



UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

Aceleración Hardware de un Algoritmo de Aprendizaje Activo para Clasificadores de Imágenes

Angelo Roberto Barbieri Figueroa

CONCEPCIÓN - CHILE

AGOSTO, 2024

Resumen

Este trabajo propone acelerar mediante hardware un algoritmo de aprendizaje activo aplicado en clasificadores de imágenes. En este sentido, se realiza una revisión de los fundamentos teóricos y trabajos realizados en clasificadores de imágenes basados en redes neuronales profundas (DNNs), que permiten la clasificación precisa de datos visuales; aprendizaje activo, que optimiza el proceso de etiquetado al seleccionar las muestras más informativas; y aceleración hardware con FPGAs, que ofrece ventajas en términos de eficiencia energética y paralelización de cálculos. Además, se realiza un análisis exploratorio de conjuntos de imágenes sobre los cuales se trabajará, estableciendo una base para la implementación y evaluación de los algoritmos.

Índice general

Resumen	3
1. Introducción	1
2. Estado del arte y motivaciones	3
2.1. Visión por computador	3
2.2. Visión por computador en clasificación	4
2.2.1. ¿Qué es una imagen? (Entrada de datos)	5
2.2.2. Preprocesamiento	6
2.2.3. Extracción de características (enfoque tradicional)	7
2.2.4. Clasificación (enfoque tradicional)	8
2.2.5. Redes neuronales convolucionales	9
2.2.5.1. Red neuronal	9
2.2.5.2. Arquitectura de red neuronal convolucional (CNN)	11
2.2.5.3. Arquitecturas Clásicas	15
2.2.6. Transformadores visuales (ViT)	22
2.2.6.1. Arquitectura de transformador visual	22
2.3. Aprendizaje activo (AL)	24
2.3.1. ¿Qué es el aprendizaje activo?	25
2.3.2. Escenarios de AL	26
2.3.3. Estrategia de consulta	26
2.4. Acelerador Hardware	27
2.4.1. ¿Qué es una FPGA?	27
3. Materiales y métodos	30
3.1. Conjuntos de datos y preprocesamiento	30
3.1.1. WM-811K	30
3.1.2. Brain Tumor Classification (MRI)	32
4. Conclusiones y trabajo futuro	35
Referencias	36

Índice de figuras

2.2.1.Pipeline de visión por computador. Fuente: Adaptado de [1].	4
2.2.2.Comparación de Imagen en Escala de Grises y en Color. Fuente: Adaptado de [1].	6
2.2.3.Preprocesamiento para quitar ruido de una imagen. Fuente: Elaboración propia.	7
2.2.4.Red neuronal convolucional. Fuente: Adaptado de [1].	9
2.2.5.Neurona artificial. Fuente: Adaptado de [1]	10
2.2.6.Red perceptrón multicapa. Fuente: Adaptado de [1]	11
2.2.7.Funciones de activación. Fuente: Adaptado de [2].	13
2.2.8.Dropout. Fuente: Adaptación de [1].	14
2.2.9.Arquitectura de LeNet. Fuente: Adaptado de [1, 3].	15
2.2.10Arquitectura AlexNet. Fuente: Adaptado de [1, 4].	16
2.2.11Arquitectura VGGNet16. Fuente: Adaptado de [1, 5].	18
2.2.12Módulo Inception. Fuente: Adaptado de [1, 6].	19
2.2.13Arquitectura GoogLeNet. Fuente: Adaptado de [1, 6].	20
2.2.14Módulo ResNet (conexión residual). Fuente: Adaptado de [1, 7]. .	21
2.2.15Arquitectura ResNet-50, Fuente: Adaptado de [1, 7].	22
2.2.16Arquitectura de transformadores visuales. Fuente: Adaptado de [8].	23
2.3.1.Pasos de un algoritmo de AL con escenario <i>pool-based</i> . Fuente: Adaptado de [9, 10].	25
2.4.1.Arquitectura de una FPGA. Fuente: Adaptado de [11, 12]	28
3.1.1.Distribución de clases de WM-811K. Fuente: Elaboración propia. .	31
3.1.2.Clases de WM-811K. Fuente: Elaboración propia.	32
3.1.3.Distribución de clases de Brain Tumor. Fuente: Elaboración propia.	33
3.1.4.Procedimiento para recortar una imagen. Fuente: Elaboración propia.	34
3.1.5.Clases de Brain Tumor. Fuente: Elaboración propia.	34

Capítulo 1

Introducción

La clasificación de imágenes permite asignar etiquetas predefinidas a las imágenes según su contenido [1]. Entre sus aplicaciones se encuentran la medicina, donde se utiliza para el diagnóstico de enfermedades a partir de imágenes médicas; los automóviles autónomos, donde ayuda a identificar señales de tráfico y peatones; la agricultura, al detectar plagas y enfermedades en cultivos; la seguridad y vigilancia, mediante la detección de actividades sospechosas en tiempo real; y la inspección de productos, al identificar defectos en líneas de producción, entre otras. En los últimos años, con el desarrollo de la inteligencia artificial (IA), las redes neuronales profundas (DNNs) se han posicionado como referentes no solo en tareas de clasificación, sino también en otras tareas de visión por computador como la detección de objetos, segmentación de imágenes, reconocimiento de rostros, superresolución de imágenes, generación de imágenes, reconocimiento de texto en imágenes (OCR), análisis de movimiento y seguimiento de objetos, entre otras [1, 3, 4, 5, 6, 7, 2].

En el entrenamiento de clasificadores de imágenes basados en redes neuronales profundas (DNNs), es crucial contar con una cantidad considerable de imágenes etiquetadas. Aunque obtener imágenes puede no ser una tarea difícil en ciertas áreas, el proceso de etiquetado suele ser costoso en términos de tiempo y recursos. Esto se debe a la necesidad de contar con anotadores especializados, como médicos que identifican tumores cerebrales en imágenes de resonancia magnética, y/o a una extensa serie de pasos experimentales que a menudo deben llevarse a cabo antes de realizar el etiquetado, como en pruebas de laboratorio [9]. Para reducir el

esfuerzo y los costos asociados al etiquetado de datos, el aprendizaje activo (AL) sugiere que el modelo identifique y seleccione progresivamente las imágenes más informativas de un conjunto de datos no etiquetados hasta cumplir con un criterio de detención [9, 10].

A pesar del excelente desempeño de los clasificadores basados en DNNs, uno de los principales desafíos es la complejidad computacional involucrada en los procesos de entrenamiento e inferencia. Esto incrementa aún más si el modelo debe ser reentrenado progresivamente como en el caso del aprendizaje activo (AL) [13, 14, 15, 16, 17]. Aunque las unidades de procesamiento gráfico (GPUs) son ampliamente utilizadas y logran acelerar de manera eficiente el entrenamiento y la inferencia, su alto consumo de potencia y gran tamaño limitan su uso en aplicaciones embebidas. Por otro lado, los arreglos de compuertas programables (FPGAs) ofrecen la ventaja de reconfiguración, permitiendo adaptar el hardware a las necesidades específicas de la aplicación [13, 11]. Esta capacidad de personalización, junto con su eficiencia energética, ha incentivado una creciente investigación y desarrollo en el uso de FPGAs para la aceleración de redes neuronales profundas (DNNs).

Capítulo 2

Estado del arte y motivaciones

2.1. Visión por computador

Es esencial que un sistema de inteligencia artificial (AI) pueda comprender su entorno y, de esta forma, tomar decisiones basadas en su comprensión. La visión por computadora (CV) se define como un área de la inteligencia artificial que se ocupa de la percepción visual, extrayendo e interpretando información relevante desde imágenes digitales, videos u otras entradas visuales. Basado en la visión humana, un sistema de visión consiste en dos componentes principales: un dispositivo de adquisición de datos que captura información (función del ojo humano) y un dispositivo que es capaz de interpretar la información detectada (función del cerebro) [1].

Dispositivos de adquisición de datos

Entre las capacidades sensoriales humanas, la visión es el sentido más avanzado. Sin embargo, mientras que la visión humana está limitada a una estrecha banda de luz visible del espectro electromagnético (EM), los dispositivos de adquisición de imágenes abarcan todo el espectro EM, desde los rayos gamma hasta las ondas de radio. Además, las tecnologías actuales permiten generar representaciones visuales a partir de fuentes que normalmente no se asocian con imágenes. Dado que los sistemas de visión están diseñados para realizar tareas específicas, es crucial seleccionar el dispositivo de detección que mejor se ajuste a las características del problema [1, 18].

Dispositivo interpretador

Como dispositivo interpretador, se utilizan algoritmos de visión por computador, que actúan como el *cerebro* del sistema de visión. Inspirados en el aprendizaje de las neuronas biológicas, donde una señal de salida se envía a otras neuronas conectadas si se activan suficientes señales de entrada, los científicos desarrollaron un cerebro artificial con neuronas artificiales. De este modo nacen las *Redes Neuronales Artificiales* (ANN). Aunque cada neurona realiza una función simple por sí sola, la agrupación de neuronas en capas y la conexión de múltiples capas entre sí forman una red capaz de aprender. El uso de redes con múltiples capas de neuronas se denomina *Aprendizaje Profundo* (DL) [1].

2.2. Visión por computador en clasificación

Clasificación es la tarea de asignar una etiqueta a una imagen desde un conjunto de categorías predefinidas [1]. Como se mencionó, los sistemas de visión se componen de dos principales elementos: dispositivo de adquisición de datos y dispositivo interpretador. El trabajo que realiza el dispositivo interpretador se divide en una serie de pasos como muestra la figura 2.2.1.

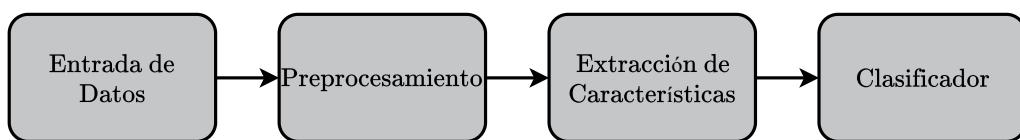


Figura 2.2.1: Pipeline de visión por computador. Fuente: Adaptado de [1].

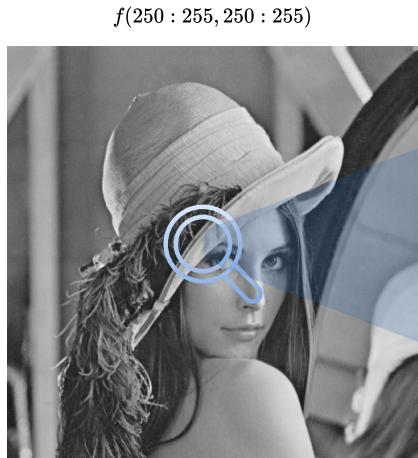
Para llevar a cabo la clasificación de imágenes, existen dos enfoques principales: el tradicional y el basado en Redes Neuronales. El enfoque tradicional utiliza métodos de extracción de características y algoritmos de clasificación convencionales. En contraste, el enfoque basado en redes neuronales emplea técnicas de aprendizaje profundo que integran tanto la extracción de características como la clasificación en un solo proceso [1, 18, 2].

2.2.1. ¿Qué es una imagen? (Entrada de datos)

Una imagen puede ser definida como una función bidimensional $f(x, y)$, donde x e y son las coordenadas espaciales en un plano, y la amplitud de f para cualquier par de coordenadas (x, y) se denomina intensidad o nivel de gris en ese punto. Es importante notar que una imagen se compone de un número finito de elementos llamados píxeles, cada uno con una posición en el plano xy y un valor específico. En una imagen en escala de grises, el valor de intensidad de un píxel abarca un rango de 0 (negro) a 255 (blanco) usando 8 bits [1, 18].

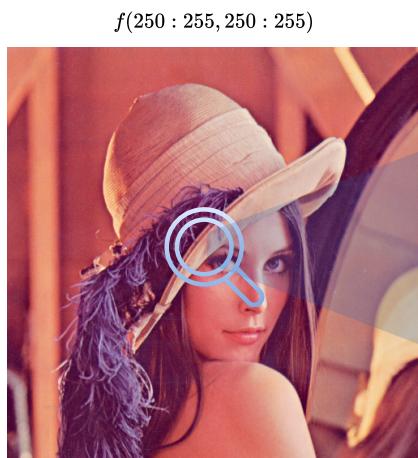
En el caso de imágenes a color, cada píxel tiene un valor de intensidad para cada canal de color. Por ejemplo, en el sistema RGB, cada píxel se representa mediante su intensidad en el canal rojo, intensidad en el canal verde e intensidad en el canal azul. Esta representación se extiende a otros sistemas de color como HSV [1].

Dada la composición de una imagen digital, en un computador estas se tratan como matrices de píxeles. En una imagen en escala de grises, cada píxel determina el valor de intensidad de un solo color, lo que se puede representar mediante una matriz 2D. En contraste, en las imágenes a color, como en el sistema RGB, se utilizan tres matrices, una para cada canal: una matriz para la intensidad del color rojo, otra para la intensidad del color verde y una tercera para la intensidad del color azul. Por lo tanto, las imágenes a color se tratan como matrices 3D, donde la profundidad es tres [1].



187	189	192	197	195	...
190	196	197	199	193	...
193	197	199	192	158	...
199	199	189	149	108	...
201	183	130	100	98	...
...

(a) Imagen en escala de grises



Canales RGP	214	215	215	223	219	...
214	217	220	219	218	...	
177	180	183	187	188	...	
191	192	190	194	185	...	96
170	169	181	180	168	...	34
175	180	176	171	173	...	32
176	180	174	163	109	...	5
169	173	161	93	79	...	1
176	147	81	74	67
...

(b) Imagen en color

Figura 2.2.2: Comparación de Imagen en Escala de Grises y en Color. Fuente: Adaptado de [1].

2.2.2. Preprocesamiento

En esta etapa, se realizan operaciones sobre las imágenes como redimensionamiento, normalización, aumento de datos, eliminación de ruido, corrección de iluminación, segmentación, transformaciones geométricas y filtrado de bordes. El objetivo es mejorar la calidad de la información, de modo que se facilite el análisis y procesamiento computacional en los pasos posteriores [1, 18].



(a) Imagen con ruido

(b) Imagen filtrada

Figura 2.2.3: Preprocesamiento para quitar ruido de una imagen. Fuente: Elaboración propia.

2.2.3. Extracción de características (enfoque tradicional)

El principal objetivo de la extracción de características es obtener la información más relevante de los datos originales y representar esta información en un espacio dimensional reducido, donde el conjunto de características extraídas se denomina vector de características [19]. En el enfoque tradicional existen distintos métodos. La forma más primitiva consiste en considerar los píxeles individuales como características. En 1999, David Lowe presentó un método llamado Características de Escala Invariante (*SIFT*) que transforma una imagen en múltiples vectores de características invariantes a transformaciones geométricas de la imagen, tales como traslación, cambios de escala y rotación. Además, estos vectores son parcialmente invariantes a los cambios de iluminación y adecuados para imágenes de 3 canales [20]. Una técnica similar a *SIFT*, pero que utiliza aproximaciones para acelerar el cálculo, reducir el tiempo de procesamiento y es más robusta, llamada Características Robusta Aceleradas (*SURF*), fue presentada por Herbet Bay, Tinne Tuytelaars y Luc Van Gool en 2006 [21]. Patrón Binario Local (LBP) es una técnica que compara el valor de intensidad de un píxel central (umbral) con todos los píxeles que lo rodean (vecinos) y asigna un número binario que representa la textura de ese vecindario. Luego, se construye un histograma para analizar la distribución de texturas en la imagen. Esta técnica, utilizada en la clasificación de texturas y reconocimiento facial, entre otras

aplicaciones, fue introducida en 1994 por Timo Ojala, Matti Pietikäinen y David Harwood [22]. En 2005, Navneet Dalal y Bill Triggs utilizaron un método de Histogramas de Direcciones de Gradientes (HOG) para la detección de peatones. Este método construye histogramas de direcciones de los gradientes dentro de celdas (subdivisiones de la imagen). Luego, forma bloques a partir de la unión de múltiples celdas; estos bloques son normalizados y concatenados para formar el vector de características, mejorando así la invariancia a la iluminación y el contraste [23]. En 2001, con la intención de reconocer rostros en tiempo real, Paul Viola y Michael Jones presentaron el método de Características de Haar. Este método utiliza características basadas en la suma de píxeles dentro de áreas rectangulares. Para calcular estas sumas de manera eficiente, introdujeron el concepto de imagen integral, una representación de la imagen que permite obtener rápidamente la suma de píxeles en cualquier rectángulo [24]. La técnica de Bolsa de Palabras Visuales (BoVW) fue introducida en 2004 por Gabriella Csurka y sus colaboradores [25]. Esta técnica extrae vectores de características de zonas específicas de una imagen y luego agrupa estos vectores utilizando técnicas de clustering para crear las "palabras visuales". Finalmente, cada imagen puede ser representada por un histograma de "palabras visuales".

2.2.4. Clasificación (enfoque tradicional)

Del mismo modo, existen varios algoritmos de clasificación, entre los cuales se destacan algunos por su relevancia histórica. Existe una colección de clasificadores basados en el teorema de Bayes, conocidos como clasificadores Naive Bayes. Los primeros desarrollos de estos clasificadores se originan en trabajos estadísticos a finales de la década de 1960. El principio detrás de estos clasificadores es que asumen independencia condicional entre las características. Esto significa que cada característica contribuye de manera independiente a la probabilidad de una clase específica, sin tener en cuenta las posibles relaciones o interdependencias entre ellas [26]. En 1967, Thomas Cover y Peter Hart introdujeron el algoritmo de K-vecinos más Cercanos (k-NN). Este método etiqueta una determinada instancia en base a la mayoría de etiquetas que poseen sus k-vecinos más cercanos [27]. Otro tipo de algoritmos utilizados en clasificación son los Árboles de Decisión (DT). Estos se basan en una estructura de árbol y funcionan como un flujo de preguntas.

Cada pregunta representa un nodo de decisión que divide el conjunto de datos en subconjuntos más pequeños hasta llegar a los nodos terminales, donde se obtiene una decisión final (etiqueta). Entre los trabajos más influyentes se encuentran los algoritmos propuestos por John Ross Quinlan [28, 29]. En 2001, Leo Breiman introdujo los Bosques Aleatorios (RF), que son un conjunto de árboles de decisión entrenados con diferentes subconjuntos del conjunto de datos; la decisión final se toma en función de la decisión de muchos árboles [30]. Uno de los trabajos más importantes fue presentado en 1995 por Corinna Cortes y Vladimir Vapnik, Máquinas de Vectores de Soporte (SVM), este algoritmo encuentra el hiperplano que separa las diferentes clases en el espacio de características de manera que maximiza la distancia entre las muestras más cercanas de cada clase. Es conocido por su eficacia en problemas de clasificación binaria [31].

2.2.5. Redes neuronales convolucionales

Esta sección explora los componentes básicos de CNNs, lo cuál es fundamental para entender su funcionamiento y decisiones tomadas en el diseño de arquitecturas expuestas más adelante.

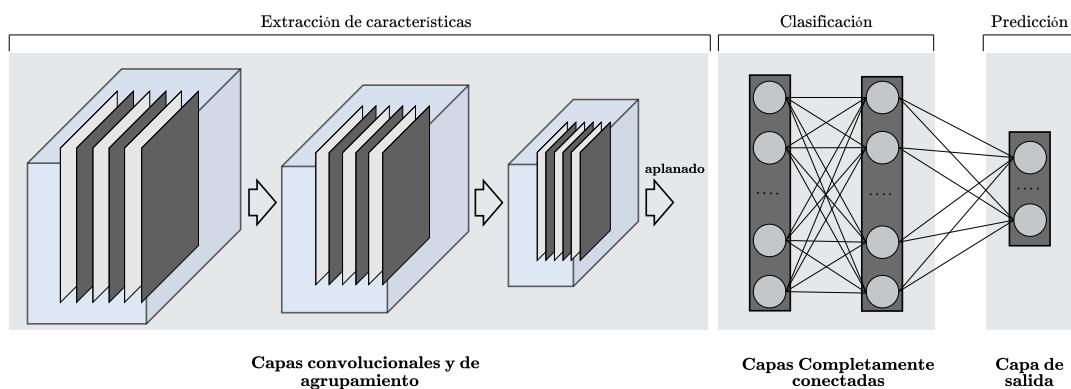


Figura 2.2.4: Red neuronal convolucional. Fuente: Adaptado de [1].

2.2.5.1. Red neuronal

Neurona

Las neuronas son la unidad elemental del sistema nervioso, estas reciben, procesan y envían información hacia otras neuronas en forma de señales químicas o eléctricas.

Inspiradas en el funcionamiento de las neuronas biológicas, se han creado las neuronas artificiales, también conocidas como perceptrones. Al igual que las neuronas biológicas, las neuronas artificiales procesan matemáticamente múltiples entradas para producir una respuesta que puede ser transmitida. La salida de una neurona se define por [1, 2]:

$$f \left(b + \sum_{i=1}^n (x_i \cdot w_i) \right) \quad (2.2.1)$$

Donde $f(\cdot)$ es la función de activación (se detallará más adelante), x_i representa la señal de entrada, n es el número total de señales, w_i es el peso asignado a la entrada x_i , y b es el sesgo.

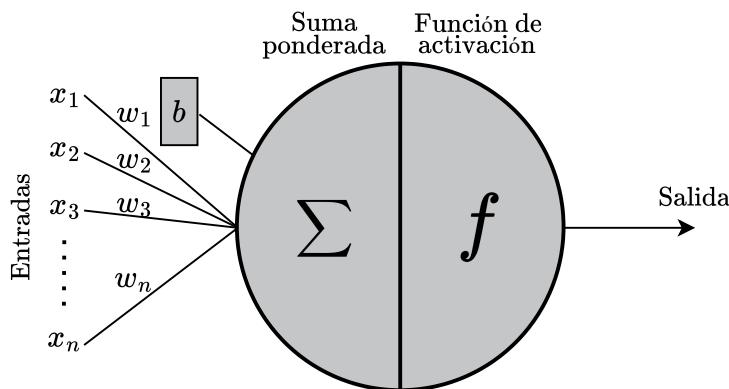


Figura 2.2.5: Neurona artificial. Fuente: Adaptado de [1]

Perceptrón multicapa (MLP)

MLP se constituye de una capa de entrada, una o más capas ocultas y una capa de salida. Cada neurona en una determinada capa, se encuentra conectada a todas las neuronas de la capa anterior y a todas las neuronas de la capa siguiente, por lo tanto, las salidas de las neuronas de una capa, son las entradas a las neuronas de la siguiente capa, permitiendo la transferencia de información entre capas a través de la red. Las capas anteriores en una red neuronal aprenden características generales de los datos, mientras que las capas posteriores se especializan en aprender características cada vez más específicas y detalladas [1, 2].

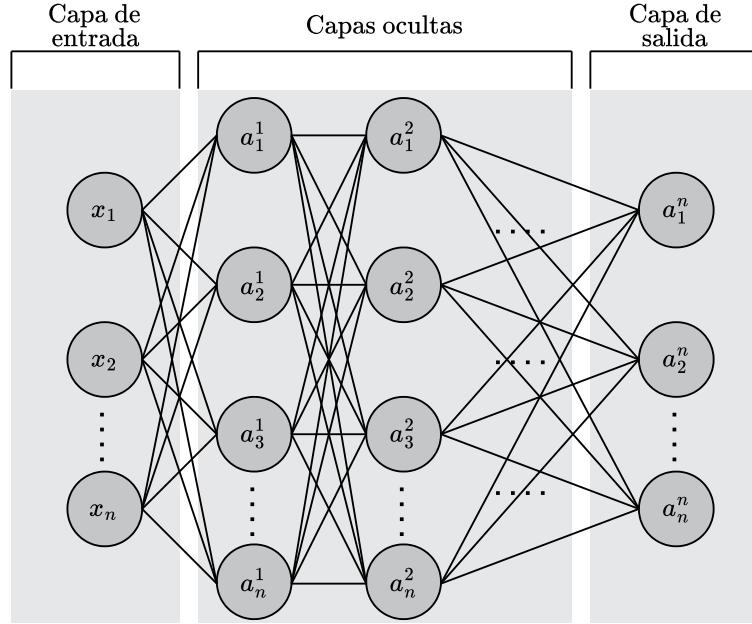


Figura 2.2.6: Red perceptrón multicapa. Fuente: Adaptado de [1]

El proceso de aprendizaje en un MLP se basa en dos fases principales: *feedforward* y *backpropagation*. En la fase de feedforward, la información se propaga desde la capa de entrada hasta la capa de salida, generando predicciones basadas en las características aprendidas. En la fase de backpropagation, se calcula el error entre las predicciones y las respuestas reales, y este error se utiliza para ajustar los pesos de la red mediante un algoritmo de optimización. La actualización para un determinado peso está dado por [1]:

$$w_{\text{nuevo}} = w_{\text{antiguo}} - \alpha \cdot \frac{\partial E}{\partial w}, \quad (2.2.2)$$

donde w_{nuevo} es el valor actualizado del peso, w_{antiguo} es el valor del peso antes de la actualización, α es la tasa de aprendizaje y $\frac{\partial E}{\partial w}$ es el gradiente de la función error E con respecto al peso w .

2.2.5.2. Arquitectura de red neuronal convolucional (CNN)

Capa convolucional

Una capa convolucional es un componente fundamental en las CNNs. Esta capa

está compuesta por varios filtros convolucionales, cada uno de los cuales contiene pesos, generalmente con dimensiones de 3×3 (9 pesos), 5×5 (25 pesos) o 7×7 (49 pesos). El número de canales de cada filtro debe ser igual al número de canales de la imagen de entrada. Estos filtros recorren la imagen en pequeñas regiones, píxel por píxel, extrayendo características relevantes y generando mapas de características. Las capas anteriores extraen características generales de las imágenes, como líneas, bordes y texturas. Por otro lado, las capas posteriores se enfocan en extraer información más específica y compleja, como formas y patrones detallados [1, 2].

Capa de *pooling* (agrupamiento)

Una capa de pooling es otro componente importante en las CNNs. Su función es reducir las dimensiones espaciales (ancho y alto) de los mapas de características, disminuyendo así la cantidad de parámetros y el costo computacional, mientras se retienen las características más relevantes. Existen varios tipos de pooling, siendo los más comunes el max pooling (agrupamiento máximo) y el average pooling (agrupamiento promedio) [1, 2].

Función de activación

La suma ponderada realizada por un perceptrón es una operación lineal que relaciona las entradas con la salida. Para abordar la necesidad de clasificar datos que no son linealmente separables, se introducen las funciones de activación, también conocidas como funciones de activación no lineales. Estas funciones añaden no linealidad a la red y mejoran su desempeño. Existen distintas funciones de activación, entre ellas se encuentran: *tanh*, que ajusta todos los valores entre -1 y 1 ; *sigmoidal*, que ajusta todos los valores a una probabilidad entre 0 y 1 ; *softmax*, una generalización de la función sigmoidal que se utiliza cuando hay dos o más clases; *ReLU*, que actúa como una función identidad para valores mayores a cero y es 0 para cualquier entrada menor o igual a cero; *Leaky ReLU*, que a diferencia de ReLU introduce una pendiente aproximadamente de $0,01$ para valores negativos; y *GELU*, que utiliza una distribución gaussiana para decidir cuánto de la entrada pasa a la siguiente capa. Debido a la rapidez de la función ReLU en el entrenamiento con descenso de gradiente, es la más utilizada en las capas ocultas de las redes neuronales, mientras que para problemas multiclas se suele utilizar la función softmax en la capa de salida, ya que proporciona una

distribución de probabilidad sobre las diferentes clases [1, 32, 3, 33, 34, 35].

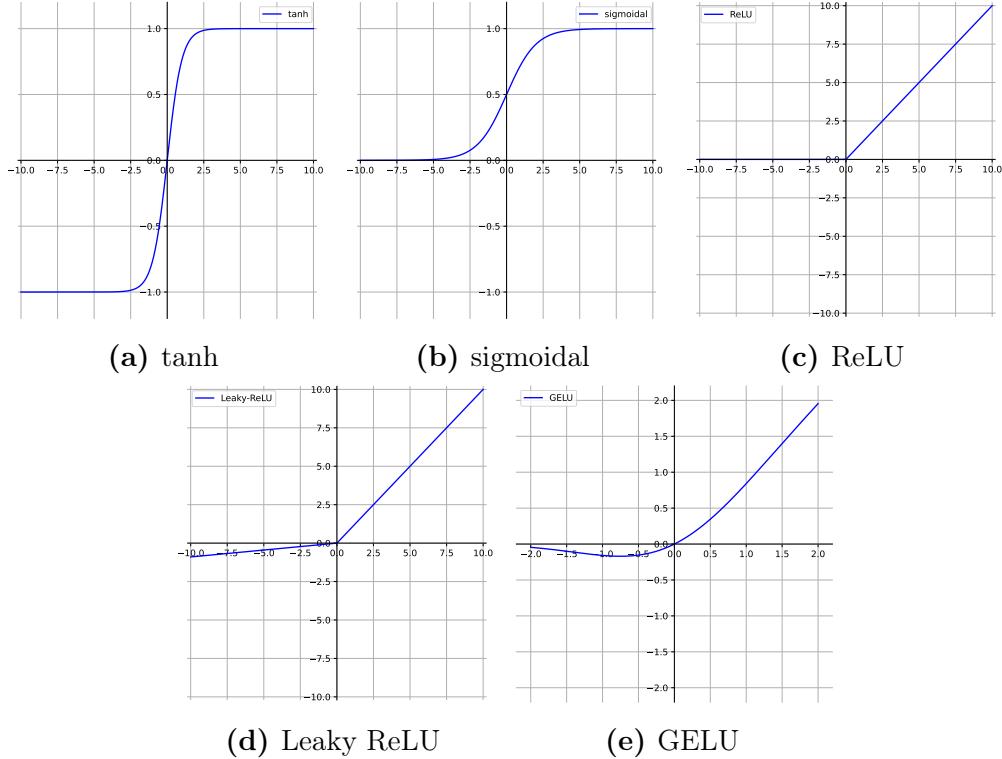


Figura 2.2.7: Funciones de activación. Fuente: Adaptado de [2].

Capa completamente conectada (FC)

Las capas completamente conectadas (FC), también conocidas como capas densas, se ubican después de las capas encargadas de la extracción de características. Estas capas están conformadas por neuronas que están completamente conectadas a todas las neuronas de la capa anterior y a todas las neuronas de la capa posterior. Las capas completamente conectadas utilizan las características extraídas por las capas anteriores para realizar la clasificación de los datos. En otras palabras, actúan como el clasificador de la red [1, 2].

Técnicas de regularización

Problemas como el sobreajuste y el subajuste están directamente relacionados con la complejidad de los datos de entrada en comparación con la complejidad de la red neuronal. Para reducir el riesgo de sobreajuste y mejorar la capacidad de generalización del modelo, se utilizan técnicas de regularización. *Dropout* implica

apagar aleatoriamente una fracción de las neuronas durante el entrenamiento, lo que evita que el modelo dependa demasiado de patrones específicos del conjunto de entrenamiento. La regularización $L2$ añade una penalización proporcional al cuadrado de los valores de los pesos a la función de pérdida, ayudando a controlar la magnitud de los pesos. Además, la normalización por lotes ajusta y escala las salidas de las neuronas en cada capa para mantener activaciones en un rango más estable, acelerando el entrenamiento y mejorando la estabilidad del modelo. La detención temprana es otra técnica utilizada para prevenir el sobreajuste, donde alguna métrica del rendimiento del modelo es monitoreada y evaluada según un criterio predefinido para decidir si el entrenamiento debe continuar o detenerse [1, 2, 36].

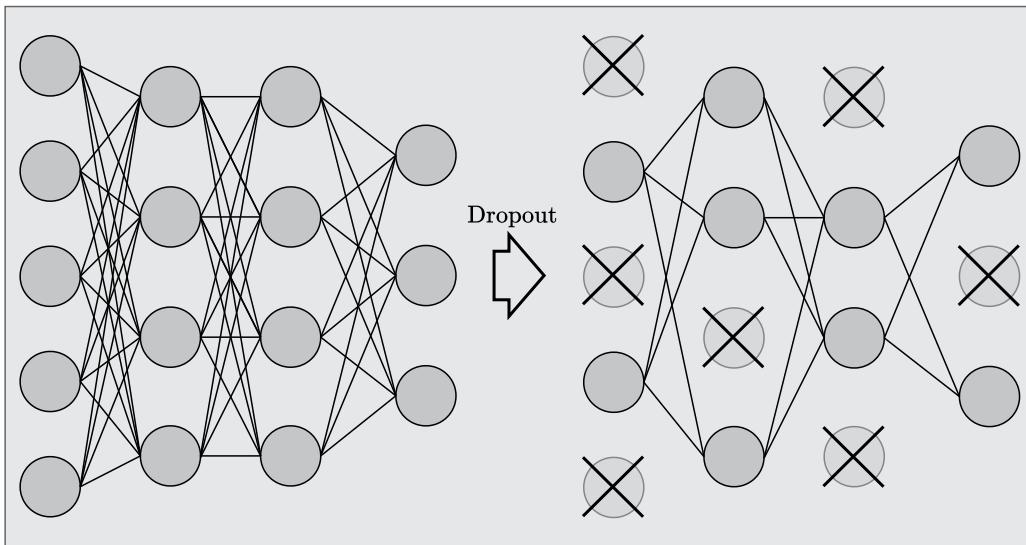


Figura 2.2.8: Dropout. Fuente: Adaptación de [1].

Función de Error

La función de error proporciona una medida del desempeño de la red basada en las predicciones generadas en la capa de salida. Entre las más utilizadas en problemas de clasificación se encuentran: el Error Cuadrático Medio (MSE), que evalúa la diferencia promedio entre las predicciones y los valores verdaderos, y la Entropía Cruzada, que mide la disimilitud entre la distribución de probabilidad predicha y la distribución real de las clases [1, 2].

Optimizador

Un mejor desempeño de la red está directamente relacionado con la reducción del error. Una vez definida la función de error, el objetivo es encontrar los pesos (es decir, los parámetros de la función de error) que minimicen dicho error, convirtiendo este proceso en un problema de optimización. El método de optimización más utilizado en redes neuronales es el descenso del gradiente. Este método busca minimizar la función de error actualizando los parámetros en la dirección opuesta al gradiente de la función. El tamaño de los pasos hacia el mínimo local está determinado por la tasa de aprendizaje α . Algunos algoritmos de descenso del gradiente son: Descenso del Gradiente Estocástico (SGD), Momentum, RMSprop, Adam y Adagrad [1, 2, 37, 38, 39].

2.2.5.3. Arquitecturas Clásicas

LeNet

Lecun y colaboradores propusieron la arquitectura LeNet-5 en 1998, cuyo objetivo era clasificar imágenes de caracteres escritos a mano [3]. Esta red cuenta con 5 capas de pesos (capas entrenables): 3 capas convolucionales (6 filtros de 5x5, 16 filtros de 5x5 y 120 filtros de 5x5), intercaladas con capas de agrupamiento promedio, y dos capas completamente conectadas (una con 84 neuronas y otra de salida con 10 neuronas). En las capas ocultas se utiliza la función de activación sigmoidal, mientras que en la capa de salida se emplea softmax.

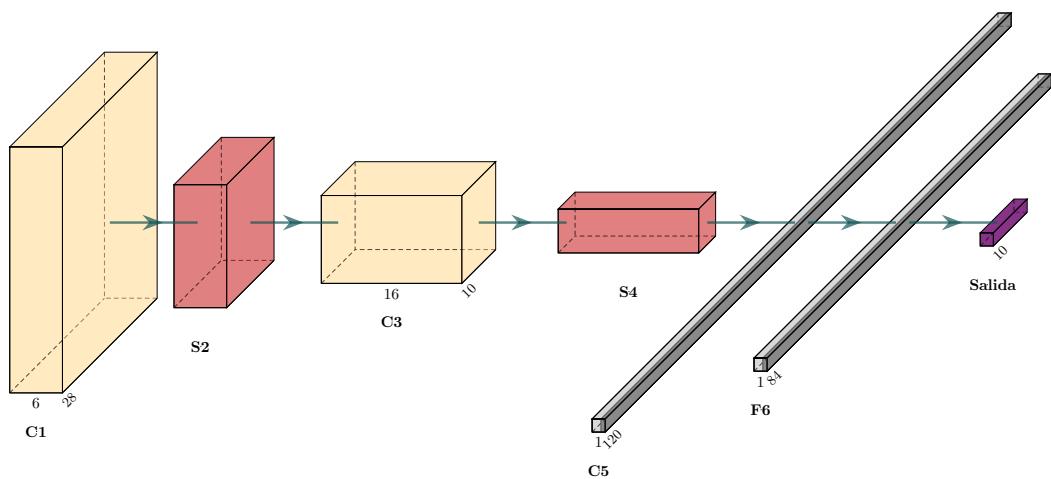


Figura 2.2.9: Arquitectura de LeNet. Fuente: Adaptado de [1, 3].

Cabe destacar que esta arquitectura fue pionera en su época y sirvió de inspiración para numerosos trabajos posteriores. A día de hoy con sus 61,706 parámetros se considera una red relativamente sencilla y menos efectiva para abordar problemas más complejos.

AlexNet

La principal motivación detrás de AlexNet fue desarrollar una red capaz de superar el desempeño en problemas más complejos [4]. Krizhevsky y sus colaboradores diseñaron una red entrenada con 1.2 millones de imágenes de alta resolución, provenientes de 1,000 clases del conjunto de datos ImageNet. La red ganó de manera destacada la competencia de clasificación de imágenes ILSVRC en 2012. La arquitectura consiste en 8 capas de pesos: 5 capas convolucionales (96 filtros de 11x11, 256 filtros de 5x5, 384 filtros de 3x3, 384 filtros de 3x3 y 256 filtros de 3x3), seguidas de capas de agrupamiento máximo (max pooling) después de las primeras dos y la última capa del extractor de características. Además, incluye 3 capas completamente conectadas (las dos primeras con 4096 neuronas cada una y la capa de salida con 1000 neuronas). La función de activación utilizada en las capas ocultas es ReLU, mientras que para la capa de salida se emplea softmax. Para prevenir el sobreajuste, se aplicó dropout en las capas completamente conectadas de 4096 neuronas con una probabilidad de 0.5. AlexNet aumenta considerablemente el número de parámetros a 62 millones.

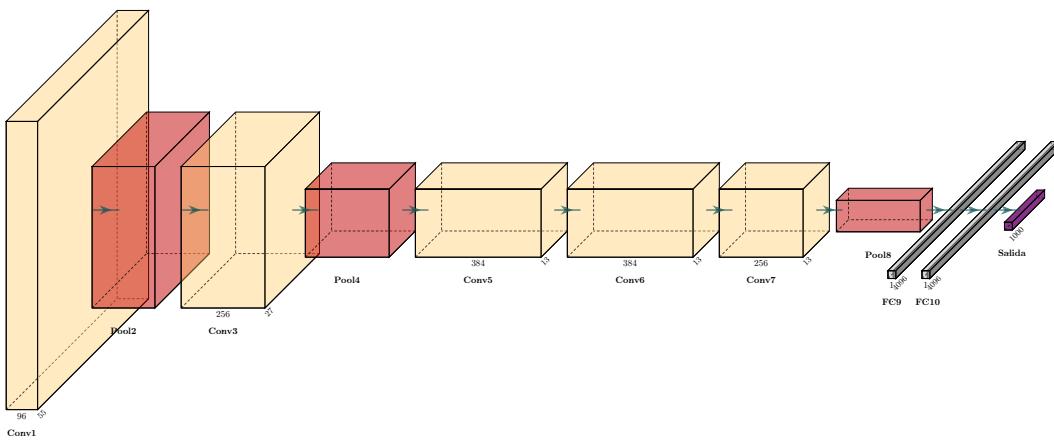


Figura 2.2.10: Arquitectura AlexNet. Fuente: Adaptado de [1, 4].

VGGNet

Karen Simonyan y Andrew Zisserman crearon en 2014 una arquitectura aún más profunda que AlexNet, la red VGGNet [5]. Esta arquitectura facilita algunas decisiones de diseño, como la elección del tamaño de los filtros en las capas convolucionales, utilizando pequeños filtros de 3×3 en cada capa. La idea detrás de esto es que múltiples pequeños filtros sucesivos añaden más profundidad y, por lo tanto, aumentan la capacidad de la red para aprender características más complejas.

VGGNet agrupa varias capas convolucionales idénticas que mantienen las mismas dimensiones de la entrada y luego agrega una capa de agrupamiento máximo que reduce las dimensiones a la mitad. Existen distintas configuraciones de VGGNet que difieren en la organización de las capas convolucionales. Una configuración común es VGG16, que se conforma de un primer bloque con dos capas convolucionales de 64 filtros de 3×3 , un segundo bloque con dos capas convolucionales de 128 filtros de 3×3 , un tercer bloque con tres capas convolucionales de 256 filtros de 3×3 , un cuarto bloque con tres capas convolucionales de 512 filtros de 3×3 y un quinto bloque con tres capas convolucionales de 512 filtros de 3×3 . Luego de los bloques, hay tres capas completamente conectadas, iguales en todas las configuraciones: las dos primeras de 4096 neuronas cada una y la capa de salida de 1000 neuronas. En las capas ocultas se utiliza ReLU como función de activación y en la salida softmax. Dropout y regularización L2 se emplean para evitar el sobreajuste. Esta configuración posee 138 millones de parámetros.

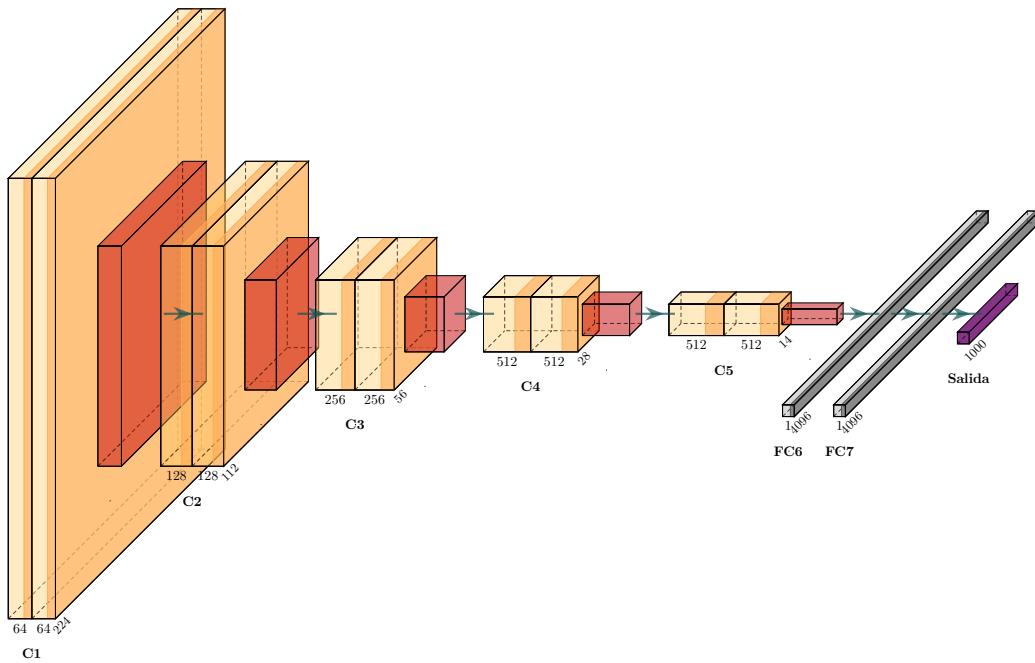


Figura 2.2.11: Arquitectura VGGNet16. Fuente: Adaptado de [1, 5].

Inception (GoogLeNet)

Inception fue presentada en 2014 por un grupo de investigadores de Google [6]. Esta arquitectura permite construir redes mucho más profundas con un menor número de parámetros (12 veces menos) en comparación con VGGNet e introduce un componente clave llamado módulo Inception. Mientras que las arquitecturas previas incluían capas convolucionales con distintos tamaños de filtros, seguidas de capas de agrupamiento intercaladas, el módulo Inception agrupa estas decisiones en un solo módulo. De esta forma, la salida de un módulo inception está dada por la concatenación de la salida de 4 ramas: una con una capa convolucional de 1×1 ; otra con una capa convolucional de 3×3 ; una tercera con una capa convolucional de 5×5 ; y una cuarta que aplica una capa de agrupamiento máximo de 3×3 . Además, dado el costo computacional que agregan los filtros de mayor tamaño (como 5×5), se añaden capas de reducción de dimensiones en cada una de las ramas, es decir, capas convolucionales de 1×1 con un bajo número de filtros.

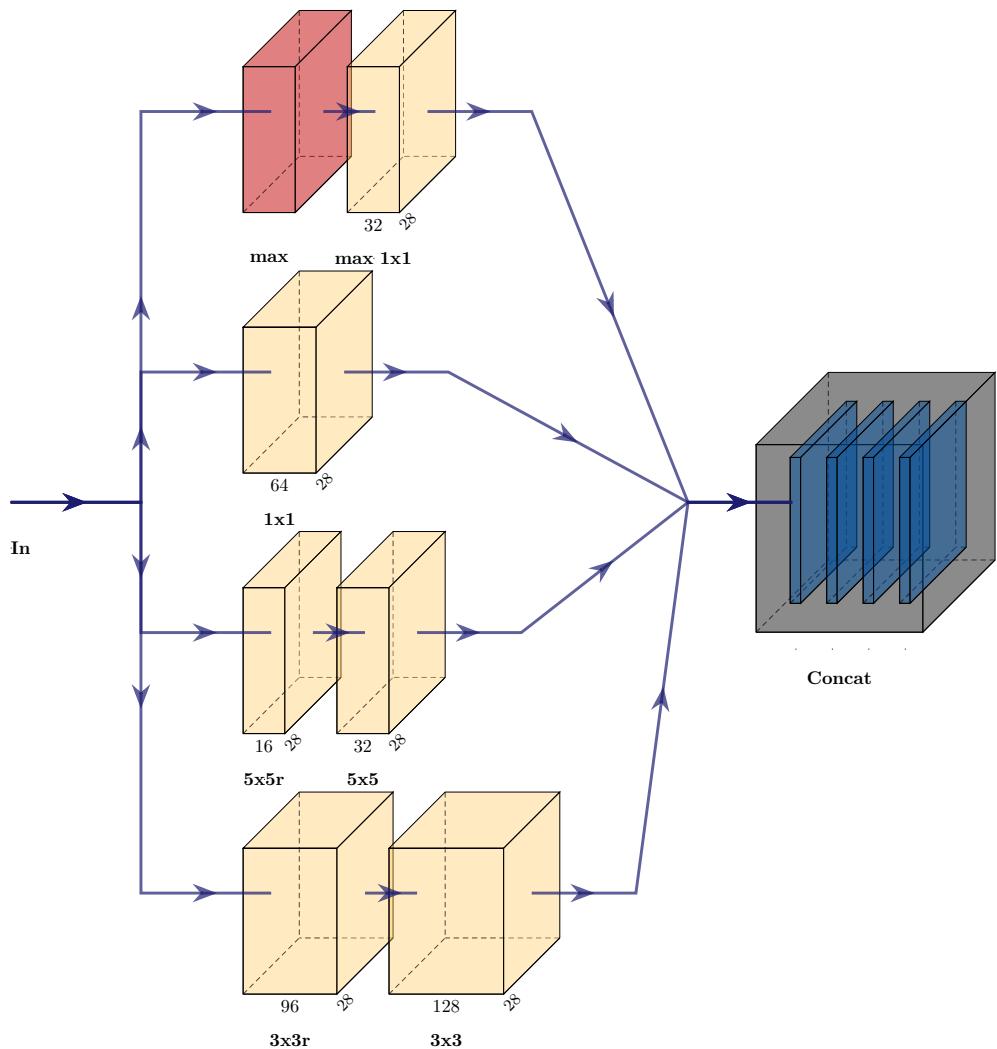


Figura 2.2.12: Módulo Inception. Fuente: Adaptado de [1, 6].

Los creadores participaron en la competencia ILSVRC en 2014 con una versión de Inception llamada GoogLeNet. Esta arquitectura se compone de tres partes principales. La parte A comienza con una capa convolucional de 64 filtros de 7×7 , seguida de una capa de agrupamiento máximo de 3×3 . Luego, se aplica una capa convolucional con 64 filtros de 1×1 (para reducción de dimensiones), seguida de una capa convolucional con 192 filtros de 3×3 , y finaliza con una capa de agrupamiento máximo de 3×3 . La parte B está formada por nueve módulos Inception organizados en tres bloques principales, con 2, 5 y 2 módulos Inception, respectivamente. Cada bloque está seguido por una capa de agrupamiento: los primeros dos bloques por una capa de agrupamiento máximo de 3×3 y el último bloque por una capa de agrupamiento promedio de 7×7 . Finalmente, la parte C

de la arquitectura incluye una capa completamente conectada de 1000 neuronas con dropout, seguida por la capa de salida con 1000 neuronas. En las capas ocultas se utiliza la función de activación ReLU, mientras que en la capa de salida se emplea softmax para la clasificación.

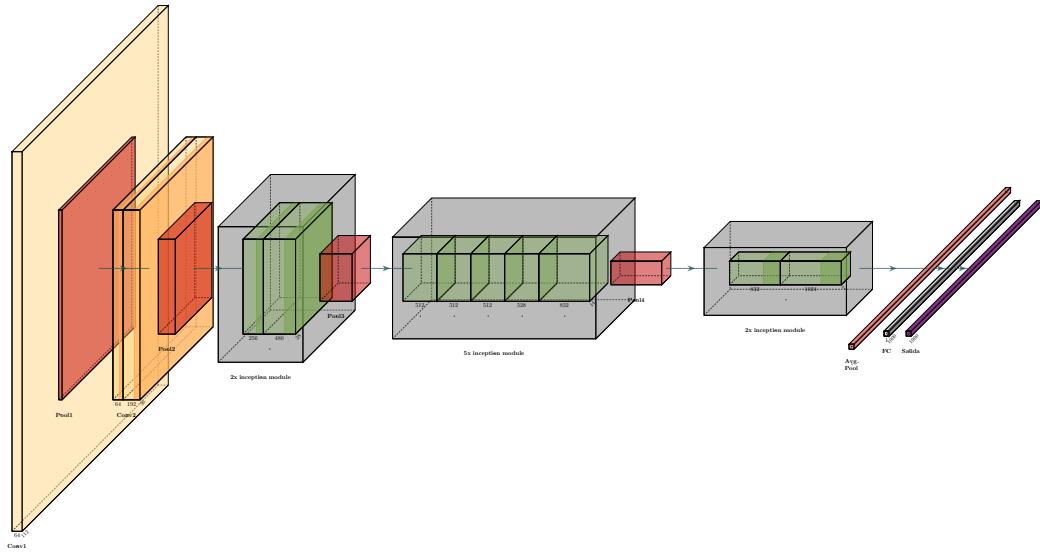


Figura 2.2.13: Arquitectura GoogLeNet. Fuente: Adaptado de [1, 6].

Red Neuronal Residual (ResNet)

La red neuronal residual fue desarrollada en 2015 por un equipo de Microsoft Research [7]. El diseño de redes muy profundas ofrece una mayor capacidad de aprendizaje, pero también presenta dos problemas principales: el sobreajuste, que ResNet mitiga mediante un extenso uso de normalización por lotes, y el desvanecimiento del gradiente, que ocurre cuando el gradiente utilizado para actualizar los pesos se vuelve muy pequeño, impidiendo el aprendizaje efectivo en las capas anteriores. Para abordar este problema, ResNet incorpora un módulo denominado módulo residual con una conexión de salto. Esta red logró resultados destacados en la competencia ILSVRC 2015.

Un módulo residual se compone de dos ramas: una de ellas incluye tres capas convolucionales con filtros de 1×1 , 3×3 y 1×1 , respectivamente. Cada una de estas capas utiliza normalización por lotes y la función de activación ReLU. La otra rama es una conexión de salto que suma directamente la entrada del módulo

a la salida de la primera rama antes de aplicar la función de activación en la última capa convolucional. Cuando las dimensiones de la entrada y la salida de la rama convolucional no coinciden, se utiliza una conexión de salto con reducción, lograda mediante una capa convolucional de 1×1 en la rama de la conexión de salto, que ajusta las dimensiones de la entrada para que coincidan con las de la salida del bloque residual, permitiendo así la suma entre la entrada y la salida.

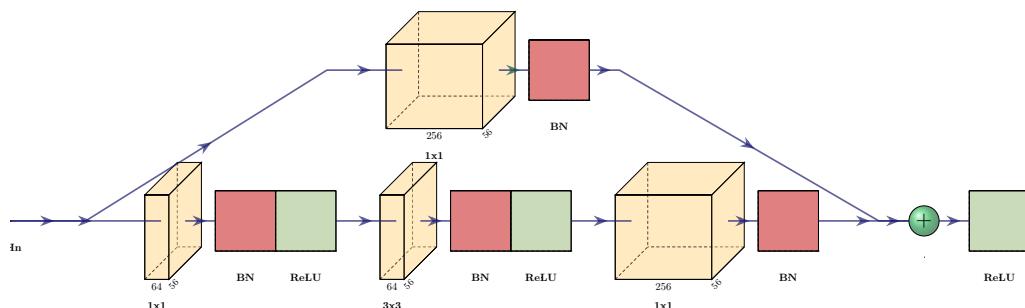


Figura 2.2.14: Módulo ResNet (conexión residual). Fuente: Adaptado de [1, 7].

Al igual que VGGNet, ResNet tiene distintas configuraciones que varían en la profundidad: ResNet-18, ResNet-34, ResNet-50, ResNet-101 y ResNet-152. En el caso de ResNet-50, esta tiene una primera parte que se compone de una capa convolucional con 64 filtros de 7×7 con función de activación ReLU, seguida de una capa de agrupamiento máximo de 3×3 . Luego, la red se divide en cuatro bloques de módulos residuales. El primer bloque contiene tres módulos con capas convolucionales de 64, 64, y 256 filtros respectivamente; el segundo bloque contiene cuatro módulos con capas convolucionales de 128, 128, y 512 filtros respectivamente; el tercer bloque incluye seis módulos con capas convolucionales de 256, 256, y 1024 filtros respectivamente; y el cuarto bloque tiene tres módulos con capas convolucionales de 512, 512, y 2048 filtros respectivamente. Finalmente, la red incluye una capa de agrupamiento promedio y una capa completamente conectada de salida con 1000 neuronas y función de activación softmax.

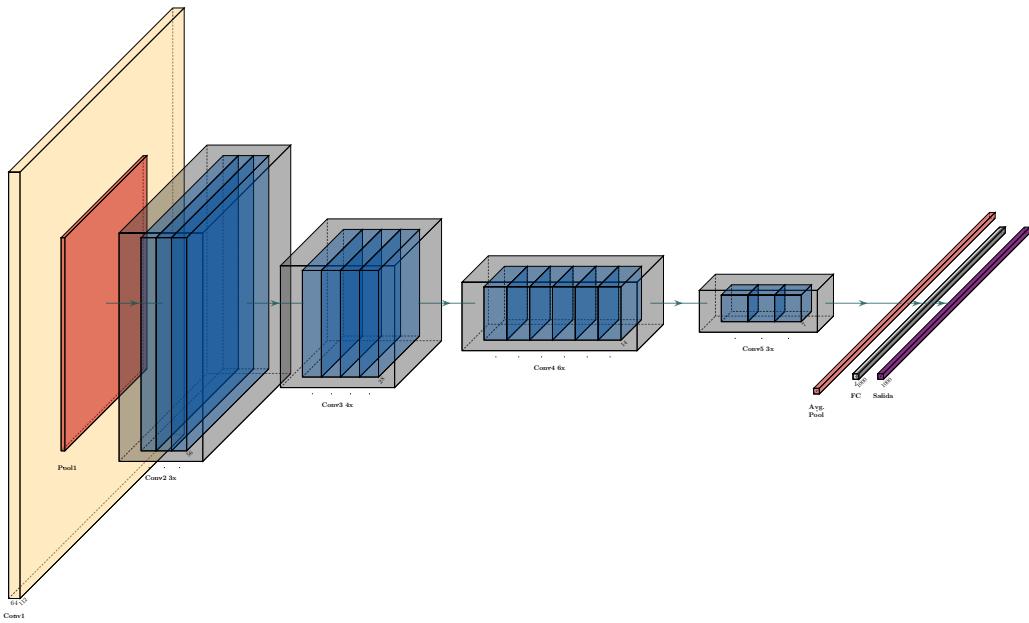


Figura 2.2.15: Arquitectura ResNet-50, Fuente: Adaptado de [1, 7].

2.2.6. Transformadores visuales (ViT)

Los transformadores visuales fueron introducidos en 2021 por Alexey Dosovitskiy y sus colaboradores [8]. Estos modelos, inspirados en la exitosa investigación de Ashish Vaswani y su equipo en 2017 [40], que se centró en el procesamiento del lenguaje natural (NLP), aprovechan un mecanismo de atención para capturar relaciones entre parches dentro de las imágenes.

2.2.6.1. Arquitectura de transformador visual

Esta sección describe las partes y procesos esenciales que conforman un ViT.

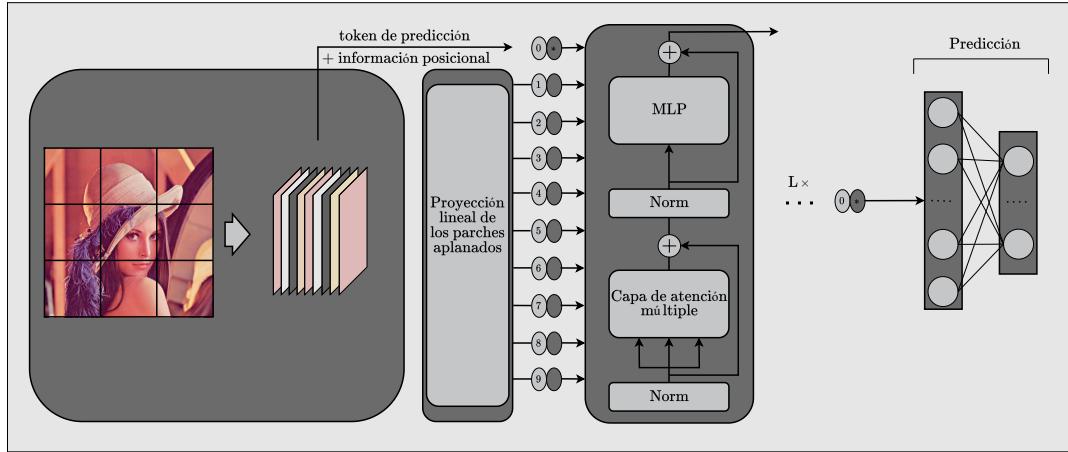


Figura 2.2.16: Arquitectura de transformadores visuales. Fuente: Adaptado de [8].

Embebido de parches

En una etapa inicial, las imágenes son subdivididas en parches de igual dimensión. Si las dimensiones de una imagen son (H, W, C) , donde H es la altura, W es el ancho, y C es el número de canales, y las dimensiones de cada parche son (P, P) , entonces el número de parches (subdivisiones de la imagen) está dado por $N = H \times W / P^2$. Luego, cada parche es aplanado (flattened) para formar un vector de características de dimensiones $(1, P^2 \times C)$, llamado token. Estos tokens se ajustan mediante una proyección lineal entrenable a una dimensión D , de modo que cada token tiene dimensiones $(1, D)$. Junto a los token, es concatenado un *token de predicción*, este token obtiene información del resto de tokens y es utilizado para realizar la predicción final. Además, se añade (suma) información relativa a la posición de los tokens dentro de las imágenes. Finalmente, una secuencia de tokens es pasada hacia el transformador [8, 40].

Atención

Es el mecanismo que busca relaciones entre las distintas subdivisiones (parches) de las imágenes. A través de este proceso, el modelo aprende a comprender el contexto global de la imagen evaluando la importancia relativa de cada parche en función de los demás parches presentes. El cálculo de la atención se define por la siguiente fórmula [8, 40]:

$$\text{attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V, \quad (2.2.3)$$

donde Q , K y V son las proyecciones lineales de las consultas, claves y valores, respectivamente, y d_k es la dimensión de las claves.

Además, *atención múltiple* permite ejecutar k operaciones de atención en paralelo denominadas *cabezas*, donde cada cabeza utiliza proyecciones lineales independientes, generalmente con dimensiones D/k . Las salidas de todas las cabezas se concatenan y proyectan a la dimensión original D . De esta manera, el modelo puede captar aspectos diferentes en las relaciones de las entradas. Esta operación es realizada por la *capa de atención múltiple* [8, 40].

Codificador

Un codificador, es una capa que se compone de dos subcapas: atención múltiple y perceptrón multicapa. A la salida de ambas subcapas, se añade normalización por lotes y una conexión de salto (conexión residual). Un transformador visual incluye varios codificadores [8, 40].

Predicción

Una vez que una determinada entrada pasa a través todas las capas de codificación, el token de predicción es separado de los demás token y pasado a un clasificador compuesto de una o más capas de densas. Este se encarga de generar la predicción final del modelo [8, 40].

2.3. Aprendizaje activo (AL)

Para el entrenamiento de clasificadores de imágenes basados en redes neuronales, es necesario recolectar suficiente cantidad de imágenes etiquetadas (datos de entrenamiento). Si bien en ciertas áreas, la tarea de obtener datos no etiquetados es lograda con facilidad, el proceso de etiquetado puede volverse costoso en tiempo y recursos, por factores como la necesidad de un anotador experto o una serie de pasos previos a la anotación. En escenarios de tal tipo, el aprendizaje activo (AL) ofrece una solución que disminuye los costos involucrados en el etiquetado. La idea detrás de AL es que, un modelo de aprendizaje automático puede alcanzar un buen desempeño siendo entrenado con menos datos etiquetados, si es capaz de

elección de los datos desde los que aprende [9, 10].

Esta sección revisa las ideas clave y componentes fundamentales de algoritmos de aprendizaje activo.

2.3.1. ¿Qué es el aprendizaje activo?

AL es un sub-área del aprendizaje automático. Con la intención de superar los problemas asociados al proceso de etiquetado, AL selecciona un pequeño conjunto de las instancias (datos) más informativas desde un conjunto de datos no etiquetados, para consultar a un anotador (experto) la etiqueta correspondiente. Este difiere del aprendizaje tradicional (PL), en el cuál las instancias son seleccionadas aleatoriamente [9, 10].

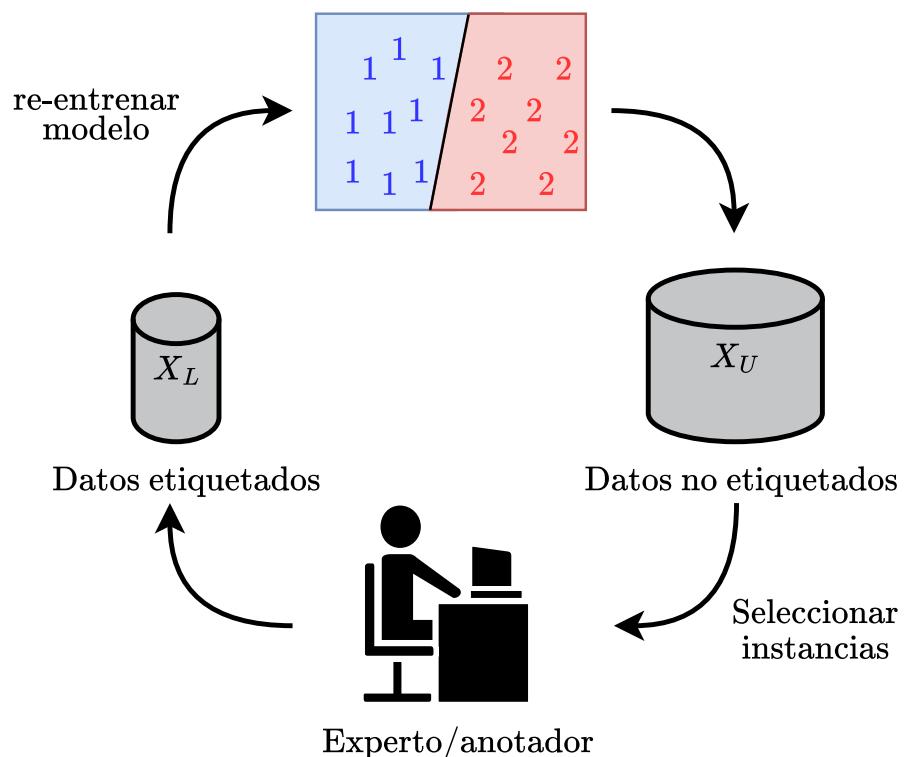


Figura 2.3.1: Pasos de un algoritmo de AL con escenario *pool-based*. Fuente: Adaptado de [9, 10].

2.3.2. Escenarios de AL

Hay tres principales escenarios en los que ALs pueden consultar por etiquetas. En *membership query synthesis* el modelo de aprendizaje activo genera instancias de forma sintética. El gran problema que presenta esta configuración es que las instancias generadas podrían no ser interpretables por un anotador humano. Sin embargo, en aplicaciones experimentales puede ser aprovechada esta característica [9, 10]. En *stream-based selective sampling*, instancias no etiquetadas son seleccionadas una a la vez y consultadas según una medida de su información contenida. Esto lo vuelve una opción aplicable cuando los recursos de memoria o procesamiento son escasos [9, 10]. El escenario más conocido, y más utilizado en clasificación de imágenes es *pool-based*. En esta configuración, se obtiene una medida de la información de todas/algunas instancias en un gran conjunto de datos no etiquetados X_U a partir de una estrategia de consulta q que utiliza un clasificador h entrenado en un pequeño conjunto de datos etiquetados X_L . Las n_q instancias más informativas son consultadas a un experto E para ser etiquetadas, de este modo, el clasificador es re-entrenado hasta que se cumple un criterio de detención SC [9, 10].

2.3.3. Estrategia de consulta

Lo que diferencia un algoritmo de AL de otro, es la estrategia de consulta empleada, es decir, la forma en que el modelo de aprendizaje activo evalúa cuáles son las instancias más informativas [9, 41]. Un enfoque basado en la información plantea seleccionar las instancias de las cuales el modelo está más incierto [9, 41]. Por ejemplo, Lewis y Gale, propusieron seleccionar la instancia más incierta para un clasificador probabilístico (0,5 en clasificación binaria) [?]. Otra técnica presentada por Schefer et al., calcula el margen entre las dos etiquetas más probables para una instancia y selecciona las instancias con el menor margen [9, 41]. En la selección basada en entropía, se elige la instancia con la mayor entropía, considerando todas las posibles etiquetas [9, 10, 42]. Otro método denominado consulta por comité (QBC) fue propuesto por Seung et al., emplea múltiples clasificadores y selecciona las instancias en las que el comité de clasificadores presenta mayor desacuerdo [43]. Otro enfoque es el basado en la representación, que se centra en la estructura (características, relaciones) de los datos no etiquetados, para representar completamente el espacio de entrada. Settles et al., presentaron una

estrategia de consulta que selecciona instancias desde las zonas con mayor densidad de instancias [42]. Otros métodos agrupan (*clustering*) los datos no etiquetados y luego consultan por aquellas instancias que encuentran más cercanas a los centroides [44, 45].

2.4. Acelerador Hardware

El entrenamiento de una red neuronal profunda mediante feedforward y backpropagation involucra una gran cantidad de operaciones matemáticas, lo que hace necesario disponer de recursos computacionales capaces de llevar a cabo la tarea [13, 14, 15, 16]. En el contexto del aprendizaje activo (AL), donde el modelo se re-entrena múltiples veces a medida que se incorporan nuevos datos etiquetados, esta demanda computacional se intensifica aún más. Generalmente, se utilizan arquitecturas de computadores seriales, que dividen un algoritmo determinado en una secuencia de operaciones aritméticas y lógicas realizadas por la ALU (unidad aritmética lógica), con la gestión de datos e instrucciones coordinada por la CPU (unidad de procesamiento central) [11]. La capacidad de realizar solo una operación a la vez convierte el tiempo en un recurso crítico con procesadores seriales [13, 14, 15, 11, 16]. Frente a esta limitación, un avance significativo es el uso de GPUs (unidades de procesamiento gráfico), que explotan el uso de muchos núcleos trabajando en paralelo y son ampliamente utilizadas hoy en día [13, 11, 16]. Aunque las CPUs y GPUs son ampliamente utilizadas en el dominio del DL, los requisitos de tamaño y consumo de energía complican su uso en aplicaciones embebidas. Una solución es el uso de FPGAs (arreglos de compuertas programables), donde es posible reconfigurar recursos de hardware para ajustarlos a las necesidades de la aplicación. Del mismo modo, los ASICs (circuitos integrados de aplicación específica) son aún más eficientes en la cantidad de recursos utilizados para acelerar la aplicación específica, pero tienen menos flexibilidad que una FPGA y, en general, el costo de fabricación asociado es elevado [13, 11].

2.4.1. ¿Qué es una FPGA?

Las FPGAs son circuitos integrados reconfigurables. La idea detrás de las FPGAs es tener un circuito genérico, que puede ser re-programado para adoptar un propósito

específico que se ajusta a los requerimientos de una aplicación [11]. De esta forma, una FPGA permite paralelizar cómputos y lograr implementaciones más eficientes en términos de velocidad, consumo de potencia, ancho de banda y utilización de recursos [13, 11]. Los componentes principales de una FPGA son los Elementos Lógicos (LEs), también conocidos como Bloques Lógicos Configurables (CLBs). Estos bloques permiten implementar funciones de lógica combinacional y secuencial. Los LEs se organizan en una estructura de rejilla y están interconectados mediante una matriz de conexiones reconfigurables, que permite conectar los bloques de diversas maneras. Además, el arreglo de LEs está rodeado por Elementos de Entrada/Salida (IOEs) que facilitan la interacción con el exterior [13, 11, 12].

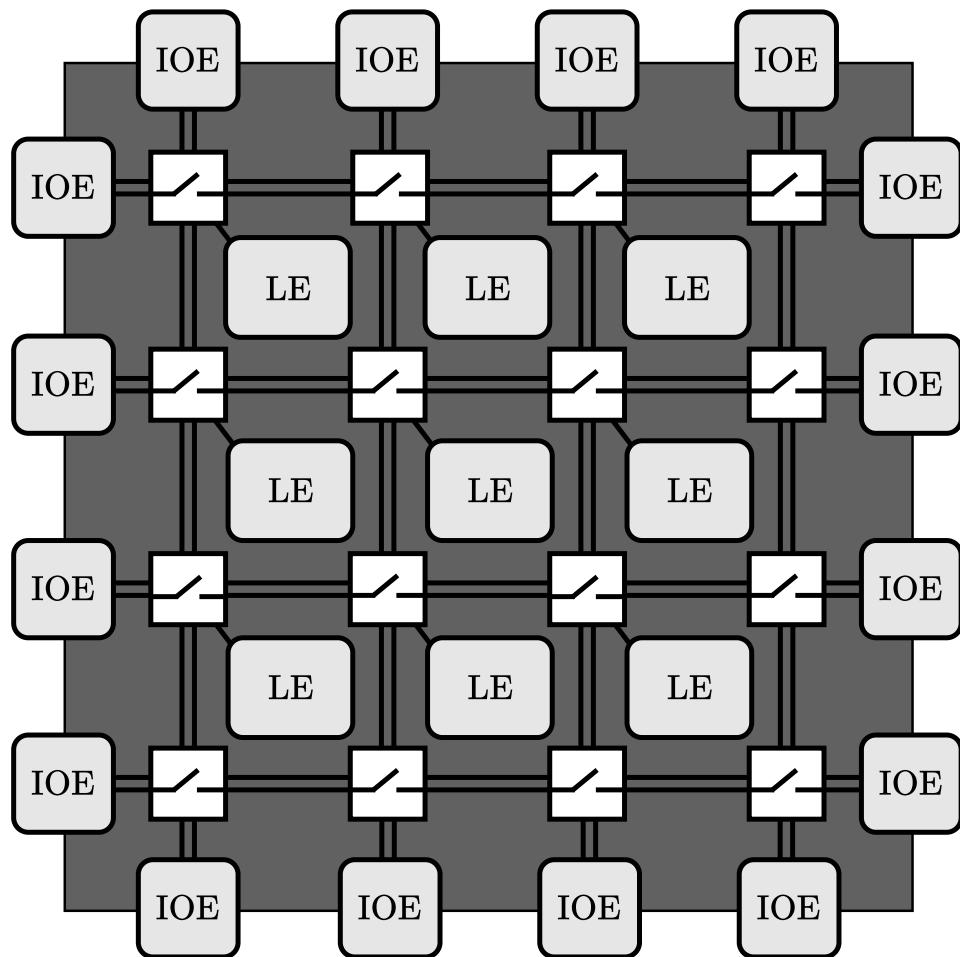


Figura 2.4.1: Arquitectura de una FPGA. Fuente: Adaptado de [11, 12]

Además, FPGAs modernas integran otros bloques funcionales como: multiplicadores, interfaces de entrada/salida de alta velocidad, procesadores de

señales digitales (DSPs), e incluso CPUs [11, 12].

Las características de las FPGAs han despertado gran interés en la investigación de aceleradores hardware aplicados en redes neuronales profundas (DNNs). La naturaleza de las operaciones de las DNNs permite que operaciones fundamentales como multiplicaciones y acumulaciones (MAC) así como operaciones matriciales se ejecuten en paralelo [13]. Además, para aprovechar el uso de recursos de FPGAs, se aplican técnicas como la poda (*prunning*), que consiste en eliminar conexiones (parámetros) no esenciales en el modelo de red neuronal, reduciendo su tamaño y complejidad sin sacrificar significativamente su rendimiento [17], y la cuantización, que convierte los parámetros del modelo de alta precisión a formatos de menor precisión, como enteros, lo que disminuye el uso de memoria y las operaciones requeridas [17]. Estas técnicas permiten que las redes neuronales sean menos complejas y más eficientes en comparación con los modelos tradicionales que no implementan estas estrategias.

Capítulo 3

Materiales y métodos

En este capítulo se exponen los conjuntos de datos utilizados, así como aspectos claves de algunos trabajos previos que realizaron clasificación con ellos.

3.1. Conjuntos de datos y preprocesamiento

Como conjunto de datos se utilizaron colecciones de imágenes pertenecientes a distintas clases, cada uno de estos conjuntos se enfoca en resolver problemas de áreas particulares, siendo cada área diferente de las que abordan los otros conjuntos.

3.1.1. WM-811K

Este conjunto de datos contiene mapas de *wafers*, es decir, imágenes que representan el estado de los discos de silicio utilizados en la fabricación de semiconductores. En total posee 811,457 imágenes, los datos etiquetados pertenecen a 9 clases. Con la intención de aumentar el porcentaje de chips que son fabricados con éxito en un disco de silicio, es fundamental identificar los tipos de fallas que ocurren en producción. Acorde al análisis exploratorio de los datos, 25,519 imágenes se distribuyen entre las 8 clases que corresponden a tipos de fallas, 147,431 son de la clase reservada para discos sin fallas (*none*) y 638,507 no tienen etiqueta.

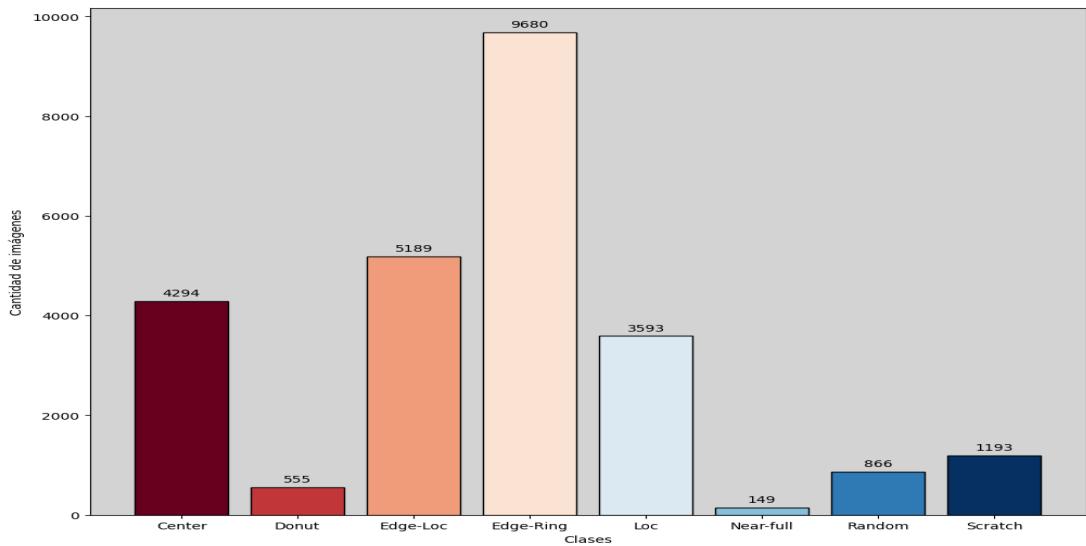


Figura 3.1.1: Distribución de clases de WM-811K. Fuente: Elaboración propia.

Las imágenes fueron preprocesadas, igualando las dimensiones de todas ellas. El tamaño escogido del redimensionamiento se basó en la media aritmética de las dimensiones en x e y de las imágenes etiquetadas en algún tipo de falla, similar a lo realizado en [46], donde emplearon la media ponderada. Todas las imágenes tienen un solo canal.

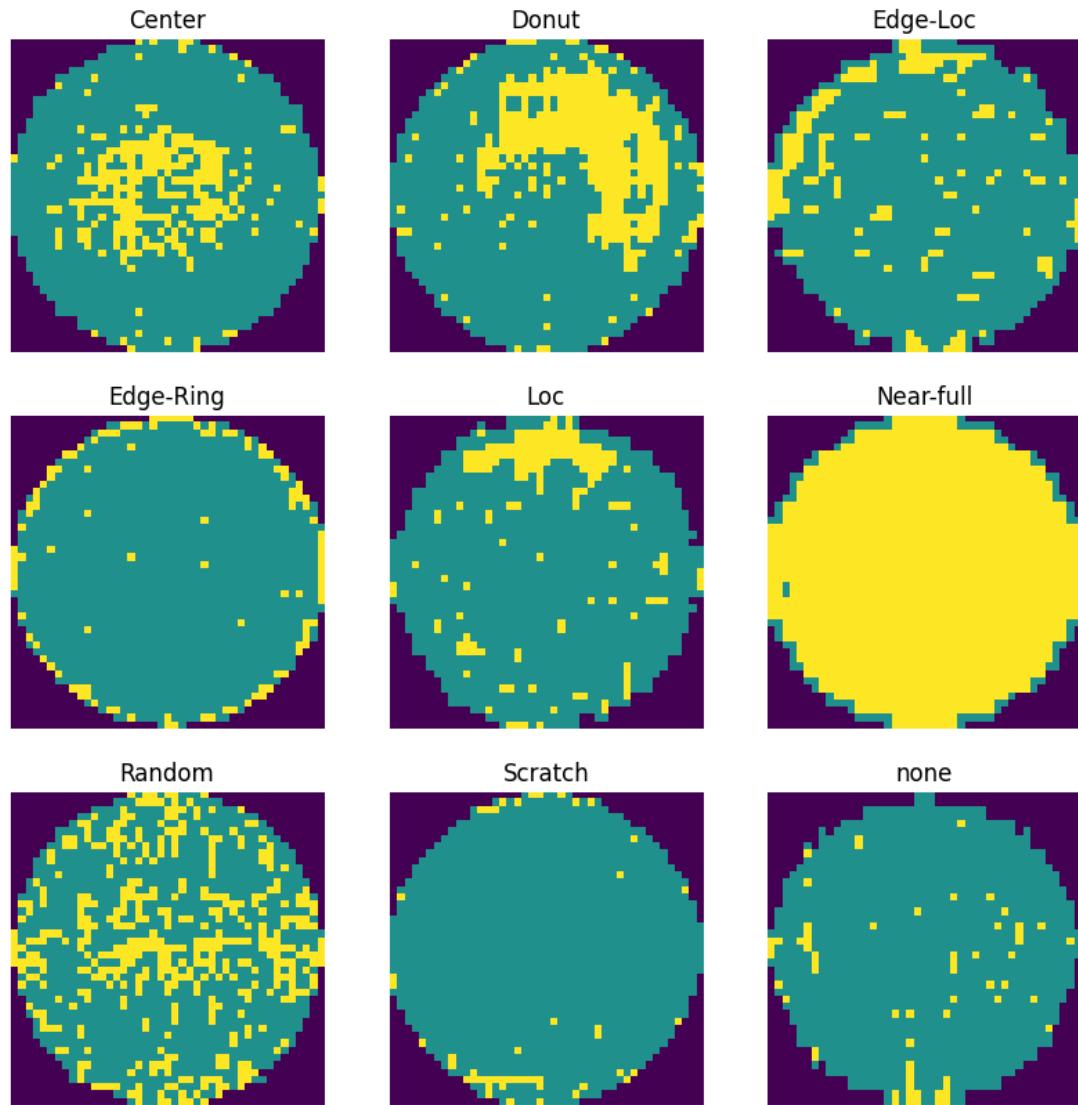


Figura 3.1.2: Clases de WM-811K. Fuente: Elaboración propia.

3.1.2. Brain Tumor Classification (MRI)

Este conjunto de datos reúne imágenes de resonancias magnéticas, que se ordenan en clases de tumores cerebrales: pituitario, glioma, meningioma y no tumor. Contiene un total de 3,264 imágenes distribuidas como muestra la Figura 3.1.3.

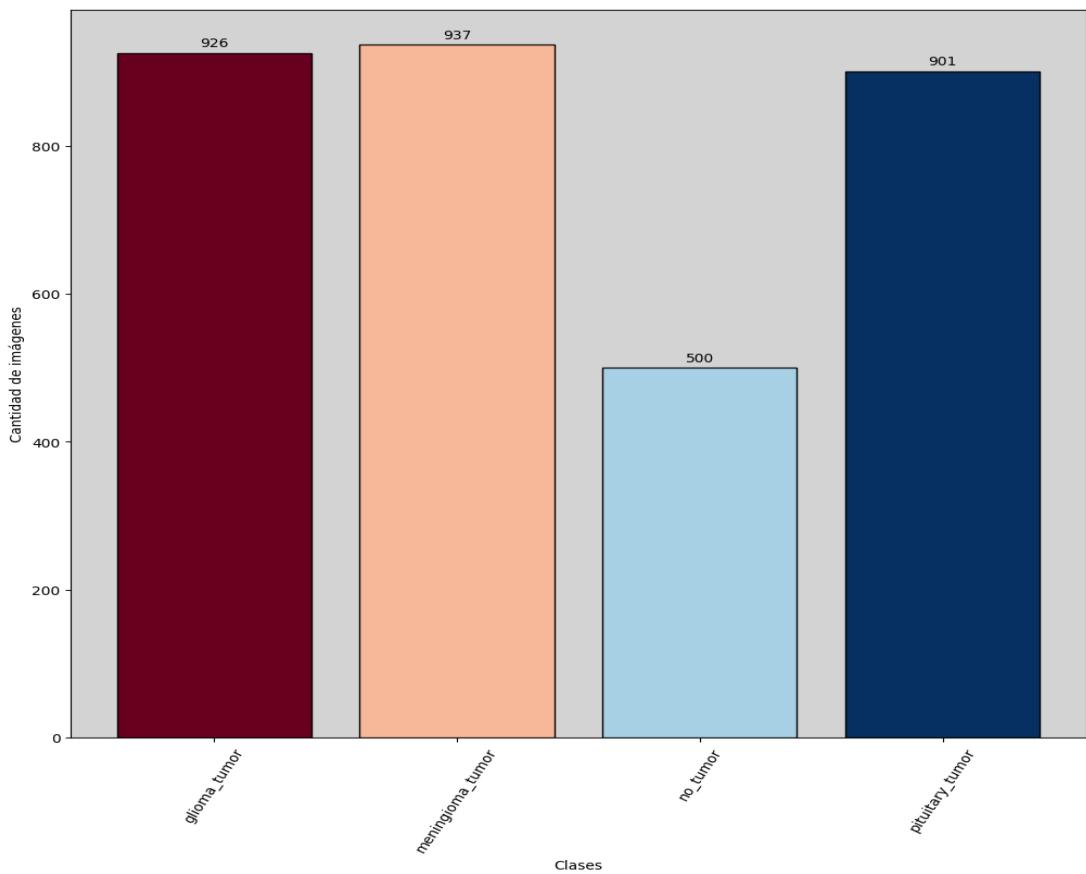


Figura 3.1.3: Distribución de clases de Brain Tumor. Fuente: Elaboración propia.

Originalmente, las imágenes se componen de 3 canales, pero como los colores no aportan información relevante para el problema en cuestión, se transforman a escala de grises (lucen igual). Además, siguiendo la metodología de otro trabajo que utiliza el mismo conjunto de datos, donde se realiza un recorte para incluir en la imagen exclusivamente la zona de interés (quitar márgenes) [47], se realiza un recorte basado en un procedimiento que a diferencia del trabajo mencionado, evita la pérdida de información (imágenes inutilizables) que puede ocurrir debido a un mal cálculo del contorno. Luego, se realiza un redimensionamiento para que todas tengan el mismo tamaño.

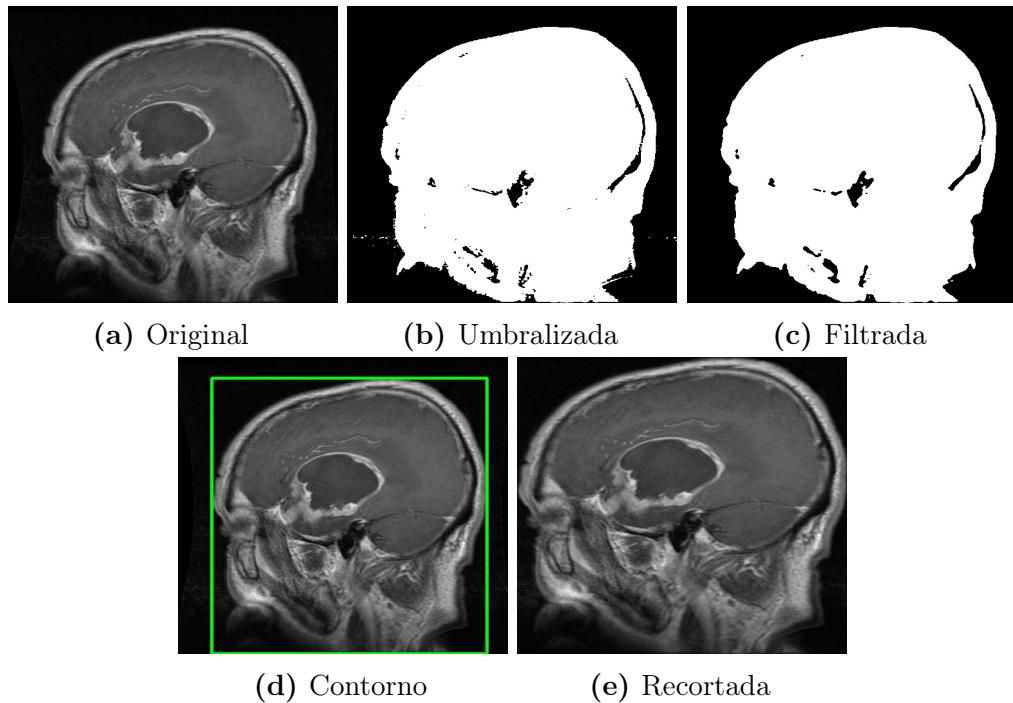


Figura 3.1.4: Procedimiento para recortar una imagen. Fuente: Elaboración propia.

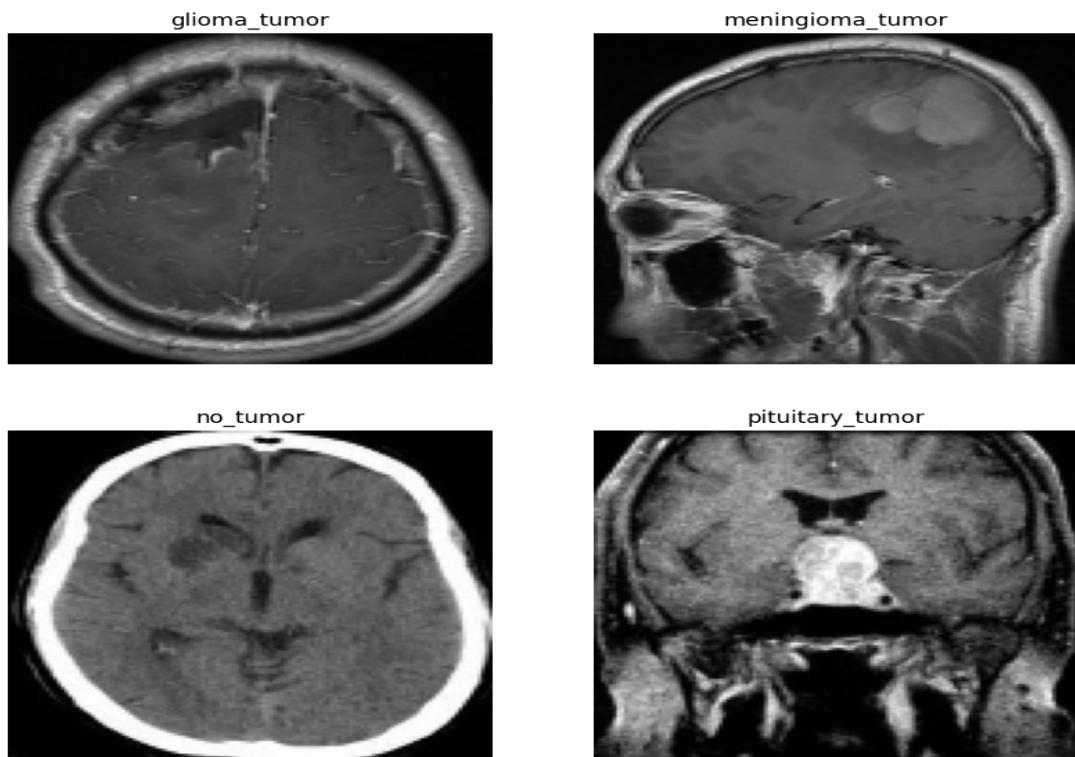


Figura 3.1.5: Clases de Brain Tumor. Fuente: Elaboración propia.

Capítulo 4

Conclusiones y trabajo futuro

La revisión bibliográfica destaca que tanto los clasificadores basados en redes neuronales profundas (DNNs) como los algoritmos de aprendizaje activo son altamente demandantes en términos computacionales. Las operaciones involucradas en el entrenamiento e inferencia de estos clasificadores, así como el re-entrenamiento del modelo en el aprendizaje activo, requieren gestionar grandes volúmenes de datos, especialmente si se trabaja con imágenes. Dado que estas operaciones son inherentemente paralelizables, se establece un entorno ideal para la aceleración mediante hardware basado en FPGA.

Como trabajo futuro, se propone desarrollar y evaluar clasificadores de imágenes basados en redes neuronales profundas utilizando los conjuntos de imágenes explorados. Posteriormente, se implementarán algoritmos de aprendizaje activo en estos clasificadores para optimizar el proceso de etiquetado. Finalmente, se diseñarán y validarán en una FPGA arquitecturas hardware destinadas a acelerar los algoritmos de aprendizaje activo, con el objetivo de realizar una evaluación comparativa entre la implementación en software y la implementación en hardware.

Bibliografía

- [1] M. Elgendi, O. for Higher Education (Firm), and an O'Reilly Media Company. Safari, *Deep Learning for Vision Systems*.
- [2] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of image classification algorithms based on convolutional neural networks," *Remote Sensing 2021, Vol. 13, Page 4712*, vol. 13, p. 4712, 11 2021. [Online]. Available: <https://www.mdpi.com/2072-4292/13/22/4712>
- [3] Y. Lecun, L. E. Bottou, Y. Bengio, and P. H. Abstract|, "Gradient-based learning applied to document recognition," 1998.
- [4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, 6 2017.
- [5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 9 2014. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.
- [7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 12 2015. [Online]. Available: <http://arxiv.org/abs/1512.03385>
- [8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR 2021 - 9th International Conference on Learning Representation*, 10 2021. [Online]. Available: <https://arxiv.org/abs/2010.11929v2>
- [9] A. Tharwat and W. Schenck, "A survey on active learning: State-of-the-art, practical challenges and research directions," *Mathematics 2023, Vol. 11, Page 820*, vol. 11, p. 820, 2 2023. [Online]. Available: <https://www.mdpi.com/2227-7390/11/4/820>

- [10] B. Settles, “Computer sciences department active learning literature survey,” 2009.
- [11] D. G. Bailey, *DESIGN FOR EMBEDDED IMAGE PROCESSING ON FPGAs*, 2023.
- [12] D. M. Harris and S. Harris, *Digital Design and Computer Architecture 2nd edition*, 2012.
- [13] P. Dhilleswararao, G. S. Member, S. Boppu, M. S. Manikandan, S. Member, and L. R. Cenkeramaddi, “Efficient hardware architectures for accelerating deep neural networks: Survey.”
- [14] A. G. Blaiech, K. B. Khalifa, C. Valderrama, M. A. Fernandes, and M. H. Bedoui, “A survey and taxonomy of fpga-based deep learning accelerators,” *Journal of Systems Architecture*, vol. 98, pp. 331–345, 9 2019.
- [15] K. Guo, S. Zeng, J. Yu, Y. U. Wang, H. Yang, and Y. Wang, “A survey of fpga-based neural network accelerator,” *ACM Trans. Reconng. Technol. Syst.*, vol. 9, 12 2017. [Online]. Available: <https://arxiv.org/abs/1712.08934v3>
- [16] J. Y. Kim, “Fpga based neural network accelerators,” *Advances in Computers*, vol. 122, pp. 135–165, 1 2021.
- [17] I. Pérez, M. Figueroa, and S. Perri, “A heterogeneous hardware accelerator for image classification in embedded systems,” *Sensors 2021, Vol. 21, Page 2637*, vol. 21, p. 2637, 4 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/8/2637>
- [18] R. C. Gonzalez and R. E. R. E. Woods, *Digital image processing*, 4th ed.
- [19] G. Kumar and P. K. Bhatia, “A detailed review of feature extraction in image processing systems,” *International Conference on Advanced Computing and Communication Technologies, ACCT*, pp. 5–12, 2014.
- [20] D. G. Lowe, “Object recognition from local scale-invariant features,” 1999.
- [21] H. Bay, T. Tuytelaars, and L. V. Gool, “Lncs 3951 - surf: Speeded up robust features,” 2006.
- [22] T. Ojala, M. Pietikainen, and D. Harwood, “Performance evaluation of texture measures with classification based on kullback discrimination of distributions,” in *Proceedings of 12th International Conference on Pattern Recognition*. IEEE Comput. Soc. Press, 1994, pp. 582–585.
- [23] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. IEEE, 2005, pp. 886–893.
- [24] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001.

- [25] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, “Visual categorization with bags of keypoints.”
- [26] I. Wickramasinghe and H. Kalutarage, “Naive bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation,” *Soft Computing*, vol. 25, pp. 2277–2293, 2 2021. [Online]. Available: <https://link.springer.com/article/10.1007/s00500-020-05297-6>
- [27] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1 1967.
- [28] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, pp. 81–106, 1986.
- [29] by J Ross Quinlan, M. K. Publishers, and S. L. Salzberg, “Programs for machine learning,” vol. 16, pp. 235–240, 1994.
- [30] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, 2001.
- [31] C. Cortes, “Support-vector networks,” pp. 273–297, 1995.
- [32] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,” *Neurocomputing*, vol. 503, pp. 92–108, 9 2022.
- [33] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” 2011.
- [34] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” 2013.
- [35] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” 2016.
- [36] Y. Tian and Y. Zhang, “A comprehensive survey on regularization strategies in machine learning,” *Information Fusion*, vol. 80, pp. 146–166, 2022. [Online]. Available: <https://doi.org/10.1016/j.inffus.2021.11.005>
- [37] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning lecture 6a overview of mini--batch gradient descent,” 2012.
- [38] J. D. JDUCHI and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization * elad hazan,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
- [39] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization.”
- [40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 5999–6009, 6 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762v7>
- [41] T. Scheffer, C. Decomain, and S. Wrobel, “Active hidden markov models for information extraction,” *Lecture Notes in Computer Science*

- (including subseries *Lecture Notes in Artificial Intelligence* and *Lecture Notes in Bioinformatics*), vol. 2189, pp. 309–318, 2001. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-44816-0_31
- [42] B. Settles and M. Craven, “An analysis of active learning strategies for sequence labeling tasks,” 2008.
- [43] H. S. Seung, M. Opper, and H. Sompolinsky, “Query by committee,” *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 287–294, 1992. [Online]. Available: <https://collaborate.princeton.edu/en/publications/query-by-committee>
- [44] S. Berardo, E. Favero, and N. Neto, “Active learning with clustering and unsupervised feature learning,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9091, pp. 281–290, 2015. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-18356-5_25
- [45] M. Wang, F. Min, Z. H. Zhang, and Y. X. Wu, “Active learning through density clustering,” *Expert Systems with Applications*, vol. 85, pp. 305–317, 11 2017.
- [46] J. Gong, “Wafer map failure pattern classification using deep learning,” 2019.
- [47] A. Kadam, S. Bhuvaji, and S. Deshpande, “Brain tumor classification using deep learning algorithms,” *International Journal for Research in Applied Science Engineering Technology (IJRASET)*, vol. 9, pp. 2321–9653, 2021. [Online]. Available: www.ijraset.com417