmo, Viola y Jones, usaron 3 características. Las características se definen como la diferencia entre la suma de los pixeles en el área en blanco y la suma de los pixeles en el área negra (vea figura3.11). Para calcular todas estas operaciones de forma eficiente se utiliza la *Imagen integral*.

Imagen Integral:

La Imagen integral es una representación intermedia de la imagen original en el proceso para calcular las características. La imagen integral en la posición x, y contiene la suma de los pixeles encima y a la izquierda de x, y, inclusive, y se representa por la ecuación:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3.19}$$

donde $ii(x,y)$ es la imagen integral y $i(x,y)$ es la imagen original. Para poder calcular la imagen integral se utilizan las siguiente recurrencias:

$$s(x,y) = s(x,y-1) + i(x,y) \tag{3.20}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{3.21}$$

donde $s(x,y)$ es la suma por fila, $s(x,-1) = 0$, y $ii(-1,y) = 0$

Usando la imagen integral se puede calcular cualquier suma de los rectángulos en 4 arreglos (vea figura 3.12).

3.5.3 Ecualizar histograma

Para tratar de estandarizar lo más posible las imágenes y que hubiera el menor efecto negativo debido a la luz en cada imagen se decidió ecualizar el histograma de color de la imagen. El histograma de una imagen es una representación gráfica de la distribución de intensidad de una imagen. Dicho

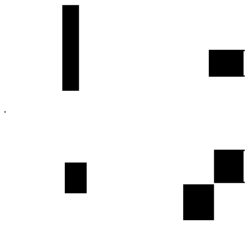


Figura 3.11: Ejemplo de características rectangulares sobre un área de la imagen a analizar. A la suma de los pixeles dentro del área en blanco se le resta la suma de los pixeles del área en negro. Características de 2 rectángulos se muestran en las imágenes a) y b). La imagen c) muestra una caraterística de 3 rectángulos y la figura d) muestra una característica de 4 rectángulos.

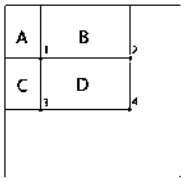


Figura 3.12: La suma de los pixeles dentro del rectángulo D puede ser calculada con 4 arreglos. El valor de la imagen integral en la posición 1 es la suma de los pixeles en el rectángulo A. El valor en la posición 2 es $A+B$, en la posición 3 es $A+C$ y en la posición 4 es $A+B+C+D$. La suma dentro de D puede ser calculada como $4+1-(2+3)$.

histograma cuantifica el número de pixeles de cada valor de intensidad considerado, en este caso la intensidad varia entre 0 y 255 [31]. El método por el cuál se ecualiza el histograma de una imagen mejora el contraste de ésta con la intención de ampliar su rango de intensidad[12]. Para este trabajo se utilizó la función *equalizeHist()* de las librerías de OpenCV [31]. La figura 3.13 muestra una de las imágenes utilizadas en este trabajo antes y después de haber ecualizado su histograma.

Figura 3.13: a) Es la imagen original en blanco y negro. b) Es la imagen que se muestra en a) después de haber equalizado su histograma de color.

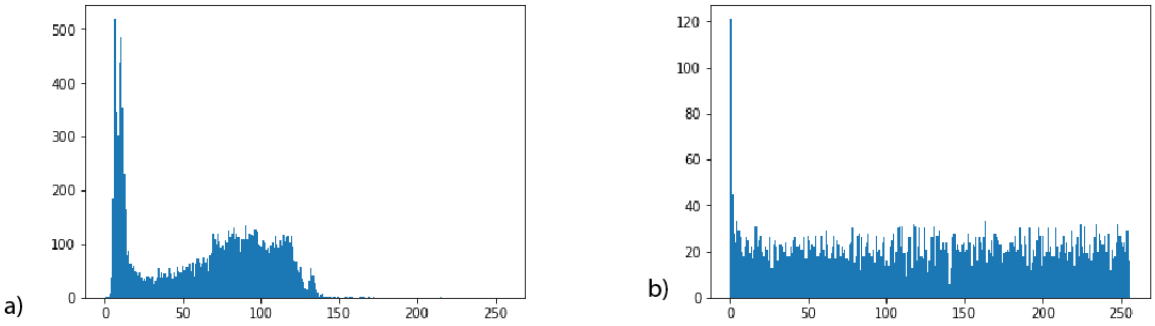


Figura 3.14: a) Es el histograma de la imagen 3.13 a). b) Es el histograma de la imagen 3.13 b). Como se aprecia en b) la intensidad de color es mucho más homogénea que en a).

Parte II

METODOLOGÍA Y EXPERIMENTACIÓN

Capítulo 4

Metodología

4.1 Herramientas

Las imágenes utilizadas fueron obtenidas de las base de datos *The Extended Cohn-Kanade Dataset* (CK+) [24], The Averaged Karolinska Directed Emotion Faces - AKDEF [25] y fotografías tomadas de alumnos de la Facultad de Ciencias Físico Matemáticas de la Universidad Michoacana de San Nicolás de Hidalgo.

Para la realización de este trabajo se emplearon las librerías de Numpy [38], Keras [21], OpenCV [32].

Numpy es un conjunto de librerías óptimas para el cómputo científico en Python. Esta librería se utilizó para crear los tensores que contienen a las imágenes de entrenamiento y validación (ver Sección 4.3)

Keras es un API de alto nivel diseñada para el desarrollo de redes neuronales artificiales, escrita en Python, que a su vez utiliza TensorFlow [48], CNTK [26], o Theano [49]. Para este trabajo se utilizó Keras con TensorFlow, en el entorno de desarrollo Spyder [43], distribuido por Anaconda[4] para el lenguaje Python 3.7.3 [34].

TensorFlow es una librería de código abierto creada para cómputo numérico utilizando flujo de grafos. Los nodos de estos grafos representan operaciones matemáticas, mientras que las aristas representan tensores [47].

OpenCV es una librería de código abierto de visión computacional y software de aprendizaje de máquina. Fue construido para proveer infraestructura común de aplicaciones de visión computacional y acelerar el uso de percepción de una máquina en productos comerciales. Esta librería cuenta con más de 2500 algoritmos optimizados. Estos algorimos pueden ser usados para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, ras-

trear objetos, etc. OpenCV se utilizó para el preprocesamiento de imágenes como se explica en la sección 3.5. Sin embargo, para el desarrollo de la red neuronal convolucional artificial se utilizó Keras.

4.2 Preprocesamiento de imágenes

Como primer paso cada imagen que se utilizó fue convertida a una imagen en escala de grises.

Dado que las imágenes no siempre muestran un rostro en la misma ubicación se utilizó el algoritmo Haar de OpenCV para encontrarlo, una vez encontrado se cortó la imagen y el resultado fue una imagen que muestra el rostro de una persona sin lo que pudiera haber alrededor (como algún fondo, cuello, hombros, etc.). El proceso anterior se ilustra en la imagen 4.1.

Figura 4.1: Para que una imagen pueda ser proporcionada a la red neuronal convolucional propuesta, ésta debe estar en escala de grises. Una vez en escala de grises se procede a utilizar el algoritmo Haar de OpenCV para encontrar la región que será recortada. a) imagen original, b) imagen en escala de grises y c) enmarca la región del rostro.

Una vez encontrado el rostro se procede a cortar la imagen para crear una imagen nueva, más pequeña, que contiene el rostro. Posteriormente se procede a redimensionar la imagen nueva unas nuevas medidas, esto para que todas las imágenes tengan las mismas dimensiones, para después aplicarle la función equalizeHist de OpenCV. El proceso se ilustra en la imagen 4.2.

Al término del proceso anteriormente mencionado se utilizó Numpy para crear 4 archivos que contienen las imágenes en forma de matrices y las etiquetas de cada una de estas imágenes. De los archivos creados, 2 corresponden a las matrices de las imágenes de entrenamiento y validación y 2 corresponden a los arreglos de las etiquetas de cada una de esas imágenes.

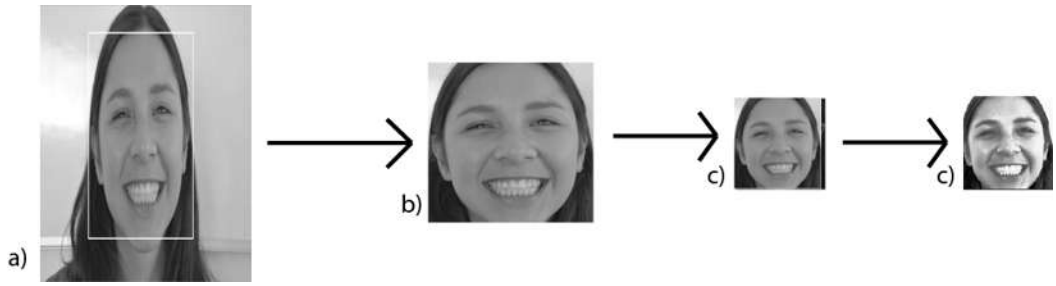


Figura 4.2: a) Imagen en la que se enmarca el rostro, b) imagen que sólo contiene la región enmarcada, c) imagen que sólo contiene el rostro, con nuevas dimensiones y c) imagen con nuevas dimensiones cuyo histograma ya fue ecualizado.

4.3 Experimentación

Como se mencionó en la sección 3.4, al ser una red reuronal un modelo de aprendizaje supervisado ésta debe ser entrenada con datos etiquetados que contienen la información que queremos que aprenda. Estos datos están guardados en los archivos mencionados anteriormente.

Para cargar dichos archivos se siguió el siguiente procedimiento:

- `imagenes_entrenamiento = numpy.load(img_entrenamiento)`
- `imagenes_prueba = numpy.load(img_prueba)`
- `etiquetas_entrenamiento = numpy.load(etiq_entrenamiento)`
- `etiquetas_prueba = numpy.load(etiq_prueba)`
- `imagenes_entrenamiento = imagenes_entrenamiento[nombre_arreglo]`
- `imagenes_prueba = imagenes_prueba[nombre_arreglo]`
- `etiquetas_entrenamiento = etiquetas_entrenamiento[nombre_arreglo]`
- `etiquetas_prueba = etiquetas_prueba[nombre_arreglo]`

Una vez cargadas las imágenes tuvieron que ser transformadas a tensores, a su vez que se obtienen las categorías, es decir, el conjunto al que pertenecen las etiquetas. Esto se logró con el siguiente procedimiento:

- `imagenes_entrenamiento = imagenes_entrenamiento.reshape((tam_datos_entrenamiento, alto, ancho, num_canales))`
- `imagenes_entrenamiento = imagenes_entrenamiento.astype('float') / 255`
- `imagenes_prueba = imagenes_prueba.reshape((tam_datos_prueba,alto, ancho, num_canales))`
- `imagenes_prueba = imagenes_prueba.astype('float') / 255`
- `etiquetas_entrenamiento = to_categorical(etiquetas_entrenamiento)`
- `etiquetas_prueba = to_categorical(etiquetas_prueba)`

Para este trabajo las categorías utilizadas fueron las siguientes:

- 0 = Miedo

- 1 = Enojo
- 2 = Disgusto
- 3 = Felicidad
- 4 = Neutral
- 5 = Tristeza
- 6 = Sorpresa

Figura 4.3: Ejemplos de imágenes de cada una de las categorías. a) Miedo, b) sorpresa, c) disgusto, d) tristeza, e) felicidad, f) neutral, g) enojo.

En Keras se debe declarar el tipo de modelo que tendrá la red neuronal artificial a construir, en este caso es un modelo secuencial, dado por la siguiente instrucción:

- red = models.Sequential()

Luego de definir el modelo se agregan capas con el método add() el cuál recibe un objeto de tipo layers. Para este trabajo los objetos de tipo layers utilizados fueron los siguientes:

- `Conv2D (filters = num_filtros, kernel_size = (a,b), activation = func_activacion, input_shape =(m,n))`.
  Esta es una capa convolucional que recibe matrices (imágenes) de $m \times n$ (este argumento sólo se utiliza en la primer capa) sobre la que se aplica la convolución en regiones de $a \times b$ creando num_filtros filtros con la función de activación func_activación.
- `MaxPooling2D (pool_size = (x,y))`.
  Esta capa hace la operación MaxPooling sobre regiones de tamaño $x \times y$
- `Flatten ()`.
  Esta capa convierte la matriz a un vector.
- `Dense (units = num_neuronas, activation = func_activacion)`.
  Esta capa no es convolucional y agrega las neuronas en forma de columna como una red neuronal con propagación hacia delante tradicional.

Posterior a definir la arquitectura de la red neuronal a probar se puede agregar:

- `red.summary()`.

  Este método se utiliza para mostrar la arquitectura definida para la red neuronal.

Una vez agregadas todas las capas deseadas se debe compilar la red neuronal de la siguiente manera:

- `red.compile(optimizer= optimizador, loss=fun_pérdida, metrics=[metrica])`.

  Este método define el optimizador, la función de pérdida y la métrica. Adicionalmente se pueden agregar más parámetros pero éstos son lo que se definen en este trabajo.

Posterior a compilar la red neuronal se procede a la fase de entrenamiento por medio del método fit() de la siguiente manera:

- `red.fit(x = imagenes_entrenamiento, y = etiquetas_entrenamiento, epochs = épocas, batch_size = tam_lote, verbose = num)`.

  `imagenes_entrenamiento` es el arreglo creado a partir de las matrices de las imágenes que se escogieron para entrenar la red, `etiquetas_entrenamiento` son las etiquetas correspondientes a cada una de las `imágenes_entrenamiento`, epocas es el número de iteraciones sobre todo el conjunto de imágenes que se desee realice la red, `tam_lote` es el número de imágenes que se propagarán a través de la red. Este método es el ejecuta el entrenamiento de la red neuronal.

- `red.evaluate(imagenes_prueba, etiquetas_prueba)`.

  Donde `imágenes_prueba` son las matrices de las imágenes escogidas para esta etapa y las `etiquetas_prueba` son las etiquetas de cada imagen de prueba.

4.3.1 Primera Red Propuesta

En todas las arquitecturas de este trabajo la última capa será Dense con 7 neuronas ya que corresponde con el número de categorías.

Por medio de este método se construyó la primer red neuronal convolucional a probar que tiene la siguiente arquitectura:

- `red.add(layers.Conv2D(filters = 64, kernel_size = (5,5), activation = 'relu', input_shape = (110,110,1)))`

- `red.add(layers.MaxPooling2D (pool_size = (2,2)))`
- `red.add(layers.Conv2D(filters = 32, kernel_size = (5,5), activation = 'relu'))`
- `red.add(layers.MaxPooling2D (pool_size = (2,2)))`
- `red.add(layers.Conv2D(filters = 16, kernel_size = (5,5), activation = 'relu'))`
- `red.add(layers.Flatten())`
- `red.add(layers.Dense(units = 64, activation = 'relu'))`
- `red.add(layers.Dense(units = 7, activation = 'sigmoid'))`

Una vez diseñada la arquitectura de la red neuronal se agregan los métodos red.compile() y red.fit() de la siguiente manera:

- `red.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accura`
- `red.fit(x = imagenes_entrenamiento, y = etiquetas_entrenamiento, epochs = 20, batch_size = 7, verbose = 1)`

Ya entrenada la red neuronal se procedió a evaular su efectividad con el método `red.evaluate()` de la siguiente manera:

- `red.evaluate(imagenes_prueba, etiquetas_prueba)`

Cuando este método finaliza su ejecución se obtiene la efectividad y, opcionalmente, se puede construir una matriz de confusión en la que se muestra el total de aciertos y el total de errores.

La red antes mencionada tuvo una efectividad de 83,43 %.

4.3.2 Segunda Red Propuesta

Con estos resultados se decidió realizar modificación en la arquitectura de la red neuronal convolucional anterior para intentar alcanzar una efectividad mayor. La arquitectura lograda para la segunda red neuronal fue la siguiente:

- `red.add(layers.Conv2D(filters = 20, kernel_size = (3,3), activation = 'tanh', input_shape = (alto,ancho,1)))`
- `red.add(layers.MaxPooling2D (pool_size = (1,4)))`

- `red.add(layers.Conv2D(filters = 20, kernel_size = (2,2), activation = 'tanh'))`
- `red.add(layers.MaxPooling2D (pool_size = (2,2)))`
- `red.add(layers.Conv2D(filters = 20, kernel_size = (2,2), activation = 'tanh'))`
- `red.add(layers.Conv2D(filters = 20, kernel_size = (2,2), activation = 'tanh'))`
- `red.add(layers.Flatten())`
- `red.add(layers.Dense(units = 100, activation = 'tanh'))`
- `red.add(units = 100, activation = 'relu'))`
- `red.add(units = 7, activation = 'sigmoid'))`

La segunda red neuronal que se probó alcanzó una efectividad máxima del 90.63 %, sin embargo, al cambiar el tamaño de la región sobre la que se hizo el MaxPooling en la segunda capa a uno de 1×6 la efectividad aumentó. A la red que incluye este cambio se le llamará la red propuesta. Dado que el porcentaje de efectividad varía entre entrenamientos, se decidió entrenar varias veces la red propuesta alcanzando porcentajes de efectividad de entre 91.65 % y 94.84 % el cual fue el valor máximo alcanzado.

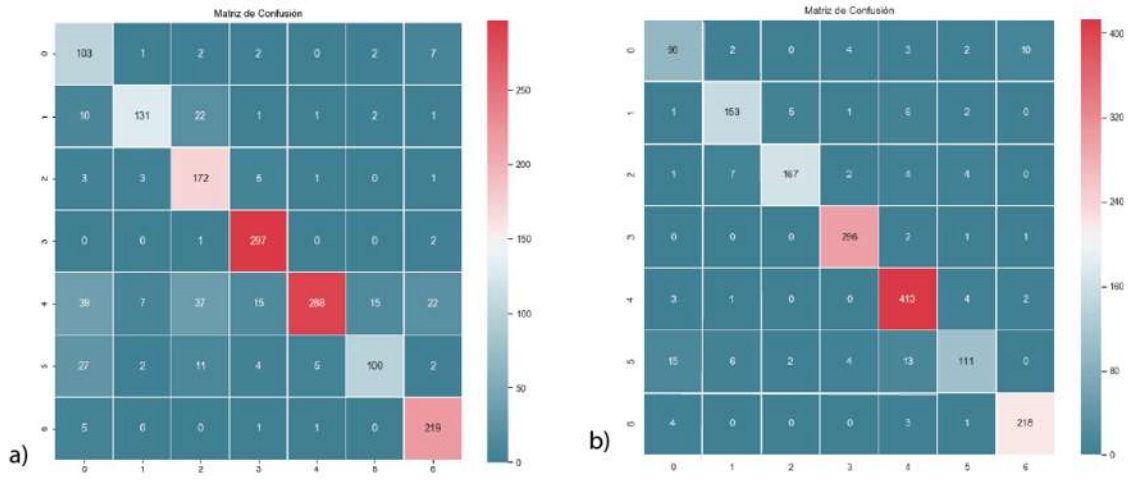


Figura 4.4: a) Muestra la matriz de confusión de una red neuronal con efectividad del 83.43 % y b) muestra la matriz de confusión de una red neuronal con efectividad del 90.63 %. Una matriz de confusión muestra cuantas predicciones hizo correctamente y en cuales predicciones se equivocó, por ejemplo, en la matriz a) la primer red neuronal confundió en una imagen un 4 (neutral) con un 6 (sorpresa).

Los resultados de ambas redes neuronales experimentadas se muestran en una matriz de confusión presentada en la Figura 4.4.

Cabe mencionar que el preprosesamiento de las imágenes jugó un papel sumamente importante ya que, al realizar pruebas con las imágenes sin haber equalizado el histograma, usando las mismas arquitecturas la efectividad más alta lograda llegó al 80.3 %

Capítulo 5

Caso de Uso: Emociones Experimentadas por un Grupo de Alumnos en una Clase Académica

Una vez alcanzada una efectividad bastante alta se procedió a utilizarla en un contexto real, en este caso fue en una clase académica. Para utilizar la red neuronal propuesta se les pidió a los alumnos de esa clase que realizaran problemas de programación mientras eran grabados con la *web cam* de la computadora. Estos videos fueron guardados para su posterior análisis. Para el análisis de los videos se aplicó el preprocesamiento mencionado en la sección 4.2 a cada cuadro de los videos obtenidos y redimensionando la imagen resultante a 110×110 pixeles. Todo este proceso resultó en analizar un total de 427,983 imágenes. Aunque podría no parecer que la variación entre los porcentajes de efectividad afecte en gran medida el análisis, se debe tomar en cuenta que clasificar correctamente un 3 % más de imágenes representa acertar en casi 13,000 casos en los que de otro modo se habría fallado.

Para la red neuronal propuesta, se inicializó con 4 valores distintos, después de la etapa de entrenamiento se consiguieron pesos diferentes, los cuales dieron 4 redes neuronales de distinta efectividad. Éstas fueron del 90.63 %, 91.65 %, 93.69 % y 94.84 %.

Estas redes se usaron para evaluar los videos de los estudiantes y cuyos resultados se muestran en las tablas siguientes. Cada barra muestra el porcentaje de tiempo que el alumno manifestó una emoción. Por ejemplo, en la Figura 5.1 se muestra que alumnos tuvieron, durante mas tiempo, dos tipos de emociones: miedo y sorpresa.

En la figura 5.5 se muestra la arquitectura de la red neuronal propuesta.

Efectividad del 90.63 %						
Emoción	Alumno 1	Alumno 2	Alumno 3	Alumno 4	Alumno 5	Alumno 6
Miedo	12.819 %	12.760 %	11.463 %	17.009 %	14.171 %	12.697 %
Enojo	15.303 %	19.019 %	3.803 %	22.751 %	8.993 %	11.377 %
Disgusto	2.467 %	4.536 %	1.157 %	7.233 %	3.970 %	3.238 %
Feliz	0.318 %	0.267 %	0.337 %	0.174 %	0.672 %	0.953 %
Neutral	6.331 %	0.705 %	0.632 %	1.305 %	1.628 %	1.972 %
Triste	1.472 %	2.240 %	1.751 %	2.134 %	3.672 %	2.968 %
Sorpresa	12.346 %	13.361 %	6.402 %	7.793 %	7.805 %	7.252 %

Cuadro 5.1: Resultados de analizar los videos con una red neuronal que provee una efectividad del 90.63 %

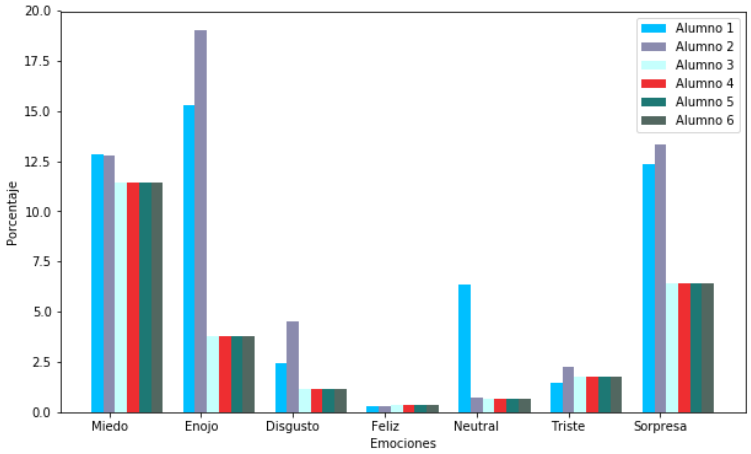


Figura 5.1: Gráfico de barras en el que se aprecia una comparación entre las emociones que expresaron los alumnos durante la dinámica de clase propuesta utilizando una red neuronal con una efectividad del 90.63 %.

Efectividad del 91.65 %						
Emoción	Alumno 1	Alumno 2	Alumno 3	Alumno 4	Alumno 5	Alumno 6
Miedo	1.052 %	3.694 %	1.917 %	2.232 %	3.695 %	3.278 %
Enojo	7.989 %	5.498 %	2.380 %	5.235 %	5.070 %	8.877 %
Disgusto	11.796 %	2.439 %	1.595 %	15.591 %	2.500 %	3.784 %
Feliz	3.825 %	2.936 %	1.281 %	13.106 %	4.651 %	4.107 %
Neutral	22.870 %	20.906 %	11.149 %	12.569 %	10.014 %	11.207 %
Triste	1.947 %	7.305 %	3.282 %	6.499 %	9.228 %	4.751 %
Sorpresa	1.578 %	10.108 %	3.942 %	3.168 %	5.752 %	4.455 %

Cuadro 5.2: Resultados de analizar los videos con una red neuronal que provee una efectividad del 91.65 %

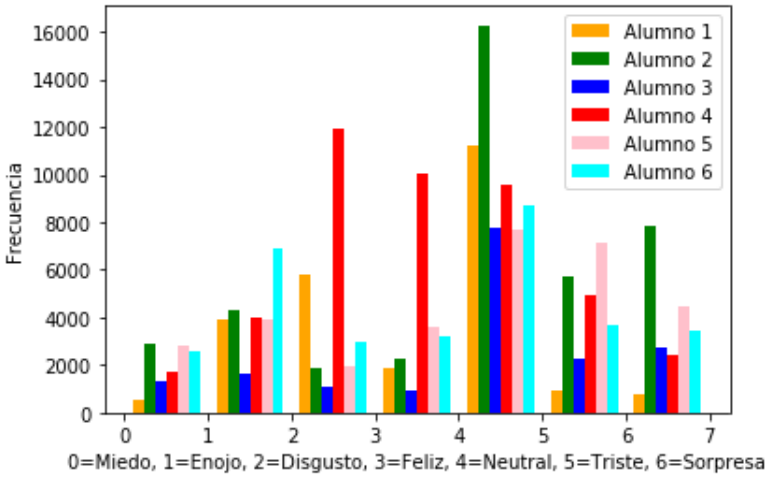


Figura 5.2: Gráfico de barras en el que se aprecia una comparación entre las emociones que expresaron los alumnos durante la dinámica de clase propuesta utilizando una red neuronal con una efectividad del 91.65 %.

Efectividad del 93.69 %						
Emoción	Alumno 1	Alumno 2	Alumno 3	Alumno 4	Alumno 5	Alumno 6
Miedo	2.326 %	1.371 %	3.382 %	2.324 %	3.268 %	2.606 %
Enojo	23.804 %	6.108 %	1.401 %	10.423 %	7.050 %	14.127 %
Disgusto	9.700 %	2.404 %	3.071 %	18.442 %	8.991 %	4.044 %
Feliz	3.556 %	3.327 %	2.305 %	11.723 %	4.257 %	2.785 %
Neutral	6.048 %	14.285 %	4.433 %	7.774 %	6.064 %	7.107 %
Triste	3.829 %	5.977 %	5.209 %	3.0009 %	5.324 %	5.111 %
Sorpresa	1.794 %	19.414 %	5.743 %	4.713 %	5.958 %	4.678 %

Cuadro 5.3: Resultados de analizar los videos con una red neuronal que provee una efectividad del 93.69 %

Efectividad del 94.84 %						
Emoción	Alumno 1	Alumno 2	Alumno 3	Alumno 4	Alumno 5	Alumno 6
Miedo	3.621 %	3.833 %	1.937 %	3.652 %	4.551 %	3.496 %
Enojo	9.441 %	11.947 %	2.142 %	4.913 %	7.250 %	14.480 %
Disgusto	2.752 %	1.028 %	3.166 %	23.969 %	4.685 %	4.188 %
Feliz	3.138 %	2.101 %	2.728 %	4.0418 %	4.285 %	4.0292 %
Neutral	20.397 %	8.387 %	3.388 %	13.377 %	6.957 %	4.889 %
Triste	8.549 %	6.260 %	1.800 %	3.916 %	3.851 %	3.191 %
Sorpresa	3.158 %	19.331 %	10.386 %	4.531 %	9.334 %	6.186 %

Cuadro 5.4: Resultados de analizar los videos con una red neuronal que provee una efectividad del 94.84 %

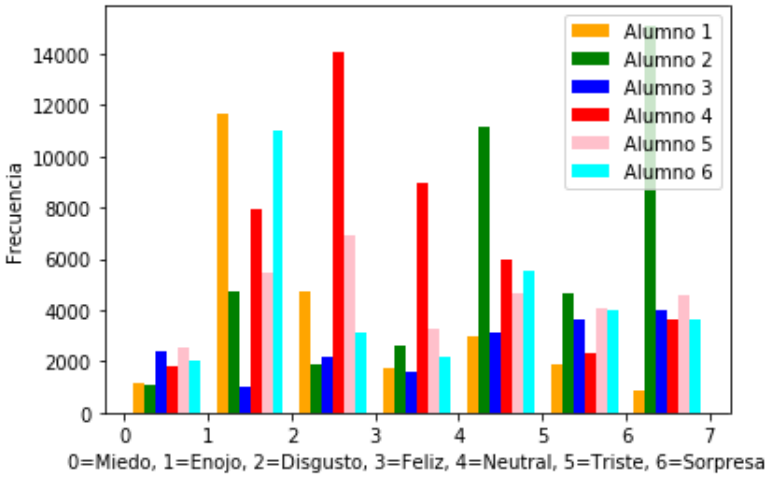


Figura 5.3: Gráfico de barras en el que se aprecia una comparación entre las emociones que expresaron los alumnos durante la dinámica de clase propuesta utilizando una red neuronal con una efectividad del 93.69 %.

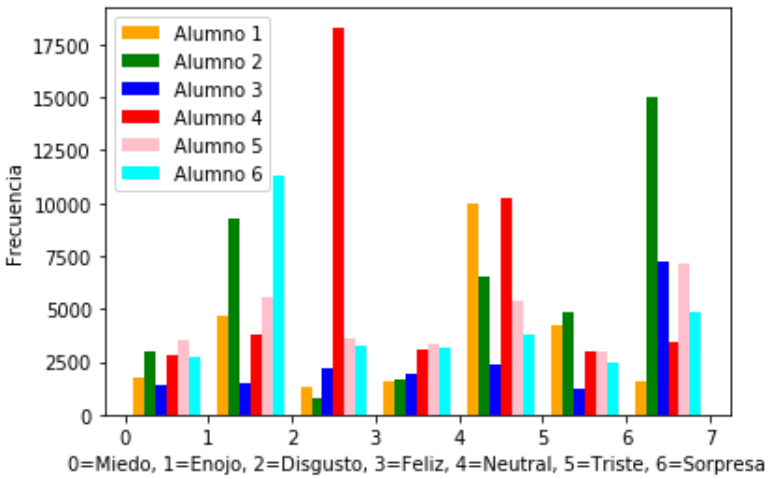


Figura 5.4: Grafico de barras en el que se aprecia una comparación entre las emociones que expresaron los alumnos durante la dinámica de clase propuesta utilizando una red neuronal con una efectividad del 94.84 %.

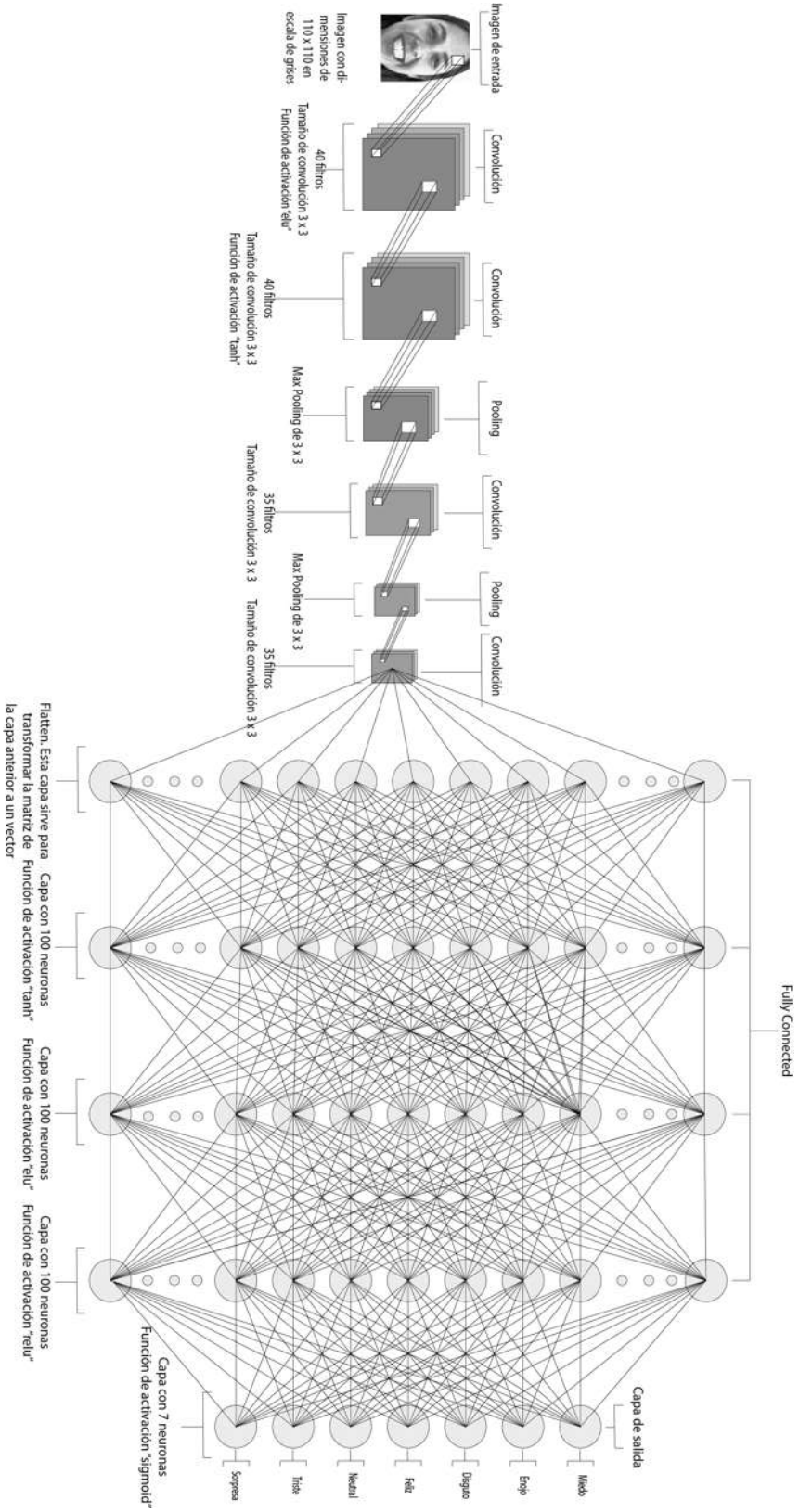


Figura 5.5: Arquitectura de la red neuronal convolucional artificial propuesta.

Parte III

CONCLUSIONES

Capítulo 6

Conclusiones y Trabajo a Futuro

Se realizó un estudio del funcionamiento de las redes neuronales convolucionales como clasificadores para después crear una red de ese tipo que pudiera catalogar expresiones faciales dentro de las categorías: *neutral, felicidad, tristeza, sorpresa, enojo, miedo* y *disgusto*. Dicha red neuronal demostró ser altamente acertada alcanzando una efectividad del 94,84 %.

Una vez alcanzada esta efectividad se utilizó dicha red para analizar videos de 6 alumnos mientras realizaban problemas de programación. Con ésto se demostró que existe más de una forma de evaluar el desempeño de un estudiante más allá de un examen o trabajos en clase.

Como trabajo futuro existe la posibilidad de llevar este tipo de análisis en tiempo real y a un mayor número de alumnos para lograr tener información más objetiva respecto al comportamiento de los alumnos a lo largo de los cursos que tomen. Para esto se requerirá crear un sistema de reconocimiento automático de identidad y lograr el seguimiendo individual de cada persona, lo cuál también se puede lograr con redes neuronales convolucionales.

Bibliografía

[1] Aggarwal, C.C.: Neural Networks and Deep Learning A Text Book. Springer (2018)

[2] Amazon: Aws amazon rekognition. Recuperado de: https://docs.aws.amazon.com/es_es/rekognition/latest/dg/rekognition-dg.pdf (2019), fecha de último acceso: 29-01-2019

[3] Amherst, U.M.: Reinforcemente learning repository. Recuperado de : http://www-anw.cs.umass.edu/rlr/, fecha de último acceso: 05-06-2019

[4] Anaconda: Anaconda. Recuperado de: https://www.anaconda.com/ (2019), fecha de último acceso: 05-06-2019

[5] Apostol, T.M.: Calculus II. Editorial Reverté (2016)

[6] Araujo, A., Pérez, J., Rodriguez, W.: Aplicación de una red neuronal convolucional para el reconocimiento de personas a través de la voz. Sexta Conferencia de Computación, Informática y Sistemas, Mérida, Venezuela (2018)

[7] Cerekovic, A.: A deep look into group happiness prediction from images. ICMI 2016 (2016), "DOI: https://dl.acm.org/citation.cfm?doid=2993148.2997628"

[8] Cloud, G.: Detecting faces | cloud vision api documentation. Recuperado de: https://cloud.google.com/vision/docs/detecting-faces (2019), fecha de último acceso: 29-01-2019

[9] CrowdEmotion: Crowdemotion api. Recuperado de: https://www.crowdemotion.co.uk/product/ (2019), fecha de último acceso: 29-01-2019

[10] Dmitry Storcheus, A.R., Kumar, S.: A survey of modern questions and challenges in feature extraction. The 1st International Workshop "Feature Extraction: Modern Questions and Challenges" **1**, 4 (2015)

[11] EmoVu: Eyeris emovu sdk. Recuperado de: http://emovu.com/docs/html/getting_started.htm (2019), fecha de último acceso: 29-01-2019

[12] Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Prentice Hall (2008)

[13] Grimaldil, R.P.: Calculus II. Editorial Reverté (1998)

[14] Hossain, M.S., Muhammad, G.: Emotion recognition using learning approach from audio-visual emotion big data. Information Fusion (2018), "DOI: https://doi.org/10.1016/j.inffus.2018.09.008"

[15] Hubel, H.D., Wisel, T.N.: Receptive fields of single neurones in the cat's striate cortex. The Journal of Physiology **124**, 574 – 591 (1959)

[16] ImageNet: Large scale visual recognition challenge. Recuperado de: `http://www.image-net.org/challenges/LSVRC/` (2019), fecha de último acceso: 18-06-2019

[17] iMotions: Affectiva. Recuperado de: `https://imotions.com/affectiva/` (2019), fecha de último acceso: 29-01-2019

[18] iMotions: Affectiva. Recuperado de: `https://imotions.com/emotient/` (2019), fecha de último acceso: 05-05-2019

[19] Jones, H.: Ciencia de los Datos. La guía definitica sobre análisis de datos, minería de datos, almacenamiento de datos, visualización de datos, Big DAta para empresas y aprendizaje automático para principiantes. Amazon Books (2019)

[20] Kairos: Kairos. Recuperado de: `https://www.kairos.com/features` (2019), fecha de último acceso: 29-01-2019

[21] Keras: Keras. Recuperado de: `https://keras.io/` (2019), fecha de último acceso: 05-06-2019

[22] Krizhevsky, A., Sutskever, I., Hilton, G.: Imagenet classification with deep convolutional neural networks. NIPS Conference pp. 1097 – 1105 (2012)

[23] Leon A. Gatys, Alexander S. Ecker, M.B.: A neural algorithm of artistic style. Recuperado de: `https://arxiv.org/abs/1508.06576`, fecha de último acceso: 05-06-2019

[24] Lucey, P., Cohn, J.F., Kanade, T., Saragih, J., Ambar, Z., Matthews, I.: The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. Proceedings of IEEE workshop on CVPR for Human Communicative Behavior Analysis, San Francisco, USA (2010)

[25] Lundqvist, D., Litton, J.E.: The averaged karolinska directed emotional faces - akdef. Department of Clinical Neuroscience, Psychology section (2010), iSBN 91-630-7164-9

[26] Microsoft: Cntk. Recuperado de: https://docs.microsoft.com/en-us/cognitive-toolkit/ (2019), fecha de último acceso: 05-06-2019

[27] Microsoft: Face api. Recuperado de: https://azure.microsoft.com/es-mx/services/cognitive-services/face/ (2019), fecha de último acceso: 29-01-2019

[28] Noldus: Facereader. Recuperado de: https://www.noldus.com/human-behavior-research/products/facereader (2019), fecha de último acceso: 29-01-2019

[29] NVISO: Insights. Recuperado de: https://www.nviso-insights.com/en/insights-develop (2019), fecha de último acceso: 29-01-2019

[30] OpenCV: Clasificador haar en cascada basado en características. Recuperado de: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html (2019), fecha de último acceso: 05-05-2019

[31] OpenCV: Histograma. Recuperado de: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html (2019), fecha de último acceso: 05-05-2019

[32] OpenCV: Opencv. Recuperado de: https://opencv.org/ (2019), fecha de último acceso: 05-05-2019

[33] Paul Ekman, e.a.: Are the basic emocions? Psicological Review **99**, 550 – 553 (2002)

[34] Python: Python 3. Recuperado de: https://www.python.org/ (2019), fecha del último acceso: 05-06-2019

[35] R. Vizcaya Cárdenas, J. M. Flores Albino, V.M.L.M.y.S.L.S.: Desempeño de una red neuronal convolucional para clasificación de señales de tránsito vehicular. The 1st International Workshop "Feature Extraction: Modern Questions and Challenges" **1**, 4 (2015)

[36] Rainer Lienhart, A.K., Pisarevsky, V.: Empirical analisis of detection cascades of boosted classifiers for rapid object detection. IEEE Neural Networks for Perception **1**, 65 – 93 (2002)

[37] Rusell, S., Norving, P.: Inteligencia artificial un enfoque moderno. Pearson - Prentice Hall (2008)

[38] Scipy: Numpy. Recuperado de: https://www.numpy.org/ (2019), fecha de último acceso: 05-06-2019

[39] Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press (2014)

[40] Shobha, G., Rangaswamy, S.: Mathematical Methods for Physics and Engineering. Cambridge (2018)

[41] Sightcorp: F.a.c.e api. Recuperado de: https://face-api.sightcorp.com/dev-documentation/ (2019), fecha de último acceso: 29-01-2019

[42] SkyBiometry: Skybiometry api. Recuperado de: https://skybiometry.com/Documentation/ (2019), fecha de último acceso: 29-01-2019

[43] Spyder: Spyder ide. Recuperado de: https://www.spyder-ide.org/ (2019), fecha de último acceso: 05-06-2019

[44] Stewart, J.: Cálculo de varias variables. Trascendentes tempranas. Cengage Learning (2012)

[45] Talabis, M.R., McPherson, R., Miyamoto, I., MArtin, J.L., Kaye, D.: Information Security Analytics: Finding Security Insights, Patterns and Anomalies in Big Data. Syngress (2015)

[46] Tan, L., Zhang, K., Wang, K., Zeng, X., Peng, X., Qiao, Y.: Group emotion recognition with individual facial emotion cnns and global image based cnns. ICMI 2017 (2017), "ACM ISBN: 978-1-4503-5543-8/17/11"

[47] TensorFlow: Github de tensorflow. Recuperado de: https://github.com/tensorflow/tensorflow (2019), fecha de último acceso: 05-06-2019

[48] TensorFlow: Tensorflow. Recuperado de: https://www.tensorflow.org/ (2019), fecha de último acceso: 05-06-2019

[49] Theano: Theano. Recuperado de: http://deeplearning.net/software/theano/ (2019), fecha de último acceso: 05-06-2019

[50] Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. IEEE Neural Networks for Perception **1**, 65 – 93 (2001)

[51] W., A.H., R., A.S., M., A.N.: Facial emotion recognition from videos using deep convolutional neural networks. International Journal of Machine Learning and Computing **9** (2019)

[52] Yoshua Bengio, A.C., Vincent, P.: Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**, 1798 – 1828 (2013)

[53] Yu, Z., Zhang, C.: Image based static facial expression recognition with multiple deep network learning. ICMI 2015 (2015), "ACM ISBN: 978-1-4503-3912-4/15/11"