ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN ALGORITMO PARA CLASIFICAR DE FORMA AUTOMÁTICA LAS REGIONES DE TEXTO, FIGURA Y TABLA DE LAS PÁGINAS DE ARTÍCULOS CIENTÍFICOS DIGITALES, MEDIANTE TÉCNICAS DE VISIÓN ARTIFICIAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

ERICK BOLÍVAR SANTOS MOROCHO

DIRECTOR: Ph.D. FELIPE LEONEL GRIJALVA ARÉVALO

CODIRECTOR: M.Sc. JOSÉ ADRIÁN ZAMBRANO MIRANDA

Quito, marzo 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Erick Bolívar Santos Morocho, bajo nuestra supervisión.

Ph.D. FELIPE LEONEL GRIJALVA ARÉVALO
DIRECTOR DEL TRABAJO DE TITULACIÓN

M.Sc. JOSÉ ADRIÁN ZAMBRANO MIRANDA
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Erick Bolívar Santos Morocho, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

ERICK BOLÍVAR SANTOS MOROCHO

DEDICATORIA

A mis padres y querido hermano
quienes con su amor y comprensión me han
acompañado en cada momento de mi vida.

A toda mi familia, para mi es un honor
dedicarles este trabajo como un gran
ejemplo a seguir para todos
mis primos y parientes.

A todas las personas que de una u otra
manera me guiaron con sabios consejos
y que siempre los llevaré en mi corazón.

Erick Santos

AGRADECIMIENTOS

Agradezco a Dios por sus bendiciones, su amor infinito, su paz inexplicable y por ser mi fortaleza en cada momento de mi vida.

A mis padres Victoria y Bolívar por ser mi fuente de inspiración, por guiarme con amor y sabiduría, por su inmenso sacrificio y dedicación por lo que no existen palabras para agradecer todo lo imposible que han hecho por mi bienestar.

A toda mi familia por su apoyo y sus sabios consejos que me han permitido desarrollarme como una mejor persona.

Al PhD. Felipe Grijalva por su paciencia, su valioso tiempo y ayuda en el desarrollo del presente proyecto.

A mis amigos y compañeros que he tenido la oportunidad de conocer y me han apoyado en cada momento de mi vida y con los que he compartido momentos inolvidables.

Finalmente, a la Escuela Politécnica Nacional por todas las oportunidades que me ha brindado para mi desarrollo profesional con un excelente nivel académico. Pero sobre todo por permitir mi desarrollo personal forjando valores como honestidad, perseverancia y altruismo.

ÍNDICE DE CONTENIDO

RESUMEN

El análisis de diseño de documentos (Document Layout Analysis) es un paso previo al procesamiento de los sistemas de interpretación de documentos (Document Understanding), el cual consiste en identificar y reconocer la estructura lógica y física de los documentos digitalizados con el objetivo de extraer su contenido y transformarlo en información estructurada que pueda aprovecharse posteriormente en la toma de decisiones.

En este sentido, todavía no se ha desarrollado un algoritmo universal de análisis de documentos que se ajuste a todos los tipos de diseños o que satisfaga todos los objetivos de análisis, por lo que se presenta un enfoque novedoso para identificar regiones de interés en artículos científicos utilizando técnicas de aprendizaje profundo y visión artificial. Este enfoque emplea aprendizaje automático supervisado con redes neuronales convolucionales para etiquetar regiones de interés de las páginas de un artículo científico en las categorías de texto, tabla y figuras a través de los espectrogramas provenientes de los histogramas de intensidad horizontal y vertical de las regiones. Posteriormente este sistema identifica líneas de ecuaciones en regiones de texto usando la técnica Bolsa de palabras visuales (Bag of Visual Words) con momentos de Zernike.

Finalmente, los algoritmos fueron entrenados y validados con la técnica de validación cruzada de K iteraciones con una base de datos de páginas de documentos previamente segmentados en regiones de interés y evaluados en términos de exactitud y precisión, así como también con el análisis de curva ROC (Receiver Operating Characteristic) y la métrica AUC (Area Under the ROC Curve).

PALABRAS CLAVE: Aprendizaje Automático, Análisis de diseño de documentos, Redes Neuronales Convolucionales, Bolsa de palabras visuales, Momentos de Zernike, ROC

ABSTRACT

Document Layout Analysis is a previous step to Document Understanding, which consists of identifying and recognizing the logical and physical structure of digitized documents in order to extract their components and transform them into structured information that can be used later in decision-making.

In this sense, a universal algorithm for Document Layout Analysis that fits all types of document layouts or satisfies all analysis objectives has not yet been developed, therefore a novel approach is presented to identify regions in scientific articles using Deep Learning and Computer Vision. This approach is a supervised machine learning system with Convolutional Neural Networks to label regions of interest of the pages of a scientific article in the categories of text, table and figure through the spectrograms from the horizontal and vertical intensity histograms of the regions. Later the system identifies lines of equations in text regions using the Bag of Visual Words technique with Zernike moments.

Finally, the algorithms were trained and validated with the K-Fold Cross-Validation technique with a database of document pages previously segmented into regions of interest and evaluated in terms of accuracy and precision. In addition, the ROC (Receiver Operating Characteristic) curve analysis and the AUC (Area Under the ROC Curve) metric are presented to quantify the performance of the trained algorithms.

KEYWORDS: Machine Learning, Document Layout Analysis, Convolutional Neural Networks, Bag of Visual Words, Zernike moments, ROC curve

1. INTRODUCCIÓN

Los documentos físicos o electrónicos son medios de comunicación entre personas o instituciones. Se crean para anunciar conferencias o nuevos productos, para resumir acuerdos, para comprender apartados de contratos, o incluso de ley, para publicar resultados de investigación innovadores, o simplemente para informar, cuestionar y convencer a otros. Los documentos siempre están presentes en trabajos donde el ser humano genera grandes volúmenes de información y con la intención de ayudar a estos procesos, la pregunta es cómo hacer que los datos de los documentos sean útiles. Este tratamiento que se realiza sobre datos no estructurados se denomina interpretación de documentos (en inglés Document Understanding) y consiste en extraer información relevante a partir de un contenido, con el propósito de transformarlo en información estructurada que luego se puede aprovechar en varias aplicaciones [1, 2].

Un enfoque fundamental en la interpretación de documentos es el sistema de análisis de documentos (Document Layout Analysis), el cual involucra algoritmos y técnicas que se aplican a documentos para obtener una descripción legible digitalizada a partir de datos de píxeles, es decir el análisis del diseño del documento identifica partes clave de un documento, como títulos, resúmenes, secciones, numeración de páginas, figuras y tablas. Un producto de análisis de documentos muy conocido es el software de reconocimiento óptico de caracteres (OCR) que reconoce caracteres en un documento escaneado. Por otro lado, algunas imágenes de documentos son difíciles de convertir a formato de texto a la perfección, debido a varios problemas como la sangría, la tinta tenue, los estilos de escritura irregulares y similares. En este caso, la recuperación de imágenes de documentos (Document Image Retrieval) es una técnica alternativa para buscar archivos de imágenes mediante consultas de texto sin conversión de imagen a texto. Así el objetivo del sistema de análisis de documentos es reconocer los componentes de texto y gráficos presentes en imágenes de documentos y extraer la información deseada como lo haría un ser humano [3].

Actualmente el formato más amplio empleado para generar documentos es el formato Portable Document Format (PDF) y corresponden alrededor de 2,5 billones de los documentos disponibles en Internet [4]. Si bien estos documentos son convenientes para el consumo humano, el procesamiento automático de estos documentos es difícil, ya que es complicado comprender el diseño del documento y extraer información utilizando este formato. Técnicas de análisis de diseño geométrico basadas en una representación de imágenes del documento combinadas con métodos de reconocimiento óptico de caracteres se han

usado sobre archivos PDF para realizar análisis de documentos y recientemente se han considerado métodos de análisis de imágenes basados en el Aprendizaje Profundo (Deep Learning) [5].

Además es común que en diferentes campos del conocimiento (e.g. académico, legal, finanzas) se generen grandes cantidad de documentos digitales cuyo análisis manual puede ser una tarea lenta y compleja. Es por esto que ya existen plataformas como Amazon Textract, Google Cloud Document Understanding AI y Microsoft Cognitive Services basadas en Inteligencia Artificial para analizar y gestionar estos documentos automáticamente, lo cual es crucial para el entendimiento de la información y posterior toma de decisiones. Sin embargo, estas plataformas tienen sus limitaciones, como la necesidad de crear reglas para extraer información de las tablas reconocidas por el sistema, además se concentran principalmente en tipos específicos de documentos, como facturas o formularios, donde el diseño del documento juega un papel relativamente importante porque pueden contener tablas, pero también grandes cantidades de texto no estructurado [6].

De esta manera, las técnicas de análisis de documentos disponibles están lejos de ser completamente automatizadas, generales y óptimas debido a que la gran diversidad de diseños de documentos y estructuras de tablas y figuras no hacen viable una clasificación de regiones basado en reglas [7]. En el presente proyecto se desarrolla un sistema de clasificación sobre documentos digitales en formato de imágenes, analizando la estructura visual de los documentos PDF mediante técnicas de visión artificial, con el propósito de que la clasificación de regiones no dependa del origen de creación del documento PDF (e.g.Latex o Word). Además, se plantea el uso de redes neuronales convolucionales, por la ventaja frente a redes neuronales convencionales en la capacidad de extracción de características y clasificación de imágenes.

1.1. OBJETIVO GENERAL

Desarrollar un algoritmo para clasificar de forma automática las regiones de texto, figura y tabla de las páginas de artículos científicos digitales, mediante técnicas de visión artificial.

1.2. OBJETIVO ESPECÍFICOS

- Estudiar los fundamentos del aprendizaje automático y las redes neuronales convolucionales.
- Transformar las regiones de texto, figura y tabla a una representación intermedia compatible con la entrada de tamaño fijo de una red neuronal convolucional.
- Proponer y entrenar una arquitectura de red neuronal convolucional para clasificar las representaciones intermedias de las regiones de texto, figura y tabla.
- Medir el desempeño del clasificador en términos de exactitud.

1.3. ALCANCE

En este proyecto de titulación se plantea un sistema para clasificación de regiones de texto, tabla, figura e identificación de ecuaciones dentro de regiones de texto mediante algoritmos de aprendizaje automático supervisado cuyo esquema general se muestra en la Figura 1.1. De este modo, las imágenes de las regiones, que son de tamaño variable, serán primero transformadas a una representación intermedia compatible con el tamaño de entrada fijo de una red neuronal sin perder sus características distintivas. Luego, las representaciones intermedias de las regiones se introducirán en una red neuronal convolucional específicamente diseñada para este problema (i.e. no se usará arquitecturas pre-entrenadas), la misma que las clasificará en tres grupos: texto, figura o tabla. Considerando que los algoritmos de aprendizaje supervisado necesitan un conjuntos de datos para los procesos de entrenamiento, validación y evaluación, se utilizará una base de datos previamente construida de imágenes etiquetadas de regiones de texto, figura y tabla. En este proyecto no se construirá la base datos ya que será provista por el proyecto de investigación PII-DETRI-2019-06 de la Escuela Politécnica Nacional. Por otro lado, este proyecto se enfoca exclusivamente en la clasificación y no en la segmentación de las regiones mencionadas, es decir, se considera que las regiones de entrada ya fueron correctamente segmentadas en etapas anteriores por

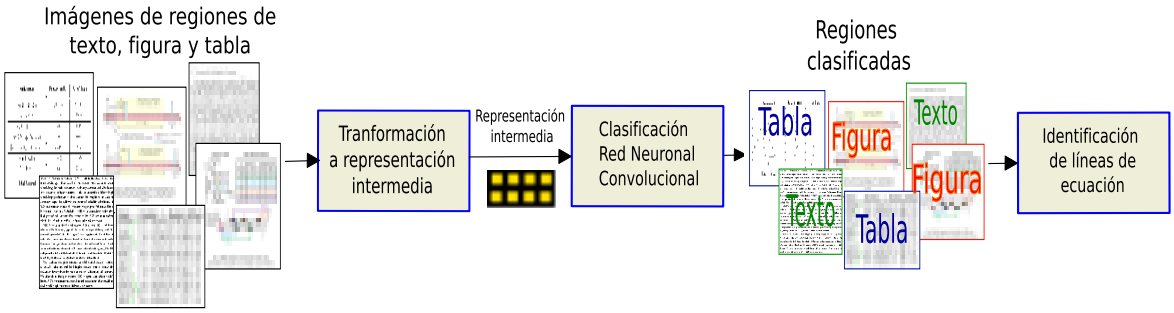


Figura 1.1: Esquema general del sistema de clasificación automática

lo que el esquema propuesto asume que las entradas son imágenes de regiones segmentadas de tabla, texto o figura. Posterior a la clasificación de regiones se añadirá una etapa que permita identificar líneas con ecuaciones dentro de las regiones que han sido clasificada como regiones de texto por la red neuronal convolucional. Finalmente, se calculará el desempeño del clasificador en términos de su exactitud. Para la implementación de estos algoritmos se usará el lenguaje de programación interpretado Python, por albergar librerías que se enfocan exclusivamente al campo de Aprendizaje Automático.

Dado que el presente proyecto no implica un desarrollo en hardware ni una interfaz para el usuario, no se generará un producto final demostrable.

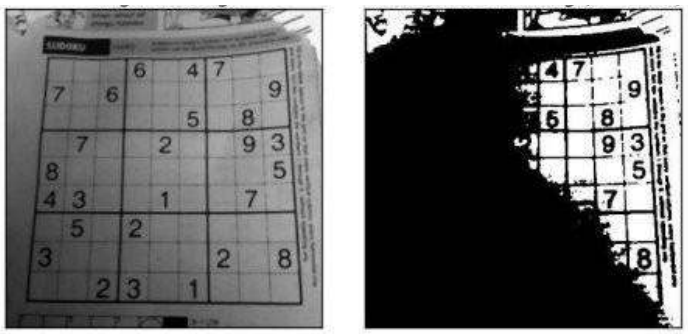
1.4. MARCO TEÓRICO

1.4.1. BINARIZACIÓN DE IMÁGENES

La binarización de imágenes es una tarea de preprocesamiento, muy útil para los sistemas de análisis de documentos, ya que convierte automáticamente las imágenes en escala de grises en un formato de dos niveles considerando un umbral, de tal manera que la información de primer plano está representada por píxeles negros y el fondo por píxeles blancos.

La selección de un buen umbral es a menudo un proceso de prueba y error. De esta manera se vuelve difícil encontrar umbral adecuado en los casos en que el contraste entre los píxeles de primer plano y el fondo es bajo. Por ejemplo cuando se requiere binarizar regiones de texto impreso sobre un fondo gris o cuando las líneas de texto son muy delgados, al no considerar un adecuado umbral el fondo podría filtrarse en píxeles de texto durante la digitalización. Este mismo resultado se podría presentar cuando una página no está iluminada uniformemente durante la captura de datos (Ver Figura 1.2) [8].

Para mitigar el problema de escoger un adecuado valor de umbral, se han desarrollado varios métodos con distintos enfoques, en los que podemos encontrar técnicas que modelan



a) Imagen original

b) Binarización con umbral global (v = 127)

Figura 1.2: Binarización de una imagen con un umbral global [9]

los píxeles de fondo y primer plano como muestras extraídas de distribuciones estadísticas y técnicas basadas en umbrales que varían espacialmente, que se conocen como umbrales adaptativos. Así también, se puede especificar múltiples umbrales, de modo que una banda de valores de intensidad se puede establecer en blanco(píxeles de primer plano) mientras que todo lo demás se establece en negro(píxeles de fondo). Otra variante común es establecer en negro todos los píxeles correspondientes al fondo, manteniendo los píxeles de primer plano en su color o intensidad original (en lugar de forzarlos a blanco), con el objetivo de que determinada información no se pierda. De forma general se pueden clasificar el algoritmo de binarización entre métodos globales y locales. Los algoritmos globales calculan un umbral para toda la imagen, mientras que los algoritmos de umbral local calculan diferentes valores de umbral dependiendo de las regiones locales de la imagen [8].

1.4.1.1. Umbral basado en picos de histogramas

Un valor de umbral adecuado que permita separar el primer plano del fondo, puede ser deducido a partir de un histograma de intensidad de la imagen. La intensidad de los píxeles dentro de los objetos en primer plano debe ser claramente diferente de la intensidad de los píxeles en el fondo. Esta información de intensidad se puede observar en los picos que se manifiestan en los histogramas, es decir debe existir un valle razonablemente claro entre los picos del histograma relacionados con los objetos y el fondo. Si tales picos no existen, entonces es poco probable que un umbral simple produzca una buena segmentación [9]. En la Figura 1.3 se muestra la binarización utilizando un umbral basada en picos de histogramas.

1.4.1.2. Umbral con el método Otsu

El método de umbral de Otsu se utiliza para la decisión automática del nivel de binarización, según la forma del histograma. El método de umbral de Otsu implica iterar a través de todos

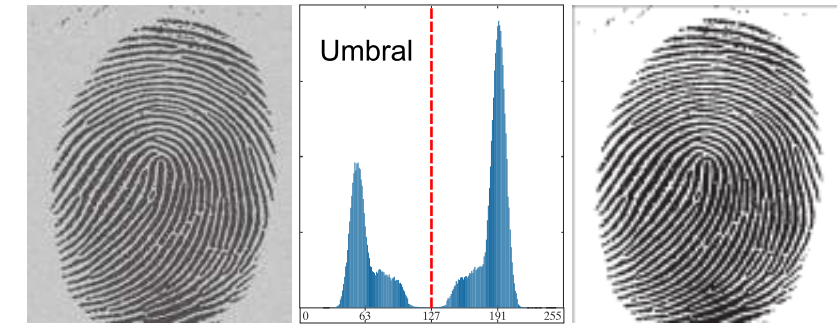


Figura 1.3: Binarización por picos de histogramas. a)Imagen original en escalas de grises. b) Histograma de la imagen original. c) Binarización con un umbral global (v=127) [9]

los posibles valores de umbral con el objetivo de maximizar la varianza entre clases. La varianza intraclase se puede definir como la suma ponderada de varianzas de las dos clases. La idea básica es que las clases debidamente definidas deben ser distintas con respecto a los valores de intensidad de sus píxeles y que un umbral que ofrezca la mejor separación entre clases en términos de sus valores de intensidad sería el mejor umbral (óptimo). En la Figura 1.4 se muestra el resultado de la binarización de una imagen obtenida con un microscopio óptico de células polimerizadas utilizando la técnica de umbral por picos de histogramas y el método de Otsu con el objetivo de segmentar las moléculas del fondo. La Figura 1.4(c) es el resultado de utilizar técnica de umbral por picos de histogramas. Debido a que el histograma no tiene valles distintos y la diferencia de intensidad entre el fondo y los objetos es pequeña, el algoritmo no logra la segmentación deseada. La Figura 1.4(d) muestra el resultado obtenido usando el método de Otsu. Este resultado obviamente es superior al de la Figura 1.4(c). El valor umbral calculado por la técnica de picos por histogramas fue 169, mientras que el umbral calculado por el método de Otsu fue 182, que está más cerca de las áreas más claras en la imagen que define las moléculas [9].

1.4.1.3. Umbral Adaptativo

Considerar técnicas de binarización con un umbral global en imágenes que tiene diferentes condiciones de iluminación en diferentes áreas, no se obtienen buenos resultados, es por esta razón que surgen los métodos de binarización con umbrales adaptativos. Los algoritmos de umbrales adaptativos determina el umbral de un píxel en función de una pequeña región a su alrededor. De esta manera se obtiene diferentes umbrales para diferentes regiones de la misma imagen, lo que brinda mejores resultados para imágenes con iluminación variable [10]. Entre los algoritmos de umbral adaptativo se puede mencionar a:

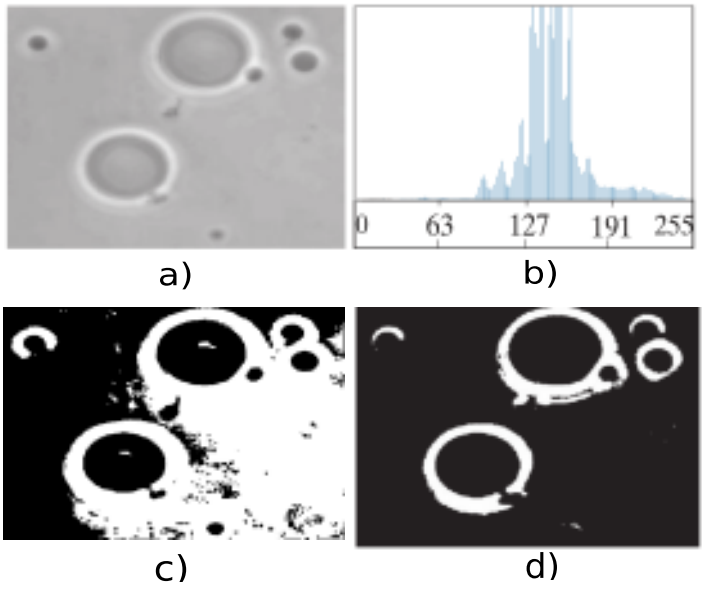


Figura 1.4: Comparación entre la técnica de umbral por picos de histogramas y por el método de Otsu. a) Imagen original en escala de grises. b) Histograma de la imagen original. c) Binarización por la técnica de picos de histograma. d) Binarización por el método de Otsu [9]

Umbral Adaptativo Promedio: Calcula un umbral local para cada píxel que depende del valor medio local y la desviación estándar local en la vecindad del píxel. Una constante determina cuánto del límite total del objeto de impresión se toma como parte del objeto dado. El tamaño del vecindario debe ser lo suficientemente pequeño para preservar el local y lo suficientemente grande para suprimir el ruido [10](Ver Figura 1.5(c)).

Umbral Adaptativo Gaussiano: Este algoritmo calcula el valor umbral como la suma ponderada de los valores de vecindad donde los pesos son una ventana gaussiana [10] (Ver Figura 1.5(d)).

1.4.2. ALGORITMO DE TRAZADO DE BORDES EN REGIONES BINARIAS

Un contorno en una imagen binarizada se define como un segmento que tiene un píxel de ancho y uno o más píxeles de largo, y un borde se define como un contorno ininterrumpido. Los contornos y los bordes proporcionan información muy importante para la representación de objetos y el reconocimiento de imágenes. Por ejemplo, se utilizan para separar objetos de sus fondos, calcular el tamaño de los objetos, clasificar formas y encontrar los puntos característicos de los objetos utilizando la longitud y la forma de sus píxeles de contorno [11]. Considerando que una imagen binarizada es una cuadrícula de píxeles donde se tiene un valor de ”1” lógico para los píxeles de la región de primer plano(píxeles de color negro) y un valor de ”0” lógico para los píxeles de fondo blanco, los puntos de transición se definen

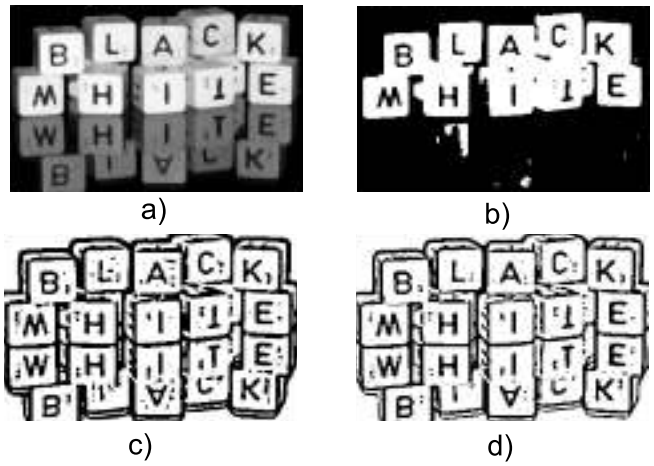


Figura 1.5: Binarización de una imagen por umbrales adaptativos. a) Imagen original en escala de grises. b) Binarización por picos de histogramas con umbral v=127. c) Binarización por umbral adaptativo promedio. d) Binarización por umbral adaptativo gaussiano [9]

como los puntos donde los valores de los píxeles cambian de ”1” a ”0” ó de ”0” a ”1” cuando se escanea la imagen en la dirección de izquierda a derecha. De esta manera los puntos de transición de los valores de píxeles de ”0” a ”1” determinan un borde izquierdo mientras que los puntos de transición desde los píxeles de valor ”1” a ”0” determinan un borde derecho [11]. Un par que consta de un borde izquierdo y un borde derecho se denomina dato de ejecución. Los puntos de contorno se definen como los puntos negros que pertenecen a una figura cuyos píxeles vecinos involucran al menos un píxel blanco. Un conjunto de puntos de contorno forman una línea de contorno [11]. Los algoritmos de contorno convencionales se pueden clasificar normalmente en tres tipos de la siguiente manera: seguimiento de píxeles, seguimiento de vértices y seguimiento basado en datos de ejecución .

1.4.2.1. Seguimiento de píxeles

El método de seguimiento de píxeles traza píxeles de contorno de una manera predefinida y luego guarda sus coordenadas en memoria de acuerdo con el orden de trazado. En la Figura 1.6, el algoritmo traza píxeles de contorno a lo largo de la dirección de las agujas del reloj desde el píxel actual, es decir, busca secuencialmente píxeles negros adyacentes del píxel actual usando un orden direccional relativo, como izquierda, frontal izquierda, frontal, frontal derecha, derecha, trasera derecha y trasera [11].

1.4.2.2. Seguimiento de vértices

El método de seguimiento de vértices traza los vértices de los píxeles de contorno que se encuentran en los bordes entre los píxeles de contorno y los píxeles de fondo. Su procedimiento es similar al del método de seguimiento de píxeles; sin embargo, traza las esquinas

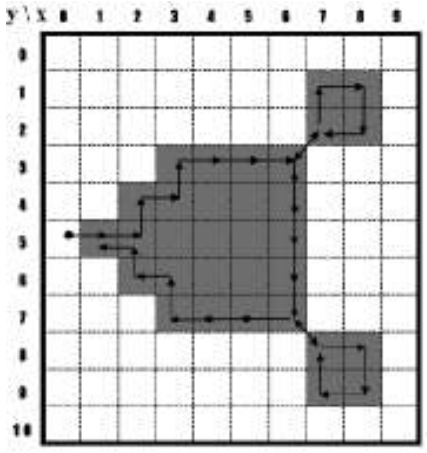


Figura 1.6: Algoritmo de trazo de bordes por seguimiento de píxeles [11]

de los píxeles de contorno y sus bordes conectados. Además, este tipo de seguimiento de borde considera solo los puntos de las esquinas de los píxeles del contorno para guardar la información del contorno trazado, y los datos se pueden comprimir reduciendo el número de puntos en una línea recta. Por ejemplo, en la Figura 1.7, se pueden resumir cinco puntos del contorno desde la coordenada ”s” $(2{,}5; 2{,}5)$ hasta la coordenada ”t” $(6{,}5; 2{,}5)$ tomando un sistema de coordenadas $(x; y)$ en el borde superior izquierdo de la imagen, almacenando solo las coordenadas ”s” y ”t”. Además, cuando se amplían las imágenes de contorno, el método de seguimiento de vértice puede proporcionar una imagen correcta porque los puntos trazados forman los límites del contorno [11].

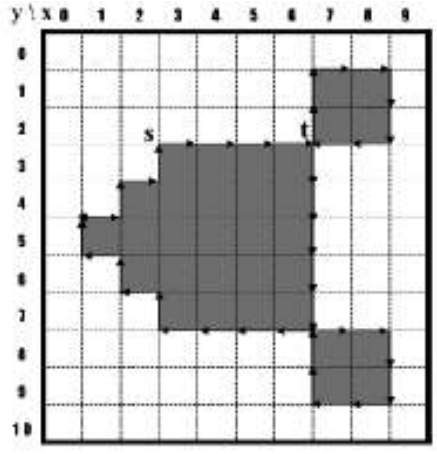


Figura 1.7: Algoritmo de trazo de bordes por seguimiento de vértices [11]

1.4.2.3. Seguimiento basado en datos de ejecución

Este método utiliza datos de ejecución en pares que constan de los bordes izquierdo y derecho de un objeto, que se obtienen utilizando líneas de exploración horizontal de izquierda

a derecha en una imagen. El objeto puede tener un contorno exterior y contornos interiores adicionales. Por lo tanto, hay cinco tipos de datos de ejecución: (i) borde izquierdo del contorno exterior con borde derecho del contorno exterior, (ii) borde izquierdo del contorno exterior con borde izquierdo del contorno interior, (iii) borde derecho del contorno interior con borde izquierdo del contorno interior, (iv) borde derecho del contorno interior con borde derecho del contorno exterior y (v) borde derecho del contorno exterior con borde izquierdo del contorno exterior. Para el seguimiento de contornos, el método de seguimiento basado en datos de ejecución construye una relación de seguimiento de ejecución entre los puntos de borde de dos líneas de exploración adyacentes [11]. En la Figura 1.8, la línea de exploración número 3 detecta el borde izquierdo 5 con el borde derecho 6, mientras que la línea de exploración número 4 detecta el borde izquierdo 7 y el borde derecho 8. Posteriormente, se generan las relaciones de seguimiento de ejecución entre el píxel 5 y píxel 7 y entre el píxel 8 y píxel 6.

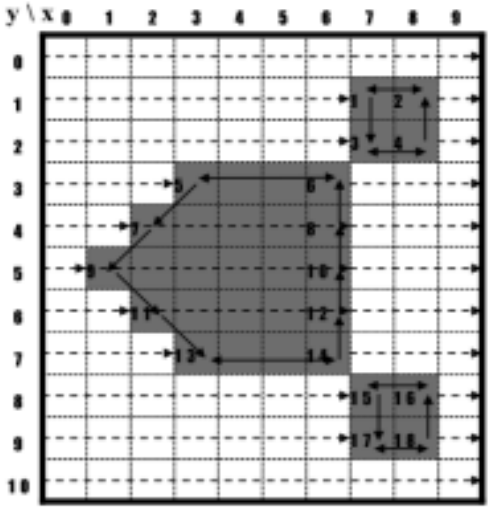


Figura 1.8: Algoritmo de trazo de bordes con el método de ejecución de datos [11]

El método de seguimiento de borde basado en datos de ejecución busca los puntos de borde y construye relaciones de seguimiento de ejecución entre puntos de borde. Por lo tanto, los resultados de seguimiento de ejecución trazados se cambian iterativamente mientras se actualizan los puntos de borde. Este método no es simple, pero es más rápido que los otros métodos para imágenes a gran escala, porque escanea todos los píxeles una vez; y no requiere que se realicen operaciones adicionales en los píxeles. Por lo tanto, es adecuado para aplicaciones basadas en imágenes a gran escala que involucran una gran cantidad de objetos, como el reconocimiento de documentos [11].

1.4.3. TRANSFORMADA DE FOURIER DE CORTO PLAZO (STFT)

La Transformada de Fourier de Corto Plazo es un método de análisis de señales en el dominio tiempo-frecuencia, es decir la STFT permite combinar la información local de un espectro de frecuencia instantánea con la información global del comportamiento temporal de una señal. Este método es útil cuando las características obtenidas en el dominio del tiempo y de la frecuencia de manera independiente no pueden proporcionar información completa sobre un señal. En este contexto, se observa que la STFT introduce la variable temporal sobre el análisis basado en Fourier para proporcionar una descripción adecuada de los cambios de contenido espectral en función del tiempo, ya que la Transformada de Fourier se convierte en una limitante cuando se realiza análisis de señales no estacionarias [12].

La transformada de Fourier de corto plazo (STFT) fue introducida por Dennis Gabor en el año 1946 y la idea principal del STFT es considerar solo una pequeña sección de la señal para el análisis en frecuencia. Con este fin, se desplaza una llamada función de ventana, que es una función diferente de cero por un corto período de tiempo. La señal original se multiplica con la función de ventana para producir una señal en ventana. Para obtener información de frecuencia en diferentes instancias de tiempo, se desplaza la función de ventana a través del tiempo y se calcula una transformada de Fourier para cada una de las señales de ventana resultantes [12]. De esta manera para señales continuas en el dominio del tiempo, la transformada de Fourier a corto plazo (STFT) es determinada a partir de una secuencia de Transformadas de Fourier de una señal que ha sido enventanada en el dominio del tiempo y se define matemáticamente como:

$$\mathcal{X}(\tau,\omega) = \int_{-\infty}^{\infty} x(t)g(t-\tau)e^{-j\omega t}\,dt \tag{1.1}$$

En la ecuación 1.1, se denota a $x\,(t)$ como una señal continua en el dominio del tiempo y a $g\,(t)$ como la función ventana.

Dado que la STFT genera valores complejos en general, a menudo se utiliza la definición de espectrograma para fines de visualización o para etapas de procesamiento adicionales. El espectrograma se puede definir como un gráfico de intensidad de la densidad espectral de potencia de la STFT, la cual es determinada como el cuadrado de su magnitud $|\mathcal{X}(\tau,\omega)|^2$.

En la Figura 1.9 se detalla el proceso de obtención de la STFT de una señal continua y de su espectrograma:

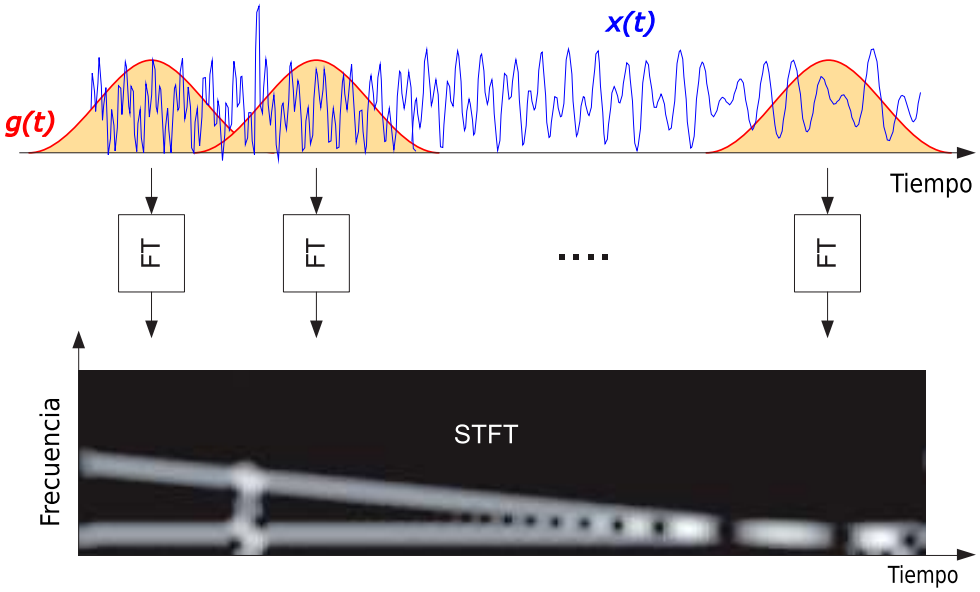


Figura 1.9: Transformada de Fourier a corto plazo (STFT) [13]

1.4.3.1. Resolución tiempo-frecuencia en STFT

En el análisis de tiempo-frecuencia existe una incertidumbre en el sentido de que no es posible conocer simultáneamente la ocurrencia instantánea de tiempo y la frecuencia de un componente de señal. Esta incertidumbre es paralela al Principio de incertidumbre de Heisenberg en mecánica cuántica con la posición y el momento de un electrón. Esta incertidumbre se debe a la relación inherente entre los dominios de tiempo y frecuencia, ya que una señal que es más corta en tiempo es más larga en frecuencia y viceversa [14]. De acuerdo al teorema Balian-Low, se establece que existe un límite para la cantidad de resolución que puede existir y se expresa cuantitativamente como:

$$(\triangle t)(\triangle f) \geq \frac{1}{4\pi} \tag{1.2}$$

donde $\triangle t$ y $\triangle f$ son las resoluciones de tiempo y frecuencia, las mismas que están definidas por:

$$\triangle t^2 = \frac{\int t^2 |x(t)|^2 dt}{\int |x(t)|^2 dt} \quad y \quad \triangle f^2 = \frac{\int f^2 |X(f)|^2 df}{\int |X(f)|^2 df} \tag{1.3}$$

De la ecuación 1.1 se considera como base elemental de la STFT a la siguiente expresión:

$$b_{w,\tau}(t) = g(t-\tau)e^{-j\omega t} \tag{1.4}$$

sobre el rango $t \in (-\infty : \infty)$ y $\omega \in (-\infty : \infty)$. La ecuación 1.4 puede entenderse como

cambios de tiempo y frecuencia de la función de ventana $g(t)$. La base STFT a menudo se ilustra mediante pequeños segmentos del plano tiempo-frecuencia, donde cada segmento representa un elemento particular de la base STFT, como se muetra en la Figura 1.10.

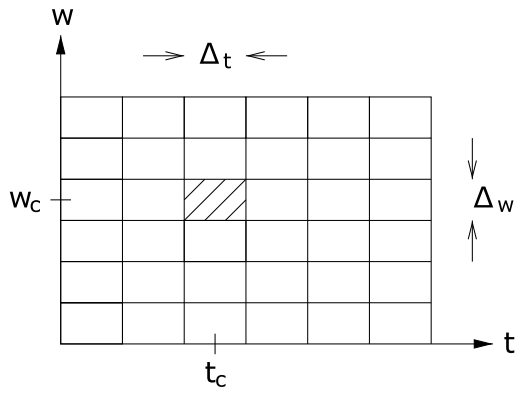


Figura 1.10: Segmentación del plano Tiempo-Frecuencia [15]

La altura y el ancho de un segmento representan los anchos espectrales y temporales de un elemento de la base STFT, respectivamente, y la posición de un segmento representa los centros espectrales y temporales del elemento de la base STFT. Además, se debe tener en cuenta que, si bien el diagrama de la Figura 1.10 sugiere que la STFT utiliza un conjunto discreto de cambios de tiempo-frecuencia, la base STFT se construye realmente a partir de cambios continuos de tiempo-frecuencia [15].
También se puede observar que al disminuir el ancho espectral $\triangle\omega$ a costa de un mayor ancho temporal $\triangle t$ ensanchando las formas de onda que componen la base STFT en el tiempo, el producto $\triangle t \triangle \omega$ se mantiene en un valor constante igual al planteado por el teorema de Balian-Low [15]. Este fenómeno se ilustra en la Figura 1.11.

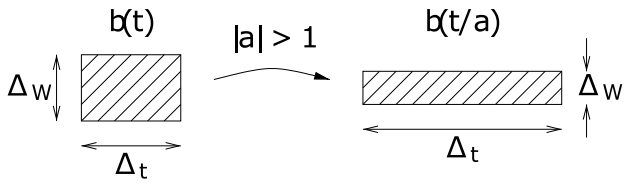


Figura 1.11: Altura y ancho de un segmento del plano tiempo-frecuencia [15]

De lo expuesto anteriormente, se puede afirmar lo siguiente con respecto a la resolución en el dominio Tiempo-Frecuencia para la Transformada de Fourier de corto plazo:

- Las resoluciones de tiempo y resoluciones de frecuencia de cada uno de los elementos de la base STFT serán iguales a las de la ventana $g(t)$. Es decir que cada uno de los segmento del plano tiempo-frecuencia tienen la misma dimensión [15].

- El uso de una ventana ancha dará una buena resolución de frecuencia pero una resolución de tiempo pobre, mientras que el uso de una ventana estrecha dará una buena resolución de tiempo pero una resolución de frecuencia pobre [15].

- La resolución combinada de tiempo-frecuencia de la base no está determinada por el ancho de la ventana sino por la forma de la ventana [15].

1.4.3.2. Definición formal de la STFT Discreta

Para la aplicación en procesamiento digital de señales, es necesario, extender la definición de STFT a aplicaciones con señales discretas. De esta manera el cálculo de STFT sobre una señal discreta implica tomar la Transformada de Fourier Discreta (DFT) en varias muestras de tiempo de la señal, obteniendo como resultado una STFT discreta tanto en tiempo como en frecuencia. Esta transformada se denomina STFT Discreta, sin embargo, es necesario diferenciarla de la STFT Discreta en tiempo, la misma que es solo discreta en el dominio del tiempo, pero continua en el dominio de la frecuencia. La STFT discreta en tiempo es particularmente útil como herramienta conceptual y analítica, mientras STFT discreta permite entender los detalles de cálculo computacional de los algoritmos que se basan en STFT [16]. La STFT discreta en tiempo está relacionado con la transformada de Fourier discreta en tiempo (DTFT), la cual es definida como:

$$\mathcal{X}(\omega) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\omega n} \tag{1.5}$$

Donde ω es una variable continua denominada frecuencia. La STFT discreta en tiempo de $x[n]$ es el conjunto de transformadas de Fourier discretas en tiempo (DTFT) correspondientes a diferentes secciones en tiempo de la señal $x[n]$. La sección de señal para el instante m_0 es obtenido multiplicando $x[n]$ con una secuencia desplazada $g[n - m_0]$. De esta forma la expresión para STFT discreta en tiempo al instante m_0 está dado por:

$$\mathcal{X}(m_0, \omega) = \sum_{n=-\infty}^{\infty} x[n]\, g[n - m_0]\, e^{-j\omega n,} \tag{1.6}$$

donde $w[n]$ se denota como la ventana de análisis y la secuencia $f_{m_0}[n] = x[n]\, g[n - m_0]$ es generalmente denominada sección de corta duración de $x[n]$ al instante m_0. Si se continúa calculado la DTFT para varias secciones de tiempo generadas al desplazar la ventana de análisis sobre la señal, se tiene un conjunto de DTFT que conforman la STFT discreta en tiempo. Entonces la STFT discreta en tiempo para cualquier instante se puede expresar

como:

$$\mathcal{X}(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]\, g[n-m]\, e^{-j\omega n,} \tag{1.7}$$

Además para la ecuación 1.7, se debe considerar que la ventana de análisis es seleccionada de tal manera que su longitud sea mucho menor al tamaño o duración de la señal $x[n]$.
Para el procesamiento digital, se utiliza la STFT discreta, la cual está relacionada con STFT discreta en tiempo, de la misma forma como la Transformada Discreta de Fourier (DFT) se relaciona con Transformada de Fourier Discreta en tiempo (DTFT). Es necesario considerar que DFT de un secuencia de duración finita $x[n]$ es obtenida al muestrear la Transformada de Fourier Discreta en tiempo sobre un periodo, es decir:

$$\mathcal{X}[k] = \mathcal{X}(\omega)\,|_{\omega=2\pi k/N} R_N[k] \tag{1.8}$$

donde N es el factor de muestreo de frecuencia y $R_N[k]$ es una secuencia rectangular de N puntos dado por:

$$R_N[k] = u[k] - u[k-N] \tag{1.9}$$

Siguiendo esta analogía, la STFT discreta es derivada de la STFT discreta en tiempo mediante la siguiente relación:

$$\mathcal{X}[m,k] = \mathcal{X}(m,\omega)\,|_{\omega=2\pi k/N} R_N[k] \tag{1.10}$$

En la ecuación 1.10, se ha muestreado la STFT discreta en tiempo con un intervalo de frecuencia de muestreo de $2\pi/N$, para obtener la STFT discreta en el dominio de la frecuencia. Sustituyendo la ecuación 1.7 en la ecuación 1.10, se deduce la relación entre STFT discreta y su correspondiente secuencia $x[n]$:

$$\mathcal{X}[m,k] = \sum_{n=-\infty}^{\infty} x[n]\, g[n-m]\, e^{-j2\pi kn/N} R_N[k] \tag{1.11}$$

De la ecuación 1.11, se tiene que $x : [0 : N-1] \to \mathbb{R}$ es una señal de tiempo discreto de longitud N obtenida por muestreo equidistante con respecto a una frecuencia de muestreo fija F_s y $g : [0 : L-1] \to \mathbb{R}$ es una función de ventana muestreada de longitud $L \in \mathbb{N}$. Además si se introduce un nuevo parámetro $H \in \mathbb{N}$ denominado tamaño de salto, el cual especifica en muestras y determina el tamaño del paso en el que la ventana se desplazará

a través de la señal, la STFT discreta queda definada como:

$$\mathcal{X}[m,k] := \sum_{n=0}^{N-1} x[n+mH]g[n]e^{-2j\pi kn/N} \tag{1.12}$$

En la ecuación 1.12, las variables m y k están comprendidos en los rangos $m \in [0:M]$ $k \in [0:K]$, respectivamente. El número $M = \frac{N-L}{H}$ es el índice que corresponde al último segmento de señal enventanado de de tal manera que el rango de tiempo de la ventana está completamente contenido en el rango de tiempo de la señal. Además, $K = L/2$ (L debe ser par) es el índice de frecuencia correspondiente a la frecuencia de Nyquist. El número complejo $\mathcal{X}(m,k)$ significa el coeficiente de Fourier k^{th} para la ventana m^{th}. De esta forma, para el espectro de cada segmento temporal m, se tiene un vector espectral de tamaño $K+1$ con coeficientes $\mathcal{X}(m,k)$ para $k \in [0:K]$. El cálculo de cada uno de estos vectores espectrales equivale a un DFT de tamaño N [17].

1.4.4. MOMENTOS DE ZERNIKE

Un problema importante en el análisis de patrones en imágenes es el reconocimiento automático de un objeto en una escena independientemente de su posición, tamaño y orientación, por lo que para estas aplicaciones se han empleado técnicas que permitan extraer un conjunto de características para la representación de imágenes y la reducción de datos. De esta manera el cálculo de momentos o funciones de momento se han empleado en numerosas aplicaciones de reconocimiento automático ya que permiten capturar información global sobre una imagen [18–20].

Los momentos de Zernike se implementan ampliamente en el reconocimiento de patrones porque permiten mapear una imagen sobre un conjunto de polinomios de Zernike complejos, con el objetivo de extraer información de las formas presentes en la imagen. Los momentos de Zernike fueron introducidos originalmente en la década de 1930 por el físico y ganador del premio Noble Fritz Zernike para describir las aberraciones ópticas [18].

La base de un momento Zernike complejo es un conjunto de polinomios ortogonales definidos sobre el interior del disco unitario $x^2+y^2=1$ que transformado al espacio de coordenadas polares es $0 \leq \rho \leq 1$ [19]. La ecuación 1.13 define el conjunto de polinomios de Zernike $V_{n,m}(x,y)$.

$$V_{n,m}(x,y) = V_{n,m}(\rho,\theta) = R_{n,m}(\rho)e^{jm\theta} \tag{1.13}$$

En la ecuación 1.13, $R_{m,n}$ corresponde a la parte real de los polinomios de Zernike (Polinomio Radial); $e^{jm\theta}$ es la parte compleja; ρ es la longitud del vector desde el origen hasta el

pixel (x, y); θ es el ángulo entre el vector ρ y el eje x en sentido antihorario [19]. Además n es un entero positivo y m es un entero positivo o negativo sujeto a la restricción de la ecuación 1.14:

$$m \in \{0, \pm 1, ..., \pm|n| \quad \text{tal que} \quad n - |m| \quad \text{par}\} \tag{1.14}$$

El polinomio radial $R_{n,m}$ se define en la ecuación 1.15.

$$R_{n,m}(\rho) = \sum_{s=0}^{n-|m|/2} (-1)^s \frac{(n-s)!}{s!\left(\frac{n+|m|}{2} - s\right)!\left(\frac{n-|m|}{2} - s\right)!} \rho^{n-2s} \tag{1.15}$$

donde $R_{n,m}(\rho) = R_{n,(-m)}(\rho)$. En la Figura 1.12 se ilustra los polinomios de Zernike hasta el quinto orden.

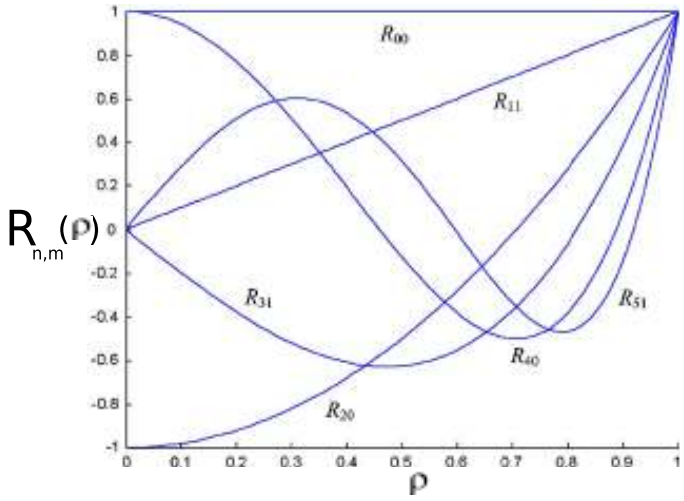


Figura 1.12: Polinomios radiales de Zernike de orden 0 hasta 5 con pocas repeticiones [20]

Si consideramos una imagen como una función continua de intensidad $f(\rho, \theta)$ en coordenadas polares, el momento complejo bidimensional de Zernike de orden n y con repetición m se define en la ecuación 1.16 [19].

$$Z_{n,m} = \frac{n+1}{\pi} \int_0^{2\pi} \int_0^1 f(\rho, \theta) V^*_{m.n}(\rho, \theta) \rho d\rho d\theta \tag{1.16}$$

Dado que los momentos de Zernike están definidos en coordenadas polares, una imagen requiere una transformación lineal de las coordenadas de la imagen a un dominio adecuado dentro de un círculo unitario como se observa en la Figura 1.13.

Las ventajas de usar polinomios de Zernike radican en su ortogonalidad ya que es posible representar las propiedades de una imagen sin redundancia o superposición de información entre los momentos Zernike [22].

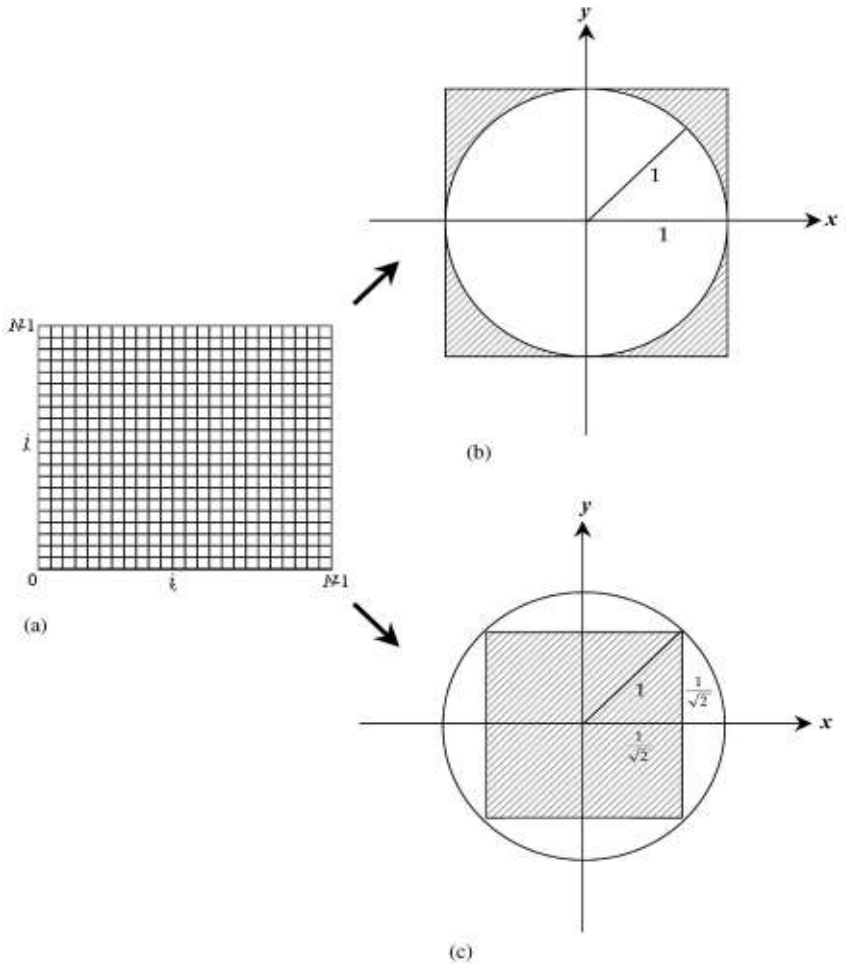


Figura 1.13: Dos esquemas de normalización de coordenadas para momentos radiales de Zernike: a) Sistema de coordenadas de imagen discreta $N \times N$. b) Normalización de coordenadas de imagen usando el mapeo $(0; N-1) \rightarrow (-1; +1)$. c) Normalización de coordenadas de imagen usando el mapeo$(0; N-1) \rightarrow (-1/\sqrt{2}; +1/\sqrt{2})$ [21]

Una de las aplicaciones que se tiene con los momentos de Zernike es la facilidad de reconstrucción de imágenes a partir de ellos ya que la propiedad de ortogonalidad permite separar la contribución individual de cada momento (su contenido de información) al proceso de reconstrucción. El número de detalles de la imagen reconstruida y el parecido con la imagen original dependen del nivel del orden del polinomio de Zernike utilizado en el proceso de reconstrucción. De esta manera los momentos de Zernike de bajo orden describen la forma general de una imagen, mientras que los momentos de Zernike de alto orden cubren aspectos más detallados de una imagen [18].

Además, los momentos de Zernike son números complejos, cuya magnitud tiene la propiedad invariancia a la rotación, es decir la rotación de una imagen no cambia las magnitudes de sus momentos Zernike. De esta manera las características definidas en los momentos de

Zernike son solo invariantes de rotación y para obtener la invariancia de escala y traslación, la imagen se somete primero a un proceso de normalización utilizando sus momentos regulares y posteriormente se extraen las características de Zernike obteniendo así invariancia de rotación [18].

Todas estas ventajas han hecho que los momentos Zernike se utilicen en la solución de problemas como: Reconocimiento de letras manuscritas [23, 24], Reconocimiento facial [25], Clasificación de masas benignas y malignas [26].

1.4.5. FUNDAMENTOS DE APRENDIZAJE AUTOMÁTICO

El Aprendizaje Automático (Machine Learning) es una rama de la Inteligencia Artificial que permite que los sistemas informáticos aprendan directamente de ejemplos, datos y experiencias. Al permitir que las computadoras realicen tareas específicas de manera inteligente, los sistemas de aprendizaje automático pueden llevar a cabo procesos complejos aprendiendo de los datos, en lugar de seguir reglas preprogramadas [27].

El principal objetivo práctico del Aprendizaje Automático consisten en generar predicciones precisas sobre conjunto de datos no conocidos por los algoritmos y en diseñar algoritmos eficientes y robustos para producir estas predicciones, incluso para problemas a gran escala. De esta manera, el proceso básico del Aprendizaje Automático es proporcionar datos de entrenamiento a un algoritmo de aprendizaje para generar un nuevo conjunto de reglas, basadas en inferencias de los datos proporcionados [27].

Los algoritmos de aprendizaje se han implementado con éxito en una variedad de aplicaciones, que incluyen:

- **Clasificación:** Asignación de categorías a un conjunto de elementos. Por ejemplo, la clasificación de documentos puede asignar elementos con categorías como política, negocios, deportes o clima, mientras que la clasificación de imágenes puede asignar elementos con categorías como paisaje, retrato o animal [27].

- **Regresión:** Predicción de un valor real sobre un conjunto de datos. Como ejemplo de regresión se puede mencionar la predicción de variaciones de variables económicas [27].

- **Ranking:** Este tarea consiste en ordenar un conjunto de datos de acuerdo algún criterio. Un ejemplo claro de Ranking es la búsqueda web donde se devuelven páginas web relevantes para una consulta en específico [27].

- **Clustering:** La agrupación en clústeres (grupos) se realiza a menudo para analizar

conjuntos de datos muy grandes. Por ejemplo, en el contexto del análisis de redes sociales, los algoritmos de agrupación intentan identificar ”comunidades” dentro de grandes grupos de personas [27].

- **Reducción de dimensionalidad:** Transforma una representación inicial de elementos en una representación de menor dimensión de estos elementos conservando algunas propiedades de la representación inicial [27].

Dado que el éxito de un algoritmo de aprendizaje depende de los datos utilizados, el Aprendizaje Automático está intrínsecamente relacionado con el análisis de datos y las estadísticas, por lo que las técnicas de aprendizaje son métodos basados en datos que combinan conceptos fundamentales en informática con ideas de estadística, probabilidad y optimización [27]. A continuación, se describe los escenarios comunes que se presenta en el aprendizaje automático. Estos escenarios difieren en los tipos de datos de entrenamiento disponibles, el orden y el método por el cual se reciben los datos de entrenamiento y los datos de prueba usados para evaluar el algoritmo de aprendizaje:

- **Aprendizaje Supervisado:** El algoritmo recibe un conjunto de ejemplos etiquetados como datos de entrenamiento y hace predicciones sobre datos que no han sido visibles por el algoritmo en la etapa de entrenamiento. Este es el escenario más común asociado con problemas de clasificación y regresión [27].

- **Aprendizaje no supervisado:** El algoritmo de aprendizaje recibe exclusivamente datos de entrenamiento sin etiquetar y hace predicciones con los datos que no han sido visibles por el algoritmo en la etapa de entrenamiento. Dado que, en general, no se dispone de ejemplos etiquetados, puede resultar difícil evaluar cuantitativamente el desempeño de un algoritmo. El clustering (agrupación) y la reducción de la dimensionalidad son ejemplos de problemas de aprendizaje no supervisados [27].

- **Aprendizaje reforzado:** el algoritmo interactúa con un entorno dinámico que proporciona retroalimentación en términos de recompensas y castigos. Las fases de entrenamiento y prueba también se entremezclan en el aprendizaje por refuerzo. Para recopilar información, el algoritmo interactúa activamente con el entorno y, en algunos casos, afecta el entorno, y recibe una recompensa inmediata por cada acción. El objetivo del algoritmo es maximizar su recompensa a lo largo de un curso de acciones e iteraciones con el entorno. Sin embargo, el entorno no proporciona una retroalimentación

de recompensa a largo plazo, por lo que el algoritmo se enfrenta al dilema de exploración versus explotación, ya que debe elegir entre explorar acciones desconocidas para obtener más información o explotar la información ya recopilada [27].

A continuación se detalla el funcionamiento de Redes Neuronales Convolucionales, Support Vector Machine y Gaussian Mixture Model que son los tres algoritmos de Aprendizaje Automático sobre los que se desarrolla el presente proyecto.

1.4.5.1. Aprendizaje Profundo

El Aprendizaje Profundo se desarrolló a partir de la red neuronal artificial y ahora es un campo predominante del Aprendizaje Automático. La investigación de la red neuronal artificial comenzó a partir de la década de 1940. McCulloch et al. propuso el modelo McCulloch-Pitts (MP) analizando y resumiendo las características de las neuronas. Hebb et al. propuso una teoría del ensamblaje celular para explicar la adaptación de la neurona cerebral durante el proceso de aprendizaje. Esta teoría tuvo una influencia importante en el desarrollo de las redes neuronales. Luego Rosenblatt et al. inventó el algoritmo del perceptrón. Este algoritmo es una especie de clasificador binario que pertenece al aprendizaje supervisado [28].
El algoritmo del perceptrón está inspiradas en el funcionamiento de la unidad computacional básica del cerebro humano, la neurona. Así como aproximadamente 86 mil millones de neuronas se pueden encontrar en el sistema nervioso humano y están conectadas con aproximadamente 10^{14} a 10^{15} sinapsis, la red neuronal artificial se estructura en el modelo del Perceptrón. La Figura 1.14 muestra una comparación de una neurona biológica y el modelo matemático de Perceptrón. Cada neurona recibe señales de entrada de sus dendritas y produce señales de salida a lo largo de su axón. El axón finalmente se ramifica y se conecta a través de sinapsis con las dendritas de otras neuronas. En el modelo computacional de una neurona, las señales que viajan a lo largo de los axones (x_0) interactúan de forma multiplicativa (w_0x_0) con las dendritas de la otra neurona en función de la fuerza sináptica en esa sinapsis (w_0). La idea es que las fortalezas sinápticas (w) se puedan aprender y de esta forma controlar la fuerza de la influencia de una neurona sobre otra. En el modelo básico, las dendritas llevan la señal al cuerpo celular donde se suman todas. Si la suma final está por encima de cierto umbral, la neurona puede dispararse y enviar una señal a lo largo de su axón. En el modelo computacional, se considera que los tiempos precisos en los que se envía esta señal no importan, y que solo la frecuencia de los disparos comunica información. Considerando esta interpretación del código de frecuencia, se puede modelar la tasa de activación de la neurona con una función de activación f, que representa la frecuencia de

los disparos a lo largo del axón. Históricamente, una opción común de función de activación es la función sigmoidea (σ), ya que toma una entrada de valor real (la fuerza de la señal después de la suma) y la modifica para que la salida sea respuesta binaria.

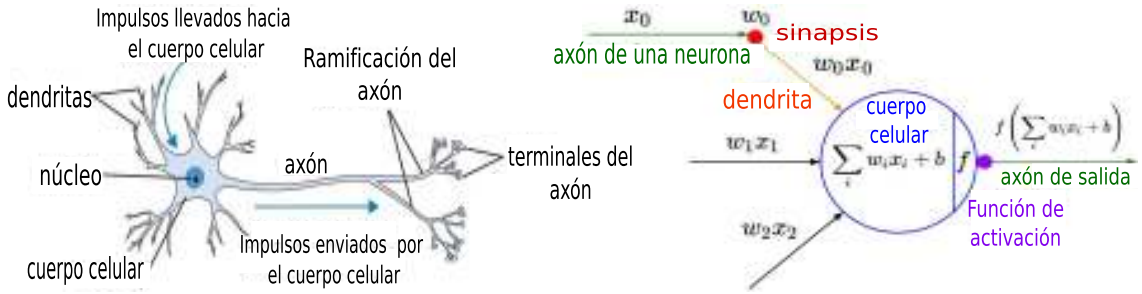


Figura 1.14: Comparación de una neurona biológica (izquierda) y el modelo matemático del Perceptrón (derecha) [29]

Las redes neuronales artificiales se pueden clasificar en tres tipos:

- Red perceptrón multicapa (Multilayer Perceptron MLP)
- Red neuronal convolucional (Convolutional Neural Network CNN)
- Red neuronal recurrente (Recurrent Neural Network RNN)

Red perceptrón multicapa (Multilayer Perceptron MLP): El perceptrón multicapa (MLP) es una red neuronal de retroalimentación que mapea conjuntos de entradas en conjuntos de salidas apropiadas. MLP tiene múltiples capas de nodos, donde cada capa está completamente conectada a la siguiente capa. Un MLP consta de tres tipos de capas: una capa de entrada, una capa de salida y una o más capas ocultas. La arquitectura específica de las redes neuronales multicapa se conoce como redes de alimentación hacia adelante, porque las capas sucesivas se alimentan entre sí en la dirección de avance desde la entrada hasta la salida. Si existe más de una capa oculta, la arquitectura se considera de aprendizaje profundo. Los pesos miden el grado de correlación entre los niveles de actividad de las neuronas que conectan [29]. En la Figura 1.15 se muestra un ejemplo de red perceptrón multicapa.

Red neuronal convolucional (Convolutional Neural Network CNN): Es una arquitectura de aprendizaje profundo inspirada en el mecanismo de percepción visual natural de los seres vivos. En 1959, Hubel y Wiesel descubrieron que las células de la corteza visual de los animales son responsables de detectar la luz en los campos receptivos [30]. Inspirado por este descubrimiento, Kunihiko Fukushima propuso el neocognitron en 1980, al que se

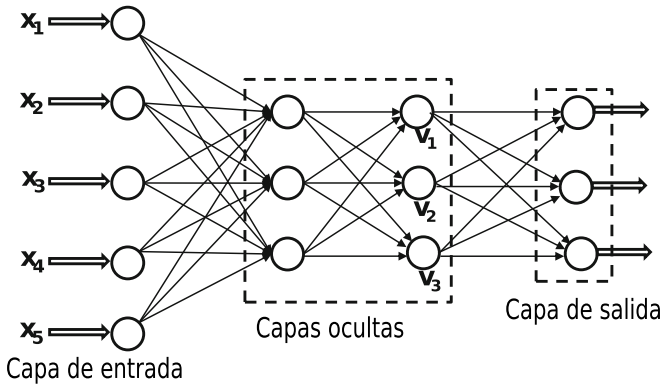


Figura 1.15: Un ejemplo de una red perceptrón multicapa con una capa de cinco entradas y una capa de salida de tres nodos [29]

podría considerar el antecesor de CNN [31]. La noción de profundidad de una sola capa en una red neuronal convolucional es distinta de la noción de profundidad en términos del número de capas. En la capa de entrada, estas características corresponden a los canales de color como RGB (es decir, rojo, verde, azul), y en los canales ocultos estas características representan mapas de características ocultas que codifican varios tipos de formas en la imagen. Las redes neuronales convolucionales se emplean ampliamente para el reconocimiento de imágenes, la detección, localización de objetos e incluso el procesamiento de texto [29]. De manera general la arquitectura está conformada por capas de convolución, submuestreo(pooling) y capas densamente conectadas como se muestra en la Figura 1.16. En la Sección 1.4.6 se describe con más detalle la arquitectura de una red neuronal convolucional.

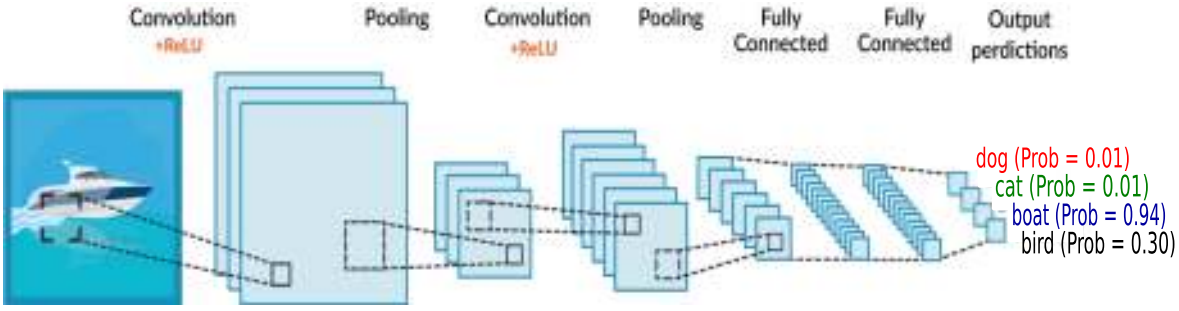


Figura 1.16: Arquitectura de una red neuronal convolucional [29]

Red neuronal recurrente (Recurrent Neural Network RNN): es un tipo especial de red neuronal artificial que tiene memoria que permite relacionar la información actual con un conocimiento previo mediante la utilización de un tipo especial de arquitectura en bucle, por lo que son redes con memoria. Es decir, RNN contiene al menos una conexión de retroalimentación, por lo que las activaciones pueden fluir en un bucle, lo que permite a las redes realizar

un procesamiento temporal y aprender secuencias, por ejemplo, realizar el reconocimiento de secuencias o predicción temporal. Estas redes en bucle se denominan recurrentes porque realizan las mismas operaciones y cálculos para cada elemento en una secuencia de datos de entrada. El modelo de memoria a largo plazo a corto plazo (LSTM) y la unidad recurrente cerrada (GRU) son las variaciones populares en los métodos de aprendizaje de representación basados en RNN [29].

Entrenamiento de una red neuronal arificial: El entrenamiento de redes neuronales o modelos de aprendizaje profundo involucran las siguientes etapas:

- **Inicialización de pesos:** El proceso de optimización requiere un punto de partida desde el cual comenzar las actualizaciones del modelo. De esta manera pequeños valores aleatorios iniciales se asignan a los pesos de las neuronas del modelo al comienzo del proceso de entrenamiento [29].

- **Propagación hacia adelante:** En esta fase, las entradas para una instancia de entrenamiento se alimentan a la red neuronal. Esto da como resultado una cascada de cálculos hacia adelante a través de las capas, utilizando el conjunto actual de ponderaciones. La salida final prevista se puede comparar con la de la instancia de entrenamiento y se calcula la derivada de la función de pérdida con respecto a la salida.

- **Función de pérdida:** La función que se usa para estimar el rendimiento de un modelo con un conjunto específico de ponderaciones en ejemplos del conjunto de datos de entrenamiento [29].

- **Descenso del gradiente:** es un algoritmo de optimización iterativo que se usa para encontrar los valores de los parámetros y pesos del modelo que minimizan la función de pérdida.

- **Retropropagación:** el objetivo de la retropropagación es cambiar los pesos de las neuronas para reducir al mínimo la función de pérdida. Si bien el aumento de la profundidad a menudo reduce el número de parámetros de la red, conduce a diferentes tipos de problemas prácticos. Propagar hacia atrás usando la regla de la cadena tiene sus inconvenientes en redes con un gran número de capas en términos de estabilidad de las actualizaciones. En particular, las actualizaciones en capas anteriores pueden ser muy pequeñas (*vanishing gradient*) o pueden ser cada vez más grandes (*exploding gradient*) en ciertos tipos de arquitecturas de redes neuronales [29].

- **Actualización de pesos:** Los pesos y parámetros se cambian a los valores óptimos de acuerdo con los resultados del algoritmo de retropropagación. Al actualizar el modelo, se deben utilizar varios ejemplos del conjunto de datos de entrenamiento para calcular la función de pérdida. Se pueden usar todos los ejemplos del conjunto de datos de entrenamiento, lo que puede ser apropiado para conjuntos de datos más pequeños. Alternativamente, se puede usar un solo ejemplo que puede ser apropiado para problemas donde los ejemplos se transmiten o donde los datos cambian con frecuencia. Se puede usar un enfoque híbrido en el que se puede elegir el número de ejemplos del conjunto de datos de entrenamiento y usarlo para estimar el gradiente de error. La elección del número de ejemplos se denomina tamaño de lote (batch size) [29].

- **Iteración hacia la convergencia:** Debido a que los pesos se actualizan cada vez que se efectúa el algoritmo de retropropagación, se requieren varias iteraciones para que la red aprenda. Después de cada iteración, el algoritmo de descenso del gradiente actualiza los pesos hacia una función de pérdida cada vez menor global. La cantidad de iteraciones necesarias para converger depende de la tasa de aprendizaje, los hiperparámetros de la red y el método de optimización utilizado [29].

1.4.5.2. Support Vector Machine(SVM)

Support Vector Machine es un algoritmo de aprendizaje automático supervisado empleado tanto para aplicaciones de clasificación como de regresión y se fundamenta en los principios de un Clasificador de Margen Máximo, el cual construye un hiperplano de dimensionalidad muy alta para separar las clases presentes en un conjunto de datos. De esta forma, los hiperplanos se consideran límites de decisión para clasificar elementos de un conjunto de datos en sus respectivas clases en un espacio multidimensional, de modo que el margen de separación entre las dos clases en los datos se amplía al máximo [32, 33].
La Figura 1.17 muestra un conjunto de datos cuya separación en clases se logra con un hiperplano lineal definida matemáticamente como $\mathcal{H} = \{\mathbf{x}|\mathbf{w}^T\mathbf{x} + b = 0\}$ y con una regla de clasificación binaria que asigna el valor de **1** a la primera clase y $-$**1** a la segunda clase, lo cual se define matemáticamente en la ecuación 1.17.

$$y_i(\mathbf{x_i}) = \text{sign}(\mathbf{w}^T\mathbf{x_i} + b) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x}_i + b > 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x}_i + b < 0 \end{cases} \tag{1.17}$$

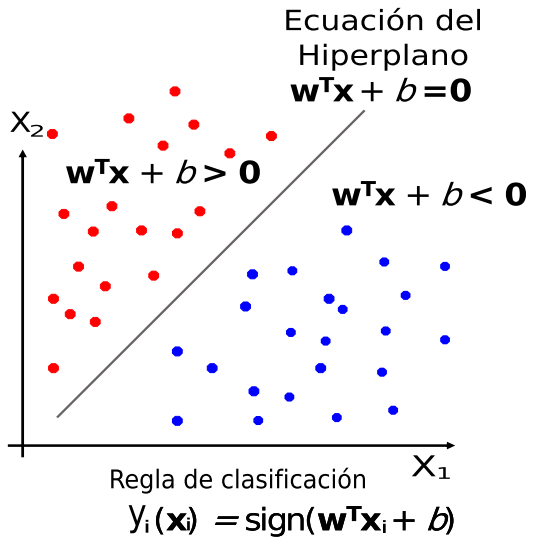


Figura 1.17: Separación de datos con un hiperplano lineal [33]

Al existir un gran número de hiperplanos que permiten separar adecuadamente los datos en clases. SVM construye un hiperplano óptimo, de modo que el margen de separación entre las dos clases en los datos se amplía al máximo. El margen de separación es la distancia entre los vectores de soporte hacia el hiperplano. Los vectores de soporte son los puntos del conjunto datos que se encuentran más cercanos al hiperplano. Estos puntos definirán mejor la línea de separación al calcular los márgenes (Ver Figura 1.18) [32, 33]. La distancia desde un vector de soporte al hiperplano se determina con la ecuación 1.18.

$$\gamma(\mathbf{w}, b) = \frac{|\mathbf{w}^T \mathbf{x_i} + b|}{\|\mathbf{w}\|} \tag{1.18}$$

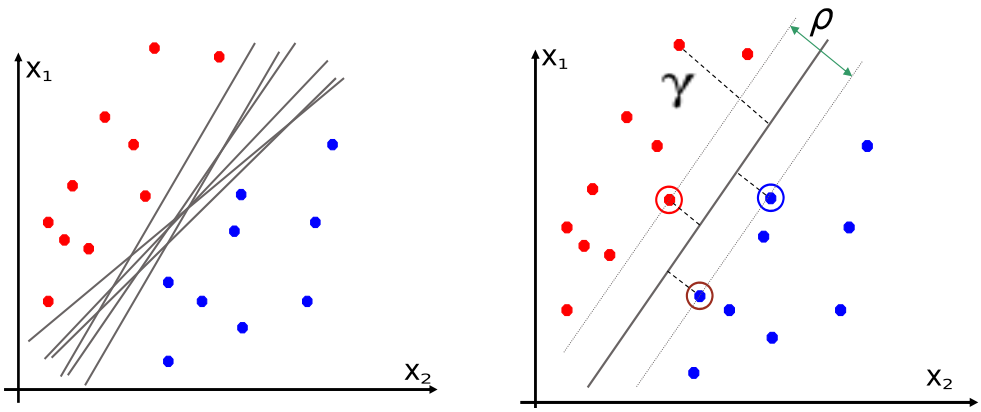


Figura 1.18: Selección del hiperplano óptimo con la maximización del margen de separación (ρ) a través de la distacia (γ) de los vectores soportes hacia el hiperplano [33]

La búsqueda del margen máximo que separa el hiperplano se resuelve como un problema de optimización restringido. El objetivo es maximizar el margen bajo las restricciones de que

todos los puntos del conjunto de datos deben estar en el lado correcto del hiperplano como se define en 1.19 [32, 33].

$$\underbrace{\max_{\mathbf{w},b}\ \gamma(\mathbf{w},b)}_{\text{Maximización del margen}} \quad = \quad \max_{\mathbf{w},b}\frac{1}{\|\mathbf{w}\|}.\min_{\mathbf{x}_i}\left|\mathbf{w}^T\mathbf{x}_i+b\right|; \quad \text{tal que} \quad \underbrace{\forall i\ y_i(\mathbf{w}^T x_i+b)\geq 0}_{\text{Región de decisión}} \tag{1.19}$$

Debido a que el hiperplano es invariante de escala, es posible fijar los valores de $(\mathbf{w}, b)$ de tal forma que se cumpla la condición de la ecuación 1.20.

$$\min_{\mathbf{x_i}}\left|\mathbf{w}^T\mathbf{x_i}+b\right|=1. \tag{1.20}$$

Así, el problema de optimización se reduce a la expresión 1.21.

$$\max_{\mathbf{w},b}\frac{1}{\|\mathbf{w}\|}=\min_{\mathbf{w},b}\|\mathbf{w}\|=\min_{\mathbf{w},b}\mathbf{w}^\top\mathbf{w}; \quad \text{tal que} \quad y_i(\mathbf{w}^T\mathbf{x}_i+b)\geq 1 \tag{1.21}$$

Si los datos son de baja dimensión, a menudo ocurre que no existe un hiperplano que separe el conjunto de datos en las clases. En este caso, no hay solución al problema de optimización mencionado anteriormente. Pero se puede solucionar permitiendo que las restricciones se violen muy levemente con la introducción de variables de holgura(ξ_i) y penalización (C) en la ecuación 1.21. La variable de holgura se introduce en el problema de optimización de dos maneras: primero, la variable de holgura determina el grado en que los puntos del conjunto de datos puede violar la restricción y en segundo lugar, al agregar la variable de holgura, el objetivo es simultáneamente maximizar el margen y minimizar el uso de las variables de holgura. Es decir, se permitirá que ciertos puntos del conjunto de datos de entrenamiento estén dentro del margen [32, 33]. De esta manera se desea que el número de puntos dentro del margen sea lo más pequeño posible(Ver Figura 1.19) [32, 33].

La ecuación 1.22 muestra el problema de optimización considerando las variables de holgura (ξ_i) y penalización (C).

$$\begin{gathered}\min_{\mathbf{w},b}\ \mathbf{w}^T\mathbf{w}+C\sum_{i=1}^{n}\xi_i \\ \text{tal que } \forall i\ y_i(\mathbf{w}^T\mathbf{x}_i+b)\geq 1-\xi_i \\ \forall i\ \xi_i\geq 0\end{gathered} \tag{1.22}$$

La variable de holgura ξ_i permite que los puntos x_i del conjunto de datos esté más cerca del hiperplano (o incluso en el lado equivocado), pero hay una penalización en la función objetivo por tal ”holgura”. Si C es muy grande, SVM se vuelve muy estricta e intenta que

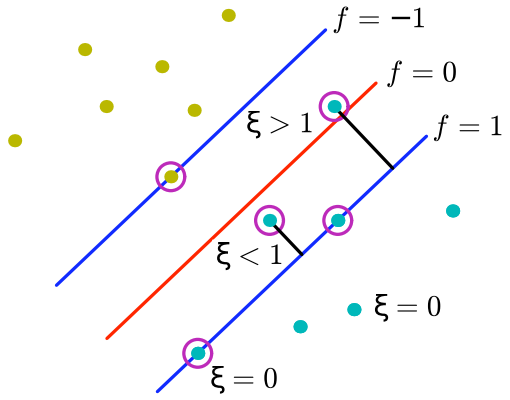


Figura 1.19: Selección del hiperplano con restricciones suaves. La variable de holgura toma valores tal que $\xi_i \geq 1$ para puntos mal clasificados y $0 \leq \xi_i \leq 1$ para puntos cerca al límite de decisión [33].

todos los puntos estén en el lado derecho del hiperplano. Si C es muy pequeño, SVM puede "sacrificar" algunos puntos para obtener una solución más simple (es decir, menor $||w||^2$) [32, 33].

SVM minimiza las variables de holgura considerando las restricciones, a través de la función de pérdida de bisagra, la cual se define matemáticamente con la ecuación 1.23.

$$\xi_i = \text{máx}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b), 0) = \begin{cases} 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i + b) < 1 \\ 0 & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{cases} \tag{1.23}$$

Con la función de pérdida de bisagra, cuando un punto de datos está clasificado correctamente y más allá del límite de decisión que el margen, la función de pérdida es cero. De esta manera, es insensible a los puntos correctamente clasificados lejos del límite. Pero cuando el punto está dentro del margen o clasificado incorrectamente, entonces la función de pérdida es simplemente la magnitud de la variable de holgura, por lo tanto, la función de pérdida de bisagra aumenta linealmente para los puntos mal clasificados [32, 33].

Si se reemplaza la ecuación 1.23 en la ecuación 1.22, se obtiene una versión sin restricciones del problema de optimización de SVM para la maximización del tamaño del margen y se detalla en la ecuación 1.24 como una función de pérdida con regularización.

$$\underset{\mathbf{w},b}{\text{mín}} \underbrace{\mathbf{w}^T\mathbf{w}}_{\text{Regularizador } l_2} + C \sum_{i=1}^{n} \underbrace{\text{máx}\left[1 - y_i(\mathbf{w}^T\mathbf{x_i} + b), 0\right]}_{\text{Función de pérdida de bisagra}} \tag{1.24}$$

La formulación de la ecuación 1.24 permite optimizar los parámetros (w, b) de SVM por medio del algoritmo de gradiente descendente.

También existen situaciones en donde no es posible encontrar un hiperplano que permita separar los conjuntos de datos en clases. En este caso las clases no son linealmente separables y para solucionar este problema se considera el truco del kernel, el mismo que consiste en trasladar los datos de entrada a un espacio de características altamente dimensional con el objetivo de encontrar un hiperplano para separar las clases [32, 33]. En la Figura 1.20 se observa cómo al añadir una dimensión nueva, se puede separar fácilmente las dos clases con una superficie de decisión.

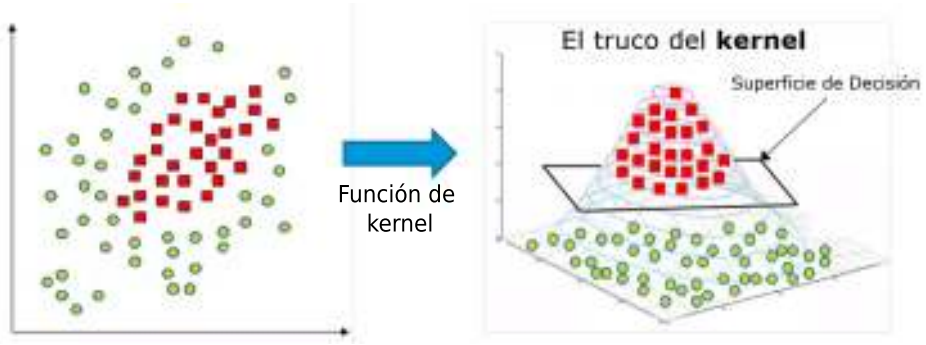


Figura 1.20: Transformación de un conjunto de datos a un espacio de características de alta dimensión [34]

La idea principal detrás de este método se basa en los llamados núcleos (kenel) o funciones del núcleo, que, bajo algunas condiciones técnicas de simetría y definición positiva, definen implícitamente un producto interno en un espacio de alta dimensión. Reemplazar el producto interno original en el espacio de entrada con núcleos definidos positivos extiende inmediatamente a SVM a una separación lineal en ese espacio de alta dimensión o, de manera equivalente, a una separación no lineal en el espacio de entrada [27]. Las funciones del núcleo populares que se emplean para SVM se detallan en la Tabla 1.1.

Tabla 1.1: Funciones Kernel empleados en SVM [27]

Tipo de SVM	**Función Kernel**
Función de base radial (RBF) o gaussiana	$K(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$
Lineal	$K(x_1, x_2) = x_1^\top x_2$
Polinómica	$K(x_1, x_2) = \left(x_1^\top x_2 + 1\right)^\rho$
Sigmoid	$K(x_1, x_2) = \tanh\left(\beta_0 x_1^\top x_2 + \beta_1\right)$

1.4.5.3. Gaussian Mixture Model (GMM)

Un modelo de mezcla Gaussiana (Gaussian Mixture Model) es un modelo probabilístico que supone que todos los puntos de datos se generan a partir de una mezcla de un número finito de distribuciones gaussianas con parámetros desconocidos [35]. La ecuación 1.25 muestra

como GMM describe la distribución probabilística de un espacio de características.

$$p(\mathbf{x}|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \sigma_k^2). \tag{1.25}$$

Donde:

- K es el número de mezclas o distribuciones
- π_k es el coeficiente de mezcla, que cumple $\sum_{k=1}^{K} \pi_k = 1$ y $\pi_k \geq 0$
- $\theta = \{(\pi_1, \mu_1, \sigma_1); ....; (\pi_K, \mu_K, \sigma_K)\}$ son los parámetros del modelo
- $\mathcal{N}(\mathbf{x}|\mu_k, \sigma_k^2)$ es la Distribución Gaussiana de dimensión D

Dado un conjunto de características $\mathbf{X} = \{x_1; ...; x_M\}$, los parámetros óptimos de GMM se aprenden mediante la Estimación de Máxima Verosimilitud (Maximum Likelihood Estimation) definida en la ecuación 1.26 [35].

$$\mathcal{L}(\theta) = ln(p(\mathbf{X}|\theta)) = \sum_{n=1}^{M} ln\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \sigma_k^2)\right) \tag{1.26}$$

La Estimación de Máxima Verosimilitud es un algoritmo estadístico que permite entrenar a Gaussian Mixture Model a partir de datos no etiquetados mediante un proceso iterativo, estimando los parámetros de las distribuciones de probabilidad para que se ajusten mejor a la densidad del conjunto de datos de entrenamiento [35]. La Estimación de Máxima Verosimilitud involucra los siguientes pasos:

- **Paso E:** se asumen componentes centrados aleatoriamente en puntos de datos y se determina para cada punto una probabilidad de ser generado por cada componente del modelo.
- **Paso M:** se ajusta los parámetros para maximizar la probabilidad de los datos dadas esas asignaciones. La repetición de este proceso garantiza la convergencia siempre hacia un óptimo local.

De forma general, GMM es considerado un método estadístico para modelar la estructura de agrupamiento subyacente en un conjunto de datos como un algoritmo de Aprendizaje Automático no Supervisado, convirtiéndose en una herramienta estadística útil para datos multivariados, especialmente porque la tecnología en desarrollo ahora permite el procesamiento de conjuntos de datos de alta dimensión. Además, GMM considera que los datos

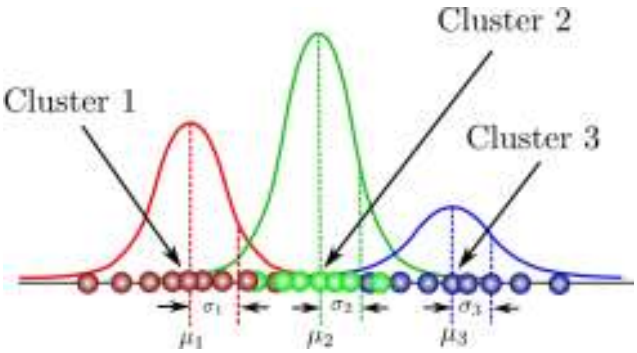


Figura 1.21: Ejemplo de densidad modelada por una mezcla de tres funciones de densidad de probabilidad gaussianas [36]

se obtienen de una mezcla de poblaciones subyacentes, cada una de las cuales corresponde a un grupo y este supuesto transforma el problema de agrupamiento (clustering) en un problema de estimación de parámetros, ya que los datos pueden modelarse como una mezcla de densidades de componentes gaussianos (ver Figura 1.21) [36]. Así un problema de agrupamiento con GMM implica optimizar los siguientes parámetros:

- La forma de densidades de los componentes (clusters) lo cual se refleja en una matriz de covarianza, la misma que representa las características geométricas como el volumen, la forma y la orientación de los grupos o componentes [36].
- El número de componentes [36].
- Función de optimización [36].
- Los criterios para determinar el modelo óptimo [36].

1.4.6. ARQUITECTURA DE REDES NEURONALES CONVOLUCIONALES

Una red neuronal convolucional es una subclase de red neuronal que se compone por diferentes tipos de capas llamadas capas convolucionales y cada una de ellas está formada por pequeños núcleos que permiten extraer características de alto nivel de una manera eficaz. El modelo estándar de una CNN está estructurada por un capa de entrada, capas convolucionales alternas, capas de agrupación o submuestreo, capas no lineales y finalmente se tiene capas densamente conectadas. [37]. En la Figura 1.22 se muestra la estructura estándar de una red neuronal convolucional.

Los datos de entrada de una red neuronal convolucional son aceptados en una matriz multidimensional, de esta manera los datos de entrada pueden ser píxeles de imagen, patrones, series de tiempo o señales de vídeo. Para las imágenes digitales, los valores de los píxeles que se almacenan en una matriz de números (Figura 1.23), alimentan a la red convolucional

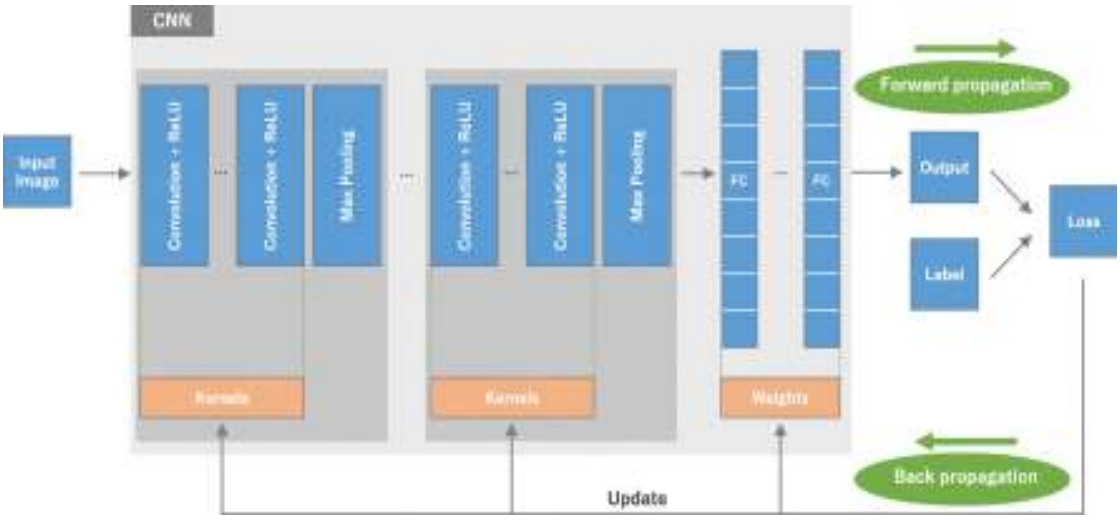


Figura 1.22: Arquitectura y etapas de entrenamiento de una red neuronal convolucional [38]

neuronal, la misma que a través de una pequeña cuadrícula de parámetros llamada kernel en cada uno de sus capas convolucionales realiza la extracción de características de forma optimizable [38].

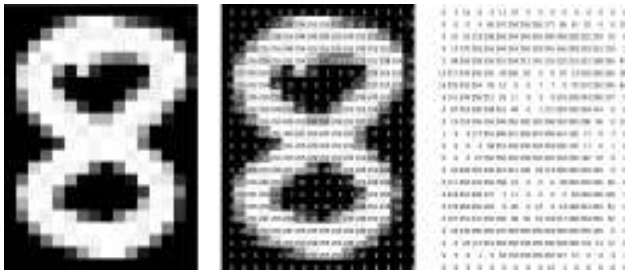


Figura 1.23: Una computadora ve una imagen como una matriz de números. La matriz de la derecha contiene números entre 0 y 255, cada uno de los cuales corresponde al brillo de píxeles de la imagen de la izquierda [38]

De esta manera el rendimiento de una arquitectura de CNN, se base en el aprendizaje de kernels y pesos particulares a través de una función de pérdida utilizando el algoritmo de propagación hacia adelante sobre un conjunto de datos de entrenamiento, y los parámetros en el kernel y los pesos obtenidos, se actualizan de acuerdo con el valor de pérdida mediante retropropagación con el algoritmo de optimización de descenso de gradiente [38].

1.4.6.1. Capa Convolucional

El propósito principal de una capa convolucional es la extracción de características de un tensor o matriz multidimensional que alimenta la entrada de la red neuronal convolucional, por lo que estas capas se componen de una serie de filtros o núcleos (kernel) optimizables que tienen como objetivo calcular un mapa de características a través de la operación con-

volución de matrices. Esta operación se efectúa entre el filtro o kernel y el tensor de entrada, es decir, se calcula el producto punto entre cada elemento del kernel y la sección del tensor que cubre el filtro obteniéndose como resultado un valor escalar. Este filtro repite el mismo proceso a lo largo de todo el tensor, desplazándose horizontal y verticalmente en un determinado número de pasos a la vez, denominado como stride. Además en este proceso se puede aplicar múltiples núcleos para formar un número arbitrario de mapas de características. En la Figura 1.24 se ilustra la obtención de un mapa de características con la operación de convolución entre un kernel y un tensor, de esta manera la operación de convolución entre cada núcleo y el tensor de entrada permite obtener un mapa de características cuya altura y ancho es reducido en comparación al tensor de entrada [38].

Entonces tres hiperparámetros clave que definen una capa convolucional son el tamaño, el número de núcleos y el valor de stride. El primero típicamente toma dimensiones de 3×3, pero a veces 5×5 o 7×7. El segundo es arbitrario y determina la profundidad de los mapas de características de salida. El tercero toma comúnmente el valor de 1; sin embargo, a veces se utiliza un stride superior a 1 para reducir la resolución de los mapas de características [38].

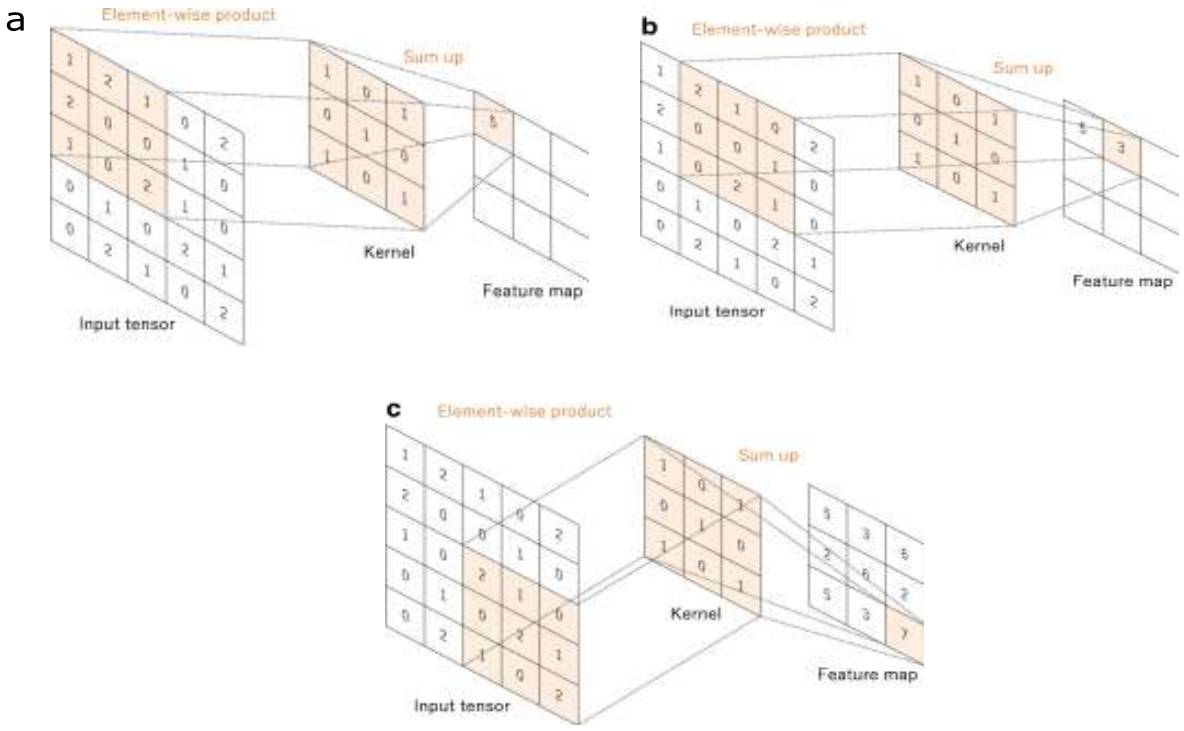


Figura 1.24: Ejemplo de operación de convolución con un tamaño de kernel de 3 × 3,con un valor de stride igual 1 [38]

Las capas convolucionales para la obtención del mapa de características, también emplean la técnica denominada Padding, lo cual permite que el centro del kernel se sobreponga con el elemento más externo del tensor de entrada. La técnica de Padding usualmente empleada

se denomina Zero-Padding, lo cual consiste en agregar filas y columnas de ceros a cada lado del tensor de entrada, de tal forma que sea posible calcular la operación de convolución entre el kernel y las regiones del tensor donde el centro del kernel coincide con los elementos más externos del tensor de entrada. Las arquitecturas modernas de CNN generalmente emplean Zero-Padding para mantener las dimensiones del mapa de características igual a las dimensiones del tensor de entrada con el fin de insertar posteriormente más capas. Sin Zero-Padding, cada mapa de características sucesivo se haría más pequeño después de la operación de convolución. En la Figura 1.25 se muestra la técnica de Zero-Padding en la operación de convolución [38].

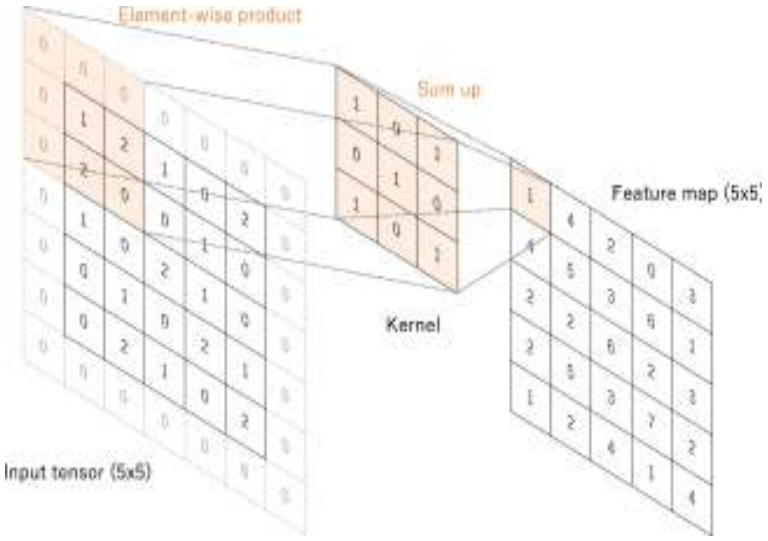


Figura 1.25: Ejemplo de operación de convolución con un tamaño de kernel de 3 × 3, con Zero-Padding y con un valor de stride igual 1 [38]

La característica clave que se obtiene de una operación de convolución es el peso compartido, es decir los núcleos se comparten en todas las posiciones del tensor de entrada. El peso compartido crea las siguientes características de las operaciones de convolución:

- Permite que los patrones de características locales extraídos por la traslación de los núcleos o filtros sean invariantes a medida que los núcleos viajan a través de todas las posiciones de una imagen y detectan patrones locales.
- El aprendizaje de jerarquías espaciales de los patrones de características mediante la reducción de resolución en junto con una operación de submuestreo, da como resultado la captura de un campo de visión cada vez más grande.
- Aumento de la eficiencia del modelo al reducir la cantidad de parámetros para aprender en comparación con las redes neuronales densamente conectadas.

El proceso de entrenamiento de un modelo de CNN con respecto a la capa de convolución es identificar los núcleos o filtros que se adapten mejor a una tarea determinada en función de un conjunto de datos de entrenamiento determinado. Los kernels son los únicos parámetros que se aprenden automáticamente durante el proceso de entrenamiento en la capa de convolución; por otro lado, el tamaño y el número de núcleos, el tamaño del Padding y el valor del stride son hiperparámetros que deben configurarse antes de que comience el proceso de entrenamiento.

Apilar varias capas convolucionales en modelos profundos permite que las capas cercanas a la entrada aprendan características de bajo nivel como bordes, esquinas, texturas y líneas, mientras que las capas más profundas del modelo aprenden características de alto orden o más abstractas, como formas u objetos específicos [39].

1.4.6.2. Unidad Lineal Rectificada (ReLU)

Si consideramos un red neuronal tradicional, para un nodo dado en una capa específica, las entradas a este nodo se multiplican por los pesos en un nodo y se suman. Este valor se conoce como la activación sumada del nodo. La activación sumada se transforma luego a través de una función de activación y define la salida específica o activación del nodo. La función de activación más simple se conoce como activación lineal, donde no se aplica ninguna transformación. Una red compuesta solo de funciones de activación lineal es muy fácil de entrenar, pero no puede aprender funciones de mapeo complejas. Las funciones de activación lineal todavía se utilizan en la capa de salida para redes que predicen una cantidad como se realiza en problemas de regresión. Sin embargo, se prefiere utilizar funciones de activación no lineales ya que permiten que los nodos aprendan estructuras más complejas en los datos. Tradicionalmente, dos funciones de activación no lineales ampliamente utilizadas son las funciones de activación sigmoidea y tangente hiperbólica [39].

Las funciones sigmoidea y tangente hiperbólica presentan un problema de saturación, es decir, que los valores grandes de activación sumada se ajustan a una salida de ”1” ó ”0” y los valores pequeños de activación sumada toman valores de ”-1” ó ”0” para tangente hiperbólica y sigmoidea respectivamente. Además, las funciones de activación no lineales son realmente sensibles a los cambios alrededor del punto medio de su entrada, como ”$0{,}5$” para sigmoidea y ”$0{,}0$” para tangente hiperbólica (Ver Figura 1.26). La sensibilidad y la saturación limitadas de las funciones de activación no lineal ocurren independientemente de si la activación sumada del nodo proporcionado como entrada contiene información útil o no. Una vez saturado, se convierte en un desafío para el algoritmo de aprendizaje seguir

adaptando los pesos para mejorar el rendimiento del modelo [39].

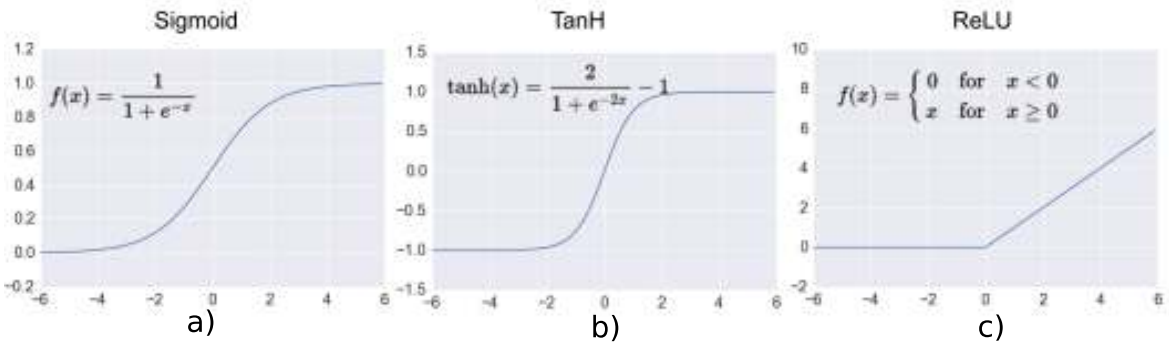


Figura 1.26: Funciones de activación: a) Función sigmoidea. b) Función tangente hiperbólica. c) Función RELU [39]

Las capas profundas en redes grandes que utilizan estas funciones de activación no lineales no reciben información útil del algoritmo de gradiente descendiente. El error se propaga nuevamente a través de la red y se usa para actualizar los pesos. La cantidad de error disminuye drásticamente con cada capa adicional a través de la cual se propaga, dada la derivada de la función de activación elegida. Esto se denomina problema del *Desvanecimiento del Gradiente (Vanishing Gradient Problem)* y evita que las redes profundas con múltiples capas aprendan de manera efectiva. Aunque el uso de funciones de activación no lineal permite que las redes neuronales aprendan funciones de mapeo complejas, impiden efectivamente que el algoritmo de aprendizaje funcione con redes profundas [39].
Las limitaciones presentadas por las funciones de activación sigmoidal y tangente hiperbólica en el rendimiento de las redes neuronales convolucionales son solventadas con la función de activación Unidad Lineal Rectificada (Ver Figura 1.26) o en inglés Rectified Linear Activation Unit (RELU) , por lo que esta función de activación es estándar para redes neuronales convolucionales profundas por las siguientes ventajas:

- Los cálculos efectuados por redes neuronales con funciones RELU son menos complejos que las funciones de activación sigmoidea e tangente hiperbólica porque no hay necesidad de trabajar con funciones exponenciales en las activaciones [40].
- Las redes neuronales con funciones de activación RELU convergen mucho más rápido que aquellas con funciones de activación que generan saturación en términos de tiempo de entrenamiento al aplicar gradiente descendente [40].
- La función RELU permite que una red obtenga fácilmente una representación dipersa. Es decir, la salida de la función RELU es ”0” cuando la entrada se encuentra en el

intervalo $x < 0$, lo que proporciona la dispersión en la activación de unidades neuronales y mejora la eficiencia del aprendizaje de datos. Cuando la entrada se ubica en $x > 0$, las características de los datos se pueden retener en gran medida, así como también acelerar el aprendizaje y simplificar el modelo [40].

- La derivada de la función RELU se mantiene en un valor constante igual "1", lo que puede evitar caer en una optimización local y resolver el efecto de desvanecimiento del gradiente que ocurre en las funciones de activación sigmoidea y tangente hiperbólica [40].

- Las redes neuronales profundas con funciones de activación RELU pueden alcanzar su mejor rendimiento sin requerir ningún entrenamiento previo sin supervisión en tareas puramente supervisadas con un gran conjunto de datos etiquetados [40].

En la Figura 1.27 se muestra la aplicación de la función de activación RELU sobre un mapa de características.

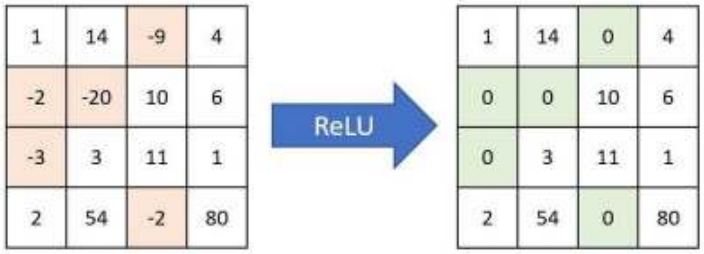


Figura 1.27: Aplicación de RELU sobre un mapa de características [39]

Sin embargo, la derivada de la función RELU cuando $x < 0$, es "0", por lo que esta función también provoca saturación para ese rango de valores. De esta manera es posible que los pesos relativos ya no se actualicen y eso lleve a la muerte de algunas unidades neuronales, lo que significa que estas unidades neuronales nunca se activarán. Otra desventaja de RELU es que el promedio de salida de las unidades es idénticamente positivo, lo que provocará un cambio de sesgo para las unidades en la siguiente capa. Los dos atributos anteriores tienen un impacto negativo en la convergencia de redes neuronales profundas [41]. Para solucionar estos inconvenientes se plantean las siguientes modificaciones sobre la función de activación RELU:

Leaky RELU (LRELU): Una de las desventajas de RELU es la de provocar un gradiente cero cuando las unidades neuronales no están activa. Esto puede hacer que las unidades que no se activen inicialmente nunca se activen ya que la optimización basada en gradientes

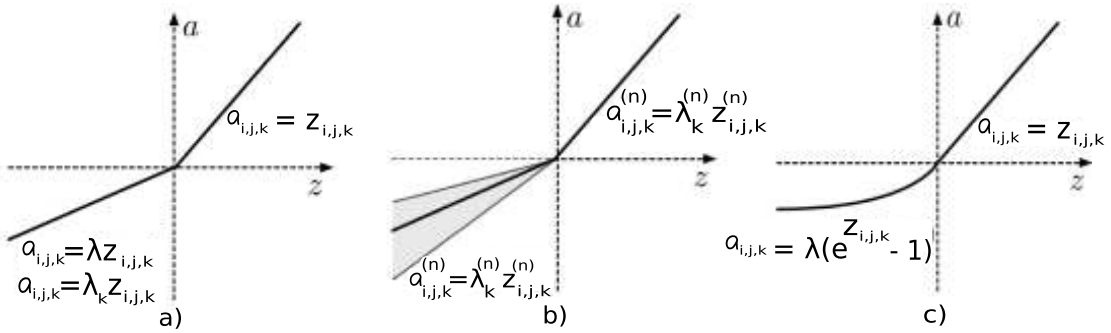


Figura 1.28: Comparación entre LRELU, PRELU, RRELU y ELU. a) **LRELU/ PRELU:** En LRELU λ es empíricamente predefinido. En PRELU λ_k se aprende de los datos de entrenamiento. b) **RRELU:** $\lambda_k^{(n)}$ es una variable aleatoria que se toma como muestra de una distribución uniforme dada en el entrenamiento y se mantiene fija en las pruebas. c) **ELU:** λ es empíricamente predefinido [42]

no ajustará sus pesos (Ver Figura 1.28). Para solventar este problema se ha planteado Leaky RELU, el cual se define como:

$$a_{i,j,k} = max(z_{i,j,k}, 0) + \lambda min(z_{i,j,k}, 0) \tag{1.27}$$

donde λ es un parámetro que varía en el rango $(0, 1)$. En comparación con RELU, Leaky RELU comprime la parte negativa en lugar de asignarla a cero constante, lo que permite un gradiente pequeño, distinto de cero, cuando la unidad no está activa [42].

Parametric RELU (PRELU): En lugar de utilizar un parámetro predefinido λ como en Leaky RELU como se indica en la ecuación 1.27, se plantea Parametric Rectified Linear Unit (PRELU) el cual aprende adaptativamente los parámetros de los rectificadores con el fin de mejorar la precisión (Ver Figura 1.28). Matemáticamente la función PRELU se define en la ecuación 1.28.

$$a_{i,j,k} = max(z_{i,j,k}, 0) + \lambda_k min(z_{i,j,k}, 0) \tag{1.28}$$

donde λ_k es el parámetro aprendido para el canal k-ésimo. Como PRELU solo introduce un número muy pequeño de parámetros adicionales, el número de parámetros adicionales es el mismo que el número de canales de toda la red, no hay riesgo adicional de sobreajuste y el costo computacional adicional es insignificante. También se puede entrenar simultáneamente con otros parámetros por retropropagación [42].

Randomized RELU (RRELU): Es otra variante de Leaky RELU, donde los parámetros de las partes negativas se muestrean aleatoriamente a partir de una distribución uniforme en

el entrenamiento y luego se fijan en la fase de validación (Ver Figura 1.28). Formalmente, la función RRELU se define como:

$$a_{i,j,k}^{(n)} = max(z_{i,j,k}^{(n)}, 0) + \lambda_k^{(n)} min(z_{i,j,k}^{(n)}, 0) \tag{1.29}$$

donde $z_{i,j,k}^{(n)}$ denota la entrada de la función de activación en la ubicación (i, j) en el canal k-ésimo del n-ésimo ejemplo, $\lambda_{i,j,k}^{(n)}$ denota su correspondiente parámetro muestreado y $a_{i,j,k}^{(n)}$ denota su salida correspondiente. RRELU podría reducir el sobreajuste debido a su naturaleza aleatoria [42].

Exponential Linear Unit (ELU): Esta función permite un aprendizaje más rápido de las redes neuronales profundas y conduce a una mayor precisión de clasificación. Al igual que RELU, LRELU y PRELU, ELU evita el problema del desvanecimiento del gradiente estableciendo la parte positiva en una función identidad. A diferencia de RELU, ELU tiene una parte negativa que es beneficiosa para el aprendizaje rápido. En comparación con LRELU y PRELU que también tienen partes negativas insaturadas, ELU emplea una función de saturación como parte negativa (Ver Figura 1.28). Esta función de saturación disminuye exponencialmente la variación de las unidades desactivadas, por lo que ELU es más resistente al ruido [42]. La función ELU se define formalmente en la ecuación 1.30:

$$a_{i,j,k} = max(z_{i,j,k}, 0) + min(\lambda(e^{z_{i,j,k}} - 1), 0) \tag{1.30}$$

donde λ es un parámetro predefinido para controlar el grado al que la función ELU se satura para entradas negativas.

1.4.6.3. Capa de agrupación, reducción o submuestreo (pooling)

Una capa de submuestreo proporciona una operación de reducción de la dimensionalidad de los mapas de características introduciendo una invariancia de traslación a pequeños cambios y distorsiones, reduciendo de esta forma el número de parámetros que la red neuronal convolucional debe aprender en capas posteriores a la capa de submuestreo. Cuando un modelo es invariante a la traslación, significa que no importa dónde se encuentra ubicado un objeto en una imagen; será reconocido de todos modos, por lo que la invariancia de traslación en un modelo beneficia enormemente su poder predictivo, ya que no se necesita proporcionar imágenes donde el objeto está precisamente en una posición deseada. Más bien, se debe proporcionar un conjunto masivo de imágenes que contengan el objeto con lo que se puede obtener un modelo predictivo con buen rendimiento [43].

Cabe señalar que no hay ningún parámetro que se pueda aprender en ninguna de las capas de submuestreo, mientras que el tamaño del filtro, el valor de stride y el empleo de Padding son hiperparámetros en las operaciones de submuestreo, similares a las operaciones de convolución [38]. Entre las operaciones de submuestreo que se puede realizar con los mapas de características se puede mencionar max pooling, average pooling, global max pooling y global average pooling.

Operación Max Pooling: En esta estrategia de submuestreo, se selecciona la activación con el valor máximo de todas las activaciones que se presentan en un campo rectangular de un mapa de características, como se muestra en la Figura 1.29.

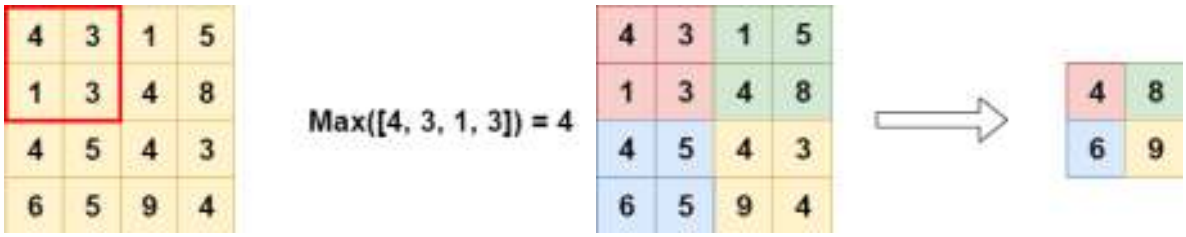


Figura 1.29: Operación de Max Pooling [39]

El mayor inconveniente de la operación de Max Pooling es que el operador de la agrupación considera solo el elemento máximo del área de agrupación e ignora otros elementos. Si la mayoría de los elementos en el área de agrupación fueran de gran magnitud, las características más exigentes desaparecen después de realizar la operación de agrupación máxima y esta situación conduce a resultados inaceptables debido a la pérdida de información [39].

Operación Average Pooling: En esta estrategia de agrupación, el mapa de características de entrada se divide en un conjunto de regiones rectangulares separadas y la salida se obtiene calculando el promedio de los valores agrupados en cada región rectangular, como se ilustra en la Figura 1.30.

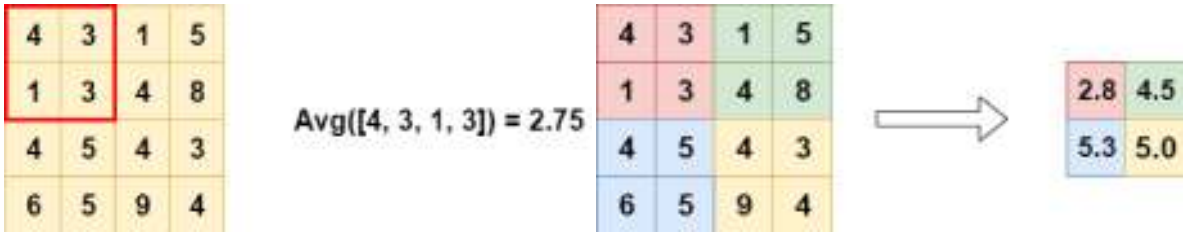


Figura 1.30: Operación de Average Pooling [39]

La operación Avarage Pooling es diferente de la operación Max Pooling en el sentido de que retiene mucha información sobre los elementos ”menos importantes” de un bloque o grupo.

Mientras que Max Pooling simplemente los descarta información eligiendo el valor máximo, Average Pooling los mezcla [39].

Operación Global Max Pooling: Esta técnica realiza la operación de Max Pooling sobre todo el mapa de característica. En la Figura 1.31 se muestra la operación de Global Max Pooling.

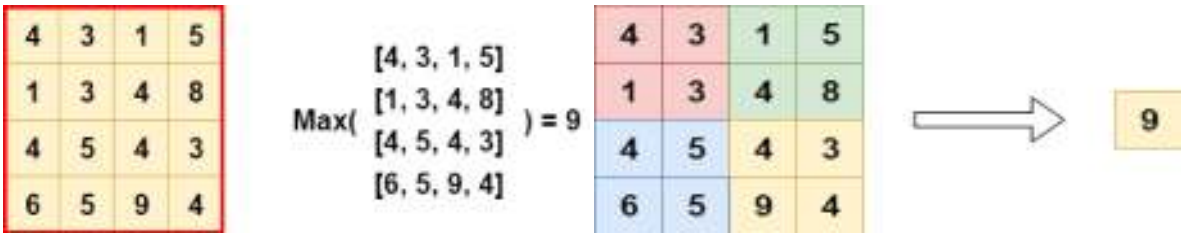


Figura 1.31: Operación de Global Max Pooling [39]

Operación Global Avarage Pooling: Esta técnica realiza la operación de Avarage Pooling considerando todo los valores del mapa de característica. En la Figura 1.32 se muestra la operación de Global Max Pooling.

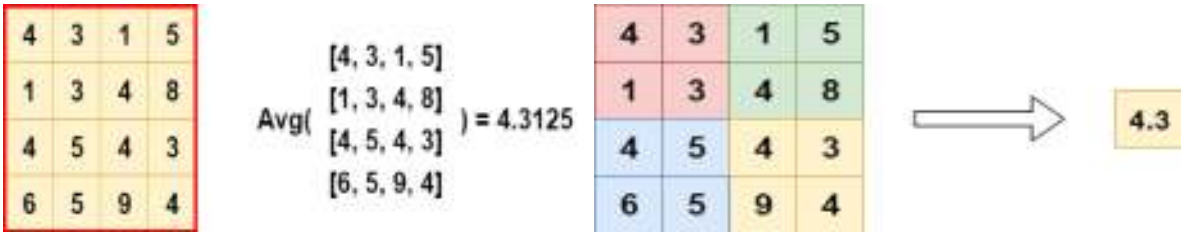


Figura 1.32: Operación de Global Avarage Pooling [39]

1.4.6.4. Capa de Normalización por lotes (Batch Normalization)

El entrenamiento de redes neuronales profundas se complica por el hecho de que la distribución de las entradas de cada capa cambia durante el entrenamiento, a medida que cambian los parámetros de las capas anteriores. Esto ralentiza el entrenamiento al requerir tasas de aprendizaje más bajas y una inicialización cuidadosa de los parámetros, y hace que sea notoriamente difícil entrenar modelos con funciones de activación no lineales que provocan saturación. De esta forma la normalización por lotes se convierte en una solución para ayudar a coordinar la actualización de múltiples capas en el modelo [44].

La normalización por lotes es una técnica para entrenar redes neuronales profundas que estandariza las entradas con respecto a una capa considerando un mini-lote. La estandarización se refiere a cambiar la escala de los datos para tener una media de cero y una desviación estándar de uno, de esta forma se tiene el efecto de estabilizar el proceso de

aprendizaje y reducir drásticamente el número de épocas de entrenamiento necesarias para entrenar redes profundas. Para las capas convolucionales, la normalización por lotes se produce después del cálculo de convolución y antes de la aplicación de la función de activación. Si el cálculo de convolución genera múltiples canales, se necesita llevar a cabo la normalización por lotes para cada una de las salidas de estos canales, y cada canal tiene un parámetro de escala independiente y un parámetro de desplazamiento, ambos escalares, como se muestra en la Figura 1.33. También la normalización por lotes puede tener un efecto de regularización, reduciendo el error al ajustar una función de forma adecuada en el conjunto de entrenamiento dado y evitar el sobreajuste [44].

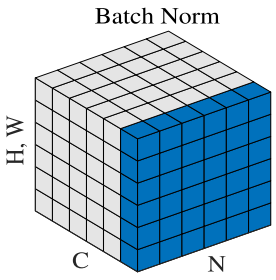


Figura 1.33: Mapa de características donde N es el eje del lote, C representa los canales, (H x W) son ejes espaciales. Los píxeles en azul están normalizados por la misma media y varianza [45].

La normalización de lotes afecta la salida de la capa de activación anterior al restar la media del lote y luego dividir por la desviación estándar del lote, como se detalla en el algoritmo 1. Dado que este desplazamiento o escalado de salidas mediante un parámetro inicializado aleatoriamente reduce la precisión de los pesos en la siguiente capa, se aplica un descenso de gradiente estocástico para eliminar esta normalización si la función de pérdida es demasiado alta [44].

El resultado final es que la normalización por lotes agrega dos parámetros entrenables adicionales a una capa: la salida normalizada que se multiplica por un parámetro $\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$, que corresponde la desviación estándar y el parámetro $\beta^{(k)} = \text{E}[x^{(k)}]$ que hace referencia a la media. Esta es la razón por la que la normalización de lotes funciona junto con los descensos de gradiente para que los datos se puedan ”desnormalizar” simplemente cambiando solo estos dos pesos para cada salida. Esto conduce a una menor pérdida de datos y una mayor estabilidad en la red al cambiar todos los demás pesos relevantes [44].

La normalización por lotes, se efectúa considerando que la activación de las unidades neuronales se ajustan a una distribución gaussiana, pero no necesariamente todas las capas de una red lo hacen. Las neuronas en capas más profundas son más selectivas y espe-

Entrada: Elemento x del mini lote: $\mathcal{B} = \{x_{1...m}\}$;
Parámetros para aprender:γ, β
Salida: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// promedio del mini lote}$$

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

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

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// escalamiento y desplazamiento}$$

Algoritmo 1: Proceso de normalización sobre un mini lote [44].

cíficas, lo que puede resultar en activaciones extremadamente grandes y de distribución desconocida. Por esta razón, la normalización por lotes puede ser más apropiado después de la función de activación si se trata de funciones no lineales como la tangente hiperbólica y sigmoidea, pero también puede ser adecuado antes de la función de activación cuando las activaciones de las unidades neuronales no siguen una distribuciones gaussiana como la función de activación lineal rectificada (RELU) [44].

1.4.6.5. Capa de Dropout

Dropout es un método de regularización que se aproxima al entrenamiento de una gran cantidad de redes neuronales con diferentes arquitecturas en paralelo. Durante el entrenamiento, algunos resultados de la capa se ignoran o se ”eliminan” aleatoriamente. Esto tiene el efecto de hacer que la capa se vea y se trate como una capa con un número diferente de nodos y conectividad con la capa anterior. En efecto, cada actualización de una capa durante el entrenamiento se realiza con una ”vista” diferente de la capa configurada [39].

Se puede utilizar con la mayoría de los tipos de capas, como capas densas totalmente conectadas, capas convolucionales y capas recurrentes, como la capa de red de memoria a corto plazo. El dropout se puede implementar en cualquiera o en todas las capas ocultas de la red, así como en la capa visible o de entrada, pero no se implementa en la capa de salida. Con la capa de dropout se introduce un nuevo hiperparámetro que especifica la probabilidad de que se eliminen las salidas de una capa [39].

Al utilizar dropout como regularización, es posible utilizar redes más grandes con menos riesgo de sobreajuste. De hecho, es posible que se requiera una red grande (más nodos por capa) ya que el dropout reducirá la capacidad de la red [39].

1.4.6.6. Capas densamente conectadas

Esta capa funciona exactamente de la misma manera que una red de alimentación directa tradicional. En la mayoría de los casos, se puede usar más de una capa completamente conectada para aumentar la potencia de los cálculos hacia el final. Las conexiones entre estas capas están estructuradas exactamente como una red de alimentación directa tradicional. Dado que las capas completamente conectadas están densamente conectadas, la gran mayoría de los parámetros se encuentran en las capas completamente conectadas [39].
Aunque las capas convolucionales tienen un mayor número de activaciones (y una mayor demanda de recursos de memoria), las capas completamente conectadas a menudo tienen un mayor número de conexiones. La razón por la que las activaciones contribuyen a la demanda de recursos de memoria de manera más significativa es que el número de activaciones se multiplica por el tamaño del mini-lote mientras se rastrean las variables en las pasadas hacia adelante y hacia atrás de la propagación hacia atrás. Es útil tener en cuenta estas compensaciones al elegir el diseño de red neuronal en función de tipos específicos de restricciones de recursos (por ejemplo, disponibilidad de datos versus disponibilidad de memoria). Además es importante destacar que la naturaleza de la capa completamente conectada puede ser sensible a la aplicación en cuestión. Por ejemplo, la naturaleza de la capa completamente conectada para una aplicación de clasificación sería algo diferente del caso de una aplicación de segmentación [39].

1.4.7. BAG OF VISUAL WORDS (BoVW)

Bag of Visual Word (Bolsa de palabras visuales) es una técnica que permite representar imágenes como una colección sin orden de características locales. El nombre se deriva de la analogía con la técnica Bag of Words (Bolsa de palabras) utilizada en la recuperación de información textual, el cual consiste en representar un documento como un histograma normalizado de recuentos de las palabras que contiene. Es decir, se construye un diccionario con las palabras que aparecen en el documento, excluyendo palabras no informativas como artículos (como ”el”) y considerando un solo término para representar un conjunto de sinónimos. Posteriormente se genera un vector disperso de términos donde cada elemento es una palabra del diccionario y el valor de ese elemento es el número de veces que el término aparece en el documento dividido por el número total de palabras que contiene el diccionario. De esta manera el vector es la representación del documento en una bolsa de palabras, denominada ”bolsa” porque se ha perdido todo el orden de las palabras en el documento [46].

El procedimiento para generar una representación de imagen con BoVW se muestra en la Figura 1.34 y se resume en tres etapas: (i) extracción de características, (ii) construcción del vocabulario visual y (iii) generación de un vector global de características.

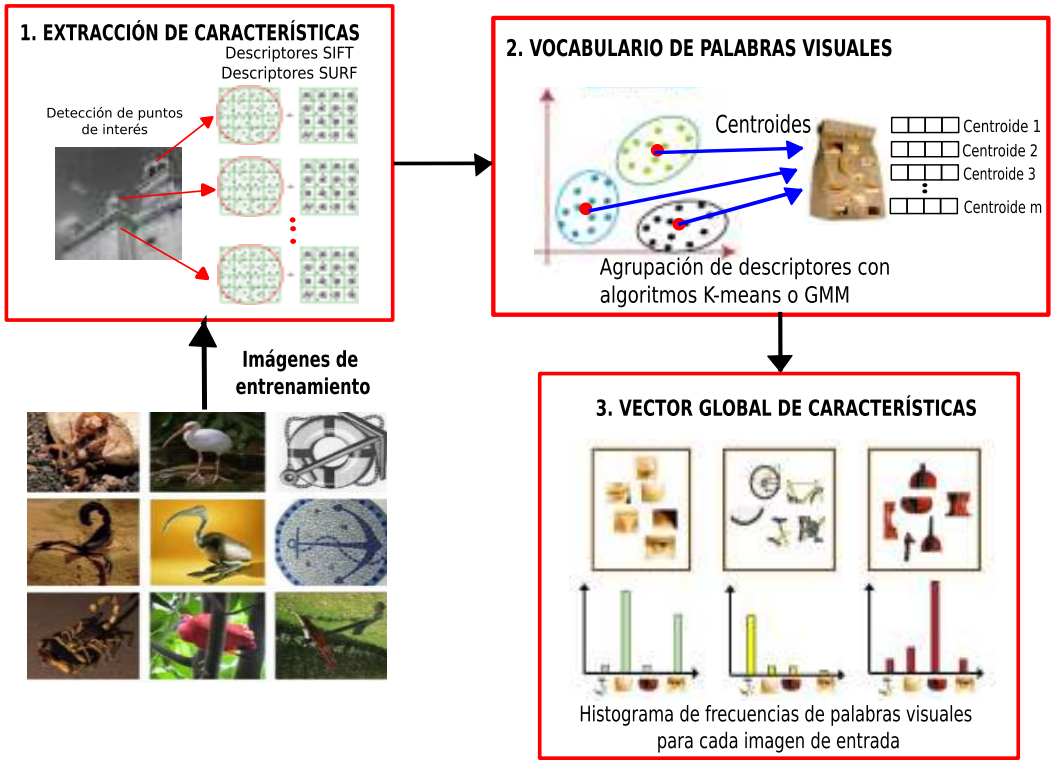


Figura 1.34: Técnica de Bag of Visual Words [46]

1.4.7.1. Extracción de características

Bag of Visual Word construye un vocabulario visual agrupando (clustering) características extraídas de un conjunto de imágenes de entrenamiento. La extracción de características de la imagen representan áreas locales de la imagen, al igual que en Bag of Words donde las palabras son características locales de un documento [46]. La extracción de características locales en una imagen generalmente se efectúa con dos enfoques distintos:

- **Detección de una región local (detector):** La detección de características es el proceso de decidir dónde y a qué escala muestrear una imagen. El resultado de la detección de características es un conjunto de puntos claves que especifican ubicaciones en la imagen con las escalas y orientaciones correspondientes [46]. Hay muchos enfoques posibles para muestrear características de imagen en los que se incluyen las técnicas de operadores de puntos de interés, prominencia visual y muestreo de cuadrícula aleatorio o determinista.

- **Descripción de la región detectada (descriptor):** Los descriptores de características codifican información de los píxeles en la vecindad de una region localizadas denominados puntos claves [46]. Entre los descriptores de características más populares se incluyen a Scale Invariant Feature Transform (SIFT) y Speeded Up Robust Features (SURF).

1.4.7.2. Construcción de vocabulario visual

La agrupación (clustering) de características es necesaria para que se pueda generar un vocabulario discreto a partir de millones (o miles de millones) de características locales extraídas de los datos de entrenamiento. Cada grupo de características es una palabra visual. [46].

1.4.7.3. Generación de un vector global de características

Dada una imagen nueva distinta a las usadas en el proceso de agrupación, las características se detectan y se asignan a sus términos coincidentes más cercanos (centros de grupos o clusters) del vocabulario visual. Así, el vector de características es entonces simplemente el histograma normalizado de las características cuantificadas detectadas en la imagen. [46].

1.4.8. HERRAMIENTAS DE APRENDIZAJE AUTOMÁTICO CON PYTHON

Tensorflow TensorFlow es una plataforma de código abierto de extremo a extremo desarrollado por Google para el Aprendizaje Automático. TensorFlow usa grafos de control de flujo unificado para representar tanto el cálculo en un algoritmo como el estado en el que opera el algoritmo. Asigna los nodos de un grafo de contol de flujo en muchas máquinas de un clúster y dentro de una máquina en varios dispositivos informáticos, incluidas CPU multinúcleo, GPU de propósito general y ASIC de diseño personalizado conocidos como Unidades de procesamiento de tensores (TPU) [47].

TensorFlow admite una variedad de aplicaciones, con un enfoque en el entrenamiento y la inferencia en redes neuronales profundas. La biblioteca central de TensorFlow está implementada en C ++ para la portabilidad y el rendimiento, de esta manera se ejecuta en varios sistemas operativos, incluidos Linux, Mac OS X, Windows, Android e iOS; además en varias arquitecturas de CPU basadas en x86 y ARM; y las microarquitecturas GPU Kepler, Maxwell y Pascal de NVIDIA [47].

Keras Keras es una librería de Aprendizaje Profundo para Python, que se ejecuta sobre la plataforma de aprendizaje automático TensorFlow y presenta las siguientes ventajas:

- Permite que un mismo código se ejecute sin problemas en CPU o GPU.
- Tiene una API(Application Programming Interface) fácil de usar que permite la creación rápida de prototipos de modelos de aprendizaje profundo.
- Tiene soporte integrado para redes neuronales convolucionales, redes neuronales recurrentes y cualquier combinación de ambas.
- Admite arquitecturas de redes neuronales arbitrarias: modelos de múltiples entradas o múltiples salidas, uso compartido de capas, uso compartido de modelos, etc. Esto significa que Keras es apropiado para construir esencialmente cualquier modelo de aprendizaje profundo, desde una red generativa de adversarios hasta una máquina neuronal de Turing [48].

Keras no maneja operaciones de bajo nivel como la manipulación y diferenciación de tensores, es decir, se basa en una biblioteca tensorial especializada y bien optimizada que sirve como motor de backend de Keras. En lugar de elegir una biblioteca de un solo tensor y vincular la implementación de Keras a esa biblioteca, Keras maneja el problema de manera modular (Ver Figura 1.35); por lo tanto, varios motores de backend diferentes se pueden conectar sin problemas a Keras. Actualmente, las tres implementaciones de backend existentes son el backend de TensorFlow, el backend de Theano y el backend de Microsoft CognitiveToolkit (CNTK). En el futuro, es probable que Keras se extienda para trabajar con aún más motores de ejecución de aprendizaje profundo. A través de TensorFlow (o Theano, o CNTK), Keras puede ejecutarse sin problemas tanto en CPU como en GPU. Cuando se ejecuta en la CPU, TensorFlow en sí mismo está envolviendo una biblioteca de bajo nivel para operaciones de tensor llamada Eigen. En GPU, TensorFlow envuelve una biblioteca de operaciones de aprendizaje profundo bien optimizadas llamada biblioteca de red neuronal profunda NVIDIACUDA (cuDNN) [48].

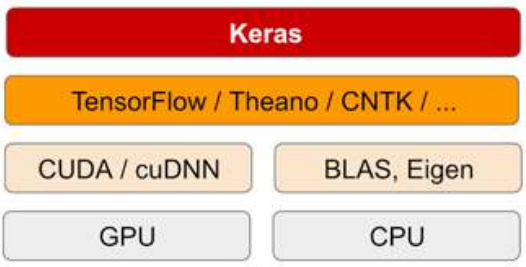


Figura 1.35: La pila de software y hardware de Aprendizaje Profundo de Keras [48]

Numpy NumPy es la principal biblioteca de programación de matrices para el lenguaje Python. Desempeña un papel esencial en las líneas de análisis de investigación en campos tan diversos como la física, la química, la astronomía, la geociencia, la biología, la psicología, la ciencia de los materiales, la ingeniería, las finanzas y la economía [49].
Una matriz en NumPy es una estructura de datos que almacena y accede de manera eficiente a matrices multidimensionales, también conocidas como tensores, y permite una amplia variedad de cálculos científicos. Consiste en un puntero a la memoria, junto con los metadatos utilizados para interpretar los datos almacenados allí, en particular el tipo de datos y la forma. El tipo de datos describe la naturaleza de los elementos almacenados en una matriz y la forma de una matriz determina el número de elementos a lo largo de cada eje, y la cantidad de ejes es la dimensionalidad de la matriz. [49].
NumPy se está convirtiendo en un mecanismo de coordinación central que especifica una API de programación de matrices bien definida y la distribuye, según corresponda, a implementaciones de matrices especializadas [49].

Pandas Pandas es una biblioteca de Python de estructuras y herramientas de datos enriquecidos para trabajar con conjuntos de datos estructurados comunes a las estadísticas, las finanzas, las ciencias sociales y muchos otros campos. La biblioteca proporciona rutinas integradas e intuitivas para realizar manipulaciones y análisis de datos comunes en dichos conjuntos de datos. Su objetivo es ser la capa fundamental para el futuro de la computación estadística en Python. Sirve como un fuerte complemento para la pila científica existente de Python mientras implementa y mejora los tipos de herramientas de manipulación de datos que se encuentran en otros lenguajes de programación estadística como R [50].

OpenCV OpenCV es una biblioteca de código abierto enfocado al campo de Visión Artificial. La biblioteca está escrita en C y C ++ y se ejecuta en Linux, Windows y Mac OS X. Existe un desarrollo activo en interfaces para Python, Ruby, Matlab y otros lenguajes [51].
La biblioteca OpenCV contiene más de 500 funciones que abarcan muchas áreas de la Visión Artificial, incluida la inspección de productos de fábrica, imágenes médicas, seguridad, interfaz de usuario, calibración de cámara, visión estéreo y robótica. OpenCV también contiene una biblioteca de Aprendizaje Automático completa y de uso general, la misma que se centra en el reconocimiento de patrones estadísticos y la agrupación (clustering) [51].

Scipy SciPy es una biblioteca de rutinas numéricas para el lenguaje de programación Python que proporciona bloques de construcción fundamentales para modelar y resolver problemas científicos. La biblioteca SciPy está organizada como una colección de paquetes.

Estos paquetes incluyen bloques de construcción matemáticos (por ejemplo, álgebra lineal, transformadas de Fourier, funciones especiales), estructuras de datos (por ejemplo, matrices dispersas, árboles kD), algoritmos (por ejemplo, optimización e integración numérica, agrupamiento, interpolación, algoritmos de gráficos, geometría computacional) y funcionalidad de análisis de datos de alto nivel (por ejemplo, procesamiento de señales e imágenes, métodos estadísticos) [52].

Scikit-learn Scikit-learn es un módulo de Python que integra una amplia gama de algoritmos de Aprendizaje Automático de última generación para problemas supervisados y no supervisados de mediana escala. Este paquete se centra en llevar el Aprendizaje Automático a los no especialistas que utilizan un lenguaje de alto nivel de uso general. Se hace hincapié en la facilidad de uso, el rendimiento, la documentación y la coherencia de la API. Tiene dependencias mínimas y se distribuye bajo la licencia BSD (Berkeley Software Distribution) simplificada, lo que fomenta su uso tanto en entornos académicos como comerciales. Concebido como una extensión de la biblioteca SciPy, scikit-learn se basa en las populares bibliotecas Python NumPy y matplotlib [53].

Matplotlib Matplotlib es una herramienta de visualización de datos multiplataforma construida sobre Numpy y Scipy y permite el trazado de gráficas 2D y 3D con calidad de producción. Admite el trazado interactivo y no interactivo, y puede guardar imágenes en varios formatos de salida (PNG, PS y otros). Puede utilizar varios conjuntos de herramientas de ventana (GTK +, wxWidgets, Qt, etc.) y proporciona una amplia variedad de tipos de gráficos (líneas, barras, gráficos circulares, histogramas y muchos más). Además de esto, es altamente personalizable, flexible y fácil de usar [54].
La naturaleza dual de Matplotlib permite su uso en scripts interactivos y no interactivos. Se puede utilizar en scripts sin una pantalla gráfica, incrustado en aplicaciones gráficas o en páginas web. También se puede utilizar de forma interactiva con el intérprete de Python o IPython [54].

Mahotas Mahotas es una biblioteca de Visión Computacional para Python que se publica bajo una licencia liberal de código abierto (Licencia MIT). Contiene funcionalidades de procesamiento de imágenes tradicionales, como operaciones de filtrado y morfológicas, así como funciones de extracción de características, incluida la detección de puntos de interés y descriptores locales. La interfaz de esta biblioteca se desarrolla en Python, un lenguaje de programación dinámico, que es muy apropiado para un desarrollo rápido, pero los algoritmos se implementan en C ++ [55].

Contiene más de 100 funciones que van desde el filtrado de imágenes tradicional y operaciones morfológicas hasta descomposiciones de Wavelet y cálculos de características locales. Además, al integrarse en el ecosistema numérico de Python, los usuarios pueden usar otros paquetes sin problemas como Scikits-learn [55].
En particular, a diferencia de las bibliotecas escritas en lenguaje C o en Java, Mahotas no necesita definir una nueva estructura de datos de imagen, sino que utiliza la estructura de matriz de la librería Numpy [55].

Google Colab Google ha creado Google Colaboratory (Colab), un servicio en la nube para difundir la educación y la investigación del Aprendizaje Automático. El tiempo de ejecución proporcionado por este servicio en la nube está completamente configurado con las principales bibliotecas de Inteligencia Artificial (AI) y también ofrece una GPU robusta. Este servicio de Google está vinculado a una cuenta de Google Drive y es gratuito. Google Colaboratory se basa en Júpiter Notebook que es una herramienta de código abierto y basada en navegador que integra lenguajes interpretados, bibliotecas y herramientas de visualización [56]. Colaboratory proporciona máquina virtual con Python 2 y 3 preconfigurados con las bibliotecas esenciales de Aprendizaje Automático e Inteligencia Artificial, como TensorFlow, Matplotlib y Keras. La máquina virtual se desactiva después de un período de tiempo, y se pierden todos los datos y configuraciones del usuario. Sin embargo, el portátil se conserva y también es posible transferir archivos desde el disco duro de la VM a la cuenta de Google Drive del usuario. Finalmente, este servicio de Google proporciona un tiempo de ejecución acelerado por GPU. La infraestructura de Google Colaboratory está alojada en la plataforma Google Cloud [56].

2. METODOLOGÍA

En este proyecto de titulación se plantea la implementación de un sistema de clasificación de regiones de interés previamente segmentados en artículos científicos digitales a través de algoritmos de Aprendizaje Automático Supervisado. El sistema permite clasificar regiones de texto, figuras y tablas que se encuentran presentes en cada página de un artículo científico con la ayuda de una Red Neuronal Convolucional. Además el sistema contiene una etapa basada en la técnica Bag of Visual Words para identificar líneas de ecuaciones en regiones de texto. En el diagrama de bloques de la Figura 2.1 se muestra de forma didáctica y resumida las etapas del sistema de clasificación automática.

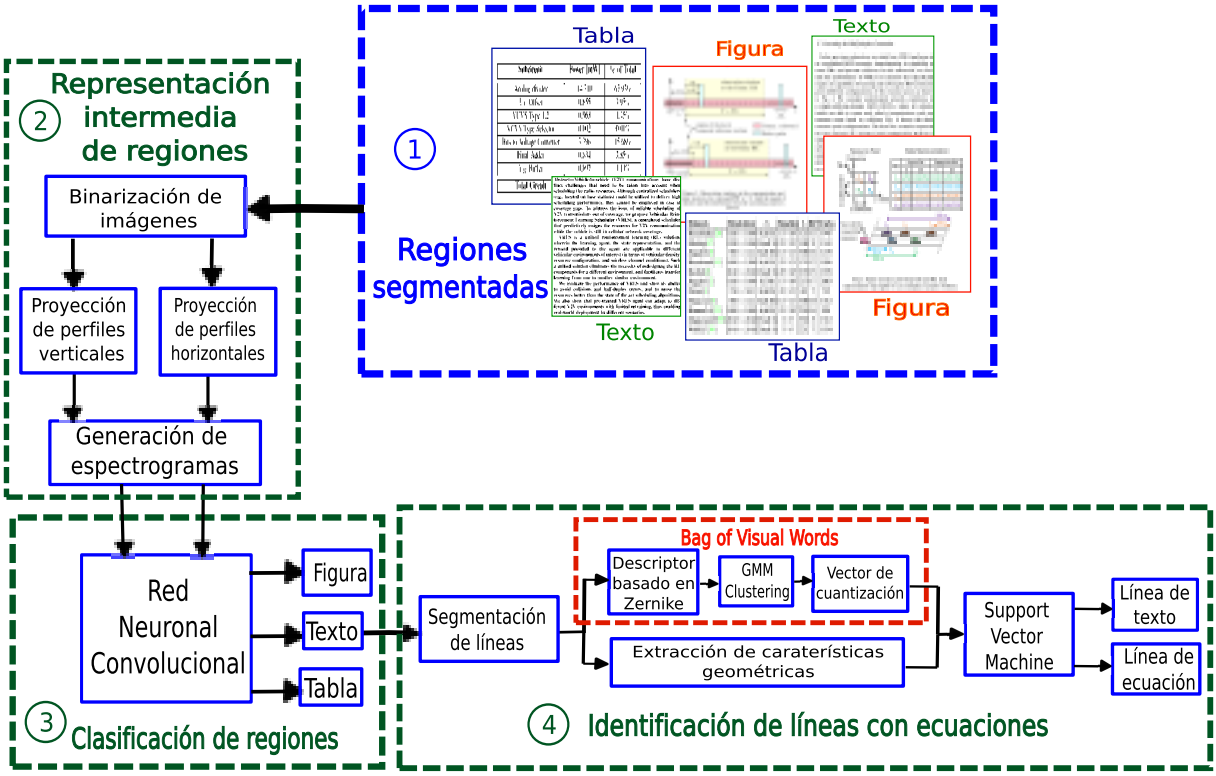


Figura 2.1: Etapas del algoritmo de clasificación automática de regiones

Inicialmente, en la Sección 2.1, se describe la estructura de una base de datos previamente construida que contiene un conjunto extenso de páginas de artículos científicos segmentadas en regiones de texto, figuras, tablas y ecuaciones. Debido a que se utiliza un enfoque de Aprendizaje Automático Supervisado es necesario un conjunto de datos etiquetados que permita efectuar una fase de entrenamiento, validación y evaluación del rendimiento de cada uno de los algoritmos implementados. En la Sección 2.2 se detalla la etapa de representación intermedia que efectúa un pre-procesamiento sobre las regiones de interés de la base

datos, con el objetivo de extraer características únicas de cada región que serán útiles para la Red Neuronal Convolucional en el proceso de entrenamiento y validación. En la Sección 2.3 se explica el diseño de cada una de las capas y de los valores de hiperparámetros empleadas en la Red Neuronal Convolucional para la clasificación de regiones de texto, tablas y figuras. Además en esta sección se especifica las técnicas empleadas para el entrenamiento y validación de la Red Neuronal Convolucional diseñada. Posteriormente, la Sección 2.4 describe en detalle la técnica de Bag of Visual Words para la identificación de línea de ecuaciones en regiones de texto. Finalmente, en la Sección 3 se cuantificará el rendimiento de cada uno de los algoritmos empleados en el sistema de clasificación utilizando un conjunto de datos diferente empleado en la fase de entrenamiento y métricas de evaluación de clasificación.

2.1. BASE DE DATOS

Para la fase de entrenamiento, validación y evaluación de cada uno de los algoritmos se considera una base de datos de páginas segmentadas con cuatro tipos de regiones: texto, figura, tabla y ecuación. Esta base de datos consta de un total de $11\,007$ archivos en formato PDF, generando un total de $121\,703$ páginas segmentadas en formato PNG con una resolución de 500 dpi y tamaño de imagen en promedio de 4250×5500 píxeles.

La base de datos almacena también información sobre las coordenadas de los cuadros delimitadores de cada una de las regiones presentes en una página con sus respectivas etiquetas. Para los cuadros delimitadores se toma en cuenta un sistema cartesiano ubicado en la parte superior izquierda de cada una de las páginas en donde cada píxel corresponde a un punto de coordenadas. La Figura 2.2 ilustra una página de la base de datos que contiene las 4 regiones de interés con sus respectivas etiquetas.

2.2. REPRESENTACIÓN INTERMEDIA

Al trabajar con una base de datos de imágenes de tamaño variable, es necesario implementar una fase de representación intermedia que permita obtener imágenes de tamaño constante de tal manera que contengan información de las características de cada una de las imágenes originales.

En esta fase se plantea extraer los perfiles horizontal y vertical de las imágenes previamente binarizadas con un umbral adaptativo. Posteriormente se extraerá los espectrogramas de los

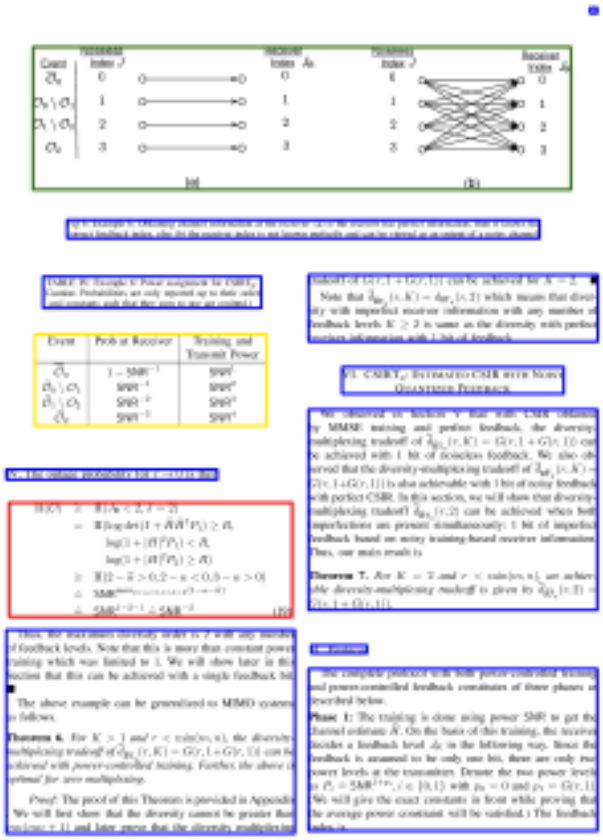


Figura 2.2: Página segmentada con las cuatro regiones: texto (etiqueta azul), figura (etiqueta verde), tabla (etiqueta amarilla) y ecuación (etiqueta roja).

perfiles obtenidos, los mismos que serán imágenes de tamaño constante para que puedan ser aprovechados por la Red Neuronal Convolucional para el proceso de clasificación.

2.2.1. BINARIZACIÓN DE IMÁGENES

La primera transformación que se realiza sobre cada una de las imágenes, es la binarización de la imagen original en escala de grises, utilizando la técnica de Umbral Adaptativo Gaussiano, es decir, que el valor de umbral será variable para distintas regiones de la imagen y serán calculados como la suma ponderada de los valores de los píxeles vecinos y cuyos pesos son determinados a partir de una ventana gaussiana.

La binarización de cada una de las regiones, se realizó con la librería **OpenCV** con el lenguaje de programación Python. Primero con la función **cv2.cvtColor()** se realiza la transformación de la imagen desde el espacio RGB a escala de grises. Esta función recibe los siguientes parámetros:

- **src:** Recibe la imagen a transformar en formato de array de Numpy.

- **code:** Especifica el código de la conversión de espacio de color a efectuar sobre la imagen. Para la conversión del espacion de color RGB a la escala de grises se utiliza el código cv2.COLOR_RGB2GRAY.

Posteriormente, con la función **cv2.adaptiveThrehold()** se obtiene la binarización con el Umbral Adaptativo Gaussiano. Los parámetros que recibe esta función son:

- **src:** Recibe la imagen en escala de grises en formato array Numpy de un solo canal.

- **maxValue:** Valor distinto de cero asignado a los píxeles para los que se cumple la condición de acuerdo a la comparación con valor de umbral para la binarización.

- **adaptiveMethod:** Código del algoritmo de umbral adaptativo a emplear. OpenCV incluye el código cv2.ADAPTIVE_THRESH_MEAN_C para el algoritmo umbral adaptativo promedio y el código cv2.ADAPTIVE_THRESH_GAUSSIAN_C para el algoritmo umbral adaptativo gaussiano.

- **thresholdType:** Código para el modo de aplicación del umbral. OpenCV incluye el código cv2.THRESH_BINARY para cambiar los valores de píxeles de imagen en escala de grises que sean mayores al umbral al valor especificado en el argumento **maxValue** y caso contrario se cambia por un valor de cero. El código cv2.THRESH_BINARY_INV cambia los valores de píxeles de imagen en escala de grises que sean mayores al umbral a un valor cero y caso contrario se cambia al valor especificado en el argumento **maxValue**.

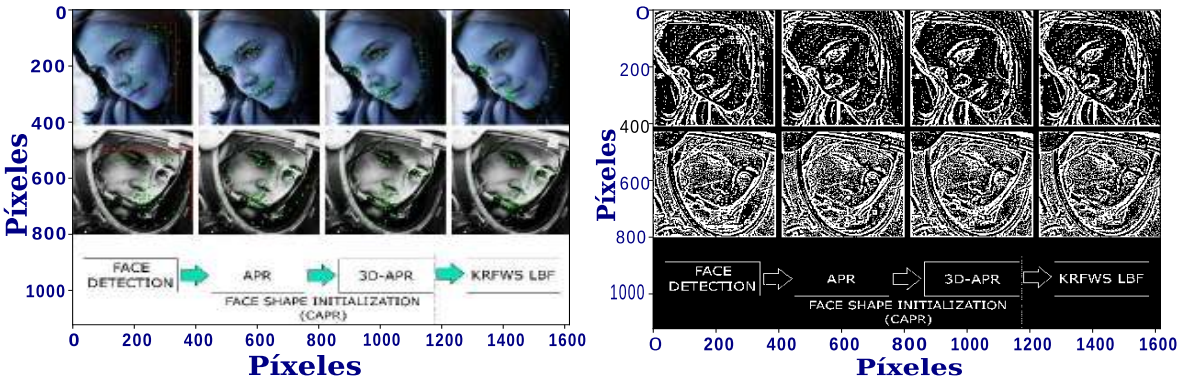
- **blockSize:** Tamaño de la vecindad de píxeles que se consideran para calcular un valor de umbral. Típicamente se escoge valores de 3, 5 y 7.

- **C:** es una constante que se resta de la media o suma ponderada de los píxeles vecinos.

En el Código 1 se presenta un ejemplo de binarización con el método de Umbral Adaptativo Gaussiano con la librería OpenCV en Python.
En las figuras 2.3, 2.4 y 2.5 se muestran los resultados del proceso de binarización usando el Código 1 para tres ejemplos de regiones. Estas regiones son extraídas de las páginas de los artículos científicos en formato de imagen PNG provenientes de la base de datos proporcionado para el desarrollo de este proyecto.

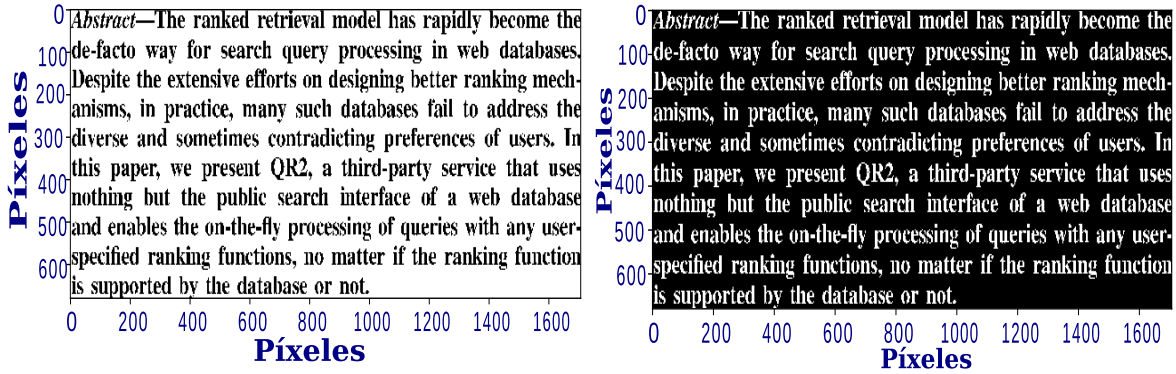
```python
"""Transformación de RGB a escala de grises"""
gray_scale = cv2.cvtColor(src = ImageOriginal,
                          code = cv2.COLOR_BGR2GRAY)

"""Binarización de la imagen en escala de grises"""
binary_region = cv2.adaptiveThreshold(src = gray_scale,
                                      maxValue = 255,
                                      adaptiveMethod = cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      thresholdType = cv2.THRESH_BINARY_INV,
                                      blockSize = 11,
                                      C = 2)
```

Código 1: Binarización de imágenes con librería OpenCV con Python



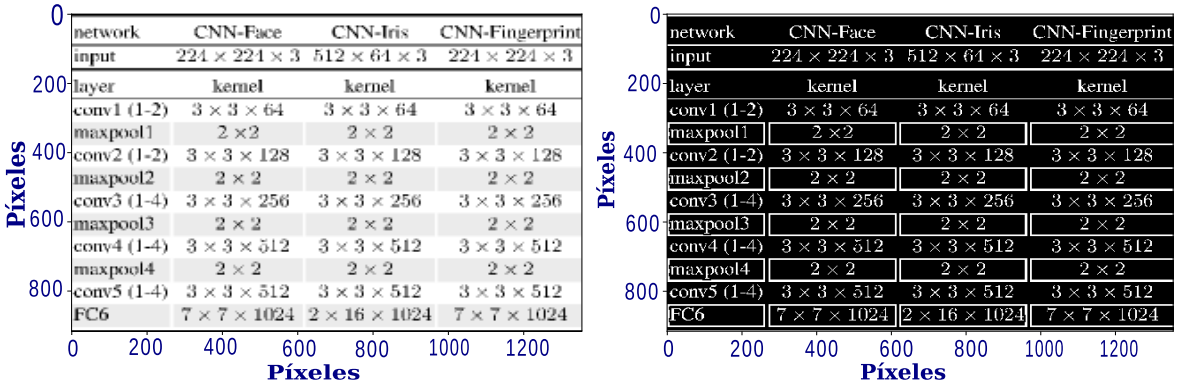
(a) Imagen formato color RGB **(b)** Imagen Binarizada

Figura 2.3: Binarización de una imagen de 1616×1123 píxeles de la base de datos etiquetada como región de "Figura"



(a) Imagen formato color RGB **(b)** Imagen Binarizada

Figura 2.4: Binarización de una imagen de 1707×680 píxeles de la base de datos etiquetada como región de "Texto"



(a) Imagen formato color RGB **(b)** Imagen Binarizada

Figura 2.5: Binarización de una imagen de 1355×916 píxeles de la base de datos etiquetada como región de ”Tabla”

2.2.2. EXTRACCIÓN DE PERFILES HORIZONTAL Y VERTICAL

Una vez efectuado la binarización de las regiones de interés, se procede a la extracción de las proyecciones de perfiles. La proyección de perfil de una imagen binarizada se define como el histograma del número de valores de píxeles acumulados a lo largo de líneas paralelas tomadas a través de la imagen, es decir se calcula sumando los valores de los píxeles de la imagen a lo largo de una dimensión específica. Para la imagen binarizada $f(x, y)$ de tamaño $m \times n$ de la Figura 2.6, la representación matemática de la proyección del perfil vertical (PPV) y la proyección del perfil horizontal (PPH) se define en la ecuación 2.1.

$$PPV(y) = \sum_{x=1}^{m} f(x, y) \quad \text{y} \quad PPH(x) = \sum_{y=1}^{n} f(x, y) \tag{2.1}$$

Por cada imagen binarizada se obtuvieron las proyecciones de perfil tanto en la dimensión vertical y como en la horizontal con el objetivo principal de encontrar patrones distintivos para cada tipo de región que se desea clasificar. De esta forma en la Figura 2.7 se observa los perfiles para una región de ”texto”. Para las regiones de texto, se identifica que la proyección vertical presenta visualmente una señal de forma cuadrada, mientras que la proyección horizontal aproximadamente es una señal constante. De manera similar, las regiones de ”tabla” también presentan visualmente una señal cuadrada en sus proyecciones verticales y horizontales, así como también pequeños pulsos . En la Figura 2.8 se muestra un ejemplo de los perfiles para una región de tabla seleccionada de la base de datos. Mientras que para las regiones de ”figuras”, las proyecciones horizontal y vertical, no presentan ningún patrón característico. En la Figura 2.9 se ilustra los perfiles horizontal y vertical de una región de figura.

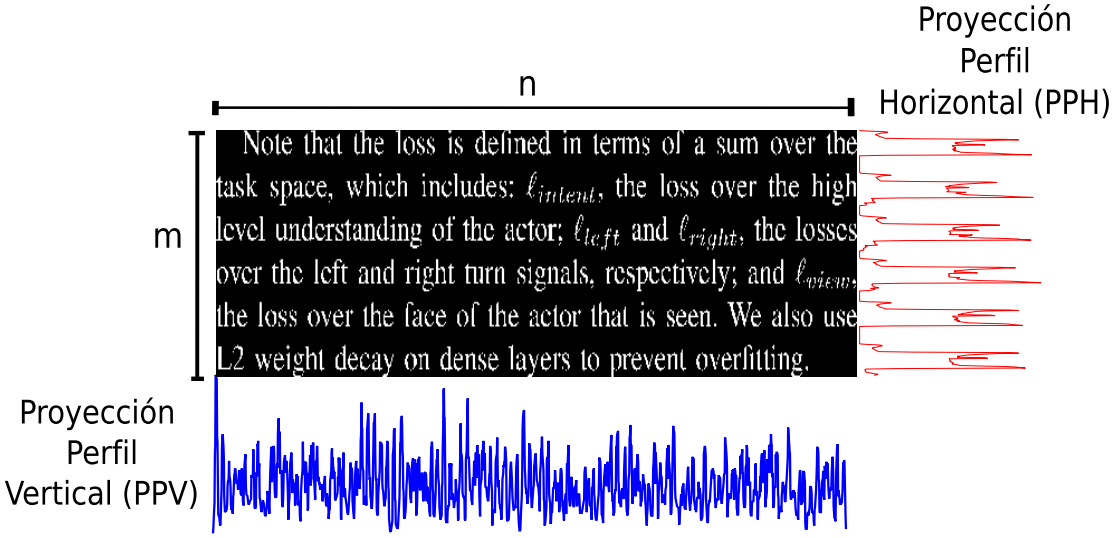
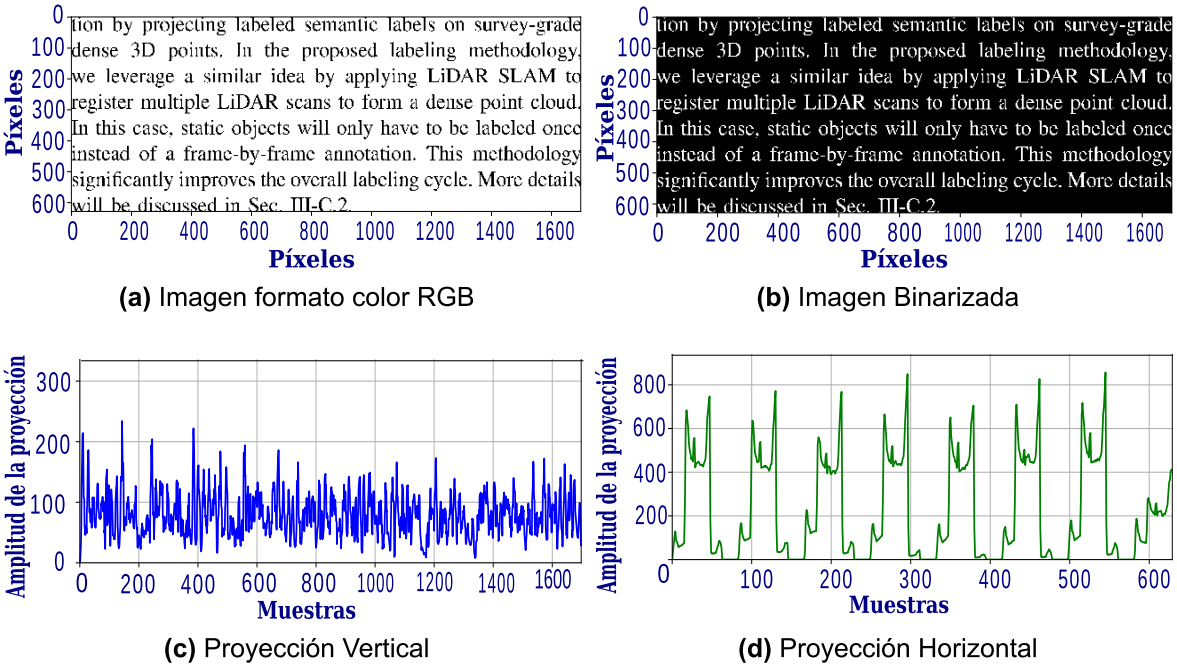


Figura 2.6: Proyecciones de los perfiles vertical y horizontal de una imagen binarizada de tamaño $m \times n$



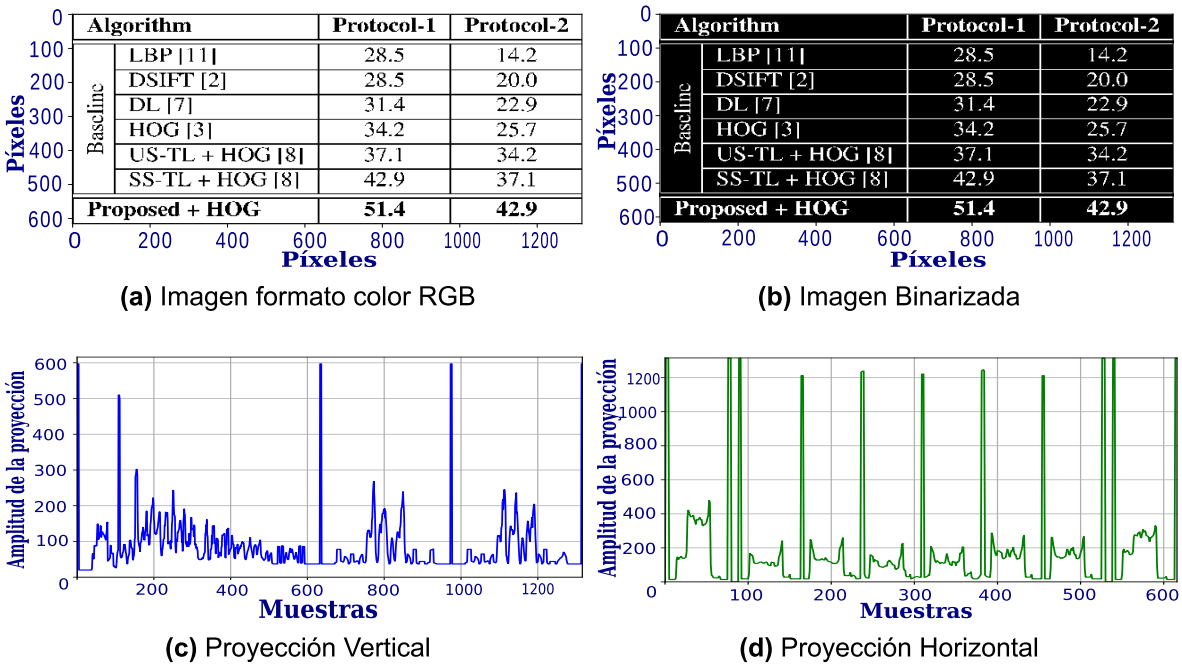
(a) Imagen formato color RGB

(b) Imagen Binarizada

(c) Proyección Vertical

(d) Proyección Horizontal

Figura 2.7: Se muestra: a) Una imagen de 1698×629 píxeles correspondiente a una región etiquetada como ”Texto”. b) Binarización de la imagen. c) Proyección vertical de la imagen binarizada. d) Proyección horizontal de la imagen binarizada



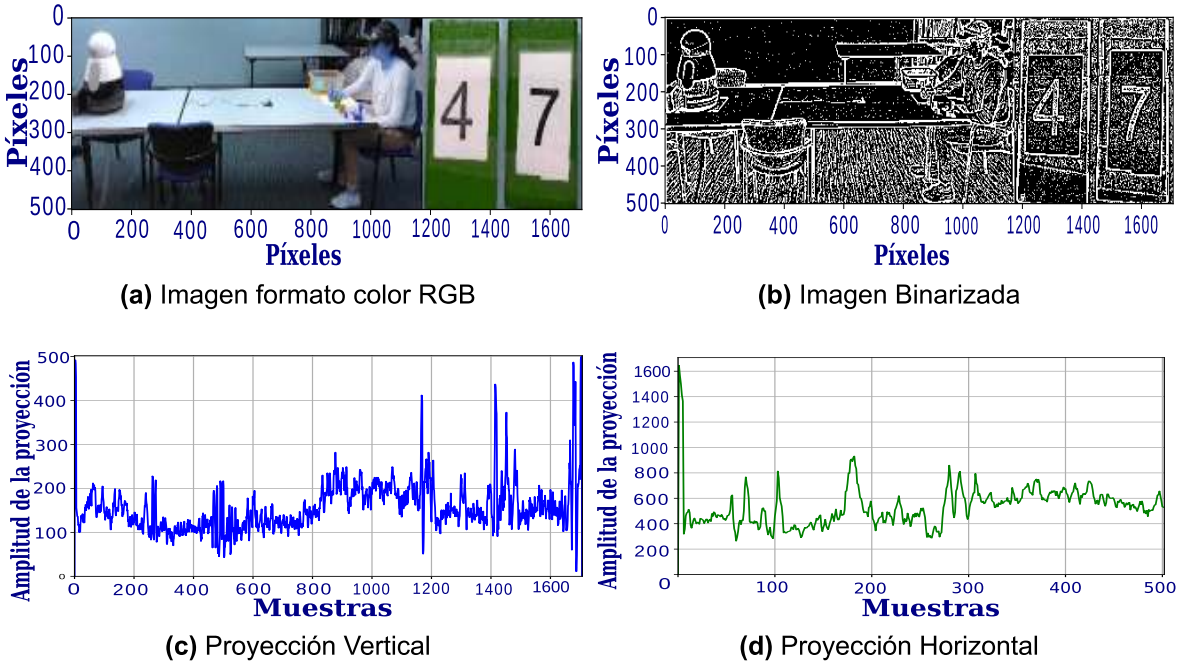
(a) Imagen formato color RGB

(b) Imagen Binarizada

(c) Proyección Vertical

(d) Proyección Horizontal

Figura 2.8: Se muestra: a) Una imagen de 1316×616 píxeles correspondiente a una región etiquetada como ”Tabla”. b) Binarización de la imagen. c) Proyección vertical de la imagen binarizada. d) Proyección horizontal de la imagen binarizada



(a) Imagen formato color RGB

(b) Imagen Binarizada

(c) Proyección Vertical

(d) Proyección Horizontal

Figura 2.9: Se muestra: a) Una imagen de 1707×502 píxeles correspondiente a una región etiquetada como ”Figura”. b) Binarización de la imagen. c) Perfil vertical de la imagen binarizada. d) Perfil horizontal de la imagen binarizada

2.2.3. GENERACIÓN DE ESPECTROGRAMAS

Los perfiles vertical y horizontal obtenidos de las imágenes binarizadas, son señales de longitud variable que presentan formas de ondas particulares dependiendo del tipo de región. Por esta razón se plantea en esta fase, obtener los espectrogramas de cada una de estas señales para extraer información de su composición a través de la representación gráfica de la densidad del espectro potencia de la Transformada de Fourier de Corto Plazo Discreta (DSTFT).

El espectrograma tanto de la proyección vertical como horizontal serán utilizados por la Red Neuronal Convolucional para la fase de clasificación, por lo que estos espectrogramas deben ser generados con un tamaño que sea aceptado por la capa de entrada de la red. De esta manera, la Red Neuronal Convolucional de este proyecto tiene una capa de entrada que recibirá los espectrogramas de las proyecciones en formato de imagen PNG con el mismo ancho y alto en píxeles con escala de color RGB (Red, Green, Blue).

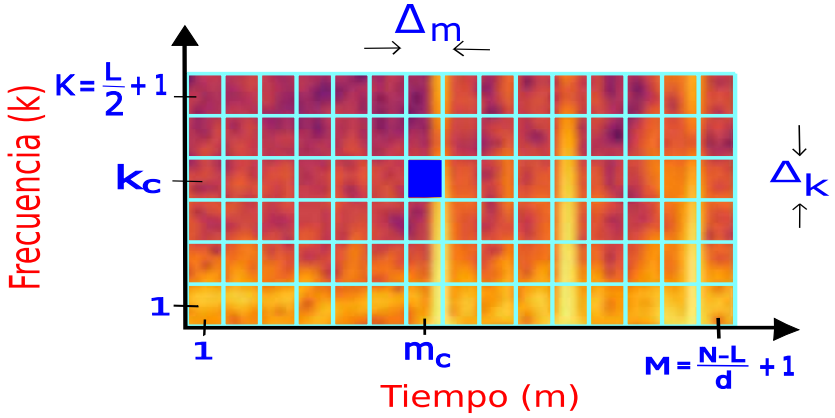


Figura 2.10: Segmentación del plano tiempo - frecuencia.

Al calcular la DSTFT de una señal discreta $x[n]$, con una función de ventana $g[n]$ y desplazamiento de d muestras como se define en la ecuación 2.2, es posible representar la señal $x[n]$ en un plano tiempo-frecuencia discretizado como se observa en la Figura 2.10. De esta manera el plano se divide en pequeñas áreas rectangulares llamadas segmentos. Estos segmentos tienen la misma área y forman una partición de la región. Su área es tal que, para cualquier intervalo de tiempo, el número de segmentos necesarios para cubrir la región correspondiente del plano es igual a la mitad del número de muestras de señal enventanada en el intervalo [57].

$$\mathcal{X}[m,k] := \sum_{n=0}^{N-1} x[n+md]g[n]e^{-2j\pi kn/N} \tag{2.2}$$

El número de segmentos verticales y horizontales en el plano tiempo-frecuencia se describe por la ecuación 2.3.

$$\begin{cases} K = \dfrac{L}{2} + 1 \\ \\ M = \dfrac{N-L}{d} + 1; \qquad d \quad \text{múltiplo de} \quad (N-L) \end{cases} \tag{2.3}$$

Con lo mencionado anteriormente, se pretende obtener un espectrograma de tamaño $K \times M$, para el caso específico cuando $K = M$. De esta manera se seleccionó un tamaño de espectrograma de 64×64 píxeles, para reducir el procesamiento en la Red Neuronal Convolucional en la fase de entrenamiento.

Inicialmente se asigna un valor para la altura del espectrograma, el cual será determinado por el tamaño de la ventana L de acuerdo a la ecuación 2.3. Considerando que se ha seleccionado una altura de 64 para el espectrograma, el tamaño de la ventana deslizante debe tener una longitud de 126 muestras.

$$K = \frac{L}{2} + 1 = 64 \Rightarrow L = 126 \tag{2.4}$$

Posteriormente se necesita determinar el valor de desplazamiento de la ventana d y la longitud N que deben tener las señales obtenidas de los perfiles de las imágenes binarizadas. De acuerdo a la ecuación 2.3, si se asigna el valor 64 para la anchura del espectrograma y una longitud de ventana deslizante de 126, se tiene que los valores para los parámetros N y d son $2\,016$ y 30 respectivamente.

$$M = \frac{N-L}{d} + 1 \Rightarrow 64 = \frac{N-126}{d} + 1 \Rightarrow N = 2016, d = 30 \tag{2.5}$$

Entonces, para que el espectrograma de las señales correspondientes a los perfiles de las imágenes binarizadas posean un tamaño de 64×64. se debe tener en cuenta que estos perfiles deben ser señales de una longitud de $2\,016$ muestras y se debe utilizar una ventana deslizante de 126 con desplazamientos de 30 muestras. Esto implica que cada una de las señales de los perfiles deben ser ajustadas a una longitud de $2\,016$ muestras.

El ajuste de las señales se realiza considerando dos enfoques. El primer caso se tiene cuando la longitud de la señal supera las $2\,016$ muestras, por lo que para la generación del espectrograma solo se considera $2\,016$ muestras centrales de la señal. Mientras que para el segundo caso que corresponde a señales cuyas longitudes son menores a $2\,016$ muestras,

se aplica una interpolación polifase.

Con la interpolación polifase la señal se submuestrea por un factor up, se aplica un filtro FIR de paso bajo de fase cero, y luego se diezma la señal por el factor $down$. La frecuencia de muestreo resultante es la frecuencia de muestreo original multiplicada por el factor $up/down$. La interpolación se lleva a cabo con un filtro FIR pasa bajo de fase cero con la ventana de Kaiser. En el Código 2 se muestra la interpolación polifase con la librería **Scipy** con la función **scipy.signal.resample_poly()** que recibe los siguientes argumentos:

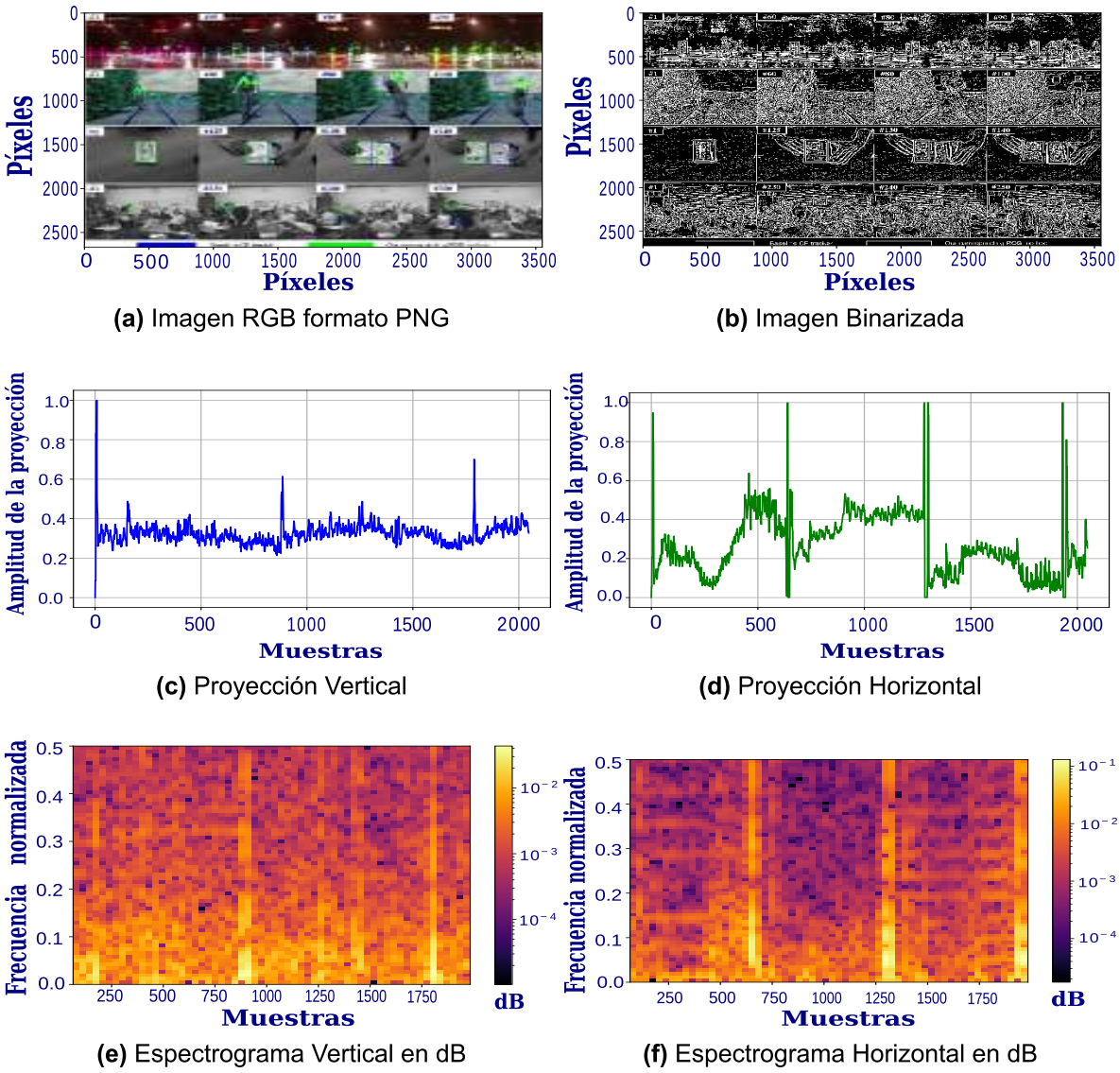
- **x:** El vector en formato de Numpy array sobre el que se efectua la interpolación o decimación.
- **up:** Factor de interpolación.
- **down:** Factor de diezmado.
- **window:** Ventana deseada a utilizar para diseñar el filtro de paso bajo, o los coeficientes de filtro FIR a emplear.

```
projection_sampling = scipy.signal.resample_poly( x = projection_array,
                                                  up = 2016,
                                                  down = len(projection_array),
                                                  window='kaiser', 5.0)
```

Código 2: Interpolación Polifase de una señal con la librería Scipy a una señal con 2016 muestras

Para la generación de los espectrogramas de tamaño 64×64 se consideró una ventana de Blackman-Harrison de 4 términos, de longitud de 126 con la función **scipy.signal.spectrogram()** de la librería **Scipy** como se indica en el Código 3. Esta función retorna la matriz de frecuencias(f), la matriz de segmentos temporales(t) y la matriz de coeficientes de la densidad espectral de potencia de STFT (Sxx)

En las figuras 2.11, 2.12 y 2.13 se muestran los resultados de la generación de espectrogramas de las proyecciones de los perfiles vertical y horizontal para tres ejemplos de regiones, provenientes de las páginas de los artículos científicos en formato de imagen PNG de la base de datos provista para este proyecto.



(a) Imagen RGB formato PNG

(b) Imagen Binarizada

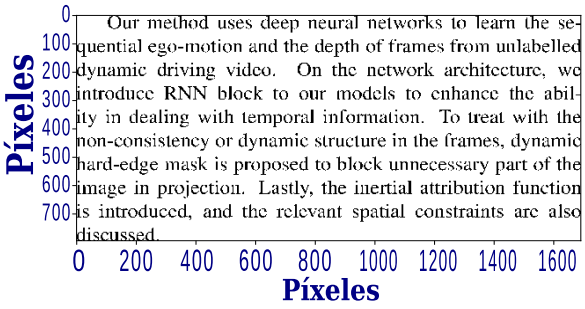
(c) Proyección Vertical

(d) Proyección Horizontal

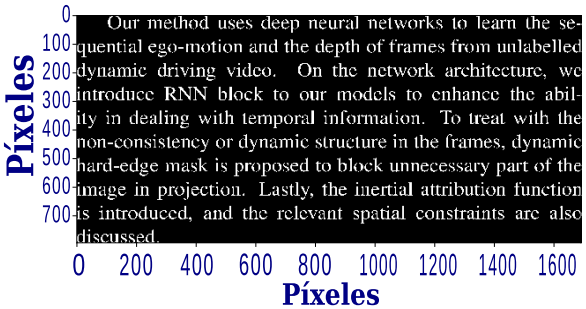
(e) Espectrograma Vertical en dB

(f) Espectrograma Horizontal en dB

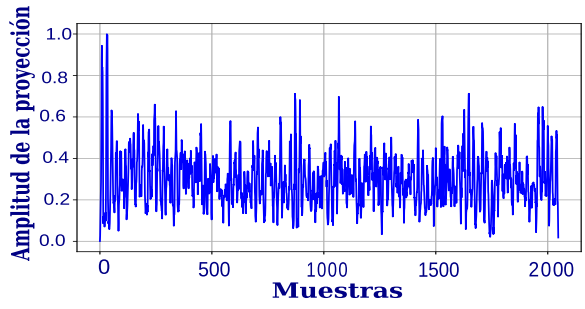
Figura 2.11: Se muestra: a) Una imagen de píxeles correspondiente a una región etiquetada como ”Figura”. b) Binarización de la imagen. c) Proyección vertical de la imagen binarizada. d) Proyección horizontal de la imagen binarizada e) Espectrograma Vertical f) Espectrograma Horizontal



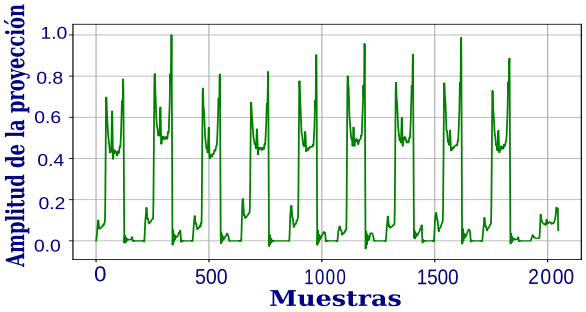
(a) Imagen RGB formato PNG



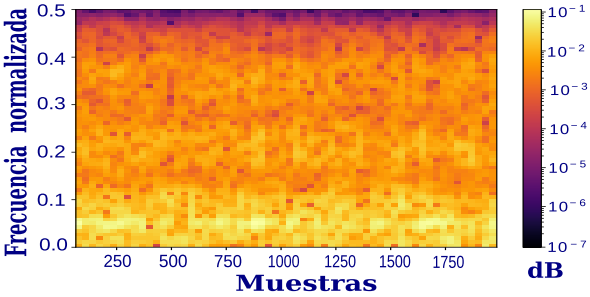
(b) Imagen Binarizada



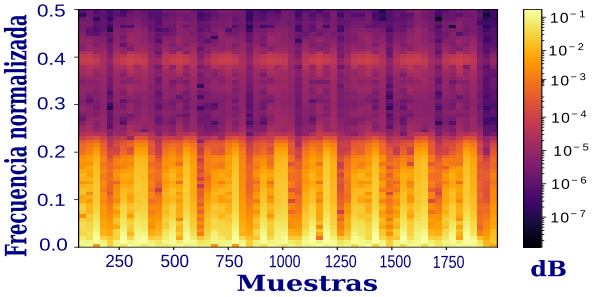
(c) Proyección Vertical



(d) Proyección Horizontal

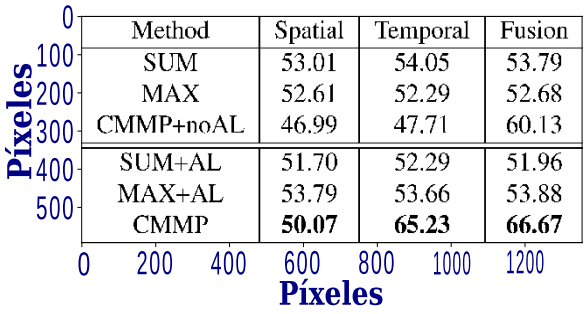


(e) Espectrograma Vertical en dB

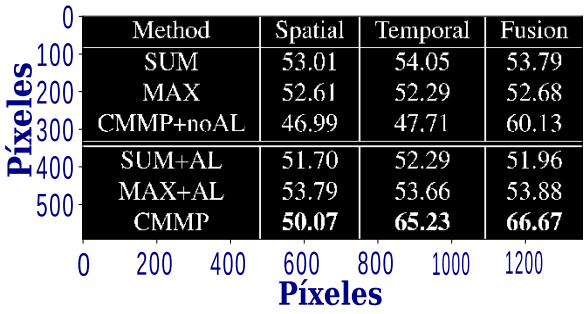


(f) Espectrograma Horizontal en dB

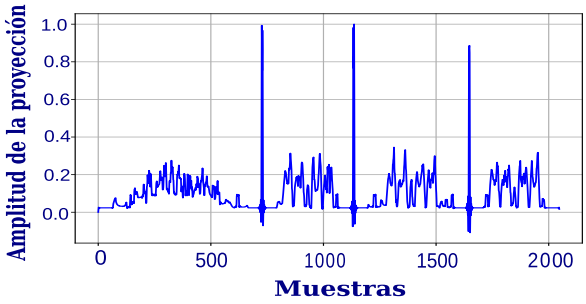
Figura 2.12: Se muestra: a) Una imagen de píxeles correspondiente a una región etiquetada como ”Texto”. b) Binarización de la imagen. c) Proyección vertical de la imagen binarizada. d) Proyección horizontal de la imagen binarizada e) Espectrograma Vertical f) Espectrograma Horizontal



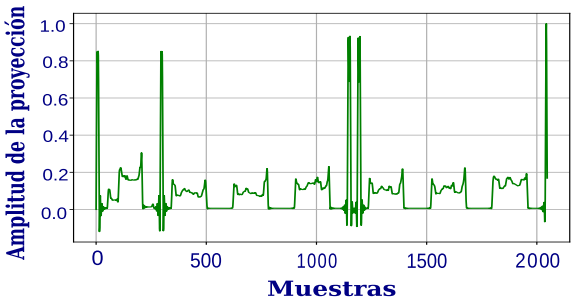
(a) Imagen RGB formato PNG



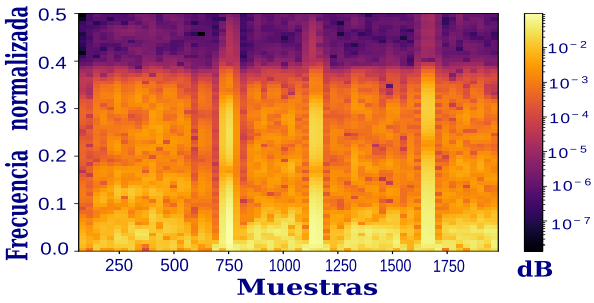
(b) Imagen Binarizada



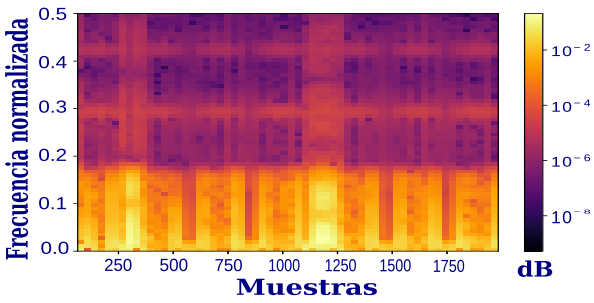
(c) Proyección Vertical



(d) Proyección Horizontal



(e) Espectrograma Vertical en dB



(f) Espectrograma Horizontal en dB

Figura 2.13: Se muestra: a) Una imagen de píxeles correspondiente a una región etiquetada como ”Tabla”. b) Binarización de la imagen. c) Proyección vertical de la imagen binarizada. d) Proyección horizontal de la imagen binarizada e) Espectrograma Vertical f) Espectrograma Horizontal

```
windowBH = scipy.signal.blackmanharris(M=126)
f, t, Sxx = scipy.signal.spectrogram( x = projection_sampling,
                                      fs=1.0,
                                      window = windowBH,
                                      noverlap=96,
                                      scaling = 'spectrum',
                                      mode = 'magnitude')
```

Código 3: Generación de espectrogramas de tamaño 64×64 con enventanado de Blackman-Harrison con la librería Scipy

2.3. DISEÑO DE LA RED NEURONAL CONVOLUCIONAL PARA CLASIFICACIÓN DE REGIONES

Los espectrogramas de las proyecciones vertical y horizontal son imágenes de tamaño 64×64 píxeles, por lo que será necesario diseñar una Red Neuronal Convolucional para el proceso de clasificación, que acepte dos entradas simultáneamente. Esta nueva red neuronal convolucional, al ser una arquitectura específicamente diseñada para la tarea de clasificación de regiones, se la debe someter a una fase de entrenamiento sobre una conjunto de datos etiquetados, para ajustar los valores de los pesos y kernels de cada una de las capas, para posteriormente ejecutar una fase de predicción sobre un conjunto de datos diferente a la etapa de entrenamiento que permitirá determinar el rendimiento de la red neuronal convolucional.

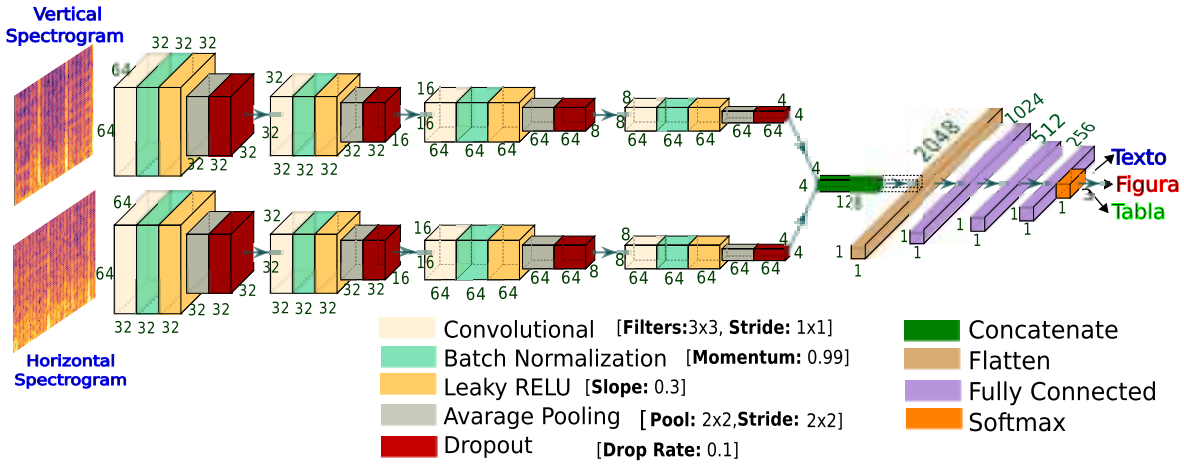


Figura 2.14: Arquitectura de la Red Neuronal Convolucional

La arquitectura de la red neuronal convolucional debe estar estruturada por bloques convolucionales y capas densamente conectadas, las primeras tendrán como función la extracción de mapa de características de los espectrogramas y las segundas permitirán determinar

la probabilidad de pertenencia a cada grupo de clasificación. La red neuronal convolucional aceptará los espectrogramas de las proyecciones vertical y horizontal, por lo que la arquitectura está conformada por dos ramas convolucionales que procesarán los espectrogramas respectivamente. En la Figura 2.14 se detalla la arquitectura de la Red Neuronal Convolucional. Cada ramal de la red se construye con 4 bloques convolucionales y cada uno estos bloques es el resultado de apilar las siguientes capas:

- Capa Convolucional
- Capa de Batch Normalization
- Capa Leaky Rectified Linear Unit (RELU)
- Capa de Avarage Pooling
- Capa de Dropout

Posteriormente se concatenan estos dos ramales para alimentar a las capas densamente conectadas y a un clasificador SoftMax.

2.3.1. ESTRUCTURA DE LOS BLOQUES CONVOLUCIONALES

Cada uno de los cuatro bloques convoluciones se construyen apilando 5 capas cuyas funciones son independientes entre sí y las mismas que permitirán la extracción de características de los espectrogramas horizontal y vertical. El diseño de cada capa se basa en la selección adecuada de sus hiperparámetros y el cálculo del tamaño del mapa de características que se desea obtener a la salida de cada bloque.

2.3.1.1. Capa Convolucional

En la sección 1.4.6.1 se detalló que una capa convolucional queda definida por las siguientes hiperparámetros:

- Número de filtros
- Tamaño de filtros
- Valor de stride
- Tamaño de padding

De esta forma la librería de Keras utiliza estos parámetros para construir una capa convolucional a través de la clase **tf.keras.layers.Conv2D()** como se ilustra en el código 4. Para

cada una de las capas convolucionales en la red se empleará tamaños de filtros de 3×3, un valor de stride de 1×1 con Zero - Padding. El número de filtros variará para cada uno de los bloques convolucionales y se resume en la Tabla 2.1.

```
tf.keras.layers.Conv2D( filters=32,
                        kernel_size = (3, 3),
                        strides=(1, 1),
                        padding='valid')
```

Código 4: Implementación de la capa convolucional con la librería de Keras. La capa contiene 32 filtros de tamaño 3×3 con un stride de 1×1 y utiliza la técnica de Zero-Padding

2.3.1.2. Capa Normalización de lotes (Batch Normalization)

Esta capa debe ser apilada en cada bloque convolucional para estandarizar las entradas en un lote a una media de cero y una desviación estándar de uno. Durante el entrenamiento de la red, esta capa realizará un seguimiento de las estadísticas de cada variable de entrada y las utilizará para estandarizar los datos. Al final del entrenamiento, las estadísticas de desviación estándar y media en la capa en ese momento se usarán para estandarizar las entradas cuando el modelo se use para hacer una predicción.

Entre los parámetros importantes que incluye Keras en esta capa se encuentra el ”momentum” (α), el cual permite aproximar la media y la varianza que se obtiene en cada mini lote al valor de la media y varianza que se obtendría al calcularla sobre toda la población de datos. Una forma de ajustar estos valores es determinar el valor de la media y varianza de la población completa utilizando todas las medias y las varianzas de los mini lotes. Pero, a veces, es difícil realizar un seguimiento de todas las variaciones y medias en los mini lotes, por lo que es conveniente plantear una ponderación exponencial para actualizar la media y la varianza de la población. En el algoritmo 2 y 3 se detalla el proceso de normalización por lotes considerando el parámetro momentum [58].

De esta forma el momentum significa el retraso en el aprendizaje de la media y la varianza, de modo que se puede ignorar el ruido debido al mini lote. De forma predeterminada, el momentum se establece en un valor alto de aproximadamente $0,99$, lo que significa un retraso alto y un aprendizaje lento. Cuando se tiene tamaños de lote pequeños, el número de pasos en una época serán más, por lo que se necesita un valor de momentum alto, lo que dará como resultado un aprendizaje lento pero constante de la media móvil [58].

Pero cuando el tamaño del lote es más grande, el número de pasos es menor en cada

Input: Lote de entrada $\mathbf{B}_{\text{in}}$. Promedio μ y varianza ν estimada, y momentum α.

$$\mu \leftarrow (1-\alpha)\cdot\mu + \alpha\cdot \textbf{\textit{mean}}(\mathbf{B}_{\text{in}}),$$
$$\nu \leftarrow (1-\alpha)\cdot\nu + \alpha\cdot \textbf{\textit{var}}(\mathbf{B}_{\text{in}}).$$

$\textbf{\textit{mean}}(\mathbf{B}_{\text{in}})$ y $\textbf{\textit{var}}(\mathbf{B}_{\text{in}})$ calculan el promedio y la varianza de $\mathbf{B}_{\text{in}}$ respectivamente.

Salida:

$$\mathbf{B}_{\text{out}} \leftarrow \frac{\mathbf{B}_{\text{in}} - \textbf{\textit{mean}}(\mathbf{B}_{\text{in}})}{\sqrt{\textbf{\textit{var}}(\mathbf{B}_{\text{in}}) + \epsilon}}.$$

ϵ es una pequeña constante para la estabilidad numérica.

Algoritmo 2: Normalización por lotes (Fase de entrenamiento) [58].

Input: Lote de entrada $\mathbf{B}_{\text{in}}$, promedio μ y varianza ν estimada. Una pequeña constante ϵ para estabilidad numérica.

Salida:

$$\mathbf{B}_{\text{out}} \leftarrow \frac{\mathbf{B}_{\text{in}} - \mu}{\sqrt{\nu + \epsilon}}$$

Algoritmo 3: Normalización por lotes (Fase de predicción) [58].

época, lo implica que las estadísticas del mini lotes son en su mayoría las mismas que las de la población. En esta situación, el valor de momento debe ser menor, de modo que la media y la varianza se actualicen rápidamente [58].

El valor de momentum que se empleará en cada bloque convolucional, es el valor predeterminado en la función de keras igual a $0,99$, ya que en la fase de entrenamiento de la red se utilizará tamaños de lote pequeños en comparación al tamaño total del conjunto de entrenamiento. Keras modeliza la capa de Normalización por lotes a través de la clase **tf.keras.layers.BatchNormalization()**, la misma que se detalla en el Código 5 y recibe los siguiente argumentos:

- **axis:** Especifica los canales del mapa de características que se debe normalizar. Si se ingresa el valor "-1" se normaliza todos los canales.
- **momentum:** Valor del momentum para determinar la media y varianza móvil.
- **epsilon:** Un valor flotante pequeño agregado a la varianza para evitar dividir por cero.
- **center/scale:** Especifica que se efectúa una normalización en cada mini-lote en media de cero y varianza unitaria.
- Se especifica los valores de inicialización para los pesos de beta, gamma, media móvil, y la variación móvil para el proceso de entrenamiento de la red neuronal convolucional.

```
tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001,
    center=True, scale=True, beta_initializer='zeros', gamma_initializer='ones',
    moving_mean_initializer='zeros', moving_variance_initializer='ones')
```

Código 5: Implementación de la capa normalización por lotes con Keras.

2.3.1.3. Capa Leaky RELU

Keras modeliza esta capa a través de la clase **tf.keras.layers.LeakyReLU()**, y su único parámetro es la pendiente de la función de activación para valores negativos. Se implementó esta capa dentro de los bloques convolucionales por las ventajas que presentan sobre el problema del desvanecimiento del gradiente como se explicó en la sección 1.4.6.2. En el Código 6 se presenta la estructura de la modelización de la capa ReLU con Keras.

```
tf.keras.layers.LeakyReLU( alpha=0.3 )
```

Código 6: Implementación de la capa LeakyRELU con Keras.

Para la construcción de los bloques convolucionales se utilizó el valor por defecto de $0{,}3$ como pendiente de la función de activación LeakyReLU para valores negativos.

2.3.1.4. Capa Avarage Pooling

Keras modeliza esta capa a través de la clase **tf.keras.layers.AveragePooling2D().** Como se mencionó en la sección 1.4.6.3, los hiperparámetros de esta capa son el tamaño de la agrupación, el valor del stride y el tamaño del Padding. En el código 7 se muestra los parámetros que emplea Keras para modelizar la capa de Avarage Pooling.

```
tf.keras.layers.AveragePooling2D(pool_size=(2, 2),
                                 strides=(2, 2), padding='valid')
```

Código 7: Implementación de la capa Max Pooling con Keras.

Para el diseño de la red, cada capa de Avarage Pooling de cada bloque convolucional se empleó los siguientes valores de hiperparámetros:

- Tamaño del pool: 2×2
- Stride: 2×2
- No se considera la técnica de Padding

2.3.1.5. Capa Dropout

Se añade una capa de Dropout en cada uno de los bloques convolucionales como método para evitar el problema de overfitting (sobre ajuste) con el entrenamiento de la Red Neuronal Convolucional. Keras genera esta capa de Dropout con la clase **tf.keras.layers.Dropout()** la misma que recibe como argumento la tasa para el dropout como se detalla en el Código 8. En cada uno de los bloques convolucionales se utiliza una tasa de Dropout de $0{,}1$.

```
tf.keras.layers.Dropout(rate = 0.1)
```

Código 8: Implementación de la capa Dropout con Keras.

En la tabla 2.1 se muestra la estructura y los hyperparámetros de las capas empleadas en la red convolucional.
La concatenación de las dos ramales se conecta a una red perceptrón multicapa con tres capas densamente conectadas con las siguientes cantidades de neuronas:

- Capa 1: 1024 neuronas
- Capa 2: 512 neuronas
- Capa 3: 256 neuronas

Cada neurona de las capas densamente conectadas utilizan la función de activación lineal RELU. Como capa de salida de la red se utiliza una capa de densa con función de activación Softmax. La función de activación Softmax o función exponencial normalizada transforma un vector de números en un vector de probabilidades, las mismas que son proporcionales a la escala relativa de cada valor en el vector. La función Softmax se utiliza en modelos multiclase donde devuelve probabilidades de cada clase, siendo la clase objetivo la que tiene la probabilidad más alta.
La ecuación 2.6 calcula la distribución de probabilidad a partir de un vector de números reales de dimensión K.

$$\begin{aligned} \sigma &: \mathbb{R}^K \to [0,1]^K \\ \sigma(\mathbf{z})_j &= \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j \in [1,K] \end{aligned} \tag{2.6}$$

Tabla 2.1: Estructura de la etapa de extracción de mapa de características en los dos ramales de la red convolucional.

Bloque	Capas	Hiperparámetros	Tamaño mapa de características
BC1	**Convolucional**	Número filtros: 32	64 x 64 x 32
		Tamaño filtros: 3 x 3	
		Stride: 1 x 1	
		Zero-Padding	
	Batch Normalization	Momentum = $0{,}99$	
		$\epsilon = 0{,}001$	
		No renormalización	
	Leaky ReLU	$\alpha = 0{,}3$	
	Avarage Pooling	Tamaño pool: 2 x 2	32 x 32 x 32
		No Padding	
	Dropout	Tasa: $0{,}1$	
BC2	**Convolucional**	Número filtros: 32	32 x 32 x 32
		Tamaño filtros: 3 x 3	
		Stride: 1 x 1	
		Zero-Padding	
	Batch Normalization	Momentum = $0{,}99$	
		$\epsilon = 0{,}001$	
		No renormalización	
	Leaky ReLU	$\alpha = 0{,}3$	
	Avarage Pooling	Tamaño pool: 2 x 2	16 x 16 x 32
		No Padding	
	Dropout	Tasa: $0{,}1$	
BC3	**Convolucional**	Número filtros: 64	16 x 16 x 64
		Tamaño filtros: 3 x 3	
		Stride: 1 x 1	
		Zero-Padding	
	Batch Normalization	Momentum = $0{,}99$	
		$\epsilon = 0{,}001$	
		No renormalización	
	Leaky ReLU	$\alpha = 0{,}3$	
	Avarage Pooling	Tamaño pool: 2 x 2	8 x 8 x 64
		No Padding	
	Dropout	Tasa: $0{,}1$	
BC4	**Convolucional**	Número filtros: 64	8 x 8 x 64
		Tamaño filtros: 3 x 3	
		Stride: 1 x 1	
		Zero-Padding	
	Batch Normalization	Momentum = $0{,}99$	
		$\epsilon = 0{,}001$	
		No renormalización	
	Leaky ReLU	$\alpha = 0{,}3$	
	Avarage Pooling	Tamaño pool: 2 x 2	4 x 4 x 64
		No Padding	
	Dropout	Tasa: $0{,}1$	

En el Código 9 se muestra la concatenación de los ramales de la red convolucional a las capas densamente conectadas.

```
# Concatenar los dos ramales
combined = Concatenate()([x.output, y.output])
combined = Flatten()(combined)

# Red perceptrón multica
dense1= tf.keras.layers.Dense(1024, activation='relu')(combined)
dense2 = tf.keras.layers.Dense(512,activation='relu')(dense1)
dense3 = tf.keras.layers.Dense(256,activation='relu')(dense2)

# Capa de salida con función softmax
predictions = tf.keras.layers.Dense(3, activation='softmax')(dense3)
```

Código 9: Implementación de las capas densamente conectadas con Keras.

2.3.2. ENTRENAMIENTO DE LA RED NEURONAL CONVOLUCIONAL

Para el proceso de entrenamiento, validación y evaluación de la red neuronal se utilizó la técnica de validación cruzada de K iteraciones (K-Fold cross validation) y un conjunto de datos compuesto por $8\,000$ regiones de texto, $8\,000$ regiones de tabla y $8\,000$ regiones de figura. De esta cantidad de regiones, se extrajeron los espectrogramas de sus respectivas proyecciones verticales y horizontales para entrenar la arquitectura detallada anteriormente en la Sección 2.3.

La validación cruzada de K iteraciones es un método estadístico que permite estimar el performance de un modelo de Aprendizaje Automático. Para lo cual se divide el conjunto de datos en K subconjuntos del mismo tamaño de forma aleatoria denominados ”Fold”. De esta manera, de los K subconjuntos se toman $K - 1$ subconjuntos tanto para la fase de entrenamiento como la de validación y el subconjunto sobrante se lo utiliza para la fase de evaluación. El proceso descrito anteriormente se repite K veces considerando en cada iteración un subconjunto distinto para la fase de evaluación. Al efectuar esta técnica se genera K estimaciones en las métricas de evaluación cuyo promedio se emplea como estimación final del performance del modelo.

Se optó el uso de la validación cruzada de K iteraciones sobre la red neuronal convolucional porque esta técnica obtiene una estimación precisa del performance de un modelo de Aprendizaje Automático ya que logra un mejor balance entre el bias y la varianza del modelo. El bias es la diferencia que existe entre la predicción promedio de un modelo y el valor correcto que el modelo trata de predecir. Se tiene un alto valor de bias cuando el modelo

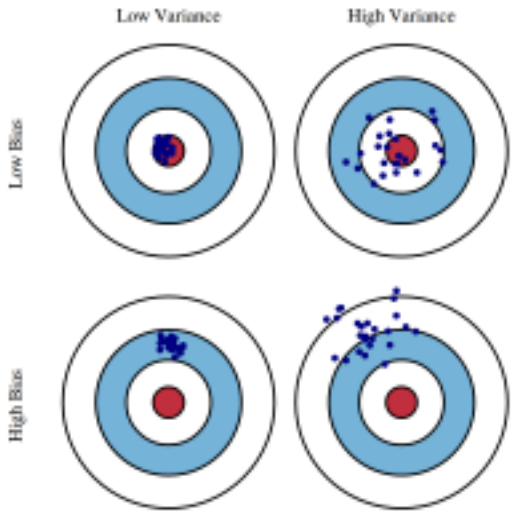


Figura 2.15: Ilustración del bias y varianza de un modelo de Aprendizaje Automático [59]

no se ajusta correctamente a los datos del conjunto de entrenamiento y conduce a que el modelo tenga altas tasas de errores en la predicción sobre observaciones que pertenecen al conjunto de entrenamiento, validación y prueba. Mientras que la varianza hace referencia a la variabilidad de la predicción de un modelo para un punto de datos dado. Así se tiene una alta varianza cuando el modelo se ajusta demasiado a los datos de entrenamiento y no se logra una generalización sobre datos que el modelo no había visto antes. Como resultado, estos modelos funcionan correctamente en los datos de entrenamiento, pero generan altas tasas de error en la predicción de observaciones del conjunto de los datos de validación y prueba. Así, el bias mide la precisión promedio de la estimación del error, mientras que la varianza mide la variabilidad de la estimación del error del modelo [59]. En la Figura 2.15 se ilustra el balance entre bias y varianza de un modelo de aprendizaje.

La compensación bias-varianza está asociado con la elección de K ya que un valor mal determinado para K puede provocar una estimación errónea del performance del modelo, que se puede manifestar en una alta varianza o un alto bias. Al incrementar el valor de K se presentan los siguientes efectos:

- El bias del estimador de desempeño disminuye. Es decir se tiene un modelo más preciso.
- La varianza de los estimadores de desempeño aumenta. El modelo tiene más variabilidad en la predicción sobre el conjunto de datos de prueba.
- El costo computacional aumenta (más iteraciones, conjuntos de entrenamiento más grandes durante el ajuste del modelo).

Mientras que al disminuir el valor de K a valores pequeños (por ejemplo, $K = 2$ ó $K = 3$) también aumenta la varianza en conjuntos de datos pequeños debido a efectos de muestreo aleatorio [59].

Típicamente se asigna valores de $K = 5$ ó $K = 10$, pero no existe una regla formal para determinar su valor. Por lo general, tomando en cuenta las consideraciones anteriores, se realizan la validación cruzada empleando $K = 5$ ó $K = 10$, ya que se ha demostrado empíricamente que con estos valores se obtienen estimaciones del error de un modelo sin un bias excesivamente alto ni una varianza muy alta [60].

El entrenamiento de la red neuronal convolucional se realizó con un tamaño de 5 Folds($K = 5$). En cada iteración, el conjunto de datos de divide en 5 subgrupos, de los cuales 4 subgrupos se utiliza para entrenamiento ($80\,$% de los datos de los 4 subgrupos) y validación($20\,$% de los datos de los 4 subgrupos) de la red neuronal convolucional y el último subgrupo se utiliza como evaluación del modelo.

Además, el entrenamiento considera la optimización de la función de pérdida con descenso de gradiente por lotes pequeños, por las siguientes ventajas:

- La frecuencia de actualización de los parámetros y pesos en cada una de las capas del modelo es alta, lo que permite una convergencia más robusta, evitando los mínimos locales.
- Permite tanto la eficiencia computacional de no tener todos los datos de entrenamiento en memoria como las implementaciones de algoritmos de optimización.

El tamaño del lote debe ser seleccionado cuidadosamente ya que con tamaños pequeños el proceso de aprendizaje de un modelo que converge rápidamente a costa del ruido en el proceso de entrenamiento. Mientras que valores grandes el proceso de aprendizaje converge lentamente con estimaciones precisas del error del gradiente [61], [62]. De esta forma en cada iteración de la validación cruzada se considera un tamaño de lote de entrenamiento de $1\,260$ muestras y un lote de validación de 315 muestras.

De esta manera, través del algoritmo Adam (Adaptive Moment Estimation) [63] se realiza la optimización de la función de pérdida Entropía Cruzada(Cross Entropy Loss) con el descenso de gradiente. La entropía cruzada se basa en la idea de entropía de la Teoría de la Información y calcula el número de bits necesarios para representar o transmitir un evento promedio de una distribución de probabilidad en comparación con otra distribución, es decir, es una medida de la diferencia entre dos distribuciones de probabilidad para una determinada variable aleatoria o un conjunto de eventos [64].

En el Código 10 se muestra la implementación del optimizador Adam con la librería **Keras**.

```
model.compile( Adam(lr=0.001),
               loss='categorical_crossentropy',
               metrics=[categorical_accuracy])
```

Código 10: Optimizador Adam con una tasa de aprendizaje inicial de $0{,}001$ y con la función de pérdida de Entropía Cruzada.

Otro aspecto importante en el entrenamiento de la red neuronal convolucional es el uso de un cronograma en la tasa de aprendizaje (learning rate schedule) con el objetivo de aumentar el rendimiento y reducir el tiempo de entrenamiento.

El cronograma de la tasa de aprendizaje empleado en el entrenamiento es la disminución de la tasa de aprendizaje gradualmente en función de la época a partir de un valor de $0,001$ hasta $0,0001$ de forma cíclica con un periodo de 25 épocas como se muestra en el Código 11. Para el entrenamiento de la red se utiliza el cronograma Cosine Annealing, ya que este método tiene el efecto de comenzar el entrenamiento de la red con una gran tasa de aprendizaje que se reduce relativamente rápido a un valor mínimo antes de volver a aumentar drásticamente. En la ecuación 2.7 se muestra el cronograma de la tasa de aprendizaje con Cosine Annealing, donde η^i_{min} y η^i_{max} son los rangos de la tasa de aprendizaje y T_{cur} representa cuántas épocas se han realizado desde el último reinicio.

$$\eta_t = \eta^i_{min} + \frac{1}{2}\left(\eta^i_{max} - \eta^i_{min}\right)\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right) \tag{2.7}$$

Con este cronograma de la tasa de aprendizaje la actualización de pesos en la Red Neuronal Convolucional está sujeto a cambios drásticos durante el entrenamiento, lo que tiene el efecto de utilizar ”buenos valores de pesos” como punto de partida para el ciclo de tasa de aprendizaje subsiguiente, evitando que la red converja hacia un óptimo local [65]. En la Figura 2.16 se detalla el cronograma de tasa de aprendizaje Cosine Annealing utilizado en el entrenamiento de la Red Neuronal Convolucional en cada iteración de la validación cruzada.

Además, como estrategia de regularización, se considera la técnica de parada temprana (Early Stopping) para evitar el sobreajuste en el proceso de entrenamiento de la red. Es decir habrá un punto durante el entrenamiento en el que el modelo dejará de generalizarse y comenzará a aprender el ruido estadístico en el conjunto de datos de entrenamiento. Este sobreajuste del conjunto de datos de entrenamiento dará como resultado un aumento en

```
schedule = CyclicalSchedule(CosineAnnealingSchedule,
                            min_lr=0.0001,
                            max_lr=0.001,
                            cycle_length=20)

cyclical_schedule = tf.compat.v1.keras.callbacks.LearningRateScheduler(schedule,
                                    verbose=1)
```

Código 11: Cronograma de tasa de aprendizaje Cosine Annealing implementado en Keras.

el error de generalización, haciendo que el modelo sea menos útil para hacer predicciones sobre nuevos datos. Este sobreajuste del conjunto de datos de entrenamiento dará como resultado un aumento en el error de generalización, haciendo que el modelo sea menos útil para hacer predicciones sobre nuevos datos [66]. Para el entrenamiento de la Red Neuronal Convolucional la métrica configurada para deterner el entrenamiento se realiza cuando la exactitud (accuracy) en la validación no mejora en 0. 02 % dentro de 25 épocas como se detalla en el Código 12.

```
tensorflow.compat.v1.keras.callbacks.EarlyStopping(monitor='val_categorical_accuracy',
                                                   min_delta=0.0002,
                                                   patience=25,
                                                   verbose=1)
```

Código 12: Técnica de EarlyStopping considerado en el entrenamiento de la Red Neuronal Convolucional.

En la Figura 2.17 se resume los valores de Exactitud (Accuracy) alcanzado en el entrenamiento y validación para los 5 folds, así como también la optimización de la función de pérdida de Entropía Cruzada (Cross Entropy Loss) para los 5 folds.

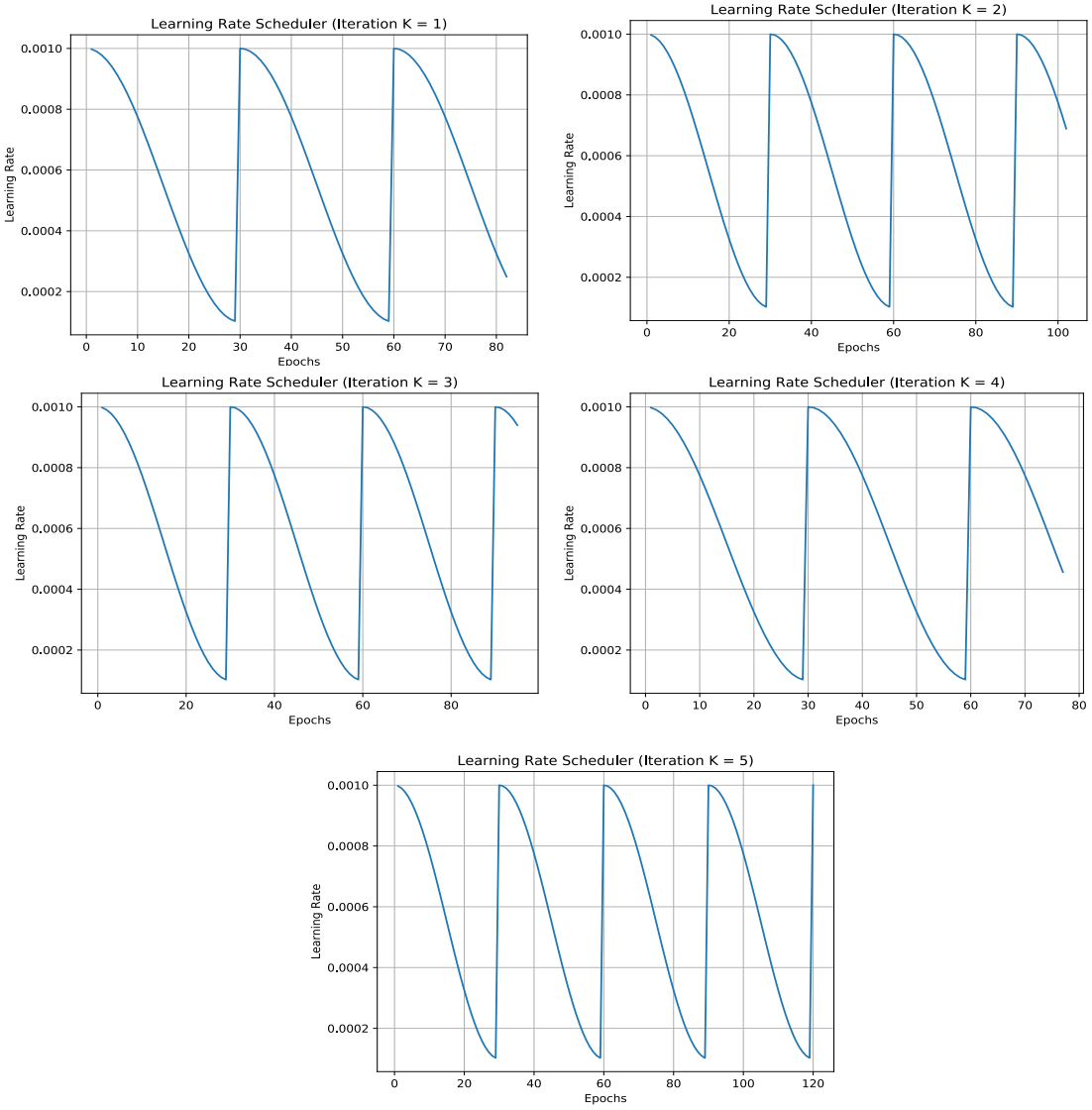


Figura 2.16: Cronograma de la tasa de aprendizaje para el entrenamiento de la Red Neuronal Convolucional para cada iteración de K-Fold Cross Validation

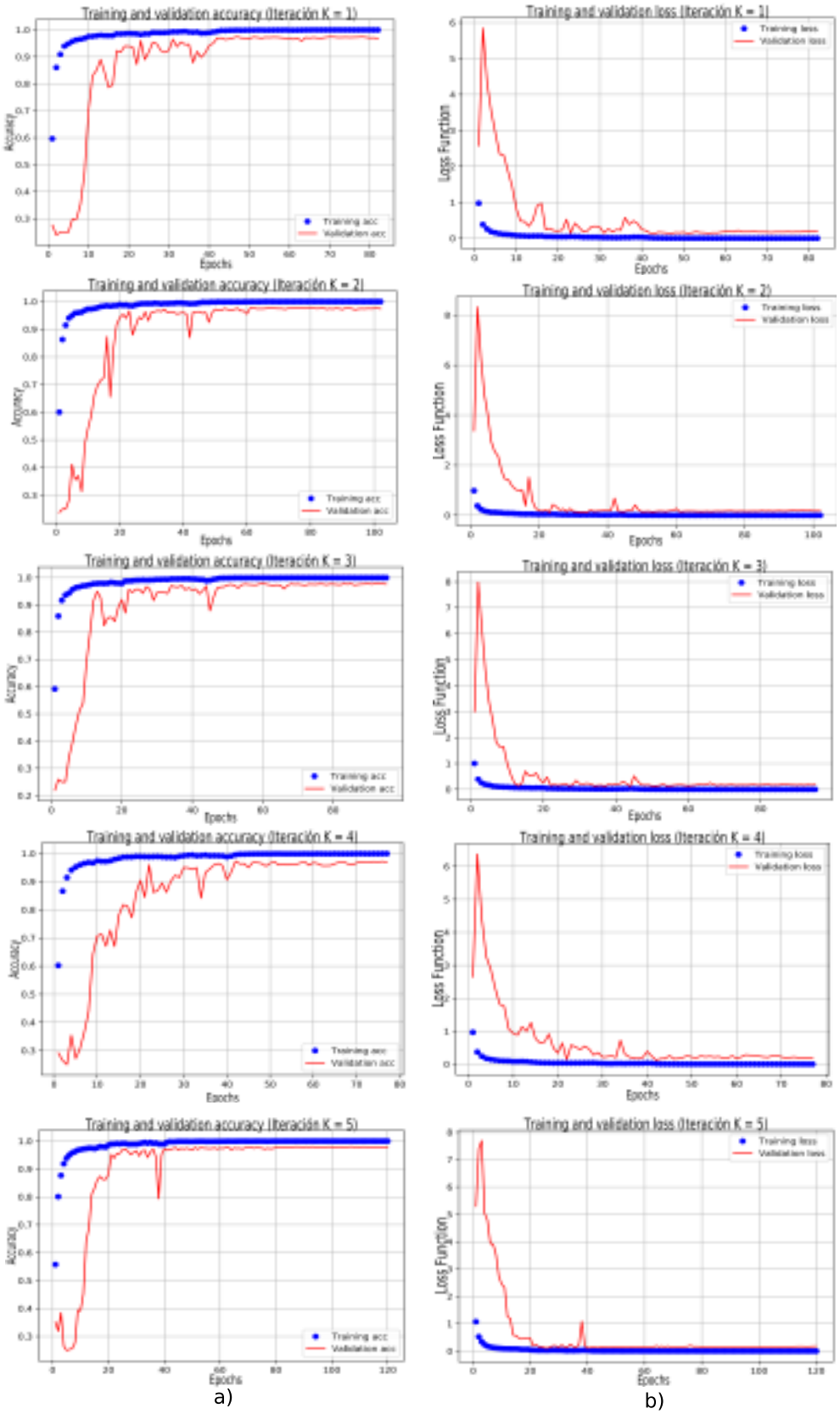
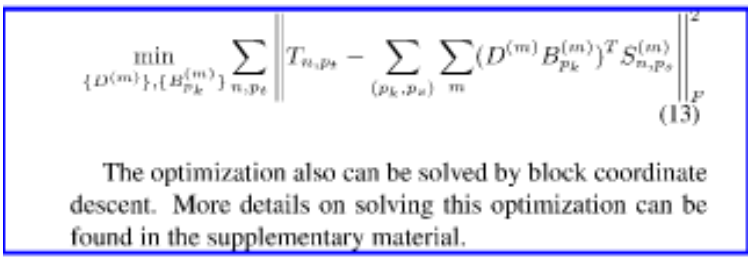


Figura 2.17: a) Comportamiento del Acuraccy en el entrenamiento y fase de validación en cada iteración de K-Fold Cross Validation. b) Optimización de la Función de pérdida en el entrenamiento y fase de validación para cada iteración de K-Fold Cross Validation

2.4. IDENTIFICACIÓN DE LÍNEAS DE ECUACIONES EN REGIONES DE TEXTO

Después de la clasificación de regiones con la Red Neuronal Convolucional, se observa que las regiones clasificadas como texto contienen líneas con ecuaciones como se muestra en la Figura 2.18. Por esta razón es necesario añadir un etapa adicional de tal forma que se permita discriminar líneas de ecuaciones presentes en un región de texto.



$$\min_{\{D^{(m)}\},\{B_{p_k}^{(m)}\}} \sum_{n,p_t} \left\| T_{n,p_t} - \sum_{(p_k,p_s)} \sum_{m} (D^{(m)} B_{p_k}^{(m)})^T S_{n,p_s}^{(m)} \right\|_F^2 \quad (13)$$

The optimization also can be solved by block coordinate descent. More details on solving this optimization can be found in the supplementary material.

Figura 2.18: Región clasificada como texto que tiene una línea de ecuación

Para este propósito las regiones de texto son segmentadas en líneas. Luego, se extrae un vector de características globales para cada línea segmentada, que se construye concatenando dos vectores. El primer vector almacena información estadística extraída directamente de los píxeles que componen la línea y el segundo es un diccionario construido a partir de la información obtenida con los momentos Zernike de cada símbolo encontrado en una línea. Finalmente, el vector de características global se envía a un clasificador de Máquina de Vectores de Soporte (Support Vector Machine).

2.4.1. SEGMENTACIÓN DE LÍNEAS EN UNA REGIÓN DE TEXTO

Para la segmentación del bloque en líneas se realiza una dilatación sobre la región binarizada con un kernel rectangular horizontal, con un tamaño vertical de 5 píxeles y un tamaño horizontal igual al ancho en píxeles del bloque de texto. La dilatación se efectuó con la función **cv2.dilate()** de la librería OpenCV con Phyton. En el Código 13 muestra un ejemplo de dilatación de una región de texto binarizada.

Posteriormente, las coordenadas de los cuadros delimitadores que encierran la línea se generan mediante el algoritmo de seguimiento de bordes por el método de datos de ejecución descrito en la Sección 1.4.2.3 la cual se encuentra implementada en la biblioteca OpenCV. El seguimiento de bordes se ejecuta con la función **cv2.findContours()**, la misma que re-

```
path_dir_image = '/home/erick/DocLayout/BlockText.png'
# Binarización de una imagen RGB
ImageRGB = cv2.imread(path_dir_image)
gray_scale = cv2.cvtColor(ImageRGB, cv2.COLOR_BGR2GRAY)
Binary_region = cv2.adaptiveThreshold(  gray_scale,
                                        255,
                                        cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                        cv2.THRESH_BINARY_INV,11,2)
# Dilatación de una región binarizada
kernel = np.ones((5,Binary_region.shape[1]))
text_dilat = cv2.dilate(Binary_region, kernel, iterations=1)
```

Código 13: Dilatación de una imagen binarizada con las librerías Numpy y OpenCV.

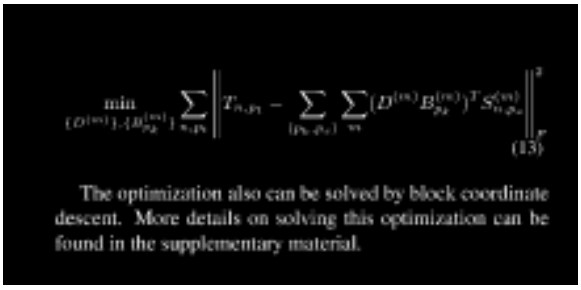
cupera los contornos de la imagen binaria usando el algoritmo desarrollado en [67]. El uso de la función **cv2.findContours()** se detalla en el Código 14 y recibe los siguientes tres parámetros de entrada:

- La imagen binarizada
- Modo de recuperación de contorno: La opción RETR_EXTERNAL recupera solo los contornos exteriores extremos. La opción RETR_CCOMP recupera todos los contornos y los organiza en una jerarquía de dos niveles. El nivel superior retorna los contornos externos de los componentes y el segundo nivel los bordes interiores.
- Método de aproximación de contorno: La opción CHAIN_APPROX_SIMPLE comprime segmentos horizontales, verticales y diagonales y deja solo sus puntos finales. Por ejemplo, un contorno rectangular se codifica con los 4 puntos de sus vértices. Si se selecciona opción CHAIN_APPROX_NONE almacena absolutamente todos los puntos del contorno.

```
contours, hierarchy = cv2.findContours( Binary_region,
                                        cv2.RETR_EXTERNAL,
                                        cv2.CHAIN_APPROX_SIMPLE)
```

Código 14: Generación de coordenadas de cuadros delimitdores con OpenCV.

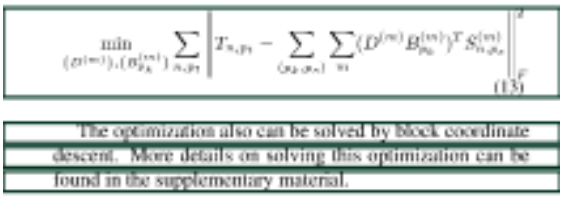
La Figura 2.19 proporciona el resultado de la segmentación de un bloque en líneas.



(a) Región de Texto binarizada.

(b) Dilatación con kernel rectangular de $5 \times$ ancho bloque de texto.

(c) Trazo de bordes con método de ejecución de datos.



(d) Coordenadas de los cuadros delimitadores de cada línea.

Figura 2.19: Resultado del proceso de segmentación de la línea.

2.4.2. EXTRACCIÓN DE CARACTERÍSTICAS DE LAS LÍNEAS DE UNA REGIÓN DE TEXTO

Una vez efectuado la segmentación de una región de texto en líneas se genera un vector global de características por cada línea segmentada para realizar una clasificación binaria con el algoritmo Support Vector Machine. El vector global de características tiene una dimensión de 259 elementos que representan la siguiente información:

- Los 3 primeros elementos del vector representan información estadística extraída directamente de la geometría presente en los píxeles que componen una línea.
- Los 256 elementos son el resultado de la técnica Bag of Visual Words considerando como extractor de características los momentos de Zernike de los símbolos contenidos en cada una de las líneas de una región de texto.

2.4.2.1. Extracción de características estadísticas

Para diferenciar una línea de texto y una línea con símbolos matemáticos se consideró las siguientes tres características locales:

Fluctuación de los centros: Esta característica evalúa la variación del centro del cuadro delimitador de cada símbolo contenido en una línea. Este parámetro se detalla en un estudio

previo [68] y la ecuación 2.8 define matemáticamente este parámetro.

$$f_c = \frac{1}{n}\sum_{i=1}^{n-1}\left|cos^{-1}\left(\frac{\vec{\mathbf{v}_i}.\vec{\mathbf{v}_h}}{||\vec{\mathbf{v}_i}||.||\vec{\mathbf{v}_h}||}\right)\right| \tag{2.8}$$

La ecuación 2.8 tiene sentido si se coloca un sistema de coordenadas cartesiano en el borde superior izquierdo de cada línea extraída y que cada píxel en la línea sea considerado como una coordenada dentro del sistema cartesiano. Así, el centro de un símbolo está determinado por la coordenada (x, y) del centro del cuadro delimitador que encierra el símbolo como se observa en la Figura 2.20 donde cada centro se representa con puntos de color rojo. El vector $\vec{\mathbf{v}_i}$ es generado desde el centro del i-ésimo símbolo hasta el centro del $(i+1)$-ésimo símbolo. El vector $\vec{\mathbf{v}_h}$ es un vector solo con componente horizontal cuya magnitud es x_m que es la media de las coordenadas x de todos los centros de los símbolos y n es el número de símbolos en la línea. Todos estos parámetros se ilustran en la Figura 2.20 considerando una línea de ecuación.

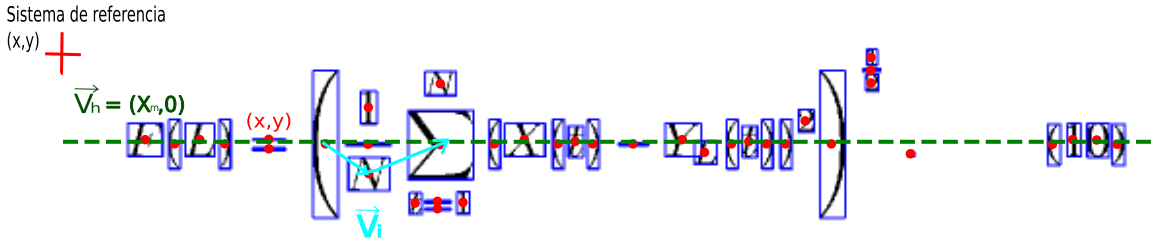


Figura 2.20: Fluctuación de los centros de los cuadros delimitadores de una línea de ecuación

De esta forma se obtiene una alta fluctuación en las líneas con símbolos matemáticos en comparación con las líneas con símbolos alfabéticos.

Varianza de la altura de los símbolos: De manera intuitiva, esta característica captura la idea de que una línea con símbolos matemáticos genera una alta varianza en las alturas de los cuadros delimitadores de cada símbolo en comparación con una línea con símbolos alfabéticos. La varianza de la alturas de los cuadros delimitadores (h_i) se determinó con la ecuación 2.9.

$$\sigma_h^2 = \frac{1}{n}\sum_{i=1}^{n}(h_i - \bar{H})^2 \tag{2.9}$$

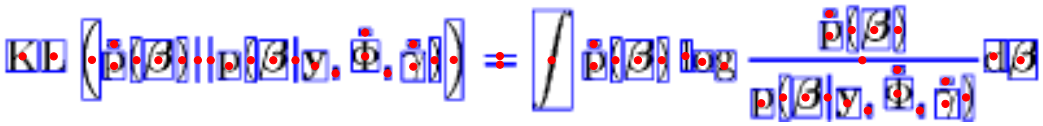
Densidad de píxeles: Este parámetro evalúa la densidad de los píxeles del primer plano con respecto al total de píxeles contenidos dentro del cuadro delimitador de la línea extraída. De esta forma, se ha observado previamente que existe una baja densidad para líneas con símbolos matemáticos.

La ecuación 2.10 se utilizó para calcular la densidad de píxeles de primer plano:

$$\delta = \frac{N^o \quad \text{de píxeles de primer plano (foreground)}}{N^o \quad \text{total de píxeles}} \tag{2.10}$$

En la Figura 2.21 se muestra una línea de texto y una línea con símbolos matemáticos sobre los cuales se extrajeron las tres características mencionadas anteriormente y se las resume en la Tabla 2.2.

(a) Línea de texto



(b) Línea de ecuación

Figura 2.21: Extracción de características para una línea de texto y ecuación.

Tabla 2.2: Fluctuación de centros, varianza de altura y densidad de píxeles para los ejemplos de la Figura 2.21

Característica	Línea de texto	Línea con ecuación
Fluctuación de centros (f_c)	0.36834	0.65452
Varianza de altura (σ_h^2)	123.41379	907.54630
Densidad de píxeles (δ)	0.12879	0.04719

2.4.2.2. Bag of Visual Words con los momentos de Zernike

Con la técnica Bag of Visual Words se construye un vocabulario visual para representar cada línea de una región de texto con un vector que representa un histograma de los símbolos contenidos en cada línea. Para generar este vector se consideran tres etapas: Extracción de características utilizando los momentos de Zernike, construcción de un diccionario de palabras visuales y codificación de características.

Extracción de patrones con los momentos de Zernike: Considerando la Sección 1.4.4 donde se detalló que la etapa de extracción de patrones en la técnica Bag of Visual Words se logra con la ayuda de descriptores sobre regiones clave de una imagen, se ha seleccionado como descriptor los momentos de Zernike sobre cada uno los símbolos. Al utilizar los momento de Zernike es posible representar la información de forma de cada uno de los símbolos sin redundancia o superposición de información entre los momentos Zernike. De esta

manera, se calcula los momentos de Zernike de cada símbolo considerando invariancia en rotación, en traslación y en escala. La invarianza de rotación es inherente de los momentos de Zernike, por lo que resta garantizar la invarianza de traslación y escala.

La invariancia de traslación se logra transformando la imagen en una nueva cuyos momentos centrales de primer orden (m'_{10} y m'_{01}), son ambos iguales a cero. En forma general los momentos centrales de orden p, q de una imagen $f(x, y)$ de tamaño $N \times N$ píxeles se define con la ecuación 2.11.

$$m_{p,q} = \sum_{x=1}^{N} \sum_{y=1}^{N} x^p y^q f(x, y) \tag{2.11}$$

Entonces la invarianza en traslación implica calcular los momentos de Zernike mapeando la imagen primero a un disco de radio unitario usando coordenadas polares cuyo centro corresponde al centroide de la imagen. Ignorando en el cálculo los píxeles de la imagen que se encuentran fuera del disco.

La invarianza de escala se obtiene redimensionando la imagen original de modo que el momento central de orden cero de la nueva imagen se establezca igual a un valor predeterminado $m'_{00} = \beta$. Para el caso de imágenes binarias, m_{00} es el número total de píxeles de primer plano (foreground) en la imagen.

En resumen una imagen $f(x, y)$ se normaliza en escala y traslación transformándola a una nueva imagen $g(x, y)$ tal que:

$$g(x, y) = f\left(\frac{x}{a} + \bar{x}, \frac{y}{a} + \bar{y}\right) \tag{2.12}$$

En la ecuación 2.12 el punto $(\bar{x}, \bar{y})$ corresponde a las coordenadas del centroide de la imagen original, el cual se determina con los momentos centrales de primer orden (m_{10} y m_{01}) y orden cero (m_{00}), como se detalla en la ecuación 2.13.

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad \bar{y} = \frac{m_{01}}{m_{00}} \tag{2.13}$$

El factor a se obtiene con el valor predeterminado para el momento de orden cero de la nueva imagen $m'_{00} = \beta$ y con el momento de orden cero de la imagen original m_{00} a través de la ecuación 2.14.

$$a = \sqrt{\frac{\beta}{m_{00}}} \tag{2.14}$$

De esta forma, los momentos de Zernike se determinaron con los polinomios de quinto orden, generando 12 momentos de Zernike complejos para cada símbolo. Para la construcción del vector descriptor solo se consideran las 12 magnitudes de estos momentos complejos

ya que son las magnitudes las que poseen la propiedad de invarianza en rotación, escala y traslación. Por lo tanto, en una línea de una región de texto se obtiene el mismo número de vectores descriptores que los símbolos que contiene.

Debido a que los símbolos presentes en una región de texto son tamaño variable, se escogió un valor de $m'_{00} = \beta = 2 * 10^6$ para normalizar todos los símbolos dentro de una región de radio de 300 píxeles.

El cálculo de los momentos complejos de Zernike se determinan con la librería **Mahotas** de Python con la función **mahotas.features.zernike_moments()** que se detalla en el Código 15 y recibe los siguientes parámetros de entrada:

- Imagen binarizada.
- El radio máximo para los polinomios de Zernike, en píxeles. El área de la imagen fuera del círculo (centrada en el centro de masa) definida por este radio se ignora.
- Grado máximo de los momentos de Zernike a utilizar.
- El centro del disco. De forma predeterminada, se considera el centro de masa de la imagen.

```
1 mahotas.features.zernike(   binary_region,
2                             degree,
3                             radius,
4                             cm={center_of_mass(binary_region)})
```

Código 15: Cálculo de momentos de Zernike de una región binarizada con la librería Mahotas

En las tablas 2.3 y 2.4 se resume los momentos de Zernike para el símbolo matemático π (ver Figura 2.22) y para la letra G (ver Figura 2.23), considerando la invarianza en la rotación y en escala. Además se obtiene valores bajos en el el coeficiente de varianza $(\frac{\sigma}{\mu})$ para cada momento de Zernike, lo que comprueba la invarianza en rotación y en escala.

Generación de un diccionario de palabras visuales: En esta etapa se captura la distribución de probabilidad de los vectores descriptores a través de un modelo generativo para construir un vocabulario de palabras visuales. Para este propósito, como modelo generativo se selecciona a Gaussian Mixture Model (GMM) porque a través del algoritmo de Maximización de Expectativas (Expectation Maximization), es posible realizar una asignación suave

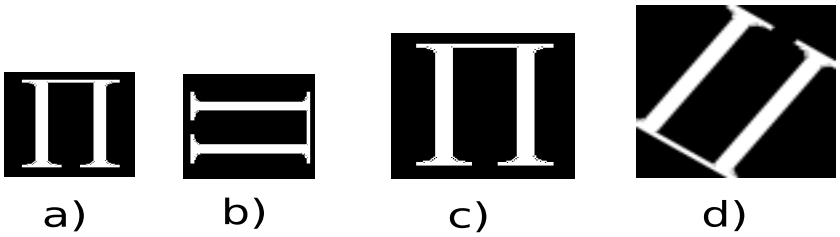


Figura 2.22: Símbolo π: a) Imagen original de 115×106 píxeles. b) Imagen de 115×106 píxeles rotada en -90^o. c) Imagen escalada a 345×318 píxeles. d) Imagen escalada a 805×742 píxeles y con rotación de 145^o

Tabla 2.3: Momentos complejos de Zernike con polinomios de quinto orden para el símbolo π de la Figura 2.22. La magnitud de los momentos se resalta en negrilla.

Orden Zernike	Original	Rotación	Escala	Rotación y escala	$\frac{\sigma}{\mu}$
$Z_{0,0}$ $\|\mathbf{Z_{0,0}}\|$	$318{,}309886 + 0j$ **318,309886**	$318{,}309886 + 0j$ **318,309886**	$318{,}309886 + 0j$ **318,309886**	$318{,}309886 + 0j$ **318,309886**	**0**
$Z_{1,1}$ $\|\mathbf{Z_{1,1}}\|$	$0 + 0j$ **0**	$0 + 0j$ **0**	$0 + 0j$ **0**	$0 + 0j$ **0**	**0**
$Z_{2,0}$ $\|\mathbf{Z_{2,0}}\|$	$-853{,}275909 + 0j$ **853,275909**	$-853{,}085117 + 0j$ **853,085117**	$-846{,}474133 + 0j$ **846,474133**	$-849{,}272083 + 0j$ **849,272083**	**0.003331**
$Z_{2,2}$ $\|\mathbf{Z_{2,2}}\|$	$-14{,}805394 - 0{,}083496j$ **14,805630**	$14{,}784746 + 0{,}085088j$ **14,784991**	$-15{,}180249 + 0{,}002828j$ **15,180249**	$-5{,}454336 + 13{,}435740j$ **14,500651**	**0.016294**
$Z_{3,1}$ $\|\mathbf{Z_{3,1}}\|$	$0{,}018414 - 0{,}472199j$ **0,472558**	$-0{,}572543 - 0{,}040526j$ **0,573976**	$-0{,}534373 - 0{,}605064j$ **0,807253**	$0{,}265398 + 0{,}319728j$ **0,415526**	**0.263857**
$Z_{3,3}$ $\|\mathbf{Z_{3,3}}\|$	$-0{,}025899 - 0{,}335960j$ **0,336957**	$0{,}290737 - 0{,}064344j$ **0,297772**	$-0{,}009898 - 0{,}316236j$ **0,316391**	$0{,}593766 - 0{,}049513j$ **0,595826**	**0.314196**
$Z_{4,0}$ $\|\mathbf{Z_{4,0}}\|$	$1118{,}397710 + 0j$ **1118,397710**	$1117{,}593470 + 0j$ **1117,593470**	$1089{,}295610 + 0j$ **1089,295610**	$1100{,}479030 + 0j$ **1100,479030**	**0.011040**
$Z_{4,2}$ $\|\mathbf{Z_{4,2}}\|$	$64{,}513464 + 0{,}356935j$ **64,514451**	$-64{,}412836 - 0{,}369882j$ **64,413898**	$65{,}182482 - 0{,}002757j$ **65,182482**	$23{,}318453 - 57{,}515516j$ **62,062749**	**0.018441**
$Z_{4,4}$ $\|\mathbf{Z_{4,4}}\|$	$-2{,}531681 + 0{,}002851j$ **2,531683**	$-2{,}567382 + 0{,}006649j$ **2,567391**	$-2{,}941631 - 0{,}000389j$ **2,941632**	$1{,}839458 + 1{,}456432j$ **2,346231**	**0.083220**
$Z_{5,1}$ $\|\mathbf{Z_{5,1}}\|$	$-0{,}092187 + 2{,}414533j$ **2,416292**	$2{,}925534 + 0{,}199177j$ **2,932307**	$2{,}882638 + 3{,}048908j$ **4,195884**	$-1{,}321491 - 1{,}607090j$ **2,080643**	**0.276635**
$Z_{5,3}$ $\|\mathbf{Z_{5,3}}\|$	$0{,}137265 + 1{,}837762j$ **1,842881**	$-1{,}599717 + 0{,}341037j$ **1,635665**	$0{,}181493 + 1{,}728265j$ **1,737769**	$-3{,}159992 + 0{,}267391j$ **3,171285**	**0.297872**
$Z_{5,5}$ $\|\mathbf{Z_{5,5}}\|$	$-0{,}001313 + 0{,}113808j$ **0,113816**	$0{,}116441 - 0{,}000057j$ **0,116441**	$0{,}045375 + 0{,}136377j$ **0,143728**	$-0{,}028212 + 0{,}166861j$ **0,169229**	**0.166227**

de los vectores descriptores a cada componente de la mezcla (cluster) en base a sus probabilidades a posteriori, es decir cada vector puede asignarse a más de un clúster con una cierta probabilidad. Además, GMM proporciona información de la media y la forma de la distribución de las palabras de código [69]. En resumen, cada cluster o grupo generado por GMM se considera una palabra visual que representa un patrón local específico compartido por los vectores descriptores en ese grupo. De esta manera, la agrupación (clusterización) con GMM crea un vocabulario visual de palabras que resume diferentes patrones locales ubicados en una línea de texto y el número de agrupaciones(clusters) determina el tamaño del vocabulario.

A través del Criterio de Información de Akaike (AIC) y el Criterio de Información Bayesiano (BIC), se determinó el número óptimo de componentes o cluster para Gaussian Mixture

Figura 2.23: Símbolo G: a) Imagen original de 124×118 píxeles. b) Imagen de 124×118 píxeles rotada en 65^o. c) Imagen escalada a 620×590 píxeles. d) Imagen escalada a 868×826 píxeles y con rotación de -155^o

Tabla 2.4: Momentos complejos de Zernike con polinomios de quinto orden para la letra ”G”de la Figura 2.23. La magnitud de los momentos se resalta en negrilla.

Orden Zernike	Original	Rotación	Escala	Rotación y escala	$\frac{\sigma}{\mu}$
$Z_{0,0}$ $\mathbf{\lvert Z_{0,0}\rvert}$	$318{,}309886 + 0j$ **318,309886**	$318{,}309886 + 0j$ **318,309886**	$318{,}309886 + 0j$ **318,309886**	$318{,}309886 + 0j$ **318,309886**	**0**
$Z_{1,1}$ $\mathbf{\lvert Z_{1,1}\rvert}$	$0 + 0j$ **0**	$0 + 0j$ **0**	$0 + 0j$ **0**	$0 + 0j$ **0**	**0**
$Z_{2,0}$ $\mathbf{\lvert Z_{2,0}\rvert}$	$-850{,}552441 + 0j$ **850,552441**	$-851{,}756608 + 0j$ **851,756608**	$-850{,}516465 + 0j$ **850,516465**	$-852{,}323331 + 0j$ **852,323331**	**0.000915**
$Z_{2,2}$ $\mathbf{\lvert Z_{2,2}\rvert}$	$0{,}897669 - 1{,}714460j$ **1,935247**	$-0{,}553256 + 1{,}455838j$ **1,557420**	$0{,}951192 - 1{,}772927j$ **2,011974**	$2{,}793141 - 0{,}857536j$ **2,921815**	**0.237880**
$Z_{3,1}$ $\mathbf{\lvert Z_{3,1}\rvert}$	$5{,}137201 + 2{,}022527j$ **5,521001**	$0{,}688691 + 5{,}756165j$ **5,797217**	$5{,}156158 + 2{,}079481j$ **5,559695**	$-3{,}877085 - 4{,}092539j$ **5,637434**	**0.018809**
$Z_{3,3}$ $\mathbf{\lvert Z_{3,3}\rvert}$	$-2{,}225808 - 1{,}074102j$ **2,471420**	$1{,}371137 + 1{,}388684j$ **1,951528**	$-2{,}199413 - 1{,}076420j$ **2,448693**	$-0{,}901655 + 1{,}856116j$ **2,063528**	**0.102894**
$Z_{4,0}$ $\mathbf{\lvert Z_{4,0}\rvert}$	$1101{,}204330 + 0j$ **1101,204330**	$1106{,}470190 + 0j$ **1106,470190**	$1101{,}053170 + 0j$ **1101,053170**	$1108{,}919820 + 0j$ **1108,919820**	**0.003075**
$Z_{4,2}$ $\mathbf{\lvert Z_{4,2}\rvert}$	$-5{,}880942 + 8{,}594308j$ **10,413818**	$2{,}904644 - 8{,}464992j$ **8,949472**	$-6{,}129164 + 8{,}874510j$ **10,785341**	$-14{,}327397 + 2{,}985690j$ **14,635186**	**0.187649**
$Z_{4,4}$ $\mathbf{\lvert Z_{4,4}\rvert}$	$-1{,}213606 - 0{,}441539j$ **1,291431**	$-0{,}341269 + 1{,}157307j$ **1,206575**	$-1{,}214349 - 0{,}443748j$ **1,292887**	$0{,}571063 - 0{,}865562j$ **1,036972**	**0.086317**
$Z_{5,1}$ $\mathbf{\lvert Z_{5,1}\rvert}$	$-27{,}562113 - 10{,}732548j$ **29,577993**	$-3{,}863047 - 30{,}924367j$ **31,164717**	$-27{,}663335 - 11{,}033875j$ **29,782654**	$20{,}974561 + 21{,}989910j$ **30,388952**	**0.020421**
$Z_{5,3}$ $\mathbf{\lvert Z_{5,3}\rvert}$	$11{,}513742 + 6{,}064851j$ **13,013404**	$-6{,}839282 - 7{,}675631j$ **10,280617**	$11{,}366575 + 6{,}076567j$ **12,888898**	$5{,}213081 - 9{,}637214j$ **10,956829**	**0.101085**
$Z_{5,5}$ $\mathbf{\lvert Z_{5,5}\rvert}$	$-0{,}177542 - 0{,}172670j$ **0,247661**	$-0{,}285599 - 0{,}074539j$ **0,295166**	$-0{,}175937 - 0{,}173549j$ **0,247130**	$-0{,}186476 + 0{,}045296j$ **0,191898**	**0.148951**

Model. AIC y BIC son criterios de selección de modelos de penalización probabilística que estiman la cantidad de información perdida cuando los datos se representan con un modelo, en comparación con otros modelos. En otras palabras, AIC y BIC permiten cuantificar el compromiso entre la bondad de ajuste (overfitting) del modelo y la simplicidad del modelo (underfitting). Sin embargo, el BIC penaliza la complejidad con más severidad que el AIC. Por lo tanto, el AIC tiende a elegir modelos más complejos que podrían sobreajustarse, y el BIC tiende a elegir modelos más simples que podrían provocar underfitting. Una buena práctica es considerar ambos criterios al evaluar un modelo, es decir, se elige el modelo con el AIC o BIC más bajo como la mejor manera de agrupar los datos [70].

Al analizar la gráfica de los puntajes de AIC/BIC del modelo de agrupamiento (GMM) tomando en cuenta diferentes números de clúster, se selecciona 256 como el número óptimo de cluster para ajustar el modelo porque tiene un valor bajo en los criterios AIC/BIC, pero no lo

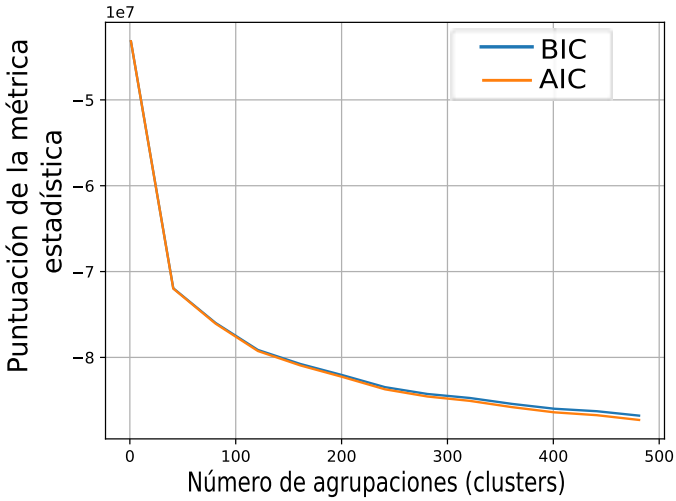


Figura 2.24: Curvas AIC y BIC para agrupar los vectores descriptores basados en Momentos Zernike de los símbolos.

suficiente como para causar un ajuste excesivo en el modelo(overfitting). Es decir, según la figura 2.24, si se escoge un número de clúster superior a 256 se obtiene valores aún más bajos en los criterios AIC/BIC, pero eso no implica que exista una mejora significativa en el rendimiento del modelo de clusterización, además de aumentar el riesgo de provocar un sobreajuste en el modelo.

El modelo GMM fue implementado con la librería **Scikit-learn** de Python a través de la clase **sklearn.mixture.GaussianMixture()** que se muestra en el Código 16 con los siguientes parámetros:

- **n_components:** El número de componentes de la mezcla.
- **covariance_type:** Tipo de parámetros de covarianza a utilizar. Así, con **'full'** cada componente tiene su propia matriz de covarianza general. Con la opción **'tied'** todos los componentes comparten la misma matriz de covarianza general. Si especifica la opción **'diag'** cada componente tiene su propia matriz de covarianza diagonal y con la opción **'spherical'** cada componente tiene su propia varianza única.
- **tol:** Es el umbral de convergencia. Las iteraciones del algoritmo EM se detendrán cuando la ganancia promedio del límite inferior esté por debajo de este umbral.
- **max_iter:** El número de iteraciones para ejecutar el algoritmo EM.
- **init_params:** El método utilizado para inicializar los pesos, las medias y las precisiones de GMM. Así, **'kmeans'** inicializa los hiperparámetros utilizando kmeans, mientras que **'random'** inicializa los hiperámetros aleatoriamente.

```
class sklearn.mixture.GaussianMixture(  n_components=256,
                                        covariance_type='full',
                                        tol=0.001,
                                        max_iter=1000,
                                        init_params='kmeans')
```

Código 16: Inicialización de parámetros para Gaussian Mixture Model con 256 componentes

Codificación de características: Este último paso de la técnica de Bag of Visual Words permite construir un vector de características global a partir de un método de codificación de los vectores descriptores. Para ello se considera la técnica de cuantificación vectorial (VQ), que es un método de codificación basado en votaciones en el que cada descriptor vota directamente por la palabra de código utilizando una estrategia específica. De esta manera, la cuantificación vectorial agrupa los vectores descriptores en su espacio de características constituido por 256 grupos obtenidos desde Gaussian Mixture Model y codifica cada vector descriptor por el índice del grupo al que pertenece. Con este vocabulario de 256 palabras visuales es posible representar cada línea de un bloque de texto como un conjunto de palabras visuales.

2.4.3. ENTRENAMIENTO DEL CLASIFICADOR SUPPORT VECTOR MACHINE (SVM)

Primero, se generaliza el modelo de cluterización (Gaussian Mixture Model) con una base de datos conformado por $487\,913$ símbolos de texto y $484\,257$ símbolos matemáticos con tamaño y forma variables. Estos símbolos se extrajeron de las regiones de texto que no se utilizaron en la etapa de entrenamiento de la Red Neuronal Convolucional. Posteriormente, se calculan los $972\,170$ vectores descriptores y se selecciona el $60\,\%$ de las muestras para ajustar los modelos GMM con distintos números de clusters o componentes y con el $40\,\%$ restante se evaluaron los modelos obtenidos con las métricas AIC / BIC para escoger el modelo con el número óptimo de clusters. De esta manera, se obtuvo que el número óptimo de clusters es 256 y la información estadística de cada uno de los 256 grupos o componentes del GMM son almacenados para que sirvan de referencia para la codificación de características sobre vectores descriptores de futuros símbolos.

Para entrenar el clasificador SVM, consideramos 800 regiones de texto y $2\,800$ regiones de ecuación, lo que resultó en $4\,948$ líneas con símbolos de texto y $5\,108$ líneas con símbolos matemáticos. Los símbolos de estas líneas provienen de regiones diferentes a las utilizadas

para ajustar el GMM para evitar la contaminación de los datos. El vector de características para cada una de estas líneas se construye extrayendo las características geométricas y el vector global de características generado con Bag of Visual Words por medio de Gaussian Mixture Model que ha sido previamente ajustado. Es decir, la etapa de entrenamiento del clasificador SVM usa $4\,948$ vectores de características con etiqueta de texto y $5\,108$ vectores de características con etiqueta de ecuación.

Al igual que la Red Neuronal Convolucional, el clasificador SVM fue entrenado con el enfoque de Validación Cruzada con K iteraciones asignado un valor de $K = 5$. Del conjunto de datos se selecciona de forma aleatoria el 80 % de estas muestras como una base de datos para entrenamiento y validación mientras que el 20 % se usa como base de datos de prueba para medir el rendimiento del clasificador.

Para el entrenamiento se eligió SVM con función de núcleo(kernel) de base radial (RBF) o Gaussiana cuyos hiperparámetros del clasificador se estiman realizando una búsqueda en grilla con valores predefinidos y junto con la validación cruzada de K iteraciones, se toma el mejor modelo con sus respectivos valores de hiperparámetros. Con la técnica de búsqueda con grilla, se pretende optimizar los hiperparámetros C y gamma (γ) ya que estos valores influyen en el proceso de optimización del mejor hiperplano que divide de una manera óptima en clases en función de los datos de entrenamiento.

El parámetro C define la penalización por errores en los datos. Un valor mayor de C aumenta la penalización y reduce el número de puntos que se permiten en el margen de error. Un número menor de C generalmente permite un margen de error mayor cuando el hiperplano está separado. El parámetro gamma (γ) es específico para la SVM gaussiana e influye en la flexibilidad de la hiperlínea. Para valores más pequeños de gamma, la línea que separa el hiperplano es casi lineal, y para números más grandes se vuelve más curva. Aumentar demasiado el valor de gamma puede provocar un ajuste excesivo de los datos de entrenamiento [71].

Para determinar los hiperparámetros óptimos con el conjunto de datos de entrenamiento se construye una grilla o cuadrícula con un rango de valores de $1 * 10^{-5}$ hasta $1 * 10^{5}$ para el parámetro C y un rango de $1 * 10^{-7}$ hasta 1 para el parámetro gamma. Para aumentar la velocidad en el entrenamiento se constituye la cuadrícula en pasos que aumentan en escala logarítmica, ya que para cada elemento de la cuadrícula se ajusta un modelo de SVM y se selecciona mejor aquel que optimiza la función de pérdida de la bisagra (hinge loss) u obtiene los mejores valores en exactitud(Accuracy) y precisión.

En la Figura 2.25 se presentan gráficos en 3D de los entrenamientos realizados sobre la

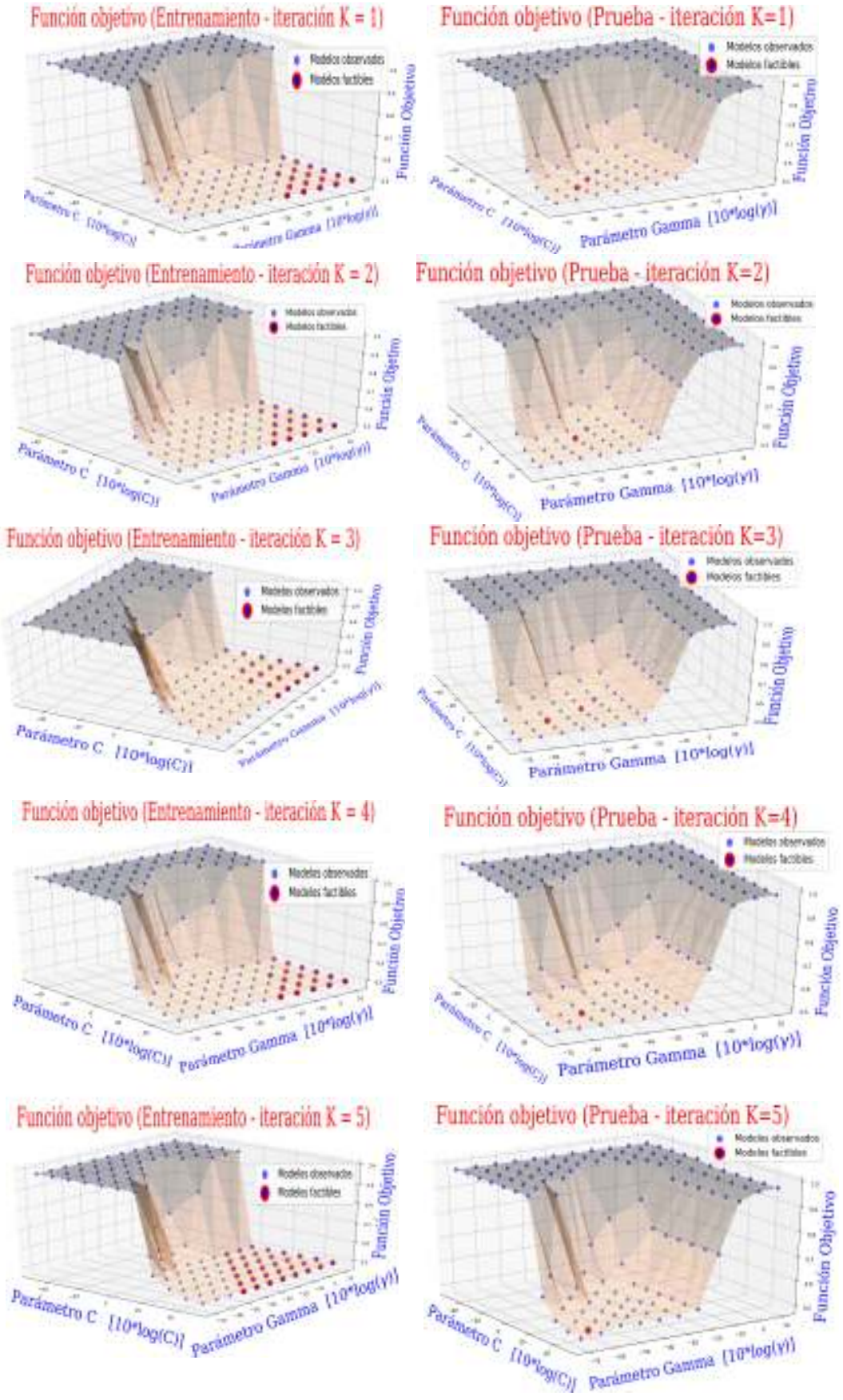


Figura 2.25: Optimización de la función de pérdida de la bisagra (hinge) considerando una grilla

cuadrícula de los parámetros C y gamma para cada uno de las 5 iteraciones de K-Fold Cross Validation para la optimización de la función de pérdida de la bisagra que se explica en la Sección 1.4.5.2.

En la Figura 2.26 se muestra el mejor modelo SVM obtenido de la validación cruzada de K iteraciones con búsqueda de hiperparámetros en cuadrícula. El mejor modelo tiene como hiperparámetros a $C = 1000$ y $\gamma = 3,98 * 10^{-}6$.

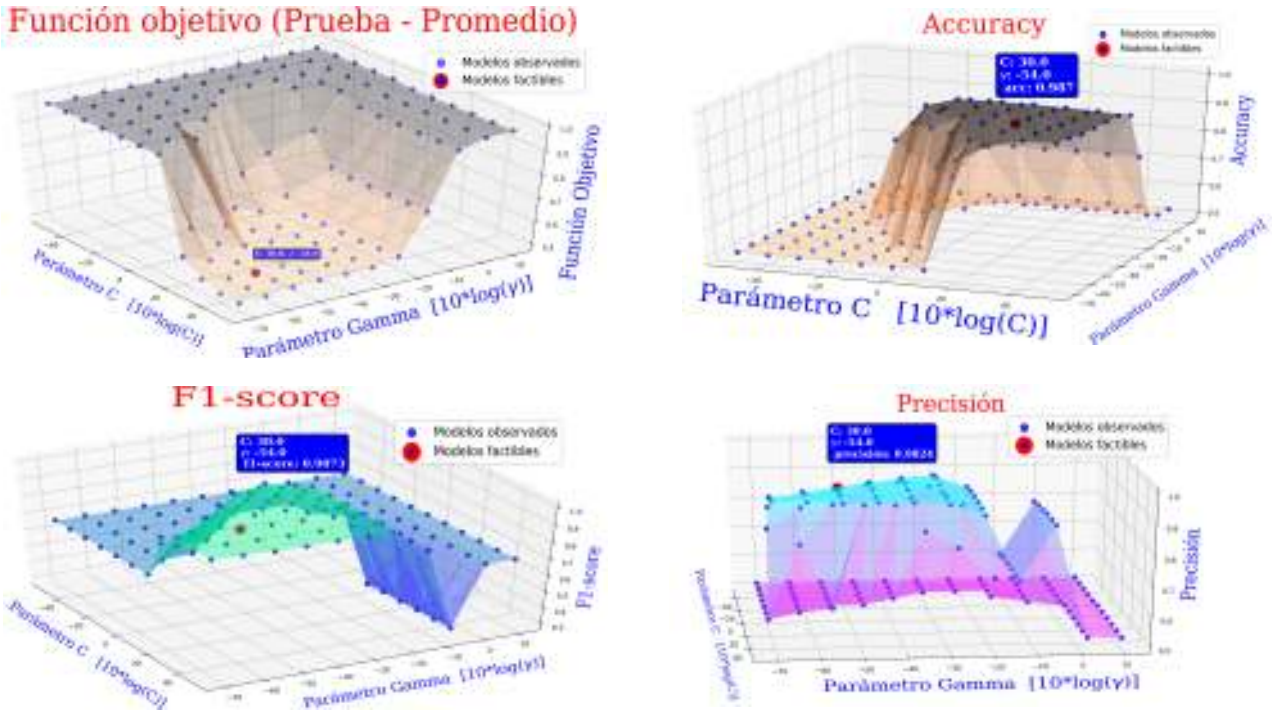


Figura 2.26: Mejor modelo SVM con hiperparámetros $C = 1000$ y $\gamma = 3,98 * 10^{-}6$

3. RESULTADOS Y DISCUSIÓN (APLICACIÓN METODOLÓGICA)

Para la cuantificación del performance de la Red Neuronal Convolucional y del clasificador de líneas de texto y ecuación se emplean las métricas de evaluación de algoritmos de Aprendizaje Automático Supervisado como precisión, sensibilidad(Recall), métrica F1 (F1-score) y se realiza un análisis de las curvas ROC (Receiver Operating Characteristic) y su métrica AUC(Area Under the Curve).

Para interpretar las métricas precisión, sensibilidad y F1-Score se considera los siguientes términos que se originan considerando la evaluación de una clasificación binaria sobre un conjunto de datos de prueba que se etiquetan como positivos o negativos:

- **Verdaderos Positivos (True Positive - TP):** Corresponde a los ejemplos correctamente etiquetados como positivos.
- **Falsos Positivos (False Positive - FP):** Se refiere a ejemplos negativos etiquetados incorrectamente como positivos.
- **Verdaderos Negativos (True Negative - TN):** Corresponden a ejemplos negativos correctamente etiquetadas como negativos.
- **Falsos Negativos (False Negative - FN):** Corresponden a ejemplos positivos etiquetados incorrectamente como negativos.

Los términos anteriores se pueden resumir en una tabla de contigencia o matriz de confusión como la Figura 3.1.

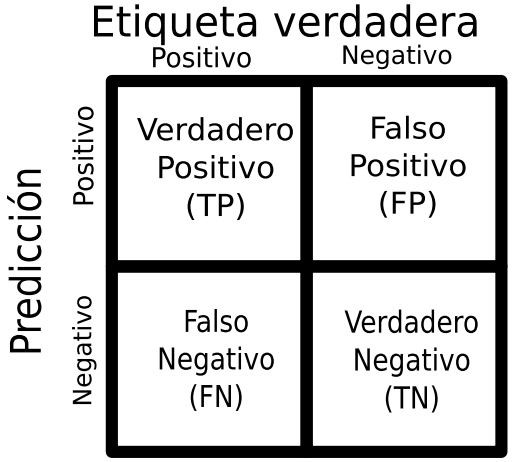


Figura 3.1: Estructura de una matriz de confusión para una clasificación binaria [72]

Precisión: La precisión es intuitivamente la capacidad del clasificador de no etiquetar como positiva una muestra que es negativa [72]. La precisión se determina con la ecuación 3.1.

$$\text{Precisión}(p) = \frac{TP}{TP + FP} \tag{3.1}$$

Sensibilidad (Recall): cuantifica la capacidad del clasificador de encontrar todas las muestras positivas, es decir, mide la fracción de muestras positivos que se clasifican correctamente [72]. La sensibilidad se determina con la ecuación 3.2.

$$\text{Sensibilidad}(r) = \frac{TP}{TP + FN} \tag{3.2}$$

Métrica F1 (F1-score): Representa la media armónica entre los valores de sensibilidad y precisión. F1 alcanza su mejor valor en 1 y la peor puntuación en 0 [72]. Esta métrica se calcula con la ecuación 3.3.

$$\text{F1\_Score} = \frac{2pr}{p + r} \tag{3.3}$$

Exactitud (Accuracy): Rsta métrica cuantifica la proporción de predicciones correctas sobre el número total de instancias evaluadas [72]. La exactitud se determina con la ecuación 3.4.

$$\text{Exactitud}(acc) = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.4}$$

La métricas anteriormente descritas pueden ser extendidas a la clasificación multiclase, al binarizar las salidas, es decir, las métricas se calculan comparando cada clase frente a las demás.

Receiver Operating Characteristic (ROC): es una técnica para visualizar, organizar y seleccionar clasificadores en función de su rendimiento. Los gráficos ROC permiten representar el equilibrio entre las tasas de aciertos y las tasas de falsas alarmas de los clasificadores. Los gráficos ROC son gráficos bidimensionales en los que la tasa de verdaderos positivos (TPR) se traza en función de la tasa de falsos positivos (FPR). Un gráfico ROC muestra las compensaciones relativas entre beneficios (verdaderos positivos) y costos (falsos positivos) [73].

Un clasificador discreto es aquel que genera solo una etiqueta de clase. Cada clasificador discreto produce un par (TPR, FPR) correspondiente a un solo punto en el espacio ROC. Los clasificadores de la Figura 3.2 son todos clasificadores discretos.

Es importante tener en cuenta varios puntos en el espacio de la curva ROC. El punto inferior

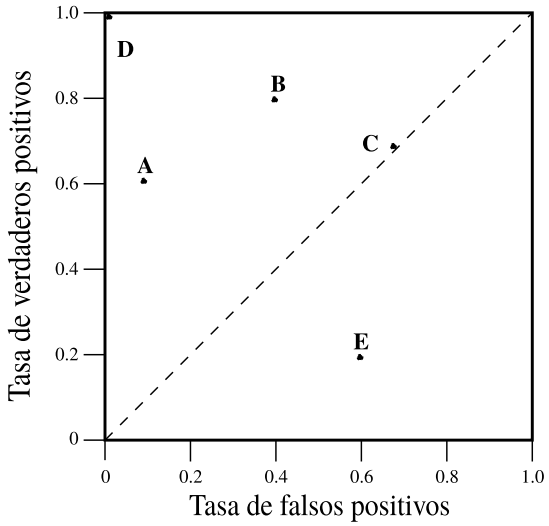


Figura 3.2: Un gráfico ROC básico que muestra cinco clasificadores discretos [73]

izquierdo $(0, 0)$ representa la estrategia de nunca emitir una clasificación positiva; tal clasificador no comete errores de falsos positivos, pero tampoco obtiene verdaderos positivos. La estrategia opuesta, de emitir clasificaciones positivas incondicionalmente, está representada por el punto superior derecho $(1, 1)$. El punto $(0, 1)$ representa una clasificación perfecta, es decir el rendimiento del clasificador D es perfecto como se muestra en la Figura 3.2.

De manera informal, un punto en el espacio de la ROC es mejor que otro si está al noroeste (TPR es mayor, la FPR es menor, o ambas) del primero. Los clasificadores que aparecen en el lado izquierdo de un gráfico ROC, cerca del eje X, pueden considerarse ”conservadores”, ya que realizan clasificaciones positivas solo con evidencia sólida, por lo que cometen pocos errores de falsos positivos, pero a menudo tienen bajas tasas de verdaderos positivos (TPR) también. Los clasificadores en el lado superior derecho de un gráfico ROC pueden considerarse ”liberales”, debido a que efectúan clasificaciones positivas con evidencia débil, por lo que clasifican casi todos los positivos correctamente, pero a menudo tienen altas tasas de falsos positivos. En la Figura 3.2, A es más conservador que B. Muchos dominios del mundo real están dominados por un gran número de instancias negativas, por lo que el rendimiento en el extremo izquierdo del gráfico ROC se vuelve más interesante [73].

La línea diagonal $y = x$ representa la estrategia de adivinar una clase al azar. Por ejemplo, si un clasificador adivina aleatoriamente la clase positiva la mitad del tiempo, se puede esperar que obtenga la mitad de los positivos y la mitad de los negativos correctos; esto produce el punto $(0{,}5; 0{,}5)$ en el espacio ROC. Si adivina la clase positiva el $90\,\%$ del tiempo, se puede esperar que el $90\,\%$ de los positivos sean correctos, pero su tasa de falsos positivos también aumentará al $90\,\%$, dando $(0{,}9; 0{,}9)$ en el espacio ROC. Por lo tanto, un clasificador aleatorio

producirá un punto ROC que se desliza hacia adelante y hacia atrás en la diagonal en función de la frecuencia con la que adivina la clase positiva. Para alejarse de esta diagonal hacia la región triangular superior, el clasificador debe aprovechar cierta información de los datos. En la Figura 3.2, el desempeño de C es virtualmente aleatorio. En $(0{,}7; 0{,}7)$, se puede decir que C está adivinando la clase positiva el 70 % del tiempo [73].

Cualquier clasificador que aparezca en el triángulo rectángulo inferior funciona peor que la adivinación aleatoria. Por lo tanto, este triángulo suele estar vacío en los gráficos ROC. Si negamos un clasificador, es decir, si se revierte sus decisiones de clasificación en cada instancia, sus clasificaciones de verdaderos positivos se convierten en falsos negativos y sus falsos positivos en verdaderos negativos. Por lo tanto, cualquier clasificador que produzca un punto en el triángulo inferior derecho se puede negar para producir un punto en el triángulo superior izquierdo. En la Figura 3.2, E se comporta mucho peor que el aleatorio, y de hecho es la negación de B. Se puede decir que cualquier clasificador en la diagonal no tiene información sobre la clase. Se puede decir que un clasificador debajo de la diagonal tiene información útil, pero está aplicando la información incorrectamente [73].

3.1. DESEMPEÑO DE LA RED NEURONAL CONVOLUCIONAL

La evaluación del rendimiento de la Red Neuronal Convolucional con validación cruzada de K iteraciones se resume a través de la matriz de confusión de la Figura 3.3 y los valores obtenidos en las métricas de precisión, sensibilidad y F1-Score que se detallan en la Tabla 3.1 considerando las tres categorías: texto, tabla y figura. De forma general, la exactitud (accuracy) de la Red Neuronal Convolucional obtenida con la arquitectura detallada en la Sección 2.3 alcanzó el $97,401$ %.

Tabla 3.1: Métricas de evaluación obtenidas en la Red Neuronal Convolucional con validación cruzada de K iteraciones con $K = 5$.

Región	Precisión	Sensibilidad	F1-Score
Texto	0.98500	0.99218	0.98858
Figura	0.96890	0.96108	0.96497
Tabla	0.96735	0.96799	0.96767

Entre los aspectos importantes que influyen en el desempeño de la Red Neuronal Convolucional se encuentra la generación de espectrogramas, ya que transforma la información fundamental contenida en las formas de onda de las proyecciones de los perfiles en un plano. Los espectrogramas se obtienen considerando una ventana de Blackman-Harris de longitud 126 a señales de dimensión $2\,016$, por lo que tenemos espectrogramas de banda an-

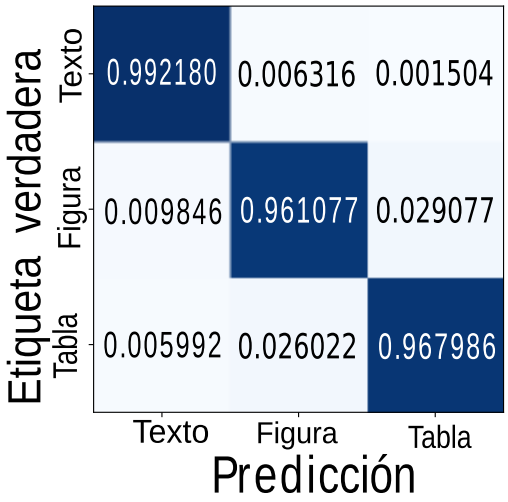


Figura 3.3: Matriz de confusión normalizado para la Red Neuronal Convolucional.

cha, por lo que el entrenamiento se realizó con espectrogramas que tienen mejor resolución en el eje horizontal (muestras) que en el eje vertical (frecuencia).

Otro punto importante en los resultados obtenidos es el diseño de la arquitectura de la Red Neuronal Convolucional, ya que al utilizar bloques convolucionales conformados por capas de convolución, normalización por lotes, leaky RELU y agrupación en las dos rama de la arquitectura se logra reducir los espectrogramas de las proyecciones vertical y horizontal de cada región a un mapa de características de tamaño $128 \times 4 \times 4$ que resumen patrones visuales informativos locales con propiedades de invarianza de traslación, que pueden ser aprovechados por la capas densamente conectadas para el proceso de clasificación.

En la matriz de confusión, se observa un alto desempeño en la clasificación de regiones de texto en comparación con las otras dos categorías, esto significa que al tomar los espectrogramas como característica de regiones reduce el rendimiento de clasificación para tablas y figuras porque algunas regiones de tablas presentan proyecciones de perfiles similares a los de una figura, causando que los espectrogramas sean también similares para estas dos regiones.

Dado que el análisis de la inclinación de las curvas ROC es importante, para maximizar la tasa de verdaderos positivos y minimizar la tasa de falsos positivos, se traza también la curva ROC con los resultados de predicción obtenidos con la técnica de validación cruzada de K iteraciones para la Red Neuronal Convolucional.

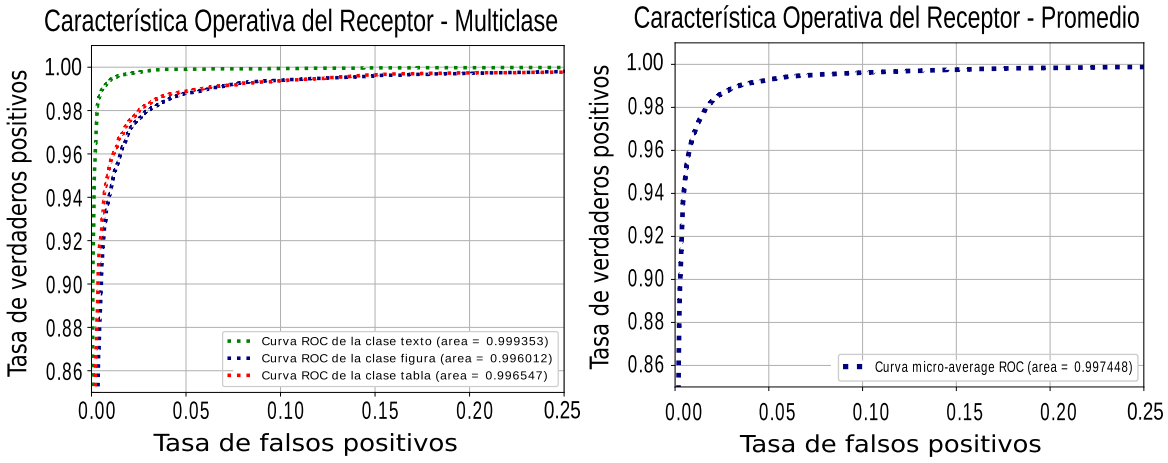
Debido a que las curvas ROC se utilizan normalmente en la clasificación binaria para estudiar la salida de un clasificador, también se puede extender la curva ROC a la clasificación de múltiples clases, para lo cual es necesario binarizar la salida del clasificador, es decir, se puede dibujar una curva ROC por etiqueta. Así en la Figura 3.4a se observa las curvas ROC

multiclase para cada una de las tres clases (texto, tabla y figura). Pero también se puede dibujar una curva ROC considerando cada elemento de la matriz del indicador de etiqueta como una predicción binaria. De esta manera, en la Figura 3.4b se ilustra la curva ROC micro-avarage para la Red Neuronal Convolucional.

Analizando la Figura 3.4 se puede mencionar que la Red Neuronal Convolucional tiene una mayor capacidad para diferenciar la clase de texto en comparación con las clases de figura y tabla, lo que se puede verificar al comparar los valores del Área bajo la Curva ROC (AUC) para cada clase (Ver Tabla 3.2). La métrica AUC tiene una propiedad estadística importante, el AUC de un clasificador es equivalente a la probabilidad de que el clasificador clasifique una instancia positiva elegida aleatoriamente por encima de una instancia negativa elegida aleatoriamente.

Tabla 3.2: Área bajo la curva ROC (AUC) para la Red Neuronal Convolucional

Región	AUC
Texto	0.999353
Figura	0.996012
Tabla	0.996547
Micro-Avarage	0.997448



(a) ROC Multiclase **(b)** Micro Avarage ROC

Figura 3.4: Performance de la Red Neuronal Convolucional

3.2. DESEMPEÑO DEL CLASIFICADOR SUPPORT VECTOR MACHINE

El desempeño de la etapa de clasificación binaria de líneas de texto y ecuaciones se refleja en la evaluación del clasificador SVM mediante la técnica de validación cruzada de K iteraciones, que se resume en la matriz de confusión en la Figura 3.5. El desempeño obtenido con el clasificador SVM se debe a la contribución de dos aspectos importantes:

- Las características con las que se entrenó el modelo provenientes de la técnica de Bag of Visual Words(BoVW) con momentos de Zernike y de la extracción de características geométricas.Con BoVW se construye un vocabulario de 256 palabras visuales robusto agrupando(clustering) los momentos de Zernike de los símbolos debidos a las propiedad de ortogonalidad en los polinomios y la invarianza de rotación, traslación y escalamientos en la magnitud de los momentos.

- La estimación de los valores de los hiperparámetros de SVM se realiza mediante búsqueda de cuadrícula con validación cruzada de $K = 5$ iteraciones. Por tanto, cada uno de las iteraciones se entrena con diferentes valores de hiperparámetros, de los cuales se selecciona el mejor modelo con el proceso de validación.

- Utilizar SVM con una función de kernel gausiano, permite solucionar el problema de la clasificación con una separación en un espacio de características de una alta dimensión.

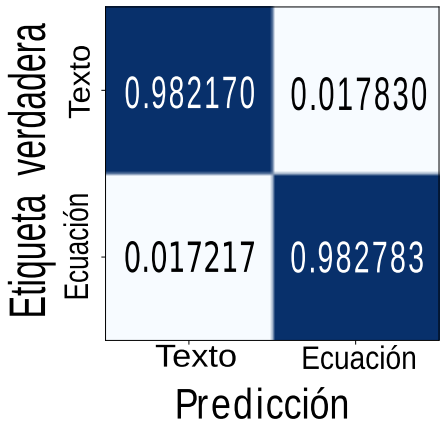


Figura 3.5: Matriz de confusión normalizada para la clasificación de líneas de ecuaciones.

La tabla 3.3 resume los valores de las métricas de Precisión, Sensibilidad y F1-Score para cada una de las dos categorías del clasificador SVM, obteniendo una exactitud (accuracy) de $98{,}249\,\%$ en la clasificación de líneas de texto y ecuación.

Tabla 3.3: Métricas de evaluación obtenidos con el clasificador SVM.

Región	Precisión	Sensibilidad	F1-Score
Texto	0.98171	0.98217	0.98194
Ecuación	0.98321	0.98278	0.98300

También en la Figura 3.6 se traza la curva ROC del clasificador SVM. Esta curva ROC se acerca a la esquina superior izquierda del gráfico, lo que significa que el clasificador logra una separación notable entre los dos grupos de clasificación (líneas de texto y líneas de ecuación) y se puede comprobar con el Área bajo la curva ROC (AUC) que alcanza un valor de $0{,}982477$. Por tanto, el clasificador SVM tiene valores elevados de especificidad y sensibilidad, lo que explica la obtención de una alta exactitud del modelo.

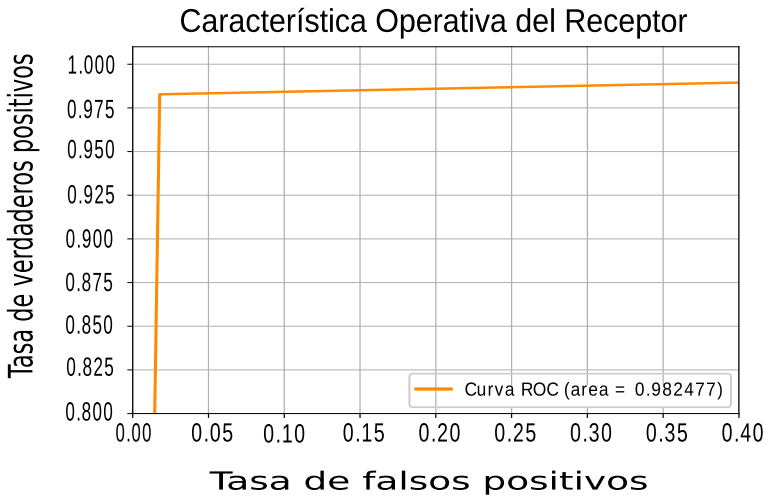


Figura 3.6: Curva ROC para clasificación de líneas de ecuaciones.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- El algoritmo propuesto en este proyecto tiene el objetivo principal de clasificar regiones de texto, figura, tabla y ecuaciones en páginas de documentos científicos digitales, que han sido previamente segmentados considerando la extracción de características directamente de la información de los píxeles de una imagen. Este enfoque permite la clasificación de regiones sin depender de la estructura con la que ha sido originado el documento científico, ya que cada página del documento necesita ser transformado al formato de imagen PNG en escala de color RGB, para ser procesado por el sistema de clasificación automática.

- La ventaja que tiene la arquitectura de la Red Neuronal Convolucional al utilizar espectrogramas de 64×64 píxeles en lugar de la región completa es la reducción del costo computacional en la fase de entrenamiento y validación ya que se tiene una rápida convergencia del gradiente descendente debido al empleo de la técnica Parada Temprana (Early Stopping) y a la configuración cíclica de la tasa de aprendizaje (Learning rate scheduler), lo que se verifica a través de las curvas de aprendizaje (Accuracy y optimización de la función de pérdida). Esto hace que el sistema de clasificación sea viable para el desarrollo de aplicaciones que requieren consumir menor cantidad de recursos de memoria.

- Una de las debilidades de nuestro algoritmo es la dependencia que existe entre la etapa de la Red Neuronal Convolucional y el segmentador de páginas en regiones. Dado que este proyecto no se enfoca en la segmentación de regiones para mantener el desempeño de la clasificación regional (texto, figura y tabla), el segmentador que se acople a la Red Neuronal Convolucional debe extraer con alta fiabilidad las regiones de interés de cada una de las páginas de un documento.

- La etapa de clasificación de líneas de ecuaciones se diseña con la técnica Bag of Visual Words utilizando momentos Zernike como extractor de características.De esta manera cabe señalar que la construcción del vocabulario visual se entrenó con símbolos del alfabeto inglés y con símbolos matemáticos, lo que significa que este algoritmo se puede utilizar tanto para el idioma español como para el inglés, pero puede ser adaptado a otro idioma ajustando el algoritmo de clusterización con el nuevo alfabeto.

4.2. RECOMENDACIONES

- Dado que la segmentación de páginas en regiones de interés influyen en el rendimiento de la clasificación de regiones con la Red Neuronal Convolucional se recomienda considerar un segmentador de regiones más preciso y exacto.

- Para mejorar el rendimiento de la Red Neuronal Convolucional en la clasificación de regiones de ”tablas” y ”figuras” se recomiendo considerar la información sobre la textura y los colores presentes en las regiones de figura y tabla ya que la proyección de perfiles son semejantes en estos dos tipos de regiones.

- Aunque hay un alto rendimiento en la clasificación de líneas de ecuaciones y líneas de texto, no se aprovecha al máximo la información que generan los momentos complejos de Zernike, ya que este trabajo solo se ha tomado en cuenta la magnitud de los momentos y se ha ignorado la fase. Este aspecto puede ser aprovechado en estudios futuros para diseñar algoritmos más robusto en la solución de detección de ecuaciones en regiones de texto.

- Finalmente, el enfoque de clasificación de líneas de ecuaciones propuesto en este trabajo no se ocupa de ecuaciones o símbolos matemáticos embebidas en líneas de texto, se recomienda que en trabajos futuros se considere el desarrollo de un reconocimiento óptico de caracteres matemático (OCR) más sofisticado aprovechando las propiedades de los momentos Zernike, tanto en fase como en magnitud, que permiten identificar la presencia de símbolos matemáticos incrustados en regiones de ”texto”.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] O. A. Ise, “Integration and analysis of unstructured data for decision making: Text analytics approach,” *International Journal of Open Information Technologies*, vol. 4, no. 10, 2016.

[2] A. M. Namboodiri and A. K. Jain, “Document structure and layout analysis,” in *Digital Document Processing*. Springer, 2007, pp. 29–48.

[3] G. M. Binmakhashen and S. A. Mahmoud, “Document layout analysis: A comprehensive survey,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–36, 2019.

[4] P. W. J. Staar, M. Dolfi, C. Auer, and C. Bekas, “Corpus conversion service: A machine learning platform to ingest documents at scale,” *CoRR*, vol. abs/1806.02284, 2018. [Online]. Available: http://arxiv.org/abs/1806.02284

[5] X. Zhong, J. Tang, and A. J. Yepes, “Publaynet: largest dataset ever for document layout analysis,” in *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2019, pp. 1015–1022.

[6] Ł. Garncarek, R. Powalski, T. Stanisławek, B. Topolski, P. Halama, and F. Graliński, “Lambert: Layout-aware language modeling using bert for information extraction,” *arXiv preprint arXiv:2002.08087*, 2020.

[7] C. Clausner, A. Antonacopoulos, and S. Pletschacher, “Icdar2017 competition on recognition of documents with complex layouts-rdcl2017,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 1404–1410.

[8] R. Kasturi, L. O’gorman, and V. Govindaraju, “Document image analysis: A primer,” *Sadhana*, vol. 27, no. 1, pp. 3–22, 2002.

[9] R. C. Gonzalez and R. E. Woods, “Digital image processing,” *ed: Prentice Hall Press, ISBN 0-201-18075*, vol. 8, 2018.

[10] P. Stathis, E. Kavallieratou, and N. Papamarkos, “An evaluation technique for binarization algorithms.” *J. UCS*, vol. 14, no. 18, pp. 3011–3030, 2008.

[11] J. Seo, S. Chae, J. Shim, D. Kim, C. Cheong, and T.-D. Han, “Fast contour-tracing algorithm based on a pixel-following method for image sensors,” *Sensors*, vol. 16, no. 3, p. 353, 2016.

[12] E. Sejdić, I. Djurović, and J. Jiang, “Time–frequency feature representation using energy concentration: An overview of recent advances,” *Digital Signal Processing*, vol. 19, no. 1, pp. 153 – 183, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S105120040800002X

[13] N. Kehtarnavaz, “Chapter 7 - frequency domain processing,” in *Digital Signal Processing System Design (Second Edition)*, second edition ed., N. Kehtarnavaz, Ed. Burlington: Academic Press, 2008, pp. 175 – 196. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123744906000076

[14] D. Y. Isukapalli, “Filter banks, time-frequency representations,” 2007. [Online]. Available: https://web.ece.ucsb.edu/~yoga/courses/Adapt/P1%20Time-Frequency%20representation.pdf

[15] P. Schniter, “Ece-700 time-frequency/wavelet notes,” 2007. [Online]. Available: https://pdfs.semanticscholar.org/0cca/bb4fa85a191d47e3d00910a24c728082d9f5.pdf

[16] S. H. Nawab and T. F. Quatieri, *Short-Time Fourier Transform*. USA: Prentice-Hall, Inc., 1987, p. 289–337.

[17] M. Müller, *Fundamentals of music processing: Audio, analysis, algorithms, applications*. Springer, 2015.

[18] A. Khotanzad and Y. H. Hong, “Invariant image recognition by zernike moments,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 5, pp. 489–497, 1990.

[19] A. Górniak and E. Skubalska-Rafajłowicz, “Object classification using sequences of zernike moments,” in *IFIP International Conference on Computer Information Systems and Industrial Management*. Springer, 2017, pp. 99–109.

[20] S.-K. Hwang and W.-Y. Kim, “A novel approach to the fast computation of zernike moments,” *Pattern Recognition*, vol. 39, no. 11, pp. 2065–2076, 2006.

[21] C.-W. Chong, P. Raveendran, and R. Mukundan, “Translation invariants of zernike moments,” *Pattern recognition*, vol. 36, no. 8, pp. 1765–1773, 2003.

[22] W.-Y. Kim and Y.-S. Kim, “A region-based shape descriptor using zernike moments,” *Signal Processing: Image Communication*, vol. 16, no. 1, pp. 95 – 102, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0923596500000199

[23] G. Abandah and N. Anssari, “Novel moment features extraction for recognizing handwritten arabic letters,” *Journal of Computer Science*, vol. 5, no. 3, p. 226, 2009.

[24] V. Christlein, D. Bernecker, and E. Angelopoulou, “Writer identification using vlad encoded contour-zernike moments,” in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 906–910.

[25] S. Farokhi, U. U. Sheikh, J. Flusser, and B. Yang, “Near infrared face recognition using zernike moments and hermite kernels,” *Information Sciences*, vol. 316, pp. 234 – 245, 2015, nature-Inspired Algorithms for Large Scale Global Optimization. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515002984

[26] A. Tahmasbi, F. Saki, and S. B. Shokouhi, “Classification of benign and malignant masses based on zernike moments,” *Computers in Biology and Medicine*, vol. 41, no. 8, pp. 726 – 735, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010482511001296

[27] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[28] X. Du, Y. Cai, S. Wang, and L. Zhang, “Overview of deep learning,” in *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. IEEE, 2016, pp. 159–164.

[29] C. C. Aggarwal *et al.*, *Neural networks and deep learning*. Springer, 2018.

[30] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture of monkey striate cortex,” *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[31] K. Fukushima and S. Miyake, “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,” in *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[32] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[33] A. Hertzmann and D. Fleet, “Machine learning and data mining lecture notes,” *Computer Science Department, University of Toronto*, 2010.

[34] A. López Díaz, “Fundamentos matemáticos de los métodos kernel para aprendizaje supervisado,” 2018.

[35] N. A. Alqahtani and Z. I. Kalantan, “Gaussian mixture models based on principal components and applications,” *Mathematical Problems in Engineering*, vol. 2020, 2020.

[36] B. Erar, “Mixture model cluster analysis under different covariance structures using information complexity,” 2011.

[37] S. Albelwi and A. Mahmood, “A Framework for Designing the Architectures of Deep Convolutional Neural Networks,” *Entropy*, vol. 19, no. 6, p. 242, may 2017. [Online]. Available: http://www.mdpi.com/1099-4300/19/6/242

[38] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, 2018. [Online]. Available: https://doi.org/10.1007/s13244-018-0639-9

[39] I. Goodfellow, Y. Bengio, and A. Courville, “Deep learning,” 2016. [Online]. Available: http://www.deeplearningbook.org

[40] B. Ding, H. Qian, and J. Zhou, “Activation functions and their characteristics in deep neural networks,” in *2018 Chinese Control And Decision Conference (CCDC)*. IEEE, 2018, pp. 1836–1841.

[41] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[42] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, “Recent advances in convolutional neural networks,” *Pattern Recognition*, vol. 77, pp. 354–377, 2018.

[43] A. Ferreira and G. Giraldi, “Convolutional neural network approaches to granite tiles classification,” *Expert Systems with Applications*, vol. 84, pp. 1 – 11, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417417303032

[44] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015. [Online]. Available: https://arxiv.org/abs/1502.03167

[45] Y. Wu and K. He, “Group normalization,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19. [Online]. Available: https://arxiv.org/abs/1803.08494

[46] S. O’Hara and B. A. Draper, “Introduction to the bag of features paradigm for image classification and retrieval,” *arXiv preprint arXiv:1101.3354*, 2011.

[47] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467*, 2016.

[48] F. Chollet, *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[49] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, “Array programming with numpy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[50] W. McKinney *et al.*, “pandas: a foundational python library for data analysis and statistics,” *Python for High Performance and Scientific Computing*, vol. 14, no. 9, pp. 1–9, 2011.

[51] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. .ᴼ’Reilly Media, Inc.”, 2008.

[52] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, “Scipy 1.0: fundamental algorithms for scientific computing in python,” *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[54] S. Tosi, *Matplotlib for Python developers*. Packt Publishing Ltd, 2009.

[55] L. P. Coelho, “Mahotas: Open source software for scriptable computer vision,” *arXiv preprint arXiv:1211.4907*, 2012.

[56] T. Carneiro, R. V. M. Da Nóbrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. Reboucas Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61 677–61 685, 2018.

[57] J. M. Vuletich, "Orthonormal bases and tilings of the time-frequency plane for music processing," in *Wavelets: Applications in Signal and Image Processing X*, vol. 5207. International Society for Optics and Photonics, 2003, pp. 784–793.

[58] X. Lian and J. Liu, "Revisit batch normalization: New understanding from an optimization view and a refinement via composition optimization," *arXiv preprint arXiv:1810.06177*, 2018. [Online]. Available: https://arxiv.org/abs/1810.06177

[59] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *arXiv preprint arXiv:1811.12808*, 2018.

[60] J. D. Rodriguez, A. Perez, and J. A. Lozano, "Sensitivity analysis of k-fold cross validation in prediction error estimation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 569–575, 2009.

[61] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[62] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.

[63] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[64] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[65] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[66] L. Prechelt, *Early Stopping - But When?* Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 55–69. [Online]. Available: https://doi.org/10.1007/3-540-49430-8_3

[67] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[68] W.-T. Chu and F. Liu, “Mathematical formula detection in heterogeneous document images,” in *2013 conference on technologies and applications of artificial intelligence*. IEEE, 2013, pp. 140–145.

[69] X. Peng, L. Wang, X. Wang, and Y. Qiao, “Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice,” *Computer Vision and Image Understanding*, vol. 150, pp. 109–125, 2016.

[70] A. Chakrabarti and J. K. Ghosh, “Aic, bic and recent advances in model selection,” in *Philosophy of Statistics*. Elsevier, 2011, pp. 583–605.

[71] P. Lameski, E. Zdravevski, R. Mingov, and A. Kulakov, “Svm parameter tuning with grid search and its impact on reduction of model over-fitting,” in *Rough sets, fuzzy sets, data mining, and granular computing*. Springer, 2015, pp. 464–474.

[72] M. Hossin and M. Sulaiman, “A review on evaluation metrics for data classification evaluations,” *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, p. 1, 2015.

[73] T. Fawcett, “An introduction to roc analysis,” *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

ANEXOS

Dado que los algoritmos de aprendizaje automático fueron entrenados en la plataforma Google Colab, se adjunta como anexo digital los scripts en extensión ”.ipynb” para cada entrenamiento y además los mejores modelos encontrados para cada etapa del sistema de clasificación en extensión ”.jolib” y ”.h5”.

ANEXO A. EntrenamientoCNN.ipynb

ANEXO B. ClusteringZernike.ipynb

ANEXO C. EntrenamientoSVM.ipynb

ANEXO D. Modelo entrenado red neuronal convolucional: BestModelCNN.h5

ANEXO E. Modelo entrenado GMM(Clustering): GMM_256_cluster.joblib

ANEXO F. Modelo entrenado SVM: BestModelSVM.jolib

ORDEN DE EMPASTADO