

INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

***“RECONOCIMIENTO DE LEUCOCITOS EN
IMÁGENES DE MUESTRAS DE SANGRE
PERIFÉRICA”***

TESIS

QUE PARA OBTENER EL GRADO DE

***MAESTRO EN CIENCIAS EN INGENIERÍA
ELECTRÓNICA***

PRESENTA:

ARMANDO REYES ESPARZA

DIRECTOR DE LA TESIS:
DR. MARIO IGNACIO CHACÓN MURGUÍA

CHIHUAHUA, CHIH., AGOSTO 2023



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®





Chihuahua, Chih. 18 de agosto de 2023

**C. ARMANDO REYES ESPARZA
PRESENTE**

En cumplimiento con los requerimientos para la obtención del grado de Maestro en Ciencias en Ingeniería Electrónica y a propuesta de su Comité Tutorial, la División de Estudios de Posgrado e Investigación le concede la autorización para imprimir la tesis titulada **"Reconocimiento de leucocitos en imágenes de muestras de sangre periférica"** dirigida por el Dr. Mario Ignacio Chacón Murguía con el siguiente contenido de capítulos:

- I Antecedentes
- II Marco teórico
- III Base de datos
- IV Diseño de clasificador profundo
- V Búsqueda de arquitectura reducida
- VI Diseño de interfaz gráfica de usuario
- VII Resultados y conclusiones
- VIII Trabajo futuro
- IX Referencias
- X Anexos

ATENTAMENTE

Excelencia en Educación Tecnológica®
"La Técnica por el Engrandecimiento de México"

ROGELIO ENRIQUE BARAY ARANA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

REBA/adcs





Chihuahua, Chih. 8 de agosto de 2023

ROGELIO ENRIQUE BARAY ARANA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
PRESENTE

En cumplimiento con los requerimientos para la obtención del grado de Maestro en Ciencias en Ingeniería Electrónica, le notificamos que el documento de tesis del alumno **C. ARMANDO REYES ESPARZA**, titulado **"Reconocimiento de leucocitos en imágenes de muestras de sangre periférica"** dirigido por el Dr. Mario Ignacio Chacón Murguía, ha sido aprobado y aceptado para su impresión.

Por lo anterior, proponemos le sea concedida la autorización de impresión correspondiente.

Agradeciendo la atención a la presente, quedamos de usted:

ATENTAMENTE

Excelencia en Educación Tecnológica®
"La Técnica por el Engrandecimiento de México"

DR. MARIO IGNACIO CHACÓN MURGUÍA
MIEMBRO DEL COMITÉ TUTORIAL

DR. JUAN ALBERTO RAMÍREZ QUINTANA
MIEMBRO DEL COMITÉ TUTORIAL

DR. ABIMAEEL GUZMÁN PANDO
MIEMBRO DEL COMITÉ TUTORIAL

DR. JESÚS ALFONSO MEDRANO
HERMOSILLO
MIEMBRO DEL COMITÉ TUTORIAL





CARTA CESIÓN DE DERECHOS

En la ciudad de Chihuahua el día 18 de agosto de 2023, el que suscribe, C. ARMANDO REYES ESPARZA con número de control G21061152, de la Maestría en Ciencias en Ingeniería Electrónica, adscrita a la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del el Dr. Mario Ignacio Chacón Murguía y cede los derechos del trabajo titulado "Reconocimiento de leucocitos en imágenes de muestras de sangre periférica", al Tecnológico Nacional de México y/o Instituto Tecnológico de Chihuahua para su difusión, divulgación, transmisión, reproducción, así como su digitalización con fines académicos y de investigación.

C. ARMANDO REYES ESPARZA





DECLARACIÓN DE ORIGINALIDAD

En la ciudad de Chihuahua el día 18 de agosto de 2023, el que suscribe, C. ARMANDO REYES ESPARZA de la Maestría en Ciencias en Ingeniería Electrónica, con número de control G21061152, adscrito a la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua, manifiesta que es autor intelectual de la tesis titulada "Reconocimiento de leucocitos en imágenes de muestras de sangre periférica" bajo la dirección el Dr. Mario Ignacio Chacón Murguía; que el contenido es original y que las fuentes de información consultadas para su fundamentación están debidamente citadas y referenciadas.

C. ARMANDO REYES ESPARZA



Chihuahua, Chihuahua, agosto de 2023

Dra. María Elena Álvarez-Buylla
Directora del CONAHCYT

Sirva la presente para saludarla e informarle que a la fecha he obtenido el grado de **Maestro en Ciencias en Ingeniería Electrónica** en la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua, motivo por el cual agradezco todo el apoyo brindado por este Consejo que usted representa, con el otorgamiento de una beca que permitió dedicarme de manera exclusiva a la realización de mis estudios de posgrado y de esta manera lograr el objetivo principal del convenio establecido.

Sin otro particular quedo a sus órdenes no sin antes reiterar mi agradecimiento.

Atentamente



(Armando Reyes Esparza)
Exbecario CONAHCYT

Agradecimientos

Queridos papá Armando, mamá Hilda, novia Andrea, doctor Mario Chacón y compañeros del laboratorio de percepción visual y robótica,

Es para mí un verdadero honor y placer dirigirme a ustedes en estas líneas para expresar mi profundo agradecimiento por su invaluable apoyo, ayuda y colaboración en la elaboración de mi tesis. No habría sido posible completar este proyecto sin su guía, motivación y paciencia.

A mi papá Armando y mi mamá Hilda, quiero agradecerles por su amor incondicional, su apoyo y su paciencia durante mi proceso de elaboración de esta tesis. Desde el inicio de mi carrera universitaria, siempre han estado ahí para apoyarme, aconsejarme y brindarme todo lo necesario para alcanzar mis metas. En esta ocasión, quiero agradecerles por su comprensión durante los días en los que me ausenté de las reuniones familiares y eventos importantes debido a la carga de trabajo que implicaba mi tesis. Gracias, papá y mamá, por ser mis mejores amigos, por sus consejos y por estar siempre presentes en cada paso que doy.

A ti, mi amada novia Andrea, quiero agradecerte por ser mi compañera de vida y mi fuente constante de motivación y apoyo. Gracias por escucharme cuando necesitaba desahogarme, por motivarme en momentos de desánimo y por celebrar conmigo cada pequeño logro que alcanzaba. Tu amor incondicional y tu presencia han sido mi principal motivación para seguir adelante y alcanzar mis metas.

Al doctor Mario Chacón, mi director de tesis, quiero agradecerle por su dedicación, orientación y conocimiento experto en el campo de mi investigación. Su paciencia y disposición para responder a mis preguntas e inquietudes han sido de gran ayuda para lograr un trabajo de calidad. Gracias, doctor, por compartir su experiencia y conocimientos conmigo, y por ayudarme a crecer académicamente.

A mis compañeros del laboratorio de percepción visual y robótica, quiero agradecerles por su apoyo, colaboración y trabajo en equipo. Sus aportes y discusiones fueron de gran ayuda en la elaboración de mi proyecto, y estoy agradecido por la oportunidad de haber trabajado con un grupo de personas tan talentosas y comprometidas.

En resumen, no hay palabras suficientes para agradecerles por todo lo que han hecho por mí. Espero que esta tesis sea un reflejo de mi gratitud hacia ustedes, y que sea un paso más en el camino de alcanzar nuestras metas y sueños juntos.

Con todo mi amor y agradecimiento,

Armando Reyes Esparza.

RESUMEN

RECONOCIMIENTO DE LEUCOCITOS EN IMÁGENES DE MUESTRAS DE SANGRE PERIFÉRICA

Ing. Armando Reyes Esparza
Maestría en Ciencias en Ingeniería Electrónica
División de Estudios de Posgrado e Investigación del
Instituto Tecnológico de Chihuahua
Chihuahua, Chih. 2023
Director de Tesis: Dr. Mario Ignacio Chacón Murguía

Una metodología para el diagnóstico de enfermedades en humanos es por medio de estudios de sangre. Dentro de los estudios de sangre más comunes existe el hemograma, el cual realiza un conteo de los glóbulos rojos, leucocitos y plaquetas. El hecho de conocer los porcentajes de cada célula ayuda a conocer el tipo de enfermedad que pudiera tener una persona. Específicamente el conteo de leucocitos se denomina leucograma, el cual se puede realizar con un clasificador con un modelo profundo.

Dentro del estado del arte existen distintos trabajos que presentan formas de clasificar leucocitos en imágenes de muestras de sangre periférica con el objetivo de realizar un leucograma. En el trabajo propuesto se utiliza la red AlexNetv2 para proponer un método novedoso de clasificación de leucocitos para una nueva base de datos y que sea competitivo en el estado del arte. Una vez que se propone dicho método, se redujo la arquitectura del método propuesto. Para dicha reducción, se parte de un espacio de búsqueda dado por el modelo AlexNetv2, con el fin de encontrar una arquitectura que continúe resolviendo la tarea de clasificación de leucocitos con un desempeño similar, pero que requiera de menos recursos computacionales para implementarse. Lo anterior con el objetivo de implementar el método propuesto en un sistema embebido y diseñar a futuro un sistema portable capaz de realizar leucogramas con imágenes de muestras de sangre periférica sin presentar un costo tan elevado como es el de los equipos que existen actualmente en el mercado.

ABSTRACT

RECONOCIMIENTO DE LEUCOCITOS EN IMÁGENES DE MUESTRAS DE SANGRE PERIFÉRICA

Ing. Armando Reyes Esparza
Maestría en Ciencias en Ingeniería Electrónica
División de Estudios de Posgrado e Investigación del
Instituto Tecnológico de Chihuahua
Chihuahua, Chih. 2023
Director de Tesis: Dr. Mario Ignacio Chacón Murguía

The blood test is a diagnostic method for humans, and the most common test is a hemogram, which counts the number of red blood cells, leukocytes, and platelets. Knowing the number of cells of each type of cell present in a sample helps to diagnose the disease that a patient could have. More specifically, the count of leukocytes is defined as a leucogram, which a deep learning classifier model can solve.

Within the state of the art exist different developments that present different methods for leukocyte classification in blood smear images with the objective of designing a leukogram. In this work, we use the AlexNetv2 model to propose a novel method for leukocyte classification in a new dataset and with a competitive performance in literature. Then we reduced the architecture of the proposed model by based on a search space established by the AlexNetv2 model with the purpose of finding a new reduced architecture to solve the leukocyte classification with similar performance and less computational resource needs. This model is expected to be implemented in an embedded system that has less computational resources than a computer and future work to develop a portable system able to make leukograms with blood smear images without the high cost of the equipment currently on the market.

CONTENIDO

LISTA DE FIGURAS	ix
LISTA DE TABLAS	x
GLOSARIO	xi
INTRODUCCIÓN	xiii
1 ANTECEDENTES	1
1.1 Conceptos básicos	1
1.1.1 Análisis de muestras de sangre periférica	1
1.1.2 Leucocitos	1
1.1.2.1 Tipos de leucocitos	2
1.1.2.2 Conteo de leucocitos o glóbulos blancos	5
1.1.3 Redes Neuronales Artificiales	5
1.2 Clasificación de leucocitos con aprendizaje de máquina tradicional	6
1.3 Clasificación de leucocitos con <i>deep learning</i>	12
1.4 Búsqueda de arquitectura en modelos profundos	17
2 MARCO TEÓRICO	22
2.1 Imágenes utilizadas	22
2.2 Aumento de datos	26
2.3 Distribución base de datos	29
2.4 Hiperparámetros	30
2.4.1 Optimizadores	30
2.4.2 Función de costo	31
2.4.3 Funciones de activación	31
2.4.4 Razón de aprendizaje	32
2.4.5 Épocas de entrenamiento	32
2.5 Métricas de desempeño	33
2.6 Modelo ResNet50	34
2.7 Modelo VGG16	35
2.8 Modelo AlexNet	35
2.9 Modelo AlexNetv2	36
2.10 Búsqueda de arquitectura	37
3 BASE DE DATOS	39
3.1 LEU_UACHDB	39
3.2 OGDB	43

3.3	TDx6_OGDB	43
3.4	Aumento de datos con GDAM-GAN	44
3.5	PREMASTERDB y MASTER.....	46
3.6	Reetiquetado de base de datos	48
3.7	Nueva base de datos OGDB	48
3.8	Nueva base de datos TDx6_OGDB	49
3.9	Nuevo aumento de datos con GDAM-GAN.....	49
3.10	Nueva base de datos MASTER	50
3.11	MASTER_CHALLENGE	51
4	DISEÑO DE CLASIFICADOR PROFUNDO	53
4.1	Detección.....	53
4.2	Segmentación	54
4.3	Selección de modelos profundos del estado del arte para la clasificación de leucocitos	57
4.4	Implementación de modelos.....	60
4.5	Entrenamiento de modelos para selección de clasificador principal	61
4.6	Segmentación con imágenes reetiquetadas.....	65
4.7	Nuevos entrenamientos de modelos para selección.....	68
4.8	Comparación de desempeño con estado del arte	72
4.9	Implementación de modelo en sistema embebido	73
5	BÚSQUEDA DE ARQUITECTURA REDUCIDA.....	76
5.1	Variación de capas convolucionales y tamaño de capas totalmente conectadas para la búsqueda de la reducción de AlexNetv2.....	76
5.2	Implementación de modelos reducidos en sistema embebido	78
6	DISEÑO DE INTERFAZ GRÁFICA DE USUARIO.....	82
6.1	GUI para computadora	82
6.2	GUI para sistema embebido	85
7	RESULTADOS Y CONCLUSIONES	88
8	TRABAJO FUTURO	90
9	REFERENCIAS	91
10	ANEXOS	96

LISTA DE FIGURAS

Figura 1.1 Representación de un monocito [6].	2
Figura 1.2 Representación de un linfocito [6].	3
Figura 1.3 Representación de un neutrófilo [6].	3
Figura 1.4 Representación de un eosinófilo [6].	4
Figura 1.5 Representación de un basófilo [6].	4
Figura 1.6 Proceso de segmentación [6].	8
Figura 1.7 Análisis de la membrana del leucocito. (a) Imagen original, (b) stretching, (c) detección de bordes con Canny, (d) dilatación y (e) relleno de agujeros [12].	9
Figura 1.8 Análisis del núcleo del leucocito. (a) Imagen original, (b) contrast stretching, (c) filtro pasa bajas, (d) detección de bordes Canny, (e) dilatación de borde y (f) rellenado de agujeros [12].	9
Figura 1.9 Erosión iterativa de núcleo de célula i veces [12].	10
Figura 1.10 Estructura paralela de redes neuronales [12].	10
Figura 1.11 Modelo LeukoX [15].	13
Figura 1.12 Modelo deep learning pre entrenado [16].	13
Figura 1.13 Arquitectura de modelo de red BSA [17].	14
Figura 1.14 Clasificación de leucocitos con CCA basado en CNN y RNN [18].	15
Figura 2.1 Ejemplo de imagen original tomada del microscopio.	22
Figura 2.2 Ejemplo de imagen etiquetada.	23
Figura 2.3 Ejemplo de imagen de GT.	23
Figura 2.4 Ejemplo de imagen de salida de programa de detección.	25
Figura 2.5 Método GDAM-GAN etapa 1 y 2.	27
Figura 2.6 Metodología de aumento de datos.	28
Figura 2.7 Arquitectura de ResNet50.	34
Figura 2.8 Ejemplo de bloque residual de ResNet50.	34
Figura 2.9 Arquitectura de VGG16.	35
Figura 2.10 Arquitectura de AlexNet.	36
Figura 2.11 Arquitectura de AlexNetv2.	37
Figura 3.1 Técnica de frotis de sangre.	39
Figura 3.2 Jerarquía de base de datos LEU_UACHDB.	40
Figura 3.3 Muestra con aumento de magnificación 100X de base de datos LEU_UACHDB sin etiquetar.	41
Figura 3.4 Muestra con aumento de magnificación 100X de base de datos LEU_UACHDB etiquetada.	41
Figura 3.5 Muestra con aumento de magnificación 100X de base de datos LEU_UACHDB ground truth.	42
Figura 3.6 Ejemplos de leucocitos de base de datos OGDB.	43
Figura 3.7 Porcentaje de cantidad de muestras reales y artificiales en base de datos MASTER y MASTER_CHALLENGE.	52
Figura 4.1 Ejemplo de imagen de salida de algoritmo de detección.	54
Figura 4.2 Esquema general del proceso para generar la base de datos de leucocitos segmentados.	56
Figura 4.3 Curvas de aprendizaje de modelo AlexNetv2 entrenado con MASTER para selección de modelo.	64
Figura 4.4 Matriz de confusión de conjunto de validación de modelo AlexNetv2 entrenado con MASTER para selección de modelo.	65
Figura 4.5 Análisis de aumento de datos tradicional en base de datos MASTER.	66
Figura 4.6 Capa de aumento de datos agregada a los modelos profundos.	67
Figura 4.7 Ejemplos de imágenes mal etiquetadas de OGDB.	67
Figura 4.8 Nuevo proceso de selección de modelos.	68
Figura 5.1 Arquitectura reducida del modelo AlexNetv2 experimento #57.	81
Figura 6.1 Apariencia de GUI ITCHLeukoAnalysisV6.0.	83
Figura 6.2 Ejemplo de imagen cargada a GUI.	84
Figura 6.3 Ejemplo de imagen cargada a GUI y modificada por usuario.	84
Figura 6.4 Ejemplo de resultado de GUI.	85
Figura 6.5 Acceso directo a GUI en sistema embebido.	86
Figura 6.6 Ejemplo de clasificación de GUI en sistema embebido.	87

LISTA DE TABLAS

Tabla 2.1 Valores de los canales RGB que definen a cada clase de leucocito.	24
Tabla 2.2 Distribución de base de datos OGDB.	26
Tabla 2.3 Elementos del espacio de búsqueda.	38
Tabla 3.1 Colores de leucocitos de GT.	42
Tabla 3.2 Distribución de base de datos OGDB.	43
Tabla 3.3 Distribución de base de datos TDAX6_OGDB.	44
Tabla 3.4 Cantidad de imágenes generadas por clase con el método GDAM-GAN.	45
Tabla 3.5 Distribución de bases de datos generadas con método GDAM-GAN depuradas.	45
Tabla 3.6 Distribución de base de datos OG_TDAX6_GANDB.	46
Tabla 3.7 Distribución de base de datos PREMASTERDB.	46
Tabla 3.8 Distribución de base de datos MASTER.	47
Tabla 3.9 Distribución de base de datos Raabin.	47
Tabla 3.10 Distribución de reetiquetado de GT.	48
Tabla 3.11 Distribución de nueva base de datos OGDB.	49
Tabla 3.12 Distribución de nueva base de datos TDAX6_OGDB.	49
Tabla 3.13 Distribución de nueva base de datos OG_GANDB.	50
Tabla 3.14 Distribución de nueva base de datos TDAX6_GANDB.	50
Tabla 3.15 Distribución de nueva base de datos OG_TDAX6_GANDB.	50
Tabla 3.16 Distribución de nueva base de datos MASTER.	51
Tabla 4.1 Análisis del estado del arte, selección de modelos.	57
Tabla 4.2 Modelos con mayor desempeño de clasificación en el estado del arte.	60
Tabla 4.3 Hiperparámetros para implementación de modelos.	61
Tabla 4.4 Experimentos para selección de modelo, metodología variación de semilla.	62
Tabla 4.5 Experimentos para selección de modelo ordenados por desempeño de clasificación.	63
Tabla 4.6 Comparativa entre experimento con MASTER y PREMASTERDB.	67
Tabla 4.7 Resultados de variación de semilla.	69
Tabla 4.8 Resultados validación cruzada modelo ResNet50.	70
Tabla 4.9 Resultados validación cruzada modelo VGG16.	70
Tabla 4.10 Resultados validación cruzada modelo AlexNetv2.	70
Tabla 4.11 Promedio de resultados de validación cruzada de modelos AlexNetv2, ResNet50 y VGG16.	71
Tabla 4.12 Resultados experimento muestras GDAM-GAN.	71
Tabla 4.13 Comparación de modelos AlexNetv2 con la literatura.	72
Tabla 4.14 Resultados de validaciones de ONNX.	74
Tabla 4.15 Tiempos de procesamiento de modelos ONNX.	75
Tabla 5.1 Experimentos de búsqueda de arquitectura reducida en PC.	77
Tabla 5.2 Modelos de búsqueda de arquitectura reducida con mejores desempeños de BAM.	79
Tabla 5.3 Tiempos de procesamiento de mejores modelos de búsqueda de arquitectura reducida.	79
Tabla 5.4 Comparación de tiempo de inferencia y arquitectura del modelo.	80
Tabla 5.5 Comparación de tiempos y desempeño entre modelo base de AlexNetv2 y mejor modelo de búsqueda de arquitectura.	80

GLOSARIO

Simbología	Valor
$\mathbb{D}$	Base de datos.
I	Imagen.
$\mathbb{D}_{UACH}$	Base de datos LEU_UACHDB.
I_m	Imagen de base de datos LEU_UACHDB.
I_e	Imagen etiquetada manualmente de base de datos LEU_UACHDB.
GT	Ground truth.
I_d	Imagen de salida de programa de detección de leucocitos.
I_{GT}	Imagen de GT de imagen de base de datos LEU_UACHDB.
LEU_UACHDB	Base de datos con imágenes de muestras de sangre periférica tomadas con un microscopio B-383 originales, etiquetas y ground truths.
f_d	Función programa de detección.
f_s	Función programa de segmentación.
I_s	Imagen segmentada de leucocito obtenida de algoritmo de segmentación.
OGDB	Base de datos con imágenes segmentadas originales reales de leucocitos.
GDAM-GAN	Generative Data Augmentation Method with Generative Adversarial Networks.
TDx6_OGDB	Base de datos con imágenes segmentadas originales reales de leucocitos y aumento de datos tradicional generando 5 imágenes nuevas por cada imagen original.
OG_GANDB	Base de datos con imágenes segmentadas de leucocitos generadas por el método de GDAM-GAN usando las imágenes de OGDB.
TDx6_GANDB	Base de datos con imágenes segmentadas de leucocitos generadas por el método de GDAM-GAN usando las imágenes de TDx6_OGDB.
OG_TDx6_GANDB	Base de datos con imágenes segmentadas de leucocitos de OG_GANDB y TDx6_GANDB.
MASTER	Base de datos con imágenes segmentadas de leucocitos de OGDB y OG_TDx6_GANDB.
Δ_θ	Optimizador.
J	Función de costo.
$f(s)$	Función de activación.
$\hat{s}$	Predicción.
α	Razón de aprendizaje.

E	Cantidad de épocas para entrenamiento de modelo de aprendizaje de máquina.
θ	Parámetros de modelo de aprendizaje de máquina.
CE	Función de costo entropía cruzada.
C	Número de clases.
i	Índice de la clase.
t_i	Etiqueta de la clase.
$\hat{s}$	Predicción de la clase.
α_{din}	Razón de aprendizaje dinámica.
VP	Verdadero positivo.
FN	Falso negativo.
VN	Verdadero negativo.
FP	Falso positivo.
BAM	Métrica de exactitud balanceada.
p	Secuencia 1 de arquitectura AlexNet.
q	Secuencia 2 de arquitectura AlexNet.
$p \parallel q$	Concatenación de secuencias de AlexNet.
O	Espacio de búsqueda.
B	Estrategia de búsqueda.
P	Estrategia para medir desempeño.
$NConv$	Número de capas convolucionales.
NFC	Número de capas totalmente conectadas.
$NNFC$	Número de neuronas de las capas totalmente conectadas.
x	Coordenada x de imagen I.
y	Coordenada y de imagen I.
β	Ángulo de rotación de imagen I.
x'	Nueva coordenada x de imagen rotada.
y'	Nueva coordenada y de imagen rotada.
β_1	Variable de control del rango de decaimiento exponencial.
L	Función de pérdida.
m_t	Actualización del momento.
v_t	Cuadrado del gradiente.
∇Q_t	Gradiente.

INTRODUCCIÓN

Los análisis de sangre son métodos de diagnóstico de enfermedades en humanos, y una forma de análisis de sangre es el hemograma, que consiste en una extracción de la muestra del paciente por venopunción para contar las células contenidas en la sangre. Una técnica para realizar el conteo de sangre es el frotis de sangre, que consiste en colocar una gota de sangre sobre un porta muestras y barrer la gota con otro porta muestras. Posteriormente, se le añade un método de tinción para resaltar las células de la sangre y se coloca la muestra en un microscopio. Por último, un analista revisa la muestra y realiza el conteo de células manualmente, y si se realiza un reporte solo del conteo de leucocitos, se genera un leucograma. Conocer la cantidad de cada clase de leucocitos presentes en la sangre se puede utilizar para diagnosticar enfermedades como infecciones, reacciones alérgicas, inflamación, o cáncer en la sangre. Existen distintos equipos que realizan un hemograma completo, pero estos son costosos. Por ello, en el estado del arte se han propuesto distintas formas de resolver la clasificación de leucocitos utilizando técnicas de aprendizaje de máquina. Por ejemplo, se presentan en la literatura métodos de clasificación de leucocitos con técnicas tradicionales de aprendizaje de máquina y con aprendizaje profundo [11]-[22]. Entre estos, los métodos basados en aprendizaje profundo reportan un mejor desempeño de clasificación y por tanto son el foco de estudio dentro de este trabajo de tesis.

Esta tesis se enfoca en resolver la clasificación de leucocitos utilizando una nueva base de datos llamada LEU_UACHDB, que presenta imágenes de muestra de sangre periférica con un apego mayor a la realidad presentada en un laboratorio de análisis con imágenes procesadas directamente de un microscopio que vería un analista y utilizando un modelo profundo e implementarlo en un sistema embebido. Por lo tanto, se propone una metodología para resolver la tarea de clasificación de leucocitos, definida por:

- Adecuación de la base de datos LEU_UACHDB.
- Selección de modelos del estado del arte.
- Implementación de modelos seleccionados del estado del arte.
- Entrenamiento de modelos para su selección.

- Implementación de modelo con mejor desempeño de clasificación en sistema embebido.
- Búsqueda de arquitectura reducida.
- Diseño de interfaz gráfica de usuario.

Al llevar a cabo los puntos mencionados anteriormente se presenta un método novedoso de clasificación de leucocitos con un desempeño competitivo con el estado del arte y que se implementa con base en AlexNetv2 un modelo profundo de arquitectura reducida con costo computacional menor al del modelo base para reducir los tiempos de procesamiento al momento de implementarse en un sistema embebido.

Este documento se encuentra organizado en ocho capítulos. El capítulo 1 de antecedentes describe los distintos conceptos básicos y el estado del arte de este trabajo de tesis. Luego, el capítulo 2 de marco teórico define matemáticamente los conceptos utilizados, así como los modelos utilizados. El capítulo 3 describe las distintas bases de datos generadas y el capítulo 4 describe en detalle las distintas etapas para diseñar, entrenar e implementar los modelos profundos. La búsqueda de arquitectura reducida se explica en el capítulo 5 y en el capítulo 6 se describe el diseño de la interfaz gráfica de usuario que se crea con el fin de educativo para alumnos que estén aprendiendo sobre tipos de leucocitos. En el capítulo 7 se engloban los resultados obtenidos y las conclusiones del trabajo. Por último, el capítulo 8 describe el trabajo a futuro.

1 ANTECEDENTES

Para entender las bases de esta investigación, es necesario definir los conceptos que se utilizan dentro de ella. La mayoría de estos conceptos están enfocados en el área de la medicina, el análisis de muestras y la inteligencia artificial. Conociendo a fondo cada uno de los conceptos encontrados dentro de estas dos ramas, se creará un entendimiento más profundo de la investigación. Los temas que se abarcan dentro del capítulo son: conceptos básicos, clasificación de leucocitos con aprendizaje de máquina tradicional, clasificación de leucocitos con *deep learning* y búsqueda de arquitectura en modelos profundos.

1.1 Conceptos básicos

1.1.1 Análisis de muestras de sangre periférica

El análisis de muestras de sangre periférica es un trabajo de rutina en un laboratorio de análisis. Normalmente, los pacientes con desórdenes hematológicos deben ser evaluados primeramente con un análisis de muestras de sangre periférica, así como conteo de glóbulos blancos e índices de glóbulos rojos y plaquetas. Realizar estos estudios permite establecer un diagnóstico de posibles enfermedades tales como anemia y alteraciones en leucocitos y plaquetas [1]. El análisis manual de muestras de sangre periférica se realiza por medio de un frotis de sangre y un microscopio para analizar la muestra. Hacer el análisis de forma manual se vuelve un trabajo tedioso que consume mucho tiempo y puede llevar a variaciones de la observación del analista ya que se mantiene observando células para el análisis [2].

1.1.2 Leucocitos

Los leucocitos o también conocidos como glóbulos blancos son componentes móviles de la sangre (células) que tienen un núcleo y citoplasma. Estas células ayudan al cuerpo humano a defenderse de enfermedades infecciosas atacando a las células infecciosas, produciendo anticuerpos o ingiriendo células muertas [3]. La cuenta de leucocitos y la forma de éstos es

sumamente relevante a la hora de realizar un diagnóstico de alguna enfermedad relacionada con la sangre [4].

1.1.2.1 Tipos de leucocitos

Existen 5 tipos de leucocitos, también llamados glóbulos blancos, en la sangre. Se dividen en monocitos, linfocitos, neutrófilos, eosinófilos y basófilos. En una gota de sangre de 1 mililitro se pueden encontrar alrededor de 4,000 a 11,000 leucocitos. La cantidad mencionada se divide aproximadamente en 40% - 70% de neutrófilos, 20% - 45% linfocitos, 2% - 10% monocitos, 1% - 6% eosinófilos y menos del 1% de basófilos [5].

Monocitos

Los monocitos, véase Figura 1.1, rodean y destruyen los microorganismos, ingieren el material extraño en la sangre, eliminan las células muertas y estimulan las respuestas inmunitarias. Son el tipo de leucocito más grande con un citoplasma azul-gris. Tienen un núcleo largo que puede tener varias formas, pero normalmente es como una herradura de caballo. La monocitosis o exceso de monocitos en la sangre es presenciada cuando se tiene tuberculosis, condiciones inflamatorias o hasta leucemia.



Figura 1.1 Representación de un monocito [6].

Linfocitos

Los linfocitos, véase Figura 1.2, elaboran anticuerpos, ayudan a destruir las células tumorales y a controlar las respuestas inmunitarias. Tienen un núcleo redondo grande casi del tamaño del citoplasma y no presentan granulado en la célula. Cuando se tiene un linfocito

inmaduro, el núcleo es prácticamente del tamaño del citoplasma [5]. Un recuento elevado de esta célula resulta en linfocitosis que, aunque es normal en niños menores de 5 años, puede significar una infección o hasta hepatitis (entre otras enfermedades) en personas mayores [7].

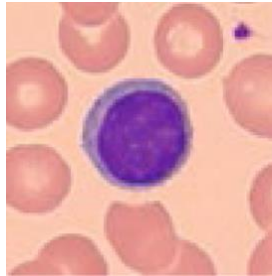


Figura 1.2 Representación de un linfocito [6].

Neutrófilos

Este tipo de leucocitos se encargan de atacar a las sustancias extrañas (bacterias o agentes externos) que entran en el organismo. En situaciones de infección o inflamación su número aumenta en la sangre. Los neutrófilos, véase Figura 1.3, se caracterizan por tener el núcleo segmentado en 2 o hasta 5 lóbulos unidos por un pequeño filamento. El citoplasma es rosa casi descolorido y tiene granulado [8]. Neutrófilos que aún no están maduros pueden variar en su forma y tener un núcleo en forma de banda. Un ejemplo de un desorden de esta célula, es que sí existe hipersegmentación de neutrófilos, esto se podría asociar a deficiencia de hierro en la sangre conocida como anemia o a fallo renal [9].



Figura 1.3 Representación de un neutrófilo [6].

Eosinófilos

Tipo de célula inmunitaria de leucocito que tiene gránulos con enzimas que se liberan durante las infecciones, las reacciones alérgicas y el asma. Los eosinófilos, véase Figura 1.4, son un poco más grandes que un neutrófilo y el núcleo normalmente se compone de dos lóbulos. Una de sus principales características es el granulado del citoplasma es de color naranja-rojizo. La eosinofilia o conteo excesivo de eosinófilos puede ser provocada por una alergia o infección por parásitos [5].



Figura 1.4 Representación de un eosinófilo [6].

Basófilos

Tipo de célula inmunitaria de leucocito que tiene gránulos con enzimas que se liberan durante las reacciones alérgicas y el asma. Los basófilos, véase Figura 1.5, son más pequeños que los neutrófilos y tienen un granulado azulado profundo que hace que se oscurezca el núcleo. La basofilia o exceso de basófilos en la sangre es presenciada en linfomas y leucemia mieloide crónica [5].



Figura 1.5 Representación de un basófilo [6].

1.1.2.2 Conteo de leucocitos o glóbulos blancos

El conteo de leucocitos en muestras de sangre periférica permite detectar la presencia de una posible enfermedad basándose en el número de cada tipo de glóbulo que se encuentre en la muestra [6]. Como se explicaba en la descripción de cada tipo de glóbulo blanco, se pueden llegar a detectar diferentes tipos de enfermedades como tuberculosis, leucemia, infecciones, inflamación, hepatitis, anemia, fallo renal, alergias, linfomas, entre otras; solamente tomando en cuenta el conteo fuera de rango de un tipo de leucocito.

1.1.3 Redes Neuronales Artificiales

Los modelos de redes neuronales artificiales están considerados como una buena herramienta en el ámbito de la medicina [10]. Una red neuronal artificial es un algoritmo que tiene la posibilidad de adecuarse a cualquier tipo de datos de entrada y trabajar con dichos datos para obtener resultados como detección y clasificación de objetos.

El análisis de muestras de sangre periférica se ha vuelto una práctica muy común en laboratorios clínicos [6]. Por ello, los investigadores se han visto motivados a desarrollar algoritmos que automaticen dicho proceso para hacerlo menos tedioso, que consuma menos tiempo y eliminar el posible error humano. Una de las soluciones propuestas por los investigadores fue el uso de redes neuronales para realizar la detección y clasificación en imágenes tomadas de las muestras de sangre periférica.

El análisis de señales, como puede ser una imagen, dentro de una red neuronal para resolver problemas de clasificación, comúnmente consiste de 3 partes: preprocesamiento de la imagen, extracción y selección de características y clasificación con base en patrones obtenidos de dichas características [6].

Existen varias técnicas de preprocesamiento de imágenes, para este caso el preprocesamiento de una imagen consiste en procesar la imagen original de entrada de forma que solo quede a disposición la región deseada para su análisis, en esta aplicación sería solamente la parte de la imagen donde aparezcan los leucocitos.

Una vez preprocesada la imagen de entrada se lleva a cabo la parte de extracción y selección de características. El diseñador escoge un modelo que sea favorable para distinguir características entre los objetos que se presenten en la imagen para que después éstos puedan ser clasificados. En esta etapa se desea que existan características que puedan distinguir a cada tipo de leucocito para poder realizar su clasificación y se obtienen resultados realizando el proceso iterativo de entrenamiento del modelo.

La última etapa para el análisis de imágenes dentro de una red neuronal es la clasificación. Una vez obtenidas las características de la imagen de entrada el algoritmo debe ser capaz de asignar cada objeto a una clase específica, en esta aplicación asignaría a cada tipo de leucocito.

1.2 Clasificación de leucocitos con aprendizaje de máquina tradicional

Existen algunos equipos y computadoras que utilizan distintos algoritmos para realizar el análisis de imágenes de muestras de sangre y realizar el mismo objetivo que un frotis de sangre. Debido a ello algunos investigadores han propuesto algoritmos que sean más eficientes que los utilizados en estos equipos.

El objetivo de clasificación de leucocitos se puede afrontar y puede dar desempeños aceptables con técnicas de aprendizaje de máquina tradicional. A continuación, se describen algunos de los trabajos encontrados en el estado del arte resolviendo el problema de clasificación de leucocitos con la rama tradicional de aprendizaje de máquina.

En [11] se presenta el trabajo de C. G. Varma, et al. quien propone un algoritmo de segmentación y clasificación de leucocitos en muestras de sangre periférica usando K-means clustering y técnicas de Support Vector Machine, SVM, para clasificación de 4 grupos de leucocitos. El proceso propuesto consiste en primera instancia en procesar la imagen de entrada, reajustar su tamaño y segmentar el núcleo de la célula. Después se extraen características y se utilizan para entrenar un clasificador SVM multiclase. La base de datos utilizada es de 366 imágenes de leucocitos divididos en: 60 eosinófilos, 50 basófilos, 82 linfocitos, 94 monocitos y 80 neutrófilos obtenidas del repositorio Kaggle [11]. La captura de

las imágenes de la base de datos fue con un zoom de 100x y con una dimensión de 960x1080 píxeles, formato JPG y se reducen a 256x256 píxeles en MATLAB. Se mejora la calidad de las imágenes con variación en contraste y nitidez. Luego se mejora la nitidez para facilitar la segmentación. Se transforma la imagen de RGB a HIS. Se utiliza K-means clustering para separar el leucocito del fondo de la imagen. El llevar a cabo este pre procesamiento reduce la cantidad de colores y facilita la segmentación.

Para la etapa de extracción de características, se seleccionan el: área, perímetro, centroides, desviación estándar, correlación, root mean square, histograma, promedio aritmético, etc. Después se utiliza el método de PCA para reducir las características y mejorar el tiempo de procesamiento.

En la clasificación se utiliza una EMC-SVM (Ensemble Multi-Class SVM). Se utilizaron 70% de los datos para entrenamiento y 30% para pruebas.

Na Dong, et al., en [6], presenta un algoritmo de clasificación automática de leucocitos con una base de datos de 500 imágenes de resolución 201x201 píxeles. Se diseñó un método adaptivo de ajuste de umbral de segmentación que calcula un umbral apropiado para la segmentación en base al histograma de color de la imagen buscando la adaptabilidad del algoritmo a imágenes de distintos tonos de color. La siguiente etapa es extraer 33 características relacionadas con forma, color y textura de las imágenes segmentadas. Las 10 características más relevantes son seleccionadas con el método de CART. Posteriormente esta selección es ingresada a un clasificador PSO-SVM.

La etapa de segmentación consiste en segmentar primero los leucocitos para diferenciarlos del fondo y después segmentar el núcleo de la célula por medio de umbrales del histograma de la imagen en escala de grises y por HSV, lo que puede determinar de mejor manera la segmentación. El proceso general de segmentación utilizado se observa en la Figura 1.6.

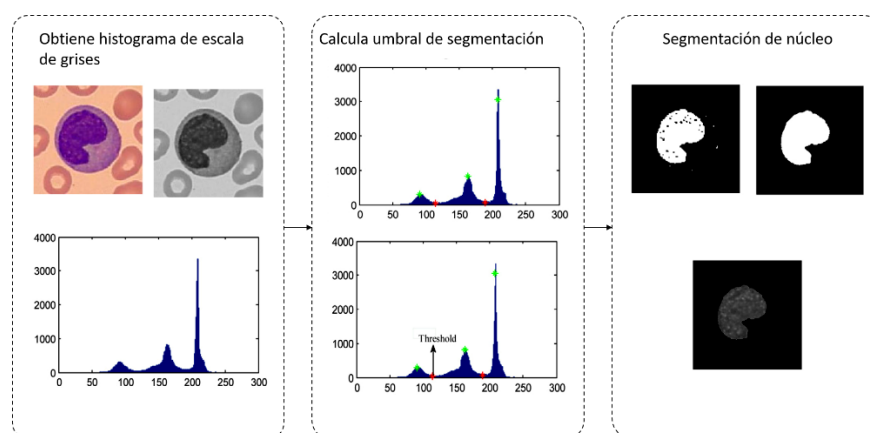


Figura 1.6 Proceso de segmentación [6].

En el enfoque de aprendizaje de máquina tradicional como en [12] se realiza primero la detección y luego la clasificación de los leucocitos. En la etapa de detección, se explica cómo se trabaja de igual manera con imágenes en escalas de grises para disminuir la posibilidad de cambios a la hora de extraer características que pueden darse gracias a los cambios de coloración, esto debido a los distintos tipos de colorantes que se utilizan en la muestra de sangre. Para diferenciar los leucocitos se toma en cuenta que el colorante que se usa en la preparación de la muestra de sangre, tiende a concentrarse más en los leucocitos que en otra parte de la muestra, por lo que los leucocitos se verán como figuras más oscuras. Los glóbulos rojos son más pequeños y su núcleo aparece más pálido que la membrana. Las plaquetas son mucho más pequeñas que los glóbulos. Gracias a estas características de las células se utiliza un prefiltro adaptivo para reconocer solo los leucocitos de la imagen de la muestra disminuyendo la presencia de los glóbulos rojos y las plaquetas. Después, estos resultados son segmentados para luego entrar a un clasificador.

Una vez segmentada la imagen del leucocito se procede a analizar la forma de su membrana y núcleo. Para analizar la forma de la membrana de la Figura 1.7 (a), primero se hace un stretching de la misma (b), luego una detección de bordes con Canny (c), después se aplica un operador de dilatación (d) y por último un relleno de agujeros (e).

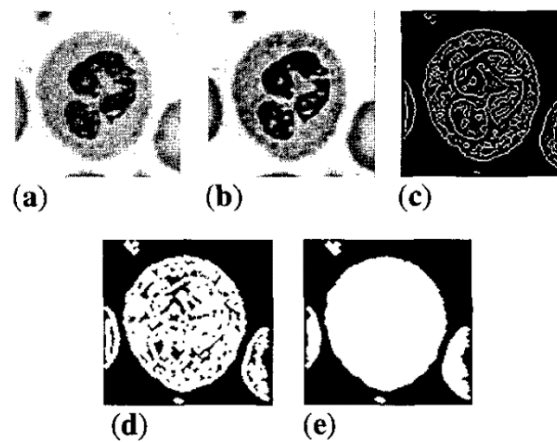


Figura 1.7 Análisis de la membrana del leucocito. (a) Imagen original, (b) stretching, (c) detección de bordes con Canny, (d) dilatación y (e) relleno de agujeros [12].

Posteriormente, para el análisis del núcleo de la Figura 1.8 (a) es necesario utilizar el análisis de la membrana con contrast stretching (b), después un filtro pasa bajas (c) que muestre solo el núcleo del leucocito, detección de bordes con Canny (d), dilatación de bordes (e) y relleno de agujeros (f).

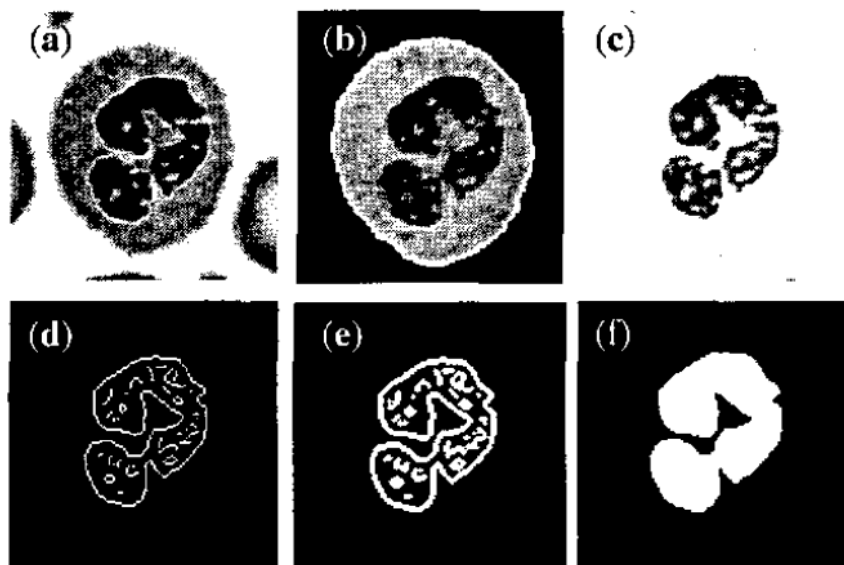


Figura 1.8 Análisis del núcleo del leucocito. (a) Imagen original, (b) contrast stretching, (c) filtro pasa bajas, (d) detección de bordes Canny, (e) dilatación de borde y (f) relleno de agujeros [12].

Para la extracción de características en este modelo se toman 23 características distintas, entre las cuales destaca el número de penínsulas que conforma a cada núcleo de leucocito.

Para extraer dicho número, se hace un análisis iterativo de erosión al núcleo de la célula para saber cuántas penínsulas lo conforman. El hecho de conocer el número de penínsulas es un factor esencial al momento de clasificar los leucocitos. En la Figura 1.9, se puede observar un ejemplo del proceso de erosión iterativo para detección de penínsulas con 1 iteración (a), 7 iteraciones (b), 10 iteraciones (c) y 12 iteraciones (d).

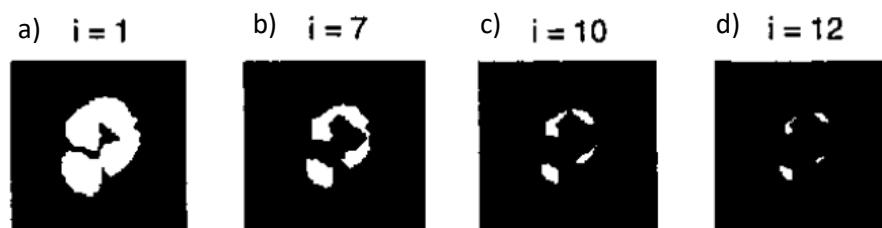


Figura 1.9 Erosión iterativa de núcleo de célula i veces [12].

Una vez analizadas las 23 características por *forward selection* se encontró que la mejor clasificación estaba dada por la intensidad del citoplasma, el área del núcleo y el área de la célula. El clasificador utilizado consta de una estructura de 5 redes neuronales que clasifican a cada clase de leucocito, luego la salida de todas las clases se compara para ver cuál fue la más alta en porcentaje y esa es la seleccionada como clasificación correcta. En la Figura 1.10 se puede observar cómo está constituida la estructura de las 5 redes neuronales implementadas en [12].

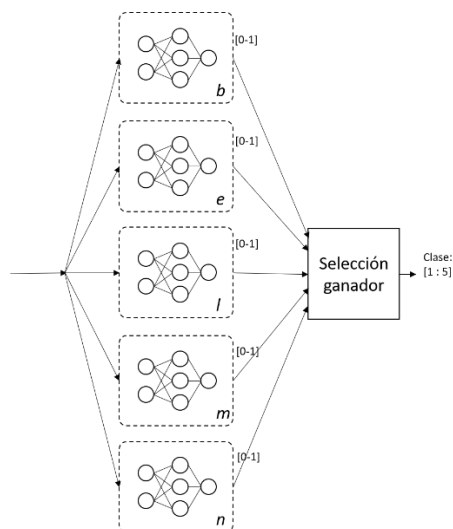


Figura 1.10 Estructura paralela de redes neuronales [12].

Dentro de aprendizaje de máquina tradicional también se busca realizar enfoques novedosos con técnicas de otras áreas. En [13] se propone un algoritmo basado en *Binary Bat Algorithm* (BBA) para clasificación de leucocitos. Dentro de este se extraen características de las imágenes de leucocitos y posteriormente se utiliza el BBA modificado para que seleccione solo las características esenciales extraídas de las imágenes. El BBA es un algoritmo inspirado en la técnica de geolocalización que utilizan los murciélagos para localizar objetos y presas. Al usar dicho algoritmo se reduce la dimensionalidad de las características. Se implementaron 4 distintos clasificadores: K-Nearest Neighbors (KNN), *logistic regression*, *random forest* y *decision tree* para comparar sus resultados y seleccionar el mejor. La base de datos utilizada para el entrenamiento y validación es LISC realizada por *Hematology-Oncology* y *BMT Research Center* de hospital Imam Khomeini en Irán, con una relación 80:20 de entrenamiento y validación respectivamente. Esta base de datos consiste en 237 imágenes de 8 sujetos en condiciones de salud normales de las 5 clases de leucocitos.

Dentro del estado del arte también se busca presentar algoritmos que puedan ser aplicados a proyectos futuros, como lo es en [14]. Se propone un *framework* con recursos asistidos por la nube para utilizar en celulares para reconocimiento de leucocitos en imágenes de muestras de sangre periférica utilizando un clasificador SVM multiclase. Dentro del proceso del *framework* realizado en la nube se segmenta el núcleo de la célula, se procesa la imagen para formar el citoplasma de la célula, se extraen características de textura, probabilísticas y de Wavelet, y por último se realiza la clasificación en las 5 clases de leucocitos. Se maneja una base de datos con 1030 imágenes del complejo de medicina Hayatabad en Pakistán. La sección del *framework* interesante para este trabajo de tesis es la segmentación y clasificación de leucocitos. Esta consta de 6 etapas: ajuste de contraste de la imagen de entrada, segmentación de la región de interés (núcleo), aplicación de transformada Wavelet, extracción de características de textura, probabilísticas y de *Wavelet*, análisis de componentes principales para reducir el espacio de características y clasificación con un clasificador SVM multiclase.

Al utilizar modelos de aprendizaje de máquina tradicional, resulta en el requerir una menor capacidad computacional que al utilizar modelos profundos. Y en ocasiones puede ser suficiente para resolver la tarea en cuestión. Por ello, es necesario realizar un análisis para

primero buscar resolver una tarea de manera sencilla antes de pasar a una forma compleja de solución.

1.3 Clasificación de leucocitos con *deep learning*

Otra forma de resolver la clasificación de leucocitos con desempeños aceptables es por medio de técnicas de *deep learning*. El hecho de seleccionar técnicas de *deep learning* como una forma de clasificación de leucocitos puede hacer más complejo el diseño del algoritmo dados los conocimientos previos que se deben conocer, pero en ocasiones resultan mejores los resultados que los obtenidos con técnicas de aprendizaje de máquina tradicional y resulta más sencillo el trabajo debido a que la etapa de extracción y selección de características se lleva de manera automática con el entrenamiento del modelo. A continuación, se presentan algunos trabajos realizados con el enfoque de *deep learning*.

En [15], Ghosh, et al. propone un algoritmo de *ensemble learning* aplicado para dos modelos de redes profundas: DenseNet21 y ResNet50, junto con una red diseñada CNN para la clasificación de 4 tipos de leucocitos denominado LeukoX. Se utiliza un Least Entropy Combiner (LEC) para combinar las clasificaciones de las redes, disminuir el costo de entropía y tener resultados más confiables. Se utilizó una base de datos de acceso libre en GitHub por Shenggan [15] que cuenta con 12,500 imágenes de leucocitos tipo: eosinófilo, linfocito, monocito y neutrófilo con 3,120 imágenes de cada uno aproximadamente. El tamaño original de las imágenes era de una resolución de 320x240 pixeles, pero se redujo a 128x128 pixeles para aumentar la velocidad del entrenamiento y se dividieron las imágenes de la base de datos en 80% de las imágenes para entrenamiento, 8% para validación y 12% para pruebas.

La implementación de un LEC conlleva ventajas que un solo modelo implementado no podría generar. Una de ellas es que en un solo modelo se podría hacer el análisis de los datos y en ocasiones los valores obtenidos podrían generar problemas o falsos. En cambio, para un LEC que contiene varios modelos combinados, puede ser una ventaja para alguno de los modelos utilizados dentro del LEC, al hacer la combinación de los datos analizados. También se genera más diversidad de datos a la salida gracias a los distintos enfoques de los pesos de los modelos y se reduce la convergencia que el algoritmo termine sobre entrenado. Al hacer

un procedimiento combinando varios modelos con LEC, resulta menos probable que se realice una clasificación errónea. En la Figura 1.11 se puede observar el modelo LeukoX diseñado en [15] con un Least Entropy Combiner.

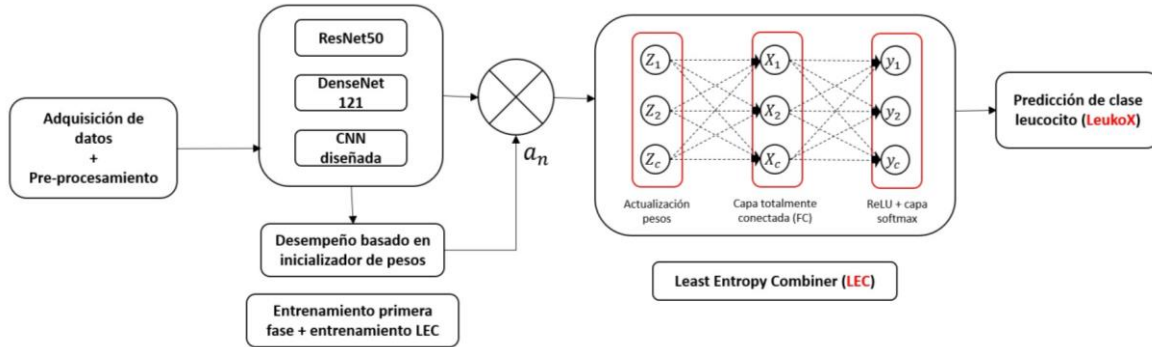


Figura 1.11 Modelo LeukoX [15].

Otro enfoque por técnicas de deep learning, se realiza en [16], para clasificar los 5 tipos de leucocitos utilizando dos redes pre entrenadas, donde una es utilizada para la segmentación de la célula (DeepLabv3+) y otra para la clasificación de los 5 tipos de leucocitos (AlexNet). Se utiliza una base de datos con imágenes de 720x576 pixeles con zoom 100x en formato bmp. Ésta contiene 53 imágenes de basófilos, 50 imágenes de neutrófilos, 39 imágenes de eosinófilos, 52 de linfocitos y 48 de monocitos. Dicha base de datos fue adecuada a una base de datos con las imágenes de las células recortadas con ayuda de la red de segmentación para poder usarlas en la clasificación. El hecho de utilizar redes pre entrenadas hace que se necesiten menos datos para poder refinar el modelo. En la Figura 1.12 se observa el modelo de deep learning diseñado con las dos redes pre entrenadas para la segmentación y clasificación de 5 tipos de leucocitos.

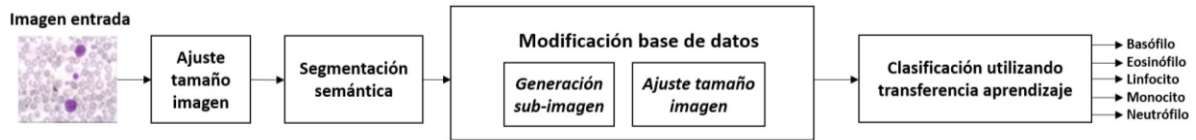


Figura 1.12 Modelo *deep learning* pre entrenado [16].

En [17] se propone una metodología para identificar leucocitos en 4 clases y consiste en una novel bilinear self-attention network (BSA Net). Este modelo utiliza un mecanismo que enfoca la atención en el objeto de interés y reduce el enfoque del fondo, para después utilizar

una estrategia de entrenamiento bilineal que se encarga de encontrar la relación que existe entre las distintas características del objeto. El mecanismo de atención realiza la función de enfocarse en las características que representan al núcleo de los leucocitos y desprecia las características encontradas en la parte restante de la imagen de muestra de sangre. La estrategia de entrenamiento bilineal o Bilinear training strategy es una arquitectura de identificación como la mostrada en la Figura 1.13. Esta consiste en dos extractores de características cuyas salidas son combinadas para poder obtener en conjunto un descriptor de la imagen, que se utiliza para clasificaciones que necesiten distinción fina entre variaciones de sus características.

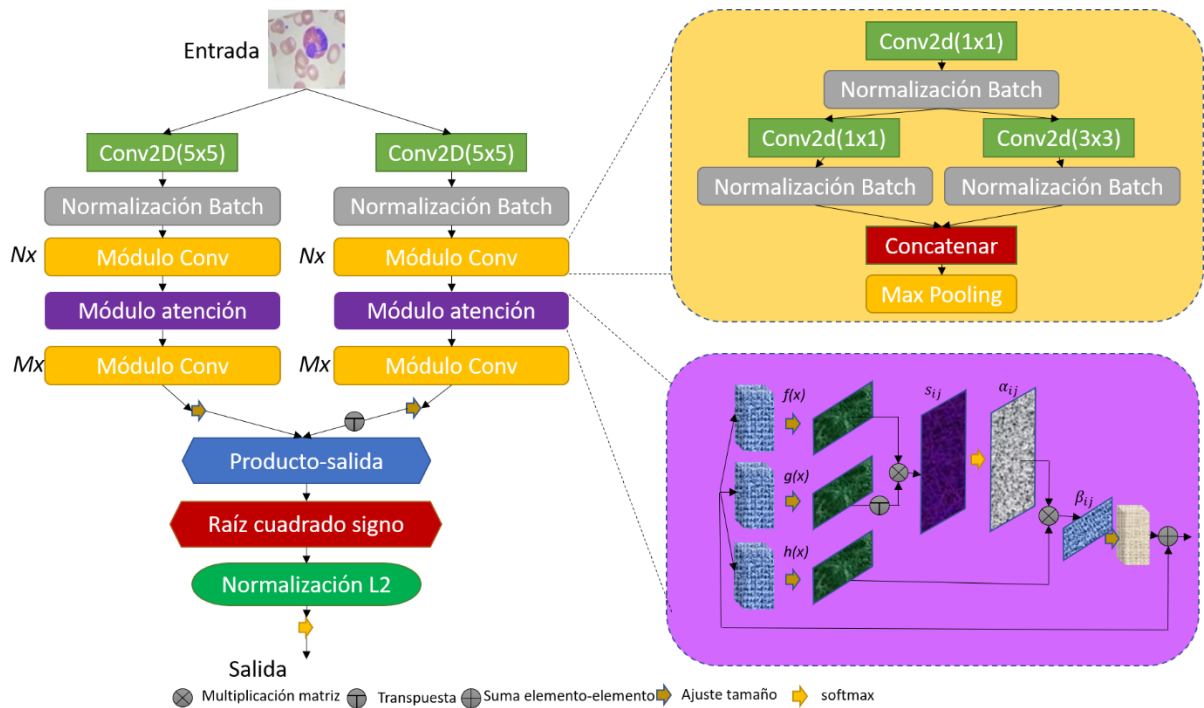


Figura 1.13 Arquitectura de modelo de red BSA [17].

El trabajo realizado en [18] se basa en la clasificación de cuatro leucocitos utilizando el método de análisis canónico de correlación. Dicho método de clasificación provee una salida que es aplicada a dos enfoques de red. El proceso de los dos enfoques consiste en interpretar simultáneamente las imágenes por medio de una red neuronal convolucional y una red neuronal recurrente. Las salidas de ambos enfoques son unidas para obtener una salida en

común que es pasada a una capa softmax para dar un resultado final de la clasificación, véase Figura 1.14.

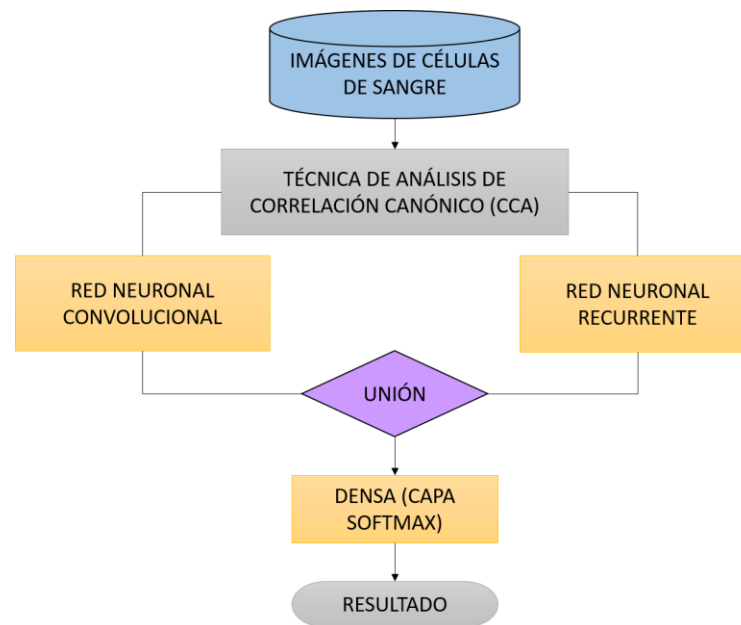


Figura 1.14 Clasificación de leucocitos con CCA basado en CNN y RNN [18].

En [19] se propone un método de Attention-aware Residual Network based Manifold Learning model (ARML) para clasificar 6 tipos de leucocitos: basófilos, monocitos, eosinófilos, linfocitos, neutrófilos de banda y neutrófilos segmentados. Se utilizaron 9000 imágenes de leucocitos para el entrenamiento y 1800 para validación. El clasificador obtuvo un rendimiento de clasificación de 95.30%. El algoritmo consiste en primera parte de realizar un recorte de las células leucocitos con una red pre entrenada RetinaNet. Posteriormente se procesa la imagen por bloques residuales que constan de capas convolucionales para la extracción de características los cuales en conjunto son llamados attentional residual unit y extraen un mapa de características. El modelo utiliza 3 de estas unidades puestas en cascada para tener 3 niveles de características. Luego se juntan esas características extraídas en un vector para ingresar a una matriz de covarianza para capturar información más precisa de las características. Utilizando Gaussian embedding y Manifold Learning se compactan las características para entrar a un clasificador de leucocitos con base en los valores de las características extraídas y concluye el modelo.

Hegde, et al. [20] realiza un estudio comparativo extrayendo características desde dos enfoques distintos: entre características extraídas utilizando procesamiento de imágenes tradicional y extracción de características con un modelo profundo. Para la clasificación utiliza una red neuronal propia. En el estudio se demuestra que es un poco mejor utilizar un modelo profundo para la extracción de características que un modelo de procesamiento de imágenes tradicional en la base de datos que obtuvieron los autores. Se reporta un promedio de exactitud y sensibilidad del 99% para la clasificación de 6 clases de leucocitos (basófilos, eosinófilos, linfocitos, monocitos, neutrófilos y leucocitos anormales). La red neuronal utilizada como clasificador consta de dos etapas de procesamiento, la primera para identificar si se trata de un leucocito anormal o normal, y la segunda para clasificar entre los 5 tipos de leucocitos normales.

En [21] los autores presentan un modelo híbrido de aprendizaje profundo con los modelos GoogleNet y AlexNet como extractores de características y un SVM como clasificador. Se utilizan ambas características extraídas de los modelos profundos y se unen en un solo vector de características que entra al SVM para llevar a cabo la clasificación de 4 tipos de leucocitos. Se comparan sus resultados con los modelos GoogleNet y AlexNet pre entrenados y se obtiene una mayor exactitud con el modelo híbrido propuesto. Al utilizar las bases de datos LISC y de Kaggle se obtienen desempeños de exactitud y métrica F1 de 99.73% y 0.99 respectivamente para la base de datos Kaggle y 98.23% y 0.98 para la base de datos LISC.

Spatial and Channel Attention Module (SCAM) se describe en [22] con el fin de clasificar leucocitos. Usualmente en las muestras de leucocitos se presenta una alta similitud entre las distintas clases y un conjunto de datos con muestras no balanceadas. Para resolver dichos retos se propone un *framework* con la unión de las redes ResNet y DenseNet pre entrenadas con SCAM. El módulo de SCAM analiza dos distintas dimensiones de espacio y elimina la información irrelevante al igual que enaltece la información importante para solucionar la limitante de similitud entre las clases de leucocitos. Se compara el *framework* propuesto con distintos módulos de atención en donde obtiene un mayor desempeño SCAM. De igual forma se compara SCAM con modelos profundos como ResNet, DenseNet, InceptionV3 utilizados como clasificadores.

El realizar un estudio de los trabajos presentados en la literatura se brinda un panorama de las soluciones aplicadas a la problemática abordada en esta tesis sobre la clasificación de leucocitos en imágenes de muestras de sangre periférica. Conociendo las metodologías utilizadas, las bases de datos y los resultados obtenidos, se puede tener un punto de partida para la búsqueda de la solución al problema de clasificación. Así como entender los detalles que con lleva trabajar con imágenes biológicas. Se partirá del análisis de la información presentada en este capítulo para plantear una solución que esté a la vanguardia de los modelos presentes en la actualidad.

1.4 Búsqueda de arquitectura en modelos profundos

Los modelos profundos en aprendizaje de máquina son una herramienta muy útil a la hora de resolver tareas complejas. Sin embargo, estos llevan un proceso que requiere de un conocimiento previo para llevar a cabo el entrenamiento del modelo y seleccionar una arquitectura que se adapte mejor a la problemática a resolver para llegar a obtener mejores resultados. El diseño de un modelo profundo es entonces un proceso tardado y hasta el momento no existe una metodología definida para llevarlo a cabo. Para buscar resolver esto, Zoph, et al. en 2017 presenta en su trabajo [23] un método que diseña arquitecturas de modelos desde cero utilizando aprendizaje de refuerzo para resolver la base de datos CIFAR-10 [24]. De esta forma, Zoph presenta por primera vez el concepto de *Neural Architecture Search* (NAS) el cual consiste en buscar de forma automática una arquitectura que permita resolver con un desempeño más alto una tarea. Este concepto consiste de manera general en un espacio de búsqueda en donde se tendrán los distintos valores entre los cuales se puede variar la arquitectura, después una estrategia de búsqueda para definir como se manejarán los elementos encontrados dentro del espacio de búsqueda y por último una estrategia para medir el desempeño de la arquitectura que se evalúe en el momento [25].

Dentro de la literatura se presentan distintos trabajos que utilizan el concepto de NAS, pero realmente entre ellos existen demasiadas variaciones tanto en el espacio de búsqueda, como en la estrategia de búsqueda y la forma de medir el desempeño, ya que cada trabajo busca encontrar algo distinto. Por ello en esta sección se busca presentar algunos trabajos que

utilicen el concepto de NAS y plantear así las distintas variaciones que estos presentan para considerarlas y plantear después una metodología de búsqueda de arquitectura que ayuda a resolver la tarea de clasificación de leucocitos con un modelo profundo.

Franchini, et al. en [26] indica como una de las etapas más tediosas de los modelos profundos es el proceso de entrenamiento y que el éxito de un buen entrenamiento se denota por los hiper parámetros seleccionados para este. Franchini propone un estudio para realizar un análisis sobre los hiper parámetros relacionados con la arquitectura de una red neuronal convolucional como el número de filtros, el tamaño del filtro y el optimizador a utilizar. Se propone una técnica automática de aprendizaje de máquina para seleccionar distintas variables de los hiper parámetros anteriormente mencionados para obtener el mejor desempeño posible. En la estrategia propuesta miden el desempeño de las variantes del modelo en las primeras épocas del entrenamiento para que sea más rápida la selección de los hiper parámetros finales. El modelo a entrenar busca resolver la tarea de eliminar artefactos de imágenes que son considerados como ruido, en esta ocasión consideran solamente rayas horizontales. Se utiliza la base de datos MNIST [27] como base del experimento que contiene imágenes de números en escala de grises con una resolución espacial de 28 x 28 pixeles. Franchini define el espacio de búsqueda de 5 hiper parámetros: largo del paso, tamaño del *mini-batch*, optimizador, número de filtros y tamaño de los filtros. Se realizan pruebas de las distintas combinaciones de los hiper parámetros por un máximo de 5 épocas evaluando el promedio del error absoluto (MAE). Al final se muestra cómo se resuelve el eliminar los artefactos de las imágenes con un MAE del 0.0063.

En [28] Zhang, et al. explica su trabajo con el fin de realizar una búsqueda de arquitectura para resolver la clasificación de imágenes con escenas remotas. Se utilizan cuatro bases de datos para realizar el entrenamiento: AID [29], NWPURESISC [30], RSI-CB [31] y *PatternNet* [32]. Utilizando las cuatro bases de datos propone un método que permite predecir el comportamiento de una red neuronal de acuerdo con la configuración de sus hiperparámetros. Se consideran dentro del espacio de búsqueda los hiperparámetros: convolución separable 3x3, convolución separable 5x5, convolución separable dilatada 3x3, convolución separable dilatada 5x5, *max pooling* 3x3, *average pooling* 3x3, identidad y

ninguna. Al realizar la variación de los hiperparámetros en la arquitectura propuesta se llegan a resultados mejores de exactitud de clasificación con hasta el 99.52% en cualquiera de las 4 bases de datos en comparación con modelos comunes de aprendizaje de máquina como VGG-16, ResNet50, GoogleNet, entre otros. El entrenamiento de la búsqueda de arquitecturas tiene una duración de 50 épocas en cada combinación de hiperparámetros, y luego que se selecciona la que tiene un más alto desempeño de exactitud se realiza otra fase de entrenamiento con 500 épocas para la verificación de dicho modelo.

Lyu, et al. en [33] presenta una metodología para búsqueda de arquitectura basada en la arquitectura de MobileNetv2 con el objetivo de implementar un modelo que requiera menor capacidad computacional para dispositivos con bajos recursos. Basa el entrenamiento del modelo en la base de datos *ImageNet* [34] para clasificación de 1000 clases y aprendizaje por refuerzo. Se presenta un espacio de búsqueda que varía el tamaño del filtro y el radio de expansión de la etapa de cuello de botella del modelo MobileNetv2. Las variaciones que plantean son: convolución 3x3 radio 3, convolución 3x3 radio 6, convolución 5x5 radio 3, convolución 5x5 radio 6, convolución 7x7 radio 3, convolución 7x7 radio 6 y sin convolución. Al variar en estos valores el ancho de cuello de botella permite una reducción en la cantidad de parámetros del modelo, lo que lo hace más susceptible a su implementación en un hardware que requiera menor capacidad computacional. Al comparar la arquitectura de MobileNetv2 encontrada, esta supera con una exactitud de clasificación Top 5 del 91.6% a modelos como MobileNetv1, MobileNetv2, NASNet, MnasNet y ProxylessNAS-R. Con estos resultados demuestran que no es necesario contar con un modelo con una gran cantidad de parámetros para llegar a obtener altos resultados de clasificación, ya que reducen el modelo MobileNetv2 de 3.4 millones de parámetros a 1.1 millones y mejoran el desempeño de clasificación.

AutoFCL es descrito en [35]. Consiste en un modelo que aprende la mejor estructura de las capas totalmente conectadas de una red neuronal convolucional utilizando un optimizador bayesiano para resolver la clasificación de tres bases de datos distintas: CalTech-101 [36], Oxford-102 Flowers [37] y UC Merced Land Use [38]. El espacio de búsqueda propuesto consta de la variación del número de capas totalmente conectadas, las funciones de

activación, el rango de *dropout* y el número de neuronas de las capas totalmente conectadas. Se utilizan cinco arquitecturas base (VGG16, ResNet50, MobileNetv1, DenseNet121 y NASNetMobile) para llevar a cabo la búsqueda de arquitectura con el método propuesto y se aplican a las tres bases de datos. Al comparar con otros trabajos que realizaron la clasificación de las bases de datos mencionadas anteriormente, se reporta como AutoFCL supera a los demás modelos del estado del arte en las 3 bases de datos. Y se explica como desean realizar en trabajo a futuro también la variación de la cantidad de capas convolucionales.

Sun, et al. en [39], al igual que Lyu, et al. en [33] busca encontrar arquitecturas que sean de menor complejidad para implementarse en dispositivos con hardware limitado. Para ello propone un algoritmo de optimización de búsqueda de arquitectura basado en *semi-Nonnegative Matrix Factorization* para capas totalmente conectadas. Para ello utiliza la base de datos ImageNet y los modelos VGG16 y AlexNet. Al variar los hiperparámetros de las capas totalmente conectadas en cantidad de neuronas y cantidad de capas es posible encontrar diferentes arquitecturas considerando *semi-Nonnegative Matrix Factorization*. Al comparar dichas arquitecturas encontradas en rendimiento con los valores publicados de las arquitecturas originales de AlexNet y VGG16 en Top1 y Top5, se encuentran mejores resultados de clasificación con el método propuesto. Como prueba de implementación en un hardware con menor capacidad computacional, implementan el modelo de AlexNet en un FPGA y comparan los recursos utilizados por el FPGA entre el modelo original y el encontrado con la búsqueda de arquitectura, se observa una reducción de hasta un 82% en uso de ancho de banda de memoria RAM.

La búsqueda de arquitectura también ha sido utilizada en recientes aplicaciones como la del COVID-19. [40] presenta un trabajo para investigar el uso de un algoritmo de optimización *Hyperband* utilizando imágenes de muestras de tomografías indicando la presencia o no de la enfermedad COVID-19 para optimizar un modelo de red neuronal convolucional. Se propone un espacio de búsqueda variando: la arquitectura *backbone*, número de módulos *inception*, número de neuronas de las capas totalmente conectadas y el valor de la razón de aprendizaje. La arquitectura *backbone* puede variar entre VGG16, InceptionV3, ResNet101, DenseNet121. Al realizar la búsqueda de arquitectuEra con el

optimizador *Hyperband* se obtiene un modelo VGG16 con 5 módulos *inception*, 128 neuronas en dos capas totalmente conectadas y una razón de aprendizaje de 0.0027 obteniendo una sensibilidad del 97%. Este resultado sobre pasa el desempeño de un humano especialista en el tema de COVID-19, por lo que se puede denotar que el algoritmo funciona de manera correcta.

EfficientNet [41] es uno de los modelos profundos más famosos de aprendizaje de máquina. Tan, en dicho trabajo presenta lo que nombra como escalamiento de modelos en donde se varia la profundidad, ancho y resolución de modelos, y en ello se basa el modelo de *EfficientNet*. Tan describe como al variar la profundidad, ancho y resolución de los modelos es posible mejorar ciertas condiciones del mismo. Escalar un modelo puede resultar en una exactitud y eficiencia mejor, aumentar la profundidad puede extraer características más complejas y variar el ancho del modelo varía la cantidad de características que son extraídas y, por lo tanto, la calidad de resolución del dato que se esté analizando. Teniendo en cuenta todas estas condiciones se menciona que es posible variar un modelo en los anteriores factores y por consecuente se puede llegar a obtener un modelo con mejor desempeño que el modelo base. Se presenta el modelo *EfficientNet-B0* como el modelo base y distintas variaciones de este con distintos desempeños evaluados con la base de datos *ImageNet*. Se obtienen 7 variantes de *EfficientNet* competitivas con otros modelos del estado del arte contra la exactitud Top1 y Top5 de *ImageNet*. El mejor modelo de *EfficientNet* obtiene el 93.3% de Top5. También se demuestra con los modelos *MobileNetv1* y *ResNet50* como al aplicar el escalamiento se obtienen variaciones del modelo con mejores desempeños de clasificación.

En los trabajos mencionados anteriormente se infiere como la búsqueda de arquitectura tiene espacios de búsqueda muy variados, se tienen distintas estrategias de búsqueda y formas de medir el desempeño. Sin embargo, todos concuerdan en que la búsqueda de arquitectura ayuda a encontrar modelos que mejoren el desempeño, ya sea de la tarea a solucionar o un objetivo en específico. Es por ello que en este trabajo de tesis se desea utilizar la búsqueda de arquitectura para encontrar un modelo profundo que, así como en [33] y [39], encuentre un modelo que requiera una menor capacidad computacional con el fin de ser implementado en un sistema embebido.

2 MARCO TEÓRICO

El presente capítulo describe matemáticamente los conceptos utilizados en el desarrollo de la tesis. Los temas que se abarcan son las imágenes utilizadas, el aumento de datos, distribución de las bases de datos para el entrenamiento, la descripción de los hiperparámetros utilizados, arquitecturas de los modelos utilizados, las métricas de desempeño y la descripción de la búsqueda de arquitectura reducida.

2.1 Imágenes utilizadas

El método de frotis de sangre utiliza un microscopio para realizar el proceso de análisis de una muestra. Para este trabajo se utiliza un microscopio B-383 Series de la marca OPTIKA ITALY que tiene una cámara OPTIKAM WiFi colocada en la parte superior. La cámara se encuentra conectada a una computadora donde se realiza la captura de imágenes conteniendo la información observada por el microscopio. Se capturan entonces las imágenes originales del microscopio I_m con una resolución espacial de 1024x728 píxeles. En la Figura 2.1 se muestra un ejemplo de I_m .

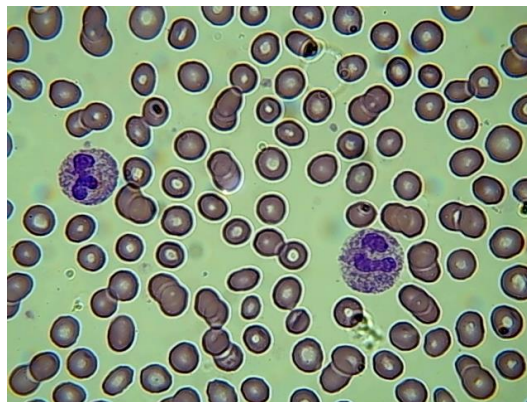


Figura 2.1 Ejemplo de imagen original tomada del microscopio.

Dichas imágenes son posteriormente analizadas por alumnos de servicio social y el Dr. Carlos Arzate de la Universidad Autónoma de Chihuahua (UACH) para ser etiquetadas manualmente identificando la clase de leucocitos presentes en la imagen, generando así una

nueva imagen etiquetada I_e . En la Figura 2.2 se muestra un ejemplo de una imagen etiquetada.



Figura 2.2 Ejemplo de imagen etiquetada.

A partir de las imágenes originales del microscopio I_m y las imágenes etiquetadas I_e se generan imágenes denominadas ground truths (GT) I_{GT} . Los GT son generados manualmente por el alumno de maestría Jesús Ríos del Instituto Tecnológico de Chihuahua (ITCH). Estos consisten en una imagen de fondo color negro y que contiene marcados los núcleos de los leucocitos que se encuentren en las imágenes I_m e I_e .



Figura 2.3 Ejemplo de imagen de GT.

Se pinta de color distinto el núcleo del leucocito identificado dependiendo de la clase de leucocito a la que pertenezca (basófilo, eosinófilo, linfocito, monocito o neutrófilo), en la Tabla 2.1 se muestran los valores de los canales RGB que definen a cada clase. Un ejemplo de imagen de GT I_{GT} se muestra en la Figura 2.3. En conjunto, las imágenes I_m , I_e e I_{GT} forman la base de datos $\mathbb{D}_{LEU_UACHDB} \mathbb{D}_{UACH}$.

Tabla 2.1 Valores de los canales RGB que definen a cada clase de leucocito.

Clase leucocito	Color	Valor canal Rojo	Valor canal Verde	Valor canal Azul
Basófilo	Verde	34	177	76
Eosinófilo	Amarillo	255	242	0
Linfocito	Azul	0	162	232
Monocito	Blanco	255	255	255
Neutrófilo	Rojo	237	28	36

Las imágenes I_{GT} se utilizan para obtener una base de datos de imágenes de leucocitos segmentadas que puedan ser utilizadas para diseñar un clasificador. Para ello, primero se utiliza GlobuConteoH®, un programa de detección de leucocitos f_d (2.1) diseñado por alumnos de ingeniería del ITCH y el Dr. Juan Ramírez del mismo instituto. Este programa utiliza técnicas de procesamiento de imágenes tradicionales para detectar los leucocitos de las imágenes I_m con análisis de color y tamaño y dar a su salida una imagen I_d similar a I_m pero con marcas presentes de círculos rojos donde se hayan detectado leucocitos. En la Figura 2.4 se muestra cómo se observa una imagen I_d .

$$f_d(I_m) = I_d \quad (2.1)$$



Figura 2.4 Ejemplo de imagen de salida de programa de detección.

Una vez obtenidas las imágenes I_m , I_{GT} e I_d , se utiliza un programa de segmentación para generar nuevas imágenes. El programa de segmentación puede ser definido como una función f_s (2.2), y se encarga de recortar los leucocitos detectados con f_d . Para ello se utilizan como entrada las imágenes I_m , I_{GT} e I_d . De I_d se utiliza la información marcada con los círculos rojos ubicando la posición del leucocito detectado, luego se hace el recorte de dicho segmento partiendo de I_m y para finalizar se utiliza I_{GT} para conocer la clase a la que pertenece el segmento recortado. El motivo de este programa de segmentación es generar de manera automática una base de datos de leucocitos segmentados de las imágenes I_m obtenidas del microscopio y que esta se encuentre separada por clases de leucocito.

$$f_s(I_m, I_{GT}, I_d) = I_s \quad (2.2)$$

Al procesar todas las imágenes tomadas del microscopio por f_d y f_s se obtiene un conjunto de imágenes segmentadas de leucocitos. Este conjunto es depurado de forma manual eliminando las imágenes segmentadas que no presenten una célula de leucocito centrada con la información adecuada. De esta forma se genera una nueva base de datos de leucocitos

segmentados denominada OGDB. La distribución de OGDB puede observarse en la Tabla 2.2. Se puede notar como la distribución de la base de datos OGDB muestra una baja incidencia de muestras por cada clase. Como se desea utilizar la base de datos OGDB para realizar el entrenamiento de un modelo profundo, se considera realizar un aumento de datos que se describe en la siguiente sección.

Tabla 2.2 Distribución de base de datos OGDB.

	Clase	Basófilo	Eosinófilo	Linfocito	Monocito	Neutrófilo	Total
OGDB	No. de imágenes	81	21	199	12	245	558

2.2 Aumento de datos

El aumento de datos es una metodología comúnmente utilizada dentro del área de aprendizaje profundo. El fin de esto es obtener una mayor cantidad de muestras para entrenar una red neuronal y llegar a una mayor generalización de la red neuronal. Como se mencionaba anteriormente, OGDB es una base de datos que desea utilizarse para entrenar un modelo profundo, más específicamente un modelo de clasificación. Pero dicha base de datos presenta una incidencia baja por clase, por lo que se realizará un aumento de datos. Se implementa una metodología donde se generen nuevas muestras artificiales que puedan ser consideradas como muestras reales basada en el método de *Generative Data Augmentation Method-Generative Adversarial Network* (GDAM-GAN) [42]. Se utiliza el método GDAM-GAN debido a la similitud que existe entre las características de las clases de leucocitos, de las cuales se puede considerar un linfocito como la clase base y las 4 clases restantes como clases deseadas a generar con la traducción imagen a imagen.

El método GDAM-GAN contiene tres elementos principales: un modelo generativo base, un modelo traducción imagen a imagen y una etapa de depuración de muestras artificiales. Para fines de este trabajo solo se utilizaron las primeras dos etapas descritas en la Figura 2.5. X_T refiriéndose al conjunto de muestras reales pertenecientes a una clase deseada que se

desea obtener una mayor cantidad de muestras, x_S definido como el conjunto de muestras de la clase base de las cuales se realizará la traducción imagen a imagen a la clase deseada, $\hat{x}_S$ un conjunto de muestras artificiales del conjunto x_S obtenidas del modelo generativo base y $\hat{x}_T$ un conjunto de muestras artificiales de la clase deseada obtenidas del modelo de traducción imagen a imagen. El modelo generativo base utiliza la arquitectura de *Wasserstein GAN* (WGAN) que consta de un generador que se encarga de producir muestras artificiales y un discriminador que decide si la muestra artificial generada se considera correcta. El modelo de traducción imagen a imagen utiliza la arquitectura *CycleGAN*, al igual que WGAN cuenta con un generador y un discriminador en su arquitectura.

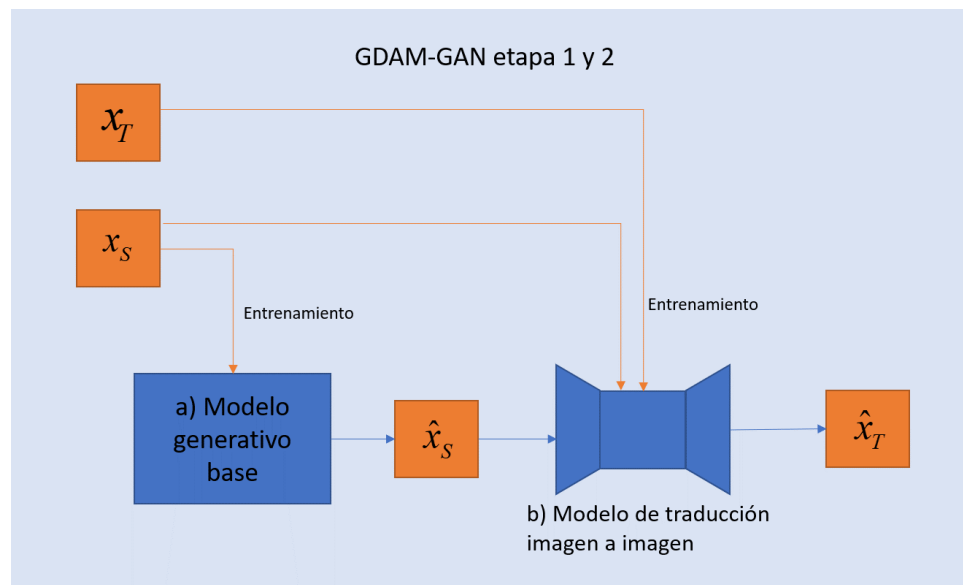


Figura 2.5 Método GDAM-GAN etapa 1 y 2.

La metodología general para el aumento de datos utilizada en este trabajo se describe en la Figura 2.6, donde se generan nuevas bases de datos a partir de GDAM-GAN para obtener finalmente una base de datos MASTER con imágenes reales y artificiales consideradas como reales.

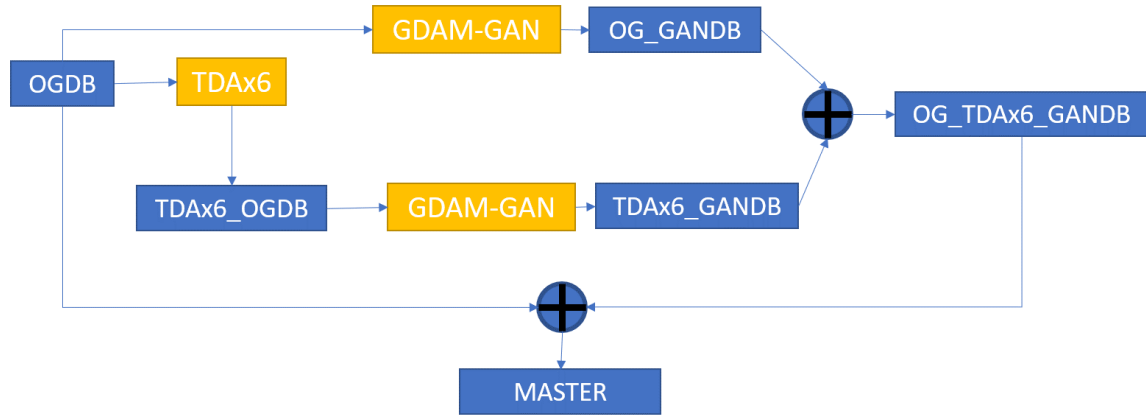


Figura 2.6 Metodología de aumento de datos.

El proceso de aumento de datos utiliza de primera instancia OGDB y un aumento de datos tradicional de ésta con transformaciones de rotaciones aleatorias (2.3) e inversiones en los ejes ‘x’ (2.4) y ‘y’ (2.5). Se aplican 5 combinaciones de transformaciones por imagen de las mencionadas y se genera una nueva base de datos TDAx6_OGDB. Una vez obtenida TDAx6_OGDB que cuenta con imágenes de leucocitos segmentadas originales y con rotaciones aleatorias e inversiones en los ejes ‘x’ y ‘y’, continua la etapa donde se utiliza GDAM-GAN. Se proponen dos caminos para utilizar GDAM-GAN, el primero realizando entrenamiento de GDAM-GAN con la base de datos OGDB y el segundo entrenando con TDAx6_OGDB. Para el entrenamiento del modelo base se utiliza todo el conjunto de muestras de x_s . La clase base para esta aplicación son los linfocitos, ya que estos contienen características similares a los otros 4 tipos de leucocitos de las cuales se podría partir más fácilmente para realizar la traducción imagen a imagen. Para el entrenamiento del modelo de traducción imagen a imagen se utiliza un conjunto de prueba de 32 imágenes y el resto para entrenamiento. En el caso de las clases que tiene una incidencia menor a 32 se clonaron algunas muestras para aumentar el conjunto de datos.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \beta & \sin \beta \\ -\sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.3)$$

Siendo x, y coordenadas de un pixel de una imagen I , β el ángulo de rotación y las nuevas x', y' coordenadas del pixel.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.4)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.5)$$

Una vez entrenado el modelo GDAM-GAN en los dos caminos propuestos se realiza la traducción imagen a imagen. Cabe aclarar que la traducción es realizada directamente de imágenes de muestras reales de la clase base (linfocitos). El resultado obtenido luego es depurado manualmente con ayuda del Dr. Arzate de biomédica de la Universidad Autónoma de Chihuahua (UACH) eliminando así las muestras artificiales que no sean consideradas como muestras reales. Se obtiene un conjunto de imágenes del entrenamiento de GDAM-GAN con la base de datos OGDB denominado OG_GANDB y otro conjunto denominado TDx6_GANDB del entrenamiento con la base de datos TDx6_OGDB. Ambos conjuntos depurados manualmente representan imágenes artificiales de leucocitos que son consideradas como muestras reales, por lo que se genera un nuevo conjunto con la unión de ambos denominado OG_TDx6_GANDB. Como último paso se juntan las muestras artificiales OG_TDx6_GANDB con las muestras reales OGDB para obtener una nueva base de datos MASTER que será utilizada para el diseño de un modelo profundo clasificador de leucocitos.

2.3 Distribución base de datos

Los modelos profundos requieren una gran cantidad de datos para ser entrenados y resolver una tarea específica, por ello, se realiza la metodología de aumento de datos presentada en la sección anterior. Con el aumento de datos se obtiene una base de datos MASTER que se utilizará para el entrenamiento de un modelo profundo. Los modelos profundos usualmente utilizan dos conjuntos de datos para realizar el proceso de

entrenamiento, un conjunto de entrenamiento y un conjunto de validación. Ambos conjuntos de datos se obtienen de la base de datos a utilizar y se distribuyen en distintas porciones. Para esta ocasión se utiliza una distribución de 80% de los datos de la base de datos MASTER para el conjunto de entrenamiento y 20% para el conjunto de validación.

2.4 Hiperparámetros

Los modelos profundos están compuestos de una arquitectura específica que los identifica y les otorga aptitudes para diferenciarlos de otros modelos. Las arquitecturas requieren de hiperparámetros para realizar y controlar el proceso de entrenamiento del modelo. Algunos hiperparámetros comunes son los optimizadores Δ_θ , funciones de costo J , funciones de activación $f(s)$, razón de aprendizaje α y épocas de entrenamiento E .

En el presente trabajo se utilizan modelos profundos con J , $f(s)$, α y E iguales. Solamente varía el optimizador Δ_θ utilizado. A continuación, se describen cada uno de los hiperparámetros utilizados por los modelos profundos.

2.4.1 Optimizadores

Los optimizadores son funciones que ayudan a desarrollar el entrenamiento de un modelo buscando minimizar el error de la función de costo J . Los optimizadores utilizados dentro de este trabajo son *Stochastic Gradient Descent* (SGD) y *Adaptive Moment Estimation* (Adam), debido a que fueron los utilizados en la competencia de ImageNet para los modelos ResNet50, VGG16 y AlexNet.

El optimizador SGD (2.6) se caracteriza por sustraer la multiplicación del gradiente con la razón de aprendizaje del parámetro a actualizar.

$$\theta_t := \theta_{t-1} - \alpha \nabla Q_t(\theta) \quad (2.6)$$

El optimizador Adam (2.7) es una extensión del optimizador SGD donde simplemente se utiliza el momento para mejorar el rendimiento.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\partial L}{\partial \theta_t} \right] v_t \quad (2.7)$$

Ambos son utilizados para el entrenamiento de modelos profundos durante el desarrollo de este trabajo de investigación.

2.4.2 Función de costo

La función de costo permite conocer que tan bien se desempeña un modelo profundo calculando el error existente entre la predicción del modelo y la salida correcta. Existen funciones de costo para resolver distintos tipos de problemáticas como: regresión, clasificación binaria, clasificación multiclase, entre otras. En esta tesis se utiliza las funciones de costo para clasificación multiclase. Una función de costo que es muy común para la clasificación multiclase es la entropía cruzada CE (2.8), donde C es el número de clases, i es el índice de la clase, t_i es la etiqueta y s la predicción. En la entropía cruzada se compara la probabilidad de cada predicción de cada clase al valor deseado y se penaliza dependiendo del error que existe entre la diferencia de la probabilidad y el valor esperado.

$$CE = - \sum_i^C t_i \log(f(s)_i) \quad (2.8)$$

2.4.3 Funciones de activación

Las funciones de activación se utilizan para definir la salida de una neurona dependiendo de una o varias entradas. Existen distintas funciones de activación como: linear, ReLU, Heaviside, Logística, Leaky ReLU, etc. Las mencionadas anteriormente son utilizadas en las capas profundas de modelos, pero normalmente en la capa de salida de un modelo para problemas de clasificación multiclase se utiliza una función de activación llamada *Softmax*. Durante esta investigación se hace uso de las funciones ReLU (2.9) y Softmax (2.10).

$$\text{ReLU} = \max(0, s) = \begin{cases} s & \text{si } s > 0 \\ 0 & \text{si } s \leq 0 \end{cases} \quad (2.9)$$

$$\text{Softmax} = \frac{e^{s_i}}{\sum_{j=1}^C e^{s_j}} \text{ para } j = 1, \dots, C \quad (2.10)$$

2.4.4 Razón de aprendizaje

La razón de aprendizaje o también llamado por su denominación en inglés *learning rate*, es el hiperparámetro utilizado para controlar el ritmo de actualización de los parámetros de un modelo dado el error calculado. De manera sencilla, la razón de aprendizaje se puede mantener fija durante el entrenamiento de un modelo profundo, pero de acuerdo con información presentada en [43] y [44] el contar con una razón de aprendizaje variable puede mejorar el proceso de entrenamiento. Por ello, se presenta una razón de aprendizaje dinámica α_{din} (2.11) que comienza en un valor de 0.01 y se va reduciendo cada 50 épocas de entrenamiento.

$$\alpha_{din} = \begin{cases} 0.01 & \text{si } 1 \leq E \leq 50 \\ 0.005 & \text{si } 51 \leq E \leq 100 \\ 0.001 & \text{si } 101 \leq E \leq 150 \\ 0.0005 & \text{si } 151 \leq E \leq 200 \end{cases} \quad (2.11)$$

2.4.5 Épocas de entrenamiento

Las épocas E , en aprendizaje de máquina, se conocen como el tiempo que dura entrenándose un modelo profundo. Más específicamente una época se refiere a un ciclo en el que el modelo profundo procesa todos los datos del conjunto de entrenamiento. Usualmente, en cada época del entrenamiento el modelo profundo va adquiriendo información relevante de los datos que le ayudan a solucionar la tarea en cuestión y entre mayor cantidad de épocas el modelo trabaja por más tiempo los datos. Al trabajar por mayor tiempo los datos el modelo puede aprender características específicas sobre ellos que le ayuden a resolver la tarea para la cual está siendo entrenado. Sin embargo, en aprendizaje profundo existe una cantidad de épocas que si se sobre pasa puede el modelo llegar a memorizar los datos cayendo en lo que

se conoce como *overfitting*. Para este trabajo se propone una cantidad arbitraria de 200 épocas (2.12) para realizar el entrenamiento de modelos profundos.

$$E = 200 \quad (2.12)$$

2.5 Métricas de desempeño

Las métricas de desempeño son un factor importante a considerar durante el entrenamiento de un modelo profundo de aprendizaje de máquina. Estas se utilizan para evaluar el proceso de aprendizaje durante el entrenamiento. Existen distintas métricas de desempeño dependiendo de lo que se desee medir, algunas como: exactitud, precisión, exhaustividad, puntuación F1, especificidad, curva de características operativas del receptor, pérdida logarítmica, índice de *jaccard*, etcétera. Para el desarrollo de esta investigación se consideran 3 métricas para medir el desempeño de modelos profundos, exactitud (2.13) que indica el porcentaje de predicciones que el modelo realizó correctamente, *recall* (2.14) el porcentaje de predicciones correctas de una clase y exactitud balanceada (BAM) indica el porcentaje promedio de predicciones correctas de todas las clases. Se seleccionaron dichas métricas ya que se trabaja en una tarea de clasificación multiclase desbalanceada. Se define *VP* como verdaderos positivos, *FN* falso negativo, *VN* verdadero negativo y *FP* falso positivo.

$$\text{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN} \quad (2.13)$$

$$\text{Recall} = \frac{VP}{VP + FN} \quad (2.14)$$

$$\text{BAM} = \frac{\sum_{c=1}^C \frac{VP(c)}{VP(c) + FN(c)}}{|C|} \quad (2.15)$$

2.6 Modelo ResNet50

Uno de los modelos seleccionados para su estudio en este trabajo de tesis es ResNet [45]. En [45] se mencionan distintas versiones del modelo ResNet que varían dependiendo del número de capas: ResNet18, ResNet34, ResNet50, ResNet101 y ResNet152. Para este fin, se utiliza el modelo ResNet50 que consta de 50 capas debido al análisis del estado del arte realizado para la selección de modelos. En la Figura 2.7 se puede observar la arquitectura general del modelo ResNet50.

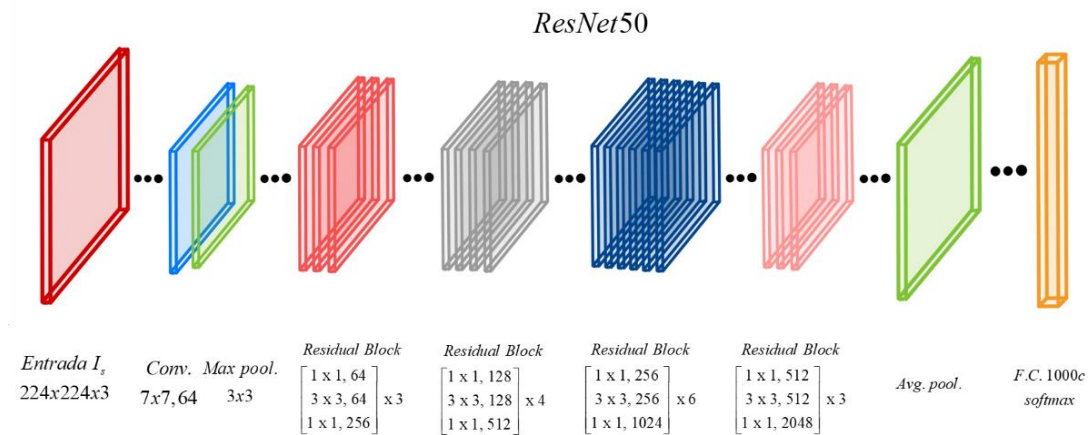


Figura 2.7 Arquitectura de ResNet50.

Lo novedoso de los modelos ResNet son los llamados bloques residuales. Estos contienen tres capas convolucionales que se aplican secuencialmente a una entrada y al final el resultado se suma con la entrada. En la Figura 2.8 se observa un ejemplo de cómo es la configuración de un bloque residual de ResNet50.

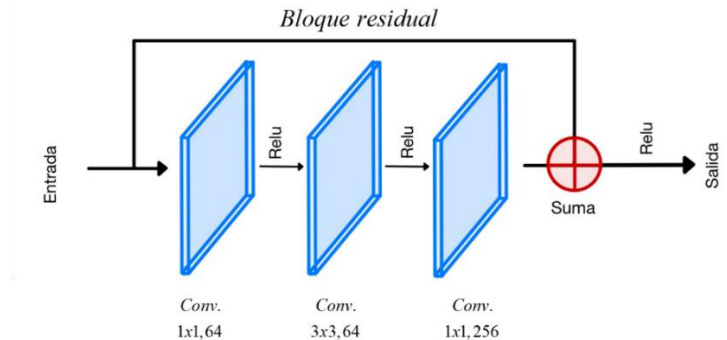


Figura 2.8 Ejemplo de bloque residual de ResNet50.

El modelo de ResNet50 se encuentra disponible con su arquitectura como aparece en el artículo en la librería de TensorFlow y se utiliza en este trabajo sin realizarle alguna modificación.

2.7 Modelo VGG16

El segundo modelo profundo para estudio del presente trabajo es VGGNet [46]. Este se presenta en dos versiones, que al igual que ResNet, varían en la cantidad de capas que constituyen a la arquitectura: VGG16 y VGG19.

Para esta aplicación se considera la versión VGG16 dado que es la de menor profundidad y complejidad con el motivo de luego implementarla en un sistema embebido que cuenta con menor capacidad computacional que una computadora de escritorio. En la Figura 2.9 se muestra la arquitectura general del modelo VGG16. Al igual que ResNet50, el modelo de VGG16 se encuentra disponible con su arquitectura como aparece en el artículo en la librería de TensorFlow y se utiliza en este trabajo sin realizarle alguna modificación.

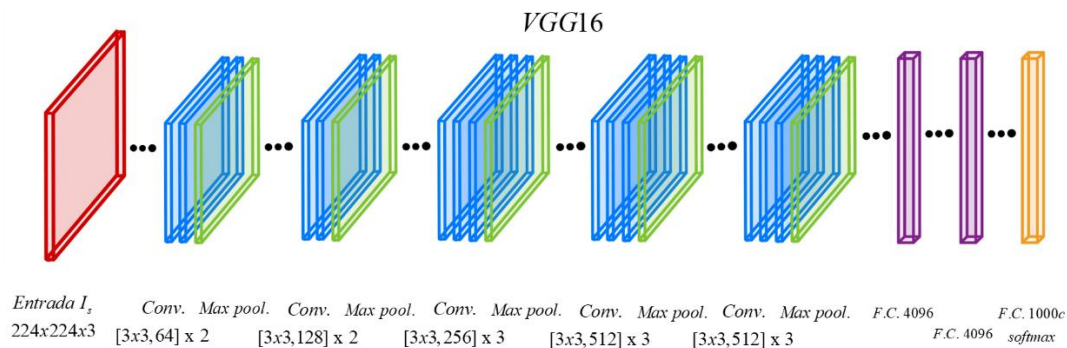


Figura 2.9 Arquitectura de VGG16.

2.8 Modelo AlexNet

El tercer modelo a considerar dentro de este trabajo es AlexNet [47]. AlexNet, consiste en un modelo profundo que está pensado para trabajar de forma paralela utilizando dos *Graphics*

Processing Unit (GPU). Esto debido a que en el momento de su publicación no existía la facilidad de alta complejidad computacional para implementarlo de forma secuencial con una sola GPU. En la Figura 2.10 se puede mostrar la arquitectura general del modelo AlexNet utilizando una arquitectura de forma paralela.

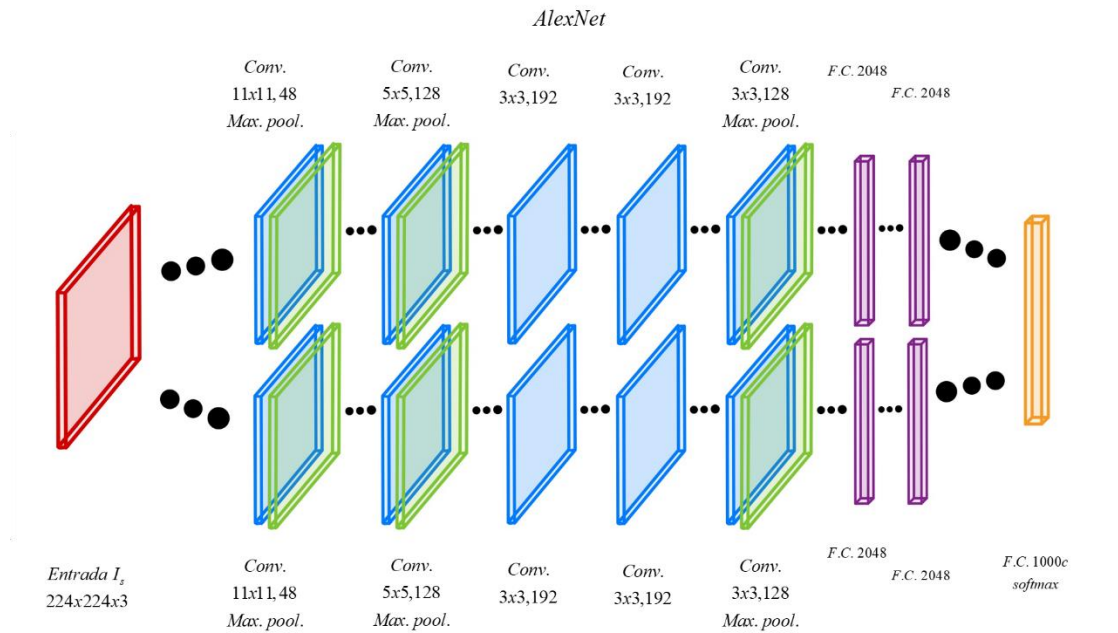


Figura 2.10 Arquitectura de AlexNet.

2.9 Modelo AlexNetv2

Debido a que al momento que se realiza el trabajo de tesis ya se cuenta con los recursos computacionales necesarios para implementar el modelo de AlexNet de manera secuencial, se propone en este trabajo de tesis un modelo AlexNetv2 que trabaje solamente con una GPU.

Como se desea utilizar el modelo de AlexNet con los parámetros publicados en la competencia de ImageNet es necesario hacer una concatenación de estos parámetros para poder trabajarlos de manera secuencial. Cabe aclarar que la implementación de ello fue por parte del tesista ya que no se encuentra dentro de la literatura que se haya implementado el modelo de AlexNet de forma secuencial con los parámetros de ImageNet.

Si se considera el modelo de AlexNet con paralelismo, y que cuenta con dos secuencias p y q . La arquitectura que funcione de manera secuencial de AlexNet está dada por la concatenación $p || q$. La arquitectura resultante de la concatenación, AlexNetv2, se puede observar en la Figura 2.11.

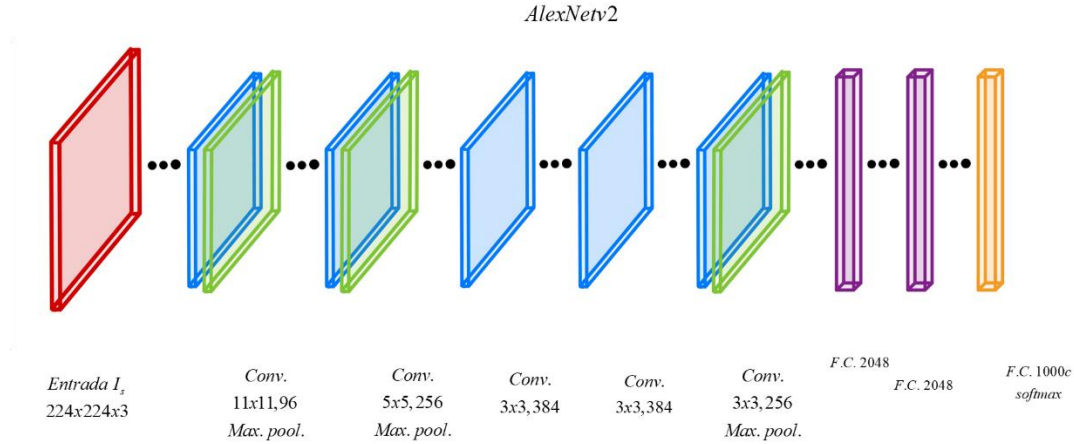


Figura 2.11 Arquitectura de AlexNetv2.

Esta arquitectura de AlexNetv2 será utilizada para realizar la etapa de selección de modelo profundo clasificador para resolver la tarea de clasificación de leucocitos, compitiendo con las arquitecturas de ResNet50 y VGG16.

2.10 Búsqueda de arquitectura

Se plantea una metodología de búsqueda de arquitectura con el fin de encontrar un modelo que utilice menor capacidad computacional para su implementación en un sistema embebido. En la búsqueda de arquitectura se requiere definir: un espacio de búsqueda O , una estrategia de búsqueda B y una estrategia para medir el desempeño P [25]. Se consideran O , B y P como conjuntos. El espacio de búsqueda está dado por variaciones de: número de capas convolucionales, número de capas totalmente conectadas y cantidad de neuronas de las capas totalmente conectadas. Estas variaciones fueron seleccionadas arbitrariamente a partir del tamaño del modelo AlexNetv2, reduciendo su valor en capas en 1 unidad y el

número de neuronas al 50%. En la Tabla 2.3 se muestra las variaciones del espacio de búsqueda y son considerados como elementos del conjunto O .

Tabla 2.3 Elementos del espacio de búsqueda.

Espacio de búsqueda		
Número de capas convolucionales N_{Conv}	Número de capas totalmente conectadas N_{FC}	Cantidad de neuronas de las capas totalmente conectadas NN_{FC}
5, 4, 3, 2, 1	2, 1	4096-4096, 4096-2048, 4096-1024, 2048-4096, 2048-2048, 2048-1024, 1024-4096, 1024-2048, 1024-1024

De esta forma, se considera la estrategia de búsqueda B como todas las posibles combinaciones del espacio de búsqueda O , 60 experimentos a realizar en la búsqueda de arquitectura considerando todas las posibles combinaciones del espacio de búsqueda. Se considera también para la estrategia de búsqueda un entrenamiento con razón de aprendizaje dinámico α_{din} , 200 épocas E y entropía cruzada CE como función de costo J .

Una vez generadas cada una de las distintas búsqueda de arquitectura con la estrategia de búsqueda, es necesario definir la estrategia para medir desempeño P . Se desea utilizar la métrica de exactitud balanceada BAM (2.15) para decidir cual arquitectura es la mejor. Se selecciona de forma manual la arquitectura que presenta un mayor desempeño de clasificación basado en la métrica BAM y que reduzca la cantidad de parámetros del modelo base de AlexNetv2.

3 BASE DE DATOS

Durante este capítulo se describirá la elaboración y composición de las distintas bases de datos que fueron utilizadas en el desarrollo de la tesis. En la sección 3.1 se presenta la base de datos que consta de imágenes tomadas directamente del microscopio, en 3.2 una base de datos con imágenes segmentadas de leucocitos, en 3.3 un aumento de datos tradicional con las imágenes de 3.2, en 3.4 una base de datos generada con aumento de datos generativo con el método de GDAM-GAN, 3.5 la base de datos PREMASTER y MASTER. En las secciones 3.6 y en adelante se describe el reetiquetado de las bases de datos de las secciones 3.2 - 3.5 que son en las que se basan principalmente los resultados de esta tesis.

3.1 LEU_UACHDB

Existen metodologías manuales y automatizadas para realizar un hemograma; la forma manual y la que se considera para este trabajo de tesis es el frotis de sangre, las técnicas automatizadas constan de equipos médicos muy costosos monetariamente hablando. El frotis de sangre es una técnica que necesita la muestra de sangre, porta muestras y un microscopio. Se coloca una gota de la muestra de sangre sobre un porta muestras y con otro porta muestras se realiza un barrido a un ángulo de 30° a 45° de inclinación. El proceso anterior es descrito visualmente en la Figura 3.1. Al obtener el frotis de sangre final, este es colocado sobre un microscopio para su análisis.

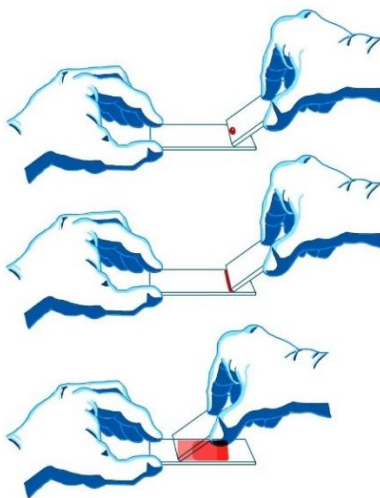


Figura 3.1 Técnica de frotis de sangre.

El presente trabajo es un proyecto colaborativo entre la Universidad Autónoma de Chihuahua (UACH) y el Tecnológico Nacional de México campus Chihuahua (ITCH). El Dr. Carlos Arzate, investigador en el laboratorio de biomédica de la UACH, propuso el proyecto de reconocimiento de leucocitos en imágenes de muestras de sangre periférica con el objetivo de crear un sistema que sea capaz de realizar el reconocimiento de leucocitos de manera automática en imágenes.

La base de datos se encuentra en una carpeta en Google Drive donde están las imágenes tomadas del microscopio, divididas en carpetas por fechas indicando el día en que se tomaron las imágenes de la muestra. Al conjunto de carpetas se le denomina LEU_UACHDB. Dentro de cada carpeta de fecha se tiene una subcarpeta indicando el aumento de magnificación utilizado en el microscopio para la toma de las imágenes. Luego se encuentran tres subcarpetas, una con las imágenes tomadas directamente de la cámara I_m , otra con las imágenes tomadas de la cámara añadiendo etiquetas manualmente en la ubicación de los leucocitos presentes en las imágenes indicando el tipo de leucocito al que pertenece I_e y otra con los ground truths GT. En la Figura 3.2 se puede observar la jerarquía que se tiene en la base de datos LEU_UACHDB.

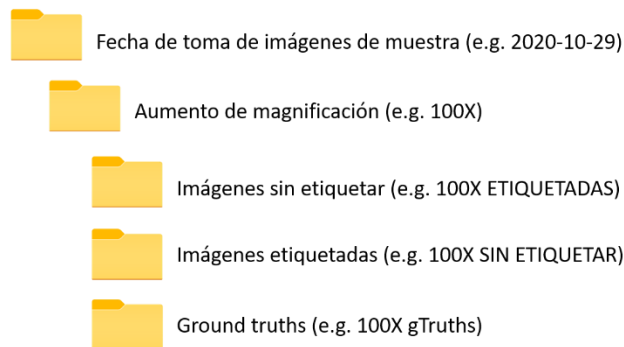


Figura 3.2 Jerarquía de base de datos LEU_UACHDB.

Las muestras tomadas con la cámara varían en su aumento de magnificación usado en el microscopio entre 40X y 100X, para el desarrollo de este trabajo de investigación solamente se trabaja con imágenes de aumento de magnificación de 100X como parte inicial de la investigación, posteriormente se analizarán las imágenes de aumento de 40X. Para apoyo visual de las imágenes contenidas en LEU_UACHDB, en la Figura 3.3 se muestra un ejemplo

de una imagen tomada con un aumento de magnificación de 100X en la fecha 29 de octubre del año 2020, en la Figura 3.4 su imagen etiquetada y en la Figura 3.5 su ground truth.

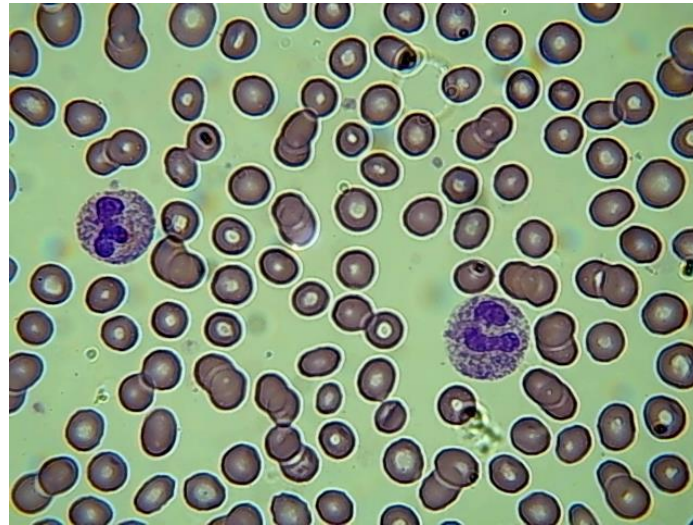


Figura 3.3 Muestra con aumento de magnificación 100X de base de datos LEU_UACHDB sin etiquetar.

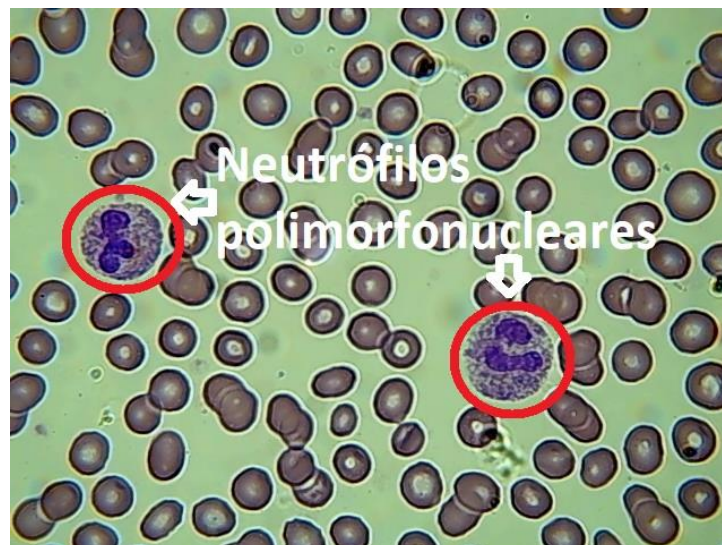


Figura 3.4 Muestra con aumento de magnificación 100X de base de datos LEU_UACHDB etiquetada.



Figura 3.5 Muestra con aumento de magnificación 100X de base de datos LEU_UACHDB ground truth.

Los GT son imágenes del mismo tamaño que las imágenes sin etiquetar I_m con un fondo negro y que tienen marcados por un proceso manual por el alumno de maestría Jesús Ríos del Tecnológico Nacional de México campus Chihuahua los núcleos de los leucocitos que se encuentran marcados en la imagen etiquetada. En este caso los GT se marcan con un color distinto los núcleos de los leucocitos, indicando cada color al tipo de leucocito al que este pertenece. En la Tabla 3.1 se indica el color perteneciente a cada clase de leucocito.

Tabla 3.1 Colores de leucocitos de GT.

Leucocito	Color	Valores de color		
		R	G	B
Linfocito	Azul	0	162	232
Monocito	Blanco	255	255	255
Neutrófilo	Rojo	237	28	36
Basófilo	Verde	34	177	76
Eosinófilo	Amarillo	255	242	0

LEU_UACHDB es la base de datos que da pie a todas las bases de datos que se presentan a continuación y de la cual se desea resolver la clasificación de leucocitos.

3.2 OGDB

OGDB representa a la base de datos con imágenes segmentadas de leucocitos obtenidas de imágenes de muestras de sangre periférica. Para obtener OGDB fue necesario procesar las imágenes de LEU_UACHDB, sección 2.1, por el algoritmo de detección y segmentación. Una vez procesadas se obtiene una base de datos con leucocitos segmentados con recortes circulares con el motivo de eliminar la mayor cantidad de componentes no deseados presentes en la imagen, considerando ruido como cualquier cosa que no sea la célula del leucocito. En la Tabla 3.2 se presenta la distribución obtenida de OGDB y en la Figura 3.6 se observan 5 ejemplos de imágenes contenidas en OGDB.

Tabla 3.2 Distribución de base de datos OGDB.

Clase	Número de muestras
Linfocito	148
Monocito	14
Neutrófilo	91
Basófilo	131
Eosinófilo	41

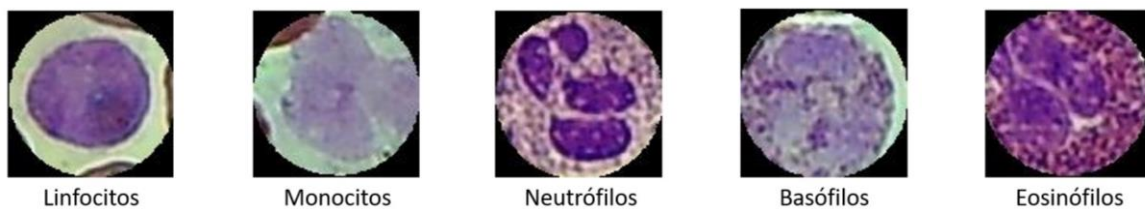


Figura 3.6 Ejemplos de leucocitos de base de datos OGDB.

3.3 TDx6_OGDB

La base de datos TDx6_OGDB surge de la necesidad de realizar un aumento de datos para entrenar un modelo profundo. El objetivo de TDx6_OGDB es ser utilizada en el entrenamiento de un modelo generativo GDAM-GAN para obtener muestras artificiales. Esta base de datos consiste en realizar transformaciones de rotaciones aleatorias e inversiones en los ejes 'x' y 'y' a cada imagen de OGDB en 5 ocasiones. Por ende, se obtiene una base de

datos con 6 veces la cantidad de muestras de OGDB. En la Tabla 3.3 se muestra la distribución de la base de datos TDAX6_OGDB.

Tabla 3.3 Distribución de base de datos TDAX6_OGDB.

Clase	Número de muestras
Linfocito	888
Monocito	84
Neutrófilo	546
Basófilo	786
Eosinófilo	246

3.4 Aumento de datos con GDAM-GAN

Como se mencionó previamente, se realizó un aumento de datos con muestras artificiales utilizando el método de GDAM-GAN. El método de GDAM-GAN consiste en tener una clase base de la cual se puedan generar nuevas imágenes correspondientes a las otras clases a partir de ruido aleatorio. Para esta aplicación la clase base a utilizar fueron los linfocitos debido a la morfología circular que estos tienen en su núcleo, de la cual se podría derivar más fácilmente la generación de las otras clases que presentan núcleos con distintas características. Al utilizar la clase base lo que hace el método de GDAM-GAN es una traducción imagen a imagen obteniendo por cada imagen de entrada de la clase base una imagen de salida de la clase deseada.

Se proponen dos caminos para utilizar GDAM-GAN, el primero realizando entrenamiento de GDAM-GAN con la base de datos OGDB y el segundo entrenando con TDAX6_OGDB. Así, se obtienen 2 resultados al realizar el aumento de datos con GDAM-GAN. El resultado de ambos aumentos de datos cuando se utiliza OGDB y TDAX6_OGDB para el entrenamiento es mostrado en la Tabla 3.4.

Tabla 3.4 Cantidad de imágenes generadas por clase con el método GDAM-GAN.

Clase	GDAM-GAN entrenada con OGDB	GDAM-GAN entrenada con TDAx6_OGDB
Linfocito	896	256
Monocito	148	148
Neutrófilo	148	148
Eosinófilo	148	148
Basófilo	148	148

Cabe aclarar que las imágenes generadas por el método de GDAM-GAN no son todas correctas al momento de compararse con una clase real de leucocito porque presentan núcleos amorfos que no van de acuerdo con la morfología de leucocitos reales. Por ello se realizó una depuración manual de todas las imágenes con ayuda del Dr. Arzate de biomédica de la UACH. Al depurar ambos entrenamientos se obtienen las bases de datos OG_GANDB y TDAx6_GANDB, del método GDAM-GAN entrenado con OGDB y TDAx6_OGDB respectivamente. En la Tabla 3.5 se muestra la distribución de ambas bases de datos generadas con el método de GDAM-GAN ya depuradas.

Tabla 3.5 Distribución de bases de datos generadas con método GDAM-GAN depuradas.

Clase	OG_GANDB	TDAx6_GANDB
Linfocito	602	95
Monocito	53	17
Neutrófilo	88	69
Eosinófilo	57	21
Basófilo	66	41

De ambas bases de datos recién creadas se generó una unión llamada OG_TDAx6_GANDB que consta de todas las imágenes de OG_GANDB y TDAx6_GANDB, en la Tabla 3.6 se muestra la distribución de la base de datos OG_TDAx6_GANDB.

Tabla 3.6 Distribución de base de datos OG_TDAx6_GANDB.

Clase	OG_TDAx6_GANDB
Linfocito	697
Monocito	70
Neutrófilo	157
Eosinófilo	78
Basófilo	107

3.5 PREMASTERDB y MASTER

En esta etapa se cuenta con una base de datos con imágenes segmentadas originales y reales de leucocitos (OGDB) y otra con imágenes segmentadas de leucocitos artificiales (OG_TDAx6_GANDB). Al combinar ambas bases de datos se genera una nueva base de datos llamada PREMASTERDB. En la Tabla 3.7 se muestra la distribución de dicha base de datos. Esta base de datos representa muestras únicas sin aumento de datos tradicional, pero se le denomina con el prefijo PRE porque se desea crear una nueva base de datos a partir de esta con aumento de datos tradicional.

Tabla 3.7 Distribución de base de datos PREMASTERDB.

Clase	PREMASTERDB
Linfocito	845
Monocito	84
Neutrófilo	248
Eosinófilo	119
Basófilo	238

Para generar la base de datos MASTER se utiliza un aumento de datos tradicional con rotaciones de la imagen cada 45° e inversiones en los ejes 'x' y 'y'. De esta manera de cada

imagen de PREMASTERDB se obtienen 15 imágenes más. La distribución de la base de datos MASTER se muestra en la

Tabla 3.8.

Tabla 3.8 Distribución de base de datos MASTER.

Clase	MASTER
Linfocito	13520
Monocito	1344
Neutrófilo	3968
Eosinófilo	1904
Basófilo	3808

El objetivo de generar la base de datos MASTER es contar con una base de datos de una magnitud de datos aceptable para entrenar un modelo de *deep learning*. Dentro del estado del arte se encuentra la base de datos Raabin [48] que presenta una distribución como la que se muestra en la Tabla 3.9. En total la base de datos Raabin cuenta con 14,514 imágenes de leucocitos y la base de datos MASTER con 24,544 por lo que se podría considerar como una base de datos competente para entrenamiento de modelos profundos dado que la base de datos Raabin es utilizada ampliamente en la literatura para entrenamientos de modelos profundos.

Tabla 3.9 Distribución de base de datos Raabin.

Clase	Raabin train	Raabin test	Raabin total
Linfocito	2427	1034	3461
Monocito	561	234	795
Neutrófilo	6231	2660	8891
Eosinófilo	744	322	1066
Basófilo	212	89	301

Una vez generada la base de datos MASTER se considera que se cuenta con una base de datos apta para realizar entrenamientos de modelos profundos.

3.6 Reetiquetado de base de datos

Durante un proceso manual de análisis de los datos y con mayor conocimiento adquirido de la morfología de leucocitos para su diferenciación entre clases, se descubre que los datos que se están utilizando se encuentran mal etiquetados. Principalmente los ground truths tienen marcado un color erróneo a la clase a la que pertenece el leucocito. Por ello, se realiza un reetiquetado de todos los 562 GT manualmente utilizando el conocimiento adquirido y el libro Atlas de Hematología Clínica [49] como referencia.

Para dar una descripción más detallada de la nueva distribución de los datos contenidos en LEU_UACHDB con el reetiquetado, se presenta la Tabla 3.10. En esta se indican cuantas fechas de toma de muestra se tienen consideradas para este trabajo de tesis, la cantidad de imágenes que contiene cada fecha y el número de leucocitos reetiquetados por clase dentro de los ground truths.

Tabla 3.10 Distribución de reetiquetado de GT.

Fecha	Cantidad de I_m, I_e , GT 100X	Basófilos en GT	Eosinófilos en GT	Linfocitos en GT	Monocitos en GT	Neutrófilos en GT
2020-10-29	42	0	3	6	0	36
2020-11-09	50	0	5	8	6	39
2020-11-10	50	52	5	13	3	11
2020-11-13	50	15	1	9	0	34
2020-11-20	50	1	1	50	0	4
2020-11-26	74	0	1	73	0	9
2020-12-18	50	5	3	25	0	36
2020-12-21	74	13	4	14	0	44
2021-01-13	50	4	0	9	1	47
2021-01-15	26	0	0	3	0	31
2021-03-17	46	0	0	0	2	24
Total	562	90	23	210	12	315

3.7 Nueva base de datos OGDB

A partir de la nueva base de datos LEU_UACHDB, se repite el proceso de ingresar las imágenes de LEU_UACHDB reetiquetadas por el algoritmo de detección y segmentación.

Una vez procesadas se genera una nueva base de datos OGDB con una nueva distribución mostrada en la Tabla 3.11.

Tabla 3.11 Distribución de nueva base de datos OGDB.

Clase	Número de muestras
Basófilo	81
Eosinófilo	21
Linfocito	199
Monocito	12
Neutrófilo	245

3.8 Nueva base de datos TDAX6_OGDB

De igual manera que en OGDB se genera una nueva base de datos TDAX6_OGDB siguiendo el mismo proceso anteriormente descrito de aumento de datos tradicional. La nueva distribución de TDAX6_OGDB se muestra en la Tabla 3.12.

Tabla 3.12 Distribución de nueva base de datos TDAX6_OGDB.

Clase	Número de muestras
Basófilo	486
Eosinófilo	126
Linfocito	1194
Monocito	72
Neutrófilo	1470

3.9 Nuevo aumento de datos con GDAM-GAN

Dado el hecho que los datos fueron reetiquetados y cambiaron las distribuciones de ellos mismos, se repite el proceso de entrenamiento del modelo GDAM-GAN. Las Tabla 3.13, Tabla 3.14 y Tabla 3.15 muestran los nuevos resultados obtenidos de las bases de datos OG_GANDB, TDAX6_GANDB y OG_TDAX6_GANDB respectivamente.

Tabla 3.13 Distribución de nueva base de datos OG_GANDB.

Clase	Número de muestras
Basófilo	46
Eosinófilo	8
Linfocito	175
Monocito	22
Neutrófilo	65

Tabla 3.14 Distribución de nueva base de datos TDAx6_GANDB.

Clase	Número de muestras
Basófilo	69
Eosinófilo	8
Linfocito	153
Monocito	5
Neutrófilo	54

Tabla 3.15 Distribución de nueva base de datos OG_TDAx6_GANDB.

Clase	Número de muestras
Basófilo	115
Eosinófilo	16
Linfocito	328
Monocito	27
Neutrófilo	119

3.10 Nueva base de datos MASTER

Para este punto se generaría la base de datos PREMASTERDB, pero después de realizar el análisis de los datos durante el reetiquetado, se decide renombrar la base de datos PREMASTERDB a MASTER y eliminar el aumento de datos tradicional que generaba 15

variaciones de la base de datos MASTER antes de reetiquetar ya que ahora se utilizará una nueva forma de aumento de datos anidada al modelo. Se elimina el proceso de aumento de datos tradicional de la anterior MASTER ya que se considera una muy alta similitud entre las muestras generadas dado que las rotaciones eran cada 24° . La distribución final de la nueva base de datos MASTER se muestra en la Tabla 3.16.

Tabla 3.16 Distribución de nueva base de datos MASTER.

Clase	Número de muestras
Basófilo	196
Eosinófilo	37
Linfocito	527
Monocito	39
Neutrófilo	364

3.11 MASTER_CHALLENGE

Debido a que la base de datos MASTER cuenta con 52% de muestras artificiales y 48% de muestras reales, considerar cualquiera de las muestras para la validación de un modelo profundo no sería correcto. Ya que, si se seleccionan las muestras de forma aleatoria, cabe la posibilidad de sesgar el modelo a validar solo con muestras artificiales que realmente no es el caso real del modelo, debido a que este encontrará muestras reales cuando se ponga a prueba. Por ello, se crea la base de datos MASTER_CHALLENGE. Esta base de datos se divide en un conjunto de entrenamiento y un conjunto de validación fijos. Para el conjunto de validación se tienen seleccionados datos solamente reales. De esta forma al entrenar un modelo con esta base de datos se puede conocer si los datos artificiales si ayudan a la generalización del modelo, o solamente lo están sesgando a la memorización de muestras artificiales. En la Figura 3.7 se muestra una comparación de la base de datos MASTER y MASTER_CHALLENGE teniendo en cuenta que ambas utilizan 80% de los datos para entrenamiento y 20% para prueba y se detalla cómo es el porcentaje de cantidad de muestras entre las muestras reales y artificiales.

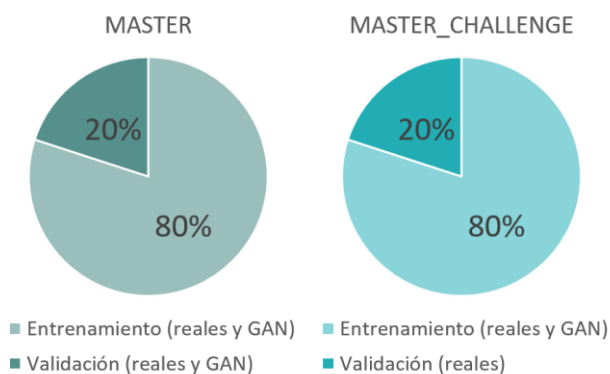


Figura 3.7 Porcentaje de cantidad de muestras reales y artificiales en base de datos MASTER y MASTER_CHALLENGE.

El desarrollo de las bases de datos mencionadas en este capítulo es la parte esencial de la investigación, ya que todos los modelos a utilizar necesitan extraer información de los datos. A partir de este punto, se utiliza la base de datos OGDB, MASTER y MASTER_CHALLENGE para llevar a cabo los entrenamientos de los distintos modelos seleccionados del estado del arte y que su diseño se describe en el capítulo 4.

4 DISEÑO DE CLASIFICADOR PROFUNDO

El presente capítulo describe el proceso para el diseño del clasificador profundo, el cual consta de las etapas de detección, segmentación, selección de modelos del estado del arte, implementación y entrenamiento de los modelos seleccionados, comparación de resultados contra el estado del arte e implementación en un sistema embebido. A lo largo de las secciones mencionadas anteriormente se puede revisar el proceso llevado a cabo para resolver la tarea de clasificación de leucocitos en imágenes de muestra de sangre periférica en la presente tesis.

4.1 Detección

Se había trabajado fuera de esta tesis anteriormente con la base de datos LEU_UACHDB en dos ocasiones, la primera para solucionar la detección de leucocitos en las imágenes y la segunda para realizar segmentación de los núcleos de los leucocitos. Ahora se trabaja en tercera ocasión con la base de datos con el objetivo principal de realizar la clasificación de leucocitos. Se pretende utilizar el trabajo realizado previamente de detección y acoplarlo con la clasificación para constituir un sistema capaz de reconocer leucocitos en imágenes de muestras de sangre periférica. En esta sección y la siguiente, se describen brevemente los trabajos anteriores realizados con la base de datos para resolver detección y segmentación de leucocitos.

El trabajo realizado para la detección de leucocitos fue desarrollado por el Dr. Juan Ramírez y alumnos de ingeniería del Tecnológico Nacional de México campus Chihuahua. Se diseñó un programa de detección de leucocitos llamado GlobuConteoH® en el cual se procesa una imagen de entrada I_m y a la salida se obtiene una imagen I_d del mismo tamaño marcada con círculos rojos en las áreas donde se detectaron leucocitos de la imagen de entrada. En la Figura 4.1 se puede observar un ejemplo de salida del algoritmo de detección.

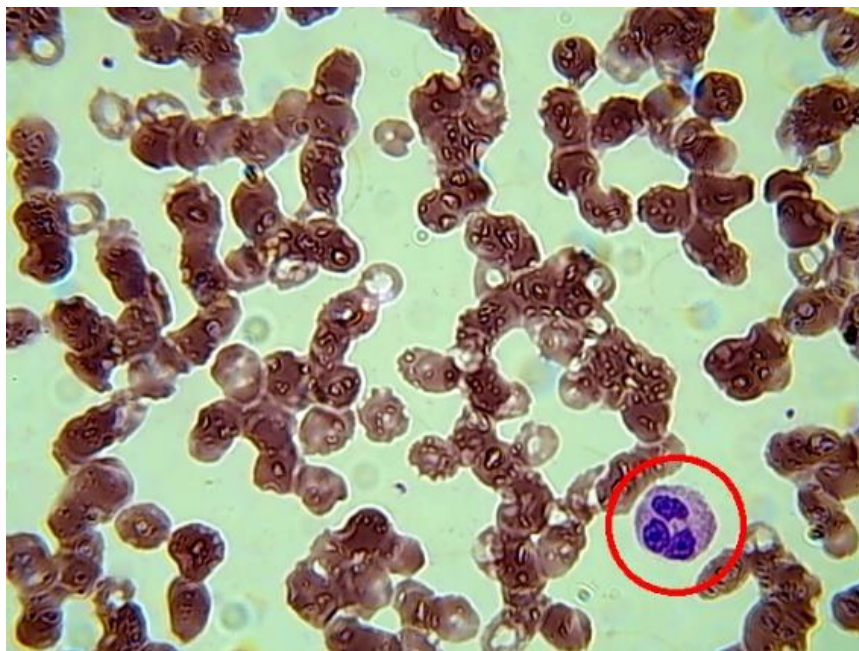


Figura 4.1 Ejemplo de imagen de salida de algoritmo de detección.

El programa de detección mencionado, basa su funcionamiento en características como tamaño, espacio de separación entre células, área, circularidad, radio, convexidad y color para realizar la detección.

4.2 Segmentación

Para comenzar con la tarea de clasificación era necesario generar una base de datos con muestras de leucocitos segmentadas (OGDB). El proceso general para realizarla se describe en la Figura 4.2, la cual necesita la base de datos LEU_UACHDB, el programa de detección y un algoritmo de segmentación (Algoritmo 1).

Algoritmo 1 Segmentación diseñado en MATLAB

Requiere: I_m , I_d , I_{GT} , r donde todas son imágenes de 3 canales

```

1: for  $j \leq \text{num\_imag}$ :
2:   for  $k \leq 768$ :
3:     for  $l \leq 1024$ :
4:       if  $I_d(k,l,1) \geq 230 \&\& I_d(k,l,2) \leq 15 \&\& I_d(k,l,3) \leq 10$ :
5:          $r(k,l,:) = 255$ 
6:       else:
7:          $r(k,l,:) = 0$ 
8:    $In3 = \text{binariza}(r)$ 
9:    $In2 = \text{binariza}(I_{GT})$ 
10:   $[\text{centros1}, \text{bounding box1}] = \text{regionprops}(In3)$ 
11:   $[\text{centros2}] = \text{regionprops}(In2)$ 
12:  Compara centros1 y centros2, y color de coordenada de centros2 en  $I_{GT}$  para indicar la
    clase de leucocito en  $\text{colores}$ .
13:  Recorte de  $I_m$  con  $\text{centros1}, \text{bounding box1}, \text{colores}$ .

```

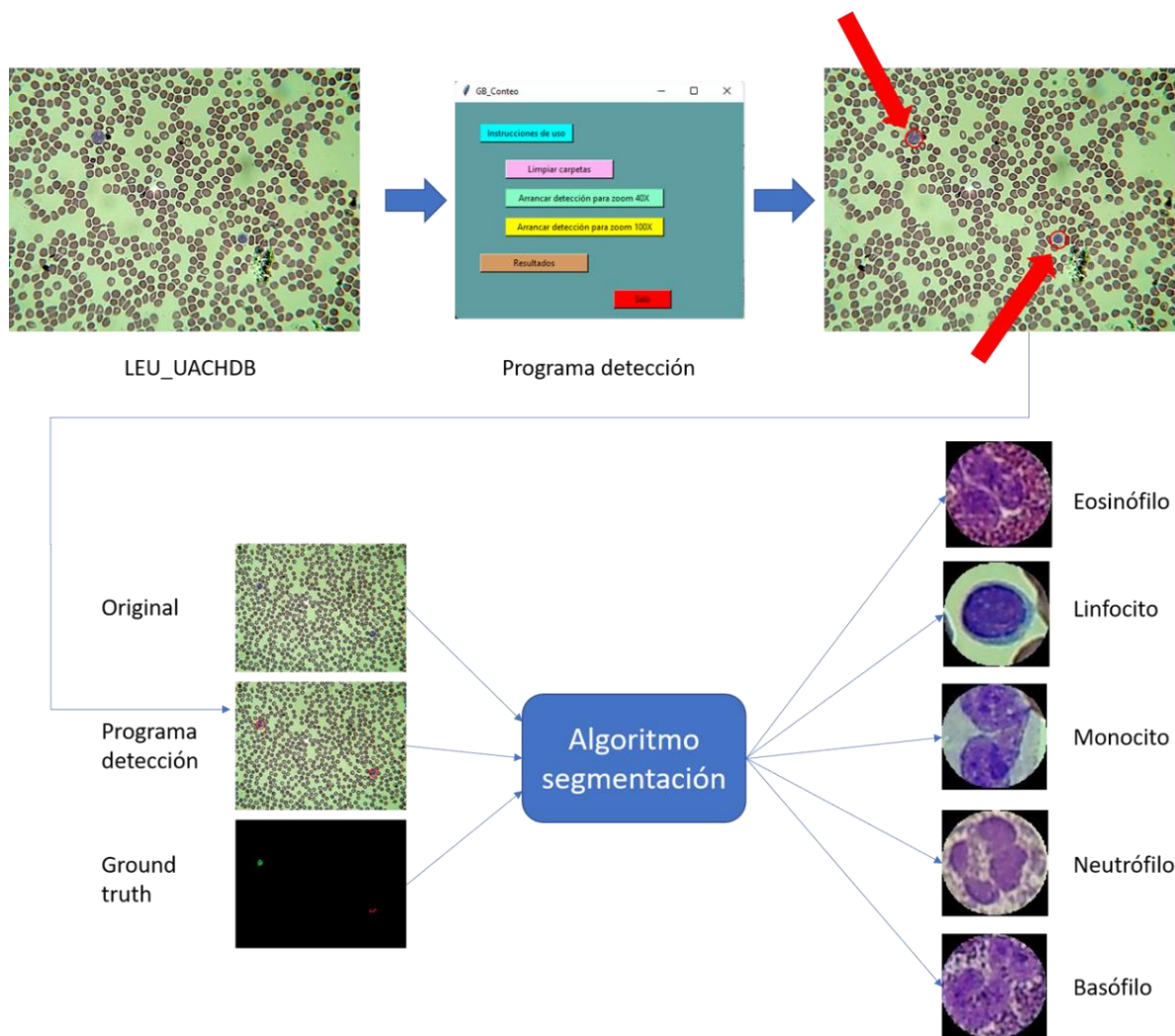


Figura 4.2 Esquema general del proceso para generar la base de datos de leucocitos segmentados.

El proceso general para obtener la base de datos de leucocitos segmentados consiste en tomar una imagen sin etiquetar de LEU_UACHDB y procesarla con el programa de detección. Este último brinda a su salida una imagen del mismo tamaño que la entrada y en la misma escala de color (RGB) marcando con un círculo rojo los leucocitos detectados. Posteriormente se utiliza el algoritmo de segmentación que requiere como entradas la imagen sin etiquetar de LEU_UACHDB, la imagen de salida del programa de detección y el GT. Dicho algoritmo utiliza los círculos marcados por el programa de detección para encontrar la coordenada de los leucocitos encontrados y comparar esa coordenada con el GT para conocer

a qué clase de leucocito pertenece. A la salida del algoritmo de segmentación se obtienen los leucocitos detectados por el programa de detección de manera segmentada para formar la base de datos OGDB a utilizar para la tarea de clasificación.

4.3 Selección de modelos profundos del estado del arte para la clasificación de leucocitos

Dentro del estado del arte existen varios trabajos que resuelven la tarea de clasificación de leucocitos con ayuda de modelos profundos. Se realizó un estudio de los trabajos publicados en los años recientes con información del tipo de modelo, año de publicación, desempeño de clasificación, entre otras. Dicha información se muestra en la Tabla 4.1, en esta “LR” hace referencia a razón de aprendizaje, “Fun. Act.” se refiere a función de activación y “Opt” se refiere a optimizador.

Tabla 4.1 Análisis del estado del arte, selección de modelos.

Año	Fuente	Modelo	Hiper/Param	Desempeño	Tiempo procesamiento	Hardware
2020	[50]	VGGNet	Pérdida: entropía cruzada Épocas: 9 LR: 0.2 Fun. Act.: softmax	98.92% Exactitud	0.216 s por 1 inferencia	Intel core i5 @ 2.5 GHz 4 GB 1600 MHz DDR3
2019	[51]	OverfeatNet, AlexNet, VGGNet	Fun. Act.: ReLu, softmax	96.10% Exactitud	0.356 por 1 inferencia	Quad-Core 2GHz Intel i7 8GB RAM, Nvidia-tesla 12GB RAM
2021	[21]	AlexNet, GoogleNet	Fun. Act.: ReLu	96.47% Exactitud	-	Intel Core i7-9750H, 8GB RAM
2022	[22]	ResNet, DenseNet	Opt.: Adam Pérdida: entropía cruzada LR: 0.00001 Épocas: 100	97.37% Exactitud	-	Intel E5-2650 2.20GHz, 256GB RAM, NVIDIA TITAN Xp 12GB

Tabla 4.1 Análisis del estado del arte, selección de modelos, continuación.

Año	Fuente	Modelo	Hiper/Param	Desempeño	Tiempo procesamiento	Hardware
2019	[20]	AlexNet	Fun. Act.: sigmoide Pérdida: entropía cruzada	98.4% Promedio exactitud	-	-
2020	[52]	ResNet	Épocas: 30 Opt: SGD LR: 0.001 Fun. Act.: ReLu, tanh	91.68% Exactitud	-	Linux OS, i7- 6800k, 32GB RAM, GTX1080Ti, PyTorch
2020	[53]	ResNet50	Opt: Adam	98.04% Exactitud	500-900 ms por 1 inferencia	Python, Tensorflow, Nvidia GTX1080
2021	[54]	VGG19, ResNet50	Opt: SGD LR: 0.01 Fun. Act.: ReLu Pérdida: entropía cruzada	96.7% Exactitud	-	-
2021	[55]	DeepLab v3+, ResNet50	Opt: SGD LR: 0.001 Capas: 50 ResNet	97.40% Exactitud	-	Intel Core i5 1.85GHz, 16 GB RAM, Intel Graphics 620 GPU
2021	[56]	RetinaNet, ResNet	Opt: Adam Stochastic Backbone Capas: 50,101,152 LR: 0.00001, cíclico con decaimiento exponencial Épocas: 200	94.84% Promedio Exactitud	219 ms por 1 inferencia	2 X GeForce GTX 1080 Ti 11GB
2020	[18]	VGG16, ResNet50, InceptionV3, Xception	Opt: Adam Épocas: 100 LR: 0.0001 Fun. Act.: ReLu	95.89% Exactitud	-	Python 3.7, Keras 2.2.4, Tensorflow 1.13.1, NVIDIA Tesla K80
2021	[15]	DenseNet121, ResNet50	Opt: Adam LR: 0.00666 Épocas: 30 Fun. Act.: ReLu	96.67% Exactitud	-	NVIDIA P100 GPU

Tabla 4.1 Análisis del estado del arte, selección de modelos, segunda continuación.

Año	Fuente	Modelo	Hiper/Param	Desempeño	Tiempo procesamiento	Hardware
2020	[16]	AlexNet	Opt.: SGD con momento LR: 0.001 Regularización L2 Capas: 8 Fun. Act.: ReLu Épocas: 9	98.87% Exactitud	-	MATLAB 2019, Intel Core i5 1.85GHz, 16GB RAM, Intel UHD Graphics 620 GPU
2020	[57]	DarkNet19	Opt.: Bayesian LR: 0.001 Épocas: 40	97.2% Exactitud	-	-
2020	[58]	ResNet50	Opt.: propuesto Fun. Act.: ReLu, tanh	94% Exactitud	-	-
2020	[19]	RetinaNet	Opt.: Adam LR: 0.001 Épocas: 300 Fun. Act.: ReLu, sigmoide	95.3% Exactitud	-	Intel Xeon E5- 2690v4@2.6GHz, 128GB RAM, Nvidia Tesla P100 GPU 16GB

Se realizó una selección de los tres modelos que tuvieran un más alto desempeño de clasificación para su futuro análisis. Se considera solo el desempeño de clasificación debido a que se desea diseñar un modelo que tenga el más alto desempeño de clasificación de primera instancia para después reducir su complejidad para luego implementarlo en un sistema embebido. Cabe mencionar que los tiempos de procesamiento son importantes hasta ese punto de diseño y que muy pocos autores reportan el tiempo de procesamiento, así como el hardware utilizado. Los 3 modelos con mayor desempeño de clasificación de leucocitos encontrados en la literatura fueron ResNet50, VGG16 y AlexNet, se encuentran mostrados en la Tabla 4.2. Estos modelos se analizaron para ver cuál de ellos resuelve de mejor manera la tarea de clasificación de leucocitos con la base de datos de la UACH.

Tabla 4.2 Modelos con mayor desempeño de clasificación en el estado del arte.

Año	Fuente	Modelo	Hiper/Param	Desempeño	Tiempo procesamiento	Hardware
2020	[50]	VGGNet	Pérdida: entropía cruzada Épocas: 9 LR: 0.2 Fun. Act.: softmax	98.92% Exactitud	0.216 s por 1 inferencia	Intel core i5 processor @ 2.5 GHz with 4 GB 1600 MHz DDR3
2020	[16]	AlexNet	Opt: SGD con momento LR: 0.001 regularización L2 Capas: 8 Fun. Act.: ReLu Épocas: 9	98.87% Exactitud	-	MATLAB 2019, Intel Core i5 1.85GHz, 16GB RAM, Intel UHD Graphics 620 GPU
2020	[53]	ResNet50	Opt.: Adam	98.04% Exactitud	500-900 ms por 1 inferencia	Python, Tensorflow, Nvidia GTX1080

En la siguiente sección se describe la implementación de cada uno de los modelos seleccionados y los hiperparámetros a utilizar.

4.4 Implementación de modelos

En esta sección se describe como se implementaron los modelos ResNet50, VGG16 y AlexNet presentados en la sección anterior para decidir cuál modelo se utilizará. Para llevar a cabo la implementación de modelos se utiliza transferencia de aprendizaje en su esquema más básico [59]. Se consideran las arquitecturas de los modelos ResNet50, VGG16 y AlexNet descritos en el capítulo 2 utilizando los parámetros publicados en el concurso *ImageNet* y entrenando solamente la capa de salida de clasificación de 1000 a 5 neuronas. También cabe aclarar que se entrenarán los modelos con la base de datos generada de leucocitos segmentados MASTER (antes del reetiquetado).

Dado que el objetivo de implementar los modelos con la base de datos propia es analizar cuál de estos se desempeña mejor en la tarea de clasificación, se implementará cada uno de ellos con los hiperparámetros que se consideren para su mejor versión (estos son los utilizados en el concurso de ImageNet). Sin embargo, se utilizarán dos hiperparámetros

constantes para los 3 modelos, el número de épocas de entrenamiento y la razón de aprendizaje (seleccionados arbitrariamente en el capítulo 2). En la Tabla 4.3 se observan los hiperparámetros que se utilizarán en cada modelo.

Tabla 4.3 Hiperparámetros para implementación de modelos.

ResNet50	VGG16	AlexNetv2
Razón de aprendizaje: α_{din}	Razón de aprendizaje: α_{din}	Razón de aprendizaje: α_{din}
Épocas E : 200	Épocas E : 200	Épocas E : 200
Optimizador Δ_{θ} : Adam	Optimizador Δ_{θ} : Adam	Optimizador Δ_{θ} : SGD
Función de costo J : CE	Función de costo J : CE	Función de costo J : CE
Función de activación $f(s)$: ReLU	Función de activación $f(s)$: ReLU	Función de activación $f(s)$: ReLU
Función de activación capa de salida: Softmax	Función de activación capa de salida: Softmax	Función de activación capa de salida: Softmax

Una vez establecidos los hiperparámetros se plantean los experimentos de entrenamientos a realizar en la siguiente sección con cada modelo para luego seleccionar de los resultados el que cuente con el mejor desempeño de clasificación. Los entrenamientos de los modelos se llevan a cabo con Python utilizando las librerías de TensorFlow 2.10 y Keras 2.10 en una Workstation Dell con un procesador Intel Xeon E5620@2.40GHz con una tarjeta gráfica NVIDIA GeForce RTX 3060.

4.5 Entrenamiento de modelos para selección de clasificador principal

Para llevar a cabo el proceso de selección de modelos ResNet50, VGG16 y AlexNet, se utilizarán la base de datos OGDB y MASTER (antes del reetiquetado) para entrenamientos. La lectura de las imágenes de la base de datos es seleccionada aleatoriamente utilizando dos semillas usadas en una función de TensorFlow para leer bases de datos llamada ‘image_dataset_from_directory’, variando así la aleatoriedad de los datos indicando que los resultados no se sesgan a una distribución específica de los datos. Por lo tanto, con cada uno

de los 3 modelos seleccionados se realizarán 4 experimentos, 2 por cada base de datos. El hecho de utilizar la base de datos OGDB y MASTER indica que tanto mejora el desempeño de clasificación de los modelos con el aumento de datos realizado e interpretar si este ayuda al proceso de entrenamiento o no.

Los datos se dividen en 80% para entrenamiento y 20% para validación en ambas bases de datos, siguiendo la distribución comúnmente utilizada en los trabajos del estado del arte. Las métricas de desempeño a utilizar para entrenamiento y validación son exactitud y BAM. Se utiliza BAM debido a que se cuenta con una base de datos con clases desbalanceadas, utilizar solamente la métrica de exactitud para validar el desempeño final del modelo sería erróneo. Esta métrica permite conocer realmente como se realiza la clasificación de cada clase porque representa un promedio de la proporción de las predicciones de verdaderos positivos correctos entre el total de predicciones correctas del modelo.

Tabla 4.4 Experimentos para selección de modelo, metodología variación de semilla.

Modelo	Base datos	Seed	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
ResNet50	OGDB	423	69.11%	75.29%	0.8715	0.839	50.24%
VGG16	OGDB	423	87.35%	76.47%	0.3929	0.6722	54.93%
AlexNetv2	OGDB	423	100%	76.47%	0.0072	0.9197	57.92%
ResNet50	OGDB	379	63.82%	78.82%	0.904	0.7646	47.52%
VGG16	OGDB	379	87.35%	83.52%	0.3496	0.6455	56.87%
AlexNetv2	OGDB	379	100%	85.88%	0.0023	0.7223	67.50%
ResNet50	MASTER	423	88.39%	87.85%	0.3244	0.3448	76.70%
VGG16	MASTER	423	96.85%	94.96%	0.0955	0.1523	90.85%
AlexNetv2	MASTER	423	100%	99.81%	0.000096916	0.0058	99.65%
ResNet50	MASTER	379	87.88%	88.10%	0.333	0.3457	77.46%
VGG16	MASTER	379	96.84%	94.96%	0.0953	0.15	90.69%
AlexNetv2	MASTER	379	100%	99.59%	0.00008323	0.0124	99.39%

Una vez definidos los modelos seleccionados, hiperparámetros, métricas de desempeño y bases de datos a utilizar es posible comenzar con la experimentación para la selección de modelo. En la Tabla 4.4 se observan los resultados de los 4 experimentos realizados con cada modelo seleccionado del estado del arte y en la Tabla 4.5 esos mismos resultados, pero ordenados por desempeño de clasificación. En esta última tabla se observan subrayados los modelos que tuvieron un mayor desempeño de clasificación para la base de datos MASTER y OGDB.

Tabla 4.5 Experimentos para selección de modelo ordenados por desempeño de clasificación.

Modelo	Base datos	Seed	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
<u>AlexNetv2</u>	<u>MASTER</u>	<u>423</u>	<u>100%</u>	<u>99.81%</u>	<u>0.000096916</u>	<u>0.0058</u>	<u>99.65%</u>
<u>AlexNetv2</u>	<u>MASTER</u>	<u>379</u>	<u>100%</u>	<u>99.59%</u>	<u>0.00008323</u>	<u>0.0124</u>	<u>99.39%</u>
VGG16	MASTER	423	96.85%	94.96%	0.0955	0.1523	90.85%
VGG16	MASTER	379	96.84%	94.96%	0.0953	0.15	90.69%
ResNet50	MASTER	379	87.88%	88.10%	0.333	0.3457	77.46%
ResNet50	MASTER	423	88.39%	87.85%	0.3244	0.3448	76.70%
<u>AlexNetv2</u>	<u>OGDB</u>	<u>379</u>	<u>100%</u>	<u>85.88%</u>	<u>0.0023</u>	<u>0.7223</u>	<u>67.50%</u>
<u>AlexNetv2</u>	<u>OGDB</u>	<u>423</u>	<u>100%</u>	<u>76.47%</u>	<u>0.0072</u>	<u>0.9197</u>	<u>57.92%</u>
VGG16	OGDB	379	87.35%	83.52%	0.3496	0.6455	56.87%
VGG16	OGDB	423	87.35%	76.47%	0.3929	0.6722	54.93%
ResNet50	OGDB	423	69.11%	75.29%	0.8715	0.839	50.24%
ResNet50	OGDB	379	63.82%	78.82%	0.904	0.7646	47.52%

Como se observa en la Tabla 4.5 el modelo que obtuvo más alto desempeño de clasificación en los experimentos para selección de modelo considerando la métrica BAM fue AlexNetv2. Esto en ambas bases de datos OGDB y MASTER, y también en las dos lecturas con distintas semillas. Al observar estos resultados se concluye que el modelo a estudiar en

los próximos capítulos será el modelo AlexNetv2. En los resultados obtenidos también se puede notar como la base de datos MASTER siempre obtuvo un desempeño mayor de clasificación comparado a todos los experimentos realizados con la base de datos OGDB, por lo que parece indicar que el aumento de datos si sirve para mejorar el desempeño del modelo profundo.

En la Figura 4.3 se observan las curvas de aprendizaje de error y exactitud del modelo entrenado con más alto desempeño de clasificación con la base de datos MASTER, y en la Figura 4.4 su matriz de confusión de los datos de validación. Las curvas de aprendizaje indican como el modelo va aprendiendo a lo largo del proceso iterativo aumentando su desempeño de exactitud y disminuyendo su error. Se hace notable como el modelo llega a un punto de no cambio a partir de la época 50 aproximadamente, por lo que el entrenamiento siguiente de las 150 épocas está de más y se aprecia un pico en la época 25 que podría indicar una perturbación en la superficie de error del problema de clasificación en cuestión. En la diagonal de la matriz de confusión se aprecian los aciertos del modelo en el conjunto de validación, teniendo erróneas solamente un total de 9 muestras del conjunto.

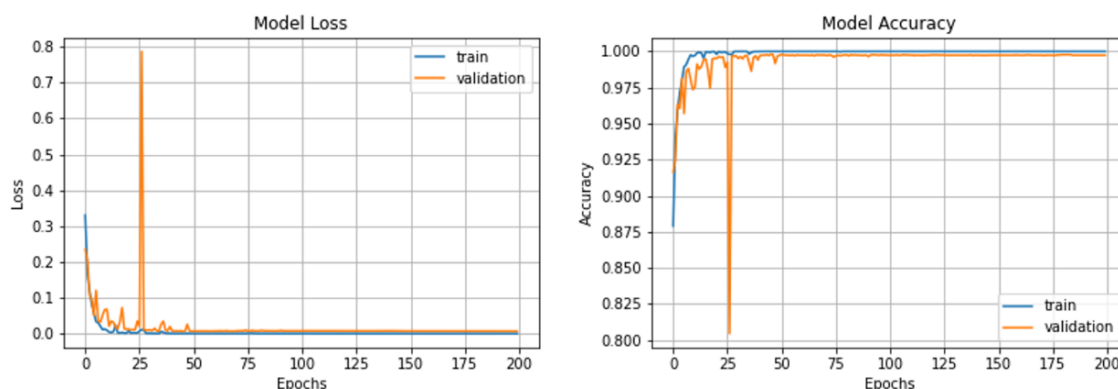


Figura 4.3 Curvas de aprendizaje de modelo AlexNetv2 entrenado con MASTER para selección de modelo.

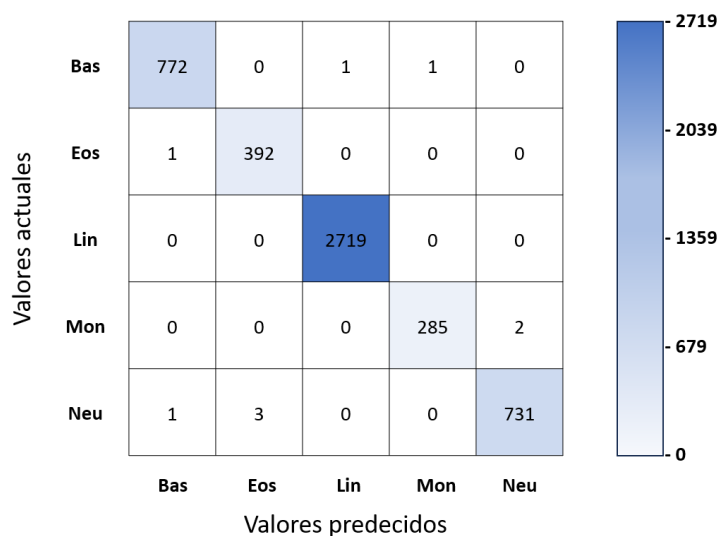


Figura 4.4 Matriz de confusión de conjunto de validación de modelo AlexNetv2 entrenado con MASTER para selección de modelo.

Los resultados obtenidos en la matriz de confusión parecen indicar que a partir de este punto ya se resuelve de manera exitosa la clasificación de leucocitos utilizando la base de datos MASTER (antes de reetiquetar) ya que contamos con 9 muestras mal clasificadas de un total de 4908 muestras, indicando esto una clasificación de leucocitos con un 99.81% de éxito que parece resolver la clasificación. Sin embargo, en la siguiente sección se detalla porque esta afirmación es incorrecta.

4.6 Segmentación con imágenes reetiquetadas

Los resultados mostrados en la sección anterior parecen indicar que se ha resuelto la tarea de clasificación de leucocitos con la base de datos MASTER (antes de reetiquetar). Sin embargo, se plantea realizar un análisis de los datos, ya que se tiene la hipótesis de una posible polarización de los datos debido al aumento de datos tradicional realizado a la base de datos PREMASTERDB que aumenta 15 imágenes por cada imagen original de leucocito con operaciones de rotación e inversión.

Al analizar visualmente la base de datos MASTER, se encontró una alta similitud entre sus muestras. Y que el hecho de que se encuentren en el conjunto de validación muestras parecidas al conjunto de entrenamiento hace que se muestre un alto desempeño de clasificación. En la Figura 4.5 se ilustra un ejemplo de una imagen de leucocito que se aumentaron 15 veces con técnicas tradicionales utilizando rotaciones e inversiones y se ejemplifica la similitud de muestras entre el conjunto de entrenamiento y validación.

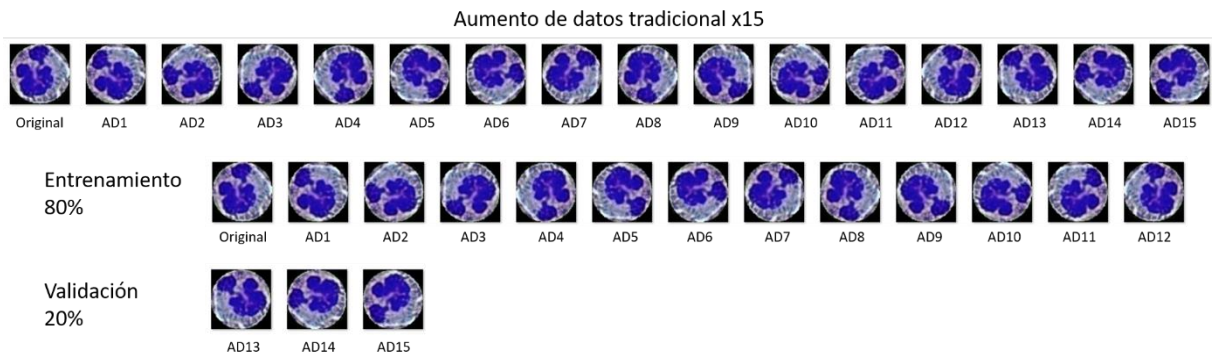


Figura 4.5 Análisis de aumento de datos tradicional en base de datos MASTER.

Para evitar este tipo de situaciones en las que sean tan parecidas las imágenes se decide utilizar la base de datos PREMASTERDB para los entrenamientos y se agrega una capa de aumento de datos dinámico a los modelos ResNet50, VGG16 y AlexNet; y esta capa dinámica solamente se aplica a las muestras de la etapa de entrenamiento. En la Figura 4.6 se muestra cómo la capa de aumento de datos dinámico se agrega a los modelos profundos entre la entrada y el inicio de la arquitectura del modelo. Las operaciones que se realizan en esta capa son las descritas en el capítulo 2: rotaciones aleatorias (2.3), e inversiones en los ejes 'x' (2.4) y 'y' (2.5). Se realiza un experimento para comparar la propuesta capa dinámica con el desempeño obtenido de la base de datos MASTER. Se observa en la Tabla 4.6 como se obtiene un desempeño aún considerado alto en la métrica BAM utilizando la base de datos PREMASTERDB y la capa de aumento de datos dinámica.

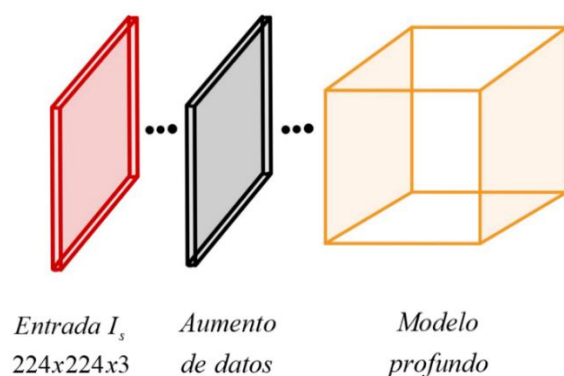


Figura 4.6 Capa de aumento de datos agregada a los modelos profundos.

Tabla 4.6 Comparativa entre experimento con MASTER y PREMASTERDB.

Modelo	Base datos	Seed	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
AlexNetv2	MASTER	423	100%	99.81%	0.000096916	0.0058	99.65%
AlexNetv2	PREMASTERDB	423	99.75%	96.41%	0.000096	0.1481	93.90%

Con la finalidad de corroborar el etiquetado de los datos se investigaron las características principales de cada clase y se revisó nuevamente la base de datos OGDB, lo que llevó a encontrar muestras mal etiquetadas en la base de datos. Algunos ejemplos son mostrados en la Figura 4.7, donde se aprecian basófilos y eosinófilos enmarcados en rojo mal etiquetados que deberían ser neutrófilos.

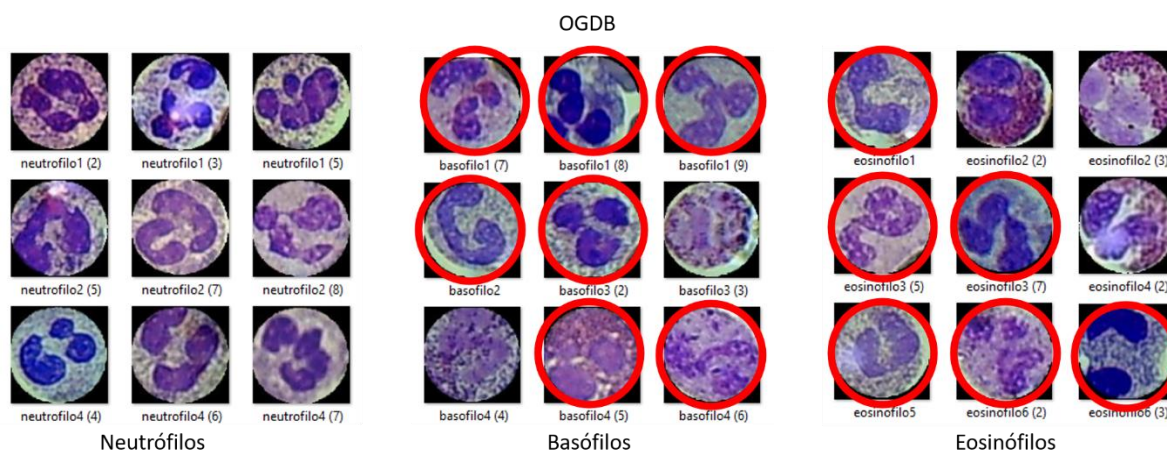


Figura 4.7 Ejemplos de imágenes mal etiquetadas de OGDB.

Dadas las situaciones anteriores, es necesario realizar el proceso de segmentación nuevamente con los GT reetiquetados obteniendo la nueva base de datos OGDB de la cual se parte para generar una nueva base de datos MASTER (anteriormente PREMASTERDB). La nueva base de datos será utilizada para realizar nuevos experimentos para la selección de modelos. Los resultados de ello se describen en la siguiente sección.

4.7 Nuevos entrenamientos de modelos para selección

Es necesario repetir los experimentos planteados en la sección 4.5 debido a los datos que se encontraban mal etiquetados. El proceso de entrenamiento se conserva como se indicó en la sección 4.5, solamente se agrega la capa de aumento de datos dinámico descrita en la Figura 4.6 a los modelos ResNet50, VGG16 y AlexNet; y ahora se utiliza la base de datos MASTER que proviene de la base de datos PREMASTERDB que surge del reetiquetado de los datos.

Anteriormente se planteaban 4 experimentos por modelo variando la base de datos entre OGDB y MASTER, y variando la semilla de lectura de la base de datos para llevar a cabo la selección de modelos. Ahora a parte se propone realizar la validación cruzada de la base de datos MASTER en cada modelo seleccionado. En la Figura 4.8 se muestra un resumen del nuevo proceso de selección de modelos.

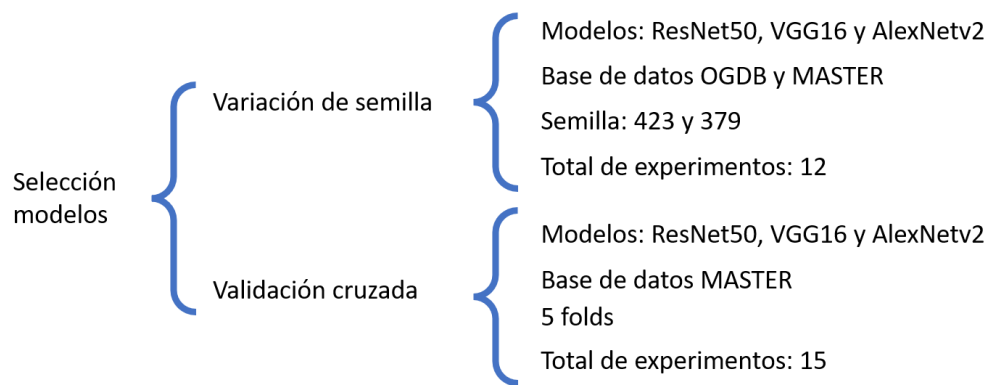


Figura 4.8 Nuevo proceso de selección de modelos.

Considerando el nuevo proceso de selección de modelos se presentan dos tipos de resultados de entrenamiento: uno para la variación de semilla y el segundo para la validación cruzada. En la Tabla 4.7 se muestran los resultados de variación de semilla. Se puede apreciar

como al considerar el valor de desempeño BAM, que el modelo de AlexNetv2 supera a los modelos ResNet50 y VGG16 en ambas bases de datos y con ambas semillas de lectura utilizadas obteniendo un BAM de hasta el 98.17%. Este porcentaje es considerado alto y competitivo dentro del estado del arte. También se aprecia como al utilizar la base de datos MASTER siempre se obtiene un desempeño mayor que cuando se utiliza OGDB en todos los modelos por lo que se puede concluir que hasta este punto si está apoyando el aumento de datos con GDAM-GAN a mejorar el desempeño de los modelos profundos.

Tabla 4.7 Resultados de variación de semilla.

Modelo	Base datos	Seed	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
<u>AlexNetv2</u>	<u>MASTER</u>	<u>423</u>	<u>99.46%</u>	<u>97.84%</u>	<u>0.0162</u>	<u>0.0762</u>	<u>98.17%</u>
AlexNetv2	MASTER	379	100.00%	96.98%	0.0028	0.1046	97.72%
VGG16	MASTER	423	96.45%	96.98%	0.1478	0.2303	95.01%
VGG16	MASTER	379	95.05%	96.55%	0.2098	0.194	94.72%
ResNet50	MASTER	379	97.42%	96.55%	0.1081	0.1285	91.26%
ResNet50	MASTER	423	90.01%	95.25%	0.3923	0.1493	88.08%
<u>AlexNetv2</u>	<u>OGDB</u>	<u>423</u>	<u>98.43%</u>	<u>95.49%</u>	<u>0.0407</u>	<u>0.1698</u>	<u>92.40%</u>
AlexNetv2	OGDB	379	93.95%	93.69%	0.1641	0.235	91.80%
VGG16	OGDB	423	94.85%	91.89%	0.1717	0.4978	87.63%
ResNet50	OGDB	379	97.98%	94.59%	0.0344	0.1811	84.60%
VGG16	OGDB	379	93.95%	93.69%	0.2396	0.3039	82.69%
ResNet50	OGDB	423	96.19%	90.99%	0.1251	0.3478	73.49%

La Tabla 4.8 presenta los resultados obtenidos de realizar la validación cruzada del modelo ResNet50 considerando el 80% de los datos para entrenamiento y 20% para

validación. Al ser un 20% de los datos para validación esto indica que se generan 5 *folds* para la validación cruzada y por lo tanto 5 resultados. En la

Tabla 4.9 se muestran los resultados de validación cruzada para el modelo VGG16 y en la

Tabla 4.10 para el modelo AlexNetv2. Finalmente, en la Tabla 4.11 se recopilan los resultados de las validaciones cruzadas de cada modelo y se observa como en el desempeño de BAM vuelve a sobre pasar el modelo de AlexNetv2 a ResNet50 y VGG16 con un valor de 95.03% de promedio de los *folds*.

También cabe recordar que al realizar la validación cruzada se comprueba que los entrenamientos de la etapa de variación de semilla no estén dando un buen resultado debido a un sesgo en los datos. En conclusión, el modelo de AlexNetv2 obtuvo un mejor desempeño de clasificación con la métrica BAM en ambos procesos de selección y por ende es el modelo seleccionado para futuro análisis dentro del trabajo de tesis.

Tabla 4.8 Resultados validación cruzada modelo ResNet50.

Modelo	Base datos	Kfold	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
ResNet50	MASTER	0	97.09%	98.71%	0.1231	0.0422	96.95%
ResNet50	MASTER	1	99.03%	96.99%	0.0504	0.4039	90.54%
ResNet50	MASTER	2	99.46%	96.56%	0.0213	0.3782	95.18%
ResNet50	MASTER	3	96.99%	95.25%	0.2272	0.6231	88.96%
ResNet50	MASTER	4	95.81%	96.98%	0.3124	0.2667	92.54%

Tabla 4.9 Resultados validación cruzada modelo VGG16.

Modelo	Base datos	Kfold	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
VGG16	MASTER	0	94.62%	96.13%	0.2225	0.2354	89.66%
VGG16	MASTER	1	98.92%	94.84%	0.0846	0.6924	88.95%
VGG16	MASTER	2	97.20%	95.27%	0.2265	0.4703	86.64%
VGG16	MASTER	3	92.91%	96.12%	1.1821	0.9476	91.85%
VGG16	MASTER	4	98.92%	97.41%	0.0769	0.5427	95.17%

Tabla 4.10 Resultados validación cruzada modelo AlexNetv2.

Modelo	Base datos	Kfold	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
AlexNetv2	MASTER	0	98.60%	98.71%	0.0509	0.0846	96.47%
AlexNetv2	MASTER	1	98.49%	96.56%	0.0402	0.0971	92.40%
AlexNetv2	MASTER	2	98.06%	96.56%	0.0494	0.1305	96.69%
AlexNetv2	MASTER	3	99.89%	97.84%	0.006	0.1024	95.65%

AlexNetv2	MASTER	4	97.74%	95.68%	0.0527	0.1484	93.93%
-----------	--------	---	--------	--------	--------	--------	--------

Tabla 4.11 Promedio de resultados de validación cruzada de modelos AlexNetv2, ResNet50 y VGG16.

Modelo	Base datos	Kfold	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
AlexNetv2	MASTER	Promedio	98.56%	97.07%	0.03984	0.1126	95.03%
ResNet50	MASTER	Promedio	97.68%	96.90%	0.14688	0.34282	92.83%
VGG16	MASTER	Promedio	96.51%	95.95%	0.35852	0.57768	90.45%

Hasta este punto se ha seleccionado el modelo de AlexNetv2 como mejor modelo de clasificación, pero se debe recordar que los experimentos utilizaron la base de datos MASTER que consta con el 52% de muestras que son artificiales y el 48% que son reales. Entonces se abre la duda si realmente el aumento de datos está ayudando a la generalización del modelo o solamente se está haciendo una memorización de los datos artificiales. Para resolver dicha duda se emplea la base de datos MASTER_CHALLENGE. Se plantea realizar un entrenamiento utilizando esta base de datos que está dividida en un conjunto de entrenamiento y validación, en el cual, el último contiene solamente muestras reales de leucocitos. Esto funcionaría como una simulación para el modelo de lo que se encontraría en un ámbito real de trabajo y para conocer su desempeño real y responder la pregunta si realmente las muestras artificiales están ayudando al desempeño del modelo. Para este entrenamiento con MASTER_CHALLENGE se consideran los mismos hiperparámetros, pero solo se realiza en el modelo seleccionado de AlexNetv2.

Tabla 4.12 Resultados experimento muestras GDAM-GAN.

Modelo	Base datos	Train_acc	Val_acc	Train_loss	Val_loss	Val_BAM
AlexNetv2	MASTER*	99.46%	97.84%	0.0162	0.0762	98.17%
AlexNetv2	MASTER_CHALLENGE	99.67%	97.41%	0.0166	0.0827	97.77%
AlexNetv2	MASTER**	98.06%	96.56%	0.0494	0.1305	96.69%

* Variación de semilla, con semilla 423.

** Fold #2 de validación cruzada.

En la Tabla 4.12 se observa el resultado de dicho experimento comparado con los resultados más altos de las dos etapas de selección del modelo: con variación de semilla y

validación cruzada. Se muestran los resultados de la validación cruzada del *fold* 2 con la base de datos MASTER de AlexNetv2, de la variación de semilla con la semilla 423 utilizando la base de datos MASTER y los resultados del experimento propuesto con MASTER_CHALLENGE. Se observa como el experimento propuesto mantiene un desempeño de BAM de 97.77%, similar a los obtenidos en los otros dos experimentos. Por ende, se puede concluir que las muestras artificiales si están apoyando a una generalización del modelo AlexNetv2 y que no existe un sesgo evidente a que el modelo memorice las muestras artificiales.

4.8 Comparación de desempeño con estado del arte

Durante la experimentación se obtuvieron distintos modelos de AlexNetv2 con altos desempeños de clasificación basados en la métrica de BAM: variación de semilla con 98.17%, MASTER_CHALLENGE con 97.77% y validación cruzada con 96.69%. Estos modelos ya presentan un desempeño de clasificación competitivo con los presentes en el estado del arte. En la Tabla 4.13 se muestran dichos resultados comparados con algunos trabajos que también resuelven la clasificación de leucocitos. Cabe aclarar que los modelos mencionados utilizan bases de datos distintas y algunos llegan a clasificar distintas cantidades de leucocitos a las presentes en este trabajo.

Tabla 4.13 Comparación de modelos AlexNetv2 con la literatura.

Año	Modelo de clasificación	Val_BAM
2019	AlexNet and Custom CNN [20]	98.96%
2020	AlexNet [16]*	98.46%
2021	AlexNet-GoogleNet-SVM [21]	98.25%
2023	<u>AlexNetv2 variación de Semilla</u>	<u>98.17%</u>
2023	<u>AlexNetv2 MASTER_CHALLENGE</u>	<u>97.77%</u>
2022	VGG16 [48]*	97.20%
2021	ResNet50, DenseNet121, Custom CNN [15]	96.73%
2023	<u>AlexNetv2 validación cruzada</u>	<u>96.69%</u>
2022	ResNet50, DenseNet, SCAM [22]	96.38%
2020	WBCNet Custom CNN [50]	96.29%
2021	CCA, Xception [18]	95.82%
2021	ARML [19]*	95.30%
2021	EIT, VGG16 [54]	95%
2020	ResNet34 [52]*	91.80%
2019	WBCsNet, AlexNet [51]	91.57%
2020	O-DA statistical model [57]*	91.32%

2020	ResNet50, TIGDA [58]	-
------	----------------------	---

* Trabajos que utilizan solo un modelo profundo para la clasificación de leucocitos.

Al comparar los modelos obtenidos de AlexNetv2 con los trabajos del estado del arte se observa como el mejor modelo de AlexNetv2 con 98.17% de BAM es el cuarto mejor modelo comparado con los modelos reportados en la literatura con una variación de 0.79% del primero lugar. Y es el segundo lugar considerando trabajos que utilicen solamente un modelo profundo para realizar la clasificación con una diferencia de 0.29% del primer lugar. Es importante este último punto, ya que en esta investigación no se desea utilizar un modelo profundo que sea muy complejo con más módulos y que obtenga el 100% de desempeño, si no que el fin es implementarlo en un sistema embebido con una menor cantidad de recursos por lo que es esencial que sea un solo modelo.

4.9 Implementación de modelo en sistema embebido

En la sección anterior se menciona como se obtiene un modelo competitivo con el estado del arte con el modelo de AlexNetv2. A partir de ese punto es posible considerar su implementación en un sistema embebido. Para ello, se utiliza Open Neural Network Exchange (ONNX), que es un formato de acceso libre para representar modelos de aprendizaje de máquina en distintos ambientes de trabajo. Esto lo logra definiendo un conjunto común de operadores y bloques de construcción de modelos en un mismo tipo de archivo para permitir a los desarrolladores de inteligencia artificial utilizar sus modelos en distintas plataformas. ONNX es compatible con una gran variedad de plataformas, pero en esta ocasión el trabajo se enfoca en la compatibilidad entre TensorFlow y NVIDIA. Esto es porque el diseño y entrenamiento de los modelos profundos presentados anteriormente de ResNet50, VGG16 y AlexNetv2 fueron hecho en Python utilizando TensorFlow y se desean implementar en un sistema embebido NVIDIA Jetson Nano. La Jetson es compatible con Python y el formato ONNX de modelos profundos. Sin embargo, es necesario realizar una conversión de los modelos clasificadores diseñados en TensorFlow a ONNX para poder ser interpretado y funcional en la Jetson Nano. Para esto se toman los 3 modelos con más alto desempeño

presentes en la Tabla 4.12. Estos son convertidos al formato ONNX y se implementan en PC y en el sistema embebido Jetson Nano. La PC cuenta con: un procesador Intel Xeon E5620 @ 2.40GHz, 15GB de RAM y una tarjeta gráfica NVIDIA GeForce RTX 3060. La Jetson Nano cuenta con: un procesador Quad-core ARM® Cortex®-A57 MPCore, 4GB de RAM y una tarjeta gráfica NVIDIA Maxwell arquitectura con 128 NVIDIA CUDA núcleos 0.5 TFLOPs (FP16). Ya implementados se midieron desempeños después de la conversión al formato ONNX y los tiempos de ejecución. En la Tabla 4.14 se muestran los valores de validación de BAM que mostraron los 3 modelos cuando fueron entrenados, cuando se corren inferencias en la PC en formato ONNX y cuando se corren inferencias en la Jetson Nano en formato ONNX. Se observa una ligera variación de 0.52% en uno de los modelos por el cambio de formato a ONNX que puede deberse al tipo de dato cuando se cambia de formato, pero es prácticamente insignificante.

Tabla 4.14 Resultados de validaciones de ONNX.

Base datos	Val. BAM modelo entrenado	Val. BAM formato ONNX en PC	Val. BAM formato ONNX en Jetson
MASTER semilla 423	98.17%	98.17%	98.17%
MASTER_CHALLENGE	97.77%	98.29%	98.29%
MASTER crossvalidation	96.69%	96.69%	96.69%

Luego en la Tabla 4.15 se muestran los tiempos de procesamiento obtenidos al utilizar los 3 mejores modelos en formato ONNX en la PC y en la Jetson Nano. Ambas tablas indican cual fue el tiempo total del programa en procesar 232 inferencias que corresponden al número de muestras del conjunto de validación, el tiempo promedio de cada lectura de imagen, el tiempo de cargar el modelo profundo y el tiempo promedio de cada inferencia. Se puede notar como en los 3 modelos se mantiene un tiempo de procesamiento similar en la inferencia cuando se utiliza en PC o Jetson Nano. Pero se nota una diferencia grande cuando se compara el tiempo de inferencia entre PC y Jetson Nano, teniendo la PC una reducción de 93.65% del tiempo por inferencia comparado con la Jetson. Sin embargo, 190 ms. por inferencia es un tiempo relativamente pequeño en un sistema embebido comparado al tiempo de procesamiento en una PC.

Tabla 4.15 Tiempos de procesamiento de modelos ONNX.

PC				
Base datos	T. Programa (ms)	T. Lectura imagen (ms)	T. Lectura del modelo (ms)	T. inferencia (ms)
MASTER semilla 423	5820.0805	7.1441	1453.2418	11.6784
MASTER_CHALLENGE	6161.2594	7.0480	1730.0097	11.9847
MASTER crossvalidation	4412.6045	4.7415	283.3232	12.9136
Jetson Nano				
Base datos	T. Programa (ms)	T. Lectura imagen (ms)	T. Lectura del modelo (ms)	T. inferencia (ms)
MASTER_423	48817.2938	8.3849	3137.3195	188.4186
MASTER_CHALLENGE	49213.1688	8.2645	3261.5587	189.7194
MASTER crossvalidation	46131.9789	5.0488	937.2715	188.8435

Hasta este punto se ha logrado implementar un modelo profundo AlexNetv2 en un sistema embebido Jetson Nano con un bajo tiempo de procesamiento. Sin embargo, se desea realizar un estudio para realizar una búsqueda de arquitectura de AlexNetv2 para encontrar una versión de modelo que tenga un desempeño de clasificación similar al reportado anteriormente, pero con una menor cantidad de parámetros y tiempo de procesamiento teniendo en consideración que se trabaja en un entorno de sistema embebido con recursos limitados. En el siguiente capítulo se detalla un poco más acerca de la búsqueda de arquitectura.

5 BÚSQUEDA DE ARQUITECTURA REDUCIDA

La búsqueda de arquitectura es un área dentro del aprendizaje de máquina que tiene distintos fines, uno de ellos es buscar una nueva arquitectura que requiera un menor costo computacional que una arquitectura base. Esto es muy útil cuando se considera que el uso final de una arquitectura será en un sistema embebido, lo cual es el caso de esta tesis. Por ello, este capítulo describe como se parte de una arquitectura base de AlexNetv2 como espacio de búsqueda para encontrar una arquitectura que mantenga un desempeño similar a los reportados en el capítulo 4 y que requiera un menor costo computacional para realizar un tiempo de inferencia menor.

5.1 Variación de capas convolucionales y tamaño de capas totalmente conectadas para la búsqueda de la reducción de AlexNetv2

Para encontrar un modelo reducido se utilizan búsquedas de arquitectura reducida, se denomina búsqueda de arquitectura reducida porque no se utilizarán los conceptos principales de *Neural Architecture Search* (NAS) como son el aprendizaje de refuerzo y la optimización bayesiana [23]. Para la finalidad de este trabajo de tesis se propone un espacio de búsqueda reducido O variando algunos hiperparámetros y se prueban todas las posibles combinaciones con entrenamientos para al final decidir cual tiene un desempeño al menos similar o mejor al obtenido anteriormente y que reduzca la cantidad de parámetros necesarios para implementar el modelo.

Se define un espacio de búsqueda dado por el modelo AlexNetv2 que se compone de 5 capas convolucionales y 2 capas totalmente conectadas de 4096 neuronas cada una. O abarca la variación del número de capas convolucionales $NConv$ (5, 4, 3, 2, 1), el número de capas totalmente conectadas NFC (2, 1) y la cantidad de neuronas en las capas totalmente conectadas $NNFC$ (4096, 2048 o 1024). Se propone este espacio O basado en la investigación mostrada en el capítulo 1 sección 4. De esta forma, se considera la estrategia de búsqueda B como todas las posibles combinaciones del espacio de búsqueda O con razón

de aprendizaje dinámico α_{din} , 200 épocas E , entropía cruzada CE como función de costo J y 60 experimentos a realizar en la búsqueda de arquitectura considerando todas las posibles combinaciones del espacio de búsqueda mostrados en la Tabla 5.1 ordenados de mayor a menor considerando la estrategia para medir desempeño P con la métrica de BAM, siendo todos estos experimentos entrenados con la base de datos MASTER_CHALLENGE. En la tabla se muestran las capas convolucionales activas del modelo AlexNetv2 con una marca $\checkmark$ y las inactivas con $\times$, para las capas totalmente conectadas se marcan de igual forma las capas inactivas y las activas con la cantidad de neuronas que la componen (4096, 2048 o 1024) cuyos valores se reducen al 50% y se seleccionó arbitrariamente dicho valor. También se muestra el valor del desempeño del conjunto de validación del modelo medido con BAM y la cantidad total de parámetros que componen al modelo.

Tabla 5.1 Experimentos de búsqueda de arquitectura reducida en PC.

# Exp	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 1	FC 2	Val. BAM	Total param.
12	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	4096	1024	97.77%	63,691,269
27	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	1024	1024	97.37%	16,747,653
57	$\checkmark$	$\checkmark$	$\times$	$\times$	$\times$	1024	$\times$	97.28%	38,405,893
16	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	1024	4096	96.53%	21,239,301
46	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	4096	$\times$	96.26%	41,521,925
36	$\checkmark$	$\checkmark$	$\times$	$\times$	$\times$	1024	1024	96.12%	39,455,493
15	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	2048	1024	95.76%	33,280,517
17	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	1024	2048	95.76%	19,129,861
26	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	1024	2048	95.72%	17,802,373
9	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	1024	1024	95.58%	14,241,541
10	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	4096	4096	95.56%	76,292,613
22	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	2048	4096	95.53%	38,262,917
3	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	4096	1024	95.40%	45,701,893
48	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	1024	$\times$	95.40%	13,191,941
25	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	1024	4096	95.36%	19,911,813
18	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	1024	1024	95.28%	18,075,141
13	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	2048	4096	95.27%	39,590,405
47	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	2048	$\times$	95.20%	22,635,269
14	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	2048	2048	95.07%	35,383,813
6	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	2048	1024	95.00%	24,728,325
4	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	2048	4096	94.97%	31,038,213
50	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	2048	$\times$	94.91%	31,187,461
54	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	1024	$\times$	94.80%	15,698,053
49	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	4096	$\times$	94.77%	59,511,301
28	$\checkmark$	$\checkmark$	$\times$	$\times$	$\times$	4096	4096	94.59%	168,451,845
24	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	2048	1024	94.58%	31,953,029
20	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	$\times$	4096	2048	94.32%	66,564,229
5	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	2048	2048	94.11%	26,831,621

* $\checkmark$ Capa activa.

** $\times$ Capa inactiva.

Tabla 5.1 Experimentos de búsqueda de arquitectura reducida en PC, continuación.

# Exp	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 1	FC 2	Val. BAM	Total param.
21	✓	✓	✓	✗	✗	4096	1024	94.03%	62,363,781
19	✓	✓	✓	✗	✗	4096	4096	94.01%	74,965,125
37	✓	✗	✗	✗	✗	4096	4096	93.81%	282,655,237
32	✓	✓	✗	✗	✗	2048	2048	93.74%	80,357,125
1	✓	✓	✓	✓	✓	4096	4096	93.38%	58,303,237
8	✓	✓	✓	✓	✓	1024	2048	93.38%	15,296,261
31	✓	✓	✗	✗	✗	2048	4096	92.69%	84,563,717
2	✓	✓	✓	✓	✓	4096	2048	92.54%	49,902,341
7	✓	✓	✓	✓	✓	1024	4096	92.34%	17,405,701
53	✓	✓	✓	✗	✗	2048	✗	92.33%	29,859,973
23	✓	✓	✓	✗	✗	2048	2048	91.93%	34,056,325
52	✓	✓	✓	✗	✗	4096	✗	91.86%	58,183,813
56	✓	✓	✗	✗	✗	2048	✗	91.72%	76,160,773
51	✓	✓	✓	✓	✗	1024	✗	91.18%	17,025,541
34	✓	✓	✗	✗	✗	1024	4096	91.08%	42,619,653
58	✓	✓	✗	✗	✗	4096	✗	91.01%	265,873,925
33	✓	✓	✗	✗	✗	2048	1024	90.76%	78,253,829
11	✓	✓	✓	✓	✗	4096	2048	90.00%	67,891,717
35	✓	✓	✗	✗	✗	1024	2048	89.51%	40,510,213
39	✓	✗	✗	✗	✗	4096	1024	89.44%	270,053,893
55	✓	✓	✗	✗	✗	4096	✗	88.57%	151,670,533
29	✓	✓	✗	✗	✗	4096	2048	88.19%	160,050,949
40	✓	✗	✗	✗	✗	2048	4096	86.36%	141,357,573
30	✓	✓	✗	✗	✗	4096	1024	86.17%	155,850,501
42	✓	✗	✗	✗	✗	2048	1024	82.18%	135,047,685
38	✓	✗	✗	✗	✗	4096	2048	81.85%	274,254,341
45	✓	✗	✗	✗	✗	1024	1024	80.67%	67,544,581
43	✓	✗	✗	✗	✗	1024	4096	79.64%	70,708,741
59	✓	✓	✗	✗	✗	2048	✗	79.26%	132,954,629
41	✓	✗	✗	✗	✗	2048	2048	76.67%	137,150,981
44	✓	✗	✗	✗	✗	1024	2048	71.25%	68,599,301
60	✓	✓	✗	✗	✗	1024	✗	71.08%	66,494,981

* ✓ Capa activa.

** ✗ Capa inactiva.

Al observar los resultados de los 60 experimentos de la Tabla 5.1, se muestran desempeños de validación con la métrica BAM desde 71.08% hasta el 97.77%. Los resultados más altos demuestran competencia en comparación con los que se obtuvieron en el capítulo 4 sección 8. En la siguiente sección se describe como se implementan los mejores modelos de la Tabla 5.1 en el sistema embebido Jetson Nano.

5.2 Implementación de modelos reducidos en sistema embebido

De los resultados obtenidos en la sección anterior, se seleccionan los 3 modelos obtenidos de la búsqueda de arquitectura con mejor desempeño de BAM ilustrados en la Tabla 5.2, el

más alto de ellos tiene un desempeño del 97.77% que es igual al obtenido con el modelo base de AlexNetv2, pero tiene una capa convolucional menos. Los 3 modelos se consideran competitivos y con un desempeño similar a los obtenidos en experimentos anteriores del capítulo 4 sección 8. Para llevar a cabo la selección del mejor modelo encontrado se convierten a formato ONNX para su implementación y prueba en el sistema embebido Jetson Nano y se compara su implementación en la PC, los resultados se muestran en la Tabla 5.3.

Tabla 5.2 Modelos de búsqueda de arquitectura reducida con mejores desempeños de BAM.

# Exp	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 1	FC 2	Val. BAM	Total param.
12	✓	✓	✓	✓	✗	4096	1024	97.77%	63,691,269
27	✓	✓	✓	✗	✗	1024	1024	97.37%	16,747,653
57	✓	✓	✗	✗	✗	1024	✗	97.28%	38,405,893

Tabla 5.3 Tiempos de procesamiento de mejores modelos de búsqueda de arquitectura reducida.

PC				
# Experimento	Tiempo ejecución programa (ms)	Tiempo lectura imagen (ms)	Tiempo lectura del modelo (ms)	Tiempo inferencia (ms)
12	12903.1913	3.6730	708.9605	48.8279
27	9733.4020	4.0404	185.0066	37.0733
57	8186.5129	3.6825	435.0113	29.6601
Jetson Nano				
# Experimento	Tiempo ejecución programa (ms)	Tiempo lectura imagen (ms)	Tiempo lectura del modelo (ms)	Tiempo inferencia (ms)
12	42263.0474	4.9812	1340.3465	171.3334
27	26292.1419	5.0044	254.1875	107.1505
57	25232.1174	5.0158	794.0547	100.2434

Al analizar la Tabla 5.3 se observa lo esperado, como el procesamiento es más rápido en la PC que en el sistema embebido con una disminución alrededor del 65%-70%. Sin embargo, se muestra información más interesante, ya que se esperaría que por la cantidad de parámetros el experimento #12 fuese más alto el tiempo de inferencia seguido del experimento #57 y por último el #27. Pero resulta que el tiempo de inferencia menor lo tiene el modelo #57 que tiene 38M de parámetros, después el modelo #27 de 16M y al final el modelo #12 de 63M. Esto indica que la cantidad de parámetros no solamente está directamente relacionado al tiempo de inferencia. Realizando un análisis de la información mostrada en la Tabla 5.2, se observa que

el tiempo de inferencia está más relacionado a la cantidad de capas del modelo. Al reducir la cantidad de capas convolucionales se genera un aumento en la dimensionalidad de los datos conectados a las capas totalmente conectadas, por lo que menor cantidad de capas convolucionales no implicará que se reduzca el tiempo de inferencia. Para hacerlo más evidente se muestra la Tabla 5.4. Entre menor cantidad de capas tiene el modelo de AlexNetv2, más rápida resulta su inferencia sin relacionar la cantidad de parámetros.

Tabla 5.4 Comparación de tiempo de inferencia y arquitectura del modelo.

# Exp	# Capas Conv.	# capas FC	Tiempo inferencia Jetson	Val. BAM	Total param.
12	4	2	171.3334	97.77%	63,691,269
27	3	2	107.1505	97.37%	16,747,653
57	2	1	100.2434	97.28%	38,405,893

Considerando el modelo obtenido en el experimento #57 y comparándolo con el modelo obtenido en la selección de modelos entrenado con la base de datos MASTER_CHALLENGE del capítulo 4 sección 7, se genera la Tabla 5.5. En esta tabla se puede observar cómo se consigue un modelo con la búsqueda de arquitectura que disminuye solamente un 0.49% comparado con el modelo original de AlexNetv2 entrenado con MASTER_CHALLENGE. Además, se observa una reducción significativa del tiempo de inferencia en un 48% con el modelo del experimento #57 encontrado con la búsqueda de arquitectura reducida en comparación con el modelo base de AlexNetv2.

Tabla 5.5 Comparación de tiempos y desempeño entre modelo base de AlexNetv2 y mejor modelo de búsqueda de arquitectura.

Jetson Nano						
Modelo	Parámetros	Val. BAM	Tiempo ejecución programa (ms)	Tiempo lectura imagen (ms)	Tiempo lectura del modelo (ms)	Tiempo inferencia (ms)
Exp. #57	38,405,893	97.28%	25232.1174	5.0158	794.0547	100.2434
MASTER_CHALLENGE	58,303,237	97.77%	49213.1688	8.2645	3261.5587	189.7194

De esta manera, el modelo diseñado final se muestra en la Figura 5.1 y se concluye de forma exitosa la búsqueda de un modelo que mantenga el desempeño de clasificación de leucocitos pero que disminuya el uso de recursos de un sistema embebido viéndose reflejado en el tiempo de procesamiento.

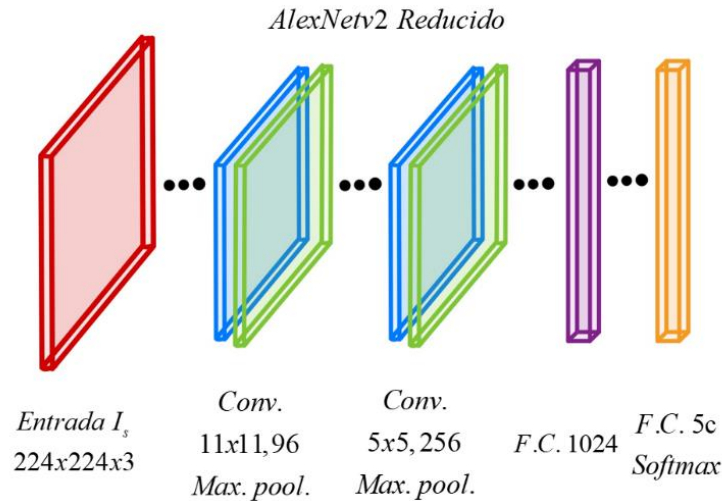


Figura 5.1 Arquitectura reducida del modelo AlexNetv2 experimento #57.

De esta manera se concluye el capítulo 5 de búsqueda de arquitectura reducida con el modelo AlexNetv2 #57 reducido en 3 capas convolucionales y 1 capa totalmente conectada con 1024 neuronas. Esta arquitectura demostró un desempeño del 97.28% con la métrica BAM, similar a los obtenidos en el capítulo 4 sección 8 y una mejora en el tiempo de procesamiento de inferencia con una reducción del 48% de tiempo por inferencia. En el siguiente capítulo se relata el diseño de una interfaz gráfica de usuario que utiliza el modelo #57 para realizar la clasificación de leucocitos.

6 DISEÑO DE INTERFAZ GRÁFICA DE USUARIO

En esta sección se describe cómo se diseña una Interfaz Gráfica de Usuario (GUI) para PC y Jetson Nano en la cual se implementa el modelo #57 obtenido en el capítulo 5 Búsqueda de arquitectura reducida. Esto con el fin de que pueda ser utilizado por usuarios para aprender a clasificar clases de leucocitos. Este capítulo se encuentra dividido en dos secciones en donde la primera describe el diseño de la GUI en una computadora, la segunda sección como se diseña la GUI para el sistema embebido Jetson Nano.

6.1 GUI para computadora

Se diseña una GUI para clasificar leucocitos de la base de datos MASTER_CHALLENGE con fines educativos y de formación para alumnos de medicina denominada ITCHLeukoAnalysis, en Anexos se puede encontrar el código fuente. La interfaz implementa el mejor modelo de la búsqueda de arquitectura reducida del capítulo 5 en formato ONNX utilizando Python en una PC con sistema operativo Windows con la librería PyInstaller. Dicha librería permite generar archivos ejecutables en Windows a partir de un archivo de Python, y se utiliza con el fin de eliminar la necesidad de hacer instalaciones para poder utilizar la GUI diseñada. La GUI introduce distintas funciones con botones para: cargar una nueva imagen, realizar la detección de una imagen de microscopio (deshabilitada por el momento), clasificar la imagen cargada a la interfaz, ajuste automático, configuración de brillo y contraste, aumento de nitidez, restaurar a valores predeterminados, guardar imagen y borrar imagen. En la Figura 6.1 se muestra la apariencia de la GUI.

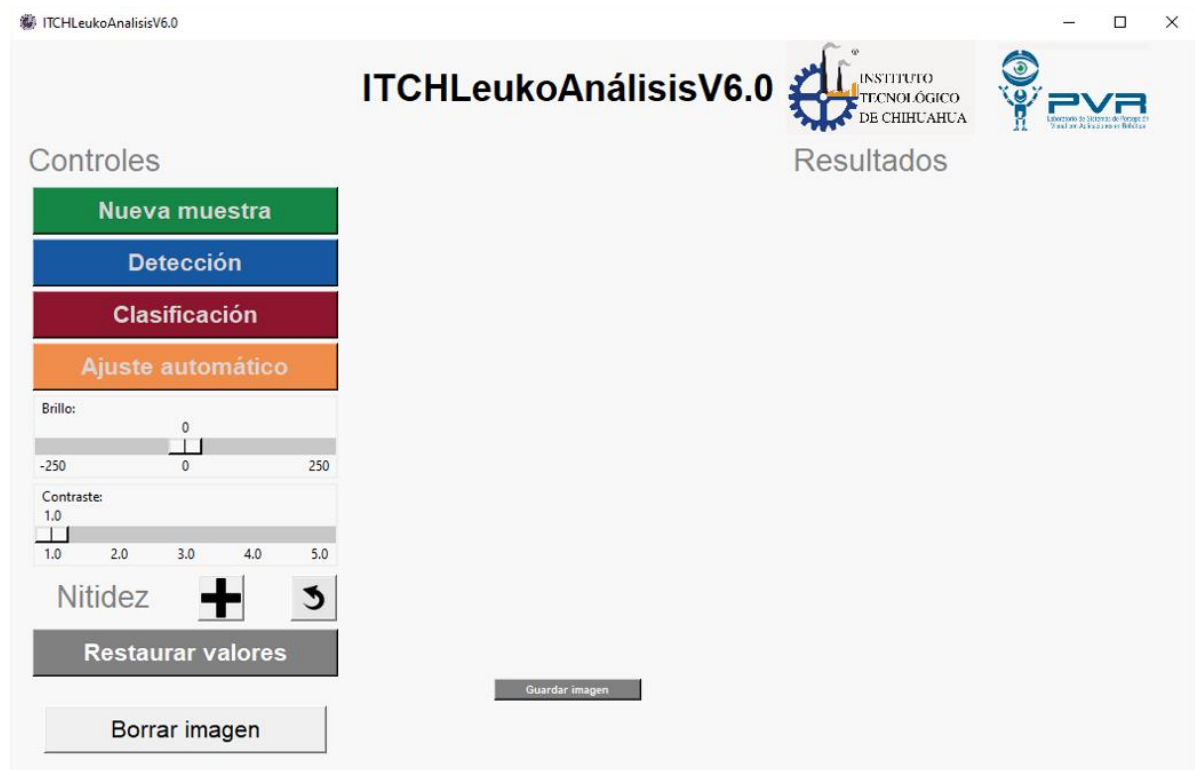


Figura 6.1 Apariencia de GUI ITCHLeukoAnálisisV6.0.

Las opciones de ajuste automático, configuración de brillo, configuración de contraste y nitidez se utilizan solamente para modificar la imagen de forma visual en la interfaz y que el usuario las modifique a su deseo. Pero estas modificaciones no afectan a la imagen que se ingresa al modelo de clasificación. En la Figura 6.2 se muestra como se ve una imagen cargada a la interfaz y en la Figura 6.3 un ejemplo de la misma muestra modificada visualmente para observar mejor los granulocitos de la célula utilizando los botones anteriormente mencionados pero que no afecta el resultado de la clasificación. En ambas figuras también se muestra como esta visible el nombre del archivo de la imagen cargada en la interfaz (en este caso eosinofilo4 (2).jpg).

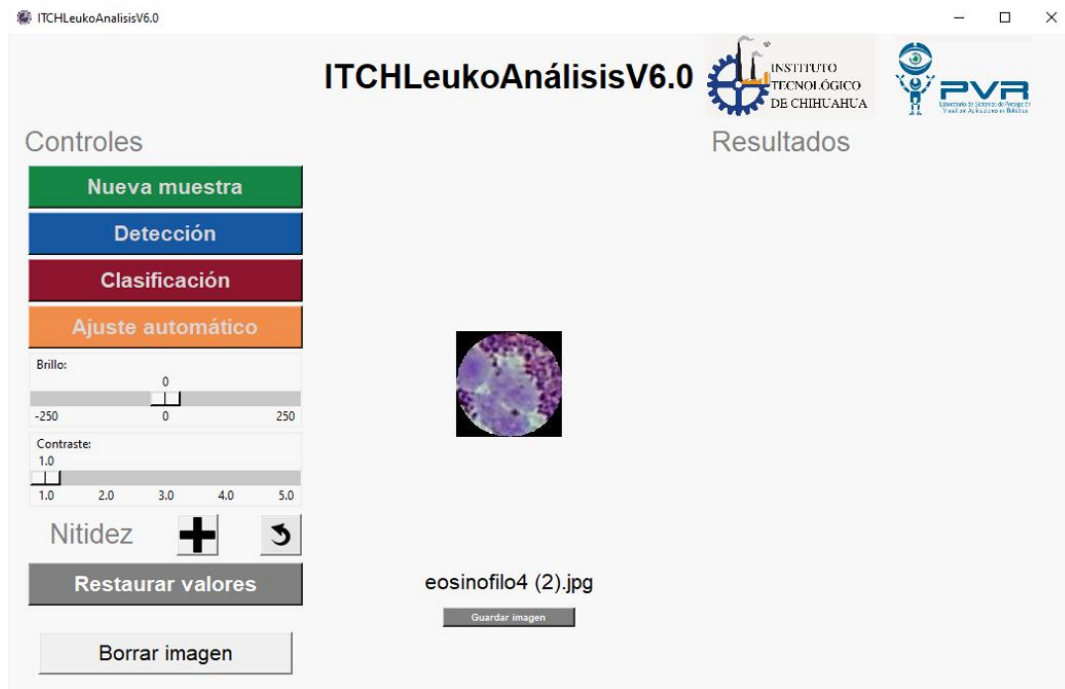


Figura 6.2 Ejemplo de imagen cargada a GUI.

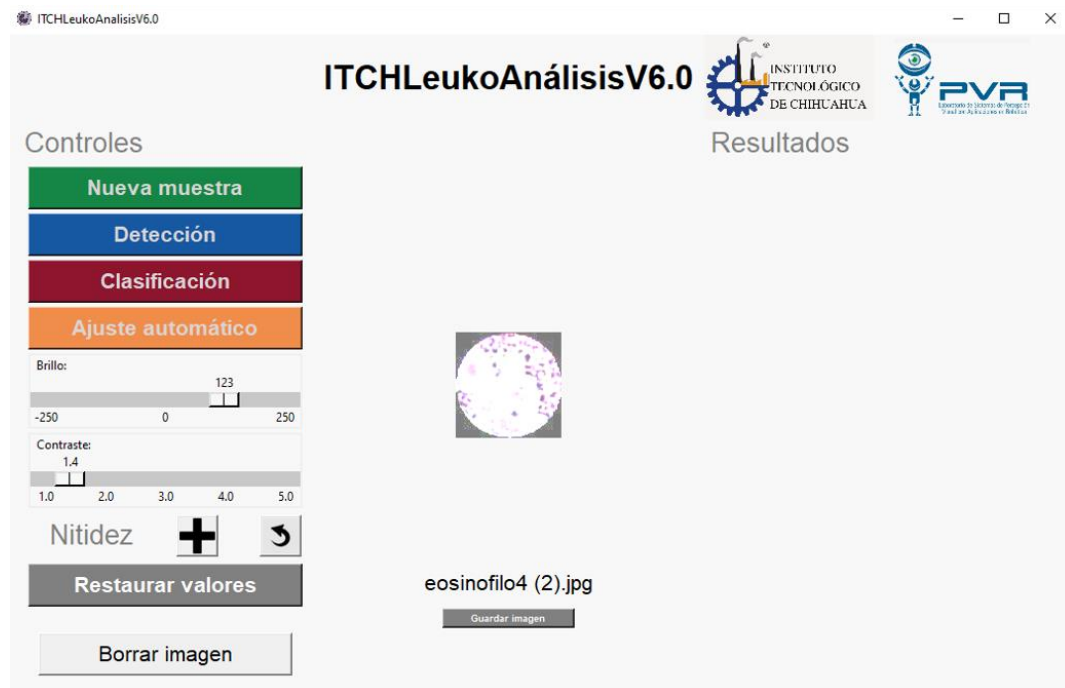


Figura 6.3 Ejemplo de imagen cargada a GUI y modificada por usuario.

Finalmente, en la Figura 6.4 se observa el resultado de una muestra clasificada, del lado derecho de la interfaz aparece el resultado de la clasificación que en el ejemplo mostrado es eosinófilo y se puede comprobar que es correcto con el nombre del archivo de la imagen que indica la clase a la que pertenece el leucocito.

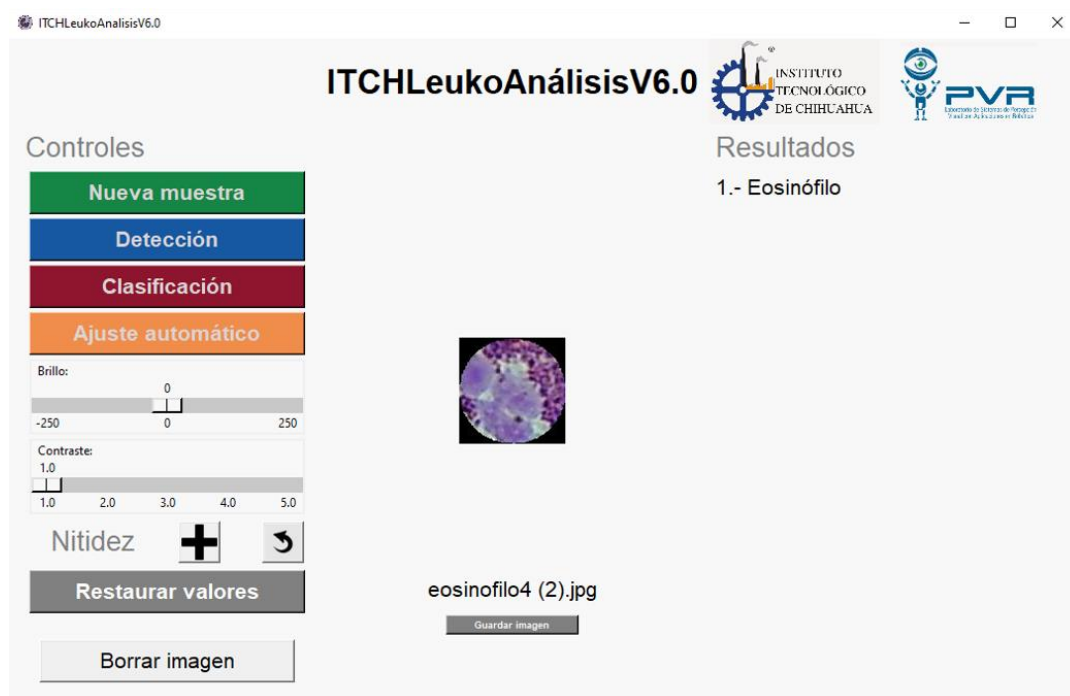


Figura 6.4 Ejemplo de resultado de GUI.

6.2 GUI para sistema embebido

El diseño de la GUI para el sistema embebido conserva el mismo formato que en PC. Pero en el caso del sistema embebido, a diferencia de la PC, no se puede generar un archivo ejecutable con PyInstaller por el sistema operativo Ubuntu que maneja la Jetson. Lo único que se puede hacer es un archivo de acceso directo al código fuente de la GUI que lo ayuda a funcionar como una aplicación. El archivo de acceso directo contiene información sobre: el nombre del acceso directo, código fuente a ejecutar, tipo de objeto a ejecutar, estado de terminal, ícono de acceso directo y dirección donde se encuentra el archivo. A continuación,

se muestran los datos contenidos en el archivo de acceso directo utilizado para la GUI ITCHLeukoAnalysis.

```
[Desktop Entry]
Name=LeukoAPP
Exec=python3 /home/lab02/Documents/DL/GUI-leuko_ARE/ITCHLeukoAnalysisV6.py
Type=Application
Terminal=false
Icon=/home/lab02/Documents/DL/GUI-leuko_ARE/leu.png
Path=/home/lab02/Documents/DL/GUI-leuko_ARE/
```

Con los datos del archivo anterior se genera un acceso directo de tipo aplicación que permite ejecutar el código fuente de la GUI. El acceso directo generado con su ícono se muestra en la Figura 6.5. Un ejemplo de clasificación de la GUI en el sistema embebido se muestra en la Figura 6.6.

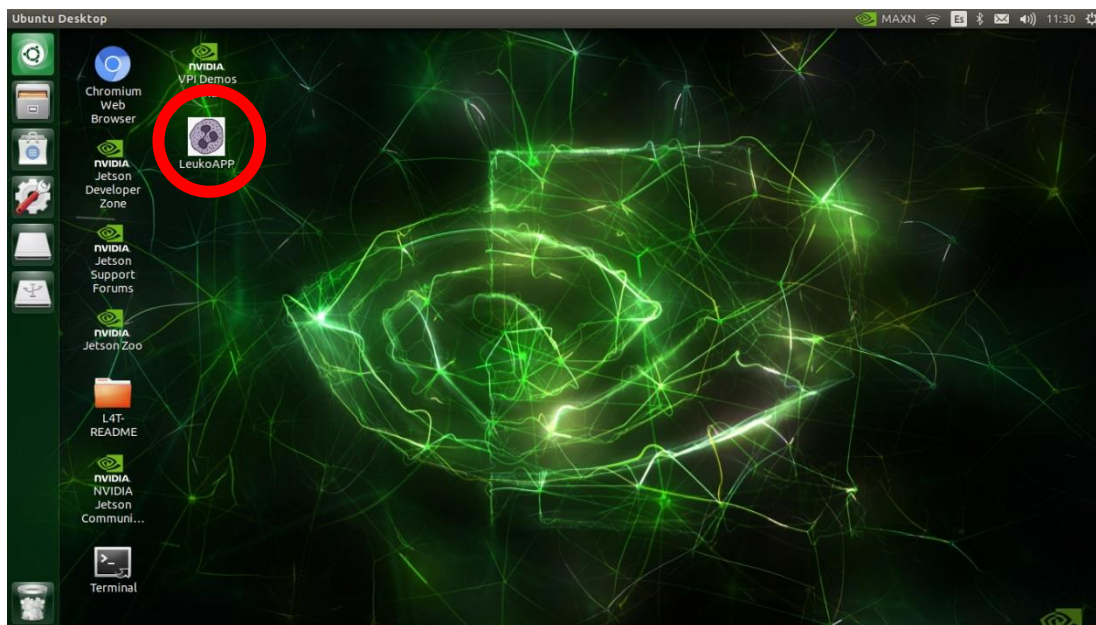


Figura 6.5 Acceso directo a GUI en sistema embebido.

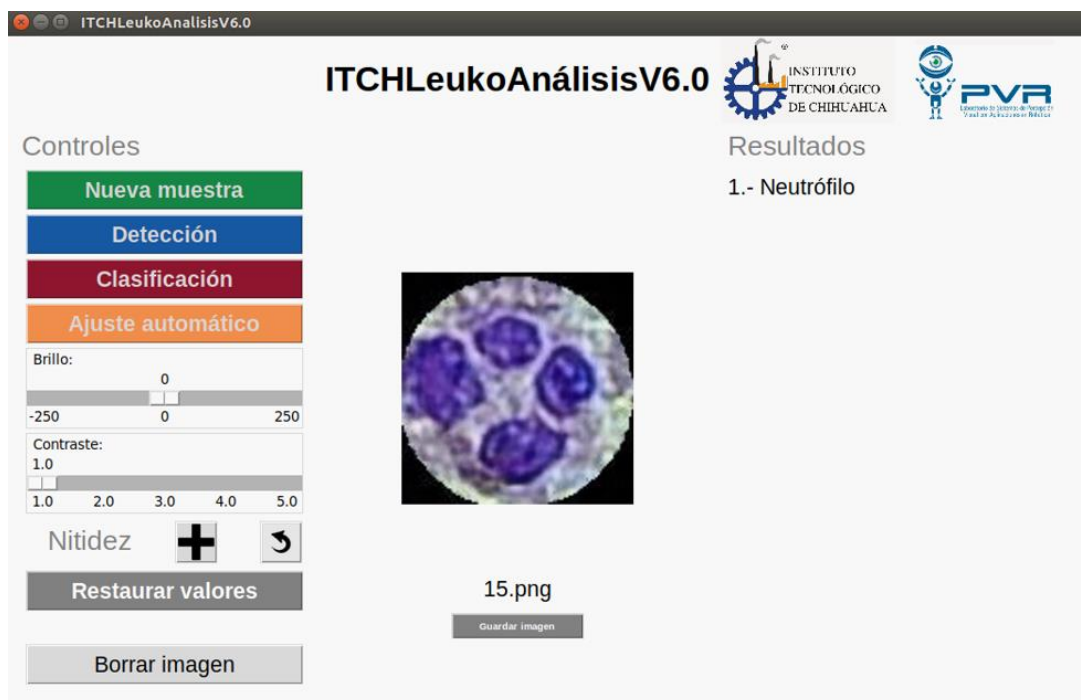


Figura 6.6 Ejemplo de clasificación de GUI en sistema embebido.

7 RESULTADOS Y CONCLUSIONES

En este trabajo se presenta una base de datos de imágenes de muestras de sangre periférica LEU_UACHDB. Esta contiene imágenes originales de la muestra del microscopio, imágenes etiquetadas manualmente y sus ground truths. Sin embargo, las etiquetas de los GT se encontraban mal etiquetadas y se corrigen reetiquetándolas manualmente con el conocimiento adquirido y el Atlas de Hematología Clínica [49]. Dicha base de datos no puede ser utilizada directamente para resolver la tarea de clasificación de leucocitos, por lo que se realiza una adaptación utilizando el algoritmo de detección diseñado por alumnos de ingeniería del ITCH. Posteriormente se diseña un algoritmo de segmentación que utiliza la salida del algoritmo de detección, la imagen original y el GT para realizar imágenes segmentadas de leucocitos formando la base de datos OGDB. Debido a la baja incidencia que presenta la base de datos OGDB se plantea una metodología de aumento de datos utilizando el método de GDAM-GAN. Se genera una base de datos MASTER para llevar a cabo el entrenamiento de modelos profundos. Son seleccionados del estado del arte modelos profundos que resuelvan la tarea de clasificación de leucocitos y se seleccionan los tres mejores de ellos (ResNet50, VGG16 y AlexNet). El modelo de AlexNet fue publicado para trabajar de forma paralela utilizando dos GPU's, por ello se propone una versión de AlexNet que trabaje de forma secuencial concatenando la arquitectura paralela en un modelo denominado AlexNetv2. Con los tres modelos seleccionados se plantea una metodología para realizar la selección del modelo de estudio. Esta metodología consta de 2 etapas: variación de semilla y validación cruzada. El modelo que obtenga el mejor desempeño de clasificación considerando la métrica BAM en ambas etapas es seleccionado, AlexNetv2. Durante los experimentos realizados para la selección de modelo con AlexNetv2 se obtienen resultados competitivos con el estado del arte para resolver la clasificación de leucocitos con un desempeño de hasta el 98.17% considerando la métrica BAM. Sin embargo, se propone realizar un experimento extra utilizando la base de datos MASTER_CHALLENGE para conocer el comportamiento de las muestras artificiales en el desempeño del modelo AlexNetv2. A pesar de que la base de datos MASTER cuenta con el 52% de muestras artificiales, el experimento realizado con MASTER_CHALLENGE y realizando la validación

del modelo únicamente con muestras reales, resulta en 97.77% de desempeño con BAM, similar a los obtenidos anteriormente en la metodología para la selección de modelos.

Al comparar los modelos obtenidos de la selección de modelos (variación de semilla y validación cruzada) y el experimento con MASTER_CHALLENGE del capítulo 4 sección 8, estos obtienen el 4° (variación de semilla), 5° (experimento con MASTER_CHALLENGE) y 8° (validación cruzada) lugar considerando la métrica de BAM. Pero el modelo de variación de semilla que obtuvo el 4° lugar tiene una diferencia solamente de 0.79% del primer lugar. Se debe considerar que el modelo de variación de semilla obtiene el segundo lugar considerando trabajos que resuelven la clasificación de leucocitos con un solo modelo profundo con una diferencia de 0.29% del primer lugar. Dichas comparaciones se pueden observar en la Tabla 4.13.

Una vez obtenidos los modelos de AlexNetv2 competitivos con el estado del arte se implementan en el sistema embebido Jetson Nano utilizando ONNX. Se miden los tiempos de procesamiento y resulta en un tiempo promedio de inferencia alrededor de 188 ms. Se desea que se utilice una menor cantidad de recursos y reducir el tiempo de inferencia para la implementación en el sistema embebido. Por ello se realiza la búsqueda de arquitectura reducida en donde se obtiene una arquitectura basada de AlexNetv2 que reduce el tiempo de inferencia en un 48% a 100 ms. cuando se implementa en el sistema embebido.

El modelo reducido luego es utilizado para su implementación en una interfaz gráfica de usuario con motivos educativos para la clasificación de leucocitos. Esta interfaz permite al usuario practicar sus conocimientos sobre las clases de leucocitos. Esta interfaz es diseñada para usarse en una computadora y en un sistema embebido Jetson Nano.

8 TRABAJO FUTURO

Como trabajo futuro, se plantean 4 puntos principales. El primero punto sería realizar un reemplazo de la etapa de detección presentada en este trabajo por una etapa que utilice aprendizaje profundo, ya que la actual detección utiliza técnicas tradicionales de procesamiento de imágenes para cumplir su cometido y no funciona muy bien con imágenes que presentan variación en el tinte de la muestra. Como segundo punto se desea utilizar el nuevo detector diseñado con aprendizaje profundo, y el clasificador obtenido para en conjunto crear un sistema capaz de realizar leucogramas de forma automática. Para ello, sería necesario agregar una última etapa después del clasificador que cuente los tipos de leucocitos encontrados por el detector. El tercer punto importante a realizar de trabajo futuro, sería implementar una automatización del método de frotis de sangre para que este no deba ser llevado de manera manual. Al automatizar este proceso (método de frotis de sangre, la detección y clasificación de leucocitos) permitiría obtención más rápida de las muestras para su análisis en el sistema mencionado en el segundo punto. Y, por último, el punto 4 sería utilizar todo el trabajo generado en los 3 puntos anteriores para diseñar un equipo completo para análisis de muestras de sangre periférica que realice leucogramas de manera automática y que sea un nuevo competidor de los equipos actuales del mercado.

9 REFERENCIAS

- [1] Lynch E.C., *Clinical Methods: The History, Physical, and Laboratory Examinations*, 3a ed. Boston: Butterworths, 1990.
- [2] E. A. Mohammed, M. M. A. Mohamed, B. H. Far, y C. Naugler, "Peripheral blood smear image analysis: A comprehensive review", *J Pathol Inform*, vol. 5, núm. 1, p. 9, 2014, doi: 10.4103/2153-3539.129442.
- [3] Miss. M. G. Bhamare y D.S. Patil, "Automatic Blood Cell Analysis by Using Digital Image Processing: A Preliminary Study", *International journal of engineering research and technology*, vol. 2, núm. 9, p. 964, 2013, doi: 10.1109/ICIAP.2003.1234129.
- [4] H. Cao, H. Liu, y E. Song, "A novel algorithm for segmentation of leukocytes in peripheral blood", *Biomed Signal Process Control*, vol. 45, pp. 10–21, 2018, doi: 10.1016/J.BSPC.2018.05.010.
- [5] A. Adewoyin y B. Nwogoh, "Peripheral blood film - A Review", *Ann Ib Postgrad Med*, vol. 12, núm. 2, p. 71, 2014.
- [6] N. Dong, M. die Zhai, J. fang Chang, y C. ho Wu, "A self-adaptive approach for white blood cell classification towards point-of-care testing", *Appl Soft Comput*, vol. 111, p. 107709, 2021, doi: 10.1016/j.asoc.2021.107709.
- [7] Taru Hays y Bette Jamieson, *Atlas of Pediatric Peripheral Blood Smears*, 1a ed. Abbott Laboratories, 2008.
- [8] Bainton DF., *Morphology of Neutrophils, Eosinophils, and Basophils*, 7a ed. Williams Hematology, 2006.
- [9] A. V. Hoffbrand, "Megaloblastic Anaemia", *Postgraduate Haematology*, vol. 6, pp. 61–82, 2010, doi: 10.1002/9781444323160.CH5.
- [10] Y. X. Chen y S. J. Ruan, "A Throughput-Optimized Channel-Oriented Processing Element Array for Convolutional Neural Networks", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, núm. 2, pp. 752–756, 2021, doi: 10.1109/TCSII.2020.3017789.
- [11] C. G. Varma, P. Nagaraj, V. Muneeswaran, K. Muthamil Sudar, M. Mokshagni, y M. Jaswanth, "Astute Segmentation and Classification of leucocytes in blood microscopic smear images using titivated K-means clustering and robust SVM techniques", *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 818–824, 2021, doi: 10.1109/ICICCS51141.2021.9432309.
- [12] V. Piuri y F. Scotti, "Morphological classification of blood leucocytes by microscope images", *2004 IEEE International Conference on Computational Intelligence for Measurements Systems and Applications, CIMSA*, pp. 103–108, 2004, doi: 10.1109/CIMSA.2004.1397242.

- [13] D. Gupta, J. Arora, U. Agrawal, A. Khanna, y V. H. C. de Albuquerque, "Optimized Binary Bat algorithm for classification of white blood cells", *Measurement (Lond)*, vol. 143, pp. 180–190, 2019, doi: 10.1016/j.measurement.2019.01.002.
- [14] M. Sajjad *et al.*, "Leukocytes Classification and Segmentation in Microscopic Blood Smear: A Resource-Aware Healthcare Service in Smart Cities", *IEEE Access*, vol. 5, pp. 3475–3489, 2017, doi: 10.1109/ACCESS.2016.2636218.
- [15] S. Ghosh, M. Majumder, y A. Kudeshia, "LeukoX: Leukocyte Classification Using Least Entropy Combiner (LEC) for Ensemble Learning", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, núm. 8, pp. 2977–2981, 2021, doi: 10.1109/TCSII.2021.3064389.
- [16] M. R. Reena y P. M. Ameer, "Localization and recognition of leukocytes in peripheral blood: A deep learning approach", *Comput Biol Med*, vol. 126, p. 104034, 2020, doi: 10.1016/J.COMPBIOMED.2020.104034.
- [17] D. Yang, H. Zhao, T. Han, Q. Kang, J. Ma, y H. Lu, "Leukocyte subtypes identification using bilinear self-attention convolutional neural network", *Measurement*, vol. 173, p. 108643, 2021, doi: 10.1016/J.MEASUREMENT.2020.108643.
- [18] A. M. Patil, M. D. Patil, y G. K. Birajdar, "White Blood Cells Image Classification Using Deep Learning with Canonical Correlation Analysis", *IRBM*, vol. 42, núm. 5, pp. 378–389, 2021, doi: 10.1016/J.IRBM.2020.08.005.
- [19] P. Huang *et al.*, "Attention-aware residual network based manifold learning for white blood cells classification", *IEEE J Biomed Health Inform*, vol. 25, núm. 4, pp. 1206–1214, 2021, doi: 10.1109/JBHI.2020.3012711.
- [20] R. B. Hegde, K. Prasad, H. Hebbar, y B. M. K. Singh, "Feature extraction using traditional image processing and convolutional neural network methods to classify white blood cells: a study", *Australas Phys Eng Sci Med*, vol. 42, núm. 2, pp. 627–638, 2019, doi: 10.1007/s13246-019-00742-9.
- [21] A. Çınar y S. A. Tuncer, "Classification of lymphocytes, monocytes, eosinophils, and neutrophils on white blood cells using hybrid Alexnet-GoogleNet-SVM", *SN Appl Sci*, vol. 3, núm. 4, p. 503, 2021, doi: 10.1007/s42452-021-04485-9.
- [22] H. Chen *et al.*, "Accurate classification of white blood cells by coupling pre-trained ResNet and DenseNet with SCAM mechanism", *BMC Bioinformatics*, vol. 23, núm. 1, pp. 1–20, 2022, doi: 10.1186/S12859-022-04824-6/FIGURES/10.
- [23] B. Zoph y Q. V. Le, "Neural Architecture Search with Reinforcement Learning", *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, nov. 2016, Consultado: el 29 de abril de 2023. [En línea]. Disponible en: <https://arxiv.org/abs/1611.01578v2>

- [24] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", *Cs Toronto Edu*, 2009, doi: 10.1.1.222.9220.
- [25] T. Elsken, J. H. Metzen, y F. Hutter, "Neural architecture search: A survey", *Journal of Machine Learning Research*, vol. 20, 2019.
- [26] G. Franchini *et al.*, "Neural architecture search via standard machine learning methodologies", *Mathematics in Engineering 2023 1:1*, vol. 5, núm. 1, pp. 1–21, 2023, doi: 10.3934/MINE.2023012.
- [27] L. Deng, "The MNIST database of handwritten digit images for machine learning research", *IEEE Signal Process Mag*, vol. 29, núm. 6, pp. 141–142, 2012, doi: 10.1109/MSP.2012.2211477.
- [28] Z. Zhang, S. Liu, Y. Zhang, y W. Chen, "RS-DARTS: A convolutional neural architecture search for remote sensing image scene classification", *Remote Sens (Basel)*, vol. 14, núm. 1, 2022, doi: 10.3390/rs14010141.
- [29] G. S. Xia *et al.*, "AID: A benchmark data set for performance evaluation of aerial scene classification", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, núm. 7, 2017, doi: 10.1109/TGRS.2017.2685945.
- [30] G. Cheng, J. Han, y X. Lu, "Remote Sensing Image Scene Classification: Benchmark and State of the Art", *Proceedings of the IEEE*, vol. 105, núm. 10. 2017. doi: 10.1109/JPROC.2017.2675998.
- [31] H. Li, C. Tao, Z. Wu, J. Chen, J. Gong, y M. Deng, "RSI-CB: A large-scale remote sensing image classification benchmark via crowdsourcing data", *arXiv*. 2017.
- [32] W. Zhou, S. Newsam, C. Li, y Z. Shao, "PatternNet: A benchmark dataset for performance evaluation of remote sensing image retrieval", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, 2018, doi: 10.1016/j.isprsjprs.2018.01.004.
- [33] B. Lyu, H. Yuan, L. Lu, y Y. Zhang, "Resource-Constrained Neural Architecture Search on Edge Devices", *IEEE Trans Netw Sci Eng*, vol. 9, núm. 1, 2022, doi: 10.1109/TNSE.2021.3054583.
- [34] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, y Li Fei-Fei, "ImageNet: A large-scale hierarchical image database", pp. 248–255, mar. 2010, doi: 10.1109/CVPR.2009.5206848.
- [35] S. H. S. Basha, S. K. Vinakota, S. R. Dubey, V. Pulabaigari, y S. Mukherjee, "AutoFCL: automatically tuning fully connected layers for handling small dataset", *Neural Comput Appl*, vol. 33, núm. 13, 2021, doi: 10.1007/s00521-020-05549-4.
- [36] L. Fei-Fei, R. Fergus, y P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories", *Computer Vision and Image Understanding*, vol. 106, núm. 1, 2007, doi: 10.1016/j.cviu.2005.09.012.

- [37] M. E. Nilsback y A. Zisserman, “Automated flower classification over a large number of classes”, en *Proceedings - 6th Indian Conference on Computer Vision, Graphics and Image Processing, ICVGIP 2008*, 2008. doi: 10.1109/ICVGIP.2008.47.
- [38] Y. Yang y S. Newsam, “Bag-of-visual-words and spatial extensions for land-use classification”, en *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, 2010. doi: 10.1145/1869790.1869829.
- [39] Y. Sun, X. Liu, y L. Liang, “Retrain-free fully connected layer optimization using matrix factorization”, en *Proceedings - International Conference on Image Processing, ICIP*, 2018. doi: 10.1109/ICIP.2017.8297016.
- [40] P. Lacerda, B. Barros, C. Albuquerque, y A. Conci, “Hyperparameter optimization for COVID-19 pneumonia diagnosis based on chest CT”, *Sensors*, vol. 21, núm. 6, 2021, doi: 10.3390/s21062174.
- [41] M. Tan y Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks”, en *36th International Conference on Machine Learning, ICML 2019*, 2019.
- [42] J. Navarro, “Design of a data augmentation methodology based on GAN and its application on a mask detection vision system”, Tecnológico Nacional de México campus Chihuahua, Chihuahua, 2022. [En línea]. Disponible en: <https://posgradoitch.mx/wp-content/uploads/2022/11/Tesis-Navarro-Acosta-Jesus-Alejandro.pdf>
- [43] L. N. Smith, “Cyclical learning rates for training neural networks”, en *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, 2017. doi: 10.1109/WACV.2017.58.
- [44] W. Wang, C. M. Lee, J. Liu, T. Colakoglu, y W. Peng, “An empirical study of cyclical learning rate on neural machine translation”, *Nat Lang Eng*, 2022, doi: 10.1017/S135132492200002X.
- [45] K. He, X. Zhang, S. Ren, y J. Sun, “Deep residual learning for image recognition”, en *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.
- [46] K. Simonyan y A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, en *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015, pp. 1–14. doi: 10.48550/ARXIV.1409.1556.
- [47] A. Krizhevsky, I. Sutskever, y G. E. Hinton, “ImageNet classification with deep convolutional neural networks”, *Commun ACM*, vol. 60, núm. 6, pp. 84–90, 2017, doi: 10.1145/3065386.
- [48] Z. M. Kouzehkanan *et al.*, “A large dataset of white blood cells containing cell locations and types, along with segmented nuclei and cytoplasm”, *Sci Rep*, vol. 12, núm. 1, p. 1123, 2022, doi: 10.1038/s41598-021-04426-x.
- [49] B. Rodak y J. Carr, “Atlas de Hematología Clínica”, *Editorial Médica Panamerica*, vol. 1, 2014.

- [50] J. Basnet, A. Alsadoon, P. W. C. Prasad, S. Al Aloussi, y O. H. Alsadoon, "A Novel Solution of Using Deep Learning for White Blood Cells Classification: Enhanced Loss Function with Regularization and Weighted Loss (ELFRWL)", *Neural Process Lett*, vol. 52, núm. 2, pp. 1517–1553, 2020, doi: 10.1007/s11063-020-10321-9.
- [51] A. I. Shahin, Y. Guo, K. M. Amin, y A. A. Sharawi, "White blood cells identification system based on convolutional deep neural learning networks", *Comput Methods Programs Biomed*, vol. 168, pp. 69–80, 2019, doi: 10.1016/j.cmpb.2017.11.015.
- [52] L. Ma, R. Shuai, X. Ran, W. Liu, y C. Ye, "Combining DC-GAN with ResNet for blood cell image classification", *Med Biol Eng Comput*, vol. 58, núm. 6, pp. 1251–1264, 2020, doi: 10.1007/s11517-020-02163-3.
- [53] A. Khouani, M. El Habib Daho, S. A. Mahmoudi, M. A. Chikh, y B. Benzineb, "Automated recognition of white blood cells using deep learning", *Biomed Eng Lett*, vol. 10, núm. 3, pp. 359–367, 2020, doi: 10.1007/s13534-020-00168-3.
- [54] R. Al-qudah y C. Y. Suen, "Improving blood cells classification in peripheral blood smears using enhanced incremental training", *Comput Biol Med*, vol. 131, p. 104265, 2021, doi: 10.1016/j.compbimed.2021.104265.
- [55] R. M. Roy y P. M. Ameer, "Segmentation of leukocyte by semantic segmentation model: A deep learning approach", *Biomed Signal Process Control*, vol. 65, p. 102385, 2021, doi: 10.1016/j.bspc.2020.102385.
- [56] B. C. Gregório da Silva, R. Tam, y R. J. Ferrari, "Detecting cells in intravital video microscopy using a deep convolutional neural network", *Comput Biol Med*, vol. 129, p. 104133, 2021, doi: 10.1016/j.compbimed.2020.104133.
- [57] M. Sharif *et al.*, "Recognition of different types of leukocytes using YOLOv2 and optimized bag-of-features", *IEEE Access*, vol. 8, pp. 167448–167459, 2020, doi: 10.1109/ACCESS.2020.3021660.
- [58] P. Pandey, A. P. Prathosh, V. Kyatham, D. Mishra, y T. R. Dastidar, "Target-Independent Domain Adaptation for WBC Classification Using Generative Latent Search", *IEEE Trans Med Imaging*, vol. 39, núm. 12, pp. 3979–3991, 2020, doi: 10.1109/TMI.2020.3009029.
- [59] F. Zhuang *et al.*, "A Comprehensive Survey on Transfer Learning", *Proceedings of the IEEE*, vol. 109, núm. 1. 2021. doi: 10.1109/JPROC.2020.3004555.

10 ANEXOS

Interfaz Gráfica de Usuario ITCHLeukoAnalysis

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 24 15:15:56 2023

@author: reyes
"""

##### Librerias
from tkinter import Tk, LEFT, Label, Button, IntVar, Scale, PhotoImage, E, messagebox, filedialog
from PIL import Image, ImageTk
import cv2
import numpy as np
import onnxruntime as rt
import os

# Para plot emergentes
# from IPython import get_ipython
# ipython = get_ipython()
# ipython.magic('matplotlib qt5')

# Mejora la calidad de DPI de la ventana
import ctypes
ctypes.windll.shcore.SetProcessDpiAwareness

##### Variables de control
dato = 0
version = '6.0'
edic = False

##### Funciones

def comando_entrada():
    global lblInputImage, alpha, beta, res_resultados, edic

    # Carga nueva imagen muestras
    imagenm, alpha, beta = nueva_muestra(root)
    edic = False
    scl2.set(alpha) # pone valor de contraste en 1
    scl.set(beta) # poner valor de brillo en 0

    if(dato == 1):
        # Muestras imagen muestra en label lblInputImage
        lblInputImage.configure(image = imagenm)
        lblInputImage.image = imagenm

    # Limpia label de predicciones obtenidas del modelo de clasificacion
```

```
res_resultados.config(text="", justify= LEFT)

def nueva_muestra(raiz):
    global imagen, original, imgsharp, imgdet, path_image, controlbc, brillo
    global contraste, nit, dato, count, img, nombre_imagen, imgedit, imsave
    borrar_imagen()
    nit = 0
    controlbc = 0
    contraste = 1
    brillo = 0
    count = 0
    dato = 0

    path_image = filedialog.askopenfilename(title = "Selecciona una muestra",
                                            filetype = [("Muestras", ".jpg"), ("Muestras", ".jpeg"), ("Muestras", ".png")]
                                            )

    textito2 = str(os.path.basename(path_image))
    nombre_imagen.config(text=textito2)

    if len(path_image) > 0:
        # global imagen, original, imgsharp, imgdet

        #Lee la imagen de entrada
        imagen = cv2.cvtColor(cv2.imread(path_image),cv2.COLOR_BGR2RGB)
        imgdet = imagen.copy()
        original = imagen.copy()
        imgsharp = imagen.copy()
        img = Image.fromarray(imagen) # convierte formato imagen muestra para mostrarla en etiqueta
        imsave = img
        img = ImageTk.PhotoImage(img) # convierte formato imagen muestra para mostrarla en etiqueta
        dato = 1
    return img, 1,0

def borrar_imagen():
    global lblInputImage, alpha, beta, res_resultados, edic, nombre_imagen
    alpha, beta = borrarimagen2(1) # Funcion para borrar imagenes leidas
    edic = False
    scl2.set(alpha) # pone valor de contraste en 1
    scl.set(beta) # poner valor de brillo en 0

    # Elimina imagen muestra en lblInputImage
    lblInputImage = Label(root)
    lblInputImage.grid(row=3, column=3, rowspan=6, sticky="nsew")
    lblInputImage.configure(background='#f8f8f8')

    # Limpia label de nombre de imagen
    nombre_imagen.config(text="")

    # Limpia label de predicciones obtenidas del modelo de clasificacion
    res_resultados.config(text="")
```

```
def borrarimagen2(borrar):
    global imagen, original, imgsharp, imgdet, controlbc, dato, count
    # nit = 0
    controlbc = 0

    # rest = 1
    count = 0
    if (borrar == 1):
        # global imagen, original, imgsharp, imgdet
        imagen = None # Borra imagen muestra
        original = None # Borra copia original de imagen muestra
        imgsharp = None # Borra copia imgsharp de imagen muestra
        imgdet = None # Borra copia imgdet de imagen muestra
        dato = 0
        borrar = 0
        return 1, 0

def autoajuste():
    global auto_result, dato, edic
    if (dato==1):
        auto_result = autoajuste2()
        edic = True
        # Muestras imagen auto ajustada en label lblInputImage
        lblInputImage.configure(image=auto_result)
        lblInputImage.image = auto_result

def autoajuste2():
    # Se autoajusta la variable 'imagen' y se regresa con auto ajuste
    global imagen, contraste, brillo, controlbc, count, imgsharp, original, imsave
    count = 0
    if(controlbc==0):
        imagen = cv2.addWeighted(imagen,float(contraste),imagen,0,int(brillo))
        controlbc=1
    imgconvert = cv2.cvtColor(imagen,cv2.COLOR_RGB2BGR)
    #Creación de la Ecualización de histograma adaptativa limitada por contraste(CLAHE)
    clahe = cv2.createCLAHE(clipLimit=3.0,tileGridSize=(8,8))
    claheg = cv2.createCLAHE(clipLimit=1.0,tileGridSize=(8,8))
    #Aplicación de la ecualización a cada canal del formato BGR
    colorimage_b = clahe.apply(imgconvert[:, :, 0])
    colorimage_g = claheg.apply(imgconvert[:, :, 1])
    colorimage_r = clahe.apply(imgconvert[:, :, 2])
    imagen = np.stack((colorimage_b,colorimage_g,colorimage_r),axis=2)
    imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)
    auto_result = Image.fromarray(imagen)
    imsave = auto_result
    auto_result = ImageTk.PhotoImage(auto_result)
    imgsharp = imagen

    return auto_result

def configbrillo(value):
    global beta, alpha, dato, edic
```

```
if (dato==1):
    beta = value
    alpha = float(barra2.get())

    imgedit = config_brillo_contraste(alpha=alpha,beta=beta)
    lblInputImage.configure(image=imgedit)
    lblInputImage.image = imgedit
    # print('Antes configbrillo = ', edic)
    if(float(alpha)==1.0 and int(beta)==0): #brillo es beta y contraste es alpha
        edic = False
    else:
        edic = True
    # print('Despu configbrillo = ', edic)

def config_brillo_contraste(alpha, beta):
    global imagen, imgsharp, contraste, brillo, controlbc, imsave
    if(controlbc==1):
        controlbc=0
    contraste = alpha
    brillo = beta
    imgedit = cv2.addWeighted(imagen,float(alpha),imagen,0,int(beta))
    imgsharp = imgedit
    imgedit = Image.fromarray(imgedit)
    imsave = imgedit
    imgedit = ImageTk.PhotoImage(imgedit)
    return imgedit

def configcontraste(value):
    global alpha, beta, dato, edic
    if (dato==1):
        alpha = value
        beta = int(barra1.get())

    imgedit = config_brillo_contraste(alpha=alpha,beta=beta)

    lblInputImage.configure(image=imgedit)
    lblInputImage.image = imgedit
    # print('Antes configcontraste = ', edic)
    if(float(alpha)==1.0 and int(beta)==0): #brillo es beta y contraste es alpha
        edic = False
    else:
        edic = True
    # print('Despu configcontraste = ', edic)

def restaurar():
    global alpha, beta, res_resultados, dato, edic
    if (dato==1):
```

```
imgedit, alpha, beta = restaurar_valores()
scl2.set(alpha) # pone valor de contraste en 1
scl.set(beta) # pone valor de brillo en 0
# print('Antes restaurar = ', edic)
edic = False
# print('Despu restaurar = ', edic)
# Muestras imagen muestra en label lblInputImage restaurada
lblInputImage.configure(image=imgedit)
lblInputImage.image = imgedit
res_resultados.config(text="")

def restaurar_valores():
    global original, imagen, rest, x, imgsharp, count, nit, imgdet, imsave
    nit = 0
    count = 0
    imagen = original
    imgsharp = original
    imgdet = original
    imagen_original = Image.fromarray(imagen)
    imsave = imagen_original
    imagen_original = ImageTk.PhotoImage(imagen_original)
    #Retornar valores de contraste = 1 y brillo = 0, segun la ecuación de la recta
    return imagen_original, 1, 0

def nitidez():
    global dato, edic
    if (dato==1):
        imgedit = nitidez2()
        edic = True
        lblInputImage.configure(image=imgedit)
        lblInputImage.image = imgedit

def nitidez2():
    global rest, imgsharp, count, nit, imsave
    nit = 1
    if (count == 3):
        messagebox.showwarning(title="Advertencia",message="Nitidez máxima alcanzada")
    else:
        count = count + 1
        kernel = np.array([[ -1,-1,-1], [-1,9,-1], [-1,-1,-1]])
        imgsharp = cv2.filter2D(imgsharp,-1,kernel)

    imgedit = Image.fromarray(imgsharp)
    imsave = imgedit
    imgedit = ImageTk.PhotoImage(imgedit)
    return imgedit

def susnitidez():
    global dato, edic
    if (dato==1):
        imgedit = susnitidez2()
```

```
    edic = False
    lblInputImage.configure(image=imgedit)
    lblInputImage.image = imgedit

def susnitidez2():
    global imagen, rest, imgsharp, contraste, brillo, controlbc, count, nit, imsave
    imgsharp = imagen
    count = 0
    nit = 0
    if(controlbc == 0):
        imagen = cv2.addWeighted(imagen,float(contraste),imagen,0,int(brillo))
        controlbc=1

    imgedit = Image.fromarray(imagen)
    imsave = imgedit
    imgedit = ImageTk.PhotoImage(imgedit)
    return imgedit

def deteccion():
    messagebox.showwarning(title="Advertencia",message="Detección no habilitada.")

def clasificacion():
    global dato
    if (dato==1):
        prediccion = clasificacion2()
        resultados(prediccion)

def clasificacion2():
    global imagen, original, imgsharp, imgdet, lista_c, nit, edic

    if(edic==True):
        messagebox.showwarning(message="El resultado de la clasificación no será afectado por la edición de la
imagen.", title="Advertencia")

    lista_c = Run(original)

    return lista_c

def Run(entrada):
    global input_name, label_name, sess, preds

    # Checar funcion de resize que esta sacando
    ss = 224

    img = cv2.resize(entrada, dsize=(ss, ss), interpolation=cv2.INTER_LINEAR)
    x = np.expand_dims(img, axis=0)

    pred_onx = sess.run([label_name], {input_name: x.astype(np.float32)})[0]
    preds = int(np.argmax(pred_onx))
```

```
return preds

def resultados(prediccion):
    global res_resultados
    textito = ""

    etiqueta = ["Basófilo", "Eosinófilo", "Linfocito",
                "Monocito", "Neutrófilo"]

    textito = str(1) + ".- " + str(etiqueta[prediccion]) + "\n"
    res_resultados.config(text="", justify= LEFT)
    res_resultados.config(text=textito, justify=LEFT)

def guardar_imagen():
    global imsave
    if (dato==1):
        file_name = filedialog.asksaveasfilename( filetypes=[( 'PNG', '*.png') ])

        if file_name.endswith(".png"):
            print(file_name)
            imsave.save(file_name)
        else:
            print(file_name + ".png")
            imsave.save(file_name + ".png")

##### Se carga el modelo profundo de clasificacion
output_path = 'alexnetv2_model_14-01-2023_cross2.onnx'
providers = ['CPUExecutionProvider']
sess = rt.InferenceSession(output_path, providers=providers)
input_name = sess.get_inputs()[0].name
label_name = sess.get_outputs()[0].name

##### Creación de la ventana
root = Tk()
# Propiedades de la ventana
root.title('ITCHLeukoAnálisisV' + version) # Titulo de ventana
width = root.winfo_screenwidth() # Calcula el ancho del monitor
height = root.winfo_screenheight() # Calcula el alto del monitor
root.geometry("1050x650")
# root.geometry("%dx%d" %(0.9*width,0.8*height)) # 1680, 1050, ajusta la pantalla a un tamaño menor al
monitor
root.configure(bg='#f8f8f8') # Aplica color casi blanco de fondo
root.resizable(1,1) # Permite que se cambie de tamaño la pantalla
root.iconbitmap('leuco.ico')
```

```
root.minsize(1050, 650)
```

```
##### Creacion de objetos de ventana
```

```
##### LABELS #####
```

```
# Label de imagen a mostrar
```

```
lblInputImage = Label(root)
```

```
lblInputImage.grid(row=3, column=3, rowspan=6, sticky="nsew")
```

```
lblInputImage.configure(background='#f8f8f8')
```

```
# Label de texto 'Controles'
```

```
titulo_botones = Label(root, text="Controles",fg='gray')
```

```
titulo_botones.config(font= ('Arial', 20),bg='#f8f8f8', borderwidth=0, relief='flat')
```

```
titulo_botones.grid(row=1,column=0,columnspan=3,padx=15,pady=5, sticky='ws')
```

```
# Label de texto 'ITCHLeukoAnálisisVx.0'
```

```
titulo = Label(root, text="ITCHLeukoAnálisisV" + version,fg='black')
```

```
titulo.config(font= ('Arial', 25,'bold'),bg='#f8f8f8', borderwidth=0, relief='flat')
```

```
titulo.grid(row=0,column=3)
```

```
# Label de texto 'Nitidez'
```

```
titulo_nitidez = Label(root, text="Nitidez",fg='gray')
```

```
titulo_nitidez.config(font= ('Arial', 20),bg='#f8f8f8', borderwidth=0, relief='flat')
```

```
titulo_nitidez.grid(row=8,column=0,padx=(20,1),pady=5, sticky='ns')
```

```
# Label de LOGO PVR
```

```
img1 = PhotoImage(file='LOGO5.png')
```

```
img1 = img1.subsample(3)
```

```
lbl_img = Label(root, image = img1)
```

```
lbl_img.config(borderwidth=0, relief='flat')
```

```
lbl_img.grid(row = 0, column=6,columnspan=2,padx=10,sticky=E)
```

```
# Label de LOGO ITCH
```

```
img2 = PhotoImage(file='LOGO4.png')
```

```
img2 = img2.subsample(3)
```

```
lbl_img2 = Label(root, image = img2)
```

```
lbl_img2.config(borderwidth=0, relief='flat')
```

```
lbl_img2.grid(row = 0, column=4,columnspan=2, padx=10,sticky=E)
```

```
# Label de texto 'Resultados'
```

```
titulo_resultados = Label(root, text="Resultados",fg='gray')
```

```
titulo_resultados.config(font= ('Arial', 20),bg='#f8f8f8', borderwidth=0, relief='flat')
```

```
titulo_resultados.grid(row=1,column=4,columnspan=2,padx=15,pady=5, sticky='ws')
```

```
# Label de texto para poner predicciones obtenidas de clasificacion
```

```
res_resultados = Label(root,fg='black')
```

```
res_resultados.configure(font= ('Arial', 16),bg='#f8f8f8', borderwidth=0, relief='flat')
res_resultados.grid(row=2,column=4,columnspan=5,rowspan=12, padx=15,pady=5, sticky='NW')

# Label de texto 'Nombre de imagen'
nombre_imagen = Label(root, fg='black')
nombre_imagen.configure(font= ('Arial', 16),bg='#f8f8f8', borderwidth=0, relief='flat')
nombre_imagen.grid(row=9,column=3)

##### BOTONES #####
# Boton para ingresar nueva imagen muestra
boton_nuevo = Button(root, text = "Nueva muestra")
boton_nuevo.config(width = 20, font=('Arial', 16, 'bold'), fg='#DAD5D6', bg= '#158645',cursor='hand2',
activebackground='#35BD6F', command=comando_entrada)
boton_nuevo.grid(row=2, column=0, padx=20,pady=2, columnspan=3)

# Boton para realizar la deteccion
boton_deteccion = Button(root, text = "Detección")
boton_deteccion.config(width=20, font=('Arial', 16, 'bold'), fg='#DAD5D6', bg= '#1658A2',cursor='hand2',
activebackground='#3586DF', command=deteccion)
boton_deteccion.grid(row=3, column=0, pady=2,columnspan=3)

# Boton para realizar la clasificacion
boton_clasificacion = Button(root, text = "Clasificación")
boton_clasificacion.config(width=20, font=('Arial', 16, 'bold'), fg='#DAD5D6', bg= '#8D152E',cursor='hand2',
activebackground='#E15370', command=clasificacion)
boton_clasificacion.grid(row=4, column=0, pady=2,columnspan=3)

# Boton para realizar un ajuste automatico a la imagen muestra
boton_ajuste = Button(root, text = "Ajuste automático")
boton_ajuste.config(width=20, font=('Arial', 16, 'bold'), fg='#DAD5D6', bg= '#F18D4B',cursor='hand2',
activebackground='darkgray', command=autoajuste)
boton_ajuste.grid(row=5, column=0, pady=2,columnspan=3)

# Boton para restaurar los valores originales de la imagen muestra
boton_restaurar = Button(root, text = "Restaurar valores")
boton_restaurar.config(width=20, font=('Arial', 16,'bold'), fg='white', bg= 'gray',cursor='hand2',
activebackground='darkgray', command=restaurar)
boton_restaurar.grid(row=9, column=0, pady=(5,2),columnspan=3)

# Boton para aumentar la nitidez
nitidez_img = PhotoImage(file = 'masnitidez.png')
boton_nitidez = Button(root)
boton_nitidez.config(font=('Arial', 16), fg='darkgray',cursor='hand2', image = nitidez_img, command=nitidez)
boton_nitidez.grid(row=8, column=1, pady=(5,2))

# Boton para disminuir la nitidez
regreso_img = PhotoImage(file = 'fback.png')
boton_regreso = Button(root)
boton_regreso.config(font=('Arial', 16), fg='darkgray',cursor='hand2', image = regreso_img,
command=susnitidez)
boton_regreso.grid(row=8, column=2, pady=(5,2))

# Boton para borrar imagen muestra
```

```
boton_borrar = Button(root, text = "Borrar imagen")
boton_borrar.config(width=20, font=('Arial', 16), fg='black', cursor='hand2',
activebackground='darkgray', command=borrar_imagen)
boton_borrar.grid(row=11, column=0, pady=(5,35), columnspan=3)

# Boton para guardar imagen mostrada
boton_imagen = Button(root, text = "Guardar imagen")
boton_imagen.config(width=20, font=('Arial', 7, 'bold'), fg='white', bg= 'gray', cursor='hand2',
activebackground='darkgray', command=guardar_imagen)
boton_imagen.grid(row=10, column=3)

##### BARRAS DESPLAZAMIENTO #####
barra1 = IntVar() # variable de valor de barra de brillo
barra2 = IntVar() # variable de valor de barra de contraste

# Barra de brillo
scl = Scale(root, from_=-250, to=250, tickinterval=250,
            length=210, resolution=1, showvalue='YES',
            orient='horizontal', variable=barra1, label="Brillo:", command=configbrillo)
scl.config(bg='#f8f8f8', borderwidth=0, relief='flat', cursor='hand2')
scl.grid(row=6, column=0, padx=20, pady=2, columnspan=3, sticky='nsew')

# Barra de contraste
scl2 = Scale(root, from_=1, to=5, tickinterval=1,
            length=210, resolution=0.1, showvalue='YES',
            orient='horizontal', variable=barra2, label="Contraste:", command=configcontraste)
scl2.config(bg='#f8f8f8', borderwidth=0, relief='flat', cursor='hand2')
scl2.grid(row=7, column=0, padx=20, pady=2, columnspan=3, sticky='nsew')

#Loop de la ventana
root.mainloop()
```