Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Reconocimiento de expresiones faciales usando redes de cápsulas con enrutado dinámico

Trabajo especial de grado presentado ante la ilustre
Universidad Central de Venezuela por

Br. José Antonio Hernández Guédez.
C.I. 23.628.993

Tutor: Rhadamés Carmona

Caracas, Febrero 2019.

Caracas, 24 de Mayo de 2019
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

REPUBLICA DE VENEZUELA
Facultad de Ciencias
Escuela de Computación
Universidad Central - Caracas

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el Bachiller José Antonio Hernández Guédez, C.I: 23.628.993 con el título "Reconocimiento de expresiones faciales usando redes de cápsulas con enrutamiento dinámico", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 24 de Mayo de 2019, a las 11:00am, para que su autor lo defendiera en forma pública, en el Centro de Computación Gráfica, lo cual realizó mediante una exposición oral de su contenido, y luego respondió satisfactoriamente a las preguntas que le fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con una calificación de veinte (20) puntos.

En fe de lo cual se levanta la presente acta, en Caracas el 24 de Mayo de 2019, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Tutor Rhadamés Carmona.

Dr. Rhadamés Carmona
(Tutor)

Prof. Eugenio Scalise
(Jurado Principal)

Prof. Francisco Moreno
(Jurado Principal)

Resumen

En los últimos años el reconocimiento de expresiones faciales ha llamado la atención de muchos investigadores. Estos han realizado una gran cantidad de estudios que han llevado al desarrollo de distintas soluciones, pero ninguno usando el modelo de redes de cápsulas con enrutado dinámico. En este trabajo especial de grado se crea un modelo de detección de emociones basado en un modelo de aprendizaje de máquina reciente, llamado red de cápsulas. Adaptamos el modelo al problema en mano, se realizan comparaciones con un modelo convolucional y se describen posibles mejoras y soluciones.

Palabras clave: Redes de cápsulas, enrutado dinámico, procesamiento de imágenes, clasificación de emociones, tensorflow, keras, python.

Índice general

Índice de figuras

Índice de tablas

Índice de ecuaciones

Introducción

La ciencia de la computación ha tratado de responder y solventar problemas desde su concepción, sus soluciones comprenden desde la resolución un de sistemas de ecuaciones lineales hasta la automatización de procesos en una cadena de producción. Notamos la evolución de la computación como respuesta a las necesidades de las personas. Al mismo tiempo que la computación evoluciona lo hacen también las necesidades y la posibilidad de responder a estas.

Desde la última década, donde la internet y las redes sociales se convirtieron en un método común para socialización, nuevos proyectos han surgido, desde identificaciones faciales hasta el análisis de grandes volúmenes de datos. Todo esto es posible, gracias al desarrollo del hardware, el cual nos permite indagar en soluciones a problemas que antes no se consideraban debido a limitaciones existentes. Así pues, nos encontramos con nuevas posibilidades y nuevas interrogantes. Entre ellas, el reconocimiento y la búsqueda de patrones en imágenes es una, sin duda, de las más interesantes.

Las imágenes nos pueden dar un sin fin de información, pero el análisis de las mismas no es algo trivial. En la actualidad existe un interés creciente en las redes neuronales para resolver problemas triviales para un ser humano. Si bien muchas de estas preguntas son simples para un ser humano, no lo son para un computador, después de todo, ver y pensar no es algo que una computadora pueda hacer de la forma en la que un humano lo hace.

La clasificación de expresiones faciales es un tema complicado para un computador pero fácil para un ser humano. Estudios en psicología muestran que el ser humano expresa la mayor parte de su carga emocional a través de expresiones faciales [25], indicando que solo el 7% es la palabra verbal, la parte vocal (entonación de la voz) el 38%, mientras que las expresiones faciales contribuye al 55%. Adicionalmente las expresiones faciales dan información acerca del estado cognitivo del individuo, por ejemplo sus intereses, entusiasmo, confusión, estrés y señales en la conversación como énfasis en la voz y sintaxis en la forma de hablar.

Las computadoras de hoy en dia no poseen la habilidad suficiente para reconocer y actuar acorde a las emociones del individuo [26]. Con el objetivo de mejorar la interacción humano-computador el interés para el estudio de este problema ha crecido, atrayendo diferentes especialidades como ciencia de la computación, ingeniería, psicología y neurología. En los últimos 20 años se han realizado una gran cantidad de estudios, empezando por Paul Ekman [27], usando un enfoque psicológico. En los 90, ingenieros empezaron a usar este trabajo para la construcción de métodos automatizados de reconocimiento de expresiones faciales en imágenes y videos [28].

En este trabajo se explora una solución que hace uso de un algoritmo de redes de cápsulas, el cual es un modelo de solución de problemas que requieren aprendizaje de máquina. Para esto, se trabajó con detección de rostros y diversas herramientas y tecnologías. El documento está divido en capítulos donde, el primer capítulo expone el problema de detección de emociones de forma general, nuestra estrategia de solución e información básica sobre los objetivos y alcances de la investigación. Seguido tenemos el capítulo 2 que establece un marco teórico sobre la historia y soluciones anteriores, así como la solución propuesta, para así posteriormente en el capítulo 3 donde especificamos los detalles del conjunto de datos a usar, la creación y evaluación del modelo de redes de cápsulas, no solo con los datos de validación, si no también con un modelo de redes convolucionales profundas. Por último en el capítulo 4 concluimos el trabajo de investigación y otorgamos directrices para trabajos futuros. Existe un apartado final llamado anexos, con información útil para el lector que desee reproducir el modelo expuesto.

CAPÍTULO I. Planteamiento del problema

En este capítulo exploramos la problemática a evaluar a lo largo del trabajo, donde formulamos una propuesta para su solución, también definimos el objetivo general y los objetivos específicos, el alcance del trabajo y la metodología a utilizar.

1.1 Problema

El ser humano ha desarrollado la capacidad de expresar como se siente a través del lenguaje y la simbología, estos métodos a pesar de ser accesibles y fáciles de entender no poseen la carga emocional que puede tener una interacción física. En la actualidad tenemos una inmensa cantidad de información disponible gracias a los avances en tecnología de comunicación, como en el caso del Internet. A pesar de esto muchas personas prefieren ir a una casa de estudio y aprender de un tutor.

Este comportamiento proviene de la necesidad del ser humano de pertenecer a la comunidad [85], esta característica está presente en nuestra historia como especie mucho antes de la existencia del lenguaje.

Se dice que los monos primates están en una era equivalente a la era de piedra que una vez el ser humano vivió [86]. En esos tiempos la comunicación en su mayoría era no verbal, donde sonidos, gestos y expresiones faciales jugaban un papel fundamental para el buen desarrollo de la comunidad.

Teniendo en cuenta que el ser humano siempre ha hecho uso de expresiones faciales para comunicar algún mensaje, este no ha perfeccionado la habilidad para interpretar su significado.

En enero de 2016, la compañía americana Apple adquirió la empresa llamada Emotient, especialista en leer las emociones del ser humano a través del análisis de las expresiones faciales. En noviembre de ese mismo año Facebook compró la compañía FacioMetrics quienes desarrollan tecnología para el seguimiento del rostro y detección de emociones. No son solo Apple y Facebook,

muchas otras compañías que incluyendo Google y Microsoft están dedicadas a desarrollar tecnologías para el reconocimiento de emociones usando expresiones faciales, combinado con una variedad de señales psicológicas.

El mundo actual se ha dado cuenta de lo importante que es el reconocimiento de emociones, por lo que observamos un mercado que en el 2016 tuvo un valor estimado de 6.7 billones de dólares, y con proyecciones de hasta 67.1 billones para el 2021.

Muchas de estas empresas buscan mejorar la experiencia de usuario recolectando datos de este último. Debido a la diversidad de características faciales y condiciones ambientales (iluminación, rotación y posición) una solución única y genérica es prácticamente imposible usando técnicas y modelos actuales.

1.1.1 Justificación de solución

El mejor modelo para la detección de emociones está basado en redes neuronales [29], pero estas poseen debilidades que pueden ser explotadas por algún otro sistema o individuo, varias vulnerabilidades son descritas en una publicación de Su Jiawei, llamada “One pixel attack for fooling deep neural networks” [30] (Ataque de un pixel para burlar las redes neuronales profundas). El autor describe la fragilidad de este modelo y demuestra que en el 70.97% de los casos el cambiar un solo píxel afecta el resultado final de la red neuronal profunda.

El modelo de redes de cápsulas [59], posee el potencial de generar mejores resultados. “Yo pienso que la manera en la que estamos trabajando sobre visión artificial es errónea”, mencionó Hinton [75], y agregó “Actualmente funciona mejor que cualquier otra cosa pero eso no significa que sea lo correcto”, refiriéndose a redes neuronales convolucionales.

1.1.2 Estrategia de solución

Las redes de cápsulas tienen un proceso parecido a las redes convolucionales. Una red convolucional toma un conjunto de imágenes, luego las propaga dentro de una estructura secuencial de capas conformadas por neuronas

artificiales, estas son encargadas de interpretar la data y expresar un resultado acorde a los pesos obtenidos en la fase de entrenamiento. Las redes de cápsulas se diferencian por la sustitución de neuronas artificiales (número entero o flotante) por cápsulas (vector n-dimensional).

Otra diferencia es el uso de capas de Max Pooling que se encargan de encontrar el valor máximo entre una ventana de muestra y pasa este valor como resumen de características sobre esa área [87]. Como resultado, el tamaño de los datos se reduce por un factor igual al tamaño de la ventana de muestra sobre la cual se opera, lo que provoca según Hinton una pérdida datos relevantes [75]. Debido a la reducción de datos, los modelos de redes convoluciones logran un rendimiento óptimo. Las redes de cápsulas al no poseer dicha capa necesitan otra manera de discernir la relevancia entre los datos mientras se reduce su dimensionalidad, es por eso que usamos un proceso llamado Enrutado Dinámico. Este proceso consiste en colocar una matriz W*ij* (Figura 1.1) entre las capas *i* y *j*, donde cada entrada representa un peso/número que relacionan a dos cápsulas de capas distintas, el valor del peso define la posibilidad de transmitir datos entre la cápsula de la capa *i* y la cápsula de capa *j* [59].

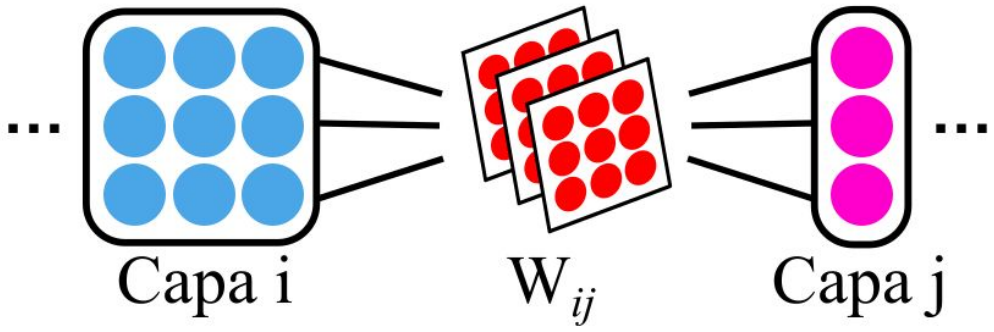


Figura 1.1 Estructura lógica de la matriz Wij en el proceso de enrutado dinámico.

1.2 Objetivos

A continuación se presentará el objetivo general y los objetivos específicos planificados para la solución propuesta.

1.2.1 Objetivo General

Desarrollar un software capaz de detectar las 6 expresiones faciales básicas (Felicidad, Tristeza, Sorpresa, Miedo, Disgusto, y Furia) basado en el modelo de cápsulas para demostrar su alcance en problemas de visión artificial.

1.2.2 Objetivos Específicos

- Estudiar el alcance de las redes de cápsulas.
- Seleccionar el conjunto de datos de entrenamiento y pruebas.
- Adaptar el modelo de cápsulas al problema de detección de emociones.
- Entrenar la red de cápsulas con el conjunto de datos seleccionados.
- Realizar las pruebas de validación.
- Comparar los resultados entre el modelo a desarrollar y una red convolucional profunda.

1.3 Alcance

El alcance del trabajo especial de grado consiste en generar un software capaz de detectar 6 emociones básicas universales (Felicidad, Tristeza, Sorpresa, Miedo, Disgusto, y Furia), el cual nos permita estudiar y generar estimaciones sobre la capacidad de las redes de cápsulas para la resolución de problemas en visión artificial.

1.4 Plataforma de desarrollo

La aplicación fue desarrollada usando un computador con las siguientes características:

- Intel i7 7700HQ CPU 2.80GHz, 4 Cores
- 16Gb de Ram
- 1Tb Disco duro (SATA) + 256Gb (SSD)
- Nvidia GeForce GTX 1070
- Windows 10

1.4.1 Lenguaje

El lenguaje de programación a usar es Python, este es un lenguaje de propósito general [63] interpretado de alto nivel, creado por Guido van Rossum, publicado en 1991. Python fue diseñado con la filosofía de facilitar la lectura del código, simplificando su sintaxis, lo que permite a los programadores expresar conceptos con pocas líneas de código [64]. También permite la construcción de programas de alto o bajo alcance [65].

Algunas de las características de Python son:

- Orientado a Objetos: La programación orientada a objetos está soportada en Python y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables.
- Funciones y librerías: Dispone de muchas funciones incorporadas en el propio lenguaje, para el tratamiento de strings, números, archivos, etc. Además, existen muchas librerías que podemos importar en los programas para tratar temas específicos como la programación de ventanas o sistemas en red o cosas tan interesantes como crear archivos comprimidos en .zip.
- Propósito general: Se pueden crear todo tipo de programas. No es un lenguaje creado especialmente para la web, aunque entre sus posibilidades también se encuentra el desarrollo de páginas.
- Multiplataforma: Hay versiones disponibles de Python en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.
- Interpretado: Quiere decir que no se debe compilar el código antes de su ejecución. En realidad se realiza una compilación, pero esta se realiza de manera transparente para el programador. En ciertos casos, cuando se ejecuta por primera vez un código, se producen unos bytecodes que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código.
- Interactivo: Python dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudarnos a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.

- Sintaxis clara: Por último, destacar que Python tiene una sintaxis muy visual, gracias a una notación indentada (con márgenes) de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave begin y end. Para separar las porciones de código en Python se debe tabular hacia dentro, colocando un margen al código que irá dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.

1.4.2 Herramientas

En esta sección se describen las librerías a usar para el desarrollo del modelo de detección de emociones.

1.4.2.1 Tensorflow

Tensorflow es una librería de código abierto para la computación numérica usando diagramas de flujo de datos [67]. Los nodos del grafo representan operaciones matemáticas, mientras que las aristas representan arreglos de datos multidimensionales llamados Tensores[1], los cuales definen el flujo entre operaciones. Esta arquitectura flexible permite desplegar soluciones computacionales a uno o más CPUs o GPUs en un computador de escritorio, servidor o dispositivo móvil sin necesidad de reescribir código. Tensorflow también incluye Tensorboard la misma es una herramienta de desarrollo usado para la visualización de datos.

Tensorflow fue desarrollado originalmente por investigadores e ingenieros del equipo de "Google Brain" dentro del departamento de investigación de inteligencia de máquinas de Google, con el propósito de conducir investigaciones en aprendizaje de máquina y redes neuronales profundas. El sistema es lo suficientemente genérico para ser usado en otros campos de investigación.

Tensorflow provee un API estable para Python y C, además de librerías en C++, Go, Java, Javascript y Swift, en las cuales no se garantiza la compatibilidad hacia atrás.

[1] Generalización de vectores y matrices n-dimensionales.

1.4.2.2 OpenCV

OpenCV posee una licencia de BSD y por lo tanto es libre para el uso académico y comercial. Tiene interfaces C ++, C, Python y Java y soporta Windows, Linux, Mac OS, iOS y Android. OpenCV fue diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real[89].

Escrito en C / C++ optimizado, la biblioteca puede aprovechar el procesamiento multi-core. Habilitado con OpenCL, puede aprovechar la aceleración de hardware de la plataforma de computación heterogénea subyacente. Adoptada en todo el mundo, OpenCV tiene más de 47 mil personas en su comunidad de usuarios y con un número estimado de descargas de más de 9 millones. Los usos van desde el arte interactivo hasta la inspección de minas, la costura de mapas en la web o la robótica avanzada.

1.4.2.3 NumPy

NumPy es una librería usada para computación científica dentro de Python[68]. Contiene entre otras cosas una variedad de funcionalidades fundamentales como:

- Una poderosa definición de objetos n-dimensionales.
- Funciones de “broadcasting” sofisticadas.
- Herramientas para integrar código en C / C++ y Fortran.
- Funciones útiles de algebra lineal, transformaciones de Fourier y capacidad de generación de números aleatorios.

Además de su capacidad obvia para uso científico, NumPy también puede ser usado como un contenedor genérico eficiente de datos multidimensionales. Es posible definir variables con tipos de datos arbitrarios. Esto permite a NumPy integrarse de forma sencilla y veraz a una gran variedad de bases de datos.

NumPy posee la licencia BSD, permitiendo su uso con algunas restricciones.

1.4.2.4 Matplotlib

Matplotlib es una librería de Python para el trazado de planos de calidad en 2D para formatos físicos y ambientes interactivos a través de distintas plataformas. Matplotlib puede ser usado en scripts de Python, consola de Python e IPython, en Jupyter notebook [90], servidores web de aplicaciones, y en herramientas de desarrollo para interfaces gráficas.

Matplotlib trata de mantener las cosas fáciles y las difíciles posibles. Se pueden generar planos, histogramas, espectros de poder, grafos de barras y errores, gráficos de dispersión, etc., con solo algunas líneas de código.

Para gráficos simples el módulo pyplot provee una interfaz parecida a la usada en MATLAB, particularmente cuando está combinado con IPython. Para el usuario experto, se tiene control total de los estilos de línea, propiedades de las fuentes, propiedades de eje, etc., a través de una interfaz orientada a objeto o a través de un conjunto de funciones familiares para usuarios de MATLAB.

1.4.2.5 Keras

Keras es un API de alto nivel para redes neuronales, escrito en Python y capaz de funcionar por encima de Tensorflow, CNTK, o Theano. Fue desarrollado con el enfoque de permitir experimentación rápida, permitiendo que el proceso de desarrollo de una idea sea posible con la menor cantidad de retrasos, lo cual es clave en el ámbito de investigación.

Se recomienda usar Keras si se necesita una librería de aprendizaje profundo que:

- Permita el uso de prototipaje rápido (a través de facilidad de uso, modularidad y extensibilidad).
- Soporte redes convolucionales y recurrentes, además de combinaciones de ambas.
- Ejecución sin interrupciones en CPU y GPU.

Keras es compatible con Python 2.7-3.6 y posee las siguientes características:

- Facilidad de uso: Keras es un API diseñado para humanos, no para máquinas. La experiencia de usuario es parte importante de su diseño. Keras sigue las mejores prácticas para reducir carga cognitiva en sus usuarios: ofrece un API consistente y simple, minimiza el número de acciones requeridas para casos comunes de uso, y provee respuestas claras y precisas en la ocurrencia de algún error de usuario.
- Modularidad: Un modelo es definido como una secuencia o un gráfico con módulos totalmente configurable que pueden ser conectados con la menor cantidad posible de restricciones. Particularmente, capas neuronales, funciones de costo, optimizadores, esquemas de inicialización, funciones de activación, esquemas de regularización son todos los módulos individuales posibles de combinarse para crear nuevos modelos.
- Fácil escalabilidad: Nuevos módulos son fáciles de agregar (como nuevas clases y funciones), y los módulos existentes poseen una amplia cantidad de ejemplos de uso. La creación de nuevos módulos permite la capacidad de usar cualquier tipo de expresión, haciendo Keras adecuado para investigaciones de nivel avanzado.
- Trabaja con Python: No existen archivos de configuración separados en formato declarativo. Los modelos son descritos en código de Python, el cual es compacto, fácil de debuggear, y permite facilidad en escalabilidad.

1.5 Metodología

Como metodología de desarrollo usaremos la metodología básica para la minería de datos[102] comprendida por las siguientes etapas:

- Recolección de los datos: Recolectamos los datos relevantes para el problema propuesto. Se refiere a los datos o las observaciones sobre las cuales se entrenan los modelos posteriores, y los datos sobre los cuales se realizará la validación de los mismos.
- Elección del algoritmo: Elegimos el algoritmo de aprendizaje que se encargará de aprender características y descubrir patrones en los datos recolectados.

- Elección de parámetros: Los algoritmos electos en el punto anterior probablemente tengan un conjunto de parámetros que influyan en su funcionamiento. Mediante el estudio y realización de experimentos, elegimos los parámetros adecuados para la resolución del problema.
- Entrenamiento: Se ajustan los algoritmos escogidos sobre los datos recolectados de entrenamiento para que sea capaz de identificar relaciones y características que puedan ser usados para discriminar entre observaciones y realizar predicciones.
- Evaluación: Evaluaremos las prestaciones de los algoritmos con datos distintos a aquellos utilizados para la etapa de entrenamiento (data de validación), para tener así un punto de referencia que nos otorgue una idea sobre su utilidad o su eficacia para el problema a tratar.

CAPÍTULO II. Marco teórico

Este capítulo introduce la teoría básica y necesaria para la comprensión de los siguientes capítulos. Se incluyen explicaciones sobre los temas: Imagen digital, aprendizaje de máquina, visión por computador, las redes neuronales, reconocimiento de rostros y emociones y, por último redes de cápsulas.

2.1 Imagen digital

Una imagen puede ser definida como una función bidimensional *f(x,y)*donde "*x"* y "*y"* son coordenadas en el plano, la amplitud de *f* en cualquier coordenada *(x,y)* es llamado intensidad o nivel de gris de la imagen en ese punto. Cuando *x*, *y* y los valores de intensidad de *f* son finitos, con cantidades discretas, dicha imagen se denomina imagen digital [1].

Las imágenes digitales se clasifican de la siguiente manera:

2.1.1 Imágenes vectoriales

Es un conjunto de coordenadas 2D conectadas usando líneas y curvas, para formar polígonos y otras formas geométricas. Su mayor ventaja es la facilidad de realizar transformaciones (traslación, rotación, escalamiento) a cada objeto dibujado sin sufrir pérdida de datos [2] [3] (Figura 2.1).

2.1.2 Imágenes rasterizadas

Las imágenes rasterizadas también llamadas bitmap [4], son gráficos guardados y renderizados como un arreglo de píxeles [3]. Este método de guardado posee menor complejidad que las imágenes vectoriales, lo que provoca que el renderizado sea rápido, pero con un peso en memoria mayor que un conjunto de polígonos [2].

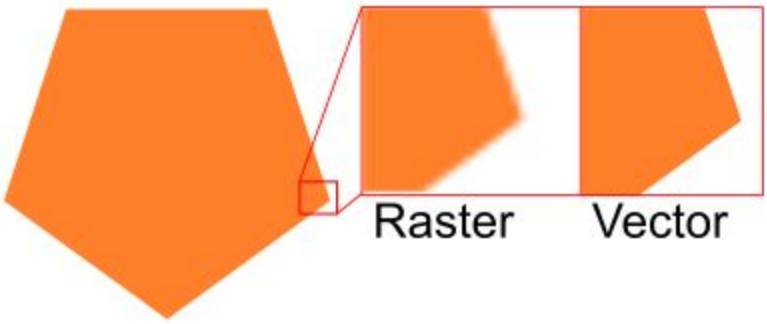


Figura 2.1 Comparación de renderizado entre imágenes raster y basadas en vectores, zoom de x10 en el área seleccionada.

2.1.3 Procesamiento digital de imágenes

El procesamiento digital de imágenes es el conjunto de técnicas donde sus entradas y salidas son imágenes y engloba procesos que extraen o agregan atributos a las imágenes [1]. Los procesos pueden tener como finalidad mejorar, restaurar, segmentar, describir, reconocer, reconstruir o transformar las características de una o varias imágenes (Figura 2.2) [5].

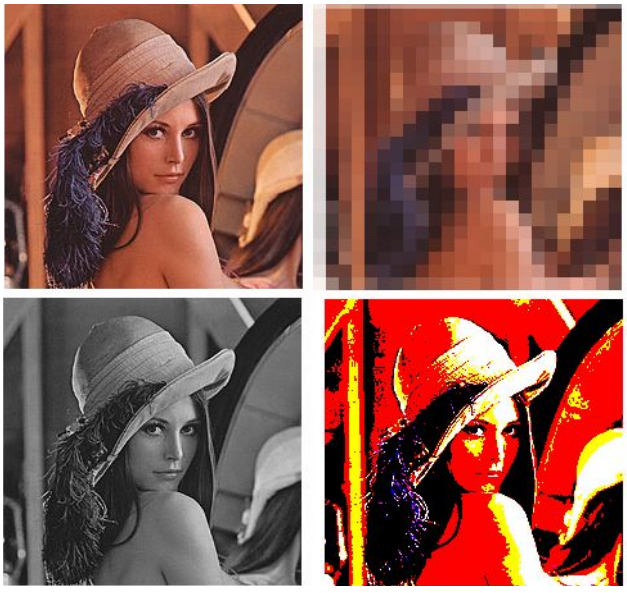


Figura 2.2 Colección de filtros aplicados a la imagen superior izquierda.

2.2 Aprendizaje de máquinas

El aprendizaje de máquinas o machine learning, es una rama derivada de inteligencia artificial que consiste en el desarrollo de algoritmos de autoaprendizaje que ganan conocimientos a partir de una base de datos para así realizar predicciones en datos futuros [6]. En vez requerir personas que encuentren reglas y construyan modelos que analizan grandes cantidades de datos, el aprendizaje de máquinas ofrece una alternativa eficiente para encontrar conocimiento. Este busca encontrar patrones en los datos los cuales gradualmente mejoran el rendimiento de los modelos predictivos y provee mejores decisiones.

2.2.1 Esquema de una aplicación de aprendizaje de máquina

En general las aplicaciones de aprendizaje de máquina sigue 5 pasos secuenciales [22]:

- Recolectar datos.
- Preparación de datos.
- Análisis de datos.
- Entrenar el modelo.
- Fase de prueba.

2.2.1.1 Recolectar datos

Estos datos son los que serán usados para el entrenamiento del modelo predictivo. Pueden ser obtenidos a partir del minado de datos en la web, base de datos, a través de un API o cualquier lugar con datos que puedan ser manejados por un computador.

2.2.1.2 Preparación de datos

Una vez que se tienen los datos es necesario realizar un preprocesamiento que coloca los datos en un formato legible para el algoritmo de aprendizaje. Este proceso también elimina valores vacíos y ordena la entrada resultante.

2.2.1.3 Análisis de datos

Este paso es ejecutado por el programador, donde usando los datos preprocesados se realiza un análisis para determinar si los datos a usar durante el

entrenamiento son confiables. Se dibujan gráficos comparativos para determinar patrones y estimar el resultado del modelo predictivo. El resultado obtenido sirve de guía para estimar la precisión del modelo [32].

2.2.1.4 Entrenar el modelo

Luego comenzamos la fase donde el algoritmo extrae conocimiento e información de los datos. Toda la información obtenida es guardada en un formato que solo el modelo puede entender. Dedicamos una sección a este paso más adelante.

2.2.1.5 Fase de prueba

Finalmente se prueba el modelo comparando los resultados con los datos obtenidos durante el análisis de datos. Comúnmente se tiene un set datos distinto al usado durante el entrenamiento para realizar las pruebas predictivas [31]. Si este test muestra resultados incongruentes con la verdad, se repiten los dos primeros pasos en busca de los fallos en la data. Una vez que se encuentra un margen de error aceptable entre la verdad y la predicción resultante, concluye el desarrollo del modelo.

2.3 Proceso de aprendizaje

La fase de aprendizaje en un algoritmo de aprendizaje de máquina es la parte más importante del proceso. Es aquí donde el modelo aprenderá patrones y relaciones usadas para realizar una o varias predicciones. Fundamentalmente se puede dividir en 2 tipos:

2.3.1 Supervisado

También llamado aprendizaje clasificatorio, se refiere a un proceso de aprendizaje donde la data de entrenamiento viene clasificada de antemano, permitiendo que el algoritmo conozca qué etiqueta(s) debe usar al momento de hacer una predicción [7].

En el aprendizaje supervisado se usa una algoritmo que mapea las variables de entrada x con la variable de salida Y. $f(x)=Y$. El algoritmo requiere de un set de datos el cual actúa como profesor que supervisa el proceso de

aprendizaje. Se asume que los datos de entrada son verdaderos; este realiza predicciones de manera iterativa y cada predicción es evaluada y corregida por el profesor [77].

2.3.2 Reforzado

El aprendizaje reforzado es una subclase del aprendizaje supervisado el cual se refiere a un sistema (agente) que mejora su rendimiento basado en las interacciones con el ambiente o en el contexto donde se encuentre. Como la información acerca del ambiente se puede ver como un tipo de señal que dispara una recompensa, es fácil confundirla con aprendizaje supervisado. Sin embargo, en aprendizaje reforzado esa recompensa no representa el valor o clase verdadera, este valor se refiere a una medida basada en que tan cerca se está de la solución (figura 2.3). A través de la interacción con el ambiente un agente puede usar el aprendizaje reforzado para aprender una serie de acciones que maximizan las recompensas obtenidas realizando pruebas y evaluando los resultados [6].

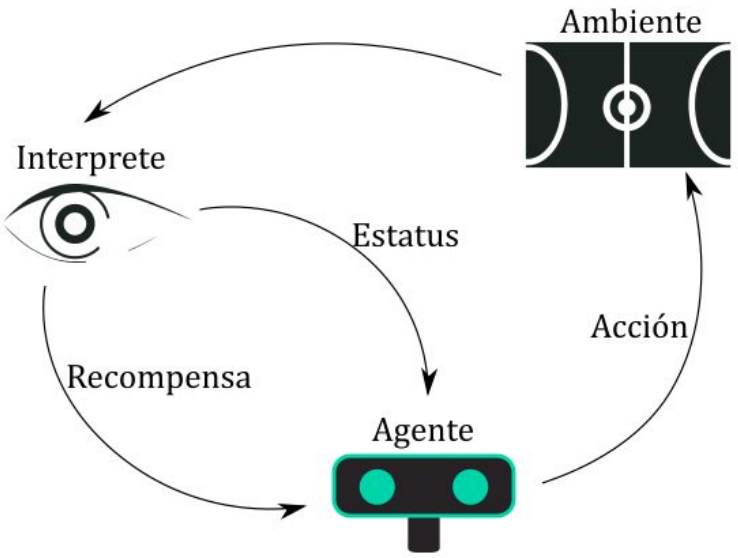


Figura 2.3 Un agente toma una acción en un ambiente, el cual es interpretado en una recompensa y una representación del ambiente, los cuales sirven de entrada para el agente durante el entrenamiento.

2.3.3 No supervisado

Cuando los datos de entrenamiento no poseen etiquetas o ni se encuentran con una estructura desconocida, debemos usar una técnica que permita el aprendizaje sin necesidad de recibir ningún tipo de feedback. El aprendizaje no supervisado se refiere a la adquisición de conocimiento usando un conjunto de datos sin clasificar[6].

El objetivo de este método es modelar la distribución o estructura de la data de entrenamiento. No existe un "profesor" que sirva de guia de aprendizaje, el algoritmo se encarga de descubrir patrones en los datos por si solo [77].

2.4 Aprendizaje profundo

Deep Learning o aprendizaje profundo es una clase de algoritmos derivados de la rama de aprendizaje de máquina la cual consiste en el aprendizaje de características a partir de técnicas de extracción de patrones o clasificación de datos, con el objetivo de extraer características, realizar transformaciones, análisis de patrones y clasificar datos [33].

Este modelo se distingue de los demás por el uso de neuronas artificiales, estas neuronas se encargan de recibir información, realizar operaciones que alteren su estado interno y enviar un resultado a una neurona de una capa de mayor nivel [34].

Deep learning es el modelo más usado en la actualidad para resolver problemas de aprendizaje de máquina. A través de los años se ha visto la facilidad que tiene para descubrir patrones y estructuras en datos de grandes dimensiones. Este modelo no solo es usado para resolver problemas de ciencias de la computación, también es útil en otras ciencias de investigación, negocios y gobiernos [35].

Adicionalmente es capaz de resolver problemas de reconocimiento de imágenes [37] y voz [36] mejor que cualquier otro modelo de aprendizaje. En algunos casos es capaz de superar o confundir a los humanos al crear modelos que

le permiten interpretar y responder usando el mismo lenguaje de comunicación que los seres humanos [38].

2.4.1 Red neuronal profunda

Una red neuronal profunda es un conjunto de capas, una detrás de otra, donde cada una de ellas está compuesta por una cantidad variante de neuronas artificiales (figura 2.4) [39]. No todas las neuronas entre cada capa están conectadas.

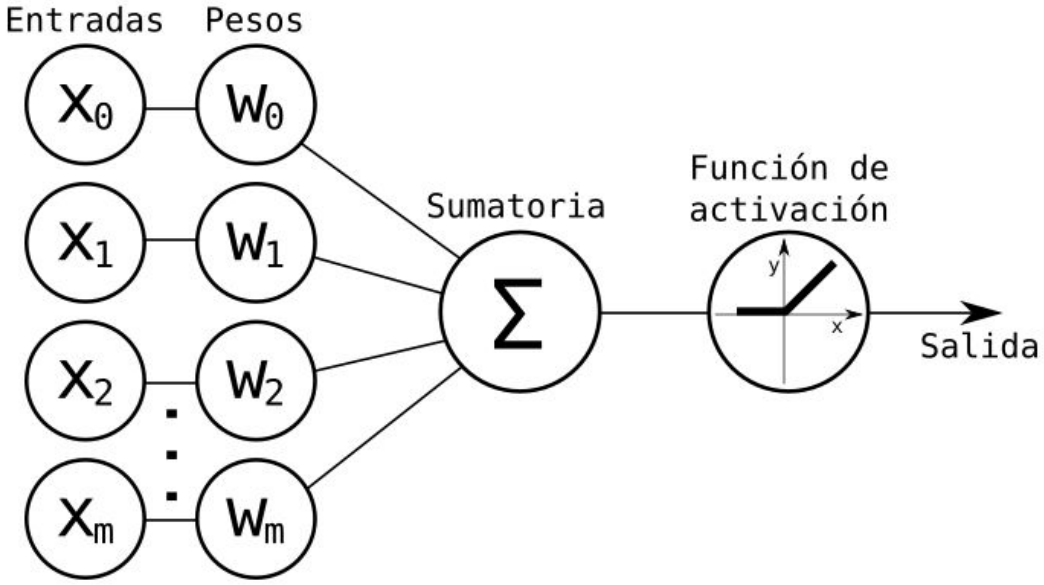


Figura 2.4 Estructura de una neurona artificial [41].

Una neurona recibe *m* escalares de entrada *x*, cada uno es asociado y multiplicado por un peso *w*. La sumatoria de todos los pesos es igual a 1, por lo tanto:

$$y = \sum_{i}^{m} x_i w_i \,,\ 0 \le i \le m \,,\ 0 \le x_i \le 1$$

Ecuación 2.1 Cálculo del valor de entrada para la función de activación.

El resultado de la sumatoria es la entrada de la función de activación *f(y)=z*, está retorna un valor *z* que representa la predicción hecha por la neurona.

La primera capa de la red recibe los datos a evaluar, y se encarga de convertir la entrada en un formato legible para la siguiente capa en la red. Las demás capas (exceptuando la última) son llamadas capas escondidas (hidden layers), cada una de ellas aplica alguna operación de activación que determina la probabilidad de que dicho punto en los datos corresponda con una predicción en el modelo. Luego se aplica max pooling, el cual toma la información de un conjunto de neuronas y retorna el mayor valor predictivo. El proceso se repite en cada capa escondida, hasta llegar a la última capa. La última capa se encarga de traducir los datos recibidos en una predicción según el problema a estudiar [40] (figura 2.5).

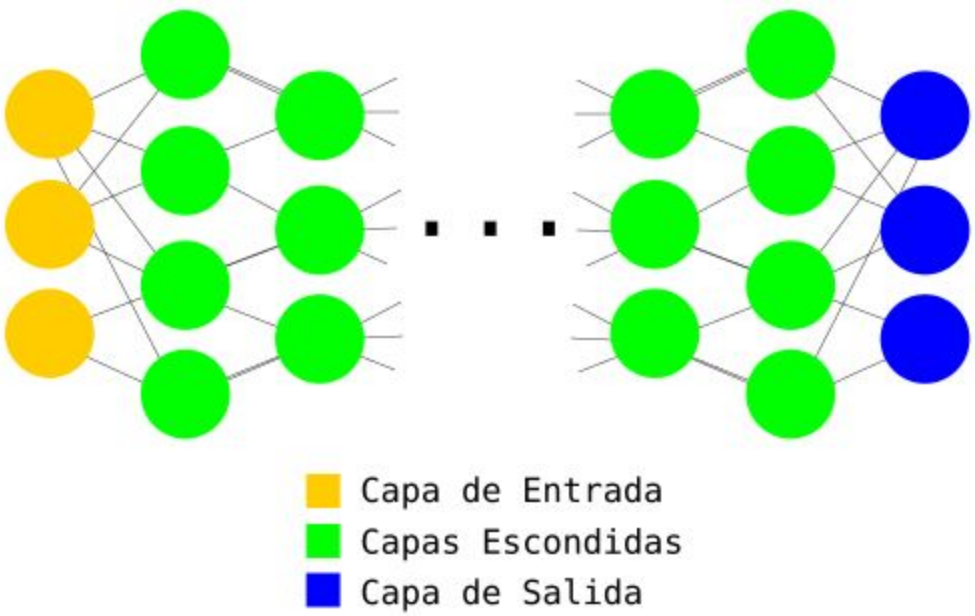


Figura 2.5 Arquitectura de red profunda.

Este tipo de redes es llamada feedforward network [41] (red neuronal prealimentada), donde toda la información fluye desde la capa inicial a la final. Durante el aprendizaje (supervisado o no) es importante alterar la importancia de la neuronas que tienen poca o mucha contribución al resultado final, esto se logra con el uso de backpropagation (propagación hacia atrás).

Cada neurona posee un escalar asociado con su resultado que representa el peso o importancia de su decisión. Backpropagation se encarga de analizar el error entre cada capa y actualizar el peso de cada neurona [42]. El proceso de minimizar el error Δw se llama método de descenso de gradiente (figura 2.6).

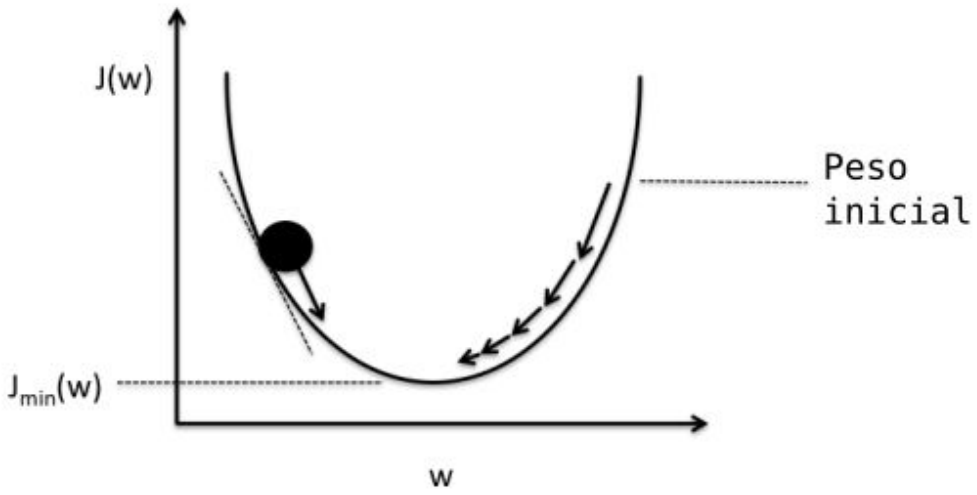


Figura 2.6 Proceso de gradiente (gradient descent).

El siguiente pseudocódigo se encarga de realizar propagación hacia atrás usando como peso inicial valores aleatorios:

inicializar pesos (lista de valores aleatorios para cada peso)
  do
    foreach ejemplo de entrenamiento in ex
      prediccion = salida-red-neuronal(network, ex) //Forward pass
      actual = salida-aprendizaje(ex)
      calcular error en las unidades de salida de la capa actual
      for pesos p_i, p_j *entre capa l y ultima capa:* $\Delta w_k \leftarrow \sum_{i=0}^{n} \sum_{j=0}^{m} (p_i - p_j)$

      for pesos p_q, p_i *entre primera capa y capa l:* $\Delta w_r \leftarrow \sum_{q=0}^{n} \sum_{i=0}^{m} (p_q - p_i)$

      actualizar pesos en network //Excepto los de la primera capa
  until todos los ejemplos han sido clasificados correctamente
  return network

2.4.2 Red neuronal convolucional

Es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico [78].

Inspiradas por el trabajo de Hubel y Wiesel [79], sobre la corteza visual en los gatos. Sabemos que la corteza visual contiene un arreglo complejo de células. Estas son sensibles a subregiones del campo visual, llamado campo receptivo. Las subregiones cubren todo el campo visual. Las células actúan como filtros sobre el espacio de entrada, los cuales buscan establecer relaciones en el espacio entre cada subregión [80].

Funcionan igual que las redes neuronales profundas, pero reciben como entrada una imagen digital. Otra diferencia es la estructura de cada capa. Existen 4 variaciones que son usadas en conjunto (figura 2.7) [81], se describe cada una en su respectivo orden:

- Capa de preprocesamiento: Esta es una capa opcional de filtros predefinidos que se mantienen fijos durante el entrenamiento. Esto implica que información adicional a la contenida en la imagen de entrada puede ser transmitida al modelo, por ejemplo bordes y gradientes.
- Capa de convolución: Se encarga de traducir los datos en la imagen en sinapsis entre neuronas. Cada capa de este tipo contiene *M* mapas de tamaño (*Mx*, *My*), un kérnel de tamaño (*Kx*, *Ky*) el cual es aplicado a distintas regiones de importancia en la imagen, y adicionalmente un factor de salto en la forma (*Sx*, *Sy*) que define cuántos píxeles saltara el kérnel en cada una de sus evaluaciones. El tamaño del mapa de salida viene dado por:

$$M_x^n = \frac{M_x^{n-1} - K_x^n}{S_x^n + 1} + 1 \, , \, M_y^n = \frac{M_y^{n-1} - K_y^n}{S_y^n + 1} + 1$$

Ecuación 2.2 Tamaño de mapa convolucional.

Donde *x*, *y* son las coordenadas en el plano 2D y *n* indica la capa actual. Cada mapa en la capa *n* está conectado a una máximo de *Mn-1* mapas en la capa *n-1*. Las neuronas de cada mapa comparten sus escalares de peso pero poseen distintos sensores receptivos, es decir, las entradas capaces de activar cada conjunto de neuronas varían entre ellas.

- Capa de submuestreo: Esta capa se encarga de reducir y normalizar los datos obtenidos en la capa de convolución. Se toma el promedio de la suma de los valores luego de ser filtrados por un kérnel (*Kx*, *Ky*). Un estudio realizado por Scherer [82] encontró que si en vez de tomar el promedio se toma el máximo, este suele dar mejores resultados de generalización de parámetros y además de converger rápidamente, este proceso se denomina max-pooling.
- Capa de clasificación: Cada neurona de la capa recibe un escalar de cada operación de submuestreo/max-pooling, cada clase resultante está asociada a un conjunto de neuronas en esta capa. Las neuronas que logran activarse otorgan un peso predictivo a la clase que corresponden.

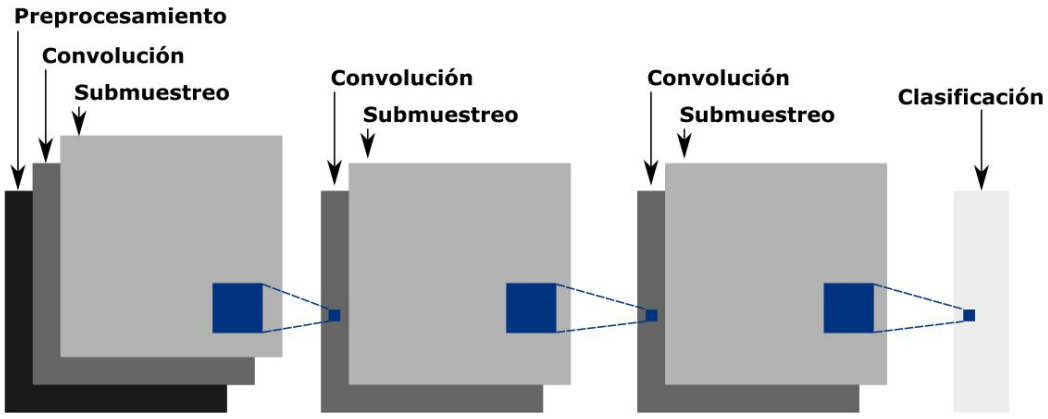


Figura 2.7 Arquitectura de una red neuronal convolucional. El primer conjunto de capas representa la capa de entrada, la capa de clasificación la capa de salida y el resto son conjuntos de capas escondidas.

2.5 Detección de caras

Antes de analizar las emociones de una persona se debe conocer las características del rostro humano. En general, un ser humano detecta una cara analizando atributos como la forma, posición relativa de los ojos, cejas, nariz y boca [43]. Los algoritmos que implementan aprendizaje son capaces de replicar ese comportamiento aprendiendo las características y atributos que componen una cara, generando así o uno o varios modelos predictivos que dan como resultado un valor probabilístico, el cual en el mejor de los casos, se aproxima a la respuesta correcta.

Fue hacia los años 70 cuando aparecieron los primeros algoritmos, basados en técnicas heurísticas y antropométricas, pero no eran eficaces ya que fallaban bastante y eran muy sensibles a cambios ambientales. La investigación se dejó de lado porque todavía no tenía utilidad. Siguiendo la investigación en los años 90 cuando, gracias al desarrollo tecnológico y al descubrimiento de algoritmos útiles en visión artificial, se reanudó la investigación [44].

2.5.1 Problemas

La extracción de características que forman una cara, incluyendo su forma y todos las partes que contiene (ojos, cejas, nariz, boca, oídos) en cualquier imagen es un proceso complicado [12]. Todas las caras humanas son similares y por lo tanto poseen características difíciles de diferenciar comparado con otras medidas biométricas (huella dactilar, iris)[14].

El problema es detectar un objeto tridimensional en un ambiente 2D, esto hace que los resultados sean susceptibles a los efectos de iluminación y la rotación del rostro, problemas que persisten en todas las técnicas conocidas[45].

2.5.1.1 Iluminación

En la figura 2.8 la misma cara es afectada por distintos cambios en la luz, en algunos casos la cara parece distinta dependiendo de la iluminación lo cual representa un problema no solo para un ser humano [15], los modelos actuales para detección de rostros también sufren de este problema.

Figura 2.8 Ejemplo donde la misma cara es afectada por distintos niveles de iluminación.

Debido a las dificultades para controlar las condiciones de la luz en ambientes reales, la variedad de efectos que puede causar la iluminación aumenta la complejidad del problema [16].

2.5.1.2 Postura

Variaciones en la postura ocurren debido a los cambios tridimensionales en la orientación y posición de la cara además de la inclusión de expresiones faciales[17].

Figura 2.9 Combinación del problema de la postura e iluminación.

La postura representa una gran cantidad de información que por desgracia dificulta y degrada el rendimiento del sistema [46]. Este problema es ilustrado en la Figura 2.9 donde la misma cara se muestra en distintas poses lo que ocasiona una distorsión al momento de identificar la cara.

2.5.2 Técnicas de adquisición de caras

Se toma una imagen cualquiera como entrada, luego se realiza un preprocesamiento en la imagen para mejorar la calidad y reducir el ruido que se encuentre [47]. En el siguiente paso, se segmenta la imagen en los lugares donde existe una cara usando alguno de los siguientes métodos [48]:

- Basados en conocimiento.
- Basados en características invariantes.
- Basados en plantillas.
- Basados en apariencia.

2.5.2.1 Métodos basados en conocimiento

Basados en la geometría y disposición de las características faciales [18]. Estos métodos describen la forma, tamaño y textura de una cara. Algunos otros describen otras características faciales como las cejas, ojos, nariz y quijada [19]. El mayor problema de este método (figura 2.10) son las distintas poses (cambios en rotación y posición) de la cara.

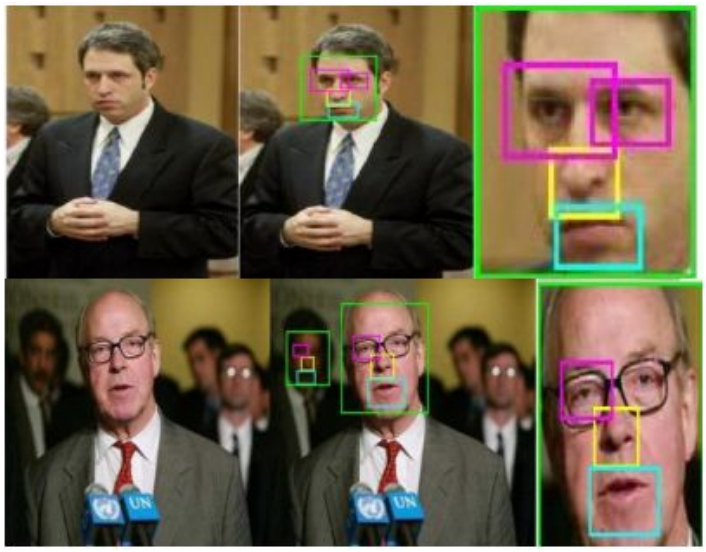


Figura 2.10 Resultados de algoritmo de segmentación de características faciales.

2.5.2.2 Métodos basados en características invariantes

El objetivo principal de este método es encontrar características estructurales de caras humanas en ambientes donde la luz no está controlada. Características como disposición de la cara, color, forma y textura son usadas para encontrar una segmentación válida [20, 21]. Estos métodos son sensibles a la iluminación, presencia de regiones de igual color que la piel humana y a caras adyacentes. Un ejemplo de estos problemas se muestra en la figura 2.11.

Figura 2.11 (a) Imagen original, (b) Imagen binaria entre color de piel (blanco) y fondo (negro), (c) Áreas marcadas con color de piel, (d) Imagen verificada usando proporción de alto y ancho [20].

2.5.2.3 Métodos basados en plantillas

Los métodos basados en plantillas son sensibles a la postura, escala y las variaciones en la forma del cuerpo humano. Existen métodos que usan plantillas deformables que mejoran los resultados frente a las variaciones de postura, escala y forma (figura 2.12). Estos métodos incluyen información acerca de la forma y la intensidad en la cual cada parte se deforma (vectores de dirección y su magnitud).

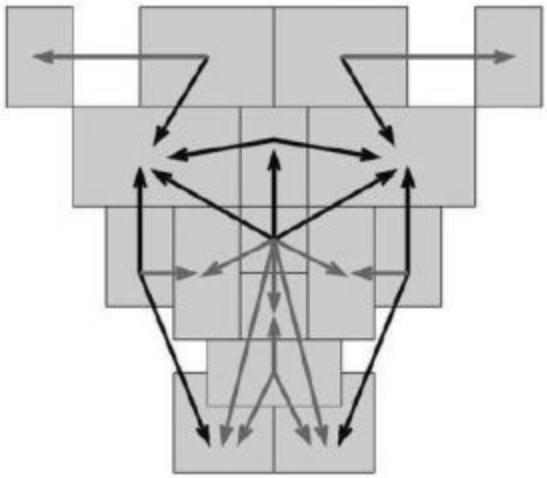


Figura 2.12 Plantilla de una cara con proporción de 14x16 [49].

2.5.2.4 Métodos basados en apariencia

Los métodos basados en apariencia buscan identificar información acorde a las heurísticas establecidas por un programador en el caso de eigenface (valor característico de cara) y análisis del discriminante lineal [50]. También es posible por medio de características aprendidas dado un conjunto de datos, como en el caso de redes neuronales (Figura 2.13) y SVM (máquinas de soporte vectorial) [51]. Por último el modelo oculto de Markov puede reconocer las características de un rostro por medio de probabilidades asociadas al patrón de aparición de cada característica encontrada [52].

En cualquiera de estos métodos la imagen es escaneada y se identifican regiones que poseen caras.

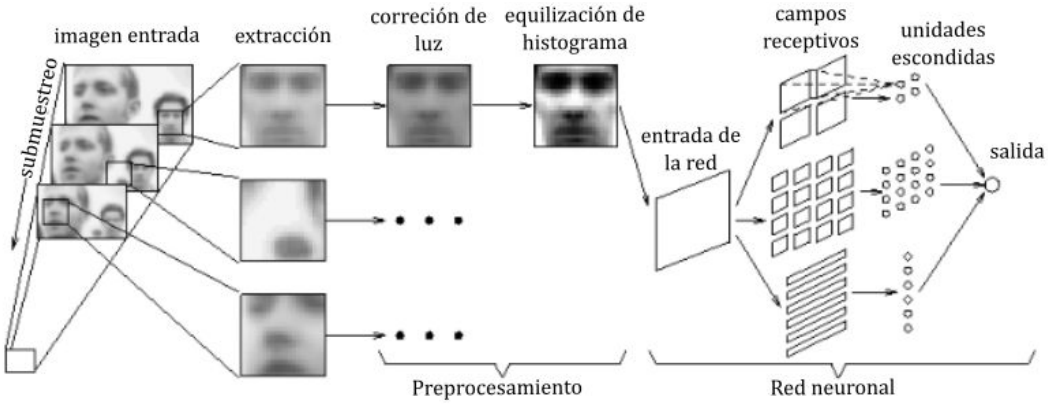


Figura 2.13 Ejemplo de proceso de extracción de caras usando redes neuronales.

2.6 Detección de emociones

La expresiones faciales representan el tipo de información necesaria para describir las emociones humanas. Podemos encontrar distintas expresiones a lo largo del día de cualquier ser humano, lo cual puede representar el estado mental o físico de la persona. Aunque los seres humanos poseen una gran cantidad maneras de expresarse emocionalmente, la psicología moderna define 6 expresiones faciales básicas: Felicidad, Tristeza, Sorpresa, Miedo, Disgusto, y Furia como emociones universales [26].

Una persona puede reconocer emociones rápidamente con un mínimo esfuerzo mental, pero el mismo proceso hecho por máquinas es un problema complejo. A continuación se describen diversas técnicas y estudios para reconocer expresiones[23]:

2.6.1 Basadas en probabilidad de movimiento

Con cada expresión facial la curvatura de la cara y las propiedades de sus elementos como las cejas, nariz, boca cambian. Su representación en píxeles (Imagen 2D) también cambia. Este método usa parámetros estadísticos que procesan estos cambios usando una red neuronal y obtiene como resultado un conjunto de vectores, cada uno asociado a una característica de la cara [53].

Ventaja:

- El proceso de reconocimiento es menor de 2 segundos para imágenes con caras de frente.

Desventaja:

- Poca tolerancia cambios en la rotación de la cara.

2.6.2 Corrección automática de los efectos de la luz

Las expresiones faciales se determinan usando puntos con distancias relativas entre ellos, llamados Unidad de Acción (AU) [58]. Se reconoce la cara usando la información de la textura y el color extraída. Luego se mapean los ojos y la boca usando reconocimiento de patrones. Los píxeles clasificados como piel son separados del fondo usando el método de Cascada de Haar.

Ventajas:

- Posible detección varias caras al mismo tiempo.
- El sistema de extracción de color permite que la cara sea extraída en ambientes donde la luz genera distorsiones.

Desventajas:

- 60% de precisión [58].
- No reconoce caras en ambientes donde la luz es escasa (mayor parte de la cara cubierta en sombras).

2.6.3 Identificación del usuario para el sistema Social Robot

Este sistema busca entrenar un robot capaz de reconocer rostros y emociones. Primero se entrena el modelo con imágenes del usuario y se clasifican emociones usando un método híbrido entre plantillas y apariencia [24]. Un seguidor de caras es usando para su detección. La información sobre la textura consiste en una serie de vectores que describen un modelo tridimensional de la cara.

Ventajas:

- La identificación y recolección de datos previos específicos del sujeto mejora el rendimiento del modelo de clasificación.

- Precisión del 82% incluyendo ambientes con condiciones varias de luz, diferentes posiciones y orientaciones de la cara.

Desventajas:

- Requiere ser entrenado con imágenes del rostro de la persona la cual posteriormente usará el sistema, es decir, es un sistema personalizado.
- La plantilla usada debe poder cubrir todas las características de la cara.

2.6.4 Basado en heurísticas con aprendizaje débil

Busca detectar rostros usando SVM[2] para luego reconocer emociones usando Adaboost. El algoritmo usado es llamado Adaboost (Adaptive Boosting) el cual tiene un aprendizaje débil[3], es decir, las características importantes son las que se refinan durante el entrenamiento[54].

$$F(x) = a_1 f_1(x) + a_2 f_2(x) + a_3 f_3(x) + \ldots + a_n f_n(x)$$

Ecuación 2.3 Fórmula de aprendizaje usando clasificadores débiles.

En la Ecuación 2.3 observamos la operación de aprendizaje Adaboost donde cada *fn(x)* se refiere a una SVM y su *an* asociado es un peso que representa la debilidad prevista de cada máquina de soporte, el valor *n* representa la cantidad de clasificadores débiles o SVM.

Este estudio no busca clasificar emociones, más bien usar la información obtenida en distintas expresiones faciales y modelar un sistema que sea tolerante a los cambios de la cara [55].

Ventajas:

- Usa reconocimiento de emociones para mejorar la detección de rostros.
- El uso de lentes no afecta el reconocimiento de emociones.

Desventajas:

- La distancia entre la cámara y el sujeto a estudiar afecta negativamente el reconocimiento de emociones.

---

[2] SVM: Support Vector Machine o en espanol Máquina de soporte vectorial.

[3] Se define un modelo como débil cuando esté aprende características que no son suficientes por sí solas para la resolución del problema.

- Obtiene resultados óptimos en ambientes donde la postura y la iluminación están controladas.

2.6.5 Sistema análisis cognitivo de rostros para televisores

En 2009 Kwang y Myung [56] crearon un sistema que reconoce rostros, los identifica y detecta qué emociones están siendo expresadas por los individuos. El algoritmo propuesto: Ada-LDA (Adaptative Linear Discriminant Analysis) usa información sobre la textura para dividir la imagen en subregiones que luego un clasificador débil separa en clases.

El análisis lineal del discriminante es usado para definir las clases obtenidas con un mayor nivel de independencia entre ellas.

Ventajas:

- Busca reconocer emociones en tiempo real.
- Puede procesar imágenes a 15 frames por segundo sin retraso.

Desventaja:

- Altamente dependiente del set de datos usado en el entrenamiento.

2.6.6 Detección de movimiento en expresiones faciales usando OpticalFlow

Primero, se usa una cámara infrarroja que ayuda a detectar las pupilas del sujeto, luego la cara es dividida en 6 segmentos simétricos, del mismo tamaño a lo largo de la cara. Estos segmentos son adaptados a un conjunto de vectores que representan la dirección que cada segmento puede moverse (todo segmento contiene su propio conjunto de vectores). Por último se analiza el peso de cada vector con el segmento para obtener el movimiento facial y retornar la emoción correspondiente [57].

Ventajas:

- Se requiere un mínimo de 3 imágenes por rostro para detectar expresiones faciales.
- No es necesario determinar la posición exacta de cada característica del rostro.

Desventaja:

- Requiere de una cámara infrarroja para posicionar la cara en el modelo.

2.7 Sistema de reconocimiento facial

Todos los sistemas de detección de expresiones faciales poseen un proceso similar, el cual se puede generalizar en 5 pasos [23]:

2.7.1 Obtención de imagen

Una imagen o una secuencia de ellas son usadas para el reconocimiento de expresiones faciales. Mientras que las imágenes 2D en escala de grises son comúnmente usadas en este proceso encontramos las posibilidad de adquirir imágenes a color (RGB[4]) las cuales pueden otorgar mayor información, por ejemplo: sonrojado, palidez. Estas imágenes son adquiridas a través de dispositivos digitales como cámaras o teléfonos inteligentes.

2.7.2 Preprocesamiento de imagen

Está etapa busca mejorar la calidad de la imagen de entrada y localiza las zonas de interés removiendo el ruido y suavizando la imagen. Elimina la redundancia en la imagen sin afectar detalles en la misma (figura 2.14). El preprocesamiento también incluye el filtrado y la normalización de la imagen el cual produce una imagen con el tamaño y rotación deseada.

[4] Red Green Blue siglas RGB, refieren a los colores básicos (rojo, verde y azul respectivamente) con los que se puede crear una imagen a color

Figura 2.14 Preprocesamiento de las imágenes a la izquierda, las siguientes fotos eliminan ruido y realzan las características de la cara.

2.7.3 Segmentación

Este paso divide la imagen en partes relevantes, es decir, se usa un método que divide la imagen en partes homogéneas, que corresponden a diferentes objetos en la imagen basados en la textura, borde y/o intensidad del color.

2.7.4 Extracción de características

Se extrae la información de interés de la imagen. Esto incluye características como forma, color y textura, y algunas otras dependiendo del método usado, por ejemplo: pareo de plantilla (pattern matching), dirección de la luz, histograma, movimiento.

Comparado con la imagen inicial, este proceso reduce y sistematiza la información obtenida, lo cual optimiza el rendimiento del algoritmo y reduce la cantidad de información guardada.

2.7.5 Clasificación

Con la información obtenida en el paso anterior, el modelo identifica los rostros encontrados con las características de cada emoción y se agrupa en la clase correspondiente. La cantidad de información a procesar aumenta la complejidad de la clasificación, es decir, a mayor información el pareo con la clase correspondiente se dificulta.

2.8 Red de cápsulas

Las redes cápsulas son un modelo propuesto por Hinton en su investigación “Dynamic Routing Between Capsules” [59]. Este busca resolver los problemas que poseen las redes convolucionales modernas simulando como el ser humano detecta objetos en un ambiente 3D. Para Hinton el cerebro realiza una operación inversa de renderizado [60], es decir, a partir de la información visual recibida, se descompone en una representación jerárquica del mundo que nos rodea y se realiza un pareamiento usando patrones aprendidos y relaciones

guardadas en el cerebro. La idea clave de este método es la representación de cualquier objeto no dependiente del ángulo el cual sea visto.

2.8.1 Cápsula

Una cápsula es un grupo de neuronas donde sus salidas representan diferentes propiedades de la misma entidad [61]. Cada cápsula recibe un vector de actividad como entrada, este vector representa la presencia de algún tipo de entidad u objeto. Su resultado también es representado como un vector de actividad. El vector posee información que nos dice la probabilidad que existe una clase y que parámetros posee la instancia.

En la tabla 2.1 comparamos la estructura de una cápsula con una neurona artificial (unidad mínima en una red neuronal convolucional), esta última recibe como entrada un escalar, luego es multiplicada por otro escalar que representa el peso, todos los resultados son usados en una sumatoria cuyo resultado en conjunto con el escalar original sirven para el uso de alguna de las funciones de activación no lineales (ejemplo: Softmax). El escalar obtenido servirá de entrada para las neuronas de la siguiente capa en el modelo.

		Cápsula	Neurona Artificial
Entrada de una cápsula/neurona en un nivel inferior		vector(u_i)	escalar(x_i)
Operación	Transformación afín	$\widehat{u}_{j\|i} = W_{ij} u_i$	---
	Sumatoria de pesos	$S_j = \sum_i c_{ij} \widehat{u}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Función de activación no lineal	$v_j = \frac{\|\|S_j\|\|^2}{1 + \|\|S_j\|\|^2} \frac{S_j}{\|\|S_j\|\|^2}$	$h_j = f(a_j)$
Salida		vector(v_j)	escalar(h_j)

Tabla 2.1 Diferencias entre cápsulas y neuronas artificiales.

En resumen, el proceso de una neurona artificial se puede describir en 3 pasos:

- Multiplicación del peso con el escalar de entrada.
- Sumatoria de los escalares resultantes.
- Pase de resultados a funciones de activación.

Por otra parte, las cápsulas tienen un proceso similar que usa vectores en vez de escalares, donde además se agrega un paso adicional [62], estos son:

- Multiplicación de matrices para los vectores de entrada.
- Multiplicación de pesos escalares con los vectores de activación.
- Sumatoria de vectores.
- Squashing.

Explicamos cada uno de estos pasos a continuación.

2.8.1.1 Multiplicación de matrices para los vectores de entrada

Los vectores de entrada que la cápsula recibe vienen directamente del resultado del preprocesamiento de la imagen o de las capas de cápsulas de niveles inferiores. La magnitud de estos vectores corresponde a la probabilidad de existencia del objeto correspondiente y la dirección del mismo describe el estado interno de los objetos detectados. Estos vectores son multiplicados por matrices de peso *W* que poseen información del espacio y relaciones entre características de niveles anteriores.

Por ejemplo: se quiere detectar que una imagen cualquiera posee una cara con nariz, entonces tenemos una cápsula *A* que representa la nariz, luego una matriz *Wa* que contiene la relación espacial entre nariz y el rostro, si la magnitud del vector de entrada de la cápsula *A* resulta en un positivo para la existencia de la nariz al ser multiplicado por *Wa* el vector resultante representa donde debe estar ubicada la cara acorde a la posición de la nariz. Si agregamos más cápsulas que representen los ojos, boca, cejas y otras características de la cara en conjunto con matrices *Wx* [59] que representan las relaciones con el rostro, obtenemos un modelo que es capaz de detectar rostros.

2.8.1.2 Multiplicación de pesos escalares con los vectores de activación

Este paso es similar al paso de multiplicación de escalares en neuronas artificiales. En el caso de las neuronas los pesos son ajustados durante propagación hacia atrás, pero en el caso de cápsulas, son determinados usando enrutado dinámico (explicado más adelante).

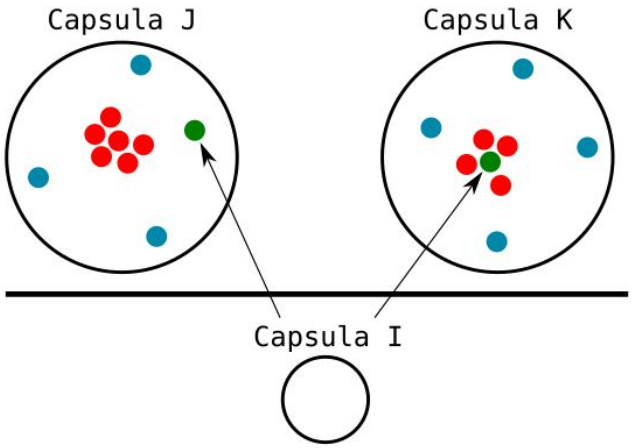


Figura 2.15 Una cápsula *i* envía su vector de activación a una cápsula de nivel superior que concuerda con su predicción, la coincidencia se representa como un cúmulo de aciertos.

En la figura 2.15 tenemos una cápsula que necesita decidir a qué cápsula enviar su vector. Está realizará su decisión de acuerdo con la dirección y la predicción que posee el vector de activación, esto conlleva a un ajuste del escalar que representa el peso, el cual también será multiplicado por el vector de activación antes de ser enviado a una capa de mayor nivel.

2.8.1.3 Sumatoria de vectores

Al igual que en neuronas artificiales, se suma el resultado del ponderado de los vectores de activación, en otras palabras, para todas las capas de cápsulas excepto la inicial, el vector de entrada para una cápsula *sj* es la suma ponderada (multiplicación con el escalar peso *cij*) de todos los vectores de activación *Uj|i* de las cápsulas de la capa anterior. *Uj|i* es la multiplicación de la matriz de peso *Wij*

con el vector *ui* (Ecuación 2.4), donde *i* representa la cápsula de la capa inferior, *j* la cápsula ubicada en la capa superior a *i*.

$$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i} \ , \qquad \hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij} \mathbf{u}_i$$

Ecuación 2.4 Fórmulas del vector de entrada *sj* y del vector de activación resultante *Uj|i.*

2.8.1.4 Squashing

Otra de las innovaciones de redes de cápsulas es el uso de una función de activación no lineal llamada *squashing*[5] (Ecuación 2.5) que toma un vector *sj* y lo reduce a un vector *vj* con magnitud no mayor que 1, sin cambiar su dirección. Este permite normalizar la información guardada en el vector *sj* sin perder información sobre la dirección y manteniendo la magnitud en el rango [-1, 1].

$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$$

Ecuación 2.5 Fórmula de squashing.

2.9 Enrutado dinámico

Como se vio anteriormente una cápsula *i* necesita decidir a cual cápsula *j* de la capa superior debe enviar su información. Esta decisión es tomada alterando el valor del escalar *cij* que multiplicará el vector de salida correspondiente. Su resultado será la entrada de la cápsula *j*.

Algunas características de *cij*:

- Su valor es un escalar positivo.
- Para cada capa de cápsulas, la sumatoria de todos los pesos *cij* es igual a 1.
- Para cada capa de cápsulas, el número de pesos es igual al número de cápsulas en el nivel superior.
- El valor final de *cij* es determinado durante el enrutado.

[5] Traducción al español: aplastamiento o minimización.

El enrutado dinámico permite que una cápsula de un nivel inferior provea su información a una cápsula de mayor nivel que esté de acuerdo con su entrada, es decir, la cápsula de nivel inferior solo mandará su información a las cápsulas que la necesiten. Este comportamiento se logra usando el siguiente algoritmo:

1. *enrutado dinamico*($\hat{\mathbf{u}}_{j|i}$, *r, l)*
2. *for toda capsula i en capa l y capsula j en capa (l + 1):* $b_{ij} \leftarrow 0$.
3. *for r iteraciones do*
4. *for toda capsula i en capa l:* $c_i \leftarrow softmax(b_i)$
5. *for toda capsula j en capa (l + 1):* $s_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
6. *for toda capsula j en capa (l + 1):* $v_j \leftarrow squash(s_j)$
7. *for toda capsula i en capa l y capsula j en capa (l + 1):* $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} v_j$
8. *return* v_j

Este algoritmo toma como entrada todas las salidas de las cápsulas de un nivel inferior al actual, el número de iteraciones de enrutamiento y la capa actual (cápsulas, sus pesos y vectores). Su salida es un vector *vj* que será la entrada para la siguiente capa del modelo.

La línea 4 calcula el valor del peso escalar ci el cual corresponde al escalar usado para decidir la ruta a tomar. Esta operación se realiza para todas las cápsulas del nivel actual. En la primera iteración cij serán iguales para todas las cápsulas y este valor se ajustará conforme el algoritmo decida cual cápsula debe recibir como entrada el vector *vj*.

Luego en la línea 5 se calcula la combinación lineal de todos los vectores de entrada para la capa de mayor nivel. Se utilizan los vectores de peso calculados en la línea 5 para estimar su valor correspondiente.

En la línea 6 se aplica squashing a todos los vectores obtenidos en la línea anterior. Por último en la siguiente línea ocurre la actualización de pesos. Este paso mira cada cápsula en un nivel superior y suma el resultado del producto punto entre el vector de entrada de la cápsula del nivel actual y el vector de salida de la capa superior al peso de la cápsula actual. El producto punto busca

similitudes entre ambos vectores (figura 2.16), si su resultado es negativo estos vectores corresponden a características distintas, si es positivo quiere decir que la cápsula del nivel superior recibirá como entrada el vector de salida de la cápsula anterior cuando sea necesario.

Las líneas 4-7 se ejecutan *r* veces donde *r* es el número de iteraciones en donde el modelo intentara crear relaciones entre las cápsulas. En la última iteración la lineal 7 no aporta ningún cambio, por lo tanto está puede ser omitida.

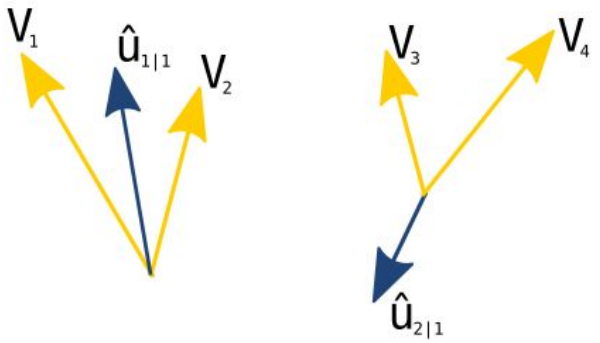


Figura 2.16 Comparación de resultados del producto punto entre vectores. Si el producto punto entre los vectores amarillos (salida) y el vector azul (entrada), es positivo (caso de la izquierda) la cápsula de la siguiente capa le otorga mayor peso a la entrada *vi*. El caso de la derecha resulta en un número negativo, es decir los vectores no son similares/correspondientes a la entrada recibida.

2.10 Arquitectura de una red de cápsulas

Una red de cápsulas consiste en un conjunto de capas de cápsulas colocadas en serie (figura 2.17).

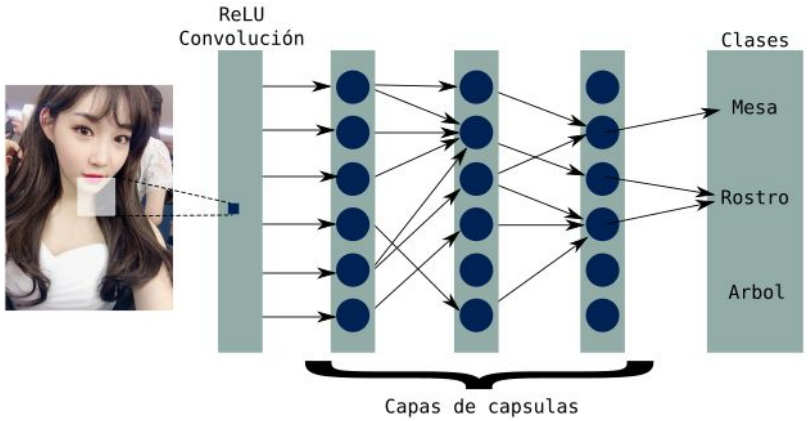


Figura 2.17 Arquitectura de red de cápsulas.

La primera capa de una red de cápsulas es una capa de neuronas convolucionales con función de activación ReLU (Rectified linear unit), esta capa se encarga de recibir y convertir la data de la imagen en vectores para la siguiente capa. Todos los vectores resultantes no poseen suficiente información espacial que permita decidir a qué cápsula enviar dicha predicción, por lo tanto se omite en enrutado dinámico entre esta capa y la siguiente.

La segunda capa en adelante son capas de cápsulas que reciben y retornan vectores de dimensión *N*, donde *N* es la cantidad de neuronas en una cápsula. En esta etapa se aplica enrutado dinámico entre cada capa donde se determina quienes reciben información y de su proveniencia.

La última capa está conformada por cápsulas que representan las clases a determinar por el modelo. Si una red de cápsulas busca determinar si en una imagen existen *X* cantidad de objetos, esta capa contendrá *X* cápsulas cada una representando a un objeto distinto.

CAPÍTULO III. Modelo de detección de emociones

En el siguiente capítulo se expone los datos de entrenamiento a usar, su análisis y preprocesamiento y los pasos que se siguieron para la creación del modelo de redes de cápsulas. También ofrecemos pruebas de rendimiento y eficiencia del modelo de cápsulas junto con un análisis comparativo entre una red convolucional usada para resolver el problema del trabajo especial de grado.

3.1 Conjunto de datos

Existen varios conjuntos de datos público:

- *PIE*[6] *Database*, CMU: contiene 41,368 imágenes de personas, categorizadas en 4 clases de emociones [91].
- *JAFFE*[7] *Database*: contiene 213 imágenes de personas, categorizadas en 7 clases de emociones [92].
- *The Child Affective Face Set*[8]: contiene 1,200 imágenes de niños, categorizados en 7 clases de emociones [93].
- Subconjunto de fer2013: *Kaggle Facial Expression Recognition Challenge*: contiene 28,709 imágenes de personas, categorizadas en 7 clases de emociones [88].

La base de datos *PIE* no es útil para nuestro problema debido a que posee solo 4 clases de emociones mientras que nosotros buscamos estudiar 6 clases distintas de emociones. *JAFFE* y *CAFS* poseen una cantidad muy pequeña de imágenes en comparación con fer2013, además que *JAFFE* son solo personas femeninas asiaticas y *CAFS* son niños de 2 a 7 años de edad. El set de datos fer2013 posee imágenes de rostros de todo tipo de persona sin importar edad, sexo o raza.

[6] Pose, Ilumination, Expression, en español: Posición, Iluminación y Expresión.
[7] Japanese Female Facial Expression, en español: Expresiones faciales femeninas de japonesas.
[8] Child Affective Face Set o CAFS, en español: Conjunto de rostros afectivos de niños.

Para este proyecto se usó un subconjunto tomado del conjunto de datos de la competencia de Kaggle, el cual contiene 23,736 imágenes divididas en 6 clases la cuales corresponden a las 6 emociones básicas universales (felicidad, miedo, sorpresa, disgusto, enfado y tristeza).

Se seleccionó tomar este conjunto debido a que posee la mayor cantidad de imágenes de entrenamiento y un conjunto de prueba con 3,589 imágenes, todas sus imágenes están asignadas a alguna de las 7 clases (6 emociones universales + neutral).

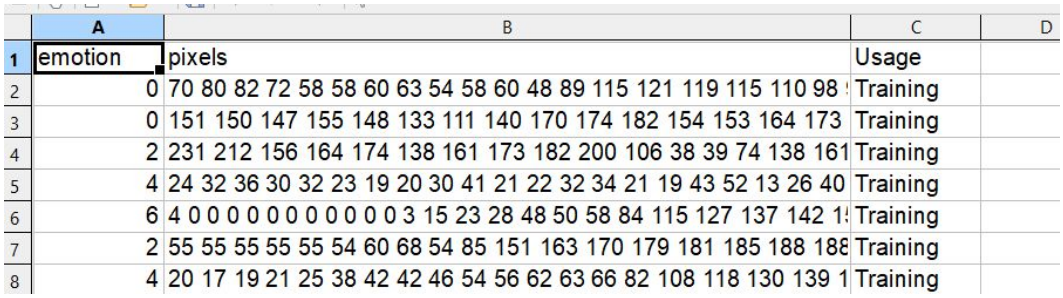
3.1.1 Análisis

El conjunto de datos contiene imágenes de rostros en escala de grises con un tamaño de 48x48. Las caras vienen centradas y ocupan el mismo espacio relativo a la imagen.

El conjunto de datos viene en archivo llamado “fer2013.csv”, el cual contiene tres columnas (Figura 3.1) “emotion”, “pixels” y “usage”, en español, emociones, pixeles y uso. En la columna de emoción existe un número del 0 al 6 que representa una de las 7 categorías (0=Enfado, 1=Disgusto, 2=Miedo, 3=Felicidad, 4=Tristeza, 5=Sorpresa, 6=Neutral). La columna de pixeles posee un string el cual contiene los valores de cada pixel separados por espacios. Por último la columna de uso contiene el string “Training” o “Test” los cuales dictaminan si la entrada es para entrenamiento del modelo o para pruebas.

El conjunto de datos en “fer2013.csv” contiene 28,709 entradas para entrenamiento y 3,589 para pruebas.

Originalmente el set de datos fue creado por Pierre-Lic Carrier y Aaron Courville, como parte de su proyecto de investigación [88].



	A	B	C	D
1	emotion	pixels	Usage	
2	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98	Training	
3	0	151 150 147 155 148 133 111 140 170 174 182 154 153 164 173	Training	
4	2	231 212 156 164 174 138 161 173 182 200 106 38 39 74 138 161	Training	
5	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 43 52 13 26 40	Training	
6	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 127 137 142 1	Training	
7	2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 181 185 188 188	Training	
8	4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108 118 130 139 1	Training	

Figura 3.1 Captura de pantalla del conjunto de datos en fer2013.csv

3.1.2 Preparación de datos

Debido a que se desea clasificar la emoción encontrada en una imagen, hace falta traducir el string en la columna de pixeles en una imagen, además de realizar el preprocesamiento pertinente para su uso como data de entrenamiento en el modelo implementado.

3.1.2.1 Traducción a imágenes

En cada entrada tenemos la clase, los pixeles y el uso de esta misma. Además se conocen los valores de los pixeles de cada imagen en escala de grises y que cada una tiene un tamaño de 48x48.

Con esta información se creó un pequeño script para convertir el conjunto de datos en el archivo fer2013.csv en un conjunto de imágenes divididas en carpetas siguiendo el uso que se le dará y la clase a la cual pertenece.

Figura 3.2 Estructura de carpetas resultante de la traducción del conjunto de datos.

Cada nodo o carpeta final de la Figura 3.2 contiene un conjunto de imágenes correspondiente a nombre de la carpeta según la emoción identificada.

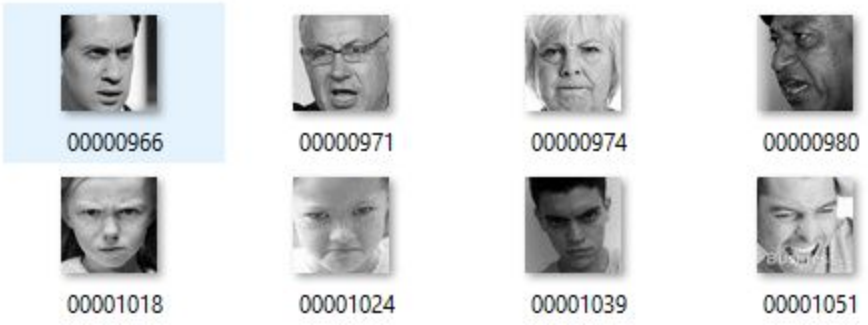


Figura 3.3 Subconjunto de imágenes en la ruta *./data/training/angry*

A esta estructura se debe eliminar en las carpetas “training” y “test” las carpetas correspondientes a la clase neutral, debido a que esta clase no está contemplada en el alcance del proyecto.

3.1.2.2 Preprocesamiento

Está fase es posible ser ignorada ya que las imágenes se encuentran en escala de grises, centradas y todas del mismo tamaño, sin embargo, al estudiar el proceso de comunicación entre capas de las redes de cápsulas se encontró que cada cápsula retorna un vector n-dimensional de magnitud x, donde $0 < x < 1$ [59]. Esta característica da a conocer que se requiere que la data de entrada también este entre los valores 0 y 1. Por lo tanto se creó una pequeña función que convierte los valores de píxeles de rango 0-255 al rango 0-1.

```
def preprocess_input_data(x_data):
    return x_data / 255.0
#End preprocess_input_data
```

Figura 3.4 Función de preprocesamiento.

3.2 Creación del modelo

En está sección se formaliza el proceso de creación del modelo de redes de cápsulas.

3.2.1 Tensorflow y Keras

Keras es una librería de nivel de modelo (por encima de Tensorflow, librería de alto nivel), que provee partes de alto nivel para el desarrollo de modelos de aprendizaje profundo. Está no maneja operaciones de bajo nivel como productos entre Tensores, convoluciones y entre otros tipos de estructuras. En vez de eso, se cuenta con una librería especializada y optimizada para la manipulación de Tensores, la cual sirve de "backend engine" o motor de segundo plano para Keras. Está librería hace que Keras maneje el problema de forma modular, permitiendo el uso de diferentes librerías en segundo plano sin inconveniente alguno.

Actualmente Keras posee tres "backend engines" disponibles: Tensorflow, Theano[9], y CNTK[10].

[9] Framework de código abierto para la manipulación de tensores.
[10] Conjunto de herramientas de código abierto para el desarrollo de aplicaciones basadas en aprendizaje de máquina.

En Keras también es posible cargar otras librerías de manipulación de Tensores además de las tres mencionadas anteriormente.

3.2.2 Diagramas

Primero desarrollamos dos diagramas uno de flujo para el entrenamiento del algoritmo (Figura 3.5) y un diagrama de clases (Figura 3.6) seguido de un conjunto de diagramas de clase complementarios donde se muestra la estructura interna de los objetos que componen cada capa. Todo esto para tener una mejor idea del algoritmo a desarrollar así como para observar requerimientos previos a su implementación.

Figura 3.5 Diagrama de flujo de entrenamiento del modelo de redes de cápsulas.

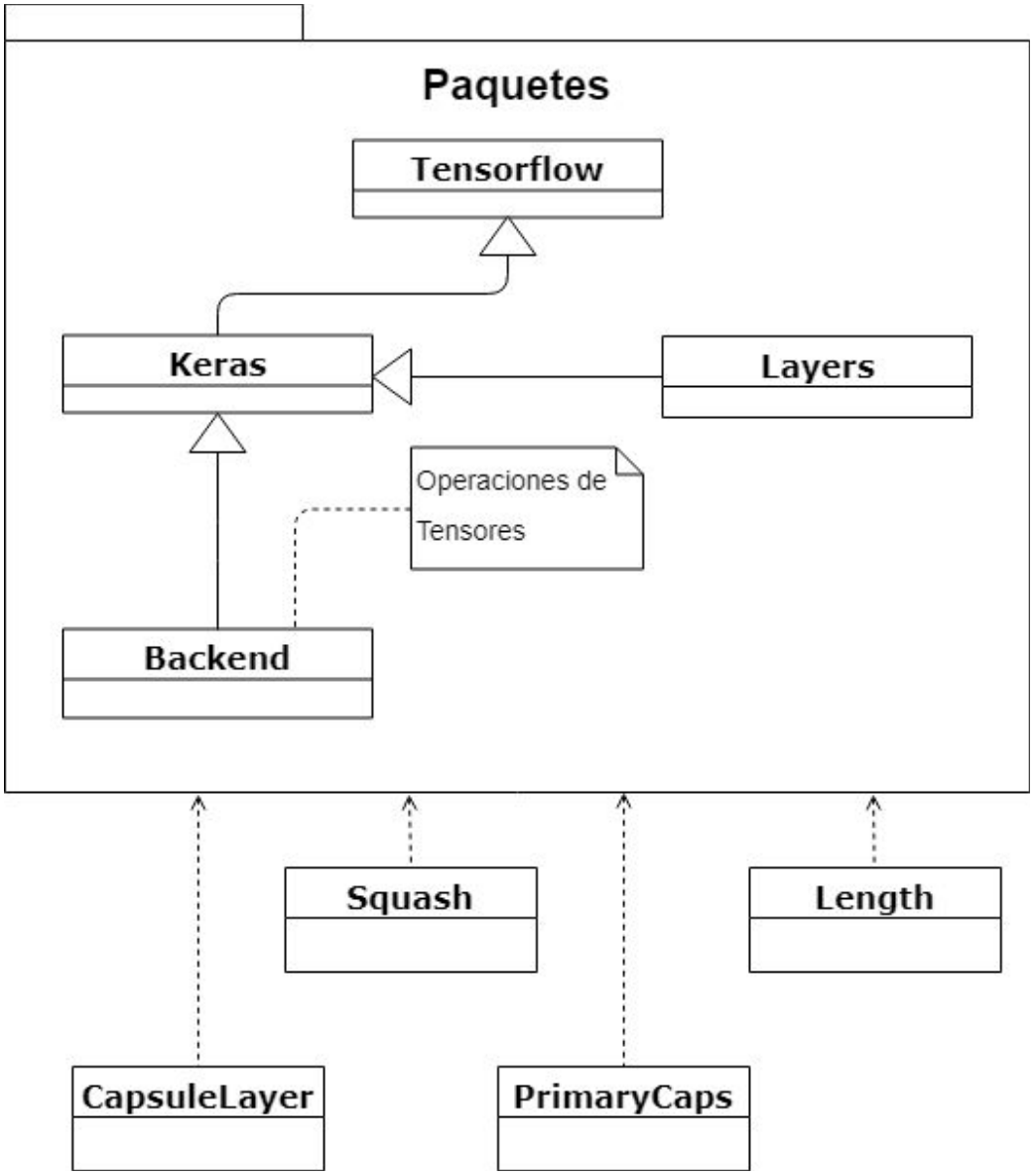


Figura 3.6 Diagrama de clases simplificado del algoritmo de redes de cápsulas en Keras.

A continuación realizamos una explicación del funcionamiento de cada clase implementada.

3.2.3 Clase de Layers y Backend

Todas las clases creadas heredan de la clase Layers ya que esta sirve de plantilla para crear nuevas capas compatibles con los módulos y funciones de Keras (Figura 3.7).

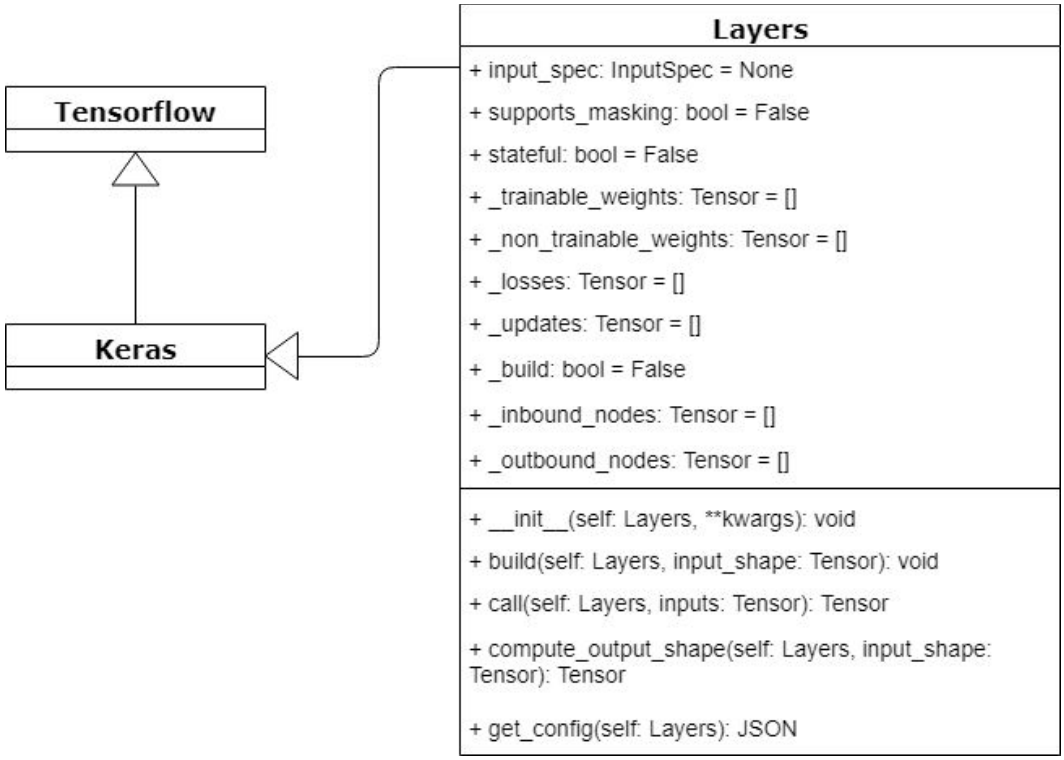


Figura 3.7 Diagrama de clases de la clase Layers.

En nuestra implementación no tocaremos las variables en esta clase debido a que Keras se encargará de su uso, pero si se sobreescribirá algunas funciones dependiendo de los requerimientos de la capa a crear.

Se debe tener en cuenta el objetivo de ejecución de cada función el cual describimos a continuación:

- *__init__*: Se llama a esta función al crear la clase (equivalente al constructor de la misma), usada para inicializar variables y creación del objeto.

- *build*: Función usada automáticamente por Keras antes de una llamada a la función *call* para crear estructuras de apoyo usada por la capa.
- *call*: Función donde se hace uso de la capa, aquí se define la utilidad de la capa para manipular los Tensores de entrada.
- *compute_output_shape*: Función para la definición explícita del tamaño del Tensor resultante de la función *call*. Cabe destacar que aquí no se cambia el Tensor resultante, está función es usada en caso de que se necesite de forma explícita el tamaño del Tensor (cualidad no requerida por algunas funciones de Tensorflow).
- get_config: Función de utilidad para mostrar la estructura de la clase en formato JSON.

El primer parámetro de todas las funciones de la clase corresponde a la referencia de la instancia la cual llamó al método. Por costumbre, este parámetro se llama self. Al escribir una llamada de una función perteneciente a una clase no es necesario pasar el objeto de clase por parámetro de forma explícita (Figura 3.8), Python maneja eso automáticamente.

```
class Restaurant(object):
    bankrupt = False
    def open_branch(self):
        if not self.bankrupt:
            print("branch opened")


>>> x = Restaurant()
>>> x.bankrupt
False
>>> x.open_branch()
branch opened
```

Figura 3.8 Ejemplo de llamada de una función dentro de una clase en Python.

3.2.4 Clase de squash

Squash es una clase que contiene una función que realiza el proceso de squashing a un Tensor explicado en el marco teórico. La última instrucción en la capa PrimaryCaps es una operación de squashing sobre el Tensor resultante. Cuando esta operación es hecha por una función, Tensorflow no refleja el tamaño del Tensor (las dimensiones modificadas por la función son definidas como

"None"), esto debido a que la librería asume que las demás capas son capaces de inferir las dimensiones y realizar su proceso de forma exitosa. En este caso como se usan capas personalizadas es necesario definir la función como su propia clase ya que de esta manera retorna de forma explícita las dimensiones del Tensor (Figura 3.9).

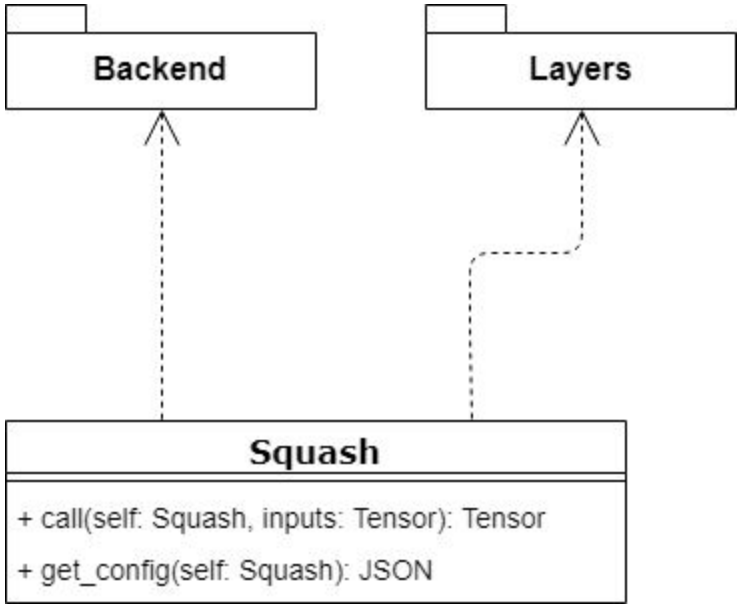


Figura 3.9 Diagrama de clases de la clase Squash.

La función de squash retorna una aproximación al algoritmo definido en el marco teórico. Esto se debe a la posibilidad de que la magnitud de alguna de las cápsulas sea cero, si esto llegase a ocurrir es posible que se requiera resolver una división entre cero. Por tal motivo agregamos un valor muy pequeño ε (epsilon[11]) al denominador de la segunda parte de la fórmula (Cálculo de vector unidad) para evitar dicho problema (Figura 3.1).

$$v_j = \frac{\left\|s_j\right\|^2}{1+\left\|s_j\right\|^2} \frac{s_j}{\left\|s_j+\varepsilon\right\|}$$

Ecuación 3.1 Función de squashing con suma de epsilon al denominador en el cálculo del vector unidad.

[11] Número infinitamente pequeño, de signo positivo.

Existe un segundo problema ligado a la posibilidad de obtener como resultado cero al calcular el valor de la norma sobre los vectores correspondientes. Tensorflow resuelve esta operación retornando *NaN* (Not a number) en vez de cero. Debido a este caso especial realizamos el cálculo de la norma de un vector sin usar la primitiva para calcular la norma de un vector ofrecida por Tensorflow (Figura 3.10)

```
s_squared_norm = K.sum(K.square(inputs), -1, keepdims=True)
```

Figura 3.10 Fragmento de código en Python para el cálculo de la norma del vector *inputs*.

3.2.5 Clase de cápsulas primarias

Las cápsulas primarias o PrimaryCaps es la clase encargadas de convertir la data de entrada en un conjunto de cápsulas (Figura 3.11).

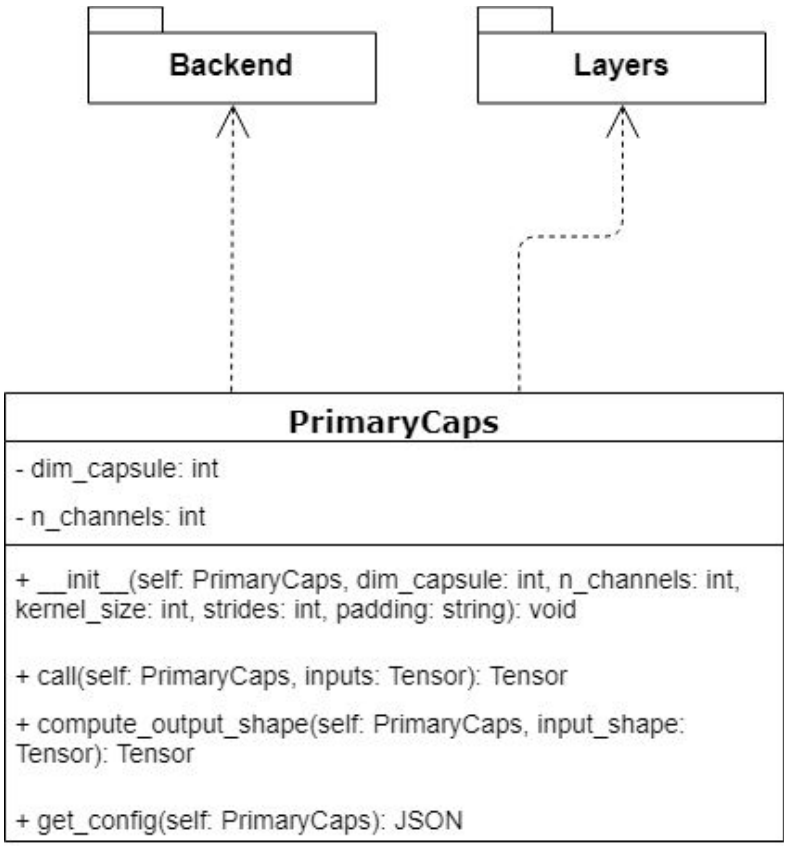


Figura 3.11 Diagrama de clases de la clase PrimaryCaps.

La función *__init__* recibe 6 parámetros de entrada:

- *self*: Referencia interna del objeto de tipo PrimaryCaps.
- *dim_capsule*: Dimensión o profundidad que tendrá cada cápsula resultante.
- *n_channels*: Cantidad de tipos distintos de cápsulas o cápsulas resultantes.
- *kernel_size*: Tamaño del kernel para la capa convolucional.
- *strides*: Cantidad de desplazamiento del filtro en la capa convolucional.
- *padding*: Puede poseer los valores "*valid*" (sin relleno en los bordes) o "*same*" (con relleno en los bordes dependiente del parámetro strides), usado para la capa de convolución.

El método *call* es usado para crear las cápsulas a partir del Tensor de entrada *inputs*, luego el método *compute_output_shape* se encarga de comprobar que el Tensor resultante contiene una cantidad de cápsulas acorde a la fórmula en la Ecuación 3.2

$$A = (ancho\ de\ imagen - kernel\_size) / strides$$
$$B = (alto\ de\ imagen - kernel\_size) / strides$$
$$número\ de\ cápsulas = A * B * n\_channels$$

Ecuación 3.2 Fórmula para obtener el total de cápsulas en el Tensor resultante de la capa PrimaryCaps.

3.2.6 Clase de capa de cápsulas

Equivalente a las capas escondidas en los modelos de redes convolucionales la clase de capa de cápsulas o CapsuleLayer (Figura 3.12) es la capa que posee información sobre como será la comunicación entre las distintas capas del mismo tipo, planteadas de forma secuencial en el modelo.

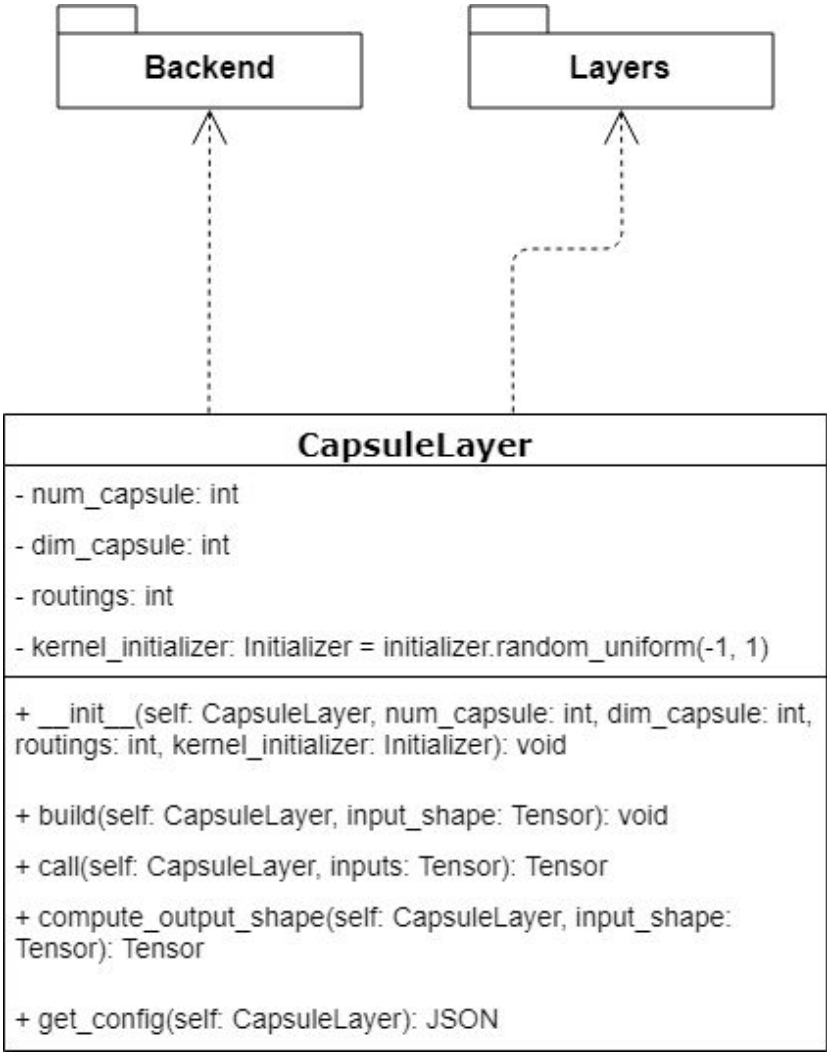


Figura 3.12 Diagrama de clases para la clase CapsuleLayer.

En nuestra implementación definimos la clase con 5 parámetros de entrada para inicialización de variables:

- *self*: Referencia interna del objeto de tipo CapsuleLayer.
- *num_capsule*: Número entero; Cantidad de cápsulas de la capa.
- *dim_capsule*: Número entero; Dimensión o profundidad de cada cápsula.
- *routings*: Número entero; Cantidad de ciclos en los que se realizará el proceso de routing (explicado en el marco teórico).
- *kernel_initializer*: Initializer; Función de inicialización de valores.

Una vez inicializada las características de la capa (función *__init__* se ejecutó) pasamos a construir la matriz *Wij*, la cual será la encargada de guardar un número equivalente a la familiaridad de las cápsulas de capa actual (capa *i*) con cápsulas de la capa anterior (capa *j*).

Al tener la matriz *Wij* se inicia el proceso definido en la función call, el cual se encarga de parear los valores de la matriz W_{ij} con las cápsulas de entrada, eso nos da un Tensor resultante que se usa para realizar el proceso de enrutado dinámico.

El proceso de enrutado dinámico nos retorna un Tensor con dimensiones en concordancia a la cantidad de cápsulas y dimensión de cápsulas definidas anteriormente.

3.2.7 Clase de magnitud de vector

La clase de magnitud de un vector o Length (Figura 3.13) es una capa para calcular la magnitud de los vectores resultantes de la última capa de cápsulas, es decir, se toma cada cápsula resultante, se trata como un vector y se calcula su magnitud la cual se espera refleje el resultado final deseado.

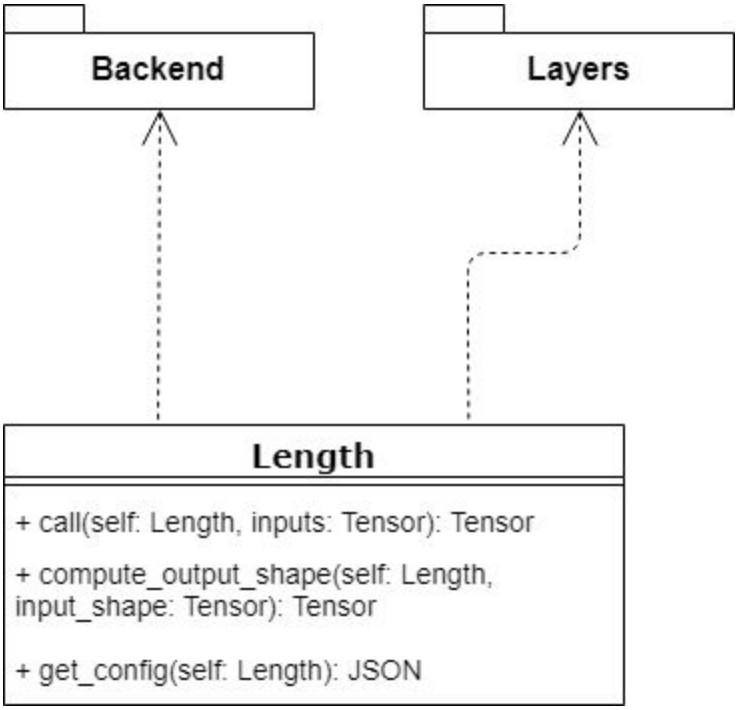


Figura 3.13 Diagrama de clases para la clase Length.

3.2.8 Definición del grafo en Keras

Como ya conocemos todas las clases o capas por las cuales nuestro modelo estará conformado, nos queda realizar la definición del grafo de capas (Figura 3.14).

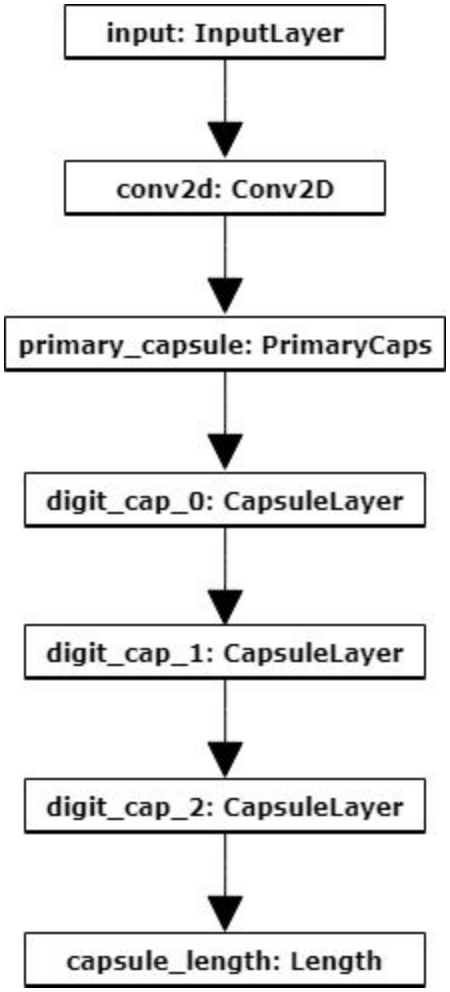


Figura 3.14 Grafo de red de cápsulas para detección de emociones.

Se puede notar que la primera capa es llamada InputLayer o capa de entrada. Esto se debe a que la data de entrada debe de ser convertida a Tensor antes de ser procesada por el modelo. Recordemos que un Tensor es un arreglo n-dimensional por lo tanto esta conversión no es más que una conversión del tipo arreglo a Tensor.

Luego se cuenta con la capa PrimaryCaps la cual se encargará de convertir ese Tensor de entrada en cápsulas, que serán procesadas secuencialmente por las siguientes tres capas de cápsulas.

Se implementan tres capas de cápsulas secuenciales debido a la efectividad que estas tienen al determinar la solución con una pequeña cantidad de cápsulas [59], además que de esta manera se mantiene el modelo de bajo tamaño.

También se busca tener un nivel de eficiencia alto colocando la cantidad de routings a tres en cada capa. Está decision que al analizar el proceso de enrutado dinámico el realizar la actualización de pesos el cambio en ellos se reduce rápidamente a mayor cantidad de actualizaciones, realizar un ciclo más no produce un cambio notorio en el resultado del algoritmo.

La última capa de cápsulas posee una cantidad de cápsulas equivalente a la cantidad de clases del problema, en nuestro caso como son 6 emociones universales tendremos 6 cápsulas que representan cada emoción. A cada una de estas cápsulas se le calcula el vector magnitud en la capa de Length y este valor es el que corresponda en la probabilidad de que una emoción esté presente o no en la imagen de entrada.

3.2.9 Optimizador y función de pérdida

Construido el grafo, hace falta definir un optimizador y una función de pérdida.

Un optimizador es una función que busca minimizar la pérdida de datos en el modelo. Este proceso se realiza comparando y sumando los errores entre el set de entrenamiento y el de validación o entre el conjunto de entrada y su predicción correspondiente, su resultado determina que tan alta o baja es la pérdida de datos del algoritmo. Esta función es llamada función de pérdida. El optimizador se encarga de reducir la pérdida de datos haciendo modificaciones a los valores entrenables[12] en el modelo.

Keras proporciona una serie de optimizadores y funciones de pérdidas listas para ser usadas en cualquier modelo hecho en este framework.

[12] Nuestro modelo posee solo parámetros entrenables.

Se utilizó como optimizador Adam[13] con una tasa de entrenamiento de 0,001. Y como función de pérdida se usó "categorical_crossentropy" o entropía cruzada categórica definida como *H(p, q)*.

$$H(p,q) = -\sum_{i}^{C} p_i log(q_i)$$

Ecuación 3.3 Fórmula para el calcula de entropía cruzada.

En la Ecuación 3.3 se observa la fórmula que sigue la función de pérdida, donde, p corresponde al vector de entrada, q al vector de predicción generado por el algoritmo y C es la cantidad de clases del modelo.

3.3 Entrenamiento del modelo

Culminada la preparación del conjunto de datos y la definición del modelo, se empieza con la fase de entrenamiento. El entrenamiento en el framework de Keras es bastante simple, por el hecho de otorgar clases y objetos para la fácil manipulación y definición de grafos de redes convolucionales.

Para el entrenamiento primero se creo un objeto que clasificara las imágenes según la carpeta en donde se encuentren, es decir, cada carpeta corresponde a una clase (Figura 3.2). Este objeto llamado ImageDataGenerator proviene de la librería Keras. También permite el pase de una función de preprocesamiento el cual se usará para normalizar los valores de las imágenes de rango 0-255 al rango 0-1.

Por último se pasa el objeto a la función *fit_generator* del modelo, junto con el tamaño del bache[14], la cantidad de epochs[15] y la cantidad de pasos por cada epochs[16] a procesar.

[13] Algoritmo de optimización usado para sustituir el proceso clásico de descenso de gradiente, el cual provee mejoras de estabilidad y velocidad de cómputo en entrenamiento de algoritmos de aprendizaje.
[14] Número de imágenes a usar durante una iteración.
[15] Cantidad de iteraciones en el set de entrenamiento..
[16] Cantidad de subconjuntos o baches tomados en cada epoch.

Durante este proceso se realizaron distintas modificaciones en los valores de los distintos parámetros y en la cantidad de capas de cápsulas del modelo explicado y nos quedamos con la mejor estructura encontrada (Figura 3.15)

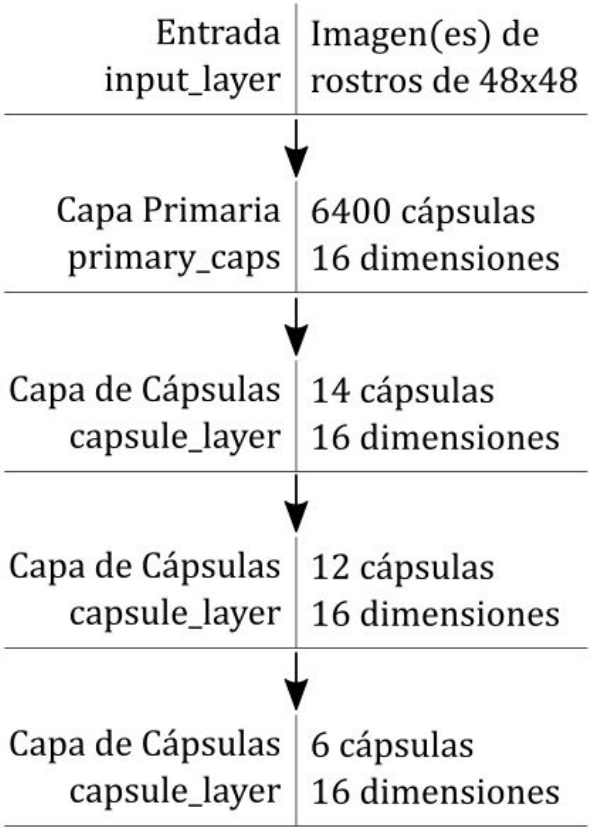


Figura 3.15 Estructura lógica del modelo con tres capas de cápsulas.

Se observa que la capa primaria (figura 3.15) posee una gran cantidad de cápsulas con respecto a las capas de cápsulas, según nuestro análisis las capas primarias realizan una captura y retención de datos eficiente por sí solas, por lo tanto buscamos que estás realicen la parte "pesada", es decir, la mayor cantidad de trabajo, para que las capas restantes solo se encargen de minimizar y definir el resultado.

Total data de entrenamiento	**Baches de entrenamiento**	**Iteraciones**
23736 Imágenes	24	700

Precisión en entrenamiento	Total data de validación	Precisión en validación
0.878 (88%)	5945 Imágenes	0.6633 (66%)

Tabla 3.2 Cuadro de parámetros de entrenamiento y validación.

El entrenamiento con los parámetros de la Tabla 3.2 tiene una duración de 9 horas aproximadas, mientras que la validación tarda 41 minutos en ejecución.

3.4 Evaluación

En la siguiente sección se estudia la precisión, exhaustividad, valor-F (o en inglés F-score) y soporte de los modelos. De este mismo modo, se define la matriz de confusión resultante. Tanto para el cálculo de precisión como para el de la matriz (y su consiguiente graficación), se usó *sklearn* como librería para la manipulación y muestra de métricas de evaluación del modelo, complementando las funciones encontradas en Keras.

3.4.1 Precisión, Exhaustividad, Valor-F y Soporte

Las dos medidas más frecuentes y básicas para calcular la efectividad de un sistema de clasificación suelen ser: la precisión y exhaustividad [94].
Para ejemplificar mejor sobre que se considera la precisión y exhaustividad, podemos observar la Tabla 3.2.

	Reales Positivos	Reales Negativos
Predicciones Positivas	TP	FP
Predicciones Negativas	FN	TN

Tabla 3.2 Cuadro de Contingencia Binaria.

Los Verdaderos Positivos y Falsos Positivos (TP y FP, respectivamente) se refieren al número de Predicciones Positivas, hayan sido clasificados de forma

correcta (TP) o incorrecta (FP). De manera similar para los Verdaderos Negativos y Falsos Negativos (TN y FN, respectivamente), se refieren a aquellos Predicciones Negativas que fueron clasificados de forma correcta (TN) o incorrecta (FN). A su vez las columnas de Reales Positivos (TP y FN) y Reales Negativos (FP y TN) son los que indican los verdaderos valores positivos y negativos respectivamente, independientemente de la predicción hecha por el modelo.

Estas cuatro variables, suman N. Por otro lado, TP, FP, FN y TN se refieren a las probabilidades conjuntas y marginales [95]. Definimos entonces, a la precisión o confianza, como la proporción de casos Positivos Predichos que son Reales Positivos. Podemos ver entonces la precisión de una forma más matemática, con la ecuación en la Ecuación 3.3, en la cual puede apreciarse que significa esta métrica.

$$precisión = \frac{TP}{TP + FP}$$

Ecuación 3.3 Ecuación para el cálculo de la precisión.

Similar se tiene el caso de la exhaustividad (en inglés recall), la cual no es más que la proporción de casos reales positivos que son correctamente predichos positivos. Esto mide la Cobertura de los casos Realmente Positivos[95]. Podemos ver la fórmula de la exhaustividad en la Ecuación 3.4.

$$exhaustividad = \frac{TP}{TP + FN}$$

Ecuación 3.4 Ecuación para el cálculo de exhaustividad.

El Valor F (F-Score en inglés o en ocasiones F measure), a su vez, se denomina como la media armónica de precisión y exhaustividad. A menudo se utiliza para medir el rendimiento de un sistema cuando se prefiere un solo número[96]. Puede verse en la Ecuación 3.5.

$$Valor\ F = \frac{2}{\frac{1}{precisión} + \frac{1}{exhaustividad}}$$

Ecuación 3.5 Ecuación para el cálculo del Valor F.

Por último se tiene el soporte puede verse como la cantidad de individuos con los que se realizó la evaluación. Sabiendo esto, podemos resumir los resultados en la Tabla 3.3. En esta se puede observar el reporte obtenidos con la red de cápsula entrenada.

	Precisión	**Exhaustividad**	**Valor F**	**Soporte**
Felicidad	0.89	0.86	0.88	1774
Disgusto	0.37	0.28	0.32	111
Miedo	0.66	0.57	0.61	1024
Tristeza	0.73	0.68	0.7	1247
Sorpresa	0.71	0.76	0.76	831
Enojo	0.62	0.89	0.73	958
Promedio/Total	0.66	0.67	0.66	5945
Ponderado	0.74	0.75	0.74	

Tabla 3.3 Cuadro de reporte de resultados usando la data de validación.

En la Tabla 3.3 existen cinco columnas donde: La primera, sin cabecera, representa las seis clases en las que los rostros pueden ser categorizados, seguido tenemos la precisión que se mide tanto por clase como en total, la exhaustividad, el valor F y por último, tenemos el soporte. Esta última columna es la que expresa el número de individuos con los que se probó el modelo. En nuestro caso la prueba involucra un total de 5945 rostros que provienen del conjunto de datos de validación de Kaggle.

Se puede notar en la columna de soporte la baja presencia que posee la clase Disgusto en comparación con las demás clases, esta falta de datos se refleja al momento de evaluar la precisión del modelo, mostrando un nivel bajo.

En caso contrario cuando se observa la precisión de la clase Felicidad, parece mostrar que a mayor soporte, mayor precisión y menor exhaustividad.

3.4.2 Matriz de confusión

Una tabla de errores *2 x 2* para una clasificación binaria, cubre las situaciones más prominentes que se pueden dar. Además, a menudo, los problemas multiclases son definidos en términos de una serie de clasificaciones binarias.

En ocasiones, como en este caso particular, muchas hipótesis están compitiendo por una sola clasificación. Aunque puede haber muchas clases, suelen ser mutuamente excluyentes y para cualquier ejemplo, solo uno puede ser aplicable. Para estos casos, en lugar de una tabla de errores de 2 x 2, se puede construir una tabla *m x m*, en la que *m* es el número de clases.

Este tipo de tabla, es denominada matriz de confusión y sirve para ofrecer una imagen de como se relacionan las variables predichas por un modelo y las verdaderas clasificaciones de estas (variables). Como ejemplo, podemos observar la Tabla 3.4, la cual ilustra una matriz de confusión de *3 x 3* en la que las etiquetas de columna son las etiquetas de clase reales (en este caso se tienen clase: 1, 2 y 3) y las etiquetas de la son las etiquetas predichas. La diagonal se convierte entonces en todas las clasificaciones correctamente predichas y el número de errores específicos es observable [96]. Es por esto, que se busca tener la mayor cantidad de valores en la diagonal principal. Si se detalla la tabla se describe como la intersección de las mismas etiquetas, por ejemplo 1 con 1, son las variables predichas como 1 y que son en efecto clase 1, en el ejemplo es el valor 80. Por otro lado si la la es 2 y la columna es 1, entonces significa que el modelo clásico erróneamente como clase 2 cuando en realidad era clase 1 (en el ejemplo son 10).

	1	2	3
1	80	15	5
2	10	70	20
3	0	0	100

Tabla 3.4 Ejemplo de matriz de confusión.

A continuación se muestra la matriz de confusión resultante de nuestro modelo de redes de cápsulas.

		Valor Verdadero					
		Felicidad	**Disgusto**	**Miedo**	**Tristeza**	**Sorpresa**	**Enojo**
Predicción	**Felicidad**	1579	39	5	48	129	27
	Disgusto	0	41	11	43	49	3
	Miedo	56	3	674	173	1	72
	Tristeza	43	16	300	913	57	223
	Sorpresa	61	0	24	61	591	37
	Enojo	35	12	10	9	4	596

Tabla 3.5 Matriz de confusión para el modelo de cápsulas usando el set de validación de Kaggle.

La matriz de confusión (Tabla 3.5) nos permite ver hacia donde tienden las predicciones incorrectas del modelo, observamos el caso entre las clases Miedo y Tristeza, es evidente notar que la similitud de ambas expresiones faciales dificulta su correcta clasificación y en alrededor del 29% de los casos la clase Miedo es confundida con la clase Tristeza y viceversa en el 14% de los casos.

También observamos un pequeño problema al diferenciar la clase Sorpresa de la clase Felicidad. Esto se debe a que la red posee mayor cantidad de

ejemplos de Felicidad que alguna otra clase y por lo tanto parece asociar la apertura de la boca como una característica común en los rostros que expresan felicidad. Otra evidencia del peso que tiene la clase Felicidad es observada al clasificar la clase Disgusto.

3.4.3 Errores

Error Medio Absoluto (MAE por sus siglas en inglés, Mean Absolute Error): mide la magnitud promedio de los errores en un conjunto de predicciones sin considerar su dirección. Es el promedio sobre la muestra de prueba de las diferencias absolutas entre la predicción y la observación real donde todas las diferencias individuales tienen el mismo peso, puede observarse Figura 3.27 [97][98].

$$\frac{1}{n}\sum_{i=1}^{n}\left|e_i\right|$$

Ecuación 3.6 Ecuación para el cálculo del Error Medio Absoluto.

Error Cuadrático Medio (MSE por sus siglas en inglés Mean Squared Error): es una regla de puntuación cuadrática que también mide la magnitud promedio del error [97]. En este caso se hace mediante el promedio de las diferencias cuadradas entre la predicción y la observación real, esto puede verse en la Ecuación 3.7 [98].

$$\frac{1}{n}\sum_{i=1}^{n}(e_i)^2$$

Ecuación 3.7 Ecuación para el cálculo del Error Cuadrático Medio.

Como puede verse en la Tabla 3.6, los errores (tanto MAE como MSE), son errores bastante bajos a pesar de tener una precisión del 66%. Esto indica que la red está dando un peso notable a la predicción correcta pero no mayor que la predicción incorrecta. Un ejemplo de este comportamiento lo podemos observar en la Tabla 3.7.

	MAE	MSE
Red de cápsulas	0.1655	0.0735

Tabla 3.6 Cuadro de reporte de los errores MAE y MSE.

	Felicidad	**Disgusto**	**Miedo**	**Tristeza**	**Sorpresa**	**Enojo**
 Enojo	0.0383	0.2242	0.2747	**0.4376**	0.0731	**0.3902**
 Enojo	0.0437	0.1260	**0.7179**	0.1218	0.0350	**0.5014**
 Tristeza	0.2232	0.0432	**0.5106**	**0.2232**	0.1034	0.0348

Tabla 3.7 Ejemplos de algunos falsos positivos encontrados en el set de validación.

3.4.4 Velocidad

Para evaluar la velocidad de ejecución creamos un pequeño programa que detecta los rostros y carga esa data a nuestro modelo de redes de cápsulas para su estudio dentro de tres ambientes:

Análisis de una imagen: Pasamos una imágen donde se busca detectar y clasificar la emoción de cada rostros detectado.

Imagen	Tamaño real: 1024x739	
Velocidad		0.47 segundos
Rostros detectados		15
Precisión media		53%

Tabla 3.8 Reporte de análisis de una imagen.

Realizamos el mismo proceso con 9 imágenes más, todas provenientes de la base datos Wider Faces [100]. Obtuvimos una velocidad promedio de 0.34 segundos, donde cada imagen tiene un promedio de 8 rostros. La precisión media es del 69%, destacando que la base de datos no proporciona data sobre qué expresión posee cada rostro, la emoción de cada rostro fue colocada por nosotros antes de su análisis.

Cámara en vivo: Tomamos la entrada de vídeo de una cámara, analizamos cada frame y clasificamos las expresiones faciales de los rostros detectados.

FPS[17]	20 ~ 30
Rostros en cámara	2
Precisión promedio	62%

Tabla 3.9 Reporte de análisis de cámara en vivo.

Observamos que el modelo es capaz de realizar detección de emociones en tiempo real (24 FPS es la velocidad de una película o vídeo promedio [101]) con una precisión promedio del 62%.

3.5 Red de cápsulas vs Red neuronal profunda

En esta sección se usará una red neural profunda para realizar comparativas de rendimiento con nuestro modelo de redes de cápsulas.

La red profunda a estudiar se refiere a un modelo capaz de detectar 7 clases de emociones, llamado Mini Xception (Figura 3.16), donde 6 de estas son las contempladas en nuestro proyecto, más una séptima que corresponde a la expresión facial neutra o sin expresión facial aparente [99]. Está red profunda usa el mismo set de datos de entrenamiento que nuestro modelo de red de cápsulas y obtiene la misma precisión (66%) al clasificar 7 expresiones faciales distintas. Cabe acotar que en el paper la red profunda resultante no solo busca clasificar emociones sino también determinar el género de la persona, pero ambas funcionalidades fueron desarrolladas independientemente de los resultados de ambas funcionalidades.

Existe otro modelo que clama poseer precisión del 99% al detectar y clasificar expresiones faciales en dispositivos móviles [29]. Realizamos un exhaustivo análisis del mismo donde nos encontramos con un error en la manera de validar los datos, donde ellos usan la misma data de entrenamiento como validación. Esto no es recomendado debido a la tendencia de los algoritmo de aprendizaje de máquina a crear un prejuicio o tendencia sobre los datos de entrenamiento. Por consiguiente decidimos usar el modelo Mini Xception el cual podemos recrear y comprobar resultados.

[17] Del inglés Frames Per Second (FPS), se refiere a la cantidad de imágenes procesadas en 1 segundo.

Primero realizamos el test de precisión, exhaustividad, valor F y soporte con la red Mini Xception (Tabla 3.10).

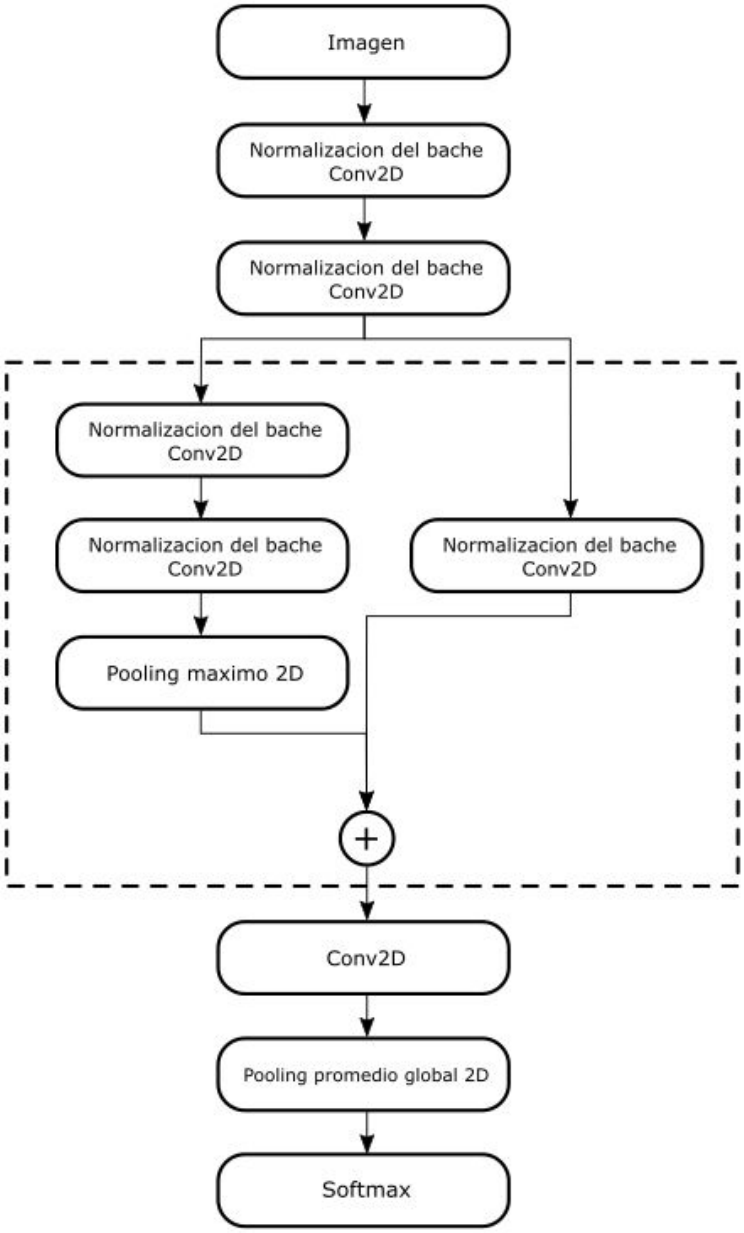


Figura 3.16 Estructura interna del modelo Mini Xception.

	Precisión	Exhaustividad	Valor F	Soporte
Felicidad	0.87	0.92	0.89	1774
Disgusto	0.59	0.63	0.61	111
Miedo	0.53	0.57	0.55	1024
Tristeza	0.57	0.56	0.57	1247
Sorpresa	0.81	0.8	0.8	831
Enojo	0.68	0.6	0.64	958
Promedio/Total	0.68	0.68	0.68	5945
Ponderado	0.70	0.71	0.71	

Tabla 3.10 Cuadro de reporte de resultados de la red Mini Xception usando la data de validación de Kaggle.

Basados en el paper de Mini Xception [99] se recreó el algoritmo y se realizó el entrenamiento con el conjunto de 6 clases que se usó para entrenar la red de cápsulas, igualmente se usó el conjunto de validación proporcionado por ese conjunto de datos para realizar un reporte de resultados.

Se observa que la precisión del modelo de redes convolucionales es mayor (68%) que nuestro modelo (66%), además en general parece establecer en mejor medida las diferencias entre cada clase, mientras que el modelo de redes de cápsulas parece estar a favor de las clases con mayor soporte.

		Valor Verdadero					
		Felicidad	**Disgusto**	**Miedo**	**Tristeza**	**Sorpresa**	**Enojo**
Predicción	**Felicidad**	1552	5	18	48	47	20
	Disgusto	0	66	13	24	0	2
	Miedo	55	4	542	199	68	88
	Tristeza	58	4	318	707	33	142
	Sorpresa	58	0	29	31	575	52
	Enojo	51	32	104	238	0	654

Tabla 3.11 Matriz de confusión para el modelo convolución Mini Xception usando el set de validación de Kaggle.

También se realizó un reporte de la matriz de confusión para el modelo Mini Xception (Tabla 3.11) donde se observa en mayor detalle como la red neuronal normalización los resultados, es decir, a pesar que la clase Disgusto solo tiene 111 imágenes de soporte está obtiene una precisión del 59%, a diferencia de la red de cápsulas con un 37%.

CAPÍTULO IV. Conclusiones y trabajos futuros

En el proyecto especial de grado expuesto, se puede concluir que a pesar de haber obtenido resultados que están por debajo de los modelos convolucionales modernos[103][104], fue posible crear un algoritmo basado en el modelo de redes de capas capaz de resolver el problema de estudio.

Se puede decir que se logró implementar y modificar el modelo de cápsulas planteado por Hinton [59], y observar su uso en un ambiente distinto a la clasificación de números escritos a mano.

También la realización de comparativas entre modelos de redes convolucionales nos permite observar de forma competitiva el rendimiento de nuestro modelo de redes de cápsulas que a pesar de poseer un nivel de precisión bajo, es capaz de ejecutar entre 20-30 FPS usando una cámara en vivo sin saltar ningún frame y realizar una predicción acertada el 66% de los casos.

Por último queremos plantear la existencia de posibles mejoras que pueden ser agregadas sobre el proyecto realizado que pueden plantearse para trabajos futuros. Entre estas funciones que se podrían plantear para el futuro, se encuentran:

- Agregar una máscara que actue como regenerador de los datos de entrada funcionando junto con la función de pérdida para mejorar la precisión del modelo, este acercamiento lo podemos observar en el paper que motivó el trabajo especial de grado [59].
- Realizar un modelo híbrido, es decir, unir capas convolucionales con capas de cápsulas e incluir métodos de regularización de datos como Dropout[18], motivado a la dificultad del modelo en converger debido a la poca presencia de mecanismos para regularizar datos.
- Cambiar el proceso de enrutado dinámico, el cual, a pesar de ser una solución eficaz para la comunicación entre capas este incrementa de forma exponencial la velocidad de la fase de entrenamiento, a mayor cantidad de cápsulas existan entre cada conjunto de capas.

[18] Proceso de ignorar algunas neuronas de forma aleatoria durante la fase de entrenamiento.

Bibliografía

[1] Gonzalez, Rafael C., Woods, Richard E., "Digital Image Processing", Tercera Edición, Pearson Prentice Hall, pp. 1-3, 2008.

[2] Kaufman, Arie, "Rendering, Visualization and Rasterization Hardware", Springer Science & Business Media, pp. 86–87, 1993.

[3] Arntson, Amy, "Graphic Design Basics", Sexta Edicion, Cengage Learning. pp. 194. 2011.

[4] "Raster Graphics - Scratch Wiki". https://wiki.scratch.mit.edu/wiki/Raster_Graphics. Accedido el 10 de Diciembre del 2017.

[5] Easton, Roger L., "Fundamentals of Digital Image Processing", Prentice Hall, pp. 4, 2010.

[6] Raschka, Sebastian, "Python Machine Learning", Primera Edición, Packt Publishing, 2-6, 2015.

[7] Witten, Ian H., Eibe, Frank, "Data Mining Practical Machine Learning Tools and Techniques", Segunda Edición, Elsevier, pp. 43, 2005.

[8] "Examples of Classification Tasks Examples of Classification Tasks", http://web.cs.ucdavis.edu/~vemuri/classes/ecs271/lecture3.pdf. Accedido el 12 de Diciembre del 2017.

[9] Kotsiantis, Sotos B., "Supervised Machine Learning: A Review of Classification Techniques", en *Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering*, pp. 2, 2007.

[10] "Cross Validation", https://www.cs.cmu.edu/~schneide/tut5/node42.html. Accedido el 12 de Diciembre del 2017.

[11] Zhang, Yagang, "Application of Machine Learning", InTech, 2010.

[12] Wong, Kwok W., Lam, Kin M., Siu, Wan C. "An efficient algorithm for human face detection and facial feature extraction under different conditions", *Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong*, 2000.

[14] Jain Anil K., Ross, Arun, Prabhakar Salil. "An introduction to biometrics recognition". en *IEEE Transactions on Circuits and Systems for Video Technology* pp. 4-20, 2004.

[15] Adini, Y., Moses, Y., Ullman, S. "Face Recognition: The problem of Compensating for Changes in Illumination Direction", en *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 721-732, 1997.

[16] Du, Shan. "Adaptive Region-Based Image Enhancement Method for Robust Face Recognition Under Variable Illumination Conditions", en *IEEE Transactions on Circuits and Systems for Video Technology*, 2010.

[17] Huang Di. "Robust face recognition based on three dimensional data". en *LIRIS - Laboratoire d'InfoRmatique en Image et Systèmes d'information*, pp. 3-4, 2011.

[18] Kouzani, Abbas Z., Fangpo He, Karl Sammut. "Commonsense knowledge-based face detection", en *Intelligent Engineering Systems, 1997. INES '97. IEEE International Conference*. pp. 215-219, 1997.

[19] Devadethan, S., Geevarghese Titus, Purushothanman, S. "Face detection and facial feature extraction based on a fusion of knowledge based method and morphological image processing", en *Emerging Research Areas: Magnetics, Machines and Drives (AICERA/iCMMD)*. pp 1-5, 2014.

[20] Fei, Zhao, Qiao, Qiang. "Face detection based on rectangular knowledge rule and face structure", en *Information Science and Engineering (ICISE)*. 2009.

[21] Yadav, Shalini, Neeta Nain. "Fast face detection based on skin segmentation and facial features", en *Signal-Image Technology & Internet-Based Systems (SITIS)*. pp. 663-668, 2015.

[22] Harrington, Peter. "Machine Learning in action", Primera Edicion, Manning, 3-8, 2012.

[23] Dubey, Monica, Singh, Lokesh. "Automatic Emotion Recognition Using Facial Expression: A Review", en *International Research Journal of Engineering and Technology*. 2016.

[24] Mateusz Zarkowski "Identification-driven Emotion Recognition System for a Social Robot" en *Methods and Models in Automation and Robotics (MMAR), IEEE.* 2013

[25] Mehrabian A., "Communication without words", Psychology today, vol. 2, n°4, pp. 53-56, 1968.

[26] Sebe Nicu, Lew Michael S., Cohen, Ira, Garg, Ashutosh, "Emotion Recognition Using a Cauchy Naive Bayes Classifier", en *Pattern Recognition, 2002. Proceedings. 16th International Conference IEEE*, 2002.

[27] Ekman P., Friesen, W.V. "Facial Action Coding System: Investigator's Guide". Consulting Psychologists Press, Palo Alto, CA, 1978.

[28] Littlewort, G., Fasel, I., Stewart, M., Movellan, J. "Fully automatic coding of basic expressions from video". University of California, San Diego. 2002.
[29] Shamim, H., Ghulam, M. "An Emotion Recognition System for Mobile Applications", en *IEEE Access (Volume: 5)*. 2017.
[30] Su, Jiawei, Vasconcellos, Danilo, Kouichi, Sakurai. "One pixel attack for fooling deep neural networks", *Japan Science and Technology Agency*, Japan, 2017.
[31] Borovicka, T., Jirina, Marcel Jr., Kordik, Pavel, Jirina, Marcel. "Selecting Representative Data Sets", en *INTECH Open Access Publisher*. 2012.
[32] "Identify Data Patterns with Natural Language Processing and Machine Learning", http://www.dataversity.net/identify-data-patterns-natural-language-processing-machine-learning/, Accedido el 10 de Enero del 2018.
[33] Deng, Li, Yu, Dong. "Deep Learning Methods and Applications". *Foundations and Trends in Signal Processing*. 2012.
[34]"Introduction to Neural networks", http://www.explainthatstuff.com/introduction-to-neural-networks.html, Accedido el 10 de Enero del 2018.
[35] LeCun, Yann, Bengio, Yoshua, Hinton, Geoffrey. "Deep Learning Review", https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf, Accedido el 11 de Enero del 2018.
[36] Mohamed, Abdel, Dahl, George E. & Hinton, Geoffrey. "Acoustic modeling using deep belief networks", en *IEEE Transactions on Audio, Speech, and Language Processing*. pp. 14–22, 2012.
[37] Raina, Rajat, Madhavan, Anand, Ng, Andrew Y. "Large-scale deep unsupervised learning using graphics processors", en *ICML '09 Proceedings of the 26th Annual International Conference on Machine Learning*. pp. 873–880, 2009.
[38] Dahl, George E., Yu, Dong, Deng, Li, Acero, Alex. "Context-dependent pre-trained deep neural networks for large vocabulary speech recognition", en *IEEE Transactions on Audio, Speech, and Language Processing*. pp. 33–42, 2012.
[39] Bengio, Yoshua. "Learning Deep Architectures for AI", *Université de Montréal*, Canada, 2009.
[40] Buduma, Nikhil. "Deep Learning in a Nutshell – what it is, how it works, why care?",

https://www.kdnuggets.com/2015/01/deep-learning-explanation-what-how-why.html, Accedido el 4 de Enero del 2018.

[41] "Introduction to Deep Neural Networks". https://deeplearning4j.org/neuralnet-overview, Accedido el 4 de Enero del 2018.

[42] "How to do math for deep learning", https://github.com/llSourcell/how_to_do_math_for_deep_learning, Accedido el 4 de Enero del 2018.

[43] Fantz, Robert L. "The Origin of Form Perception", W. H. Freeman, pp. 66–73. 1961.

[44] "Detección de caras y análisis de expresiones faciales", http://alojamientos.us.es/gtocoma/pid/pid10/detecccioncaras.htm, Accedido el 12 de Diciembre del 2018.

[45] "Robust Face Recognition Technique under Varying Illumination", https://www.sciencedirect.com/science/article/pii/S1665642315300080, 2015, Accedido el 13 de Diciembre del 2018.

[46] Zhao, Wen Y., Chellappa, Rama, "Image based Face Recognition Issues and Methods", 2002.

[47] Zhao, Wen Y., Chellappa, Rama, Phillips P. J., Rosenfeld A., "Face Recognition: A Literature Survey", en *ACM Computing Surveys (CSUR) Volume 35 Issue 4*, Diciembre 2003.

[48] Sandeep, Kumar, Singh, Sukhwinder, Jagdish, Kumar, "A Study on Face Recognition Techniques with Age and Gender Classification", 2017.

[49] Rizvi, Qaim M., Agarwal, Bal G., Rizwan Beg. "A Review on Face Detection Methods.", en *Cloud Computing, Data Science & Engineering - Confluence, 2017 7th International Conference*. 2017.

[50] Turk, Matthew., Pentland, Alex., "Face recognition using eigenfaces", en *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91*. 1991.

[51] Shaikh, Izhar, "Emotion Recognition Using Scikit-learn & OpenCV", 2016, https://github.com/its-izhar/Emotion-Recognition-Using-SVMs, Accedido el 13 de Enero del 2018.

[52] Nefian, A., Hayes, M. "Face detection and recognition using hidden Markov models", en *Image Processing, 1998. ICIP 98.* 1998.

[53] Dixit, Bharati A., Gaikwad A.N. "Statistical Moments Based Facial Expression Analysis", en *Advance Computing Conference (IACC), 2015 IEEE International*. 2015.

[54] Freund, Y., Schapire, Robert E., "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", en *Journal of computer and system sciences Vol. 55*, pp. 119-139, 1997.
[55] Liu, Shuai, Wang, Wansen, "The application study of learner's face detection and location in the teaching network system based on emotion recognition", en *International Research Journal of Engineering and Technology*. 2010.
[56] An, Kwang H., Chung, Myung J., "Cognitive Face Analysis System for Future Interactive TV", en *IEEE Transactions on Consumer Electronics*. 2009.
[57] Naghsh, Ahmad R., Nilchi, Mohammad, Roshanzamir, "An Efficient Algorithm for Motion Detection Based Facial Expression Recognition using Optical Flow", 2008.
[58] Kumar, S., Thyaghrajan K.K., "Facial Expression Recognition with Auto-Illumination Correction", en *Green Computing, Communication and Conservation of Energy (ICGCE)*. 2013.
[59] Hinton, Geoffrey E., Sabour, S., Frost, N., "Dynamic Routing Between Capsules", Google Brain, Toronto. 2017.
[60] Hinton, Geoffrey E., Krizhevsky, A., Jaitly, N., Tieleman, T., Tang Y., "Does the Brain do Inverse Graphics?". http://helper.ipam.ucla.edu/publications/gss2012/gss2012_10754.pdf, Accedido el 14 de Enero del 2018.
[61] Autores anónimos (En revisión), "Matrix capsules with EM routing", 2017.
[62] Pechyonkin, M., "Understanding Hinton's Capsule Networks. Part II: How Capsules Work." 2017, https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-ii-how-capsules-work-153b6ade9f66, Accedido el 9 de Diciembre del 2017.
[63] Python official page. https://www.python.org/, Accedido el 15 de Enero del 2018.
[64] McConnell, Steve, "Code Complete", Microsoft Press, 2009.
[65] Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises". 2013.
[66] IDLE, https://docs.python.org/3/library/idle.html, Accedido el 15 de Enero del 2018.
[67] Tensorflow official page, https://www.tensorflow.org/, Accedido el 15 de Enero2018.

[68] Numpy python library, http://www.numpy.org/, Accedido el 15 de Enero del 2018.
[69] tqmd, Fast, Extensible Progress Meter, https://pypi.python.org/pypi/tqdm, Accedido el 15 de Enero del 2018.
[70] Keras: The Python Deep Learning library, https://keras.io/, Accedido el 15 de Enero del 2018.
[71] Soo, Sander, "Object detection using Haar-cascade Classifier", https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html, Accedido el 23 de Enero del 2018.
[72] Real Academia Española. "Diccionario de la lengua española". Edición n°22. 2001.
[73] Python Data Analysis Library, https://pandas.pydata.org/, Accedido el 27 de Enero del 2018.
[74] scikit-learn: Machine learning in Python, http://scikit-learn.org/stable/, Accedido el 27 de Enero del 2018.
[75] Google's AI Wizard Unveils a New Twist on Neural Networks, https://www.wired.com/story/googles-ai-wizard-unveils-a-new-twist-on-neural-networks/, Accedido el 27 de Enero del 2018.
[76] Extreme Programming: A gentle introduction, http://www.extremeprogramming.org/, Accedido el 27 de Enero del 2018.
[77] Brownlee, Jason. Supervised and Unsupervised Machine Learning Algorithms, https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/, Accedido el 29 de Enero del 2018.
[78] Matsugu, M., Mori, K., Mitari, Y., Kaneda, Y., "Subject independent facial expression recognition with robust face detection using a convolutional neural network", *Canon Research Center, 5-1, Morinosato-Wakamiya, Atsugi 243-0193, Japan*. 2003.
[79] Hubel, D., Wiesel, T., "Receptive fields and functional architecture of monkey striate cortex". Journal of Physiology, pp. 215–243, 1968.
[80] Convolutional Neural Networks (LeNet), http://deeplearning.net/tutorial/lenet.html, Accedido el 30 de Enero del 2018.
[81] Cires¸an, Dan C., Meier, U., Masci, J., Gambardella, Luca M., Schmidhuber J., "Flexible, High Performance Convolutional Neural Networks for Image Classification". en *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[82] Scherer, D., Muller, A., Behnke, S., "Evaluation of pooling operations in convolutional architectures for object recognition". en *International Conference on Artificial Neural Networks*, 2010.
[83] Programacion Extrema, http://www.chuidiang.org/ood/metodologia/extrema.php, Accedido el 20 de Febrero del 2018.
[84] Extreme programming for one, http://wiki.c2.com/?ExtremeProgrammingForOne, Accedido el 20 de Febrero del 2018.
[85] Maslow, A. "Hierarchy of Needs: A Theory of Human Motivation", Kindle Edition, 2011.
[86] Chimpanzees and monkeys have entered the stone age, http://www.bbc.com/earth/story/20150818-chimps-living-in-the-stone-age, Accedido el 20 de Febrero del 2019.
[87] Géron, A."Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems". Segunda edición, 2017.
[88] Challenges in Representation Learning: Facial Expression Recognition Challenge, https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data, Accedido el 17 de Diciembre del 2018.
[89] Opencv, https://opencv.org/, Accedido el 17 de Diciembre del 2018.
[90] Jupyter, https://jupyter.org/, Accedido el 17 de Diciembre del 2018.
[91] The CMU Multi-PIE Face Database, http://www.cs.cmu.edu/afs/cs/project/PIE/MultiPie/Multi-Pie/Home.html, Accedido el 10 de Octubre del 2018.
[92] The Japanese Female Facial Expression (JAFFE) Database, http://www.kasrl.org/jaffe.html, Accedido el 10 de Octubre del 2018.
[93] The Child Study Center at Rutgers University, https://www.childstudycenter-rutgers.com/, Accedido el 10 de Octubre.
[94] Manning, C., Raghavan, P, y Hinrich, I., "Introduction to information retrieval". Cambridge University Press, 2009.
[95] Martin, D., "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation". International Journal of Machine Learning Technology, 2011.

[96] N. Ye. "The Handbook of Data Mining. Human Factors and Ergonomics". Lawrence Erlbaum Associates, Incorporated, 2004.
[97] Wesner, J., MAE and MSE - which metric is better? https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d, Marzo 2016. Accedido el 2 de Enero del 2019.
[98] P. Uhe y M. Thatcher. "A spectral nudging method for the access 1.3 atmospheric model". Geoscientic Model Development Discussions, Agosto 2014.
[99] Arriaga, O., Ploger, P., y Valdenegro, M., "Real-time Convolutional Neural Networks for Emotion and Gender Classification", 2017.
[100] WIDER FACE: A Face Detection Benchmark, http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/, Accedido el 26 de Febrero del 2019
[101] Escobar, E., "Hacking Film: Why 24 Frames Per Second?", https://www.filmindependent.org/blog/hacking-film-24-frames-per-second/, Accedido el 26 de Febrero del 2019.
[102] Tan, Pang-Ning, Steinbach, M., "Introduction to Data Mining", Segunda Edición, 2005.
[103] A. T. Lopes, E. de Aguiar, A. F. De Souza, T. Oliveira-Santos, "Facial expression recognition with convolutional neural networks: coping with few data and the training sample order", 2017.
[104] Z. Wang, Q. Ruan, G. An, "Facial expression recognition using sparse local fisher discriminant analysis", Neurocomputing, 2016.

Anexos

A.1 Instalación de librerías para el desarrollo de la aplicación usando GPU-CUDA en Windows 10

A.1.1 Requisitos

- Python 3.6 (https://www.python.org/downloads/release/python-360/)

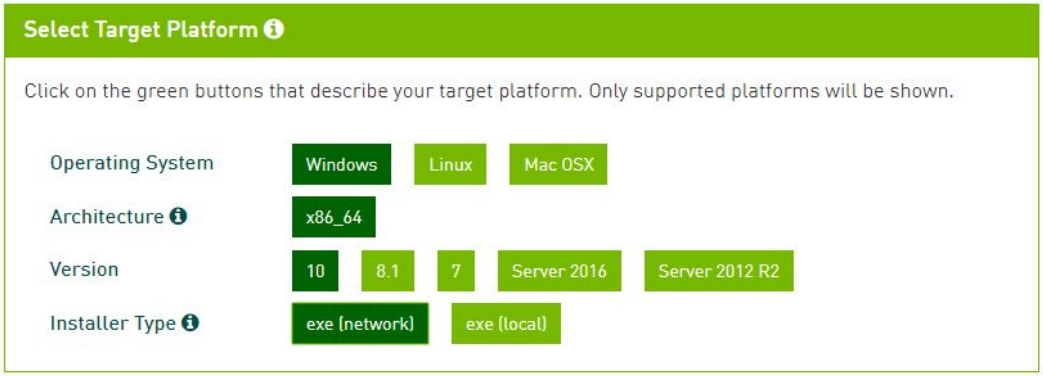
A.1.2 Pasos de instalación

Primero instalamos Keras y Tensorflow usando la consola con los comandos:

```
C:\Desktop> pip install keras==2.2.2
C:\Desktop> pip install tensorflow-gpu==1.11.0
```

Luego instalamos la librería Visual C++ Build Tools 2015 usando el link: https://visualstudio.microsoft.com/es/vs/older-downloads/?rr=https%3A%2F%2Fwww.tensorflow.org%2Finstall%2Fsource_windows.

Despues debemos instalar la version 9 de CUDA, encontrada en la pagina https://developer.nvidia.com/cuda-90-download-archive, colocamos las siguientes opciones y descargamos el instalador base y los parches correspondientes.



Ejecutamos el instalador base, y esperamos a que se termine de instalar. Luego vamos a la pagina https://developer.nvidia.com/rdp/cudnn-archive, y descargamos la versión más reciente de cuDNN para CUDA 9.

Una vez descargado extraemos el archivo y agregamos a la variable PATH la ruta del archivo extraído:

```
C:\Desktop> set path=<root>\cuda\tools\bin;%path%
```

Luego instalamos los parches descargados anteriormente en el orden en el que se encuentran en la página y listo. Para comprobar que todo funciona correctamente corremos el siguiente código en Python:

```
import tensorflow as tf
from keras import backend as K
print (K.tensorflow_backend._get_available_gpus())
```

A.2 Detección de rostros desde una cámara usando Opencv

A.2.1 Requisitos

Primero debemos descargar el filtro para la detección de caras frontales disponible en https://github.com/opencv/opencv/tree/master/data/haarcascades, en nuestro ejemplo usaremos “haarcascade_frontalface_default.xml”, este archivo debe estar en la misma carpeta que el siguiente código.

A.2.2 Código

```
import cv2

#Objeto para captura de video
video = cv2.VideoCapture(0)

#Ruta e inicializacion del filtro
cascPath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascPath)

while True:
    check, frame = video.read() #Obtener frame

    #Obtener caras en escala de grises
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    faces = faceCascade.detectMultiScale(gray, minSize=(36, 36))

    for (x, y, w, h) in faces:
        croppedFace = frame[y: y+h, x: x+w]

        #Dibujar rectangulo alrededor de la cara
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    cv2.imshow("Video", frame) #Mostrar frame

    key = cv2.waitKey(1) #Obtener tecla presionada
    if (key == ord('q')): #Tecla presionada == 'q' cerrar programa
        break

video.release() #Libera memoria reservada por la camara
```