UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Estudio de algoritmos de redes neuronales convolucionales en dataset de imágenes médicas

Autor:

Antonio Ferreras Extremo

Tutor:

Dra. Isabel de la Torre Díez

Valladolid, 31 de marzo de 2021

TRABAJO DE FIN DE GRADO

TÍTULO:	**Estudio de algoritmos de redes neuronales convolucionales en dataset de imágenes médicas**
AUTOR:	**Antonio Ferreras Extremo**
TUTOR	**Dra. Isabel de la Torre Díaz**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. Miguel López Coronado**
VOCAL:	**Dra. Beatriz Sainz de Abajo**
SECRETARIO:	**Dra. Isabel de la Torre Díaz**
SUPLENTE:	
SUPLENTE:	

FECHA:	**6 de abril de 2021**
CALIFICACIÓN	

Resumen

Cada año mueren en el mundo más de 400.000 personas de malaria, a pesar de que la malaria es una enfermedad prevenible y relativamente fácil de tratar si se detecta de forma precoz. El objetivo principal de este trabajo es el de proponer un sistema de ayuda de soporte a las decisiones médicas para la detección de malaria a partir de imágenes de frotis de microscopio de células sanguíneas, utilizando un modelo de red neuronal convolucional (CNN) y otras técnicas de aprendizaje profundo. Esta CNN se ha creado utilizando como base una arquitectura EfficientNet. La contribución clave de este trabajo es presentar los resultados de un modelo CNN *ensemble* creado a partir de combinar los modelos base EfficientNet0 obtenidos durante un proceso de validación cruzada de 10 iteraciones. En este documento se presentan los resultados de clasificación de imágenes de pacientes infectados y pacientes sanos. Se ha obtenido una exactitud en la clasificación del 98,29%, significativamente superior al de trabajos similares encontrados en la literatura. El modelo propuesto CNN utilizando EfficientNet presenta unos resultados con valores comparables, e incluso superiores, a los obtenidos en el estado del arte actual: un valor de *recall* de 98,82%, un valor de precisión de 97,74%, un valor de F1-score de 98,28% y un valor para el AUC (Área bajo la Curva ROC) de 99,76%. Para concluir, el diseño propuesto ofrece un sistema para el diagnóstico automático de la malaria que ayuda a los profesionales médicos a realizar mejores decisiones.

Palabras Clave

Malaria, Diagnóstico, Redes Neuronales, Aprendizaje Profundo, Python, EfficientNet, Albumentations, Ensembled, Image Augmentation

Abstract

Each year, more than 400,000 people die of malaria; surprisingly enough, malaria is a preventable and treatable disease if early detection is achieved. The key objective of this work is to propose a medical decision help system for detecting malaria from microscopic peripheral blood cells images, using the application of a convolutional neural network (CNN). This CNN has been created using an EfficientNet architecture. The key contribution is to introduce the findings of an ensemble CNN model created combining the base models obtained through a 10-fold stratified cross-validation of a EfficientNet0-based architecture. This paper presents the classification findings using images from malaria patients and normal patients. An accuracy value for binary classification of 98.29% is obtained. The proposed CNN model using EfficientNet presents state-of-the-art values: recall value of 98.82, a precision value 97.74%, F1-score of 98.28% and a ROC value of 99.76%. To conclude, the suggested design offers an automatic medical diagnostics system to assist malaria specialists to make enhanced decision.

Keywords

Malaria, Diagnosis, Neural Networks, Deep Learning, Python, EfficientNet, Albumentations, Ensembled, Image Augmentation

Agradecimientos

Dedicado a mi madre

Contenido

Índice de Figuras

Índice de Tablas

1 Introducción

1.1.1 La Malaria

La malaria es una enfermedad grave y en muchos casos mortal causado por una infección del parásito *Plasmodium* y transmitida por la picadura del mosquito Anófeles (ver Figura 1-1). Afecta a humanos y otros animales. Los parásitos maduran en el hígado del huésped, y son liberados en el torrente sanguíneo humano e infectan los glóbulos rojos de la sangre, con la aparición de los síntomas de la enfermedad, en muchos casos con consecuencias fatales.

Figura 1-1. El mosquito anofeles es el principal transmisor de la enfermedad[1]

Existen varias especies de parásitos de la malaria, entre ellos *Plasmodium falciparum*, *P. vivax*, *P. ovale, P. Knowlesi*, y *P. malariae*. Es *P. falciparum* el parásito que puede ser letal e infecta a la mayoría de la población mundial. De acuerdo con el informe de 2020 de la *World Health Organization* (WHO), se estima que ha habido en el mundo unos 1,5 miles de millones de casos de malaria en el periodo el año 2000-2019 y en torno a 7,6 millones de muertos [1]. Sólo en el año 2019, se reportaron 229 millones de nuevos casos y hubo un total de 409.000 muertes (ver Figura 1-2 y Figura 1-4). Los niños de edad inferior a 5 años son los más vulnerables a la enfermedad, dando cuenta de un 61% del balance de muertes. Esta enfermedad tiene la mayor prevalencia en África, seguida del sudeste asiático y de las regiones del este del Mediterráneo. Se estima que 3,1 mil millones de

[1] Fuente: Shutterstock

dólares se dedican anualmente a nivel mundial en las estrategias de control y eliminación de la malaria en los países donde la enfermedad es endémica. Además, África pierde en torno a 12 miles de millones por año debido al impacto de la enfermedad en la industria, los servicios y el turismo. Los síntomas iniciales de la malaria, tales como escalofríos, fiebre, dolor de cabeza o vómitos, pueden ser potencialmente moderados y difíciles de identificar. Pero, sin un tratamiento adecuado, la malaria puede derivar en una enfermedad grave y causar la muerte. [2].

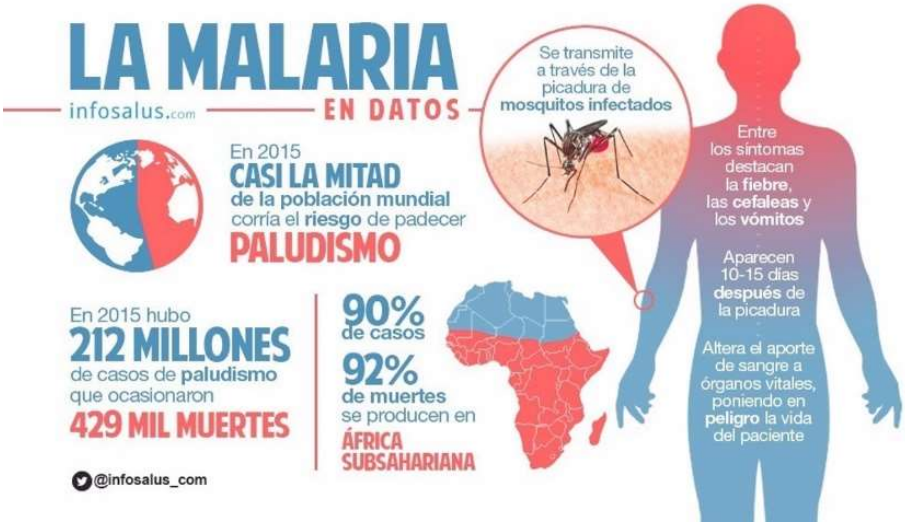


Figura 1-2. La malaria en datos[2]

Un diagnóstico y tratamiento precoz es el método más eficaz de prevenir la enfermedad [3]. El método más fiable y comúnmente conocido para el diagnóstico de la enfermedad es el análisis de las imágenes al microscopio de los glóbulos rojos de los pacientes posiblemente infectados [4]. Estas imágenes de los eritrocitos son examinadas por personal médico experto en las técnicas de microscopía [5] [6]. Sin embargo, el diagnóstico manual es un proceso muy trabajoso, que impacta seriamente en la exactitud del diagnóstico, por la fiabilidad de muchos factores, como son la variabilidad de las observaciones inter o intraindividual y la monitorización a gran escala, particularmente en los países donde la enfermedad es endémica que tienen muchas restricciones de recursos económicos disponibles [7]. Una distribución mundial de la enfermedad por países se puede ver en la Figura 1-3.

[2] Fuente: Europa Press

El proceso de reconocimiento manual es un proceso que consume mucho tiempo, donde los expertos necesitan clasificar manualmente 5.000 células para adquirir la experiencia adecuada. Sería deseable tener a disposición métodos de diagnóstico rápidos, baratos y fiables; por eso, desde 2005 se han venido empleando técnicas de Big Data y M*achine Learning* para el reconocimiento automático de la enfermedad, tanto para frotis gruesos y finos [8].

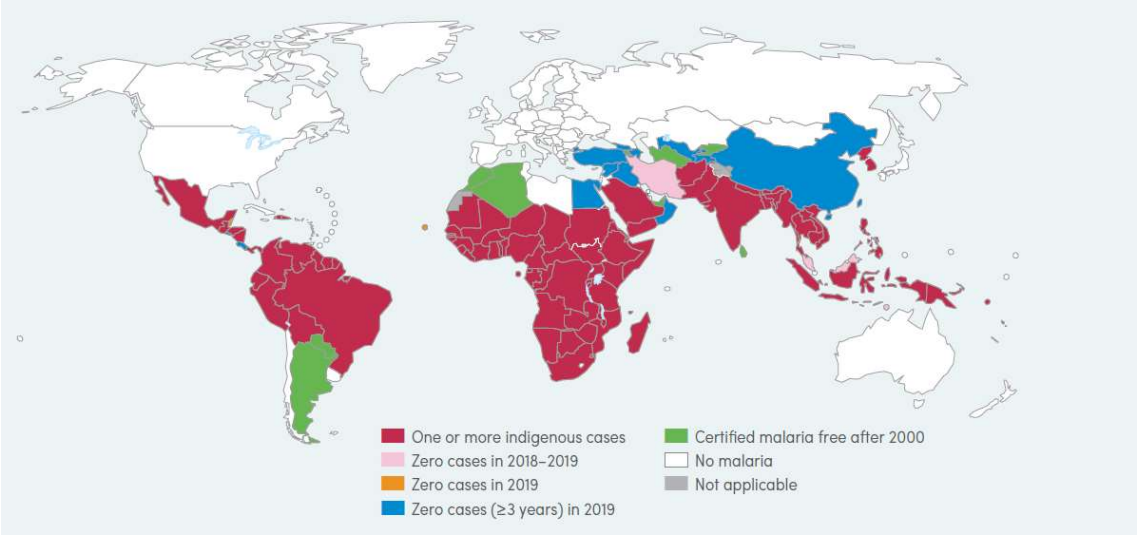


Figura 1-3. Distribución mundial de la malaria[3]

1.1.2 Primeros modelos

El diagnóstico asistido por ordenador (CADx) utilizando técnicas de *Machine Learning*, aplicadas a las imágenes de microscopio de los glóbulos rojos tienen el potencial de reducir la carga clínica al asistir en los procesos de triaje e interpretación de la enfermedad. Estas herramientas procesan las imágenes médicas y destacan las características patológicas contenidas en ellas para complementar la toma de decisiones clínicas. Sin embargo, la mayoría de estas herramientas, cuando se aplican al diagnóstico de la malaria, necesitan de un algoritmo de extracción de características optimizado para una concreta variabilidad de maquinaria, dimensiones, posición y orientación de la

[3] Fuente WHO

región de interés (ROI) [9]. Los métodos de *Deep Learning* han demostrado mejorar el rendimiento de esos métodos de clasificación más tradicionales.

TABLE 3.1.

Global estimated malaria cases and deaths, 2000–2019 Estimated cases and deaths are shown with 95% upper and lower confidence intervals. *Source: WHO estimates.*

Year	Number of cases (000)				Number of deaths		
	Point	Lower bound	Upper bound	% *P. vivax*	Point	Lower bound	Upper bound
2000	**238 000**	222 000	259 000	6.9%	**736 000**	697 000	782 000
2001	**244 000**	228 000	265 000	7.4%	**739 000**	700 000	786 000
2002	**239 000**	223 000	260 000	7.1%	**736 000**	698 000	783 000
2003	**244 000**	226 000	268 000	7.8%	**723 000**	681 000	775 000
2004	**248 000**	227 000	277 000	8.0%	**759 000**	708 000	830 000
2005	**247 000**	229 000	272 000	8.3%	**708 000**	662 000	765 000
2006	**242 000**	223 000	268 000	7.2%	**716 000**	675 000	771 000
2007	**241 000**	222 000	265 000	6.8%	**685 000**	644 000	735 000
2008	**240 000**	222 000	264 000	6.5%	**638 000**	599 000	685 000
2009	**246 000**	226 000	271 000	6.5%	**620 000**	572 000	681 000
2010	**247 000**	226 000	273 000	7.0%	**594 000**	546 000	658 000
2011	**239 000**	218 000	262 000	7.2%	**545 000**	505 000	596 000
2012	**234 000**	213 000	258 000	6.6%	**517 000**	481 000	568 000
2013	**225 000**	206 000	248 000	5.3%	**487 000**	451 000	538 000
2014	**217 000**	201 000	236 000	4.3%	**471 000**	440 000	511 000
2015	**218 000**	203 000	238 000	3.9%	**453 000**	422 000	496 000
2016	**226 000**	210 000	247 000	4.0%	**433 000**	403 000	478 000
2017	**231 000**	213 000	252 000	3.4%	**422 000**	396 000	467 000
2018	**228 000**	211 000	250 000	3.2%	**411 000**	389 000	458 000
2019	**229 000**	211 000	252 000	2.8%	**409 000**	387 000	460 000

P. vivax: Plasmodium vivax; WHO: World Health Organization.

Figura 1-4. Número de casos estimados de malaria y muertes asociadas[4]

Las redes neuronales convolucionales (CNNs) son un tipo de redes neuronales utilizados en diferentes aplicaciones [10] [11] [12]. Las CNNs utilizan habitualmente una capa de entrada, varias capas internas (*hidden layers*), y una capa de salida [13]. Las capas internas están compuestas habitualmente por capas completamente interconectadas, capas convolucionales, y capas de activación como ReLu [14] [15]. Las CNNs suponen un gran avance como método de clasificación de imágenes, ya que no requieren pre-procesado de las imágenes como otros algoritmos más convencionales de *machine learning* [16] [17] [18].

[4] Fuente WHO

El rendimiento prometedor de las redes neuronales convolucionales se atribuye a la disposición de ingentes cantidades de datos. En los casos de sólo disponer cantidades limitadas de datos, como es el caso de las imágenes médicas, se adoptan las estrategias de *transfer learning* (apartado 2.3.10).

Los estudios publicados muestran la aplicación de algoritmos de *Machine Learning* convencionales para la detección de la malaria en imágenes de microscopio de glóbulos rojos. En un estudio reciente [19], se compara el rendimiento de algoritmos basados en Kernel, como *Support Vector Machine* (SVM), y redes neuronales convolucionales (CNN) frente una clasificación de células sanas e infectadas. Un conjunto pequeño de imágenes se dividió aleatoriamente en conjuntos de entrenamiento, test y validación. Se observó que los métodos CNN conseguían una exactitud del 95%, mejorando los resultados del clasificador SVM que obtuvo una exactitud del 92%. Las CNNs detectaron de forma autónoma las características de las imágenes directamente de los datos, sin intervención humana. En otro estudio [20] se realizó un estudio de validación cruzada a nivel celular para evaluar el rendimiento de modelos CNN pre-entrenados y entrenados ad-hoc, con un rendimiento superior de estos últimos con una exactitud del 97.37%, en línea con otros estudios posteriores. Estos estudios se realizaron a nivel celular, utilizando conjunto de datos pequeños, a menudo utilizando múltiples imágenes de pocos individuos.

Aunque los resultados son prometedores, se necesitan estudios de validación cruzada a nivel de paciente con un conjunto de datos clínicos extenso para soportar la robustez y generalización de la aplicación en el mundo real; esta tarea se realizó con una red CNN pre-entrenada (ResNet50) para extraer las características básicas, con una bases de datos grande de imágenes clínicas, obteniendo una exactitud del 95.9%, validando su rendimiento tanto a nivel individual como de paciente, discriminando células infectadas y sanas [21].

1.1.3 Ensemble learning

Sin embargo, ya hemos visto que los modelos de *Deep Learning* aprenden a través de optimización estocástica y tienen un rendimiento limitado debido a la alta varianza en la predicción que surge por su alta sensibilidad a pequeñas fluctuaciones del conjunto

de datos usado por el entrenamiento, lo que provoca que también se modele el ruido aleatorio de los datos de entrada y se produzca el temido *overfitting*.

Para evitar esta varianza, se pueden entrenar modelos múltiples y diversos y combinar sus predicciones [22]. Este procedimiento de *ensemble learning* da como resultado predicciones que son mejores que las de todos los modelos individuales [23]. Los modelos *Deep Learning* junto con *ensemble learning* son conocidos por sus inherentes beneficios en el proceso de toma de decisiones; la combinación de ambas estrategias puede minimizar de forma efectiva la varianza y mejorar el aprendizaje.

Por *Ensemble Learning* se entiende a las técnicas utilizadas para general algoritmos de aprendizaje artificial a partir de otros modelos ya previamente entrenados, combinando sus salidas a modo de un comité de tomadores de decisiones. Se asume que la decisión del grupo, combinando las decisiones individuales de forma correcta, debería tener una exactitud final mejorada sobre la de los individuos participantes en el comité, al calcular su promedio [24]. Diversos experimentos teóricos y experimentales han demostrado que los modelos *ensemble* habitualmente consiguen mejores exactitudes en sus resultados que los modelos específicos que los integran. Los modelos *ensemble* se han utilizado con éxito en diferentes áreas de salud como el diagnóstico de la diabetes tipo -II [25], la predicción del tiroides en las mujeres [26], desórdenes neurológicos [27] o predicción del cáncer [28].

Las estrategias de *ensemble learning* se aplican a menudo para obtener modelos predictivos estables y prometedores. Estos modelos son los ganadores de los desafíos Kaggle[5] y otras competiciones de *Machine Learning*. Este método también se ha utilizado para tareas de clasificación de imágenes médicas; en concreto se han utilizado redes neuronales convolucionales para la detección de tuberculosis en radiografías de pecho, con un valor de AUC (área bajo la curva de la ROC[6]) de 0,99 [29].

El objetivo principal de este trabajo es el de realizar un sistema de apoyo a la toma de decisiones médicas para detectar malaria en a partir de imágenes de frotis de

[5] https://www.kaggle.com/

[6] La curva ROC (acrónimo de *Receiver Operating Characteristic*, o Característica Operativa del Receptor) es una representación gráfica de la sensibilidad frente a la especificidad para un sistema clasificador binario según se varía el umbral de discriminación.

microscopio de glóbulos rojos utilizando como base un modelo EfficientNet [30] [31]. El modelo propuesto es un modelo ensemble compuesto por los 10 modelos parciales obtenidos en el proceso de validación cruzada de 10 iteraciones (10Fold) de un modelo base EfficientNet0. Hasta donde se ha podido investigar, no existe ningún estudio en la literatura que proponga una arquitectura EfficientNet para el diagnóstico automatizado de la malaria. Se han descrito numerosos experimentos y aplicaciones en la literatura, relacionados con este tema. Es importante liberar, y poner a disposición de la comunidad científica, los procedimientos y herramientas utilizados y permitir así que futuros investigadores puedan reproducir los resultados obtenidos y discutir sus conclusiones. Por tanto, en el trabajo se presentan todos los materiales y el código Python de los scripts utilizados compatibles con *Jupyter notebook*.

El modelo EfficientNet se puede utilizar para transferencia de aprendizaje [32] y es más eficaz que la mayoría de los otros modelos de CNNs como VGG16 o VGG19 [33], ResNet50 [34] o InceptionV3 [35]. La arquitectura EfficientNet se compone de 8 modelos, enumerados desde B0 a B7, donde cada número sucesivo del modelo implica variaciones incrementales con más parámetros y complejidad que los anteriores y, habitualmente, una exactitud esperada mayor. Además, los modelos EfficientNet utilizan técnicas de transferencia de aprendizaje para mejorar la velocidad de aprendizaje y ahorrar potencia de procesamiento. Por tanto, habitualmente ofrece mejores exactitudes que otros modelos, debido al uso de un escalado inteligente de resolución, anchura y profundidad. Este trabajo utiliza el modelo B0, que tiene el mínimo número de parámetros que, no obstante, lo consideramos adecuado para nuestro trabajo, ya que las imágenes médicas no son tan complejas como las naturales y tienen menos variabilidad. El número de parámetros para los modelos B1-B7 se incrementa en gran medida [31]. Además, este estudio utiliza un dataset separado para realizar la validación del modelo CNN, utilizando imágenes que no se han utilizado durante el proceso de entrenamiento y testeo. Se ha evaluado el algoritmo utilizando una validación cruzada estratificada de 10 iteraciones. Finalmente, se ha compuesto el modelo *ensemble*, compuesto por los 10 modelos intermedios generados durante la fase de validación cruzada. Esta aproximación mejora significativamente la exactitud de la clasificación de cualquiera de los modelos individuales. Los experimentos garantizan

la ausencia de *overfitting* por la validación realizada con un *dataset* separado, formado por imágenes que no se han utilizado en el proceso anterior de aprendizaje. El código fuente se muestra en uno de los apéndices.

El trabajo se articula como sigue: el apartado 2 realiza un estudio sobre la teoría en la que se fundamentan las redes neuronales, desde el perceptrón, pasando por las redes neuronales convolucionales, hasta llegar al modelo EfficientNet; también se explican las metodologías de optimización de las pérdidas y la "augmentación" de imágenes en el entrenamiento de las redes, procesos que han sido claves en los buenos resultados obtenidos. En el apartado 3 se resume el trabajo en el área encontrado en la literatura, haciendo foco en aquellos trabajos con los cuales más tarde se compararán los resultados de este trabajo. El apartado 4 describe los materiales y métodos utilizados en esta investigación. Los resultados del modelo EfficientNetB0 se presentan en el apartado 5, tanto de los modelos individuales como del modelo *ensemble*. En el apartado 6 se discute y compara los resultados del trabajo con otros disponibles en el estado del arte de la tecnología. Por fin, el apartado 6 presenta las conclusiones.

2 Fundamentos Teóricos

2.1 Redes Neuronales

2.1.1 El Perceptrón

Los ordenadores son realmente buenos cuando se trata de realizar operaciones aritméticas o de realizar una lista de instrucciones secuencial. Pero si queremos hacer otras cosas, quizás más interesantes, como escribir un programa que lea un documento escrito a mano, las cosas se complican [36].

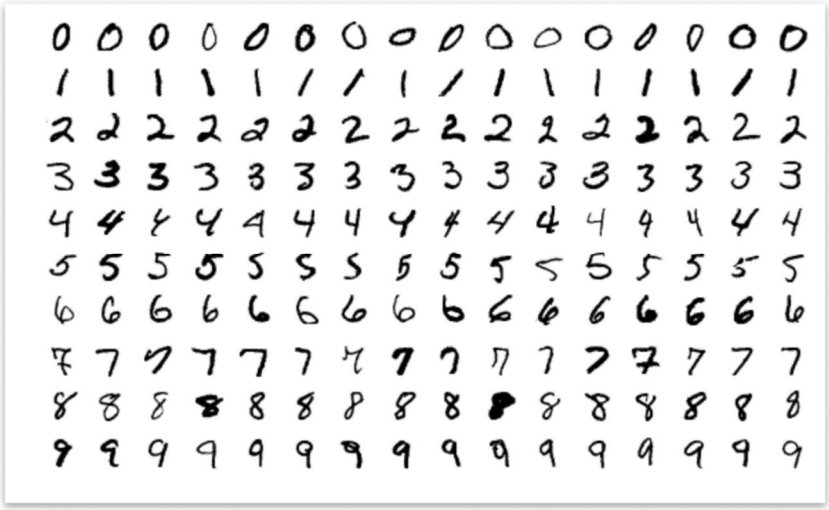


Figura 2-1. Imagen de la colección de dígitos escritos a mos del conjunto MNIST[7]

Aunque cada dígito de la Figura 2-1 está escrito de forma ligeramente diferente, los humanos podemos reconocer cada dígito de la imagen de forma correcta. Incluso podemos dar las características de cada clase, por ejemplo, un cero se caracteriza por un único bucle cerrado. Pero hay dígitos que no están completamente cerrados; además es más difícil distinguir un tres de un cinco, o entre un cuatro y un nueve. El proceso de especificar las características de cada número se va complicando. Muchos otros problemas son de este tipo: reconocimiento de objetos, comprensión del habla o traducción automática. Inicialmente no es nada sencillo escribir un programa que trate con estos problemas.

[7] Fuente: LeCun, 1998

Para tratar estos últimos problemas de reconocimiento de imágenes en los últimos años se han popularizado un conjunto de tecnologías denominadas como *Deep Learning*. *Deep Learning* es un subconjunto del campo más general de inteligencia artificial denominado *Machine Learning.* En *Machine Learning*, en vez de enseñar a un ordenador un conjunto enorme de reglas para resolver un problema, le damos un modelo con el cual puede evaluar los ejemplos del problema, y un pequeño conjunto de reglas para que pueda modificar el modelo cada vez que se equivoque en la tarea solicitada. Se espera que, con el tiempo, consiga un modelo optimizado que sea capaz de resolver el problema de forma precisa.

Para definir las anteriores afirmaciones de una manera matemática, definamos nuestro modelo como una función $h(\mathbf{x}, \theta)$. La variable $\mathbf{x}$ es un ejemplo del problema expresado en forma vectorial. Por ejemplo, si $\mathbf{x}$ es una imagen en escala de grises, los componentes del vector sería la intensidad de cada píxel como se muestra en la Figura 2-2 [37].

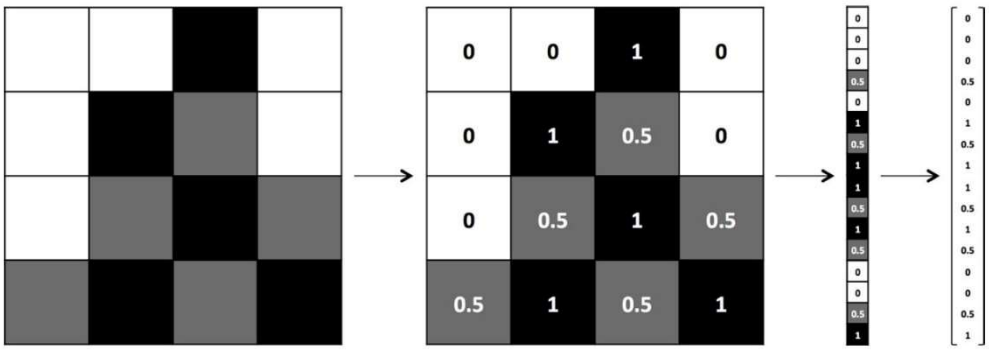


Figura 2-2. Proceso de vectorización de una imagen[8]

La variable θ es el vector de parámetros del modelo. El programa de *Machine Learning* trata de ajustar los valores de esos parámetros a medida que se enfrenta a más y más ejemplos del universo del problema que trata de resolver.

Para entender mejor el funcionamiento, veamos el modelo del *perceptrón* lineal que se viene usando desde la década de los 50's [38]. Supongamos que queremos predecir el resultado de un encuentro de nuestro equipo de futbol en función del número de horas de entrenamiento físico y de horas de siesta realizadas durante la semana anterior.

[8] Fuente: Patterson, 2016

Recogemos una multitud de resultados, $\mathbf{x} = [\mathrm{x}_1, \mathrm{x}_2]^{\mathrm{T}}$, con el número de horas de entrenamiento (x_1) y el número de horas de siesta (x_2), y si el equipo ha ganado (1) o no (-1). Nuestro objetivo entonces podría ser el de optimizar un modelo $h(\mathbf{x}, \theta)$ con un vector de parámetros $\theta = [\theta_0\ \theta_1\ \theta_2]^T$ tales que:

$$h(\mathbf{x}, \theta) = \begin{cases} -1 & \text{si} \quad \mathbf{x}^T \cdot \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} + \theta_0 < 0 \\ 1 & \text{si} \quad \mathbf{x}^T \cdot \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} + \theta_0 \geq 0 \end{cases}$$

Técnicamente, nuestro problema así expresado es un clasificador lineal que divide nuestro espacio de coordenadas en dos mitades [39]. El objetivo es el de encontrar el vector de parámetros θ tal que nuestro modelo haga predicciones correctas (-1 si pierde, 1 si gana) dado su comportamiento $\mathbf{x}$ durante la semana. Supongamos que nuestros datos son como los que se muestran en la Figura 2-3., donde $+$ representan partidos ganados, y – los perdidos o empatados.

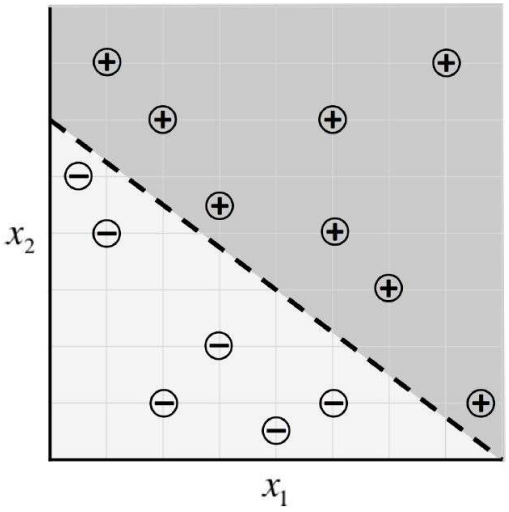


Figura 2-3. Datos de ejemplo para nuestro predictor del algoritmo perceptrón[9]

Podría resultar, que si seleccionamos $\theta = [-2\ 3\ 5]^T$, nuestro algoritmo de *machine learning* realiza una predicción perfecta en cada punto:

$$h(\mathbf{x}, \theta) = \begin{cases} -1 & \text{si} \quad 3x_1 + 5x_2 - 2 < 0 \\ 1 & \text{si} \quad 3x_1 + 5x_2 - 2 \geq 0 \end{cases}$$

[9] Fuente: Buduma, 2017

La forma de calcular es parámetros de θ se realiza mediante la técnica de optimización. Un optimizador consigue maximizar el rendimiento del modelo por medio de un ajuste iterativo de los parámetros hasta que el error en la predicción se minimiza, habitualmente por un proceso denominado *gradient descent* (apartado 2.2.1). Por otro lado, parece que el modelo lineal propuesto de dos variables es bastante limitado con respecto a lo que puede aprender. Por ejemplo, las distribuciones de datos que se describen en la Figura 2-4 no pueden describirse con el modelo del perceptrón lineal. Si el problema es incluso más complejo, como el reconocimiento de objetos o el análisis de textos, las relaciones del modelo son extremadamente no-lineales.

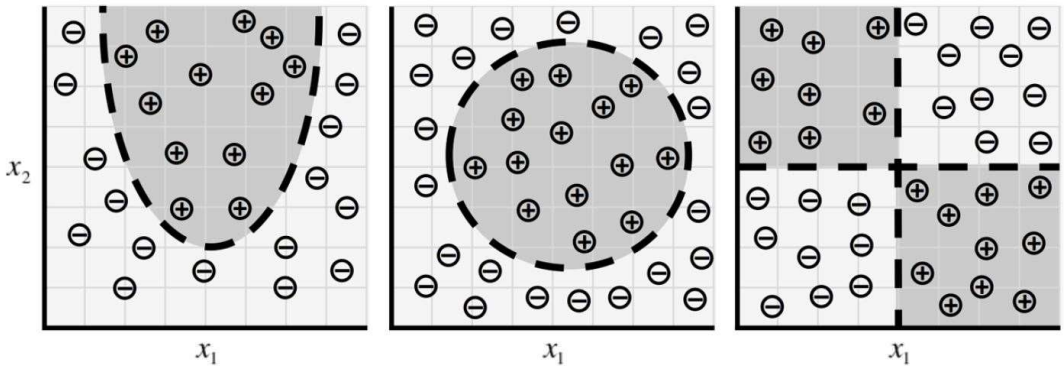


Figura 2-4. Datos complejos, donde el perceptrón lineal no puede ajustarse[10]

2.1.2 La Neurona

Para tratar estos problemas más complejos, los científicos han desarrollado modelos que recuerdan las estructuras del cerebro humano. La unidad básica del cerebro es la neurona. Aunque el núcleo de la neurona solamente mide entre 5 y 135 µm, tiene una media de 6.000 conexiones con otras neuronas [40]. La neurona esta optimizada para recibir información de otras neuronas, procesa esta información en un único sentido y envía el resultado a otras células.

[10] Fuente: Buduma, 2017

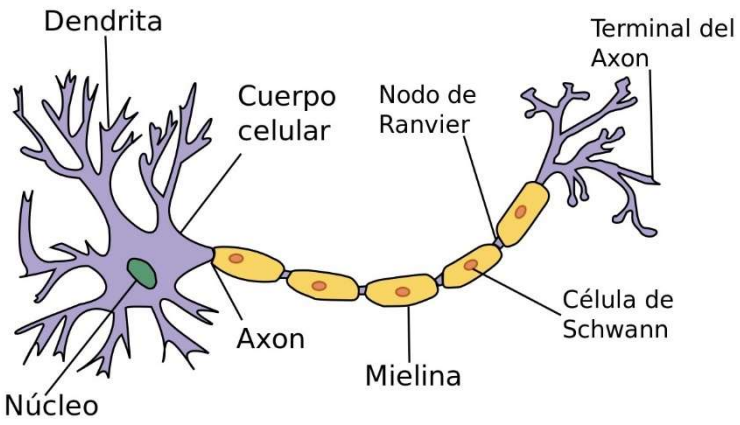


Figura 2-5. Descripción biológica de una neurona[11]

Como se muestra en la Figura 2-5, la neurona recibe sus entradas a través de las estructuras denominadas dendritas. Cada una de esas conexiones se refuerza o debilita de forma dinámica en función de la frecuencia de su uso (de esta forma se realiza el aprendizaje), y es la fuerza de cada conexión lo que determina su contribución al resultado de salida a través de un único axón. La suma de las contribuciones se realiza en el núcleo de la neurona.

Podemos trasladar el concepto de neurona a un modelo matemático para representarlo en un ordenador [41]. Definamos un número arbitrario de entradas, $x_1, x_2, \dots, x_n$, cada una de las cuales se multiplica por un peso específico $w_1, w_2, \dots, w_n$, para sumarlas todas ellas y producir la salida o *logit* de la neurona, $z = \sum_{i=0}^{n} w_i \cdot x_i$. En muchos casos, el modelo incluye una constante o *bias* en el sumatorio (que no se muestra en la Figura 2-6), el resultado o *logit* se pasa por una función, denominada de activación, para dar la salida que se pasa a las siguientes neuronas como entrada.

[11] Fuente Wikipedia

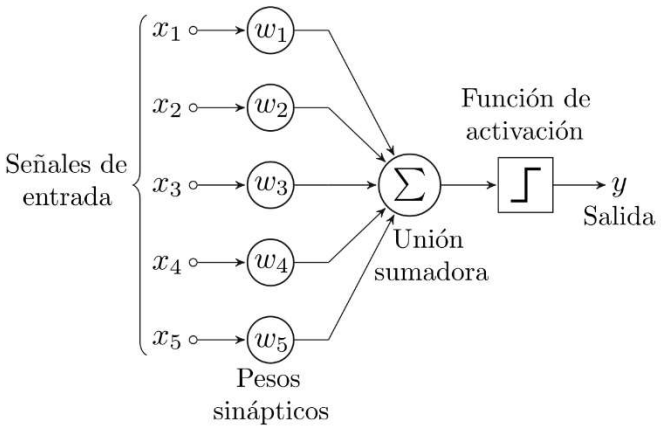


Figura 2-6. Esquema de neurona en Inteligencia Artificial[12]

Reformulando el problema. Si $\mathbf{x} = [x_1, x_2, \dots, x_n]$ es el vector de entrada, $\mathbf{w} = [w_1, w_2, \dots, w_n]$ el vector de pesos, podemos expresar la salida como una función del producto escalar de esos vectores de la forma:

$$y = f(\mathbf{x} \cdot \mathbf{w} + b)$$

donde con b expresamos el término constante o *bias*.

2.1.3 Redes neuronales progresivas

Aunque es fácil intuir que la neurona vista anteriormente es más potente que el perceptrón lineal, está muy lejos de poder representar problemas complejos, al igual que le pasaría a nuestro cerebro si estuviera formado por una sola neurona. Para poder emular problemas más complicados podemos conectar múltiples neuronas en capas, donde la información fluye de las capas inferiores a las superiores sin bucles ni realimentaciones, como se muestra en la Figura 2-7, donde se muestra lo que hemos denominado una red neuronal artificial [41]. La capa de la izquierda o capa de entrada introduce los datos de entrada a la red. La capa de la derecha calcula la salida final del sistema. Las capas intermedias, también denominadas capas ocultas, conectan la entrada con la salida. Hay que notar que en nuestro ejemplo no hay bucles ni realimentaciones, siendo el flujo de información de izquierda a derecha, de forma progresiva.

[12] Fuente: Wikipedia

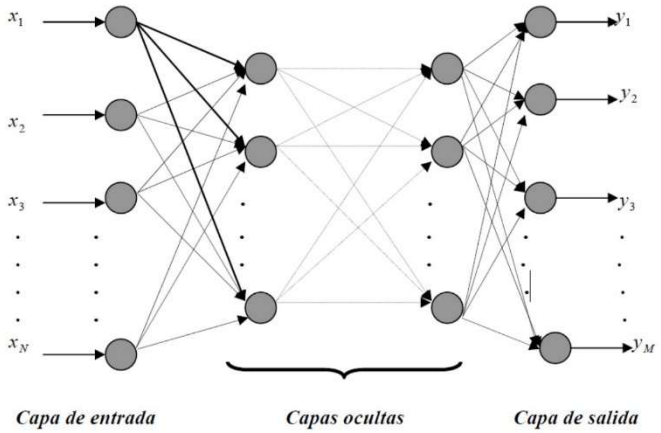


Figura 2-7. Ejemplo de una red neuronal progresiva[13]

Podemos expresar de forma matemática el funcionamiento de la capa *i-ésima* de la red neuronal, considerando el vector de las entradas a esa capa como $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ e $\mathbf{y} = [y_1, y_2, \ldots, y_m]$ l vector de las salidas resultante de propagar el vector de las entradas a través de esa capa. Podemos expresar de forma sencilla como un producto matricial si construimos la matriz de pesos $\mathbf{W}$ de tamaño $m \times n$ y un vector *bias* de tamaño n:

$$\mathbf{y} = f(\mathbf{W}^T \cdot \mathbf{x} + \mathbf{b})$$

donde f es la función de activación.

2.1.4 Función de activación

Las tres funciones de activación que se utilizan en la práctica introducen no-linealidades en el proceso de cálculo. La primera de ellas es la función *sigmoide* que tiene una forma:

$$f(z) = \frac{1}{1 + e^{-z}}$$

De forma intuitiva, quiere decir que cuando el *logit* es muy negativo, la salida de la neurona es cercana a 0. Cuando el *logit* es grande, la salida de la neurona se aproxima a 1. Entre los dos valores intermedios la función adquiere una función de *S* como se ve en la Figura 2-8 (a).

[13] Fuente: http://inteligenciaartificialespammfl.blogspot.com/

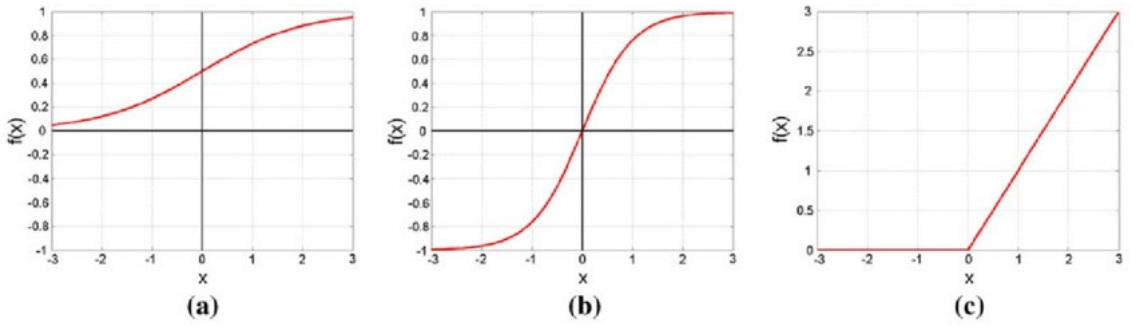


Figura 2-8. Funciones de activación no lineales. a) Sigmoide b) Tanh c) ReLu[14]

La segunda función es la de la tangente hiperbólica, $\tanh()$, Figura 2-8 (b). Tiene una no-linealidad similar a la anterior, pero en vez de tener un rango de 0 a 1, la salida de la neurona recorre el rango de -1 a 1, con la característica deseable de estar centrada en el 0.

$$f(z) = \tanh(z)$$

La tercera es la función denominada *restricted linear unit (ReLU),* Figura 2-8 (b), con forma de un stick de hockey, definida por la función:

$$f(z) = \max(0, z)$$

La función *ReLU* es la función activación preferida para muchas tareas, a pesar de alguno de sus inconvenientes [42], como es que no es derivable en el origen.

Una función más compleja se utiliza cuando la función de salida que se desea es una distribución de probabilidad sobre un conjunto de clases mutuamente exclusivos. Por ejemplo, clasificar los dígitos escritos a mano, donde las clases (0…9) son mutuamente exclusivos, y es difícil tener una confianza del 100% en la clasificación. Si se utilizan distribuciones de probabilidad, se da una mejor idea de la confianza en la predicción que se realiza; como resultado se obtiene un vector de probabilidades $[p_0, p_1, \dots, p_9]$, que cumplen:

$$\sum_{i=0}^{9} p_i = 1$$

[14] Fuente: https://www.researchgate.net/

Esto se consigue con una función de activación especial denominada capa *softmax*. A diferencia de otros tipos de funciones de activación, la salida un este tipo de neurona depende de la salida de las otras neuronas que se encuentran en la misma capa, ya que la suma de todas las salidas debe ser igual a 1. Si llamamos z_i al *logit* de la i-ésima neurona *softmax*, podemos conseguir la normalización de este tipo utilizando la función:

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Una predicción de confianza tendrá una única componente del vector con un valor próximo a 0, mientras las otras permanecerán próximas a 0. Una predicción débil, tendrá varios de sus valores aproximadamente iguales.

2.2 Entrenamiento de las redes neuronales

Una vez definido el modelo, se trataría de identificar todos los coeficientes de la red neuronal para que se ajuste a la tarea que le vamos a encomendar. Si tenemos un conjunto grande de ejemplos para entrenar. Dada una red neuronal arbitraria, con unos pesos predefinidos, podemos calcular la salida de la red para el ejemplo i-ésimo usando la fórmula del apartado 2.1.3. El objetivo es entrenar a las neuronas de manera que seleccionemos los mejores pesos posibles, los pesos que minimizan el error que realizamos con las muestras de ejemplo. En este caso, pongamos que queremos minimizar el valor cuadrático del error cometido sobre todas las muestras disponibles. Matemáticamente, si sabemos que $t^{(i)}$ es la respuesta correcta para la muestra i e $y^{(i)}$ el valor calculado por la red neuronal, se trata de minimizar el valor de la función de error, E:

$$E = \frac{1}{2}\sum_i \left(t^{(i)} - y^{(i)}\right)^2$$

Ese valor cuadrático es cero cuando nuestro modelo hace precisiones correctas para cada ejemplo del entrenamiento. Además, cuanto más cercano sea E a 0, mejor es nuestro modelo. Por tanto, nuestro objetivo se seleccionar nuestro vector paramétrico θ tal que E sea lo más cercano posible.

Si nuestra red neuronal fuera lineal (para lo que tendría que ser lineal nuestra función de activación) el problema tendría una solución analítica. Desgraciadamente la mayoría de los problemas que queremos resolver no tienen una naturaleza lineal, por lo que tenemos que buscar métodos alternativos para buscar esos vectores óptimos.

2.2.1 Gradient Descent

Podemos visualizar como se puede minimizar el error cuadrático cometido simplificando el problema. Digamos que nuestra red neuronal sólo tiene dos entradas y por tanto sólo dos pesos $[w_1, w_2]$. Podemos representar un espacio tridimensional donde las dos direcciones horizontales se correspondan con los pesos w_1y w_2y la dimensión vertical se corresponda con el valor de la función de error. En este espacio, los puntos de la dimensión horizontal se corresponden con los diferentes valores de los pesos, y la dimensión vertical con el error cometido. Si calculamos el error sobre todos los puntos, obtendremos una superficie en el espacio tridimensional, similar a la Figura 2-9.

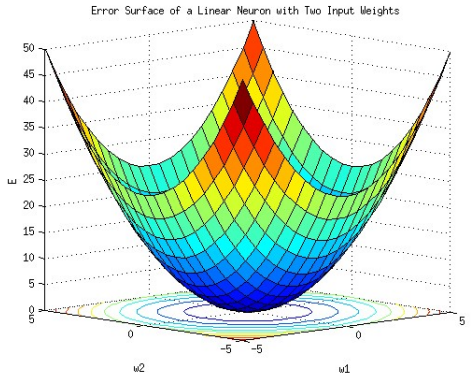


Figura 2-9. Error cometido por un neurona con dos entradas[15]

Para una neurona con k pesos diferentes, el dibujo sería otro paraboloide elíptico, pero de $k + 1$ dimensiones.

El método *Gradient Descent* requiere calcular la derivada de la función de error con respecto de todos los pesos de la red. Esto se realiza habitualmente utilizando *backpropagation*: si suponemos que la red neuronal sólo tiene una salida (la última capa sólo tiene una neurona), y denominamos a la función del error cuadrático como:

[15] Fuente: Wikimedia

$$E = L(t, y)$$

con:

- E la pérdida (error) para la salida y y el valor real t, para una muestra concreta.
- t es el calor objetivo para esa muestra.
- y es la salida de la neurona de la última capa

Para cada neurona j, su salida o_j se define como:

$$o_j = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

donde φ es una función de activación no-lineal y diferenciable (válida también para la *ReLU*) excepto en el 0. Utilizando como función de activación la *sigmoide* (apartado 2.1.4):

$$\varphi(z) = \frac{1}{1(1 + e^{-z})}$$

Cuya derivada se puede expresar como:

$$\frac{d\varphi(z)}{dz} = \varphi(z) \cdot \big(1 - \varphi(z)\big)$$

Siguiendo la notación indicada en la Figura 2-10, el *logit* o net_j de una neurona, es la suma ponderada por los pesos de las salidas o_k de las neuronas previas. Si la neurona está en la primera capa, o_k es el valor x_i de la entrada i de la red. El número de entradas de una neurona es n. La variable w_{kj} representa el peso entre la neurona k de la capa previa y la neurona j de la capa actual.

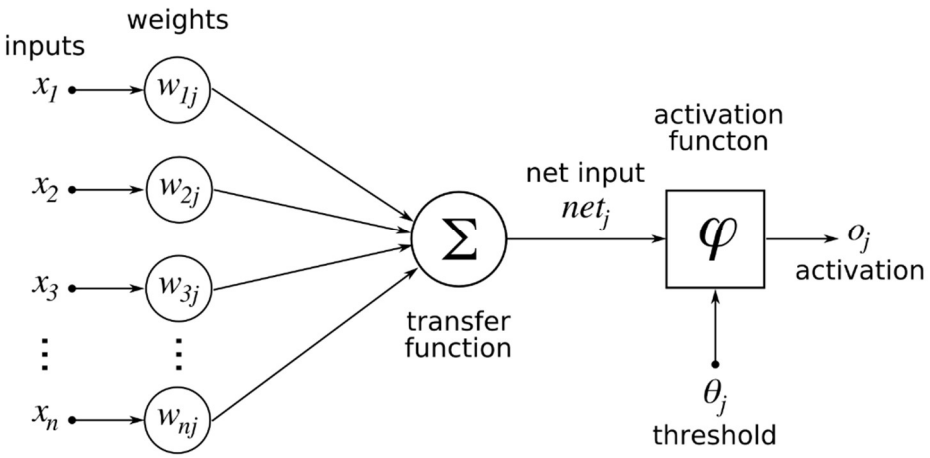


Figura 2-10. Esquema de la neurona para la notación de Gradient Descent[16]

Podemos utilizar la regla de la cadena dos veces para calcular la derivada del error con respecto al peso w_{ij}:

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta o_j}\frac{\delta o_j}{\delta w_{ij}} = \frac{\delta E}{\delta o_j}\frac{\delta o_j}{\delta net_i}\frac{\delta net_i}{\delta w_{ij}}$$

La última derivada parcial de la ecuación anterior, sólo uno de los términos lineales de la función depende de w_{ij}.

$$\frac{\delta net_i}{\delta w_{ij}} = \frac{\delta}{\delta w_{ij}}\left(\sum_{k=1}^{n} w_{kj}o_k\right) = \frac{\delta}{\delta w_{ij}} w_{ij}o_i = o_i$$

Si la neurona considerada estuviera en la primera capa, entonces $o_i = x_i$.

La derivada de la salida de la función de activación de la neurona *j* con respecto a su entrada es simplemente la derivada de la función de activación:

$$\frac{\delta o_j}{\delta net_i} = \frac{\delta \varphi(net_j)}{\delta net_i}$$

que para el caso de la *sigmoide*:

$$\frac{\delta o_j}{\delta net_i} = \frac{\delta \varphi(net_j)}{\delta net_i} = \varphi\big(net_j\big)\Big(1 - \varphi\big(net_j\big)\Big) = o_j(1 - o_j)$$

[16] Fuente: WIKIMEDIA

La primera derivada parcial es directa si la neurona está en la capa de salida, ya que $o_j = y$ y:

$$\frac{\delta E}{\delta o_j} = \frac{\delta E}{\delta y}$$

Si la función utilizada es la del valor cuadrático del error entonces podemos escribir:

$$\frac{\delta E}{\delta o_j} = \frac{\delta E}{\delta y} = \frac{\delta}{\delta y}(t-y)^2 = y - t$$

Pero si *j* es una capa arbitraria del interior, encontrar la derivada de E respecto de o_j es menos obvio. Considerando E como una función de las entradas de todas las neuronas $L = \{u, v, \dots, w\}$ que reciben su entrada de la neurona *j*:

$$\frac{\delta E}{\delta o_j} = \frac{\delta E(net_u, net_v, \dots, net_w)}{\delta o_j}$$

y realizando la derivada total con respecto a o_j, se puede obtener una expresión recursiva para la derivada:

$$\frac{\delta E}{\delta o_j} = \sum_{l \in L}\left(\frac{\delta E}{\delta net_l}\frac{\delta net_l}{\delta o_j}\right) = \sum_{l \in L}\left(\frac{\delta E}{\delta o_l}\frac{\delta o_l}{\delta net_l}\frac{\delta net_l}{\delta o_j}\right) = \sum_{l \in L}\left(\frac{\delta E}{\delta o_l}\frac{\delta o_l}{\delta net_l}\frac{\delta net_l}{\delta o_j}\right) =$$

$$= \sum_{l \in L}\left(\frac{\delta E}{\delta o_l}\frac{\delta o_l}{\delta net_l}w_{jl}\right)$$

Por tanto, la derivada con respecto a o_j se puede calcular si se conocen todas las derivadas de las salidas de la capa siguiente, o_l, las más próximas a la neurona considerada. Operando con las ecuaciones anteriores:

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta o_j}\frac{\delta o_j}{\delta net_j}\frac{\delta net_j}{\delta w_{ij}} = \frac{\delta E}{\delta o_j}\frac{\delta o_j}{\delta net_j}o_i$$

$$\frac{\delta E}{\delta w_{ij}} = o_i \delta_j$$

con

$$\delta_j = \frac{\delta E}{\delta o_j}\frac{\delta o_j}{\delta net_j} = \begin{cases} \frac{\delta L(o_j, t)}{\delta o_j}\frac{\delta \varphi(net_i)}{\delta net_j} & si\ j\ es\ una\ neurona\ de\ salida \\ \left(\sum_{l\in L} w_{jl}\delta_l\right)\frac{\delta \varphi(net_i)}{\delta net_j} & si\ j\ es\ una\ neurona\ interior \end{cases}$$

Si φ es la función de activación, y el error medido es el error cuadrático:

$$\delta_j = \frac{\delta E}{\delta o_j}\frac{\delta o_j}{\delta net_j} = \begin{cases} (o_j - t_j)o_j(1 - o_j) & si\ j\ es\ una\ neurona\ de\ salida \\ \left(\sum_{l\in L} w_{jl}\delta_l\right)o_j(1 - o_j) & si\ j\ es\ una\ neurona\ interior \end{cases}$$

Para actualizar el peso w_{ij} usando el método de *gradient descent*, se debe escoger una tasa de aprendizaje (*learning rate*), $\eta > 0$. El cambio de los pesos refleja el impacto sobre el error, E de una variación, incremento o decremento en w_{ij}. Si $\frac{\partial E}{\partial w_{ij}} > 0$, un incremento de w_{ij} incrementa E; recíprocamente, si $\frac{\partial E}{\partial w_{ij}} < 0$, un incremento de w_{ij} disminuye E. Este nuevo incremento del peso, Δw_{ij} se añade al anterior valor del peso, y el producto del *learning rate*, multiplicado por -1 garantiza que w_{ij} cambia de modo que siempre disminuye E; de otra forma, en la ecuación siguiente, $-\frac{\eta \partial E}{\partial w_{ij}}$ siempre cambia w_{ij} de tal manera que disminuye E.

$$\Delta W_{ij} = -\frac{\eta \partial E}{\partial w_{ij}} = -\eta o_i \delta_j$$

2.2.2 Función de pérdidas

La función de pérdidas o *loss function* es una función que relaciona una o más variables a un número real que intuitivamente representa al "coste" asociado a esos valores. Para la función de *backpropagation*, la *los function* calcula la diferencia entre la salida real de la red y su salida esperada, después de que un ejemplo se ha propagado a través de la red.

La expresión matemática de la *loss function* debe cumplir dos condiciones para que pueda ser utilizada en el algoritmo de *backpropagation* [43]; la primera es que pueda ser escrita como un promedio sobre los valores de error E_x de n ejemplos de entrenamiento individuales, x:

$$E = \frac{1}{n}\sum_x E_x$$

El motivo es que el algoritmo de *backpropagation* calcula la función del error del gradiente para una única muestra de ejemplo, que necesita ser generalizada para la función de error total. La segunda condición es que pueda ser escrita en función de las salidas de la red neuronal.

Como ejemplo, sean y, y' vectores de $\mathbb{R}^n$. Elijamos una función de error $E(y, y')$ que mida la diferencia entre las dos salidas. La elección habitual es el cuadrado de la distancia euclídea entre los vectores y e y':

$$E(y, y') = \frac{1}{2}\|y - y'\|^2$$

La función de error sobre n muestras de entrenamiento se puede escribir como el promedio del error de los ejemplos individuales:

$$E(y, y') = \frac{1}{2n}\sum_x \|y - y'\|^2$$

Como limitaciones del método de aprendizaje del *Gradient Descent* con *Backpropagation* podemos indicar las siguientes:

- No se puede garantizar encontrar el mínimo global de la función, sino sólo un mínimo local; por otro lado, tiene problemas para cruzar las "mesetas" de las curvas de las funciones de error, debido a posibles no-convexidades de esas curvas.
- El aprendizaje *backpropagation* no requiere una normalización de los vectores de entrada; sin embargo, en muchos casos la normalización mejora el rendimiento.
- *Backpropagation* necesita que las derivadas de la función de activación se conozcan cuando se diseña la red neuronal.

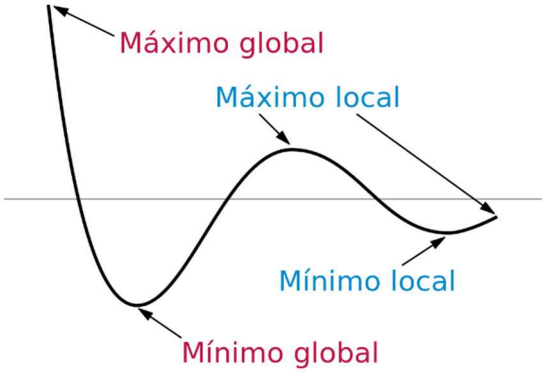


Figura 2-11. Gradient descent puede encontrar un mínimo local[17]

2.3 Redes Neuronales Convolucionales

Este tipo de redes neuronales han supuesto una revolución en el campo de la visión artificial, desde que en el año 2012 Alex Krizhevsky [44] las usara para ganar la competición de *ImageNet* (también conocidas como las olimpiadas anuales de la visión por ordenador) mejorando el récord del error de clasificación desde el 26% al 15%. Desde entonces, la mayoría de las grandes compañías han venido utilizando esta tecnología: Facebook utiliza las redes neuronales para sus algoritmos de etiquetado automáticos; Google; para su búsqueda de fotos; Amazon para su recomendación de productos; Pinterest para la personalización de la página *home*; e Instagram para su infraestructura de búsqueda.

Sin embargo, el caso de uso de estas redes más extendido es para el procesamiento de las imágenes, en concreto la clasificación de las imágenes.

La clasificación de imágenes es la tarea de asignar cada imagen de entrada a una clase de salida (perro, gato) o a una probabilidad de clases de salida, que mejor describa la imagen. Para los humanos, esta tarea de reconocimiento es una de las primeras habilidades que aprendemos desde el momento que nacemos y es una que no requiere de ningún esfuerzo para los adultos. Incluso sin pensarlo dos veces, somos capaces de identificar en el entorno los objetos que nos rodean, rápidamente y de forma fluida. Estas capacidades de reconocimiento no son compartidas por las máquinas.

[17] Fuente: Wikimedia

Cuando un ordenador ve una imagen (toma una imagen como entrada), ve una matriz de píxeles, cada una con un valor entero; se enfrenta a una matriz de 32x32x3 números (el 3 se refiere a los valores de color RGB). Por ejemplo, si suponemos que el ejemplo es una imagen a color en formato JPG con un tamaño de 480x480 píxeles. La correspondiente matriz es de 480x480x3. Cada uno de estos elementos es un número entero entre 0 y 255 que describe la intensidad del píxel en ese punto. Estos números, aunque son ininteligibles para nosotros, son la única entrada disponible para los ordenadores. La idea es que al dar a la red neuronal convolucional esta matriz de números y su salida sea la probabilidad de que la imagen pertenezca a una clase determinada (0,80 para gato, 0.20 para perro).

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

Figura 2-12. Imagen real; lo que "ve" el ordenador[18]

Se pretende que el algoritmo diferencie entre todas las imágenes dadas, y encontrar cuáles son las características únicas que definen un perro, y cuáles otras definen un gato. Este es el proceso que discurre en nuestras mentes de forma inconsciente de forma natural; por ejemplo, cuando vemos una fotografía de un perro, inmediatamente conocemos sus características como que tiene cuatro patas. De forma similar, los ordenadores son capaces de realizar clasificaciones de imágenes examinando las características de bajo nivel como bordes y curvas, y a partir de ahí construir conceptos más abstractos a través de las capas convolucionales.

[18] Fuente: https://dzone.com/

Siendo más específicos, las redes neuronales convolucionales (CNN) toman una imagen, la pasan a través de diferentes procesos como convoluciones, no-lineales, *pooling* (o *downsampling*) y *fully connected layers*; cada uno de estos procesos se realiza en una capa diferente de la red neuronal profunda. La salida puede ser una clase simple o un vector de probabilidades como se ha indicado anteriormente.

2.3.1 Capa Convolucional

La primera capa de una CNN es siempre una capa convolucional. La entrada a esta capa es un array de valores de píxeles de mxnx3 (siendo el ancho y el alto de la imagen en píxeles, supongamos por claridad que sea 32x32x3 como en la Figura 2-13). Se define otra matriz más pequeña (llamada filtro o *kernel*), en el ejemplo de tamaño 5x5 que contiene un conjunto de números que son los parámetros de ese filtro. Si tenemos en cuenta que en realidad hay 3 capas de entrada, debido a los tres colores, el filtro es realmente una matriz de 5x5x3. Se comienza situando el filtro sobre la esquina superior izquierda y utilizamos ese filtro como *convolución* sobre toda el área de la imagen, multiplicando los valores de la imagen por los coeficientes del filtro, y cuya suma nos da un nuevo píxel de la salida (la primera capa "escondida"). Desplazando píxel a píxel el filtro por toda el área de la imagen, cada posición nos dará otro píxel de salida, con lo que tendremos una matriz de tamaño 28x28, que son precisamente las 784 diferentes posiciones en las que se puede situar un filtro de 5x5 sobre las 32x32 píxeles de la imagen de entrada. A esta matriz de salida se la denomina *activation map* o *feature map*.

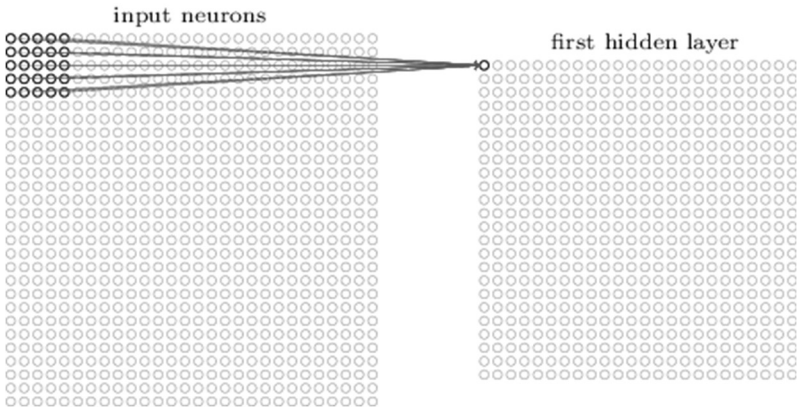


Figura 2-13. Esquema de un filtro convolucional de 5x5[19]

Si tuviéramos dos filtros diferentes de 5x5x3 entonces el tensor de salida tendría un tamaño de 28x28x2; al utilizar más filtros somos capaces de preservar mejor las dimensiones espaciales. Cada uno de estos filtros puede ser entendido como un identificador de características, entendiendo a este primer nivel las características como líneas rectas, colores simples o curvas. Modificando los valores de los coeficientes de manera adecuada en el filtro de 5x5, la salida dependerá de los valores de los píxeles subyacentes [45].

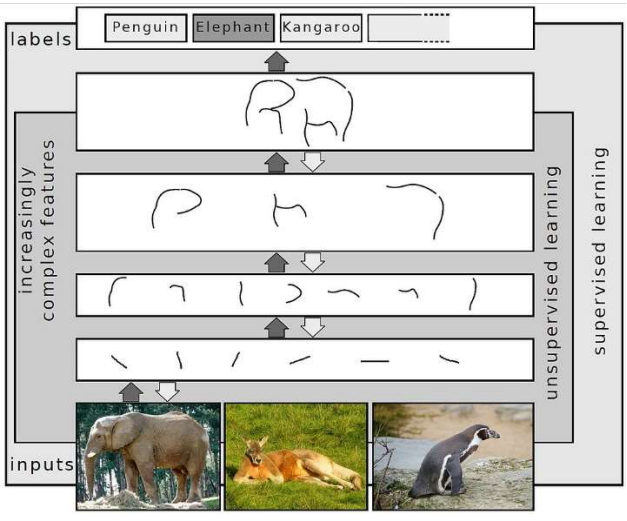


Figura 2-14. Extracción de características básicas de una imagen[20]

[19] Fuente: Nielsen 2015
[20] Fuente: Schutlz 2012

Básicamente el proceso consiste en implementar en el filtro la forma básica que queremos detectar, y cuando la zona de la imagen se parezca a esa forma, todos los coeficientes sumarán "en fase", dando como salida un valor muy alto de salida.

Evidentemente la explicación se ha simplificado mucho. En la Figura 2-15 se muestran algunos ejemplos de los valores reales de los filtros de la primera capa convolucional de una red ya entrenada; se ve que cada filtro trata de descubrir un patrón, sea línea, color o textura. No obstante, el argumento es el mismo: los filtros de la primera capa convolucionan con la imagen de entrada y se "activan" cuando la característica específica que tratan de detectar se encuentra debajo [46].

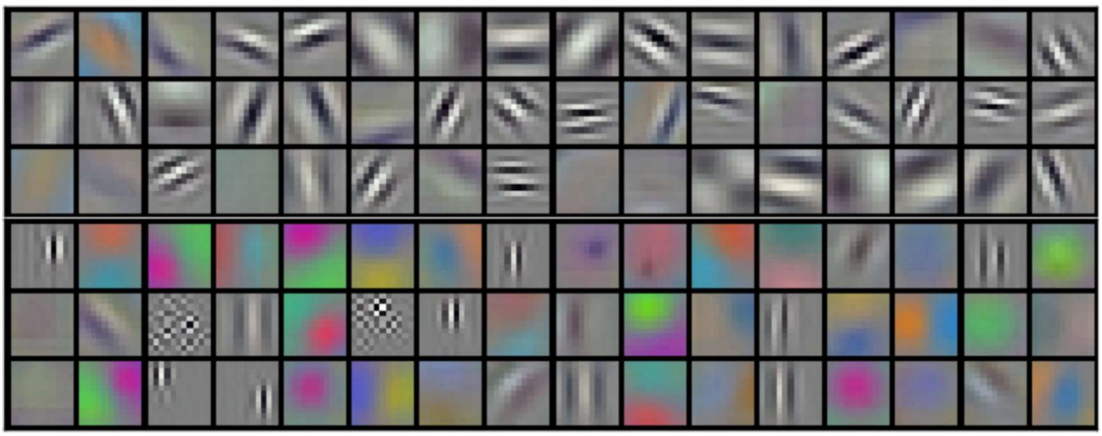


Figura 2-15. Visualización de los filtros para capas convolucionales[21]

2.3.2 Capas interiores

En una arquitectura de red neural existen otras capas que se entremezclan con estas capas convolucionales. En general, su función es la de producir no-linealidades y preservar las dimensiones para mejorar la robustez de la red y prevenir el sobreentrenamiento. Una arquitectura clásica de red convolucional se parecería a:

```
Input → Conv → ReLU → Conv → ReLu → Pool → Relu →
      → Conv → ReLU → Pool → FullyConnected
```

Hemos visto que la primera capa convolucional (`Conv`) se utiliza para detectar características básicas como rectas y curvas. Pero se necesita detectar características de

[21] Fuente: Zeiler 2013

alto nivel, como orejas, ojos o patas, para lo que necesitamos capas convolucionales adicionales dentro de la red neuronal. Pero la entrada de estas nuevas capas (pongamos la segunda) son precisamente la salida de las anteriores, por lo que podrán detectar características de nivel "superior", como círculos, cuadrados o texturas básicas, más allá de las líneas, curvas y colores que detectaba la primera capa.

Según progresamos en la red neuronal y atravesamos más capas convolucionales, obtendremos mapas de activación que representan características cada vez más complejas. Al final de la red, podremos tener diferentes filtros en los que cada uno se active sólo con una característica de alto nivel de entrada, como si tiene dos ojos o si tiene pelo en alguna parte. Otra característica interesante es que los filtros de las capas más profundas reciben la información de áreas cada vez más extensas de la imagen de entrada, es decir, responden a áreas de píxeles cada vez más grandes de la imagen de entrada.

2.3.3 Capa de salida

Al final de la red neural existe lo que se denomina una *fully connected layer* (capa conectada completamente). Básicamente esta capa lo que hace es tomar como entrada la salida de la última capa convoluciones, con las características de más alto nivel detectadas en la red, y da como salida un vector N dimensional, donde N es el número de clases para la clasificación a la que ha sido entrenada. Por ejemplo, si queremos un programa de clasificación de dígitos, N sería 10. Y si el vector resultante es [0.1 0.75 0 0 0 0 0 0 0 0.15], entonces nos indica que hay un 10% de probabilidades de que sea el dígito 0, 75% de que sea el 1 y un 15% de que sea el 9.

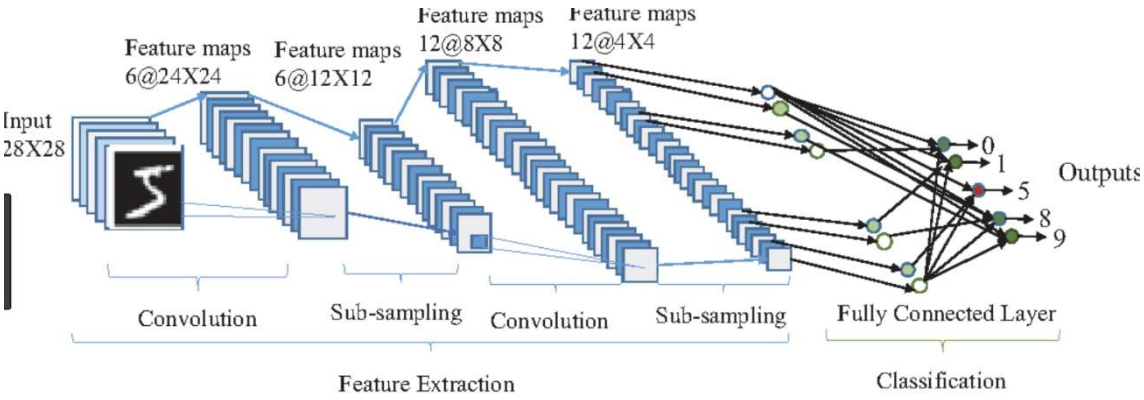


Figura 2-16. Red neuronal convolucional completa[22]

2.3.4 Stride y Padding

Hay dos parámetros fundamentales que podemos cambiar para modificar el comportamiento de cada capa convolucional, además del tamaño del filtro debemos elegir los parámetros conocidos como *stride* y *padding*.

El *stride* controla como el filtro convoluciona con el tensor de entrada. En el ejemplo anterior, el filtro convolucionaba con un desplazamiento de un píxel cada vez. La cantidad o el salto con el que el filtro se desplaza se denomina *stride*. El *stride* se define habitualmente de forma que el tensor de salida sea un entero y no una fracción. Es fácil ver que, si el *stride* pasa de 1 a 2, el tensor de salida encoge y se producen tensores de menor tamaño. Generalmente, el *stride* se incrementa para que los campos de salida estén menos solapados y menores dimensiones espaciales.

Por otro lado, hemos viso que, al aplicar los filtros, tal como los hemos definido, las dimensiones espaciales de salida disminuyen. Aplicando un filtro de 5x5 a una imagen de 32x32, nos da una salida de 28x28. Si aplicamos varias capas convolucionales sucesivas pudiera ser que el tamaño se redujera más de lo que quisiéramos. Esto se evita con el *padding* o relleno de bordes a la salida (con valores de 0) para restaurar aquellos píxeles "perdidos". El tamaño del tensor de salida viene dado por la fórmula:

$$O = \frac{W - K + 2P}{S} + 1$$

[22] Fuente: StackOverflow

Con O el ancho/alto de salida, W el ancho/alto de la entrada, K el tamaño del filtro, P es el *padding* y S el *stride*.

2.3.5 Capas ReLU

Después de cada capa convolucional, es habitual aplicar una capa de activación no-lineal, como veíamos en las redes neuronales convencionales. El propósito de esta capa es el de introducir no-linealidades en el sistema lineal de las capas convolucionales. La capa más popular en redes convolucionales es la ReLU (apartado 2.1.4), frente a las más tradicionales *sigmoide* o $\tanh()$, ya que es computacionalmente más eficiente (entrena las redes más rápido) y además ayuda a aliviar el problema del *vanishing gradient*[23] por el que las capas más bajas de la red neuronal entrenan de forma muy lenta.

La capa de activación ReLU aplica la función $f(x) = \max(0, x)$ a los valores de su entrada. Básicamente, esta capa simplemente cambia todos los valores de activación negativos a 0. Este tipo de capas ReLU incrementa las propiedades no-lineales del modelo y de la red completa sin afectar el campo de recepción de la capa convolucional.

2.3.6 Capas Pooling

A continuación de algunas de las capas ReLU, muchos científicos de datos utilizan *pooling layers*. Se utilizan para realizar *downsampling*, o reducir el tamaño de los tensores que viajan por la red. Para este tipo de capas, existen varias opciones, siendo la más popular la denominada *maxpooling*: básicamente consiste en un filtro (normalmente de tamaño 2x2) con un *stride* de la misma longitud. Se aplica entonces al tensor de entra y como salida ofrece el valor máximo en cada subregión definida por el filtro.

[23] https://en.wikipedia.org/wiki/Vanishing_gradient_problem

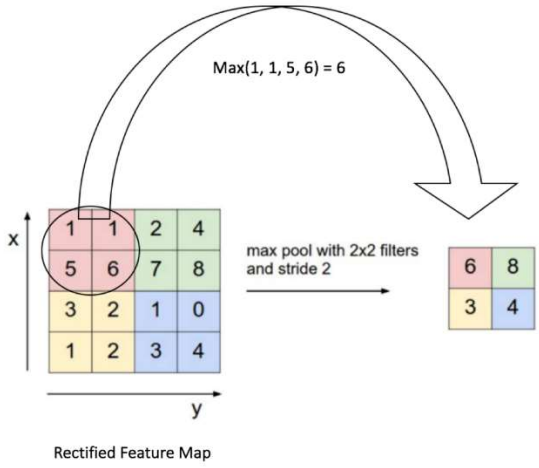


Figura 2-17. Ejemplo de maxpool con un filtro de 2x2 y un paso de 2[24]

Otras opciones para las capas de *pooling* son el promedio o la normalización L2 [43].

La idea intuitiva que subyace para estas capas es que una de las capas de la red neuronal conoce que una característica específica se encuentra en el tensor de entrada (generalmente una de las capas de activación más altas); su localización exacta no es tan importante como su posición relativa con respecto a las otras características. Es fácil ver que este tipo de capas reducen drásticamente las dimensiones espaciales (el ancho y el alto, aunque no la profundidad). Con lo que se consigue que la cantidad de parámetros o pesos se reduzca un 75%, disminuyendo con ello los costes de computación. Además, se logra disminuir el *overfitting*[25], o sobreentrenamiento de la red, que ocurre cuando las mismas muestras de entrada se han utilizado muchas veces durante el entrenamiento y el modelo no generaliza bien para otras muestras.

2.3.7 Capas Dropout

Las capas *dropout* se utilizan precisamente para evitar el *overfitting* visto en el apartado anterior. Para evitar que los pesos de la red se sintonicen excesivamente a las muestras concretas que se utilizan durante el entrenamiento, y no a sus características, este tipo de capas descartan (*dropout*) un conjunto aleatorio de activaciones en esa capa asignándolas un valor de cero a la salida [47]. Este simple procedimiento fuerza a la red a ser redundante, es decir, la red debe ser capaz de dar la clasificación correcta incluso

[24] Fuente: The Data Science Blog
[25] https://elitedatascience.com/overfitting-in-machine-learning

si algunas de las activaciones se desconectan, con lo que se asegura de que no hay sobreentrenamiento. Es importante hacer notar que este tipo de capas se utilizan durante el entrenamiento y no durante el testeo o la validación.

2.3.8 Capa Network in Network

La capa '*network in network*' hace referencia a un tipo de capa convolucional donde se utiliza un filtro 1x1. La utilidad de este filtro se explica si se tiene en cuenta que habitualmente las imágenes de entrada y, por tanto, los tensores que se propagan a través de la red, tienen tres dimensiones, con profundidad N (habitualmente 3 por los tres componentes RGB) además del ancho y el alto. Estas capas realizan una convolución en la dimensión profundidad, siendo un filtro de dimensión 1x1xN, donde N es el número de filtros aplicados a la capa. De hecho, estas capas realizan una convolución de N elementos, donde N es el tamaño de la dimensión de la profundidad del tensor de entrada.

2.3.9 Aplicaciones de las redes neuronales convolucionales

Hasta ahora se ha hecho énfasis en la tarea de la clasificación de imágenes, o el proceso de tomar una imagen como entrada y dar como salida el número de categoría al que pertenece entre un número de ellas predeterminada. Sin embargo, si la tarea consiste en una localización de objetos, el trabajo no es sólo producir una etiqueta de clase, sino también las coordenadas de un rectángulo que delimite dónde se encuentra ese objeto dentro de la imagen [48]

También es habitual realizar la tarea de detección de objetos, en donde la localización debe hacerse para todos los objetos de la imagen, con lo que tendremos una salida con múltiples rectángulos de coordenadas y múltiples etiquetas de clasificación.

Finalmente, la segmentación de objetos tiene por objeto estimar la etiqueta de la clase, así como definir el contorno del objeto dentro de la imagen.

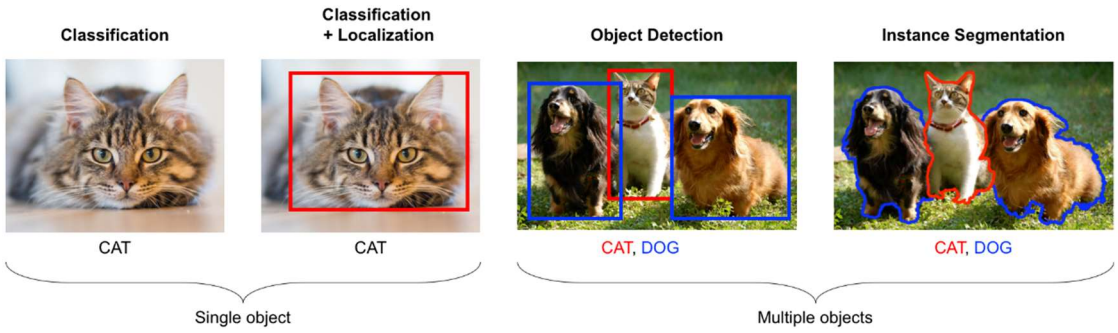


Figura 2-18. Ejemplos de localización, detección y segmentación de objetos[26]

2.3.10 Transferencia de aprendizaje

No es necesario tener la cantidad de datos que posee Google para crear redes neuronales de aprendizaje profundo efectivas. Aunque habitualmente se necesitan cantidades ingentes de datos para crear una red *Deep Learning*, el concepto de *transfer learning* ha ayudado a disminuir la demanda de datos y de computación en el entrenamiento de estos modelos. *Transfer learning* es el proceso de tomar un modelo previamente entrenado (donde los pesos y parámetros de una red han sido entrenados con una gran base de datos por un tercero) y "sintonizar" el modelo con un conjunto de datos propio. La idea es que el modelo pre-entrenado actuará como un extractor de características. Las últimas capas del modelo pre-entrenado se sustituyen con un clasificador propio, dependiendo de cuál sea el problema a resolver. Se "congelan" los pesos de las capas del modelo pre-entrenado y se entrena el conjunto normalmente; por congelar (*freeze*) se entiende no cambiar el valor de los pesos durante el proceso de optimización del *gradient descent*.

Para entender cómo funciona, supongamos que tenemos un modelo ya entrenado con el conjunto de datos Imagenet[27] [49]. Cuando hablamos de las capas bajas de una red, explicábamos que detectará características de bajo nivel, como bordes y curvas. Excepto que nuestro problema sea muy específico, en nuestro problema también necesitaremos detectar esas características de bordes y curvas; en vez de entrenar todo un modelo con pesos iniciales aleatorios, podemos utilizar los pesos de un modelo previamente entrenado y congelarlos, poniendo foco en las capas más importantes, las de más arriba

[26] Fuente: https://www.oreilly.com
[27] Imagenet es un conjunto de muestras que contiene 14 millones de imágenes de 1.000 clases diferentes.

en la red, en el aprendizaje. Si nuestro conjunto de imágenes es muy diferente de las contenidas en Imagenet, podríamos entrenar mayor número de capas y congelar unas pocas capas de los niveles inferiores.

2.3.11 Técnicas de Data Augmentation

Finalmente, se exponen otro método para aliviar el problema de la necesidad de tener grandes cantidades de datos en el aprendizaje profundo. Cuando nuestra red toma una imagen como entrada, lo hace en forma de tensor N-dimensional. Si a esa imagen la desplazamos, píxel a píxel, una pequeña cantidad en una dimensión, el cambio visualmente será imperceptible para nosotros, pero para el ordenador supondrá un tensor completamente diferente al original. Estas técnicas artificiales que expanden el conjunto de imágenes de entrada produciendo otras con la misma etiqueta, pero con un tensor diferente se conocen como *Data Augmentation.* Existen muchas formas de expandir el conjunto de datos: además de los desplazamientos en dos dimensiones, es habitual las rotaciones, reflexiones frente a un eje, distorsiones, cambios en la saturación, el contraste o brillo, color *jitters*, recortes y muchas otras. Aplicando simplemente un par de estas técnicas podemos doblar o triplicar de forma efectiva el número de ejemplos de entrenamiento.

2.4 Modelo EfficientNet

En 2012, el modelo AlexNet [44] ganó la *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) derrotando al segundo clasificado por una diferencia cercana al 10% en la precisión de sus resultados. AlexNet utilizó en su modelo en torno a 62 millones de parámetros entrenables.

En seguida, muchos científicos descubrieron muchas mejoras en AlexNet que incrementan su eficiencia. GoogleNet [50], el modelo que ganó el ILSVRC de 2014, utiliza sólo 6,8 millones de parámetros a la vez que es significativamente más exacto que AlexNet. Después de que las ineficiencias iniciales fueran conocidas y arregladas, en los años siguientes las mejoras se lograron incrementando significativamente el número de parámetros de los modelos.

Arquitectura	Año	Precisión	Parámetros
AlexNet	2012	56,55%	62 M
GoogleNet	2014	74,8 %	6,8 M
SENet	2017	82,7 %	145 M
GPipe	2018	84,3 %	557 M

Tabla 2-1. Evolución de los modelos de Deep Learning ganadores de ILSVRC

Examinando la Tabla 2-1 se observa que los incrementos de exactitud traen consigo grandes incrementos en el número de parámetros y, por tanto, en el tamaño del modelo. Por ejemplo, hemos visto que GoogleNet tiene 6,8 millones de parámetros; el objetivo sería el diseñar de forma sistemática y eficiente un modelo de la mitad de tamaño, aunque sea menos exacto. Algunos de los métodos para lograrlo serían:

1. Compresión del modelo. Modificando el modelo original mediante:
    a. *Pruning*. Eliminación sistemática de los parámetros que no contribuyan a la exactitud del modelo
    b. Cuantización. Examinando la distribución de los parámetros en el espacio real y codificándolos en variables de 32 bits, con la menor pérdida de precisión posible.
2. Elaboración manual de modelos. Diseño de los modelos desde cero con el objetivo de minimizar su tamaño, como los modelos SqueezeNets, MobileNets y ShuffleNets [51].
3. Búsqueda de red. Se trata de una búsqueda automatizada de los parámetros óptimos de la red: profundidad, ancho y tamaño de los kernels de la convolución, como por ejemplo el modelo MnasNet [52].
4. Escalado de modelos. En este caso se utiliza un modelo estándar como GoogleNet o ResNet [53] y se escala hacia arriba (usando más parámetro) o hacia abajo, cambiando la profundidad o anchura de la red, o el tamaño de las imágenes de entrada.

Profundizamos en este último método, ya que es el utilizado por el modelo EfficientNet [31], utilizado en este trabajo.

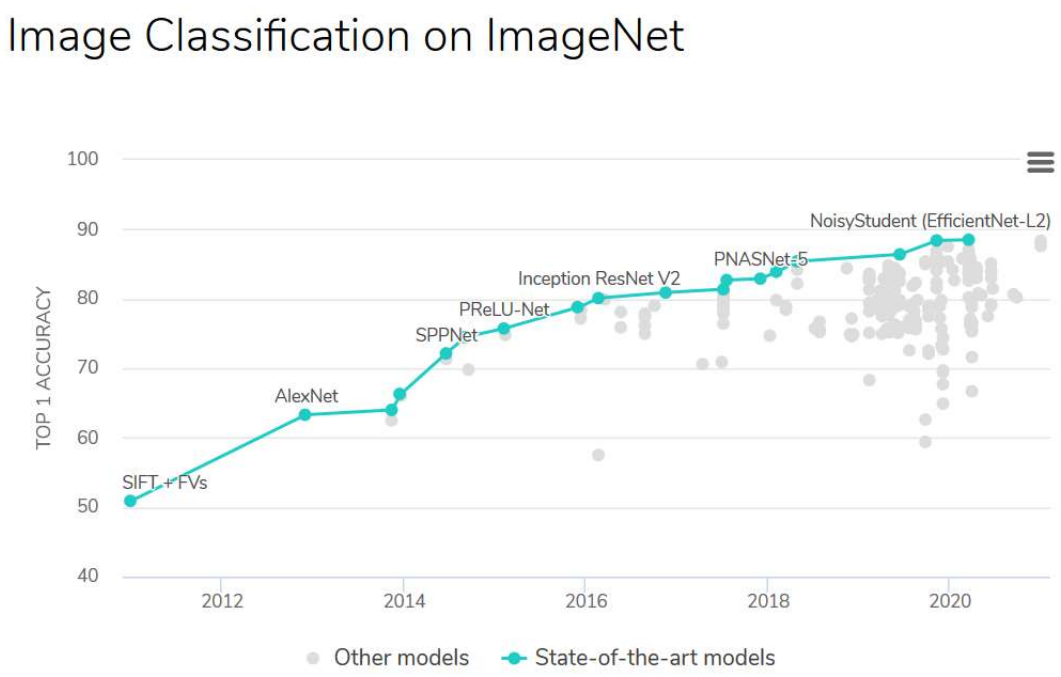


Figura 2-19. Comparativa de modelos en el estado del arte para Deep Learning[28]

2.4.1 Escalado del Modelo

La forma de escalar un modelo predefinido y modificarlo se basa en las siguientes técnicas:

- Cambiar la profundidad del modelo. Una CNN consiste en varias capas convolucionales. Estas capas aprenden a codificar diferentes niveles de abstracción de las imágenes de entrada. Una CNN con mayor número de capas (más profunda) retendrá más detalles y por tanto habitualmente será más exacta que un modelo con menor número de capas. Por ejemplo, ResNet-18 y ResNet-200 están ambas basadas en la arquitectura ReNET pero la segunda es mucho más profunda que la primera, y por tanto más precisa. Pero, por otro lado, ResNet-18 es más pequeña y ligera de ejecutar.

  Las redes muy profundas se enfrentan a un par de problemas:

  - Son más difíciles de entrenar debido al problema de *vanishing gradient*.
  - La ganancia en precisión satura a cierto nivel.

[28] Fuente: https://paperswithcode.com/

- Cambiar la anchura del modelo. Una capa CNN tiene también varios canales, como son los colores RGB de la imagen. Una red con más canales por capa se considera más ancha. Una red con más canales por capa se considera más ancha que una red con menos. WideRedNet y MobileNets no son muy profundas, pero sí anchas. Este tipo de redes son más fáciles de entrenar, pero también sufren los siguientes problemas:
    - Redes anchas y poco profundas tienen dificultades para capturar las características de alto nivel.
    - La ganancia en precisión satura a cierta anchura.
- Resolución de las imágenes de entrada. Las arquitecturas CNN toman como entrada imágenes de tamaño fijo. Con lo que uno puede cambiar la arquitectura para soportar imágenes de entrada más grandes (es obvio que una imagen de 512x512 tiene más información que una de 256x256). Este incremento en precisión requiere más capacidad de proceso porque una imagen de 512x512 tiene 4 veces más píxeles que una de 256x256. Igual que en los casos anteriores, la ganancia en precisión satura después de una determinada resolución.

Los autores del modelo realizaron las siguientes observaciones [31]:

- Escalar cualquier dimensión de la red en ancho, profundidad o resolución mejora la precisión, pero las ganancias son incrementalmente menores cuando las dimensiones van siendo progresivamente más grandes.
- Para conseguir ganancias óptimas, es crítico balancear todas las dimensiones en el proceso de escalado.

Las observaciones anteriores parecen obvias, pero no es tan obvio cuando se trata de elegir estos incrementos en cada una de las dimensiones. Buscar en una dimensión es muy costoso en tiempo de proceso; buscar en un espacio de tres dimensiones es prácticamente imposible.

La propuesta es realizar un escalado de componentes utilizando un coeficiente común, ϕ, para escalar de forma sistemática y uniforme los tres componentes, ancho, profundidad y resolución, mediante la siguiente fórmula:

$$\text{profundidad: } d = \alpha^{\phi}$$

$$\text{anchura: } w = \beta^{\phi}$$

$$\text{resolución: } r = \gamma^{\phi}$$

$$\text{con: } \begin{matrix} \alpha \cdot \beta^2 \cdot \gamma^2 \cong 2 \\ \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{matrix}$$

- ϕ es un coeficiente propio del modelo que controla los recursos (por ejemplo, las operaciones en punto flotante, o FLOPS), que es el parámetro principal de escalado del modelo.
- α, β, γ distribuyen la profundidad, ancho y resolución respectivamente.

Los FLOPS consumidos en una red convolucional son proporcionales a d, w^2 y r^2, lo que se refleja en la primera restricción (los autores restringen $\alpha \cdot \beta^2 \cdot \gamma^2$ a 2, e tal manera que para cada ϕ, los FLOPS requeridos se multiplican por 2^{ϕ}.

2.4.2 EfficientNet-B0

Las ecuaciones anteriores podrían sugerir que podemos escalar cualquier modelo CNN. Aunque es correcto, los autores han encontrado que la elección del modelo inicial influye en gran manera en los resultados finales que se obtengan. Por lo que desarrollaron su propia arquitectura básica llamada EfficientNet-B0. Como MnasNet, se entrenó con una arquitectura neuronal multi-objetivo, para optimizar tanto la precisión como FLOPS. Así, la arquitectura final (Tabla 2-2) es similar a MnasNet.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Tabla 2-2. Arquitectura de EfficientNet0[29]

2.4.3 Escalado del modelo EfficientNet-B0

Comenzando por el modelo EfficientNet-B0, los autores usaron la siguiente estrategia de escalado:

1. Se comienza con $\phi = 1$
2. Cada paso en el escalado supone duplicar la necesidad de recursos del paso anterior.
3. Se hace una búsqueda en *grid*, en el espacio de valores α, β y γ para comprobar que se cumple el paso anterior.
4. Los autores encontraron que con $\alpha = 1.2$, $\beta = 1.1$ y $\gamma = 1.15$ el modelo tenía un funcionamiento óptimo.
5. Fijadas α, β y γ como constante y escalando EfficientNet-B0 para diferentes valores de ϕ obtuvieron las nuevas rede escaladas EfficientNet-B0 a EfficientNet-B7.

La razón por la que no se reevalúan α, β y γ en cada paso es porque es muy costoso computacionalmente.

[29] Fuente: Tan 2019

2.4.4 Rendimiento EfficientNet

La Figura 2-20 muestra la curva de rendimiento de la familia EfficientNet, comparados con otros modelos populares en la literatura, cuando se enfrentan a la tarea de clasificación del conjunto de imágenes ImageNet.

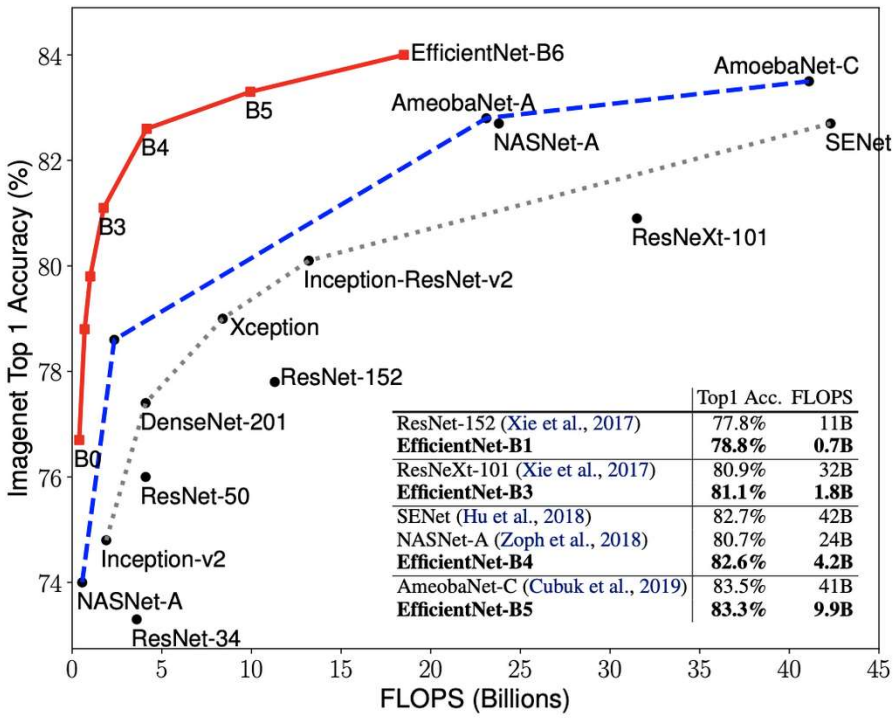


Figura 2-20. Comparativa de los modelos EfficientNet con otros[30]

Muestra que, para el mismo FLOPS, la precisión de EfficientNet es superior a cualquier otra arquitectura existente. Si estuviéramos planeando utilizar, por ejemplo, Inception-v2 [54], deberíamos considerar el uso alternativo de EfficientNet-B1. Del mismo modo, es preferible el uso de EfficientNet-B2 frente a ResNet-50.

En la mayoría de los casos prácticos, los científicos e investigadores comienzan con un modelo pre-entrenado y lo optimizan (sintonizan) para su aplicación específica. Cabría preguntarse si ya que EfficientNet supera a las otras arquitecturas con el *dataset* de ImageNet, entonces también lo superará en otras aplicaciones con conjuntos de datos de imágenes diferentes. Los autores, después de realizar diferentes experimentos, así lo

[30] Fuente: Tan 2019

afirman, con lo que parece que estas mejoras generalizan para otras tareas de visión artificial.

2.5 Optimización

Cada vez que una red neuronal termina un lote de procesamiento a través de la red, debe decidir como utiliza la diferencia de los resultados para ajustar los pesos de los nodos para que la red avance hacia la solución. El algoritmo que determina el paso que debe modificar los coeficientes se conoce como algoritmo de optimización.

Repasamos en este apartado los algoritmos de optimización más conocidos y disponibles en la librería Keras [55].

2.5.1 SGD

Stochastic Gradient Descent, es el algoritmo clásico de optimización: se computa el gradiente de la función de pérdidas de la red con respecto a cada peso de la misma; se calculan los gradientes que se crean para cada uno de los pesos multiplicados por una tasa de aprendizaje, para mover los pesos en la dirección que apuntan sus respectivos gradientes. Es el algoritmo más simple, tanto conceptualmente como por su comportamiento. Dada una tasa determinada de aprendizaje, SDG simplemente sigue el gradiente de la superficie de costes. Los nuevos pesos se generan en cada iteración que siempre serán estrictamente mejores que los de las iteraciones previas.

La simplicidad de SGD lo convierte en una buena opción para redes de pocas capas. Sin embargo, SGD converge significativamente más despacio que otros algoritmos más avanzados; además, en general no es capaz de escapar de los mínimos locales (“trampas”) que existen en la superficie de costes, por lo que SGD no se utiliza, o no es recomendado para su uso en redes neuronales profundas de múltiples capas.

2.5.2 SGD con momento

El momento Nesterov fue una de las primeras innovaciones que mejoró la velocidad de convergencia del algoritmo de optimización anterior. Las técnicas de momentos utilizan información de los pasos previos en la determinación del paso actual. Mientras que SGD sigue estrictamente la superficie de costes en la medida que la tasa de aprendizaje se lo

permite, el uso de momento Nesterov, permite tener cierta "inercia" en el movimiento en el sentido en seguir en una determinada dirección, incluso frente a cambios bruscos en la dirección del gradiente, hasta que se acumula cierta energía en la nueva dirección para hacer cambiar el sentido. Se asemeja al momento inercial conocido de la física.

El uso del momento tiene dos ventajas: por un lado, ayuda a evitar las trampas de mínimos locales; por otro lado, permite a los optimizadores aprender más deprisa, ya que soporta el uso de tasas de aprendizaje más elevadas. La forma más sencilla de aplicar momentos es, para cada iteración de aprendizaje, crear un vector cuyo valor medio sea una media decreciente de los valores anteriores, y sumar ese vector al gradiente actual para aplicar la corrección a los pesos.

El momento Nesterov es una ligera variación del anterior, que funciona mejor en la práctica, ya que evita la tendencia a sobrepasar los valles (*overshooting*). Ahora, en el paso predictor se extrapola linealmente la trayectoria actual, para que luego, en el punto predicho, se evalúa el gradiente y se hace la corrección de la trayectoria. Con ello se logra una aproximación de segundo orden de la trayectoria con un costo computacional similar al del momento normal. Se mantiene el efecto de inercia, pero se reducen los sobrepasos.

2.5.3 Adagrad

Adagrad es una técnica más avanzada que el SGD que realiza una *gradient escent* con una tasa variable de aprendizaje dependiendo de los nodos, es decir, se trata de programar la tasa de aprendizaje por nodo dentro del algoritmo, en vez de utilizar la misma tasa para todos los nodos. Se basa en la siguiente ecuación:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(G_t + \epsilon)}} \cdot g_t$$

Donde θ_t es el peso del nodo en el paso t; G_t es una matriz diagonal que contiene los cuadrados de todos los gradientes previos; ϵ es un infinitésimo del orden 10^{-8}, que se utiliza como término de regularización para prevenir las divisiones por 0; g_t es el vector de gradientes para el paso actual; y η es la tasa de aprendizaje. La clave del algoritmo se encuentra en el denominador que realiza la ponderación de los pesos; su inconveniente

es precisamente que la matriz G_t crece continuamente, por lo que la tasa de aprendizaje tiende a cero, impidiendo convergencias adicionales. La literatura recomienda comenzar con una tasa de aprendizaje de 0,01 que funciona bien en la mayoría de los casos.

2.5.4 Adadelta

Se trata de una adaptación del método Adagrad para impedir el problema del decrecimiento continuo de la tasa de aprendizaje. Adadelta utiliza una adaptación del gradiente en el que cada peso se pondera por la suma del gradiente actual y un promedio decreciente exponencialmente de un promedio de un limitado número de gradientes anterior. Como el denominador del gradiente no crece indefinidamente, el algoritmo es más robusto. Aunque Keras pide una tasa de aprendizaje en este modelo, técnicamente no lo necesita.

2.5.5 RMSprop

En este caso es una corrección de Adagrad, independiente de Adadelta; es similar a Adadelta, con una diferencia: la tase de aprendizaje se divide adicionalmente por un promedio que decae exponencialmente de todos los gradientes al cuadrado, es decir, un parámetro global.

2.5.6 Adam

El algoritmo de optimización *Adaptative Moment Estimation*, además utilizar el promedio de cuadrados anterior, también utiliza el promedio decreciente de los anteriores gradientes, similar al método de los momentos.

En este caso se estiman los momentos de primer y segundo orden del gradiente con las siguientes fórmulas:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_1 v_{t-1} + (1-\beta_2) g_t^2$$

Donde v_t es el promedio (exponencialmente decreciente) de los cuadrados de los gradientes anteriores y m_t es el promedio (exponencialmente decreciente) de los gradientes anteriores. β_1 y β_2 son las tasas de decaimiento.

El problema al usar estas fórmulas es que están sesgadas hacia 0. Para evitarlo, este método utiliza una corrección adicional:

$$\widehat{m_t} = \frac{m_t}{1+\beta_1}$$
$$\widehat{v_t} = \frac{v_t}{1-\beta_2}$$

Con lo que finalmente. La fórmula para la actualización de la tasa de aprendizaje es:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon} \cdot \widehat{m_t}$$

Podemos visualizar este modelo como el de una bola sobre la superficie de costes, con momento y fricción.

2.5.7 AdaMax

Adam, RMSProp y otros utilizan la varianza del gradiente en su formulación. El uso de la varianza, o de la norma L_2 del gradiente, estrictamente hablando es arbitrario, podríamos utilizar otros momentos definidos como:

$$L_1 = g$$
$$L_2 = \sqrt{g^2}$$
$$L_3 = \sqrt[3]{g^3}$$
$$L_n = \sqrt[n]{g^n}$$

siendo g el vector de gradientes. Las normas superiores a 2 no son muy útiles porque no son numéricamente estables. Sin embargo, sorprendentemente, la norma *infinita* sí que lo es, y se puede simplificar como:

$$v_t = \max(\beta_2 \cdot v_{t-1}, |g_t|)$$

AdaMax es un algoritmo de actualización de gradiente más robusto que Adam, y tiene una estabilidad numérica superior a Adam.

2.5.8 Nadam

Nadam es Adam, pero con el momento Nesterov, en vez de un momento ordinario, con los mismos beneficios que se obtienen que en el caso de SGD.

2.5.9 AMSgrad

Es una mejora reciente del método Adam. En algunos casos Adam no es capaz de converger hacia una solución óptima, como hace SGD. Parece ser que para algunos *datasets* (de imágenes principalmente), Adam tiende a bajar la prioridad de los gradientes más informativos, frente a otros gradientes espurios menos informativos, haciendo que el algoritmo se pase el punto óptimo sin explorarlo suficientemente.

AMSgrad propone resolver este problema calculando el valor de $\widehat{v_t}$, pero tomando como promedio en la fórmula:

$$v_t = \max(\widehat{v_t}, v_{t-1})$$

Al tomar ese máximo, la influencia de los gradientes más grandes se conserva. Este algoritmo se comporta sólo ocasionalmente mejor que Adam, pero esas ganancias no son generalizables y no ha conseguido desplazarle.

3 Trabajos relacionados

Existen numerosas actividades de investigación en la literatura[31] relacionadas con los métodos de optimización, tanto con métodos de clasificación como de agrupamientos (*clustering*). Siguiendo a Pootschi et al. [56] casi todos los métodos de clasificación se han empleado para el diagnóstico de malaria con muestras finas de frotis de sangre (*thin blood smear*), desde los métodos no supervisados de *clustering* K-Mean [57] hasta otras técnicas no supervisadas como el árbol Naïve Bayes [58], Ada-boost [59], Decision Tree [60], Support Vector Machine [61] o el Discriminante Lineal [62].

Tabla 3-1. Métodos de clasificación utilizados en el diagnóstico de malaria[32]

Tipo de frotis		**Metodología de Clasificación**
Fino	No supervisado	K-mean Clustering
		Quaternion Fourier Transform (QFT)
	Supervisado	Thresholding
		Bayesian classifier
		Annular ring ratio method
		Naïve bayes Tree
		Logistic Regression Tree
		Linear Programming
		Euclidian Distance Classifier
		Decision Tree
		Template Matching
		Ada-boost
		Nearest Mean Classifier (NM)
		Fuzzy Interface System
		Normalized cross-correlation
		Support Vector Machine (SVM)
		Linear Discriminant (LD)
		Crowd Source Games
		Neural Network
		Deep Learning
Grueso	No supervisado	K-Mean Clustering
	Supervisado	Naïve Bayes Tree
		Randomized Tree Classifier
		Nearest Mean Classifier (NM)
		Thresholding
		Support Vector Machine (SVM)
		Neural Network
		Geneetic Algorithm

[31] Para la búsqueda de los artículos referenciados en el texto no se ha utilizado ninguna herramienta concreta (Google Scholar, Science Direct…) sino búsquedas directas en Internet.
[32] Fuente: Poostchi - 2018

Sin embargo, la mayoría de ellos están dedicados al estudio de la interacción entre la clasificación con las características de las imágenes o la segmentación de las mismas, y muy pocas de ellas están dedicadas explícitamente a la investigación de la detección del parásito en las imágenes de los glóbulos rojos.

Resulta difícil comparar el rendimiento de todos esos estudios. Por un lado, principalmente porque la mayoría de los estudios no utilizan un dataset de imágenes común, ni siquiera un número de muestras comparable. Sin embargo, si se puede observar una contraposición entre el tiempo de proceso y la exactitud: cuanto mayor es el tiempo de proceso, mejor es la exactitud conseguida. Además, la arquitectura del algoritmo empleado tiene mucha influencia en la ejecución del proceso, por ejemplo, la clasificación mediante redes neuronales en más lenta que la que utiliza Support Vector Machine.

Los algoritmos de aprendizaje profundo se han utilizado recientemente para aumentar el rendimiento en varias áreas relacionadas con la medicina. Liang et al. [20] fueron unos de los primeros investigadores que aplicaron CNN al diagnóstico de la malaria. Utilizaron un modelo clásico de transferencia de aprendizaje un modelo propio de CNN para clasificar automáticamente imágenes de células sanguíneas *thin smear*, obtenidas mediante muestras de microscopio óptico clásico. Sus resultados mostraron que el modelo propio CNN obtenía un mejor rendimiento (97,37% exactitud) que los métodos más tradicionales de transferencia de aprendizaje.

Rajaraman et al. [21] evaluaron diversas CNN pre-entrenadas basadas en algoritmos de aprendizaje profundo como un extractor de características previo a la clasificación. Estudiaron hasta 6 arquitecturas diferentes (AlexNet, VGG16, ResNet50, Xception, DenseNet121 y un modelo propio) obteniendo unos resultados de exactitud que variaron desde el 91,50% hasta 95,9%. Los métodos VGG16 y ResNet demostraron ser superiores al resto de sus competidores en esta tarea. Concluyeron que las CNNs pre-entrenadas son una herramienta prometedora para la extracción de características.

Rahman et al. [63] usaron el dataset del National Institute of Health, el mismo que el utilizado en el presente trabajo, y un esquema de validación cruzada de 5 iteraciones (5-*fold cross-validation scheme*), para medir sus modelos: un modelo propio de CNN

(exactitud 96,29%), un modelo basado en VGG16 (exactitud 97,77%) y un extractor de características CNN aplicado a un modelo SVM (exactitud 94,77%). Observaron que el uso de algunas de las técnicas más habituales de pre-procesado, como la normalización o la estandarización, no tenían un impacto significativo en el rendimiento final. Sin embargo, los procedimientos de *data augmentation* aplicados a las imágenes de entrenamiento revelaron resultados muy esperanzadores.

Shah et al. [64] desarrollaron un algoritmo de clasificación de imágenes basados en CNNs y lo probaron con un dataset de imágenes previamente etiquetadas. Obtuvieron un valor de exactitud final del 94,77%. En el artículo manifiestan que tuvieron limitaciones relacionadas con los recursos de computación, y sugirieron que los resultados obtenidos pueden mejorar con una mayor potencia de cálculo.

Quan et al. [65] trabajaron en un modelo nuevo y ligero basado en CNNs, combinando conceptos de redes densas y redes basadas en residuos y empleando mecanismos de atención. Titularon el método propuesto como *Attentive Dense Circular Net* (ACDN). Los resultados obtenidos se compararon con los obtenidos mediante el empleo de otros métodos del estado del arte de la técnica disponibles en la literatura como DenseNet121 [66] o DPN92 [67]. Los resultados mostraron un rendimiento superior con un valor de exactitud de 97,47% frente a unos valores de 90,94% y 87,88% obtenidos mediante DenseNet121 y DPN92, respectivamente. Además, el modelo propuesto ACDN presentó mayor precisión y una rápida velocidad de convergencia.

Finalmente, Yang et al. [68] implementaron un modelo interesante, que utilizaba imágenes sanguíneas de frotis grueso, y se desarrolló específicamente para ser ejecutado en teléfonos móviles. Su modelo implicaba un primer paso de revisión rápida de las imágenes para detectar las zonas de las imágenes que podías ser candidatos para contener el parásito. Por un lado, clasificaron las imágenes disponibles públicamente por la comunidad científica con un modelo propio CNNs y obtuvieron una exactitud del 97,6%. Por otro, compararon su modelo con otros disponibles en la literatura, aplicando de igual modo el pre-procesado citado. Su método obtuvo unos valores de exactitud mejores que los de otras arquitecturas como ResNet50 (exactitud 93.88%), VGG19 (exactitud: 93.72%) y AlexNet (exactitud: 96.33%).

4 Métodos y materiales

En este apartado se presentan los métodos y materiales concretos utilizados en este trabajo. La sección 4.1 describe los datasets de las imágenes utilizados para entrenar y testar el método propuesto. Los algoritmos CNN se presentan en la sección 4.3. Por último, la sección 3 introduce el setup experimental y el método de validación del experimento.

4.1 Origen de los datos

La colección de imágenes de células sanas y parasitadas utilizadas en este trabajo se hicieron públicas por Rajaraman et al. [21]. Estas imágenes fueron recogidas de pacientes infectados por *P. falciparum*y fotografiadas con la cámara fotográfica de un smatphone. Provenían de un conjunto de microfotografías de células rojas tintadas con el procedimiento Giemsa, de un conjunto de 150 pacientes infectados y 50 pacientes sanos del Hospital Chittagong Medical College [69]. Estas imágenes fueron anotadas/clasificadas de forma manual por personal médico experto de la Unidad Mahidol Oxford Tropical Medicine Research en Bangkok [21] y archivadas. El Institutional Review Board (IRB) de la National Library of Medicine (NLM) (National Institutes of Health (NIH)) aprobó la realización del estudio en sus dependencias. Están disponibles para su descarga en la propia página Web de la NLH[33]; desde la Web de Kaggle[34] también está disponible el mismo conjunto de datos.

El conjunto de imágenes consiste en 27.558 imágenes, en formato JPEG, con igual número de unidades para células sanas y parasitadas; las imágenes parasitadas consisten en glóbulos rojos infectados por *Plasmodium*, las sanas a veces incluyen algún tipo de ruido, como interferencias del tintado o polvo. Las imágenes se encuentran comprimidas para su descarga en un fichero “cell_images.zip”. Las células que contienen *Plasmodium* están etiquetadas como positivas mientras que las muestras normales se encuentran sanas (negativo) aunque puedan incluir impurezas o algún tipo de manchas. Las imágenes están ajustadas a un tamaño uniforme de 100x100 píxeles y media

[33] https://lhncbc.nlm.nih.gov/publication/pub9932
[34] https://www.kaggle.com/

normalizada, para conseguir una convergencia del modelo más rápida. Un ejemplo de cómo la identidad del paciente está codificada en el nombre del fichero: por ejemplo, en la imagen "C33P1thinF_IMG_20150619_114756a_cell_179.png"'P1' indica la identidad del paciente.

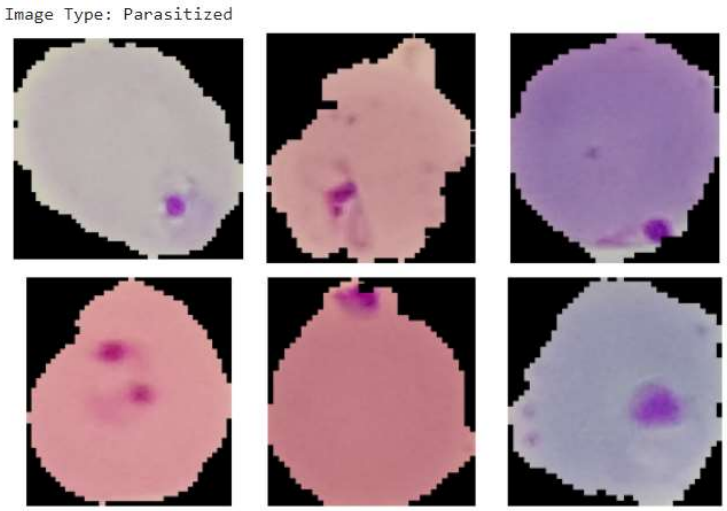


Figura 4-1. Ejemplo de células positivas, infectadas, del conjunto de datos

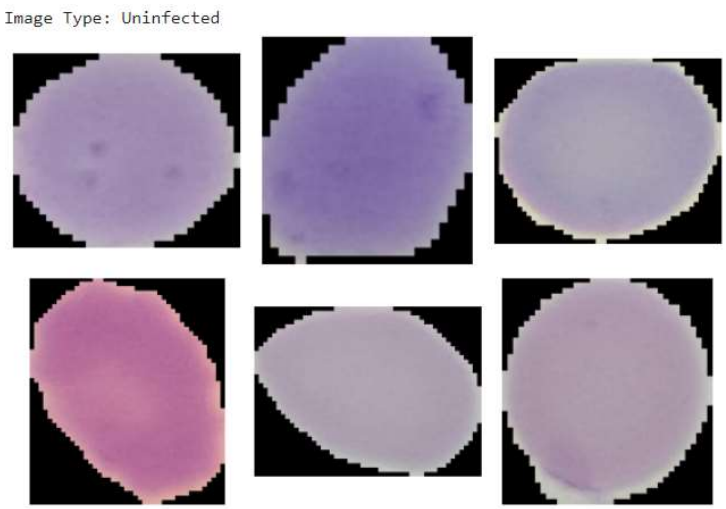


Figura 4-2. Ejemplo de células negativas, sanas, del conjunto de datos

4.2 Malaria Dataset

Es importante garantizar contra con un mismo número de muestras que cubran las dos clases propuestas, normal y malaria, para validar de forma correcta el rendimiento del experimento. Por tanto, se han utilizado el mismo número de imágenes de ambos tipos

para el entrenamiento y el test. En la Tabla 4-1 se muestran en número de imágenes utilizadas en el trabajo. En total, 22.046 muestras se han usado para cada una de las clases, MALARIA y NORMAL. Esas imágenes se han utilizado en el proceso de validación cruzada de 10 iteraciones (10-FOLD).

Además, el algoritmo se ha validado utilizando un dataset separado. Este dataset contiene 5.512 imágenes y está simétricamente distribuido con 2.756 imágenes para la clase NORMAL y 2.756 para la clase MALARIA. Este último dataset no se empleó ni en la fase de entrenamiento ni en la fase de test. Este proceso se utilizó para comprobar que no hubo *overfitting* en el proceso de entrenamiento.

Tabla 4-1. Información del Dataset.

Tipo	Imágenes Train/Test	Imágenes validación
NORMAL	11.023	2.756
MALARIA	11.023	2.756

4.3 CNN propuesta

Se ha utilizado el modelo EfficientNetB0, con un proceso previo de transferencia de aprendizaje. Se ha añadido una capa `global_average_pooling2d` para reducir el *overfitting* en base a reducir el número de parámetros. Sobre ese modelo, se han añadido en secuencia tres capas internas densas (*dense layers*) junto con capas *dropout* y funciones de activación ReLu. Para evitar *overfitting*, se ha utilizado una tasa de *dropout* del 30%. Finalmente, se ha añadido una capa final densa (*dense layer*) junto con una unidad de salida para realizar la clasificación binaria; en este caso, la función de activación es una `softmax` que proporciona el sistema final automatizado de decisión. El orden de las capas, el número de parámetros entrenables y no-entrenables (pesos) en cada capa, las dimensiones de salida de cada capa se pueden ver en la Tabla 4-2. El número de parámetros del modelo final es de 4.223.934.

Tabla 4-2. Capas, dimensiones y parámetros del modelo

Capa (Tipo)	Output shape	Parámetros #
EfficientNetB0 (Model)	7 x 7 x 1280	4.049.564
global_average_pooling2d	1280	0
dense (Dense)	128	163.968
dropout (Dropout)	128	0
dense_1 (Dense)	64	8.256
dropout_1 (Dropout)	64	0
Dense_2 (Dense)	32	2.080
Dropout_2	32	0
Dense_3 (Dense)	2	66
Parámetros totales:	4.223.934	
Parámetros entrenables:	4.181.918	
Parámetros no-entrenables:	42.016	

Todas las librerías y software empleados en este trabajo son *open source*. Para replicar los resultados en futuros trabajos, todo el software realizado es compatible con la plataforma Google Colaboratory, seleccionando el entorno de ejecución GPU. Esta plataforma se puede utilizar sin coste ya que Google la ofrece para propósitos de investigación; el hardware sobre el que corre es una potente Tesla K80 GPU de 12 GB. La arquitectura EfficientNet es un tipo de red neural convolucional (CNN) escalada y previamente entrenada, que se utiliza habitualmente para problemas de clasificación de imágenes por medio de transferencia de aprendizaje. Este modelo se desarrolló por el departamento Google AI en mayo de May 2019 y está preparado para su uso desde la plataforma Github [70]. De igual forma, Google AI también ha desarrollado la librería `Albumentations` y también está disponible desde los repositorios de Github [71]. En resumen, todo el software se puede utilizar sin problemas de licencia ya que no tiene coste y es *open source*. El método propuesto también fue utilizado con éxito por otro equipo del departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática para el diagnóstico de COVID-19 con imágenes de Rayos-X [72].

El modelo utiliza tres librerías diferentes; la librería `EfficientNet` como el módulo base, la librería `ImageDataAugmentator` y la librería `Albumentations`.

En primer lugar, se construye el algoritmo EfficientNet con técnicas de escalado simples pero efectivas. Este método permite escalar desde una red neuronal convolucional de línea base hasta cualquier requisito que impongan las fuentes de imágenes, pero manteniendo el rendimiento del modelo, adquirido previamente mediante el método de transferencia de aprendizaje a partir de datasets de imágenes externos. En general, las redes EfficientNet permiten alcanzar una mayor exactitud y una eficiencia superior sobre otras CNNs, como pueden ser MobileNetV2, GoogleNet, AlexNet,e ImageNet [31]. EfficientNet posiblemente se utilizará como el nuevo paradigma base para las futuras tareas de visión artificial. EfficientNet comprende ocho modelos diferentes, etiquetados de B0 a B7, oscilando el número de parámetros de cada modelo desde 4 millones para el modelo B0 hasta más de 60 millones para el modelo B7. La configuración de este trabajo ha implementado el modelo EfficientNetB0 que utiliza 4.049.564 parámetros, que se ha considerado adecuado dados los recursos disponibles y el objetivo de nuestro trabajo.

En segundo lugar, el software `Albumentations` es utilizado ampliamente en ingeniería, en investigación de aprendizaje profundo, en concursos de inteligencia artificial y en desarrollo *open source*. La librería proporciona diferentes métodos para transformación de imágenes y está muy optimizada para la ejecución. Este software también incluye una interfase para tares de visión artificial, incluyendo detección de objetos, segmentación y clasificación. En este trabajo se ha utilizado la función `Compose` del paquete `Albumentations`. Se ha comprobado que el uso de `Albumentations` disminuye el *overfitting*, mejora el rendimiento de los clasificadores y reduce el tiempo de ejecución [73]. Cuando se implementa esta librería para la *augmentation* de imágenes para cada una de las iteraciones, la exactitud del modelo se incrementa, y el tiempo de ejecución disminuye.

Por fin, la librería `ImageDataAugmentator` es una librería *image data generator* para Keras, con una interfase Python para su uso con redes neuronales artificiales, soportando la utilización de librerías avanzadas de *augmentation* (como por ejemplo

`imgaug` y `albumentations`) [74]. Esta librería se configura de forma compatible con la librería `albumentations` para reducir el tiempo de ejecución. En el código, se implementa un *data generator* mediante el método constructor de la clase `ImageDataAugmentator`. Se necesitan dos parámetros, el primero es `rescale`, para el que se ha utilizado un valor de 1/255, utilizado para convertir cada píxel desde su valor en un rango comprendido entre [0, 255] para escalarlo a [0, 1]. El Segundo es el parámetro `augment`, el cual está diseñado para usarse como la salida del método `compose` del software `albumentation`. Este *data generator* se ha usado para pre-procesar todas las imágenes del dataset. La Figura 4-3. muestra el diagrama de bloques del algoritmo.

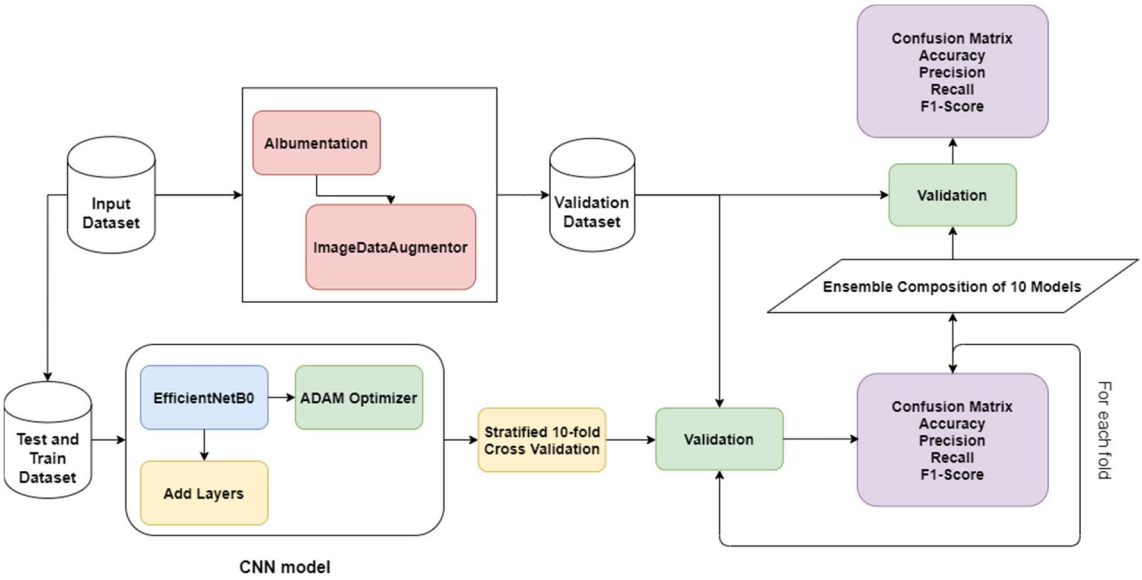


Figura 4-3. Diagrama de bloques del algoritmo

4.4 Configuración experimental y de validación

El programa se ejecutó sobre un ordenador portátil DELL XPS 15 9560, Intel Core i7, 16GB RAM, equipado con una tarjeta gráfica NVIDIA GeForce GTX 1050 que permitió los cálculos en punto fijo, por el soporte a cálculos GPU. La fase entrenamiento del modelo EfficientNetB0 adaptado se realizó llevando la técnica de validación cruzada KFOLD con 10 iteraciones. En conjunto, cada iteración se compuso de un máximo de 33 epochs. De otra forma, cada iteración consistió en 1.240 pasos- El valor elegido como mini-batch fue de 16.

La validación del modelo se ha realizado en dos fases. En primer lugar, se ha utilizado una técnica de validación cruzada de 10 iteraciones para entrenar y testear el modelo; y, en segundo lugar, un dataset diferente, con imágenes que no han sido previamente utilizadas en la fase anterior se ha utilizado para validar el rendimiento del modelo. Después se ha calculado la matriz de confusión. A continuación, se han calculado los valores de exactitud, precisión, *recall*, y F1-score para cada una de las clases. Por fin, los valores promedios de las iteraciones se han calculado. La Tabla 4-3 muestra la configuración experimental práctica utilizada en el experimento.

Tabla 4-3: Configuración Experimental

Entorno	1. *Selección del entorno Google Colab o local e instalación de las librerías necesarias.*
Input	2. *Descarga de las imágenes para las dos categorías.*
Configuración	3. *Importación de las imágenes.*
Configuración de Directorios	4. *Creación de dos directorios de imágenes con etiquetas acordes a cada una de las clases.*
	5. *Configuración del entrenamiento, testing y validación utilizando una validación cruzada de 10 iteraciones.*
Muestra de imágenes aleatorias	6. *Muestra algunas imágenes de cada tipo.*
Configuración del Data Generator	7. *Definición del flujo de augmentation utilizando la función Compose de la librería albumentation.*
	8. *Creación del data generator como un objeto de la librería ImageDataAugmentator y configuración del flujo augmentation obtenido en (7).*
Entrenamiento y Testing	9. *Creación del modelo utilizando EfficientNetB0 y dense layers con una función de activación ReLu y una capa de salida con una función de activación softmax.*
Validación cruzada de 10 iteraciones	10. *Compilación del modelo utilizando el optimizador ADAM con una tasa de aprendizaje de 0.0001 y una función Categorical_Crossentropy para el cálculo de las pérdidas.*
	11. *Ajuste del modelo utilizando 33 epochs y una función ReduceLROnPlateau para reducir la tasa de aprendizaje cuando las métricas no mejoren.*
	12. *Guardado del modelo para usarse en el proceso de validación y creación del modelo ensemble.*
	13. *Configuración del dataset para testing.*
	14. *Cálculo de los parámetros de rendimiento en cada iteración.* a. *Gráfico de la evolución de las pérdidas del modelo.* b. *Gráfico de la evolución de la exactitud del modelo.* c. *Informe del test de clasificación.* d. *Validación del modelo.* e. *Curva AUC-ROC.* f. *Matriz de confusión.* g. *Informe de la validación de la clasificación.*
Modelo Ensembled	15. *Cálculo de los parámetros de rendimiento del modelo ensemble.* a. *Informe de la clasificación ensemble.* b. *Curva AUC-ROC.* c. *Matriz de confusión.*
Estudio de errores	16. *Localización las imágenes mal clasificadas.*
	17. *Dibujo de algunas imágenes de cada tipo.*

5 Resultados

El entrenamiento del modelo requirió un total de 409.200 iteraciones de imágenes individuales. El tiempo de ejecución para la fase de entrenamiento del modelo fue de 1 día y 23:38:44. La tase de aprendizaje inicial se estableció en 0,0001. El modelo utilizó la técnica del *callback* `ReduceLROnPlateau` ya que se precisa una reducción de la tasa de aprendizaje tan pronto como el modelo deja de mejorar de manera significativa; con una tasa de aprendizaje menor estos modelos buscan de forma más efectiva el óptimo. Esta función *callback* comprueba la tasa de mejora, y si no se confirma un progreso significativo durante un número= 'PATIENCE' de epochs, se reduce la tasa de aprendizaje. Se estableció el valor de PATIENCE a 6, y una tasa de aprendizaje mínima de `min_lr`=0.000001 antes de la finalización definitiva del programa- Se eligió la técnica de optimización de ADAM [75] como el método para el cálculo del valor de pérdida en el mecanismo de *backward* propagation. En las figuras y tablas que siguen se da información detallada de los flujos de datos en el programa dentro de cada iteración, así como los valores obtenidos, como los valores de exactitud, precisión, *recall* o área bajo la curva de la gráfica ROC. Una vez que cada uno de los 10 modelos del proceso KFOLD se han entrenado, los modelos se utilizan en los test de validación, usando siempre un dataset separado de imágenes que no ha sido empleado para el entrenamiento. el rendimiento obtenido en las simulaciones es muy esperanzador. En los anexos se da información detallada, incluyendo el script Python utilizado. Estos anexos también incluyen información detallada de los resultados obtenidos.

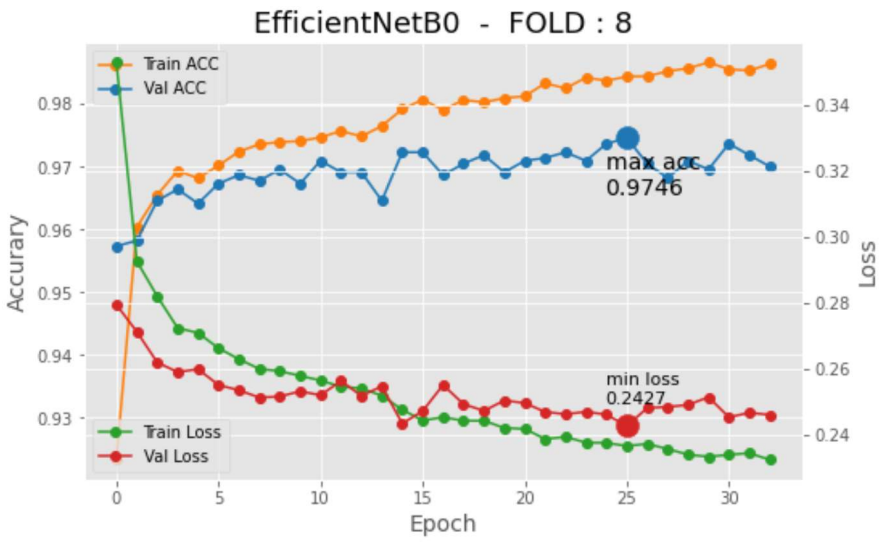


Figura 5-1. Evolución de la exactitud y de las pérdidas en el aprendizaje

5.1 Resultados experimentales de la clasificación binaria

En total, se utilizaron 22.046 imágenes diferentes para el entrenamiento y 5.512 imágenes para el testing. Se ofrecen los resultados obtenidos para cada una de las 10 iteraciones, junto con el valor promedio de todos ellos. Se han calculado los valores de exactitud, *recall*, precisión y F1-score tanto para cada clase por separado, como para ambas clases de manera conjunta.

La Tabla 5-1 muestra los resultados de la clasificación relativos a la clase malaria. El rendimiento mínimo de la clase malaria se obtiene para las iteraciones 0, 1 y 4. El valor de precisión mínimo de 97,72% en la iteración 1. Además, el valor mínimo de *recall* fue de 96,19% en la iteración 8. El valor mínimo de F1-score fue de 97,02% también en la iteración. Los valores promedio de la precisión, *recall*, y F1-score fueron de 98,07%, 97,05% y 97,55%, respectivamente.

Tabla 5-1. Resultados de la clasificación binaria para la clase malaria.

Iteración	*Precisión*	*Recall*	*F1-score*
0	0,977231	0,971920	0,974569
1	0,978221	0,976449	0,977335
2	0,981702	0,971920	0,976787
3	0,978102	0,971014	0,974545
4	0,978042	0,968297	0,973145
5	0,984259	0,963735	0,973889
6	0,979909	0,972801	0,976342
7	0,985441	0,981868	0,983651
8	0,978782	0,961922	0,970279
9	0,985185	0,964642	0,974805
Promedio	*0,980687*	*0,970457*	*0,975535*

Los resultados de la ejecución del modelo respecto de la clasificación para la clase normal se muestran en la Tabla 5-2 El valor mínimo de *recall* es 97,72% obtenido en la iteración 0. De igual forma, el valor más bajo para la precisión es de 96,25% para la iteración 9. Por último, la iteración 4, presenta un valor mínimo de F1-score de 97,33%. Los valores promedios para la precisión, *recall* y F1-score son 97,07%, 98,08% y 97,66%, respectivamente.

Tabla 5-2. Resultados de la clasificación binaria para la clase normal.

Iteración	*Precisión*	*Recall*	*F1-score*
0	0,971996	0,977293	0,974638
1	0,976428	0,978202	0,977314
2	0,972122	0,981835	0,976954
3	0,971145	0,978202	0,974661
4	0,968525	0,978202	0,973339
5	0,964444	0,984574	0,974405
6	0,972949	0,980018	0,976471
7	0,981900	0,985468	0,983681
8	0,962500	0,979110	0,979734
9	0,965302	0,985468	0,975281
Promedio	*0,970731*	*0,980837*	*0,976648*

Los valores para la exactitud, precisión, *recall* y F1-score para la clasificación de ambas clases, consideradas de forma conjunta se presentan en la Tabla 5-3. La exactitud promedio obtenida es de 97,56%. Además, se obtuvieron unos valores promedio para la precisión de 97,57%, para el *recall* de 97,56% y para el F1-score de 97,56%.

Tabla 5-3. Resultados de la clasificación binaria para ambas clases.

Iteración	*Exactitud*	*Precisión*	*Recall*	*F1-score*
0	0,974603	0,974617	0,974603	0,974603
1	0,977324	0,977326	0,977324	0,977324
2	0,976871	0,976919	0,976871	0,976870
3	0,974603	0,974628	0,974603	0,974603
4	0,973243	0,973290	0,973243	0,973242
5	0,974150	0,974356	0,974150	0,974147
6	0,976407	0,976432	0,976407	0,976406
7	0,983666	0,983672	0,983666	0,983666
8	0,970508	0,970649	0,970508	0,970506
9	0,975045	0,975253	0,975045	0,975043
Promedio	*0,975642*	*0,975714*	*0,975642*	*0,975641*

5.2 Experimental validation results of classification

En esta fase, el objetivo es el de validar el rendimiento del algoritmo y verificar que no se ha producido *overfitting*. Para ello, en el trabajo se ha utilizado un *dataset* de imágenes que no se había utilizado en la fase anterior. Este *dataset* externo incluye 2.756 imágenes de la clase malaria y 2.756 de la clase normal.

La fase de validación se realizó para cada uno de los 10 modelos obtenidos en la fase anterior de validación cruzada KFOLD. Estos resultados de validación se muestran en la Tabla 5-4, donde también se muestra el parámetro de "Área bajo la Curva" (ROC). El valor medio de la exactitud es de 97,70%. Además, los valores promedio para la precisión, el *recall* y el F1-score son 97.70%, 97,69% y 97,69% respectivamente. Se puede apreciar que la varianza de los valores entre los experimentos es mínima, tanto para la exactitud como para el ROC ($\cong 4 \cdot 10^{-6}$, $1{,}5 \cdot 10^{-7}$ para el ROC) y, por tanto, este trabajo demuestra la validez del método propuesto.

Tabla 5-4. Resultados del proceso de validación de la clasificación

Iteración	Exactitud	Precisión	*Recall*	F1-score	ROC
0	0,977142	0,977143	0,977141	0,977141	0,996143
1	0,978788	0,978801	0,978774	0,978773	0,997121
2	0,974084	0,974292	0,973875	0,973868	0,996094
3	0,978254	0,978278	0,978229	0,978228	0,996904
4	0,979563	0,979626	0,979499	0,979497	0,996379
5	0,973411	0,973491	0,973331	0,973328	0,996098
6	0,977894	0,977922	0,977866	0,977865	0,996798
7	0,978270	0,978310	0,978229	0,978228	0,996274
8	0,976499	0,976583	0,976415	0,976412	0,996893
9	0,975631	0,975754	0,975508	0,975504	0,996243
Promedio	0,976953	0,977020	0,976887	0,976884	0,996495

Con respecto a la exactitud, el modelo 4 demostró ser el mejor entre el conjunto de los 10. En la Figura 5-2 se muestra la *Receiver Operating Characteristics* (ROC) del modelo de la iteración 5 y la Figura 5-3 presenta la matriz de confusión para el conjunto de imágenes de validación.

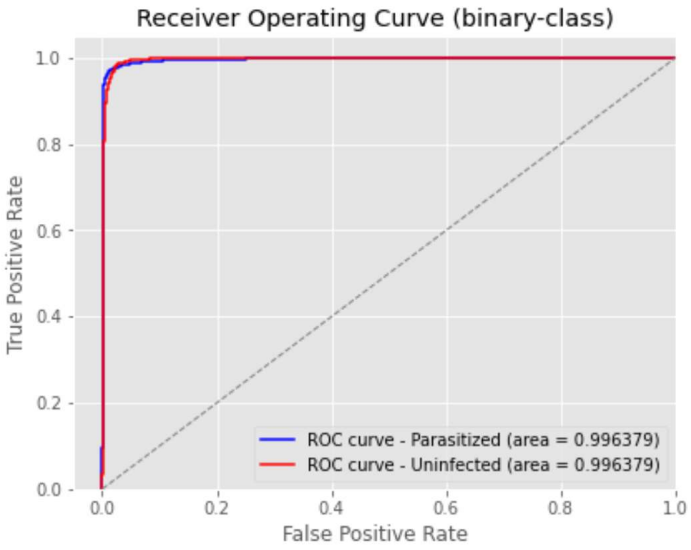


Figura 5-2. ROC para la iteración número 4

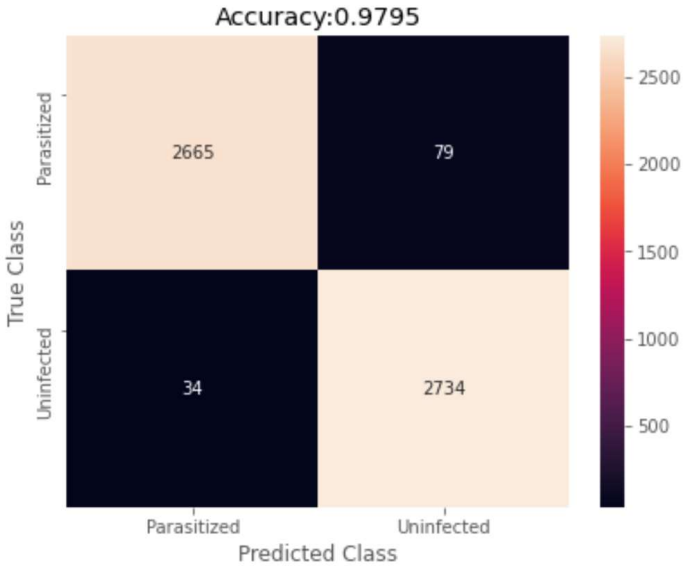


Figura 5-3. Matriz de confusión para la iteración 4

5.3 Resultados experimentales del modelo *ensemble*

El paso final de mejora del modelo consistió en la elaboración de los resultados conjuntos (*ensembled*). El modo habitual de realizar un *ensembled* modelo es el de utilizar diferentes modelos teóricos. En este trabajo, el modelo *ensembled* se formó con cada uno de los 10 modelos previamente entrenados. Hay que resaltar dos puntos:

- Los diez modelos son realmente el mismo, EfficientNet0, pero entrenados con *datasets* ligeramente diferentes (una diferencia de imágenes del 20 % entre entrenamientos)
- No se realiza ningún entrenamiento adicional en esta fase.

A pesar de utilizar el mismo modelo, y con una coincidencia del 80% en las imágenes entre los distintos entrenamientos, los resultados obtenidos han sido realmente buenos. La exactitud subió hasta 98,29% desde un valor promedio del 97,70%. Este valor representa una disminución de la tase de error de 25,75% error de los valores del ensemble respecto de los valores promedio de los modelos individuales.

La Figura 5-4 presenta la matriz de confusión[35] para este experimento final de este trabajo. Una matriz de confusión contiene información sobre los resultados de

[35] http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html

clasificación reales y predichos hechos por un sistema de clasificación [76]. En las cuatro subáreas de la gráfica se muestran los *True Positives* (TP: 2.682 muestras positivas correctamente clasificadas), *False Positives* (FP: 62 muestras negativas erróneamente clasificadas como positivas), *True Negatives* (TN: 2.736 muestras negativas correctamente clasificadas) y *False Negatives* (FN: 32). A partir de los datos de esta tabla es fácil calcular los valores de exactitud, recall, precisión o F1-score. En nuestro ejemplo, la tasa de falsos positivos casi dobla la tasa de falsos negativos, lo que parece indicar que todavía hay espacio para lograr mejoras adicionales en futuros trabajos.

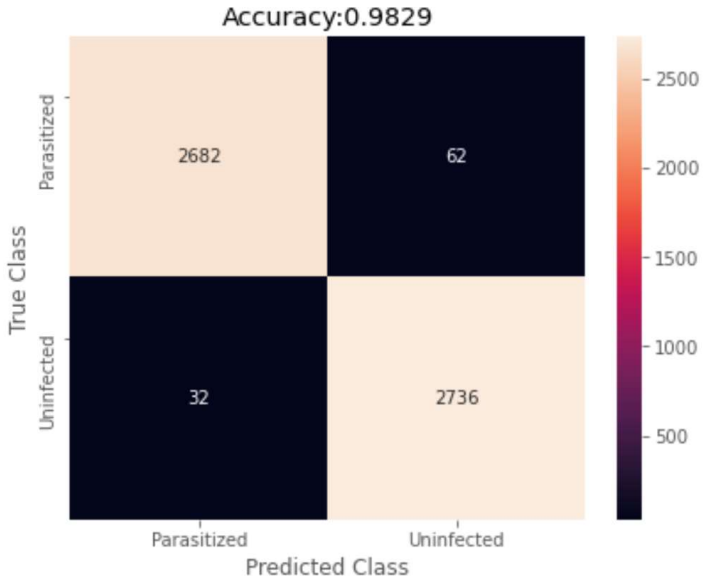


Figura 5-4. Matriz de confusión del modelo final (ensembled)

Finalmente, la Figura 5-5 muestra la curva *Receiver Operating Curve* de modelo *ensemble*, para el *dataset* de validación. La Curva ROC es otra forma alternativa de mostrar el rendimiento de un clasificador [77]. Una curva ROC es una gráfica con la tasa de falsos positivos en el eje X y la tasa de positivos correctos en el eje Y. El punto $(0,1)$ representa el clasificador perfecto. Es $(0,1)$ porque la tasa de falsos positivos es 0 (ninguno), y la tasa de positivos correctos es 1 (todos). El punto $(0,0)$ representa un clasificador que predice todos los casos negativos, mientras que el punto $(1,1)$ se corresponde con un clasificador que predice cada caso como positivo. En muchos casos, el clasificador tiene un parámetro (umbral) que se puede ajustar para incrementar la TP (*True Positives*) a costa de incrementar los falsos positivos, o viceversa. Cada valor del umbral resulta en un par de valores (FP, TP), y el conjunto de todos ellos permite dibujar

la curva ROC. En la literatura es común aceptar el área bajo la curva ROC como un parámetro mejor que la exactitud para comparar la bondad de los clasificadores, especialmente cuando las distribuciones de probabilidad de las muestras no están balanceadas. En nuestro ejemplo el valor del área bajo la curva (o AUC) es del 99,76%, muy próximo a su valor máximo del clasificador perfecto de 1,0.

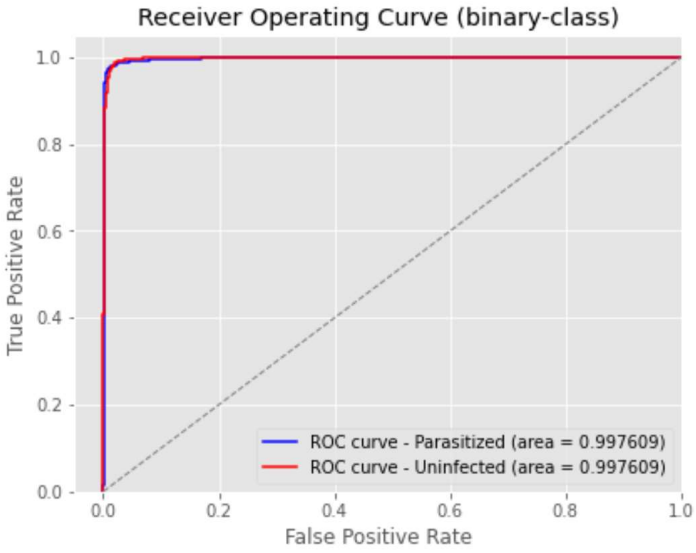


Figura 5-5. ROC del resultado final (ensembled)

6 Discusión

Finalmente, se ha comparado el modelo con trabajos similares que se pueden encontrar en las revistas especializadas del tema. No obstante, se debe señalar que la comparación debe entenderse limitada debido a las diferencias en los conjuntos de muestras empleadas, además de las diferencias lógicas en los parámetros utilizados en la configuración de los diferentes modelos de aprendizaje automático, como tamaño de entrada o filtros de entrada aplicados. Además, la mayoría de los artículos no proporcionan el software ni los datos, por lo que no es viable evaluar en profundidad las otras técnicas de la comparación con los mismos datos de este trabajo o viceversa.

Se pueden encontrar un buen número de publicaciones referidos a métodos de CNN aplicados a la diagnosis de la malaria, y ese número se incrementa de día en día [56]. Por el momento, el foco de estas técnicas se sitúa en la ayuda a los profesionales de la salud. Los diferentes autores proponen una diversidad de arquitecturas diferentes de redes neuronales convolucionales para la diagnosis automática de la malaria, como ResNet, DenseNet, DPN92 o ADCN [65]. En este apartado se compara el modelo con los estudios más similares que se han podido encontrar en la literatura y ofrecen unos mejores resultados; por otro lado, todos ellos utilizan modelos que están en el estado del arte de la técnica. La Tabla 6-1 presenta los resultados de la comparativa, con hasta otros 13 modelos diferentes. Por "*own CNN*", se refiere a un modelo basado en red neuronal de diseño propio.

Tabla 6-1. Comparativa con otros modelos para el diagnóstico de la malaria

Referencia	Arquitectura	Exactitud	*Recall*	Precisión	F1-Score
Liang [20]	Transfer Learning	0,9199	0,8900	0,9512	0,9024
Liang [20]	Own CNN	0,9737	0,9775	0,9699	0,9736
Rajaraman [21]	Own CNN	0,9400	0,9310	0,9512	0,9410
Rajaraman [21]	ResNet50	0,9570	0,9450	0,9690	0,9570
Rahman [63]	Own CNN	0,9629	0,9234	0,9804	0,9495
Rahman [63]	VGG16	0,9777	0,9720	0,9719	0,9709
Shah [64]	Own CNN	0,9477	0,9526	0,9437	0,9481
Quan [65] [66]	DenseNet121	0,9094	0,9251	0,8960	0,9103
Quan [65] [67]	DPN92	0,8788	0,8681	0,8892	0,8785
Quan [65]	ADCN	0,9747	0,9520	0,9350	0,9434
Yang [68]	VGG19	0,9372	0,8731	0,5299	0,6595
Yang [68]	AlexNet	0,9633	0,8215	0,7023	0,7573
Yang [68]	Own CNN	0,9726	0,8273	0,7898	0,8081
Este PFG	*EfficientNet0*	*0,9829*	*0,9882*	*0,9774*	*0,9828*

En la tabla, se han elegido como criterios de comparación: la exactitud, *recall*, precisión y F1-score. Como se puede ver en la tabla, el método empleado en este trabajo tiene mejor comportamiento que todos ellos.

Los resultados que se muestran en la tabla parecen indicar que los modelos CNN más profundos (por el número de capas que emplean) y mejor entrenados que se pueden encontrar en la literatura, parecen no funcionar de manera óptima en este tipo de tareas. Esas redes tienen habitualmente tienen unas arquitecturas “pesadas” con millones de parámetros a entrenar, por lo que las arquitecturas CNN más simples, con una fracción más pequeña del número de parámetros a entrenar, se adaptan mejor a estos problemas donde las imágenes son menos complejas que las naturales y tienen menor varianza. Estos resultados indican que no siempre el incremento de complejidad trae consigo un mejor comportamiento. Como se ha indicado, el resultado se puede

explicar considerando las diferencias entre las imágenes médicas y las imágenes naturales, que provoca que la transferencia de aprendizaje entre las redes previamente entrenadas no sea efectiva.

Se puede comprobar que los valores de exactitud de la Tabla 6-1 varían desde 87,88% a 97,47%. El método de este trabajo es superior a todos ellos con un valor de 98,29%. Por tanto, el uso de los modelos EfficientNet muestran resultados esperanzadores para el diagnóstico automatizado de malaria. En este trabajo, se ha empleado una validación *k-fold* estratificada de 10 iteraciones para evaluar el modelo, por lo que los resultados son altamente confiables. Hemos visto en la Tabla 5-4 los resultados de la exactitud, precisión, *recall* y F1-score de los modelos de cada iteración para el proceso de clasificación binaria utilizando el conjunto de datos separado de validación. Después, se ha creado un modelo *ensemble*, con cada uno de los diferentes modelos obtenidos en cada una de las 10 iteraciones del proceso de validación; este último modelo es mejor que todos y cada uno de los modelos que lo componen. Los resultados muestran una exactitud de 98,29%, un *recall* de 98,82%, una precisión de 97,74% y un F1-score de 98,28%, lo cual mejora de forma significativa los resultados individuales y promedio, que se muestran en la Tabla 5-4. El área bajo la curva es de 99,76% y la curva ROC del modelo final se muestra en la Figura 5-5. Finalmente, la matriz de confusión se muestra en la Figura 5-4.

Hasta donde se ha podido investigar, no existe un trabajo similar en la literatura para la realización de diagnósticos de malaria, que incluya una serie de características como las siguientes:

1. El modelo se basa en EfficientNet y utiliza un modelo pre-entrenado para la transferencia de aprendizaje.
2. Se ha empleado para validar el modelo una validación cruzada de 10 iteraciones (KFOLD). Esta técnica utiliza conjuntos diferentes de imágenes para el entrenamiento y para el testeo. Se puede demostrar que se reduce el sesgo de los resultados, y cada una de las imágenes disponible se utiliza 9 veces en el proceso de entrenamiento y 1 sola vez en la fase de test.

3. El método incluye una validación con un conjunto de imágenes diferente, no utilizado previamente en la fase de *training/testing*. El proceso de validación ha empleado 5.512 imágenes no utilizadas previamente. No se ha detectado la presencia de *overfitting* n los resultados.
4. El modelo final es un modelo ensamblado (*ensembled*) realizado a partir de los 10 modelos, todos ellos con la misma arquitectura deEfficientNet0, entrenados en la fase de la validación 10-Fold.
5. Se ha empleado la librería `albumentation` para disminuir el *overfitting*, mejorar el proceso de transferencia de aprendizaje, expandir el volumen del *dataset* de imágenes disponible y disminuir el tiempo de ejecución.
6. Todo el código Fuente se ha hecho accesible a la comunidad científica con la previsible publicación del artículo asociado a este trabajo.

El modelo usa la técnica de optimización ADAM. A su vez, ADAM utiliza otras técnicas como AdaGrad y RMSProp. Casi todos los artículos con experimentos similares al de este trabajo utilizan el algoritmo de optimización ADAM. Además, ADAM es hoy en día el algoritmo de optimización recomendado ya que generalmente proporciona mejores resultados que RMSProp. No obstante, suele merecer la pena probar con el modelo SGDNesterov Momentum como alternativa. Una posible continuación a este trabajo podría ser la integración del optimizador ADAM con un Análisis de Componentes Principales (PCA) y un Mapa Auto-Organizativo (*Self-Organization Map*) para mejorar el rendimiento del modelo [78].

Una nota final a resaltar es que la calidad de las imágenes parece influir de manera profunda en la exactitud de los resultados- La Figura 6-1 muestra tres células infectadas que, no obstante, han sido categorizadas como normales. La razón de este comportamiento podría ser que los síntomas no son todavía obvios, o que el patógeno no ha sido adecuadamente tintado.

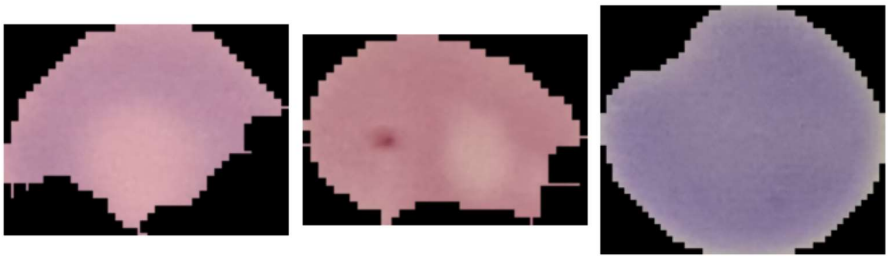


Figura 6-1. Imágenes de células con malaria clasificadas como normales.

Por otro lado, la Figura 6-2 muestra las imágenes de tres células normales que se han clasificado erróneamente como infectadas. Esto ha podido ser debido a impurezas o artefactos en forma de manchas, las cuales tienen una apariencia muy similar al patógeno tintado.

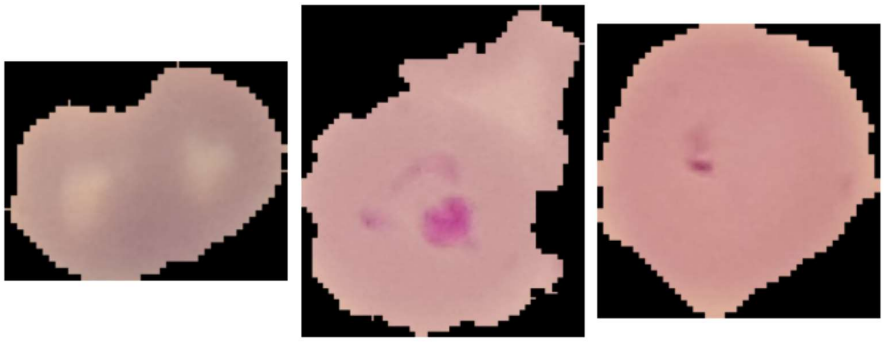


Figura 6-2. Imágenes de células normales clasificadas como con malaria.

En resumen, este trabajo ha puesto de manifiesto los resultados prometedores del algoritmo EfficientNet como herramienta para el apoyo del diagnóstico de la malaria. Adicionalmente, en este trabajo se propone el uso de las librerías `Albumentation` y `ImageDataAugmentator`. El trabajo apoya el actual cuerpo de conocimiento científico, ya que da una respuesta efectiva al diagnóstico automatizado de la malaria, en el entendimiento de que este tipo de algoritmos no pretende sustituir el trabajo de los expertos médicos; en su lugar, estos métodos intenta apoyarlos y reducir su tiempo de contacto con la malaria.

7 Conclusión

En este proyecto fin de grado se ha trabajado en un sistema automatizado, basado en un algoritmo de redes neuronales convolucionales, para la asistencia a la identificación de malaria en los glóbulos rojos de los pacientes. El método propuesto hace uso del modelo de EfficientNet y se ha verificado haciendo uso de la validación cruzada de 10 iteraciones (10-fold). Además, se ha utilizado un conjunto de datos independientes para validar. Los resultados finales se han obtenido por medio de un modelo *ensembled* compuesto de los 10 modelos previamente calculados en el proceso de validación cruzada. Se han obtenido muy buenos resultados. con una exactitud de 98,29%, un *recall* de 98,82%, una precisión de 97,74 % y un F1-score de 98,28% en el proceso de clasificación binaria de las células. El área bajo la curva (AUC) es de 99,76%. El autor de este trabajo no ha encontrado un estudio similar en la literatura con que utilice el método de EfficientNet como una técnica automática para detectar malaria.

Se ha constatado muchas de las limitaciones comunes a todos los métodos de aprendizaje automático. A pesar de la inmensa cantidad de individuos infectados con malaria, las colecciones de imágenes accesibles para investigación no son lo suficientemente robustas. Sin embargo, por la experiencia que ya se tiene con las CNN es de esperar que el rendimiento de estas técnicas mejorará con el aumento de la cantidad de imágenes disponibles para usarlas como entrenamiento. Además, es importante estudiar detenidamente el comportamiento de esta clase de técnica tomando en consideración como parámetro la fase de la enfermedad en que se encuentre el paciente. Es posible que los algoritmos sean capaces de detectar la enfermedad en una fase avanzada de la misma, pero es incluso más importante enfocarse en las etapas tempranas de la enfermedad donde estas técnicas son más útiles y, sin embargo, presentan un rendimiento menor.

Todo el código fuente utilizado en este proyecto, se ha puesto a disposición de la comunidad científica como un documento adjunto a una publicación enviada para su aprobación. Al compartir el software y los datos, la comunidad científica puede reproducir los resultados y, de esta forma, se apoya el desarrollo de futuras actividades

de investigación. Por supuesto, de esta forma se permite que los investigadores comprueben, actualicen, revisen y/o modifiquen diferentes parámetros para mejorar los resultados.

8 Referencias

[1] WHO, World Malaria Report 2020, Geneva: World Health Organization, 2020, p. 299.

[2] H. Carballo y K. King, «Emergency department management of mosquito borne illness: malaria, dengue and west nile virus,» *Emergency medicine practice, ,* vol. 16, nº 5, pp. 1-23, 2014.

[3] J. Muñoz, G. Rojo y G. Ramírez, «Diagnosis and treatment of imported malaria in Spain: recommendations from the malaria Working group of the Spanish Society of tropical medicine and international health (SEMTSI),» *Enfermedades Infecciosas y Microbiología Clínica,* nº 33, pp. 1-13, 2015.

[4] WHO, Malaria microscopy quality assurance manual , version 2, Ginebra: World Health Organization, 2016.

[5] WHO, Guidelines for the Treatment of Malaria, 3 ed., Geneva: World Health Organization, 2015.

[6] K. Makhija, S. Maloney y R. Norton, «The utility of serial blood film testing for the diagnosis of malaria,» *Pathology,* vol. 47, nº 1, pp. 68-70, 2015.

[7] K. Mitiku, G. Mengitsu y B. Gelaw, «The reliability of blood film examination for malaria at the peripheral health unit,» *Ethiopian Journal of Health Development,* vol. 17, nº 3, pp. 149-246, 2003.

[8] T. Tokumasu, R. Fairhurst y G. Ostera, «Band 3 modifications in Plasmodium falciparum-infected AA and CC erythrocycles assayed by autocorrelation analysis using quantum dots,» *Journal of Cell Science,* vol. 118, nº 5, pp. 1091-1098, 2005.

[9] D. Das, M. Gosh, M. Pal, A. Maiti y C. Chakraborty, «Machine learning approach for automated screening malaria parasite using light microscopic images,» *Micron,* nº 45, pp. 97-106, 2013.

[10] C. Dong, C. Loy y X. Tang, «Accelerating the super-resolution convolutional neural network,» de *Compuer Vision - ECC*, Cham, Springer International Publishing, 2016, pp. 391-407.

[11] S. Lawrence y C. C. T. A. Giles, «Face recognition: a convolutional neurl-network aproach,» *IEEE Transactions on Neural Networks,* vol. 8, nº 1, pp. 98-113, 1997.

[12] D. Das, S. Koley, S. Bose, A. Maiti, B. Mitra, G. Mukherjee y P. Dutta, «Computer aided tool for automatic detection and delineation of nucleus from oral histopathology images for OSCC screening,» vol. 8, 2019.

[13] X. Ji, Q. Yu, Y. Liu y S. Kong, «A recognition method for Italian alphabet gestures based on convolutional neural network,» de *Intelligent Computing Theories and Application*, Springer International Publishinf, 2019, pp. 663-664.

[14] A. Tavanei, M. Ghodrati, S. Kheradpiesheh, T. Masquelier y A. Maida, «Deep learning in spiking neural networks,» *Neural Networks,* pp. 47-63, 2019.

[15] R. Karthik, M. Hariharan, S. Ananda, P. Mathikshara, A. Johnson y R. Menaka, «Attention embedded residual CNN for disease detection in tomato leaves,» *Applied Soft Computing,* vol. 86, nº 105933, 2020.

[16] Y. Wang, X. Wei, H. Shen, L. Ding y J. Wan, «Robust fusion for RGB-D tracking using CNN features,» *Applied Soft Computing,* vol. 92, nº 106302, 2020.

[17] J. Rangel, J. Martínez-Gómez, C. Romero-González, J. García-Varea y M. Cazorla, «Semi-supervised 3D object recognition through CNN labeling,» *Applied Soft Computing,* vol. 65, pp. 603-613, 2018.

[18] C. Wang, Z. Zhao, Y. Xu y Y. Yu, «A novel multi-focus image fusion by combining simplified very deep convolutional networks and patch-based sequential reconstruction strategy,» *Applied Soft Computing,* vol. 91, nº 106253, 2020.

[19] Y. Dong, Z. Jiang, H. Shen, D. Pan, L. Williams, V. Reddy, B. W. y A. Bryan, «Evaluations of deep convolutional neural networks for authomatic identification of malaria infected cells,» de *IEEE EMBS international conference on biomedical and health informatics*, Piscataway, 2017.

[20] Z. Liang, A. Powell, I. Ersoy, M. Poostchi, K. Silamu, K. Palaniappan, P. Guo, M. Hossain, A. Sameer, R. Maude y e. al., «CNN-Based Image Analysis for Malaria Diagnosis,» de *International Conference on bioinformatics and biomedicine*, Shenzhen, 2016.

[21] S. Rajaraman, S. Antani, M. Pootschi, K. Silamut y M. Hossain, «Pre-trained convolutional networks as feature extractors toward improved malaria parasite detection in thin blood smear images,» *PeerJ,* vol. 6, nº 4, p. 4578, 2018.

[22] O. Sagi y L. Rochard, «Ensemble learning: A survey,» *Wiley Interdisciplinary Reviews Data Mining and Knowledge Discovery,* vol. 8, nº 4, 2018.

[23] T. Dietterich, «Ensemble Methods in Machine Learning,» de *Proceedings of the First International Workshop on Multiple Classifier Systems*, Cagliari, 2000.

[24] G. Brown, «Ensemble Learning,» de *Encyclopedia of Machine Learning*, Boston, MA: Springer, 2011.

[25] A. Sarwar, M. Ali, M. Jatinder y V. Sharma, «Diagnosis of diabetes type-II using hybrid machine learning based ensemble model,» *International Journal of Information Technology,* vol. 12, pp. 419-428, 2020.

[26] D. Yadav y S. Pal, «To Generate an Ensemble Model for Women Thyroid Prediction Using Data Mining Techniques,» *Asian Pacific Journal of Cancer Prevention,* vol. 20, nº 4, pp. 1275-1281, 2019.

[27] M. Kaur, A. Malhi y H. Pannu, «Machine learning ensemble for neurological disorders,» *Neural Computing and Applications,* vol. 32, nº 8, pp. 12697-12714, 2020.

[28] Y. Xiao, J. Wu, Z. Lin y X. Zhao, «A deep learning-based multi-model ensemble method for cancer prediction,» *Computer Methods and Programs in Biomedicine,* vol. 153, nº 1, pp. 1-9, 2018.

[29] S. Rajaraman, S. Antani, M. Poostchi, K. Silamut, M. Hossain, R. Maude, S. Jaeger y G. Thoma, «Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images,» *PeerJ,* 2018.

[30] L. Duong, P. Nguyen, C. Di Sipio y D. Di Ruscio, «Automated fruit recognition using efficientnet and mixnet,» *Artículos académicos para comput. electron. agric.,* vol. 171, 2020.

[31] M. Tan y Q. Le, «Efficcientnet: Rethinking model scaling for convolutional neural networks,» de *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, California, 2019.

[32] M. Tan, «Efficientnet: Improving accuracy and efficiency through automl and model scaling,» 29 05 2019. [En línea]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Último acceso: 24 01 2021].

[33] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» *arXiv 1409.1556,* vol. 9, 2015.

[34] Y. Tai, J. Yang y L. X., «Image Super-Resolution via Deep Recursive Residual Network,» de *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[35] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab y H. Radha, «Deep learning algorithm for autonomous driving using GoogLeNet,» de *IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, 2017.

[36] Y. LeCun, L. Bottou, B. Y. y P. Haffner, «Gradient-Based Learning Applied to Document Recognition,» *Proceedings of the IEEE,* vol. 86, nº 11, pp. 278-2324, Noviembre 1998.

[37] J. Patterson y A. Gibson, Deep Learning. A Practitioner's Approach, Sebastopol: O'Reilly Media, Inc., 2016.

[38] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain,» *Psychological Review,* vol. 65, nº 6, p. 386, 1958.

[39] N. Buduma y N. Lacascio, Fundamentals of Deep Learning, Sebastopol:, 2017., Sebastopol: O'Reilly Media, Inc., 2017.

[40] R. G. D. Restak, The Secret Life of the Brain, Washington, D.C.: Joseph Henry Press, 2001.

[41] W. McCulloch y W. W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» *The Bulletin of Mathematical Biophysics,* vol. 5, nº 4, pp. 115-118, 1943.

[42] N. Binod y G. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines,» de *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, Haifa, 2020.

[43] M. Nielsen, «How the backpropagation algorithm works,» de *Neural Networks and Deep Learning*, New York, Determination Press, 2015.

[44] A. Krizhevsky, G. Sutskever y G. Hinton, «ImageNet classification with deep convolutional neural networks,» *Communications ACM,* vol. 60, nº 6, pp. 84-90, 2017.

[45] H. Shulz y S. Behnke, Deep Learning, Berlin: Springer, 2012.

[46] M. Zeiler y R. Fergus, «Visualizing and Understanding Convolutional Neural Networks,» de *European Conference on Computer Vision,*, Zurich, 2014.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, S. I. y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *Journal of Machine Learning Research,* nº 14, pp. 1929-1958, 2014.

[48] R. Shanmugamani, Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras, Birmingham: Birmingham: Packt, 2018.

[49] J. Deng, W. Dong, R. Socher, L. L. y L. Fei-Fei, «Image Net: a large-scale hierarchical image database,» de *IEEE conference on computer vision and pattern recognition*, Piscataway, 2009.

[50] C. Szegedy, L. Wei y J. Yangqing, «Going Deeper with Convolutions,» 17 09 2014. [En línea]. Available: https://arxiv.org/abs/1409.4842. [Último acceso: 2020 10 12].

[51] K. Okan, K. Meslihan, G. Ahmet y R. Gerhard, «Resource Efficient 3D Convolutional Neural Networks,» 4 4 2019. [En línea]. Available: https://arxiv.org/abs/1904.02422. [Último acceso: 12 10 2020].

[52] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler y A. A. Howard, «MnasNet: Platform-Aware Neural Architecture Search for Mobile,» 31 7 2018. [En línea]. Available: https://arxiv.org/abs/1904.02422. [Último acceso: 12 10 2020].

[53] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» 10 12 2015. [En línea]. Available: https://arxiv.org/abs/1512.03385. [Último acceso: 12 10 2020].

[54] C. Szegedy, S. Ioffe, V. Vanhoucke y A. Alemi, «Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,» 23 2 2016. [En línea]. Available: https://arxiv.org/abs/1602.07261v2. [Último acceso: 12 10 2020].

[55] A. Gulli y S. Pal, Deep learning with Keras, Packt Publishing Ltd., 2017.

[56] M. Poostchi, K. Silamut, R. Maude, S. Jaeger y G. Thoma, «Image analysis and machine learning for detecting malaria,» *Translational Research,* vol. 194, nº 4, pp. 36-55, 2018.

[57] Y. Purwar, S. Shah, G. Clarke, A. Almugairi y A. Muehlenbachs, «Automated and unsupervised detection of malarial parasites in microscopic images,» *Malaria Journal,* vol. 10, nº 364, 2011.

[58] D. Das, C. Chakraborty, B. Mitra, A. Maiti y A. Ray, «Quantitative microscopy approach for shape-based erythrocytes characterization in anaemia,» *Journal of Microscopy,* vol. 249, nº 2, pp. 136-149, 2013.

[59] J. Vink, M. Laubscher, R. Vlutters, K. Silamut, R. Maude, M. Hasan y G. Haan, «An automatic vision-based malaria diagnosis system,» *Journal of Microscopy,* vol. 250, nº 3, pp. 166-178, 2013.

[60] S. Sio, W. Sun, S. Kumar, W. Bin, S. Tan, S. Ong, H. Kikuchi, Y. Oshima y K. Tan, «MalariaCount: an image analysis-based program for the accurate determination of parasitemia,» *Journal of Microbiological Methods,* vol. 68, nº 1, pp. 11-18, 2007.

[61] S. Savkare y S. Narote, «Automated system for malaria parasite identification,» de *015 International Conference on Communication, Information & Computing Technology (ICCICT)*, Mumbai (India), 2015.

[62] L. Malihi, K. Ansari-Asl y A. Behbahani, «Malaria parasite detection in giemsa-stained blood cell images,» de *2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP)*, Zanjan (Iran), 2013.

[63] A. Rahman, H. Zunair, M. Y. J. Rahman, S. Biswas, A. Alam, N. Alam y M. Mahdy, «Improving malaria parasite detection from red blood cell using deep convolutional neural networks,» 23 07 2019. [En línea]. Available: https://arxiv.org/abs/1907.10418. [Último acceso: 26 2 2021].

[64] D. Shah, K. Kawale, M. Shah, S. Randive y R. Mapari, «Malaria Parasite Detection Using Deep Learning (Beneficial to humankind),» de *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2020.

[65] Q. Quan, J. Wang y L. Liu, «An Effective Convolutional Neural Network for Classifying Red Blood Cells in Malaria Diseases,» *Interdisciplinary Sciences: Computational Life Sciences,* vol. 12, pp. 217-225, May 2020.

[66] G. Huang, Z. Liu, L. Maaten y K. Weinberger, «Densely Connected Convolutional Networks,» 25 8 2016. [En línea]. Available: https://arxiv.org/abs/1608.06993?source=post_page. [Último acceso: 26 02 2021].

[67] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan y J. Feng, «Dual Path Networks,» 6 07 2017. [En línea]. Available: https://arxiv.org/abs/1707.01629. [Último acceso: 26 02 2021].

[68] F. Yang, M. Poostchi, H. Yu, Z. Zhou, K. Silamut, Y. J., M. R.J., S. Jaeger y S. Antani, «Deep Learning for Smartphone-based Malaria Parasite Detection in Thick Blood Smears,» *IEEE J Biomed Health Inform,* vol. 24, nº 5, pp. 1247-1438, May 2020 .

[69] I. Ersoy, F. Bunyak, J. M. Higgins y K. Palaniappan, «Coupled edge profile active contours for red blood cell flow analysis,» de *9th IEEE International Symposium on Biomedical Imaging (ISBI)*, Barcelona, 2012.

[70] P. Yakubovskiy, «Implementation of EfficientNet model. Keras and TensorFlow Keras,» [En línea]. Available: https://github.com/qubvel/efficientnet. [Último acceso: 30 01 2021].

[71] A. Parinov, «albumentations,» albumentations-team, [En línea]. Available: https://github.com/albumentations-team/albumentations. [Último acceso: 30 01 2021].

[72] G. Marques, D. Agarwal y I. de la Torre, «Automated medical diagnosis of COVID-19 through EfficientNet convolutional neural network,» *Applied Soft Computing Journal,* vol. 96, nº 106691, 2020.

[73] A. Buslaev, V. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin y A. Kalinin, «Albumentations: Fast and Flexible Image Augmentations,» *Information,* vol. 11, nº 2, p. 125, 2020.

[74] M. Tukiainen, «ImageDataAugmentor,» [En línea]. Available: https://github.com/mjkvaak/ImageDataAugmentor. [Último acceso: 18 1 2021].

[75] I. Mohd Jais, A. Ismail y N. S.Q., «Adam Optimization Algorithm for Wide and Deep Neural Network,» *Knowledge Engineering and Data Science (KEDS),* vol. 2, nº 1, pp. 41-46, 2019.

[76] R. Kohavi y F. Provost, «Glossary of terms,» de *Editorial for the Special Issue on Applications of Machine*, Discovery Oress, 1998.

[77] J. Swets, «Measuring the Accuracy of Diagnostic Systems,» *Science, New Series,* vol. 240, nº 4857, pp. 1285-1293, 1988.

[78] N. Ali, G. Sarowar, L. Rahman, J. D. N. Chaki y T. J.M., «Adam Deep Learning With SOM for Human Sentiment Classification,» *International Journal of Ambient Computing and Intelligence (IJACI),* vol. 10, nº 3, pp. 92-116, 2019.

9 Anexo: Script PYTHON utilizado en las simulaciones

Incluye información detallada de los resultados de la ejecución.

Este script se puede ejecutar directamente en la plataforma Google Colab[36], disponible de forma gratuita.

[36] https://colab.research.google.com/notebooks/intro.ipynb#recent=true

efn0_malaria_binary

January 30, 2021

1 Deep Learning for Automated Medical Diagnosis of Malaria

Gonçalo Marques, Antonio Ferreras, Isabel de la Torre

Department of Signal Theory and Communications, and Telematics Engineering University of Valladolid, Paseo de Belén, 15, 47011, Valladolid, Spain

1.1 Environment

1.1.1 Install required libraries

- **twine**: Utility for publishing Python packages on PyPI.
- **scikit-learn**: e Various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means
- **ImageDataAugmentor**: Custom image data generator for Keras supporting the use of modern augmentation modules
- **efficientnet**: Convolutional Neural Network model

[ ]:
```
!pip install -q twine
!pip install -U -q scikit-learn
!pip install -q git+https://github.com/mjkvaak/ImageDataAugmentor
!pip install -q efficientnet
```

```
  Building wheel for ImageDataAugmentor (setup.py) … done
```

1.1.2 Parameters

Set of parameters to carry out different simmulations, with different models and different sizes of image sets

[ ]:
```
QUICK_AND_DIRTY = False  # False: Full simmulation.
                         # True: Fast simulation for cheking the code

# To differenciate the running environment: Google Colab or Local machine
import os
COLAB = ('COLAB_GPU' in os.environ)  # True if we are in a Colab environment

DEVICE = "GPU"               # TPU NOT WORKING AT THE MOMENT

# Origin of Zip file
```

```
FILENAME = 'cell-images-for-detecting-malaria.zip'

if COLAB:
    DIRNAME = '/content/drive/My Drive/Colab Notebooks/MALARIA/'
else:
    DIRNAME = './'

# Where to decompress the content of the ZIP file
ORIDIR = "cell_images/"

# Local Directory for classified images (train / test / validate)
DESTDIR = "data/"

# Aleatory parameter to reproduce the experiments
SEED = 1234

if QUICK_AND_DIRTY:
    PERC = 5        # Percentage of images to deal with (5, 25, 50 ...)
    KFOLD = 2       # Numbers of folds in the k-fold process
    EPOCHS = 3      # Number of EPOCHS to run in each fold
    VERBOSE = 1     # How much the functions talk about its problems
    PATIENCE = 3    # Parameter for ReduceLROnPlateau()
else:
    PERC = 0        # Complete set 22,046 images
    KFOLD = 10      # 10-kfold method
    EPOCHS = 33     # long run
    VERBOSE = 0     # Plea, be quiet!
    PATIENCE = 6    # Parameter for ReduceLROnPlateau()

BATCH_SIZE = 16

SAVE_RESULTS = True   # Only if there is only one simulation per model
FILERES = 'results.p' # Filename to store results

# Need to load the different models to get the funciotn name available
import efficientnet.tfkeras as efn
import tensorflow.keras.applications as app


## Diffferent configuration parameters to run different models simmultaneously
# modelos: Array with the parameters to perform a specific simmulation with a␣
 ↪model
# [name, function, preprocessing, learningSet, width, hight, freezeModel,␣
 ↪process]
#    name: name of the model of the Keras application
#    function: function of the Keras Application
#    preprocessing: Keras function for preprocessing images
```

```
#    learningset: 'noisy-student' or'imagenet'
#    width: Input width (in pixels) of the images required by the model
#    height: Input height (in pixels) of the images required by the model
#    freezeModel: TRUE. The base model is frozen and its parameters are not␣
 ↪trained
#    precess: True if we want to process this line
modelos = [
    ['EfficientNetB0', efn.EfficientNetB0, None, 'noisy-student', 224, 224,␣
 ↪False, True],
    ['EfficientNetB0', efn.EfficientNetB0, None, 'noisy-student', 224, 224,␣
 ↪True, False],
    ['EfficientNetB1', efn.EfficientNetB1, None, 'noisy-student', 240, 240,␣
 ↪False, False],
    ['EfficientNetB2', efn.EfficientNetB2, None, 'noisy-student', 260, 260,␣
 ↪False, False],
    ['EfficientNetB3', efn.EfficientNetB3, None, 'noisy-student', 300, 300,␣
 ↪False, False],
    ['EfficientNetB4', efn.EfficientNetB4, None, 'noisy-student', 380, 380,␣
 ↪False, False],
    ['EfficientNetB5', efn.EfficientNetB5, None, 'noisy-student', 456, 456,␣
 ↪False, False],
    ['EfficientNetB6', efn.EfficientNetB6, None, 'noisy-student', 528, 528,␣
 ↪False, False],
    ['EfficientNetB7', efn.EfficientNetB7, None, 'noisy-student', 600, 600,␣
 ↪False, False],
    ['Xception', app.Xception, app.xception.preprocess_input, 'imagenet', 229,␣
 ↪229, False, False],
    ['VGG16', app.VGG16, app.vgg16.preprocess_input, 'imagenet', 224, 224,␣
 ↪False, False],
    ['VGG19', app.VGG19, app.vgg19.preprocess_input, 'imagenet', 224, 224,␣
 ↪False, False],
    ['ResNet50', app.ResNet50, app.resnet.preprocess_input, 'imagenet', 224,␣
 ↪224, False, False],
    ['ResNet101521', app.ResNet101, app.resnet.preprocess_input, 'imagenet',␣
 ↪224, 224, False, False],
    ['ResNet152', app.ResNet152, app.resnet.preprocess_input, 'imagenet', 224,␣
 ↪224, False, False],
    ['ResNet50V2', app.ResNet50, app.resnet_v2.preprocess_input, 'imagenet',␣
 ↪224, 224, False, False],
    ['ResNet101V2', app.ResNet101V2, app.resnet_v2.preprocess_input,␣
 ↪'imagenet', 224, 224, False, False],
    ['ResNet152V2', app.ResNet152V2, app.resnet_v2.preprocess_input,␣
 ↪'imagenet', 224, 224, False, False],
    ['MobileNet', app.MobileNet, app.mobilenet.preprocess_input, 'imagenet',␣
 ↪224, 224, False, False],
```

```
    ['InceptionV3', app.InceptionV3, app.inception_v3.preprocess_input,␣
↪'imagenet', 229, 229, False, False],
    ['InceptionResNetV2', app.InceptionResNetV2, app.inception_resnet_v2.
↪preprocess_input, 'imagenet', 229, 229, False, False],
    ['DenseNet121', app.DenseNet121, app.densenet.preprocess_input, 'imagenet',␣
↪224, 224, False, False],
    ['DenseNet169', app.DenseNet169, app.densenet.preprocess_input, 'imagenet',␣
↪224, 224, False, False],
    ['DenseNet201', app.DenseNet201, app.densenet.preprocess_input, 'imagenet',␣
↪224, 224, False, False],
    ['NASNetLarge', app.NASNetLarge, app.nasnet.preprocess_input, 'imagenet',␣
↪224, 224, False, False],
    ['NASNetMobile', app.NASNetMobile, app.nasnet.preprocess_input, 'imagenet',␣
↪224, 224, False, False]
    ]

print('Ready!!')
```

```
Ready!!
```

1.1.3 GPU / TPU excution mode selection

Select GPU/TPU environment TPU must be selected in the `Google Colab` environment

From `Kaggle`: **Triple Stratified KFold with TFRecords** (Chris Deotte)

(TPU: Not working at the moment / can't be used)

[ ]:
```
import tensorflow as tf

if DEVICE == "TPU":
    print('connecting to TPU...')
    try:
        tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
        print('Running on TPU', tpu.master())
    except ValueError:
        print('Could not connect to TPU')
        tpu = None

    if tpu:
        try:
            print('Initializing TPU...')
            tf.config.experimental_connect_to_cluster(tpu)
            tf.tpu.experimental.initialize_tpu_system(tpu)
            strategy = tf.distribute.TPUStrategy(tpu)
            print('TPU initialized')
        except _:
            print('Failed to initialize TPU!!')
```

```
    else:
        DEVICE = "GPU"

if DEVICE != 'TPU':
    print("Using default strategy for CPU and single GPU")
    strategy = tf.distribute.get_strategy()

if DEVICE == 'GPU':
    print("Num  Available GPUs:",
          len(tf.config.experimental.list_physical_devices('GPU')))

AUTO = tf.data.experimental.AUTOTUNE
print(f'AUTO: {AUTO}')
REPLICAS = strategy.num_replicas_in_sync
print(f'REPLICAS: {REPLICAS}')

print("\ntf.__version__ is", tf.__version__)
print("tf.keras.__version__ is:", tf.keras.__version__)
```

```
Using default strategy for CPU and single GPU
Num  Available GPUs: 1
AUTO: -1
REPLICAS: 1

tf.__version__ is 2.3.0
tf.keras.__version__ is: 2.4.0
```

1.2 Input

Mounting Local Drive and download the images (only if needed: Colab environment)

In Local environment, the zip file is already present and previously extracted

[ ]:
```
if COLAB:
    # Mount drive
    from google.colab import drive
    import shutil
    import os
    from zipfile import ZipFile

    drive.mount("/content/drive")
    shutil.copy(DIRNAME+FILENAME, FILENAME)
    ZipFile(FILENAME, 'r').extractall()
    os.remove(FILENAME)
    print(ORIDIR+': {} files'.format(len(os.listdir(ORIDIR))))

TIPOS = os.listdir(ORIDIR)        # Automatic List of classes extracted from␣
 ↪directoies
```

```
print('Types: {}'.format(TIPOS))
```

```
Types: ['Parasitized', 'Uninfected']
```

1.3 Configuration

1. Import the images
2. Create two directories od the images with their labels according to classes
3. Configure training, testing and validate the model using stratified 10-fold cross validation.

Create DataFrames

[ ]:
```
import pandas as pd
import sklearn.model_selection
import random
import shutil
import os

ficheros = []  # The list of image file names

if os.path.isdir(DESTDIR):  # remove dir if exists (to avoid errors)
    shutil.rmtree(DESTDIR)

os.mkdir(DESTDIR)           # create destinate directory

df_total = pd.DataFrame()   # dataframe for filenames + class

for tipo in TIPOS:
    ficheros = os.listdir(ORIDIR+tipo)
    df_total = pd.concat([df_total,
                          pd.DataFrame(list(zip(ficheros,
                                                [tipo]*(len(ficheros)))),
                                       columns=['file','label'])])
    for fichero in ficheros:   # cleaning files from stange types
        if fichero.lower().endswith(('.png', '.jpg', '.jpeg',
                                     '.tiff', '.bmp', '.gif')):
            shutil.copy(ORIDIR+tipo+'/'+fichero, DESTDIR)
        else:
            os.remove(ORIDIR+tipo+'/'+fichero)
            print(f'Invalid file {fichero}')
            df_total = df_total.drop(df_total[df_total['file']==fichero].index)

if PERC != 0:     # Keep only a fraction of images to speed up the simmulation
    df_total = df_total.sample(n=round(len(df_total)*PERC/100.))

# Ramdonly split the images: 0.80 for learning /  0.20 for validatind
df_learn, df_val = sklearn.model_selection.train_test_split(df_total,
                                                            test_size=0.2)
df_learn = pd.concat([df_learn, pd.get_dummies(df_learn['label'])], axis=1)
```

```
df_val = pd.concat([df_val, pd.get_dummies(df_val['label'])], axis=1)

print("DataFrame Learn : {} files".format(len(df_learn)))
print("DataFrame Learn : {} files".format(len(df_val)))
```

```
DataFrame Learn : 22046 files
DataFrame Learn : 5512 files
```

1.4 Display random images

For testing purposes

[ ]:
```
import random
import matplotlib.pyplot as plt

N = 9  # Number of images of each class to display
for tipo in TIPOS:
    ficheros = list(df_total['file'][df_total['label']==tipo])
    images = random.sample(ficheros, N)
    print("\nImage Type: {}".format(tipo))
    plt.figure(figsize=(8,8))
    for i in range(N):
        plt.subplot(3, int(N/3),i+1)
        img = plt.imread(DESTDIR+images[i])
        plt.imshow(img)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

del [ficheros, df_total]
```

```
Image Type: Parasitized
```

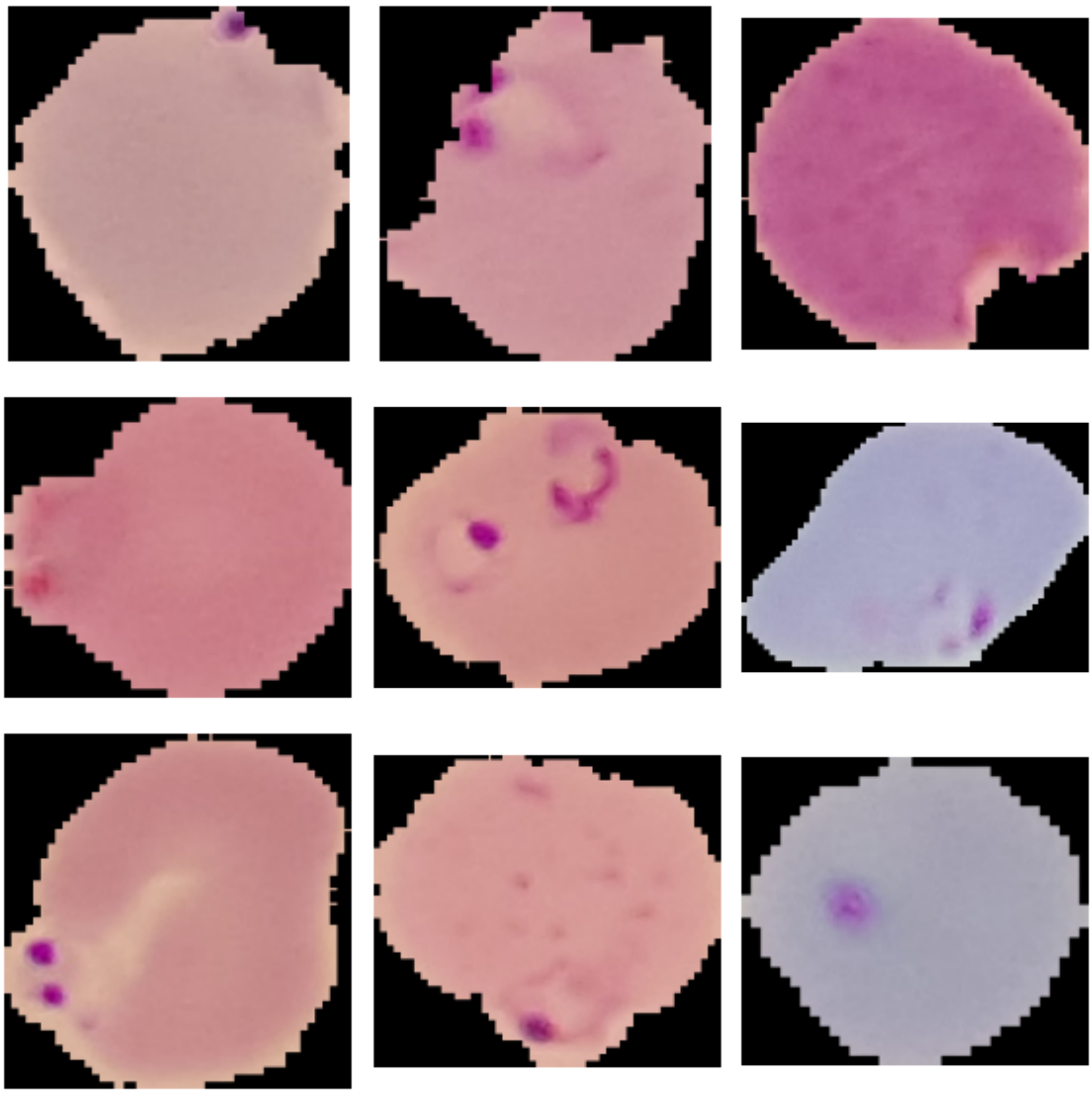
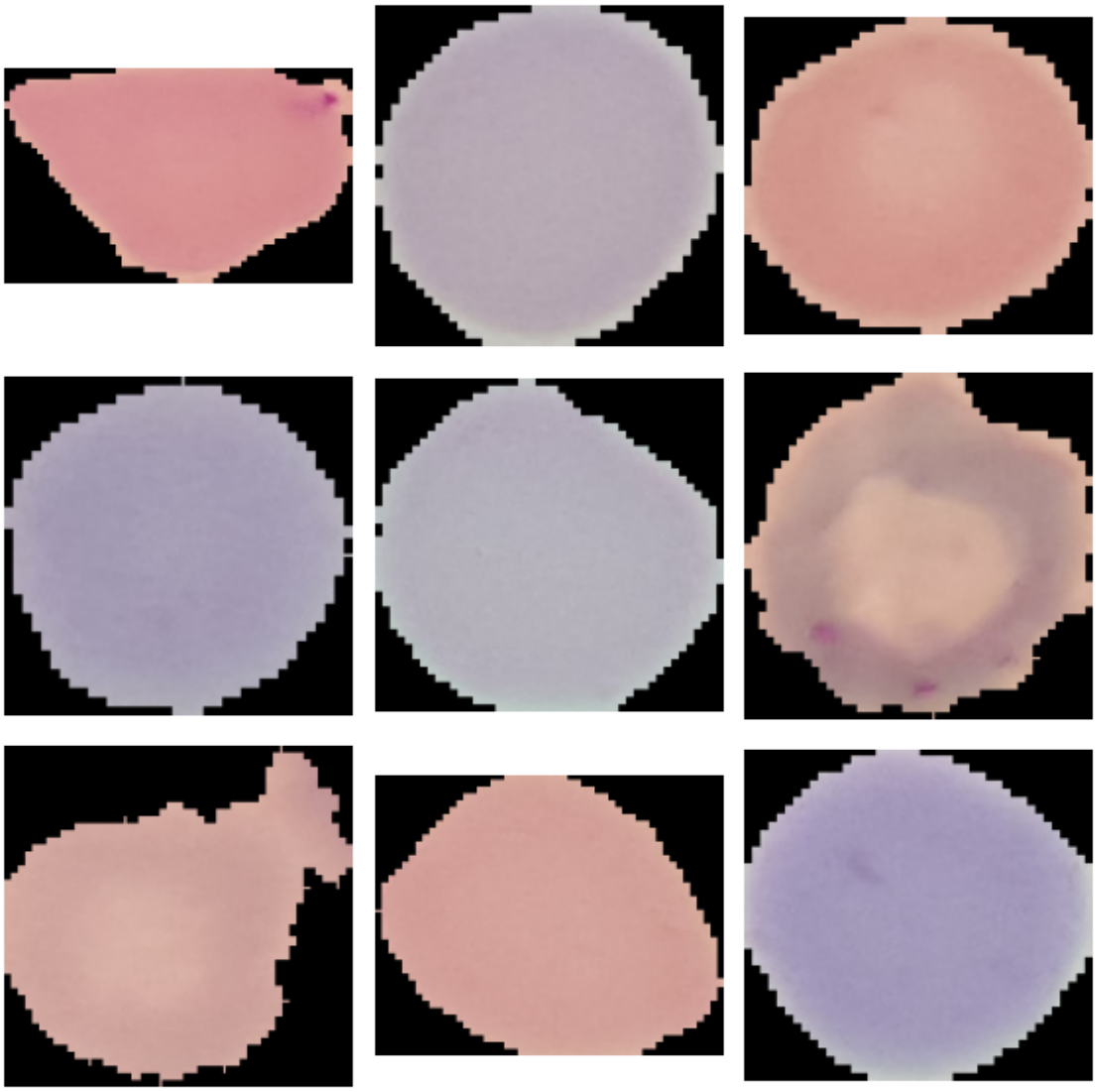


Image Type: Uninfected



2 MODEL

2.1 Auxiliary functions for plotting

[ ]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc
from itertools import cycle

# Plot confussion matrix
# y_true: real values
# y_pred: predicted values
```

```
# tipos: list with the names of the classes
def plot_confussion_matrix(y_true, y_pred, tipos):
    cm = confusion_matrix(y_true.values.argmax(axis = 1), y_pred.argmax(axis =␣
 ↪1))
    plt.figure(figsize=(6.5, 5))
    sns.heatmap(cm, annot=True, fmt="d")
    plt.title('\nAccuracy:{0:.4f}'.format(accuracy_score(y_true, y_pred)))
    plt.ylabel('True Class')
    plt.xlabel('Predicted Class')
    plt.xticks(np.arange(len(tipos))+0.5, tipos)
    plt.yticks(np.arange(len(tipos))+0.25, tipos)
    plt.show()

# Plot ROC (Area under the curve)
# y_true: real values
# y_pred: predicted values
# tipos: list with the names of the classes
def plot_roc_curve(y_true, y_pred, tipos):
    plt.style.use('ggplot')
    plt.figure(figsize=(6.5, 5))
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    n_classes=len(tipos)
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_true.values[:, i], y_pred[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Max. 6 classes
    colors = cycle(['blue', 'red', 'green', 'black', 'violet', 'brown'])

    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=1.5, \
                 label='ROC curve - {0} (area = {1:0.6f})'\
                 .format(TIPOS[i], roc_auc[i]))
    plt.plot([0, 1], [0, 1], linestyle='--', color = 'grey', lw=1)
    plt.xlim([-0.05, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')

    if n_classes == 2:
        plt.title('Receiver Operating Curve (binary-class)')
    else:
        plt.title('Receiver operating Curve (multi-class)')
    plt.legend(loc="lower right")
    plt.show()
```

```
# Plot history of the training
# hist: data to plot
# model: nName of the model
# fold: number of the fold of the k-fold algorithm
def plot_history(hist, model, fold):
    plt.figure(figsize=(8,5))
    plt.plot(hist['accuracy'],'-o',label='Train ACC',color='#ff7f0e')
    plt.plot(hist['val_accuracy'],'-o',label='Val ACC',color='#1f77b4')
    x = np.argmax(hist['val_accuracy']); y = np.max( hist['val_accuracy'])
    xdist = plt.xlim()[1] - plt.xlim()[0]; ydist = plt.ylim()[1] - plt.ylim()[0]
    plt.scatter(x,y,s=200,color='#1f77b4');
    plt.text(x-0.03*xdist,y-0.13*ydist,'max acc\n%.4f'%y,size=14)
    plt.ylabel('Accurary',size=14);
    plt.xlabel('Epoch',size=14)
    plt.legend(loc=2)

    plt2 = plt.gca().twinx()
    plt2.plot(hist['loss'],'-o',label='Train Loss',color='#2ca02c')
    plt2.plot(hist['val_loss'],'-o',label='Val Loss',color='#d62728')
    x = np.argmin(hist['val_loss'] ); y = np.min(hist['val_loss'] )
    ydist = plt.ylim()[1] - plt.ylim()[0]
    plt.scatter(x,y,s=200,color='#d62728');
    plt.text(x-0.03*xdist,y+0.05*ydist,'min loss\n%.4f'%y, size=11)
    plt.ylabel('Loss',size=14)
    plt.title('%s  -  FOLD : %i'%(model, fold), size=18)
    plt.legend(loc=3)
    plt.show()

print('Functions created!')
```

```
Functions created!
```

2.2 Auxiliary Functios for modeling

```
[ ]: from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout, Dense, GlobalAveragePooling2D
from tensorflow.keras.callbacks import ModelCheckpoint,ReduceLROnPlateau
import tensorflow as tf

# Create Keras model
# fnctn : function application of the base model
# wghs: pretrained weights to us in the model
# wdth: input width of the model
# hght: input heigh of the model
# frz: (Boolen) ¿train base model parameters?
def create_model(fnctn, wghs, wdth, hght, frz):
```

```
    model = fnctn(weights = wghs, include_top=False,
                  input_shape = (wdth, hght,3))

    if frz:
        model.trainable = False  # FIXING model

    x = model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = Dense(64, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = Dense(32, activation="relu")(x)
    x = Dropout(0.3)(x)
    predictions = Dense(len(TIPOS), activation="softmax")(x)
    model = Model(inputs=model.input, outputs=predictions)
    return model

# Callbacks for the model
# ReduceLROnPlateau: reduce the learning rate when learning dismishes
# ModelCheckPoint: to store models for future use
def create_callbacks(fold):
    return [ReduceLROnPlateau(monitor = 'val_loss', factor = 0.5,
                              patience = PATIENCE, min_lr = 0.000001),
            ModelCheckpoint('model_{}.hdf5'.format(fold), save_best_only = True,
                            monitor = 'val_loss', mode ='min',␣
 ↪save_freq='epoch')]

# Create custom_loss with smoothing parameter
def custom_loss(y_true, y_pred):
  return tf.keras.losses.categorical_crossentropy(y_true, y_pred,
                                                  label_smoothing=0.1)

# Convert y_pred to 0 or 1, selecting the maximum value of softmax
def binary_decission(y_pred, tipos):
    decission = np.zeros((len(y_pred), len(tipos)), dtype=int)
    for i in range(len(y_pred)):
        decission[i, int(np.where(y_pred[i] == np.amax(y_pred[i]))[0])] = 1
    return decission

print('Functions created!')
```

```
Functions created!
```

2.3 Augmentation

Use **albumentations** for image augmentation. The purpose of image augmentation is to create new training samples from the existing data. We use the following transformation in our model:

- `Flip`: Flip the input either horizontally, vertically or both horizontally and vertically.
- `Transpose`: Transpose the input by swapping rows and columns.
- `IAAAdditiveGaussianNoise`: Add gaussian noise to the input image.
- `GaussNoise`: Apply gaussian noise to the input image.
- `MotionBlur`: Apply motion blur to the input image using a random-sized kernel.
- `MedianBlur`: Blur the input image using a median filter with a random aperture linear size.
- `Blur`: Blur the input image using a random-sized kernel.
- `ShiftScaleRotate`: Randomly apply affine transforms: translate, scale and rotate the input.
- `OpticalDistortion`: Apply an optical distortion to the full image.
- `GridDistortion`: Apply a grid distortion with padding.
- `IAAPiecewiseAffine`: Place a regular grid of points on the input and randomly move the neighbourhood of these point around via affine transformations.
- `CLAHE`: Apply Contrast Limited Adaptive Histogram Equalization to the input image.
- `IAASharpen`: Sharpen the input image and overlays the result with the original image. This augmentation is deprecated. Please use Sharpen instead.
- `IAAEmboss`: Emboss the input image and overlays the result with the original image.
- `RandomContrast`: Randomly change contrast of the input image.
- `RandomBrightness`: Randomly change brightness and contrast of the input image.

[ ]:
```
from albumentations import *

aug=Compose([RandomRotate90(),
             Flip(),
             Transpose(),
             OneOf([IAAAdditiveGaussianNoise(),
                    GaussNoise(),], p=0.2),
             OneOf([MotionBlur(p=.2),
                    MedianBlur(blur_limit=3, p=.1),
                    Blur(blur_limit=3, p=.1),], p=0.3),
             ShiftScaleRotate(shift_limit=0.0625,
                              scale_limit=0.2,
                              rotate_limit=45, p=.2),
             OneOf([OpticalDistortion(p=0.3),
                    GridDistortion(p=.1),
                    IAAPiecewiseAffine(p=0.3),], p=0.3),
             OneOf([CLAHE(clip_limit=2),
                    IAASharpen(),
                    IAAEmboss(),
                    RandomContrast(),
                    RandomBrightness(),], p=0.3),
             ], p=1)
print('Ready!')
```

```
Ready!
```

2.4 Training

For each model:

- Define augmentation pipeline using Compose function of albumentation library.
- Create the data generator as an object of the ImageDataAugmentator library and configure the augmentation pipeline obtained previously. Do this for train / test / vlaidation dataset.
- Create the model using the appropriate function and dense layers with relu activation function and an output layer with a softmax activation function.
- Compile the model using the ADAM optimizer and Categorical_Crossentropy function for loss calculation.
- Model fitting using 33 epochs and ReduceLRonPlateau function to reduce the learning rate when the metrics stops improving.
- Save the model to be used for validation testing and ensemble model.
- Configure testing dataset.
- Generate performance score values for each fold.
    1. Model loss graph.
    2. Model Accuracy graph.
    3. Test Classification Report.
    4. Validate Model.
    5. AUC-ROC curve.
    6. Confusion Matrix.
    7. Validation Classification Report.
- Generate performance score values dor ensembled model
    1. Ensembled Classification Report.
    2. AUC-ROC curve.
    3. Confusion Matrix.

[ ]:
```
from ImageDataAugmentor.image_data_augmentor import *

from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import StratifiedKFold
import gc
from sklearn.metrics import classification_report, accuracy_score, f1_score
from datetime import datetime
import pickle

# To reproduce results. SEED defined in the PARAMETER section
random.seed(SEED)

# If saving results is desired
if SAVE_RESULTS:
    total_results = {'epochs' : EPOCHS, 'batch_size' : BATCH_SIZE,
                     'filename' : FILENAME, 'KFOLD' : KFOLD}

# Loop to simulate. modelos list defined in the parameter section
```

```
for (nombre, funcion, preproc, pesos, ancho, alto, freeze, procesar) in modelos:

    # Skip model if required
    if not procesar:
        continue
    else: # Presentation for starting simmulation
        print('#'*50) ; print('### ', nombre); print('#'*50);
        if freeze:
            print('Model frozen....')

    # Start point for Time measurement
    start_model = datetime.now()

    # List of result values initialization
    test_true = {}; test_pred = {}; val_true = {}; val_pred = {}

    # Define custom image data generator with support for albumentations
    data_gen = ImageDataAugmentor(rescale=1/255, augment = aug,
                                  preprocess_input = preproc)

    # Stratified K-Folds cross-validator. Provides train/test indices to split␣
↪data
    # in train/test sets. KFOLD: number of folds; defined in PARAMETER section
    kf = StratifiedKFold(KFOLD, shuffle = True, random_state = 50)

    # Loop for K interations (from k-fold)
    for fold, (train_index, test_index) in enumerate(kf.split(df_learn,
                                                              df_learn['label'])):
        # fine grain measurement for each fold
        start_fold = datetime.now()

        # presentation
        print('*'*50) ;print('** Model: {}  fold: {} '.format(nombre, fold))
        print('Training...')

        # Create sets of train and test images
        df_train = df_learn.iloc[train_index,:]
        df_test = df_learn.iloc[test_index,:]

        # Create data generator for train, test and validation
        # use de ImageDataAugmentor object previously defined
        train_generator = data_gen.flow_from_dataframe(
            df_train, directory= 'data',
            target_size=(ancho, alto), x_col = "file", y_col = TIPOS,
            class_mode = 'raw', shuffle = True, batch_size = BATCH_SIZE)

        test_generator = data_gen.flow_from_dataframe(
```

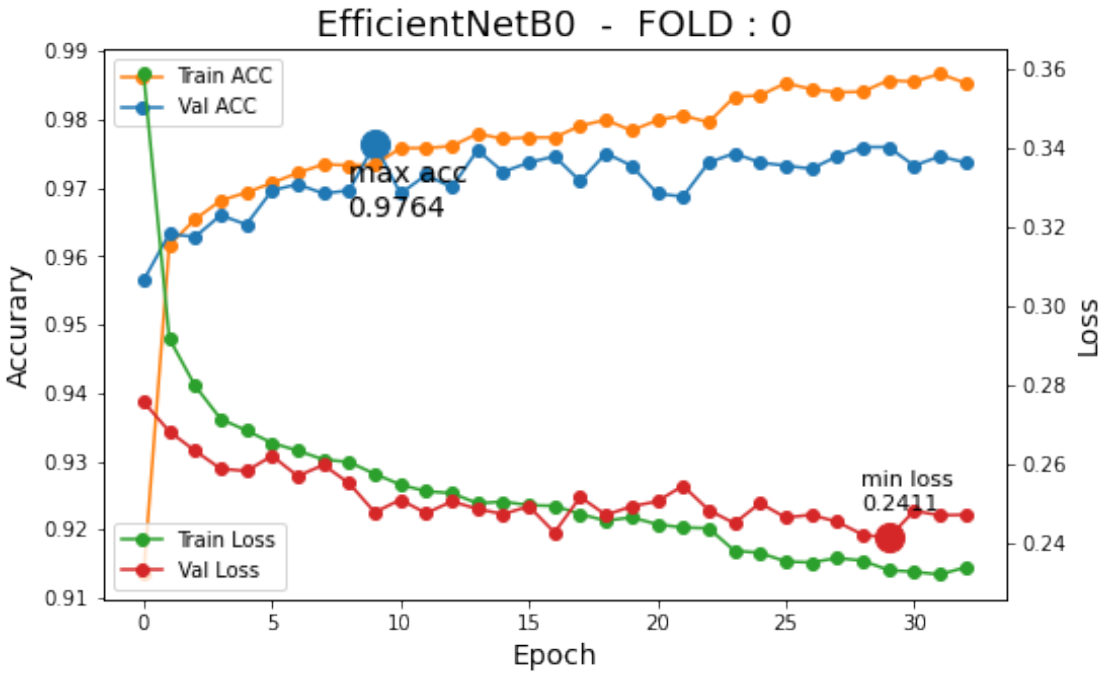
```
        df_test, directory='data',
        target_size=(ancho, alto), x_col = "file", y_col = TIPOS,
        class_mode ='raw', shuffle = False, batch_size = BATCH_SIZE)

    val_generator = data_gen.flow_from_dataframe(
        df_val, directory='data',
        target_size = (ancho, alto), x_col = "file", y_col = TIPOS,
        class_mode ='raw', shuffle = False, batch_size = BATCH_SIZE)

    # Create the model
    model = create_model(funcion, pesos, ancho, alto, freeze)

    # Compile the model
    model.compile(optimizer=Adam(0.0001), loss=custom_loss,
                  metrics=['accuracy'])

    # Train
    results = model.fit(train_generator, epochs = EPOCHS,
                        steps_per_epoch = train_generator.n/BATCH_SIZE,
                        validation_data = test_generator,
                        validation_steps = test_generator.n/BATCH_SIZE,
                        callbacks = create_callbacks(fold),
                        verbose = VERBOSE)

    # Plot history
    plot_history(results.history, nombre, fold)

    # Get best model of the training phase for this fold
    model.load_weights('model_{}.hdf5'.format(fold))

    # Predict class for test images
    print('Predicting...')
    test_generator.reset()
    test_true[f'fold{fold}'] = df_test.iloc[:,2::]
    test_pred[f'fold{fold}'] = model.predict(test_generator,
                                steps=test_generator.n/BATCH_SIZE,
                                verbose=VERBOSE)

    # Get and print/plot results for testing
    decission = binary_decission(test_pred[f'fold{fold}'], TIPOS)
    print('Accuracy {:.6f}'.format(accuracy_score(test_true[f"fold{fold}"],
                                                  decission)))
    print(classification_report(test_true[f"fold{fold}"], decission,
                                target_names = TIPOS, digits = 6))

    # Validate results with the appropriate set of samples
    print('Validating...')
```

```
        val_generator.reset()
        val_true[f'fold{fold}'] = df_val.iloc[:,2::]
        val_pred[f'fold{fold}'] = model.predict(val_generator,
                                                steps=val_generator.n/
↪BATCH_SIZE,
                                                verbose=1)

        # Get and print/plot results for validation
        plot_roc_curve(val_true[f'fold{fold}'], val_pred[f'fold{fold}'],TIPOS)
        decission = binary_decission(val_pred[f'fold{fold}'], TIPOS)
        plot_confussion_matrix(val_true[f'fold{fold}'], decission, TIPOS)
        print(classification_report(val_true[f"fold{fold}"], decission,
                                    target_names = TIPOS, digits = 6))

        # Timmings for each k-fold iteration
        print('End of Fold {} - Elapsed Time: {}'.format(fold, datetime.now() -␣
↪start_fold))

    # End of K-FOLD proccess. Clean environment
    del model, data_gen, train_generator
    tf.keras.backend.clear_session()
    gc.collect()

    # Calculate ensembled results from the K models
    print('\n Validate emsembled results...')
    val_ensemble = np.array([[0.,0.]]*len(df_val))
    for fold in range(KFOLD):
        val_ensemble = val_pred[f'fold{fold}'] + val_ensemble
    val_ensemble = np.divide(val_ensemble, np.array([KFOLD,KFOLD]))

    # Print / Plot results
    val_true = df_val.iloc[:,2::]
    plot_roc_curve(val_true, val_ensemble,TIPOS)
    decission = binary_decission(val_ensemble, TIPOS)
    print('Accuracy {:.6f}'.format(accuracy_score(val_true, decission)))
    plot_confussion_matrix(val_true, decission, TIPOS)
    f1 = f1_score(val_true, decission, average="macro")
    print('f1 score: {:.6f}'.format(f1))
    print(classification_report(val_true, decission,
                                target_names = TIPOS, digits = 6))

    # Total time spent
    time_taken = datetime.now() - start_model;

    print('\n END Model {} - Elapsed Time: {}'.format(nombre,
                                                     datetime.now() - start_model))
    # Saving results
```

```
    if SAVE_RESULTS:
        total_results[nombre] = {'test_true' : test_true, 'test_pred' :␣
 ↪test_pred,
                                 'val_true' : val_true, 'val_pred' : val_pred,
                                 'time_taken' : time_taken }
if SAVE_RESULTS:
    with open(DIRNAME+FILERES, "wb") as f:
        pickle.dump(total_results, f)
```

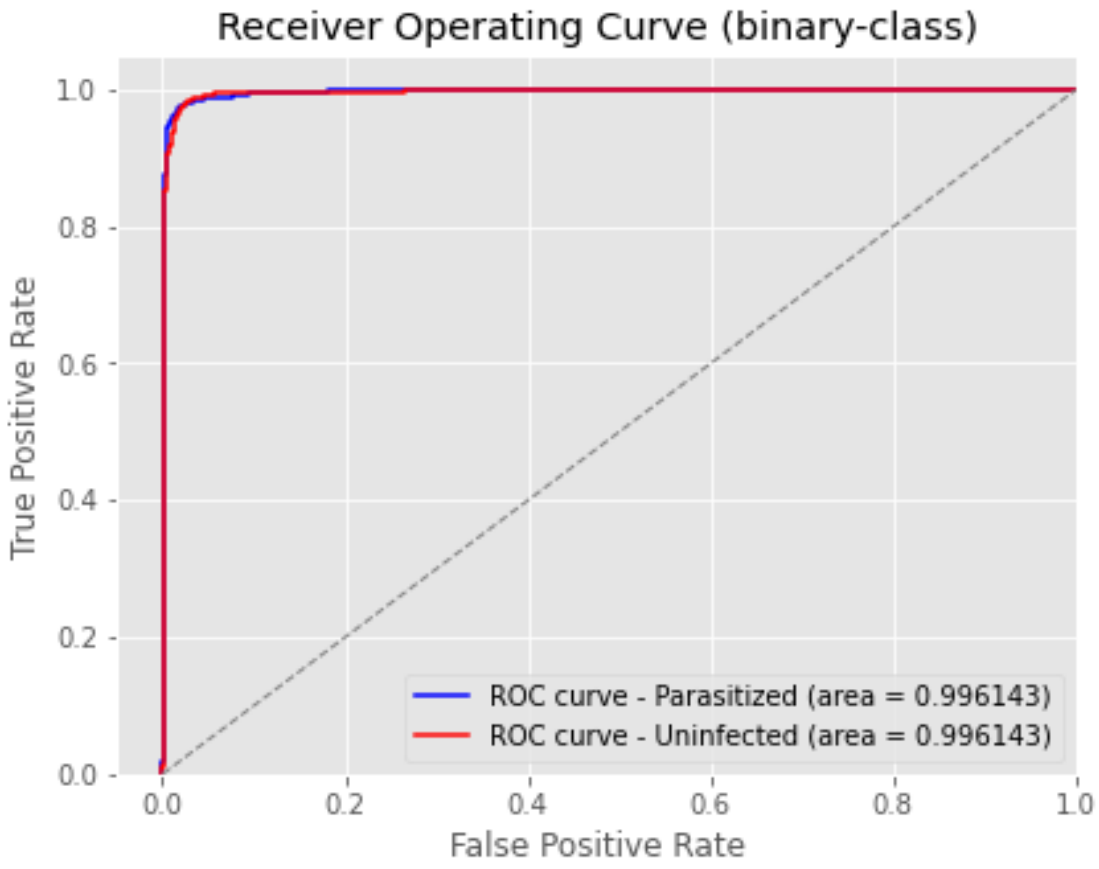
```
Using TensorFlow backend.
```

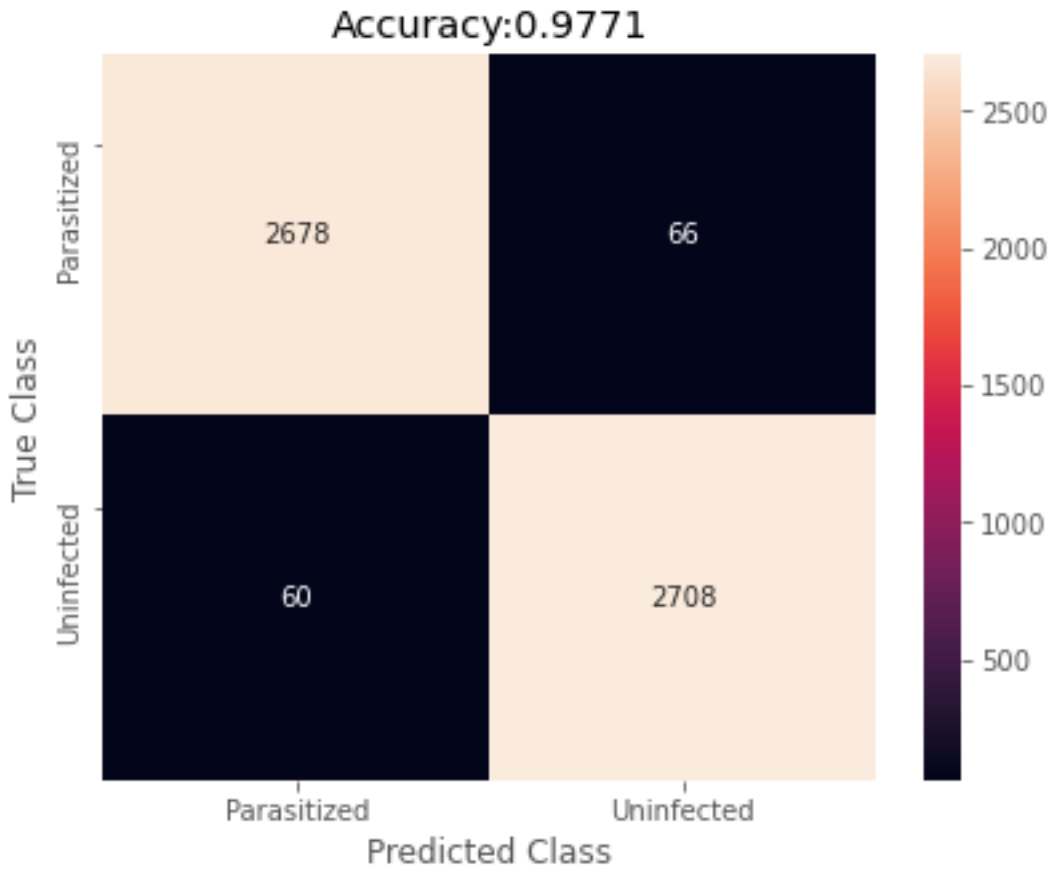
```
######################################################
###  EfficientNetB0
######################################################
******************************************************
** Model: EfficientNetB0  fold: 0
Training…
Found 19841 validated image filenames.
Found 2205 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.1006s vs `on_train_batch_end` time: 0.2411s). Check
your callbacks.
```




```
Predicting…
Accuracy 0.974603
```

```
              precision    recall  f1-score   support

 Parasitized   0.977231  0.971920  0.974569      1104
  Uninfected   0.971996  0.977293  0.974638      1101

   micro avg   0.974603  0.974603  0.974603      2205
   macro avg   0.974614  0.974607  0.974603      2205
weighted avg   0.974617  0.974603  0.974603      2205
 samples avg   0.974603  0.974603  0.974603      2205

Validating…
345/344 [==============================] - 109s 317ms/step
```

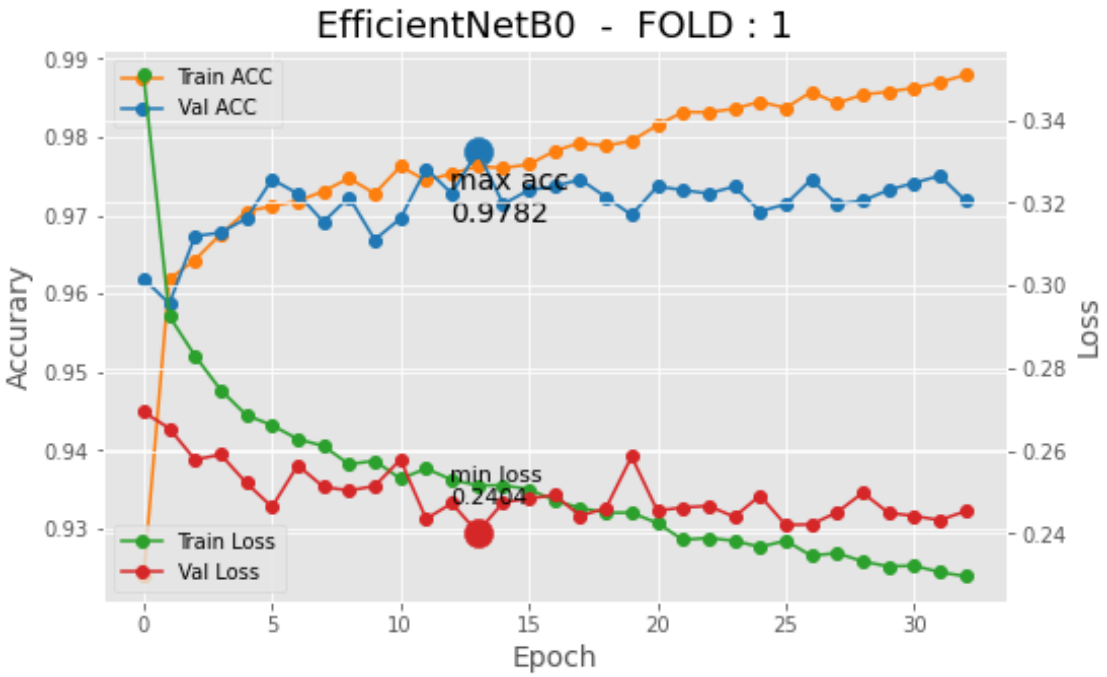




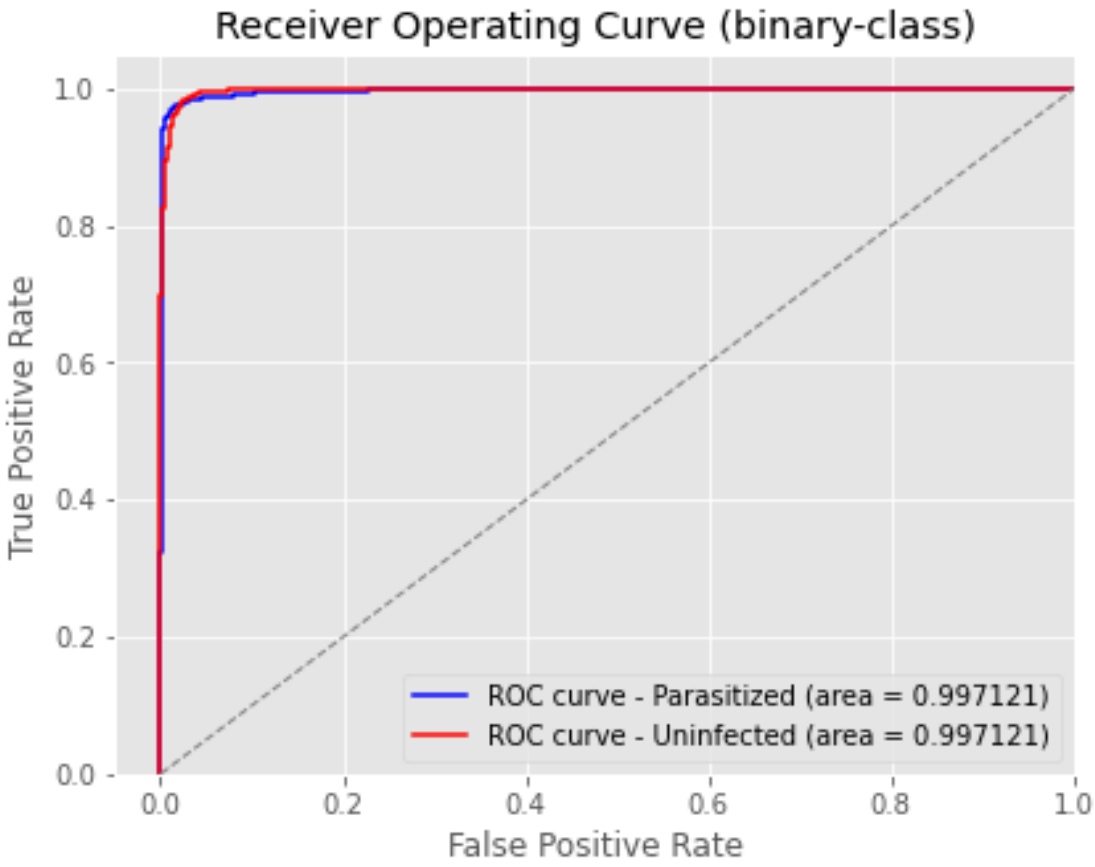
	precision	recall	f1-score	support
Parasitized	0.978086	0.975948	0.977016	2744
Uninfected	0.976208	0.978324	0.977265	2768
micro avg	0.977141	0.977141	0.977141	5512
macro avg	0.977147	0.977136	0.977140	5512
weighted avg	0.977143	0.977141	0.977141	5512
samples avg	0.977141	0.977141	0.977141	5512

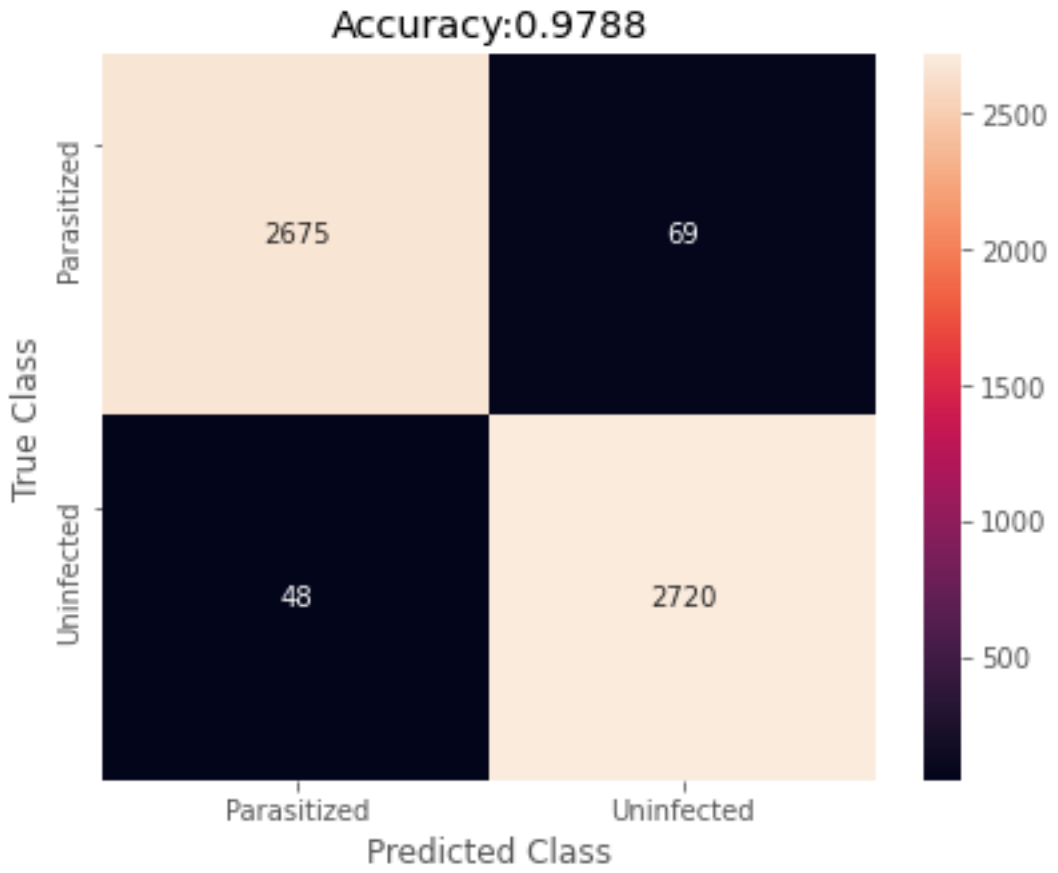
```
End of Fold 0 - Elapsed Time: 4:57:09.745551
**************************************************
** Model: EfficientNetB0  fold: 1
Training…
Found 19841 validated image filenames.
Found 2205 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

```
batch time (batch time: 0.1017s vs `on_train_batch_end` time: 0.2404s). Check
your callbacks.
```




```
Predicting…
Accuracy 0.977324
              precision    recall  f1-score   support

 Parasitized   0.978221  0.976449  0.977335      1104
  Uninfected   0.976428  0.978202  0.977314      1101

   micro avg   0.977324  0.977324  0.977324      2205
   macro avg   0.977325  0.977325  0.977324      2205
weighted avg   0.977326  0.977324  0.977324      2205
 samples avg   0.977324  0.977324  0.977324      2205

Validating…
345/344 [==============================] - 84s 243ms/step
```

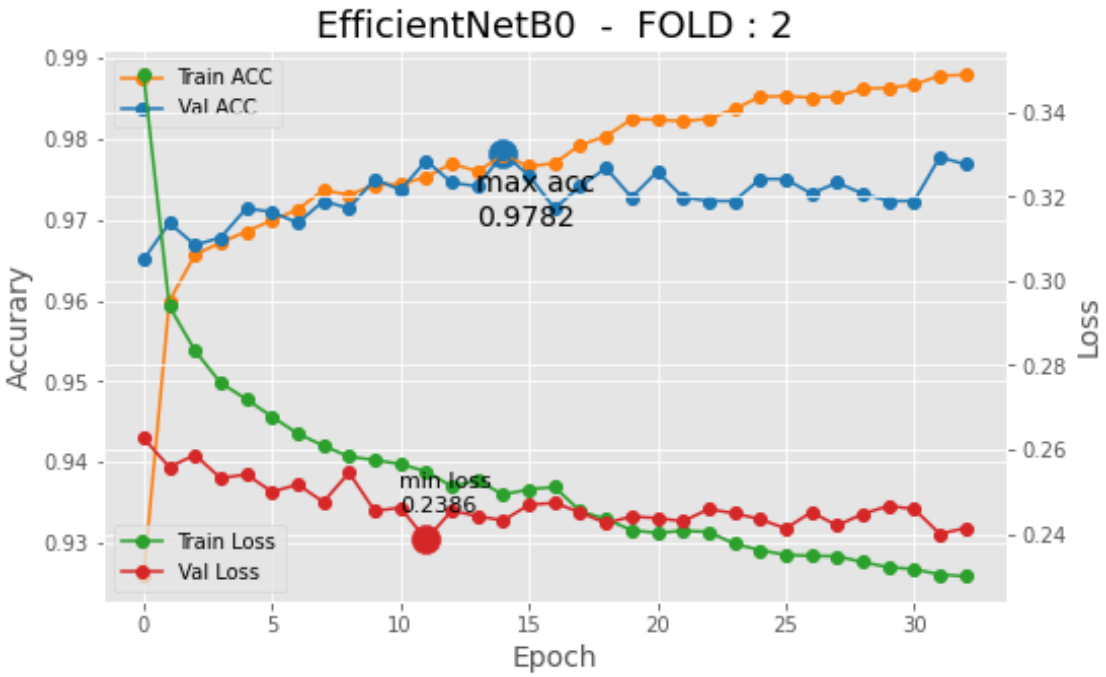

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.997121)
ROC curve - Uninfected (area = 0.997121)



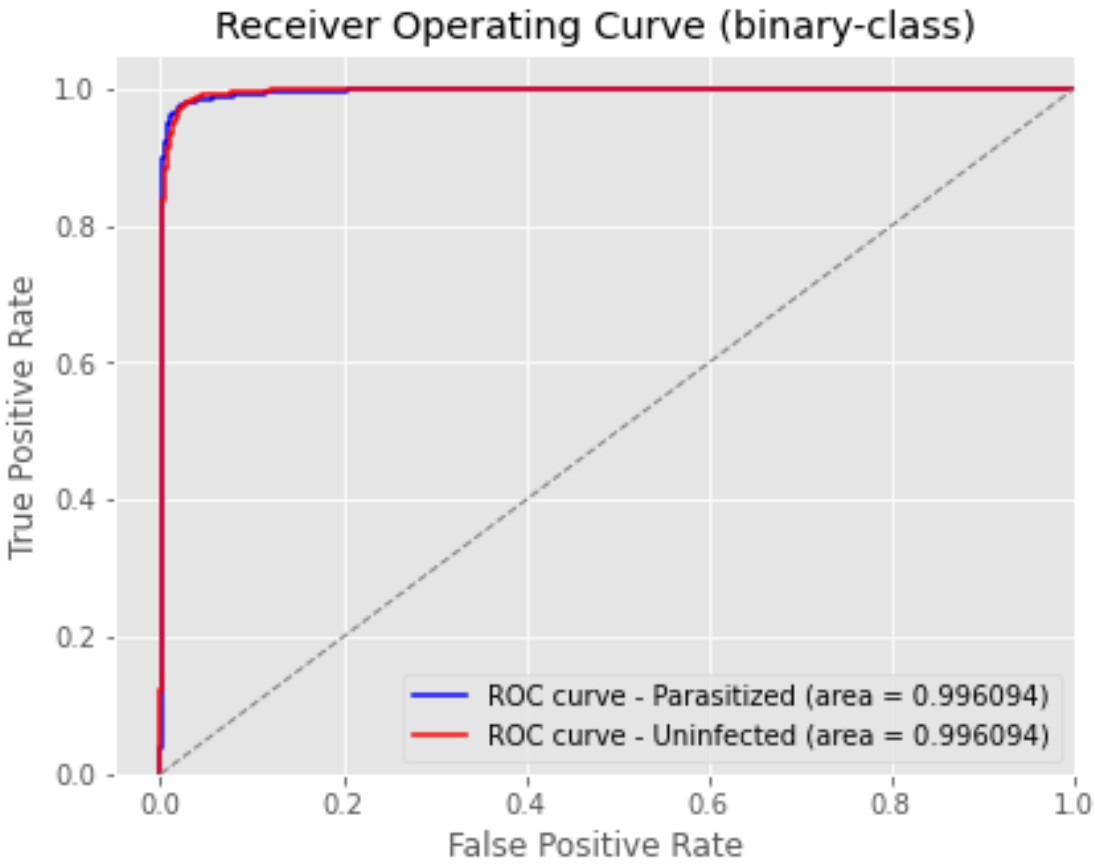
	precision	recall	f1-score	support
Parasitized	0.982372	0.974854	0.978599	2744
Uninfected	0.975260	0.982659	0.978945	2768
micro avg	0.978774	0.978774	0.978774	5512
macro avg	0.978816	0.978757	0.978772	5512
weighted avg	0.978801	0.978774	0.978773	5512
samples avg	0.978774	0.978774	0.978774	5512

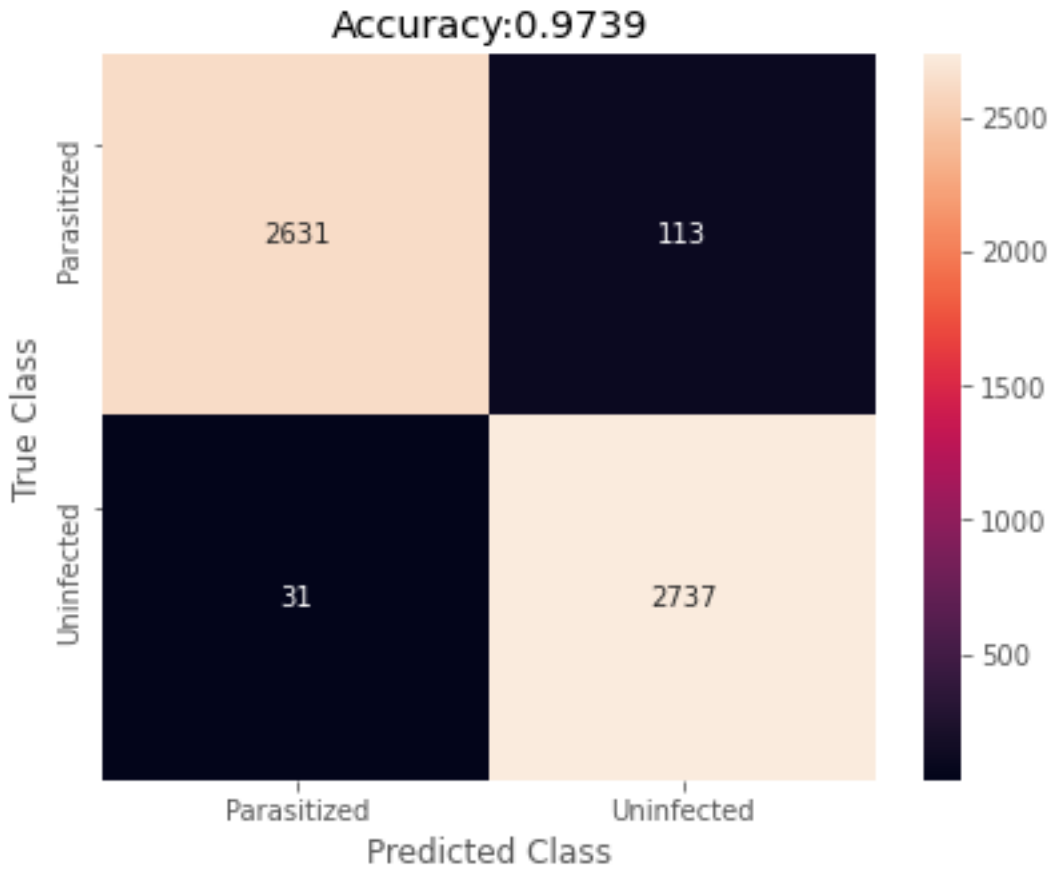
```
End of Fold 1 - Elapsed Time: 4:43:37.369983
************************************************
** Model: EfficientNetB0  fold: 2
Training…
Found 19841 validated image filenames.
Found 2205 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

batch time (batch time: 0.1007s vs `on_train_batch_end` time: 0.2453s). Check your callbacks.




```
Predicting…
Accuracy 0.976871
              precision    recall  f1-score   support

 Parasitized   0.981702  0.971920  0.976787      1104
  Uninfected   0.972122  0.981835  0.976954      1101

   micro avg   0.976871  0.976871  0.976871      2205
   macro avg   0.976912  0.976877  0.976870      2205
weighted avg   0.976919  0.976871  0.976870      2205
 samples avg   0.976871  0.976871  0.976871      2205

Validating…
345/344 [==============================] - 80s 231ms/step
```

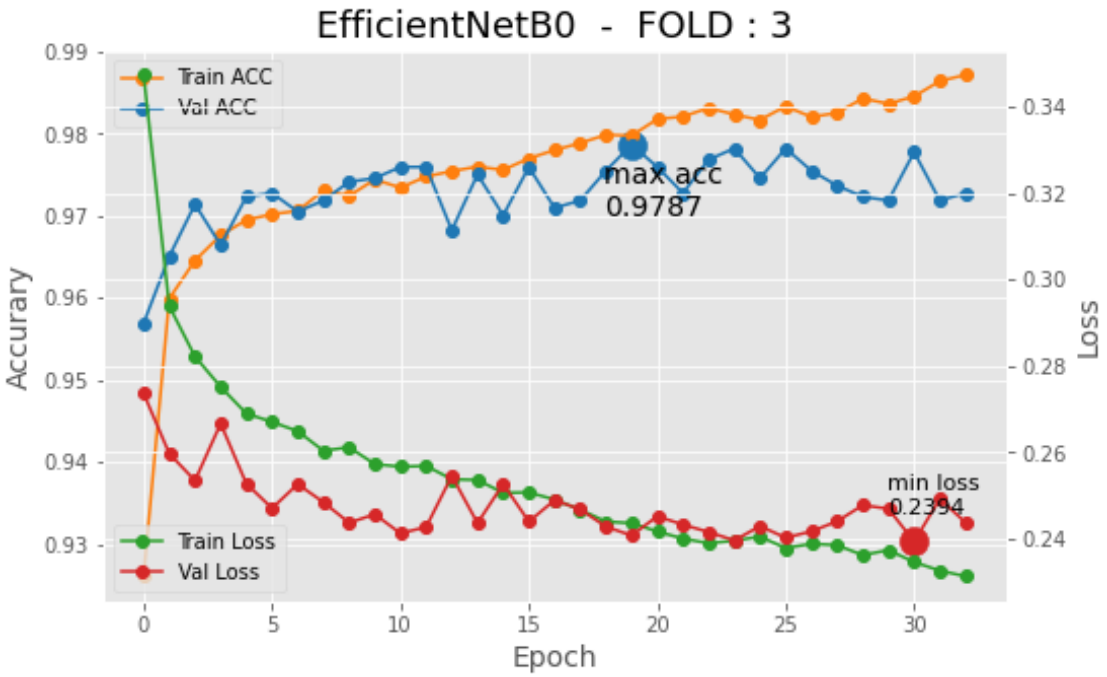

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996094)
ROC curve - Uninfected (area = 0.996094)



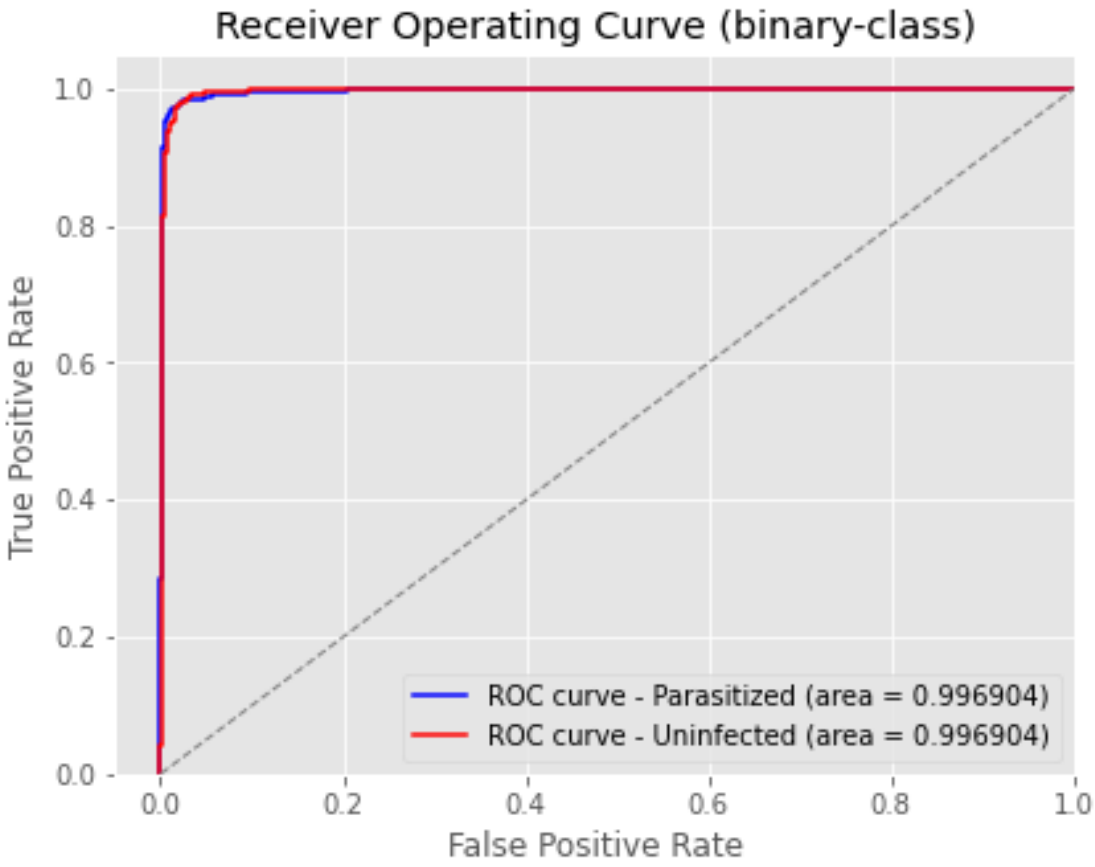
	precision	recall	f1-score	support
Parasitized	0.988355	0.958819	0.973363	2744
Uninfected	0.960351	0.988801	0.974368	2768
micro avg	0.973875	0.973875	0.973875	5512
macro avg	0.974353	0.973810	0.973866	5512
weighted avg	0.974292	0.973875	0.973868	5512
samples avg	0.973875	0.973875	0.973875	5512

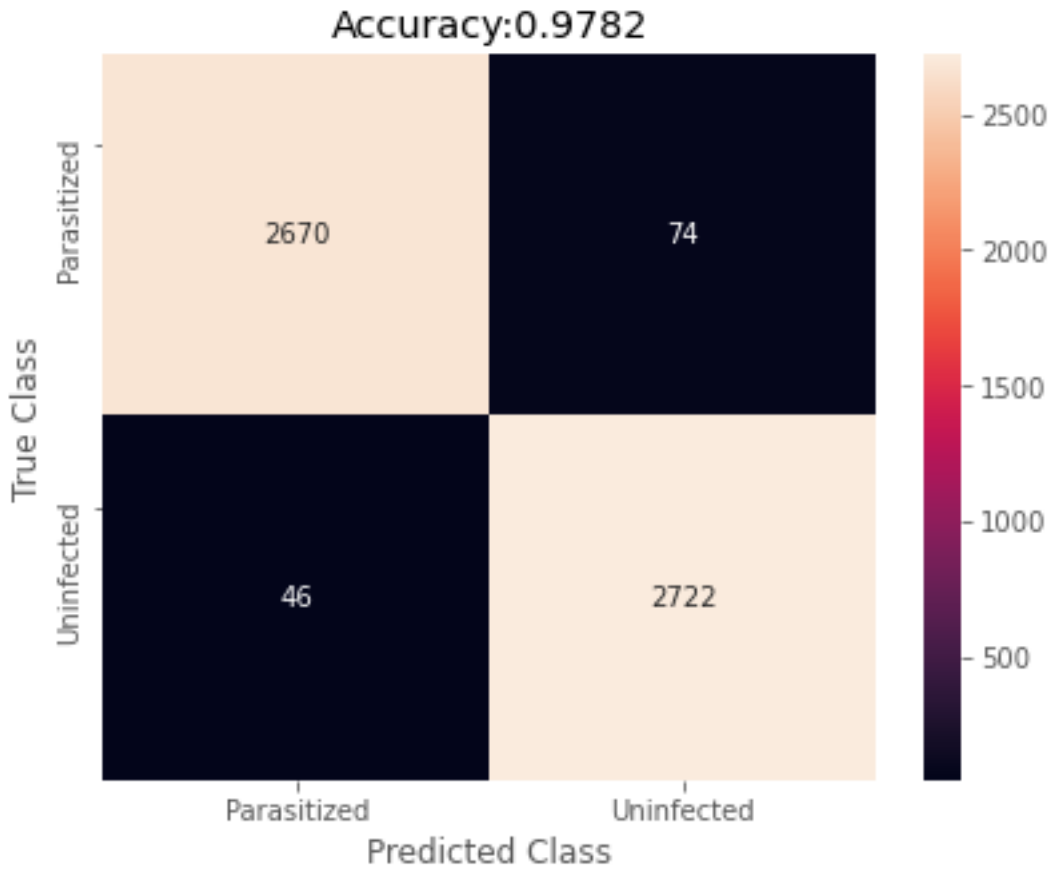
```
End of Fold 2 - Elapsed Time: 4:38:21.031119
************************************************
** Model: EfficientNetB0  fold: 3
Training…
Found 19841 validated image filenames.
Found 2205 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

batch time (batch time: 0.1017s vs `on_train_batch_end` time: 0.2464s). Check your callbacks.




```
Predicting…
Accuracy 0.974603
              precision    recall  f1-score   support

 Parasitized   0.978102  0.971014  0.974545      1104
  Uninfected   0.971145  0.978202  0.974661      1101

   micro avg   0.974603  0.974603  0.974603      2205
   macro avg   0.974624  0.974608  0.974603      2205
weighted avg   0.974628  0.974603  0.974603      2205
 samples avg   0.974603  0.974603  0.974603      2205

Validating…
345/344 [==============================] - 79s 230ms/step
```

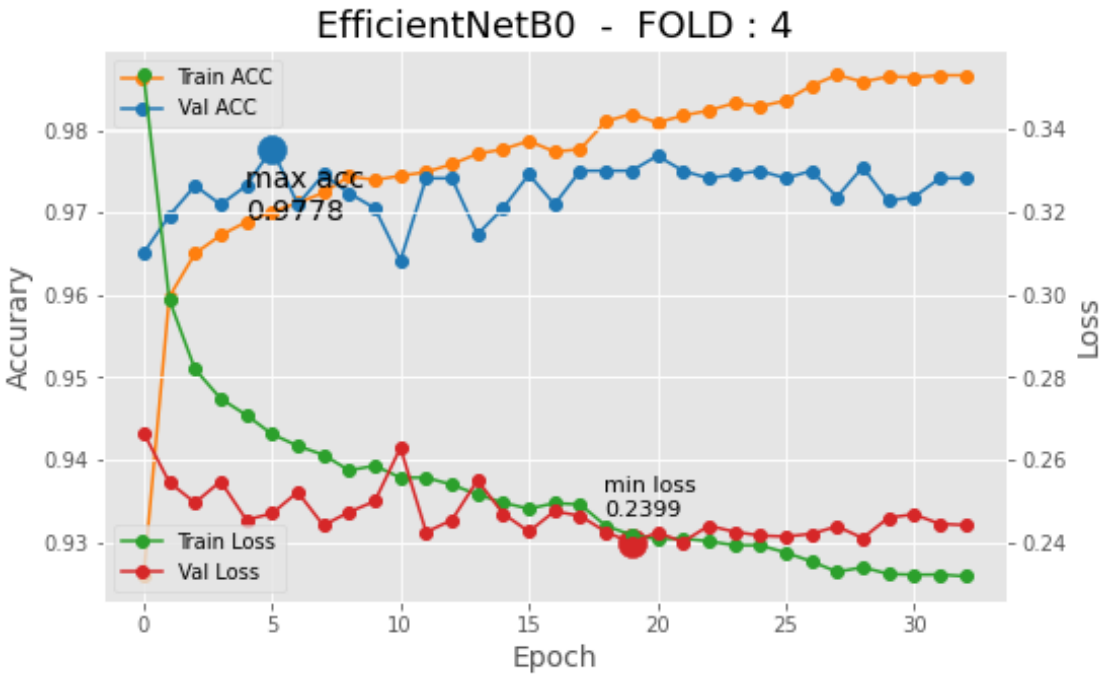

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996904)
ROC curve - Uninfected (area = 0.996904)



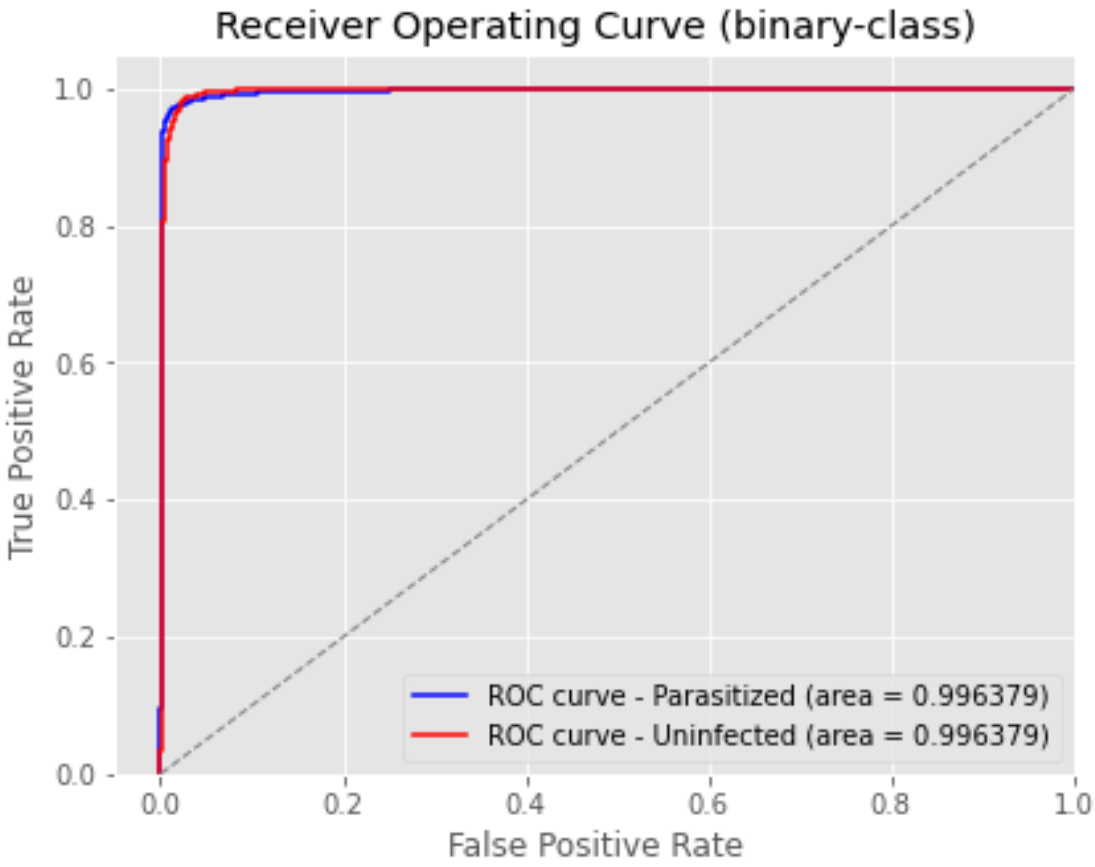
	precision	recall	f1-score	support
Parasitized	0.983063	0.973032	0.978022	2744
Uninfected	0.973534	0.983382	0.978433	2768
micro avg	0.978229	0.978229	0.978229	5512
macro avg	0.978298	0.978207	0.978227	5512
weighted avg	0.978278	0.978229	0.978228	5512
samples avg	0.978229	0.978229	0.978229	5512

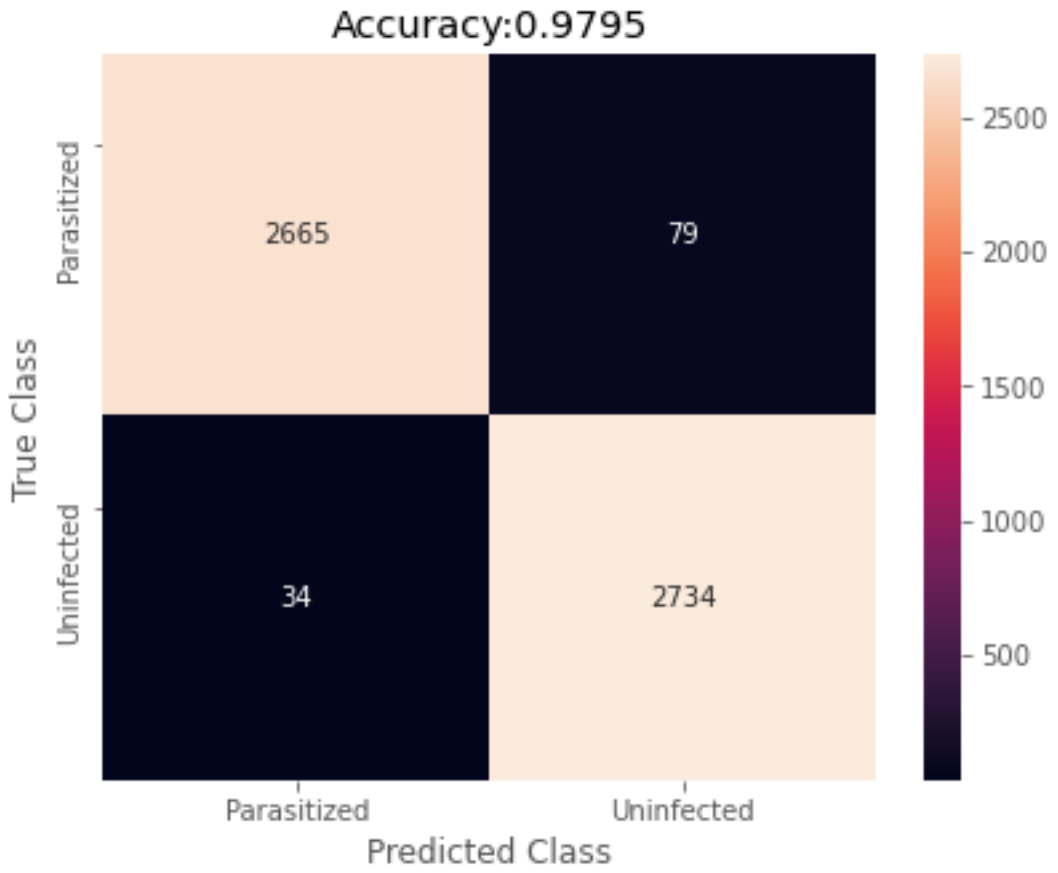
```
End of Fold 3 - Elapsed Time: 4:35:22.530893
************************************************
** Model: EfficientNetB0  fold: 4
Training…
Found 19841 validated image filenames.
Found 2205 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

batch time (batch time: 0.1007s vs `on_train_batch_end` time: 0.2443s). Check your callbacks.




```
Predicting…
Accuracy 0.973243
              precision    recall  f1-score   support

 Parasitized   0.978042  0.968297  0.973145      1104
  Uninfected   0.968525  0.978202  0.973339      1101

   micro avg   0.973243  0.973243  0.973243      2205
   macro avg   0.973284  0.973249  0.973242      2205
weighted avg   0.973290  0.973243  0.973242      2205
 samples avg   0.973243  0.973243  0.973243      2205

Validating…
345/344 [==============================] - 78s 227ms/step
```

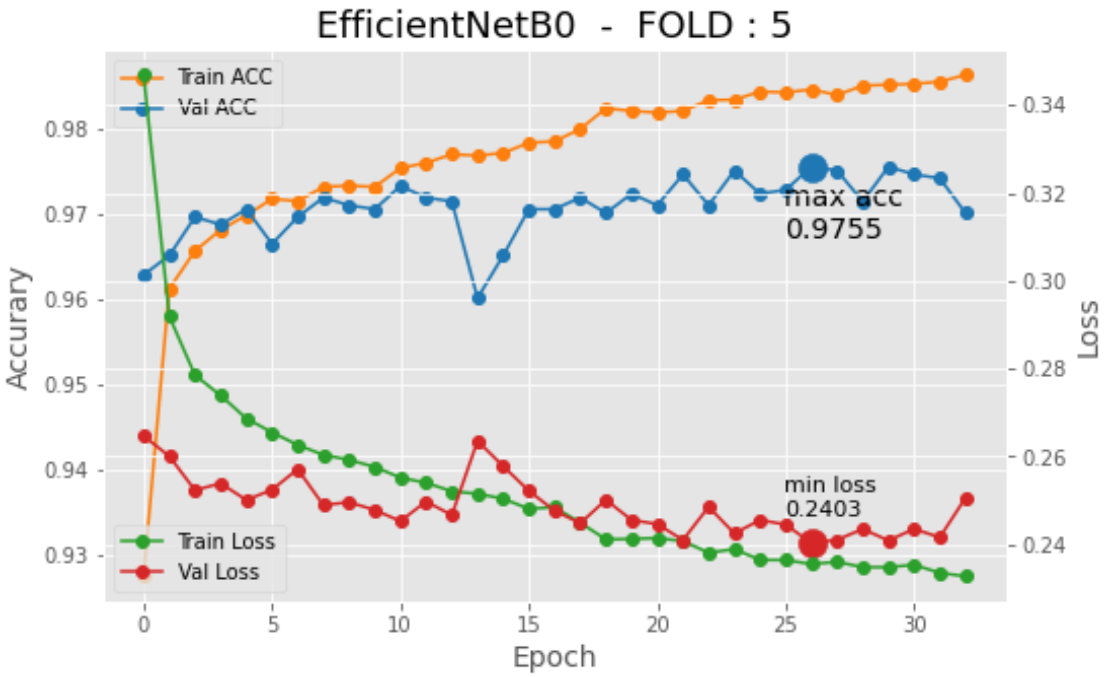

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996379)
ROC curve - Uninfected (area = 0.996379)



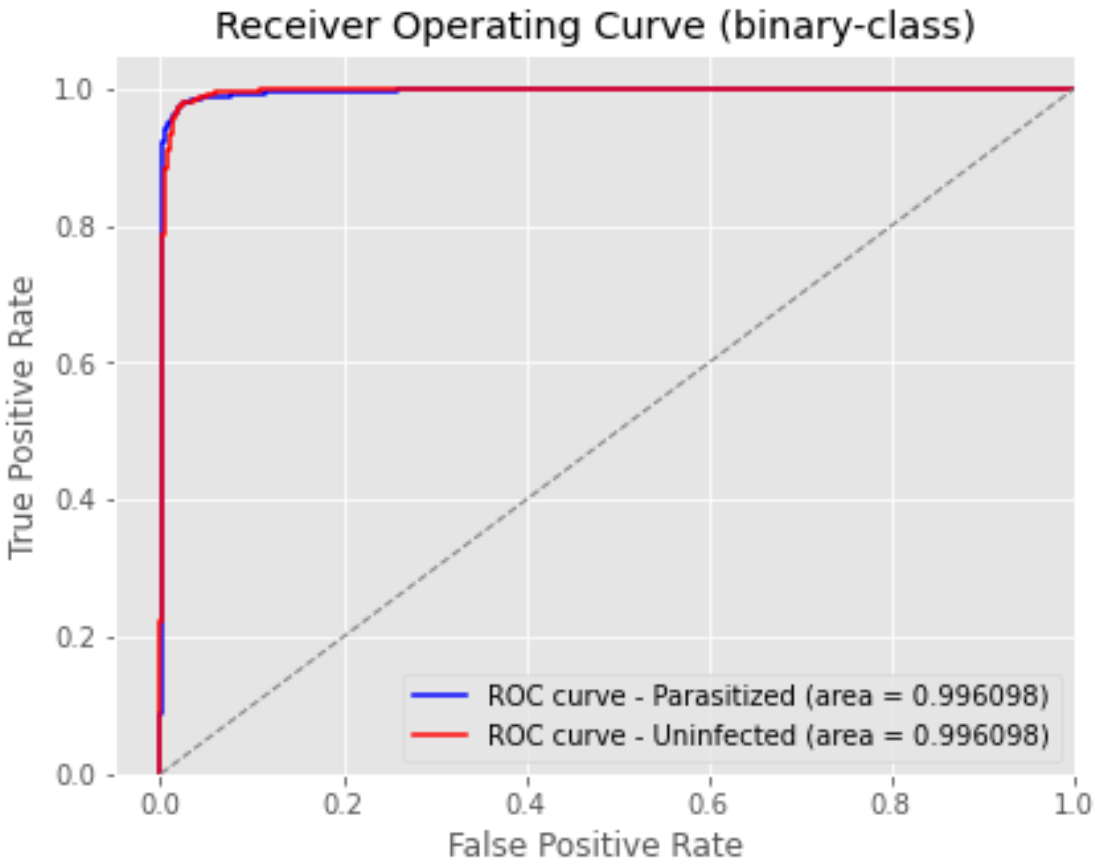
	precision	recall	f1-score	support
Parasitized	0.987403	0.971210	0.979239	2744
Uninfected	0.971916	0.987717	0.979753	2768
micro avg	0.979499	0.979499	0.979499	5512
macro avg	0.979659	0.979463	0.979496	5512
weighted avg	0.979626	0.979499	0.979497	5512
samples avg	0.979499	0.979499	0.979499	5512

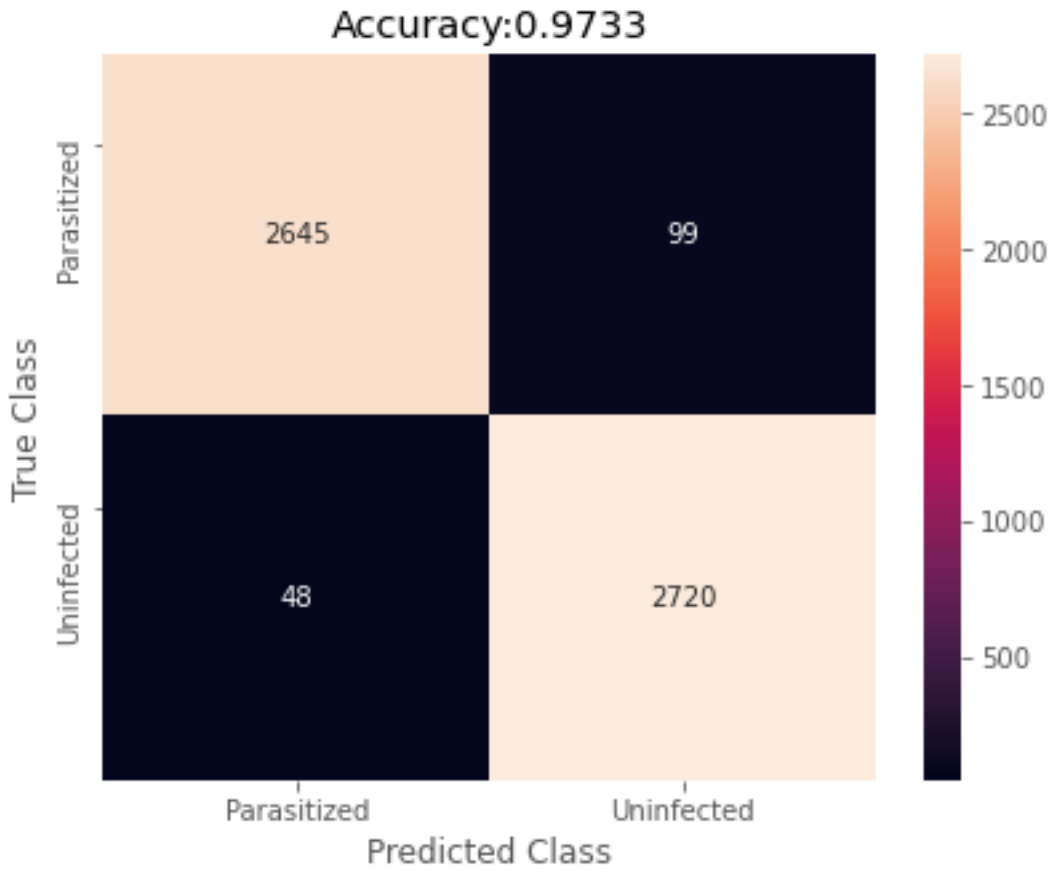
```
End of Fold 4 - Elapsed Time: 4:40:23.665173
************************************************
** Model: EfficientNetB0  fold: 5
Training…
Found 19841 validated image filenames.
Found 2205 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

```
batch time (batch time: 0.1466s vs `on_train_batch_end` time: 0.2433s). Check
your callbacks.
```




```
Predicting…
Accuracy 0.974150
              precision    recall  f1-score   support

 Parasitized   0.984259  0.963735  0.973889      1103
  Uninfected   0.964444  0.984574  0.974405      1102

   micro avg   0.974150  0.974150  0.974150      2205
   macro avg   0.974352  0.974154  0.974147      2205
weighted avg   0.974356  0.974150  0.974147      2205
 samples avg   0.974150  0.974150  0.974150      2205

Validating…
345/344 [==============================] - 81s 236ms/step
```

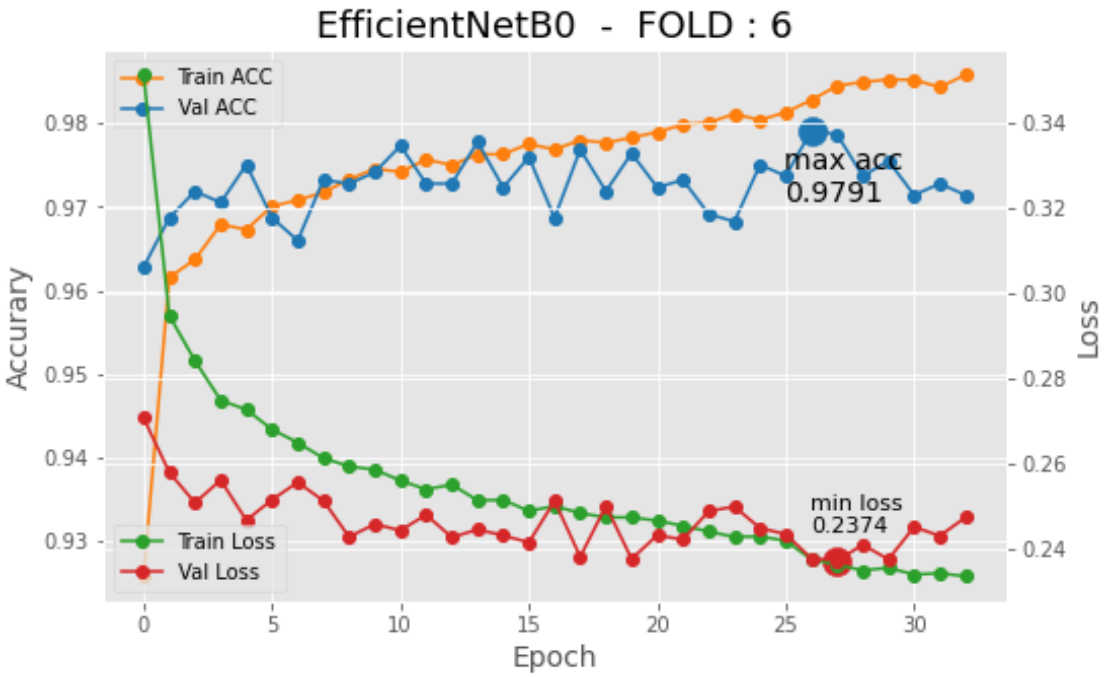

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996098)
ROC curve - Uninfected (area = 0.996098)



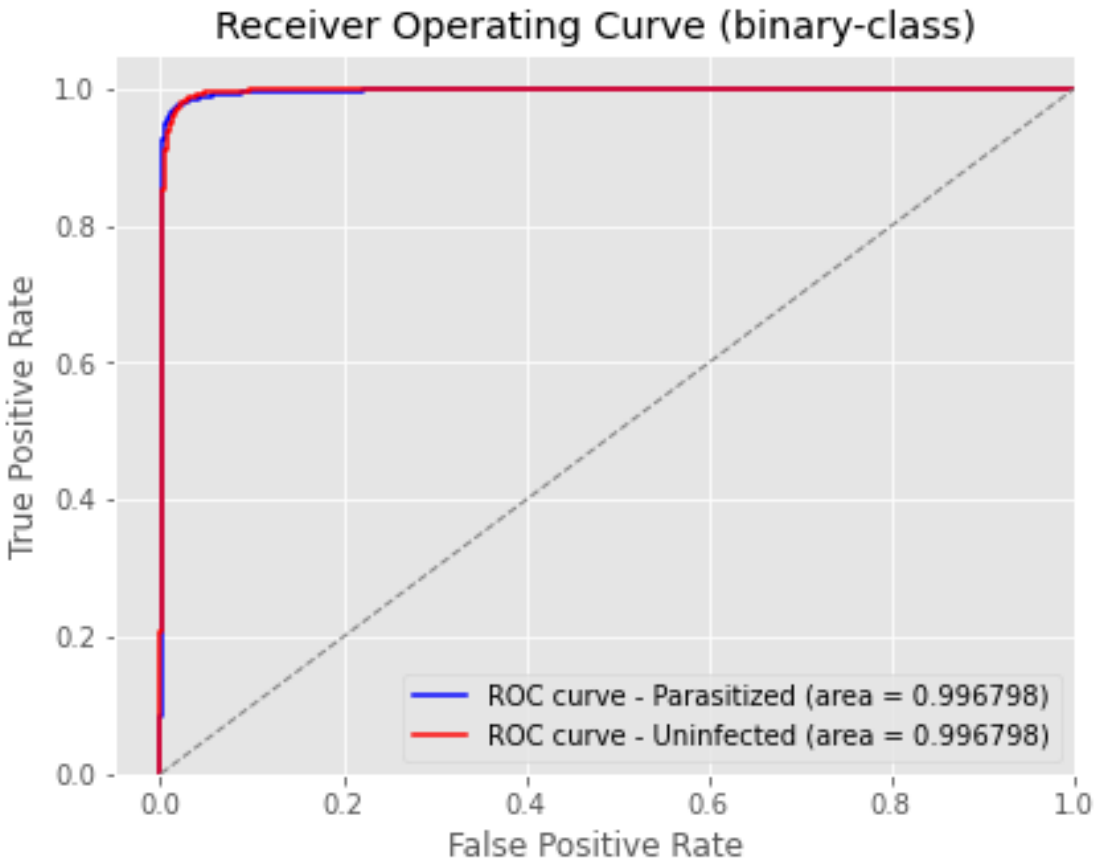
	precision	recall	f1-score	support
Parasitized	0.982176	0.963921	0.972963	2744
Uninfected	0.964881	0.982659	0.973689	2768
micro avg	0.973331	0.973331	0.973331	5512
macro avg	0.973529	0.973290	0.973326	5512
weighted avg	0.973491	0.973331	0.973328	5512
samples avg	0.973331	0.973331	0.973331	5512

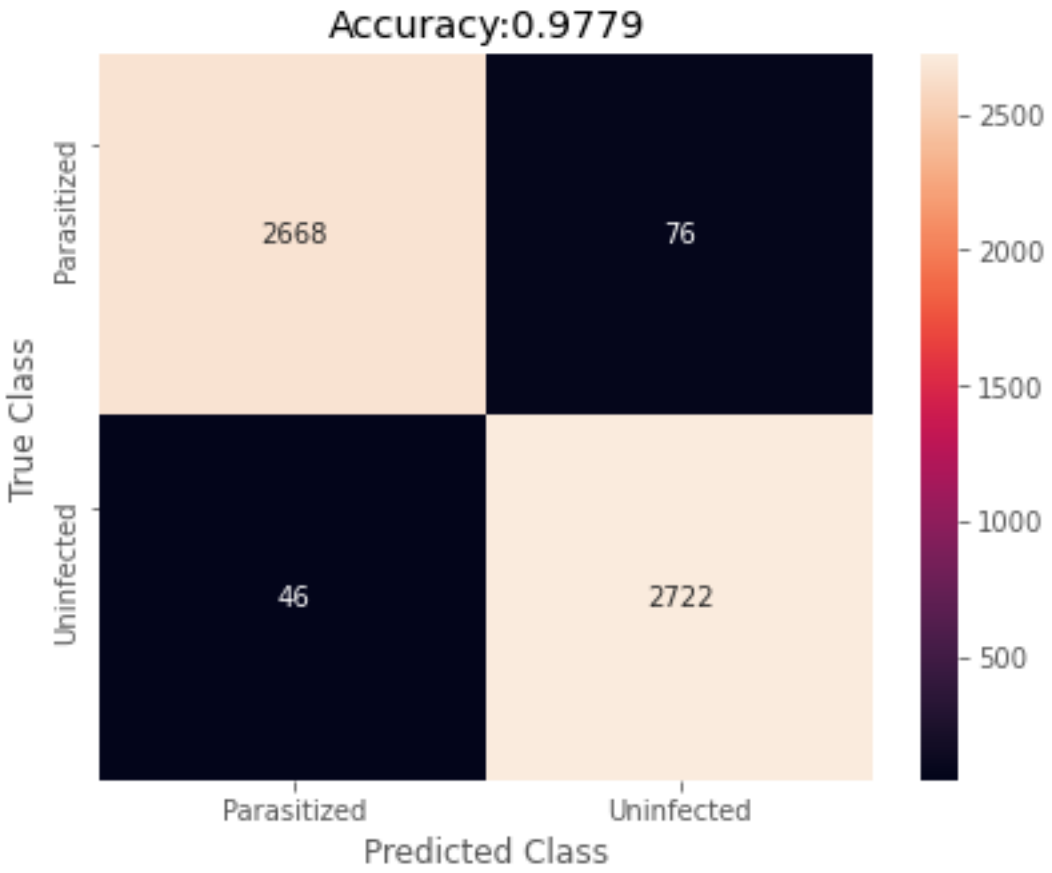
```
End of Fold 5 - Elapsed Time: 4:46:31.801410
************************************************
** Model: EfficientNetB0  fold: 6
Training…
Found 19842 validated image filenames.
Found 2204 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

```
batch time (batch time: 0.0988s vs `on_train_batch_end` time: 0.2463s). Check
your callbacks.
```




```
Predicting…
Accuracy 0.976407
              precision    recall  f1-score   support

 Parasitized   0.979909  0.972801  0.976342      1103
  Uninfected   0.972949  0.980018  0.976471      1101

   micro avg   0.976407  0.976407  0.976407      2204
   macro avg   0.976429  0.976410  0.976406      2204
weighted avg   0.976432  0.976407  0.976406      2204
 samples avg   0.976407  0.976407  0.976407      2204

Validating…
345/344 [==============================] - 79s 230ms/step
```

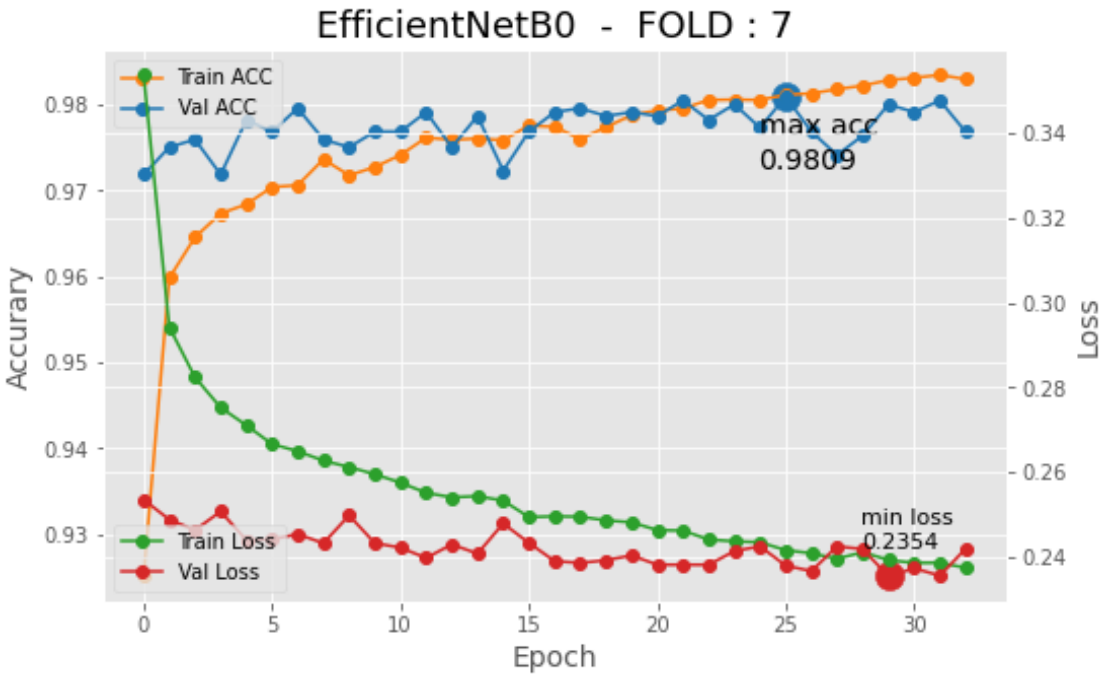

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996798)
ROC curve - Uninfected (area = 0.996798)



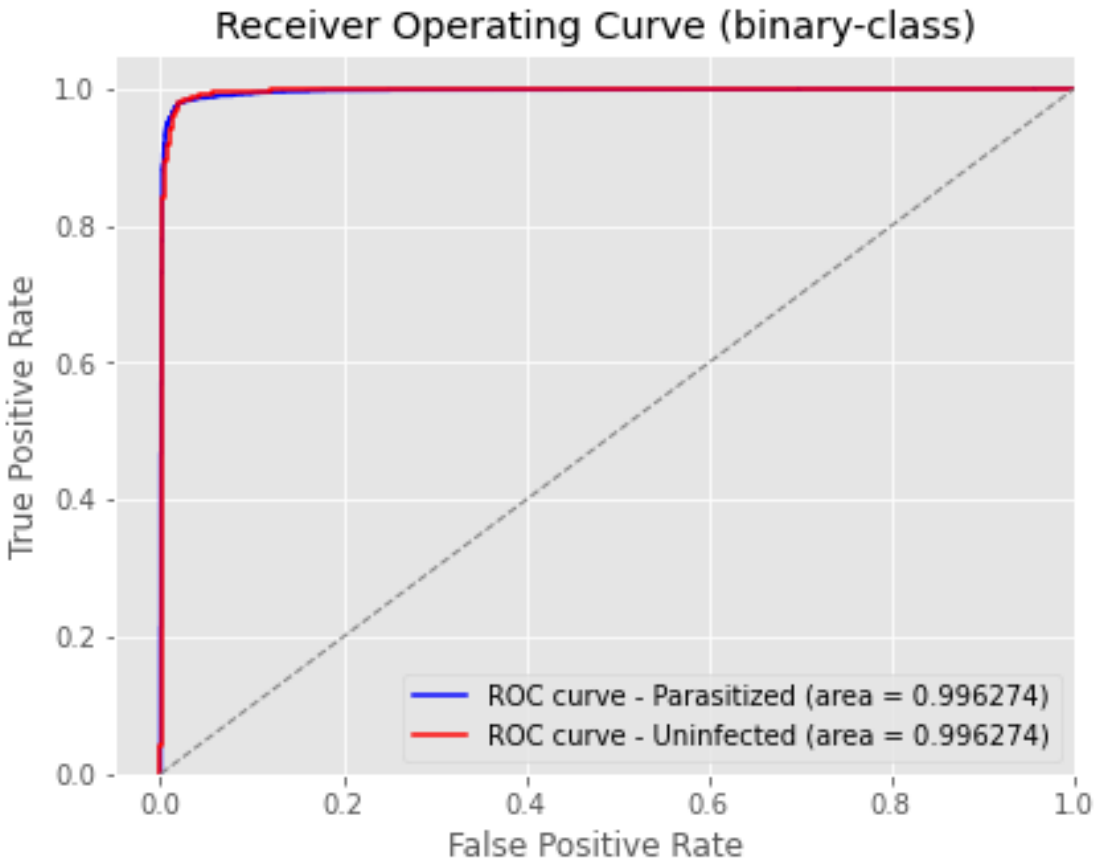
	precision	recall	f1-score	support
Parasitized	0.983051	0.972303	0.977647	2744
Uninfected	0.972838	0.983382	0.978081	2768
micro avg	0.977866	0.977866	0.977866	5512
macro avg	0.977944	0.977842	0.977864	5512
weighted avg	0.977922	0.977866	0.977865	5512
samples avg	0.977866	0.977866	0.977866	5512

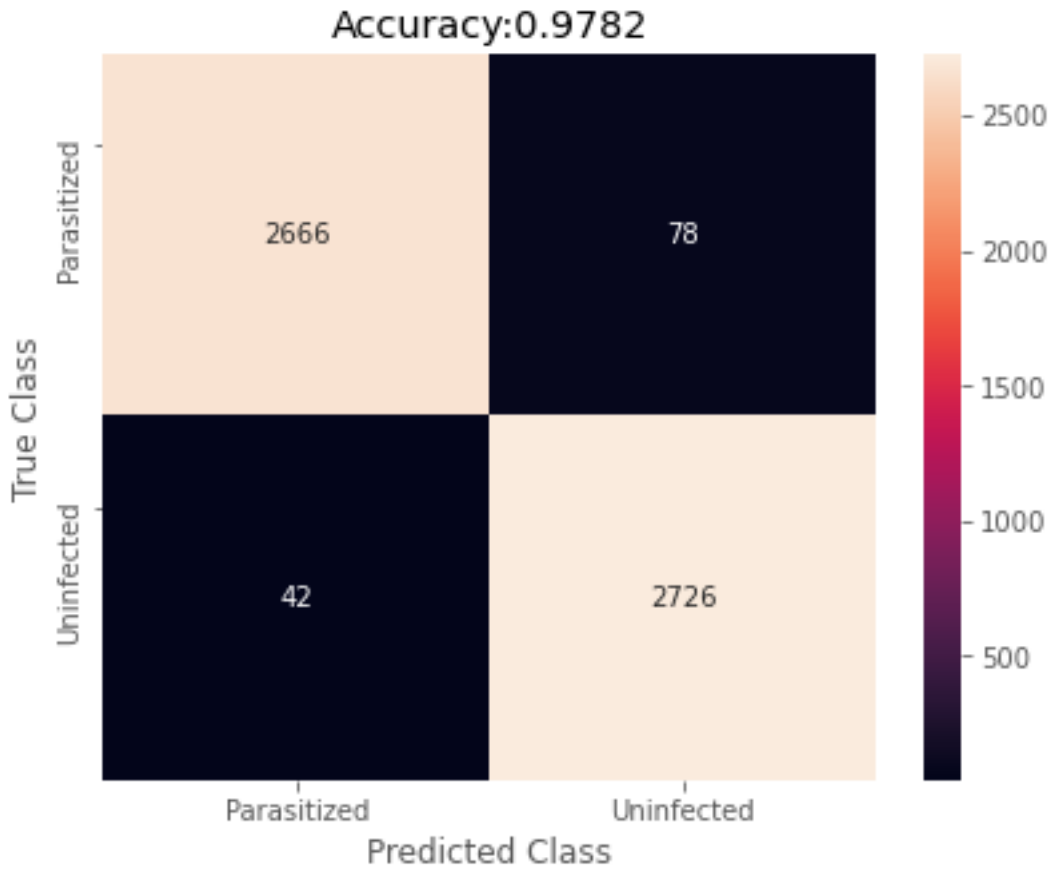
```
End of Fold 6 - Elapsed Time: 4:46:42.534757
************************************************
** Model: EfficientNetB0  fold: 7
Training…
Found 19842 validated image filenames.
Found 2204 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

batch time (batch time: 0.1037s vs `on_train_batch_end` time: 0.2443s). Check your callbacks.




```
Predicting…
Accuracy 0.983666
              precision    recall  f1-score   support

 Parasitized   0.985441  0.981868  0.983651      1103
  Uninfected   0.981900  0.985468  0.983681      1101

   micro avg   0.983666  0.983666  0.983666      2204
   macro avg   0.983671  0.983668  0.983666      2204
weighted avg   0.983672  0.983666  0.983666      2204
 samples avg   0.983666  0.983666  0.983666      2204

Validating…
345/344 [==============================] - 78s 228ms/step
```

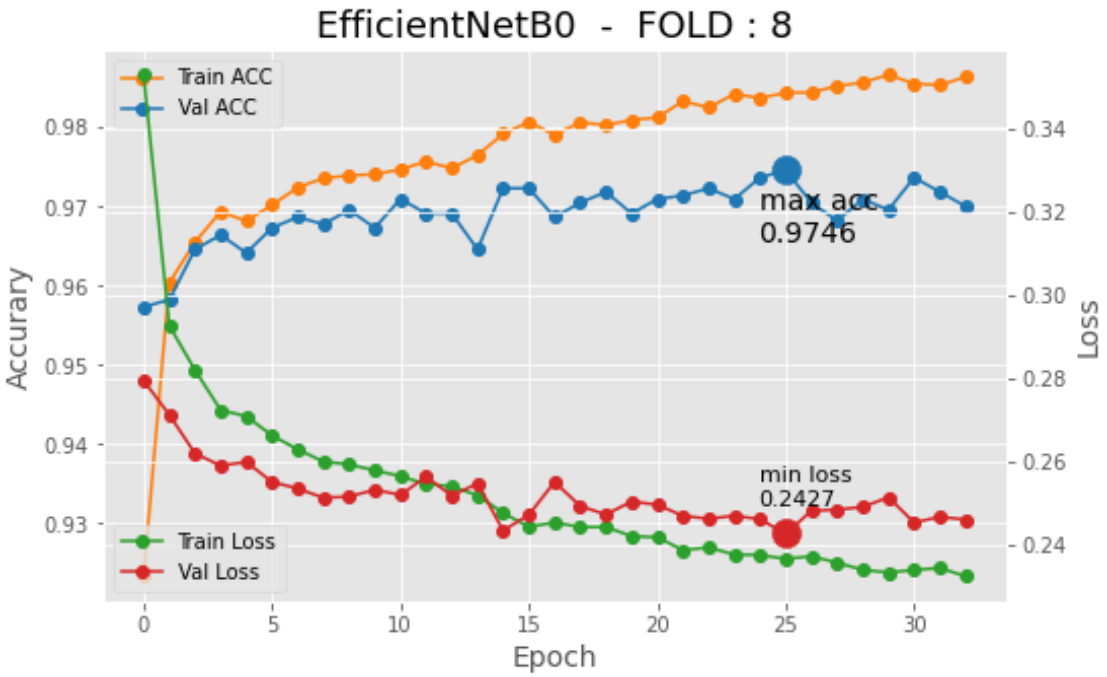

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996274)
ROC curve - Uninfected (area = 0.996274)



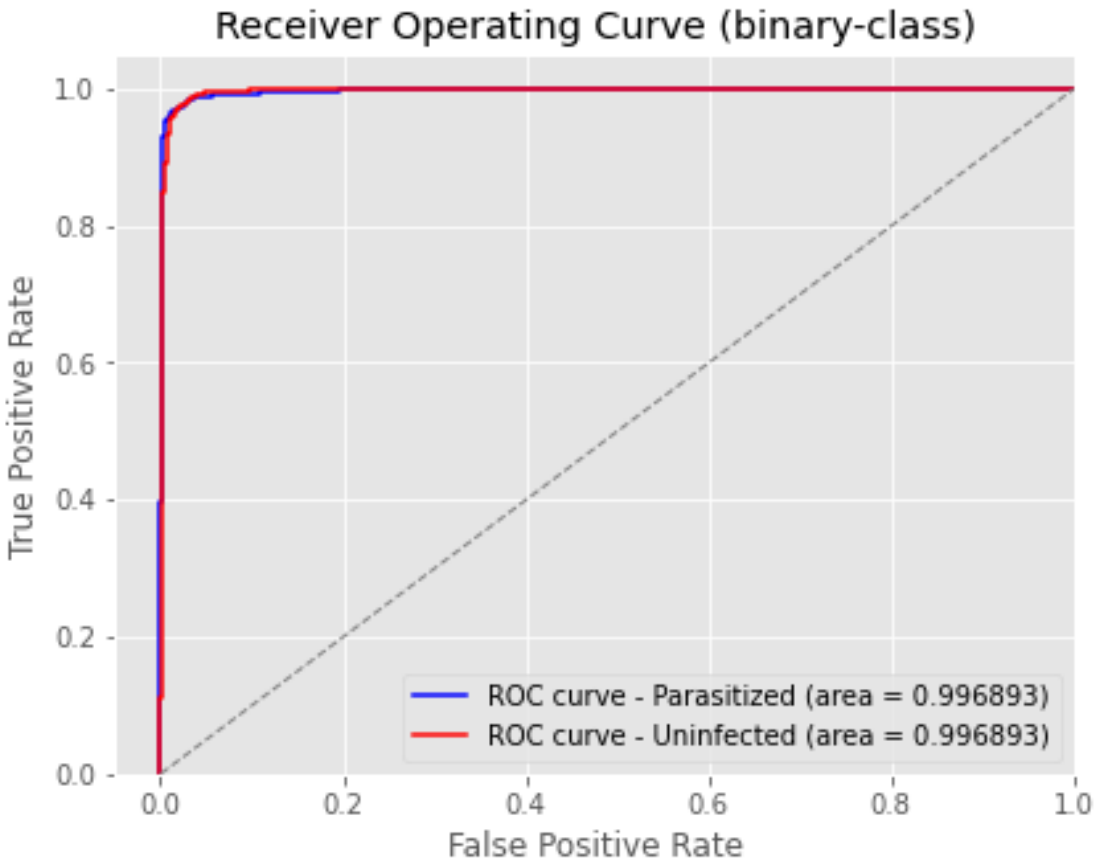
	precision	recall	f1-score	support
Parasitized	0.984490	0.971574	0.977990	2744
Uninfected	0.972183	0.984827	0.978464	2768
micro avg	0.978229	0.978229	0.978229	5512
macro avg	0.978336	0.978200	0.978227	5512
weighted avg	0.978310	0.978229	0.978228	5512
samples avg	0.978229	0.978229	0.978229	5512

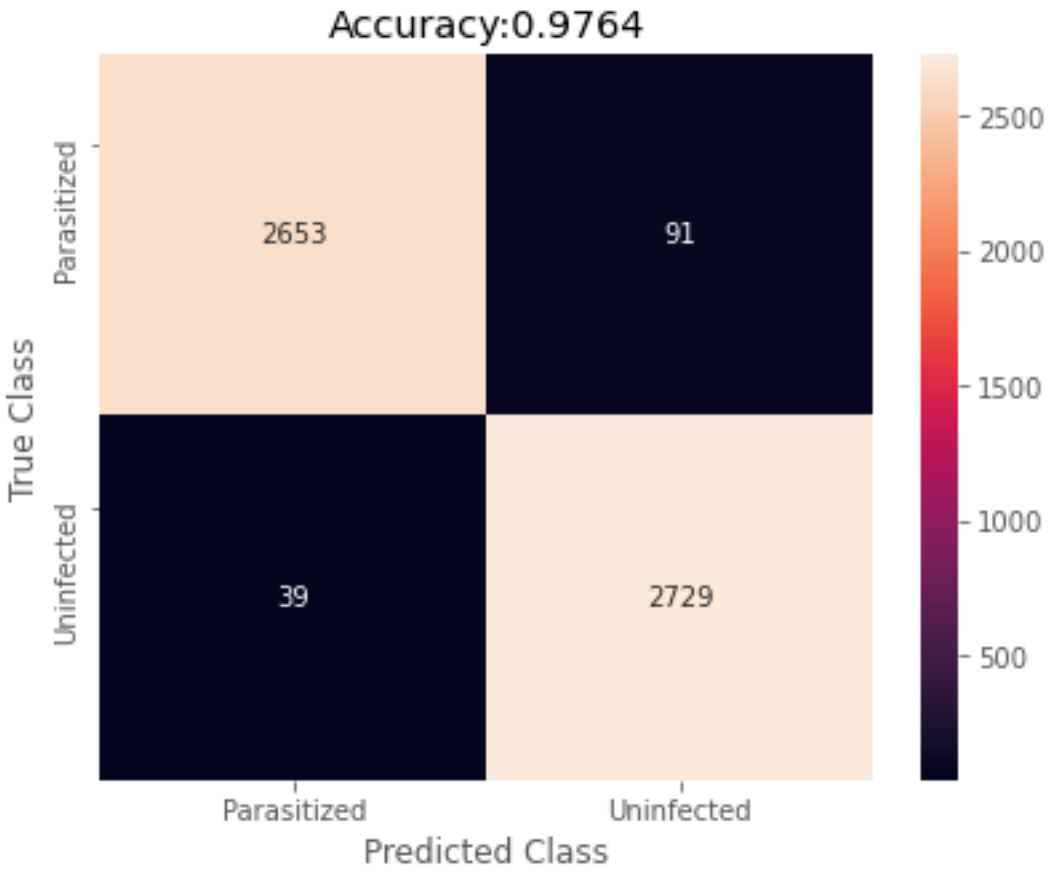
```
End of Fold 7 - Elapsed Time: 4:43:31.059613
************************************************
** Model: EfficientNetB0  fold: 8
Training…
Found 19842 validated image filenames.
Found 2204 validated image filenames.
Found 5512 validated image filenames.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
```

batch time (batch time: 0.0998s vs `on_train_batch_end` time: 0.2533s). Check your callbacks.



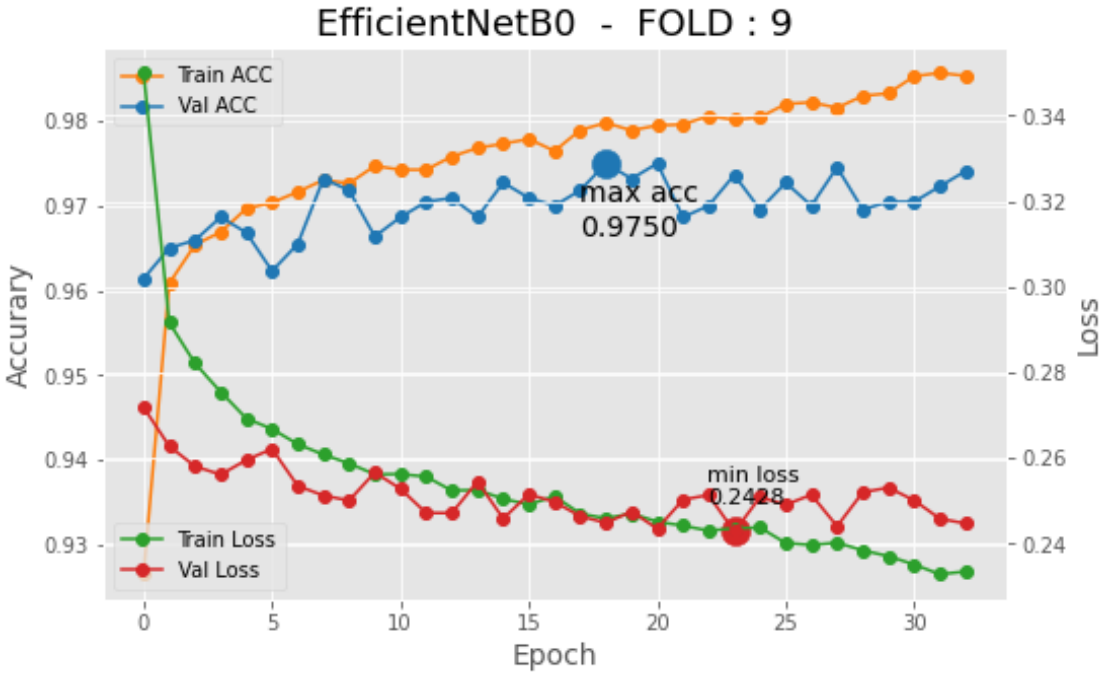

```
Predicting…
Accuracy 0.970508
              precision    recall  f1-score   support

 Parasitized   0.978782  0.961922  0.970279      1103
  Uninfected   0.962500  0.979110  0.970734      1101

   micro avg   0.970508  0.970508  0.970508      2204
   macro avg   0.970641  0.970516  0.970506      2204
weighted avg   0.970649  0.970508  0.970506      2204
 samples avg   0.970508  0.970508  0.970508      2204

Validating…
345/344 [==============================] - 84s 242ms/step
```


Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996893)
ROC curve - Uninfected (area = 0.996893)

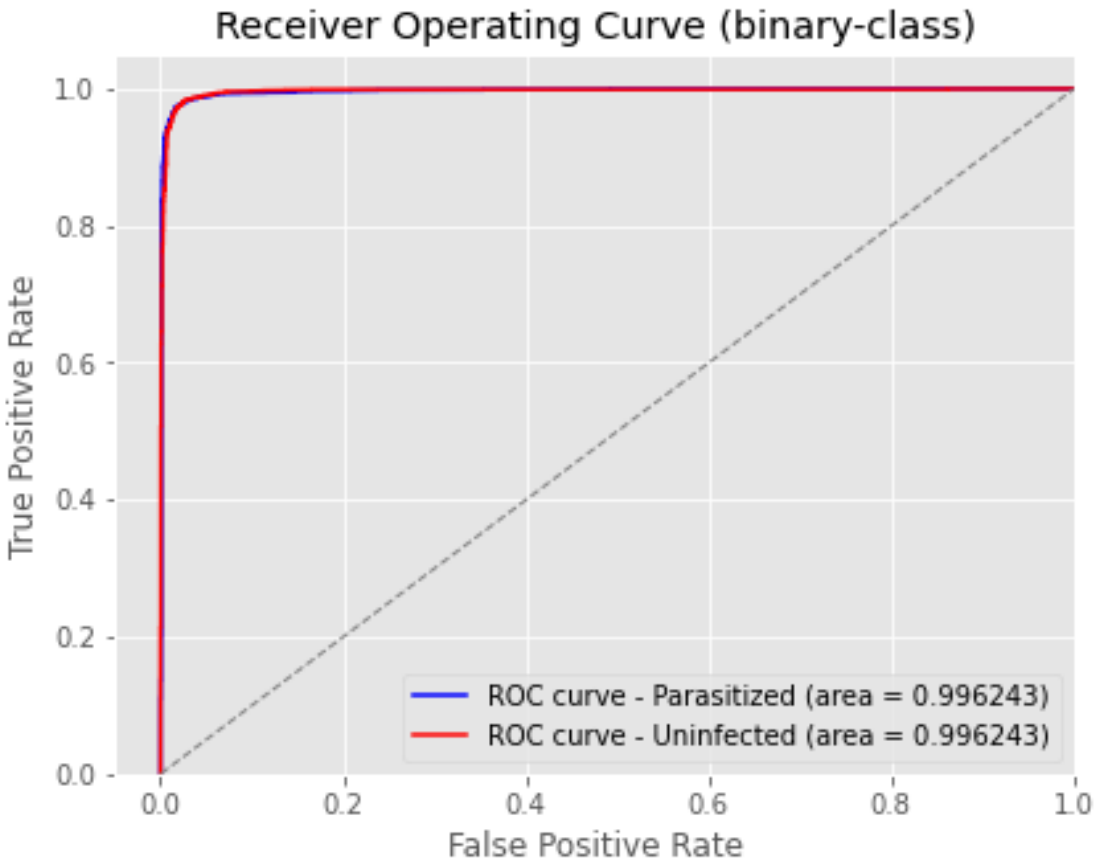


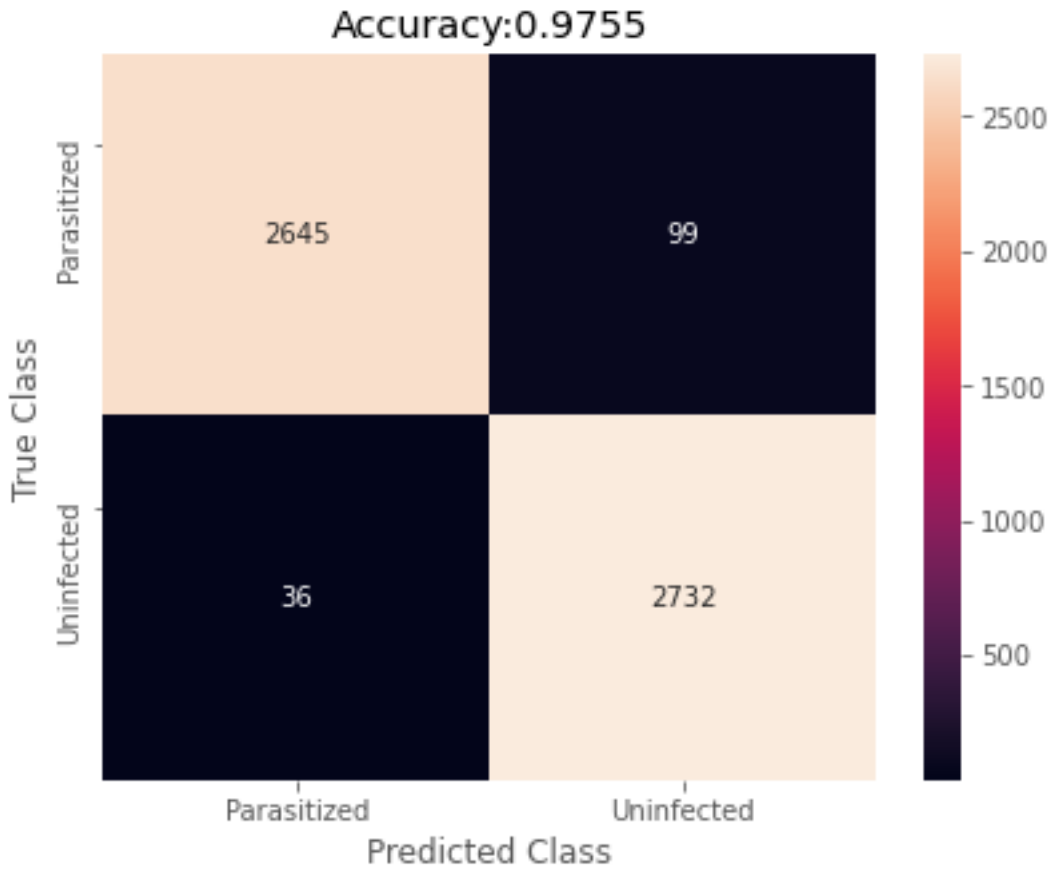
	precision	recall	f1-score	support
Parasitized	0.985513	0.966837	0.976085	2744
Uninfected	0.967730	0.985910	0.976736	2768
micro avg	0.976415	0.976415	0.976415	5512
macro avg	0.976622	0.976374	0.976411	5512
weighted avg	0.976583	0.976415	0.976412	5512
samples avg	0.976415	0.976415	0.976415	5512

```
End of Fold 8 - Elapsed Time: 4:48:50.469830
************************************************
** Model: EfficientNetB0  fold: 9
Training…
Found 19842 validated image filenames.
Found 2204 validated image filenames.
Found 5512 validated image filenames.
```




```
Predicting…
Accuracy 0.975045
              precision    recall  f1-score   support

 Parasitized   0.985185  0.964642  0.974805      1103
  Uninfected   0.965302  0.985468  0.975281      1101

   micro avg   0.975045  0.975045  0.975045      2204
   macro avg   0.975244  0.975055  0.975043      2204
weighted avg   0.975253  0.975045  0.975043      2204
 samples avg   0.975045  0.975045  0.975045      2204

Validating…
345/344 [==============================] - 86s 250ms/step
```

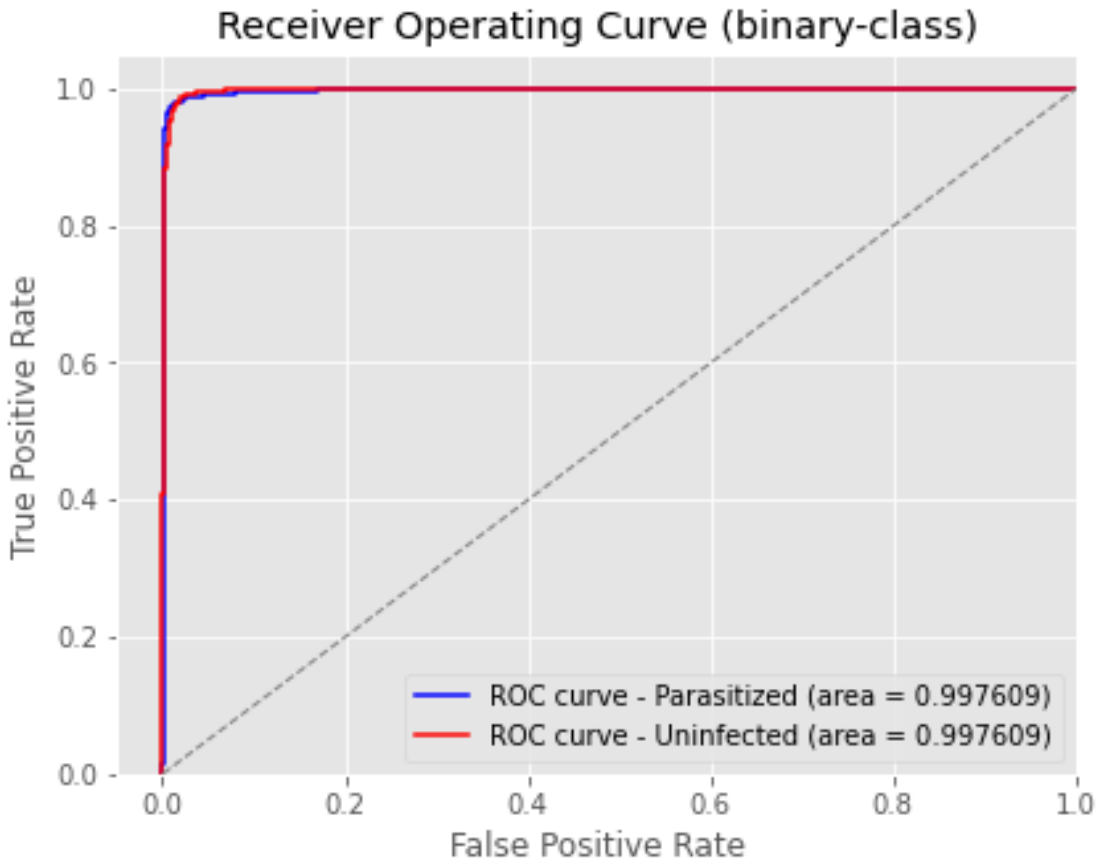

Receiver Operating Curve (binary-class)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve - Parasitized (area = 0.996243)
ROC curve - Uninfected (area = 0.996243)



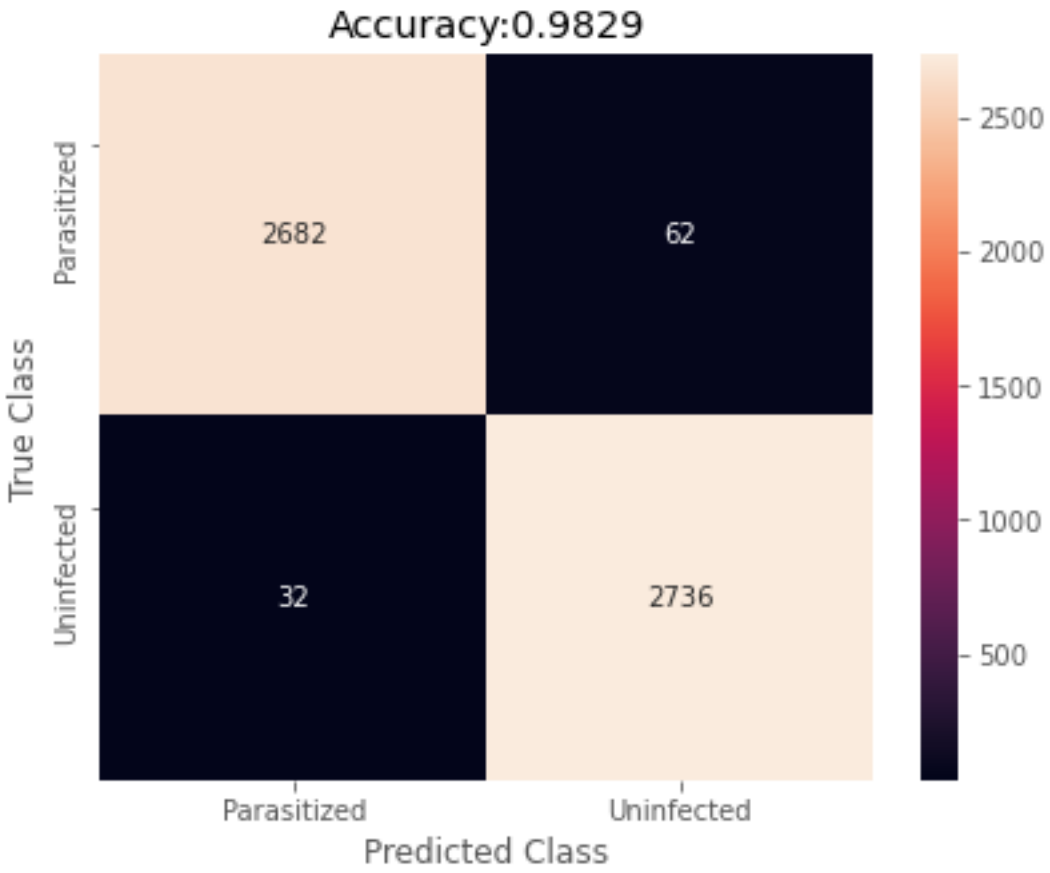
	precision	recall	f1-score	support
Parasitized	0.986572	0.963921	0.975115	2744
Uninfected	0.965030	0.986994	0.975889	2768
micro avg	0.975508	0.975508	0.975508	5512
macro avg	0.975801	0.975458	0.975502	5512
weighted avg	0.975754	0.975508	0.975504	5512
samples avg	0.975508	0.975508	0.975508	5512

End of Fold 9 - Elapsed Time: 4:58:10.694449

Validate emsembled results…




```
Accuracy 0.982946
```




```
f1 score: 0.982945
              precision    recall  f1-score   support

 Parasitized   0.988209  0.977405  0.982778      2744
  Uninfected   0.977841  0.988439  0.983112      2768

   micro avg   0.982946  0.982946  0.982946      5512
   macro avg   0.983025  0.982922  0.982945      5512
weighted avg   0.983003  0.982946  0.982945      5512
 samples avg   0.982946  0.982946  0.982946      5512


 END Model EfficientNetB0 - Elapsed Time: 1 day, 23:38:44.175101
```

2.5 Examine / Summarize Errors

Plot a sample of failed prediction. Both types:

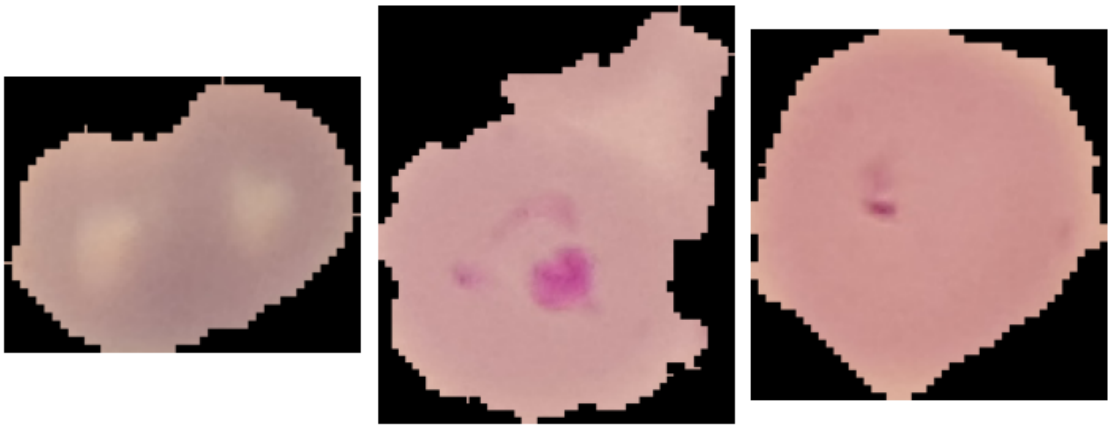
- Real: Parasitized -> Predicted: Uninfected

- Real: Uninfected -> Predicted: Parasitized

[ ]:
```
import random
import matplotlib.pyplot as plt

M = 3               # Minimum number of images to see of each error type

N = len(TIPOS)
for i in range(N):              # i: class number for y_true
    for j in range(N):          # j: class number for y_pred
        if i==j:   # Only errors
            continue
        indices = (val_true.values[:,i]==1) & (decission[:,j]==1)
        errores = sum(indices)
        if (errores < M):
            continue

        images = random.sample(list(df_val['file'][indices]), M)
        print('Real: {0} -> Predicted: {1}      NUM. ERRORS: {2}' \
              .format(TIPOS[i], TIPOS[j], errores))
        plt.figure(figsize=(10,10))
        for k in range(M):
            plt.subplot(1, M, k+1)
            img = plt.imread(DESTDIR+images[k])
            plt.imshow(img)
            plt.axis('off')
        plt.tight_layout()
        plt.show()
```

```
Real: Parasitized -> Predicted: Uninfected      NUM. ERRORS: 62
```

```
Real: Uninfected -> Predicted: Parasitized      NUM. ERRORS: 32
```



10 Anexo: Artículo enviado a publicación

An Ensemble Approach for Automated Medical Diagnosis of Malaria Through Efficientnet Architecture

Gonçalo Marques, Antonio Ferreras, Isabel de la Torre-Diez[1]
Department of Signal Theory and Communications, and Telematics Engineering
University of Valladolid, Paseo de Belén, 15, 47011, Valladolid, Spain

Abstract: Each year, more than 400,000 people die of malaria. Malaria is a mosquito-borne transmissible infection that affects humans and other animals. According to World Health Organization (WHO), 1.5 billion malaria cases and 7.6 million related deaths have been prevented from 2000 to 2019. Malaria is a disease that can be treated if early detected. We propose a support decision system for detecting malaria from microscopic peripheral blood cells images through convolutional neural networks (CNN). The proposed model is based on EfficientNetB0-based architecture. The results are validated with 10-fold stratified cross-validation. This paper presents the classification findings using images from malaria patients and normal patients. The classification accuracy is 98.29%. The proposed model is compared and outperforms the related work. The proposed ensemble method shows a recall value of 98.82%, a precision value of 97.74%, an F1-score of 98.28% and a ROC value of 99.76%. This work suggests that EfficientNet is a reliable architecture for automatic medical diagnostics of malaria.

1 Introduction

Malaria is a mosquito-borne transmissible infection that affects humans and other animals. According to World Health Organization (WHO) [1], 1.5 billion malaria cases and 7.6 million related deaths have been prevented from 2000 to 2019. Moreover, 229 million new malaria cases have been recorder and 409.000 people passed away in 2019. Approximate two-thirds of fatalities are among children under the age of five and most of them occurred in sub-Saharan Africa. Moreover, around 12 billion dollars a year due to its impact on the services, industry, and tourism in Africa. Initial symptoms of malaria, such as chills, fever, headache or vomiting, can potentially be moderate and difficult to identify. However, without treatment malaria could produce severe sickness and cause death [2]. Therefore, early detection is the best manner to treat this disease.

Microscopic thick/thin-film blood smear examination conducted by qualified professionals is a reliable method for malaria diagnosis [3], [4]. This is a high time spending process where experts usually need to manually identify at least 5000 cells to verify the condition. Rapid detection tests are available and largely used. However, these tests are expensive and prone to errors. Moreover, manual diagnosis has other limitations which severely impacts the diagnostic accuracy. Large-scale screening or variability, especially in countries where the illness is widespread with limited resource settings [5]. Consequently, automatic

[1] *e-mail addresses:* goncalosantosmarques@gmail.com (**G. Marques**), antonio.ferrerasextremo@gmail.com (**A. Ferreras**), isator@tel.uva.es (**I. de la Torre**).

pattern recognition tools supported by Big Data and machine learning have been employed to malaria blood smears since 2005 [6].

Convolutional Neural Networks (CNNs) are a type of deep neural networks used for different applications [7] [8] [9]. CNNs usually have one input layer, several hidden layers, and one output layer [10]. The inner hidden layers normally are made of fully connected layers, convolutional layers, and activation layers, as ReLu [11] [12]. CNNs improved automatic image classification methods as pre-processing the images is not required unlike other conventional machine learning algorithms [13] [14] [15].

Currently, ensemble models are an integral part of the state of the art [16]. Ensemble Learning refers to the techniques used to generate new models form other several trained models, combining their outputs such as a "committee" of decision-makers. The assumption is that the group decision, with singled predictions combined properly, should have an improved final accuracy over any individual committee participant, on average [17]. Several theoretical and empirical experiments suggest that ensemble models achieve better accuracy than the models that they integrate. Ensemble models have been applied in different areas of health diagnosis such as diabetes type-II [18], women thyroid prediction [19], neurological disorders [20] and cancer prediction [21]. Therefore, we propose an ensemble approach composed of the 10 partial models obtained in the 10-fold validation process of an EfficientNetB0 base model. This approach has presented relevant findings in previous research studies [22] [23] [24] [25].

This study aims to recommend a computer-method for decision support system to detect malaria in blood smear images using EfficientNet [26] [27]. According to our analysis, no similar study proposes EfficientNet for automated diagnosis of malaria. Numerous experiments for malaria diagnosis have been described in the literature. It is critical to reveal all the procedures and tools to enable the readers to reproduce the results presented. Therefore, we present all the materials and Python code created with the Jupyter notebook. This work uses open-source technologies and by providing the supplementary files the readers can reproduce the experiments for additional investigation. EfficientNet can be utilized for transfer learning with better [28] than others CNNs such as VGG16 or VGG19 [29], ResNet50 [30] or InceptionV3 [31]. EfficientNet architecture is made of eight models (B0 to B7), higher version number higher parameters and complexity. Moreover, EfficientNet models use transfer learning techniques to improve execution time and save computational power. Therefore, it usually gives better accuracy than other models concerning a smart scaling of resolution, width, and depth. This work uses the B0 model, with the minimum number of parameters. This model is selected since presents promising results and is feasible for our experimental setup. The number of parameters of B1-B7 models increases largely [27]. Moreover, this study used separated datasets have been used to validate the CNN model and to guarantee the non-existence of overfitting. The validation dataset includes samples that were not used during training and testing. The authors have evaluated the algorithm using stratified 10-fold cross-validation. Finally, an ensemble model has been composed of ten intermediate models generated during the 10-fold stratified validation. This approach significantly increases the accuracy of any of the partial models. The source code is supplied as supplementary file A.

2 Related work

Numerous researcher activities related to optimization methods, as well as classification or clustering methods. According to Pootschi et al. [32] almost every classification method has been used for malaria diagnosis with thin blood smear samples, ranging from the unsupervised K-Mean Clustering [33] to other supervised techniques, as Naïve Bayes Tree [34], Ada-boost [35], Decision Tree [36], Support Vector Machine [37] or Linear Discriminant [38]. However, most of them are devoted to the study of the interaction between classification with features and segmentation, and few of them investigate explicitly parasite detection in blood cell images.

Comparing the performance of all different studies have limitations. On the one hand, most of the studies do not use the same image dataset set and number of samples. Nevertheless, a trade-off between processing time and accuracy is observed, the longer the run-time, the better the accuracy. Moreover, the algorithm architecture also influences the runtime of the process [32].

Deep learning algorithms have recently been used to increase performance in several healthcare domains. Liang et al. [39] were one of the first researchers who apply CNN to malaria diagnosis. They used a more classical transfer model and a custom CNN model to automatically classify thin blood smear image cells, obtained by traditional optical microscope slides. Their results showed that their custom CNN model obtained a better performance (97.37% accuracy) over the more "traditional" transfer models.

Rajaraman et al. [40] evaluated several pre-trained CNN based deep learning algorithms as feature extractors toward classification. Up to 6 different architectures were used (AlexNet, VGG16, ResNet50, Xception, DenseNet121 and custom models) with accuracy results ranging from 91.50% to 95.9%. VGG16 and ResNet outperformed their competitors. They concluded that pre-trained CNNs are a promising tool for feature extraction.

Rahman et al. [41] used the dataset from the National Institute of Health, and a 5-fold cross-validation scheme, to test their models. A custom CNN (96,29% accuracy), a VGG16 (97.77%) and a CNN extracted features applied to SVM (94.77%). They concluded that the use of different pre-processing practices such as normalization or standardization does not impact the final performance. However, data augmentation procedures used on the training images reveals encouraging results.

Shah et al. [42] developed an image classification algorithm based on CNNs and tested it with a labelled image dataset. Their findings show 94.77% accuracy. They state relevant limitations related to computationally resources and suggest that the proposed results can be improved with higher computing power.

Quan et al. [43] worked on a novel and lightweight model based on CNNs, combining concepts from dense and residual networks and employing attention mechanisms. The proposed method was entitled Attentive Dense Circular Net (ACDN). The findings have been compared with the related work in the literature as DenseNet121 [44] or DPN92 [45]. The results show higher performance with 97.47% versus 90.94% and 87.88% reported by DenseNet121 and DPN92, respectively. Moreover, the proposed ACDN model presents high precision and a fast convergence speed.

Finally, Yang et al. [46] developed an interesting model, which uses thick blood smear images and was developed to run on smartphones. Their model involves fast screening of the images to detect parasite candidates. On the one hand, they classify the set of images publicly available by the community with a

customized CNNs and obtained a 97.26% accuracy. On the other hand, they compared their model with others in the literature, also applying the previous fast screening step. The proposed model outperformed the accuracy reported by other architectures such as ResNet50 (accuracy 93.88%), VGG19 (accuracy: 93.72%) and AlexNet (accuracy: 96.33%).

3 Methods and materials

The authors aim to clearly present all methods and materials used in this work. Section 3.1 describes the Malaria image datasets. The proposed CNN algorithm is described in Section 3.2. Ultimately, Section 3.3 introduces the experimental setup and validation procedure.

3.1 Malaria Dataset

The images used to test the proposed approach have been obtained from a open dataset offered by the USA National Institutes of Health (NIH). The red blood cell micrographs in the dataset were gathered from Giemsa-stained thin blood smear slides. In total 150 different malaria-infected and 50 healthy patients treated at Chittagong Medical College Hospital are included [47]. Every image was manually labelled, de-identified and archived by a professional in the Mahidol Oxford Tropical Medicine Research Unit in Bangkok [40]. The database includes 27,558 red blood cell images, half infected (labelled as positive samples) and half clean (negative samples). The parasitized images consist of red blood cells affected by plasmodium. However, the uninfected ones can include several noise factors such as stain interferences or dust impurities. Samples of infected cell images can be shown in Fig. 1, whereas Fig. 2 shows three normal cell images.

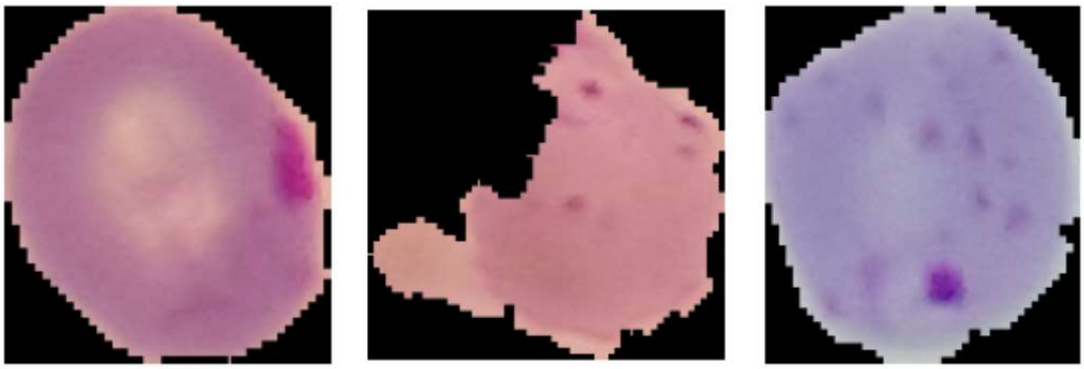


Fig. 1. Three infected cells (positive samples) randomly selected from the dataset.

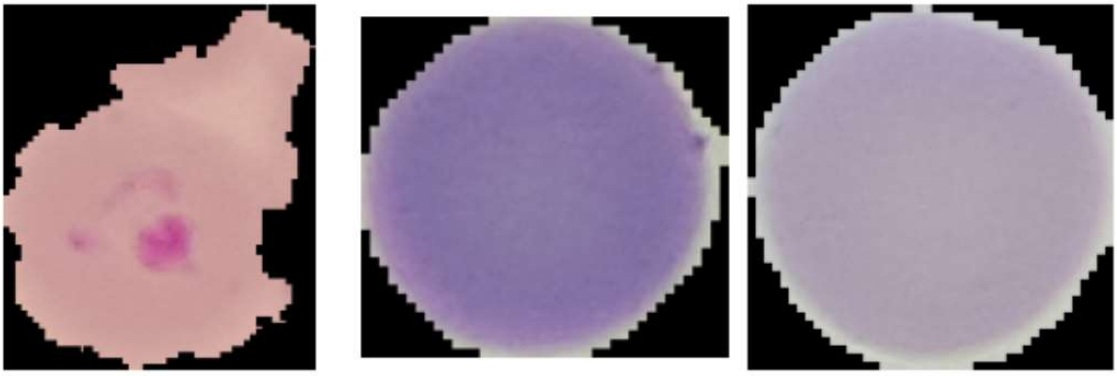


Fig. 2. Three normal cells (negative samples) randomly selected from the dataset

We use an equal number of samples for both categories to correctly validate the system performance as suggested by [48]. In total, 22,046 samples have been used for "malaria" and "normal" classes (Table 1). Those images were utilized in the stratified cross-validation process. Moreover, the algorithm has been validated using a separate dataset. This dataset has 5,512 samples and contains 2,756 images for "normal" class and 2,756 images for "malaria" class. The later datasets were not employed neither in the training or in testing phase. This process was carried to guarantee the prove the non-existence of overfitting.

Table 1. Dataset information.

Type	Training/testing images	Validation images
Normal	11,023	2,756
Malaria	11,023	2,756

3.2 Proposed CNN

EfficientNetB0 model has been used, with a previous transfer learning process. To reduce overfitting by lowering the quantity of parameters a global_average_pooling2d layer was added. On top of that model, 3 inner dense layers with dropout layers and ReLu activation functions were added in sequence. To avoid overfitting, a 30% random dropout rate has been implemented. One final output dense layer with to output units were added for binary classification; in this case, the activation function is a softmax that provides the final computer-aided system. The order of the layers, the number of trainable and non-trainable parameters (weights) in each layer, and the output shape of each layer are shown in Table 2. The proposed model have 4,223,934 parameters.

Table 2. Layer types, output shape and parameters of the model.

Layer (Type)	Output shape	Param #
EfficientNetB0 (Model)	7 x 7 x 1280	4,049,564
global_average_pooling2d	1280	0
dense (Dense)	128	163,968
dropout (Dropout)	128	0
dense_1 (Dense)	64	8,256
dropout_1 (Dropout)	64	0
Dense_2 (Dense)	32	2,080
Dropout_2	32	0
Dense_3 (Dense)	2	66
Total Parameters:	4,223,934	
Trainable Parameters:	4,181,918	
Non-trainable Parameters:	42,016	

We use open-source libraries and software in this experiment are. The readers can use the Google Colaboratory platform, selecting the GPU running environment to reproduce the findings. This platform can be used without cost since Google provides it for research purposes. The hardware is a Tesla K80 GPU of 12 GB. The EfficientNet architectures are scaled and pre-trained CNNs, that are used for image classifications applications by means of transfer learning. That model was developed by Google AI in 2019 and is ready for use from the GitHub platform [49]. Google AI developed the Albumentations library is also and is accessible from GitHub repositories [50]. The proposed model was successfully used to diagnosis of COVID-19 with X-Ray images [51].

The model uses three main different libraries such as EfficientNet as the base module, ImageDataAugmentator and the Albumentations. On the one hand, the EfficientNet algorithms are built with highly effective but simple compound scaling techniques. The method allows to scale up from baseline convolutional network to any required resource limitations whilst retaining model performance, acquired from transfer learning from external datasets. EfficientNet networks reach both better accuracy and superior efficiency over other CNNs such as MobileNetV2, GoogleNet, AlexNet, and ImageNet [27]. The current experiment implemented EfficientNetB0 than includes 4,049,564 parameters, as it is appropriate given the resources available and our objective. On the other hand, the Albumentations software is broadly utilized in engineering, deep learning investigation, artificial intelligence contests, and opensource developments. The library provides several image transformations well-optimized for execution. This software includes an augmentation methods for computer vision, comprising object detection, segmentation and classification. This study has employed the Compose function of the Albumentations package. It has been shown that Albumentations decreases overfitting, enhance the performance of classifiers and then reduce running time as suggested in [52]. Augmentation is implemented in each fold, the model accuracy of the model increases, and execution time decreased.

ImageDataAugmentator library is an image data generator for Keras, a Python interface for artificial neural networks, backing the utilization of advanced augmentation libraries (imgaug and albumentations) [53]. The library configures the Image data generator in accordance with the albumentations library to reduce

execution time. A data generator is implemented by means of the constructor method of the ImageDataAugmentator class. Two parameters are needed, the first one is rescale, set as 1/255, used to convert each pixel value from a [0, 255] scale to [0, 1]. The second one is the augment parameter, which is designed to be used as the output of the compose method of the Albumentation software. Data generator has used further to process the image datasets. Fig. 3 shows an overview of the experiments conducted.

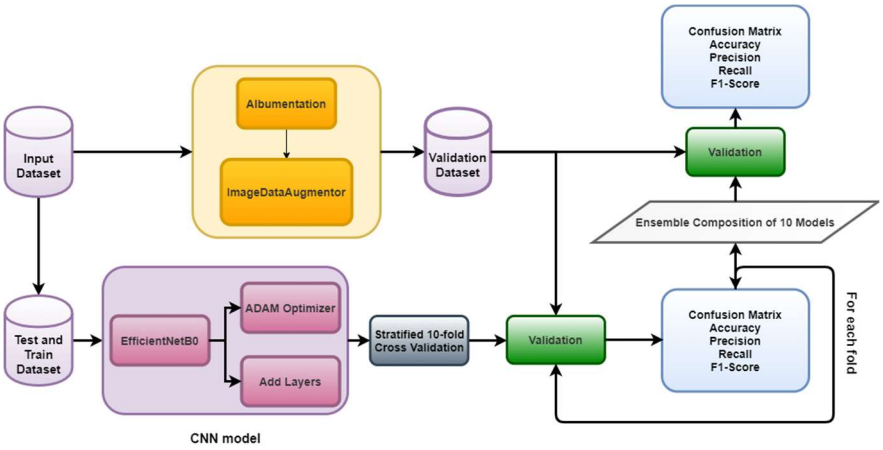


Fig. 3. Overview of the experiments conducted.

3.3 Validation Process

Two different phases were carried out to validate the model. First, the 10-fold cross-validation technique used the same dataset of images for training and testing; and second, a different dataset which includes images that were not used during the previous phase has been used to validate the model performance. Then the confusion matrix has been calculated. Precision, recall and F1-score have been computed for each one of both classes. At last, the averaged values for each fold were estimated. Algorithm 1 shows the practical setup utilised to carry out the experiment.

Algorithm 1: Experimental Setup

Environment	*1.*	*Select Google Colab or local environment and install the required libraries.*
Input	*2.*	*Download Images of two categories.*
Configuration	*3.*	*Import the images.*
Directories Configuration	*4.*	*Create two directories of the images with their labels according to classes.*
	5.	*Configure training, testing and validate the model using stratified 10-fold cross-validation.*
Display random images	*6.*	*Display some image samples of each type*
Data Generator Configuration	*7.*	*Define augmentation pipeline using the Compose function of albumentation library.*
	8.	*Create the data generator as an object of the ImageDataAugmentator library and configure the augmentation pipeline obtained in (7).*
Training and Testing	*9.*	*Create the model using EfficientNetB0 and dense layers with ReLu activation function and an output layer with a softmax activation function.*
Apply 10-fold stratified cross-validation	*10.*	*Compile the model using the ADAM optimizer with a learning rate of 0.0001. Categorical_Crossentropy function for loss calculation.*
	11.	*Model fitting using 33 epochs and ReduceLROnPlateau function to reduce the learning rate when the metrics stop improving.*
	12.	*Save the model to be used for validation testing.*
	13.	*Configure testing dataset.*
	14.	*Generate performance score values for each fold.*
		a. Model loss graph.
		b. Model Accuracy graph.
		c. Test Classification Report.
		d. Validate Model.
		e. AUC-ROC curve.
		f. Confusion Matrix.
		g. Validation Classification Report
Ensembled model	*15.*	*Generate performance score values for the ensembled model.*
		a. Ensembled Classification Report.
		b. AUC-ROC curve.
		c. Confusion Matrix.
Examine Errors	*16.*	*Locate misclassified images.*
	17.	*Plot some samples of each type*

4 Results

The Python code was executed on a laptop notebook DELL XPS 15 9560, Intel Core i7, 16GB RAM, equipped with an NVIDIA GeForce GTX 1050 video card, GPU support. The training phase of the customized EfficientNetB0 model was carried out employing a stratified 10-fold cross-validation technique. Altogether, each fold run 33 epochs. Furthermore, each fold consisted of 1,240 steps. 16 was the value chosen for the mini-batch parameter.

Model training required a total of 409,200 iterations. The time elapsed for the training model was 1 day, 23:38:44. The initial learning rate was set to 0.0001. The model used a ReduceLROnPlateau technique

since a reduction in the learning rate is required as soon as the improvement does not increase any longer. The *callback* function checks the enhancement, and if no progress is confirmed for a 'PATIENCE' value of epochs, the learning rate is lowered. PATIENCE level is set to 6, and min_lr=0.000001 as the minimum learning rate before definitely stopping. ADAM optimization [54] was chosen as the solver method. In the supplementary files, the detailed data and the training and validation graphs for each fold of the proposed model are provided. Moreover, loss, confusion matrix, and area under the curve of receiver operating characteristics are also included as supplementary material. Each of the 10-fold models is trained, the models are used for the validation testing, using separated datasets. The performance reported in the experiments is encouraging. Fig. 4 shows the history graph of the 8th integration of the 10-fold cross-validation process. The usual learning pattern is clearly seen in the evolution of the graphs, both accuracy and losses. The learning process stops when it does not improve any longer. Detailed information, including the software scripts, relating to the experiments can be seen as supplementary material.

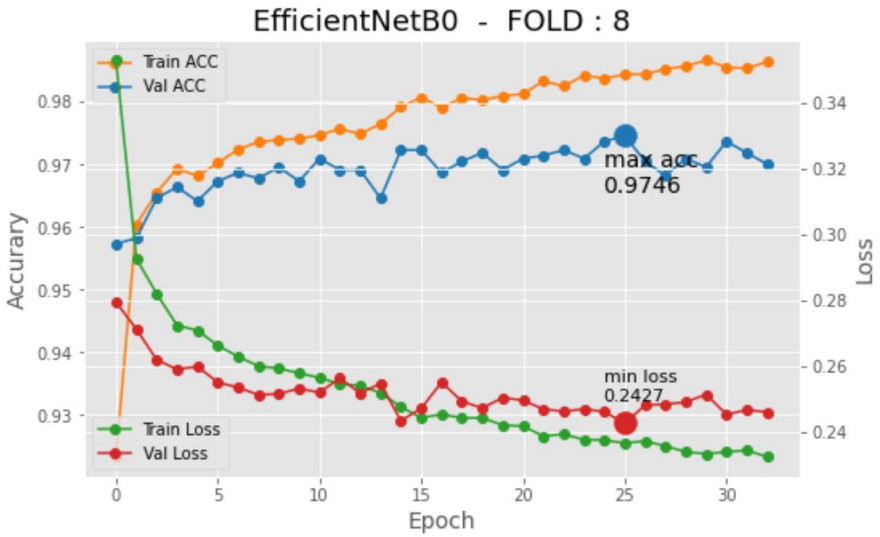


Fig. 4. History graph (model loss and accuracy) for an iteration of the K-fold algorithm

4.1 Experimental results

Overall, training data included 22,046 different samples and 5,512 samples have been used for testing. The findings are given for each one of the 10 folds, together with the average value. The precision, recall, F1-score and accuracy are provided for each class and for the average between classes.

Table 3 shows results related to the "malaria" class. Minimal performance values are obtained for the 0, 1, and 4-fold. The minimum precision values of 97.72% occurred in the 1-fold. Moreover, the minimum recall value is 96.19% in 8-fold. The minimum F1-score is 97.02% also at 8-fold. The average precision, recall, and F1-score are 98.07%, 97.05% and 97.55%, in that order.

Table 3. Binary classification for "malaria" class.

Fold	***Precision***	***Recall***	***F1-score***
0	0.977231	0.971920	0.974569
1	0.978221	0.976449	0.977335
2	0.981702	0.971920	0.976787
3	0.978102	0.971014	0.974545
4	0.978042	0.968297	0.973145
5	0.984259	0.963735	0.973889
6	0.979909	0.972801	0.976342
7	**0.985441**	**0.981868**	**0.983651**
8	0.978782	0.961922	0.970279
9	0.985185	0.964642	0.974805
Average	*0.980687*	*0.970457*	*0.975535*

The results regarding the normal class are shown in Table 4. The minimum recall value of 97.72% is reported in the 0-fold. The lowest precision value is 96,25% for 9-fold. Finally, 4-fold presents a minimum F1-score value of 97.33%. Average precision, recall, and F1-score values are 97.07%, 98.08% and 97.66%, respectively.

Table 4. Binary classification for normal class.

Fold	***Precision***	***Recall***	***F1-score***
0	0.971996	0.977293	0.974638
1	**0.976428**	0.978202	0.977314
2	0.972122	0.981835	0.976954
3	0.971145	0.978202	0.974661
4	0.968525	0.978202	0.973339
5	0.964444	0.984574	0.974405
6	0.972949	0.980018	0.976471
7	0.981900	0.985468	**0.983681**
8	0.962500	0.979110	0.970734
9	0.965302	**0.985468**	0.975281
Average	*0.970731*	*0.980837*	*0.976648*

Accuracy, precision, recall and F1-score values between classes are presented in Table 5. The calculated average accuracy is 97.56%. Furthermore, a precision value of 97.57%, a recall value of 97.56% and an F1-score value of 97.56% are reported.

Table 5. Binary classification average between classes.

Fold	***Accuracy***	***Precision***	***Recall***	***F1-score***
0	0.974603	0.974617	0.974603	0.974603
1	0.977324	0.977326	0.977324	0.977324
2	0.976871	0.976919	0.976871	0.976870
3	0.974603	0.974628	0.974603	0.974603
4	0.973243	0.973290	0.973243	0.973242
5	0.974150	0.974356	0.974150	0.974147
6	0.976407	0.976432	0.976407	0.976406
7	**0.983666**	**0.983672**	**0.983666**	**0.983666**
8	0.970508	0.970649	0.970508	0.970506
9	0.975045	0.975253	0.975045	0.975043
Average	*0.975642*	*0.975714*	*0.975642*	*0.975641*

4.2 Experimental validation

In this phase, the objective is to ensure the non-existence of overfitting. Therefore, we used an image dataset with samples that are not been used during the previous phase. The external dataset includes 2,756 images of the "malaria" class and 2,756 for the "normal" class. The validation phase has been carried out with each of the 10 models obtained in the previous 10-fold cross-validation phase. These validation results for classification between classes are shown in Table 6, where the parameter "Area Under the Curve" or "Receiving Operating Characteristics" (ROC) is also presented. The average accuracy value is 97,70%. Moreover, the averaged valued for precision, recall and F1-score are 97.70%, 0.97.69% and 97.69% respectively. A minimum variance between experiments ($\cong 4 \cdot 10^{-6},\ 1{,}5 \cdot 10^{-7}$ for ROC) is clearly shown in the table. Therefore, we can ensure the absence of overfitting.

Table 6. Results of classification average between classes.

Fold	**Accuracy**	**Precision**	**Recall**	**F1-score**	**ROC**
0	0.977142	0.977143	0.977141	0.977141	0.996143
1	0.978788	0.978801	0.978774	0.978773	**0.997121**
2	0.974084	0.974292	0.973875	0.973868	0.996094
3	0.978254	0.978278	0.978229	0.978228	0.996904
4	**0.979563**	**0.979626**	**0.979499**	**0.979497**	0.996379
5	0.973411	0.973491	0.973331	0.973328	0.996098
6	0.977894	0.977922	0.977866	0.977865	0.996798
7	0.978270	0.978310	0.978229	0.978228	0.996274
8	0.976499	0.976583	0.976415	0.976412	0.996893
9	0.975631	0.975754	0.975508	0.975504	0.996243
Average	0.976953	0.977020	0.976887	0.976884	0.996495

Regarding accuracy, 4-fold has proven to be the best in class. Fig. 5 presents its receiver operating characteristics for binary data of 5-fold and Fig. 6 presents the confusion matrix for the cross-validation test.

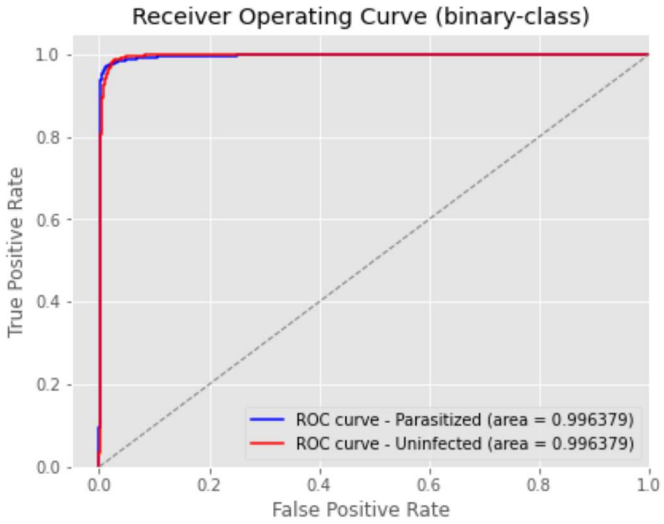


Fig. 5. ROC for 4-fold

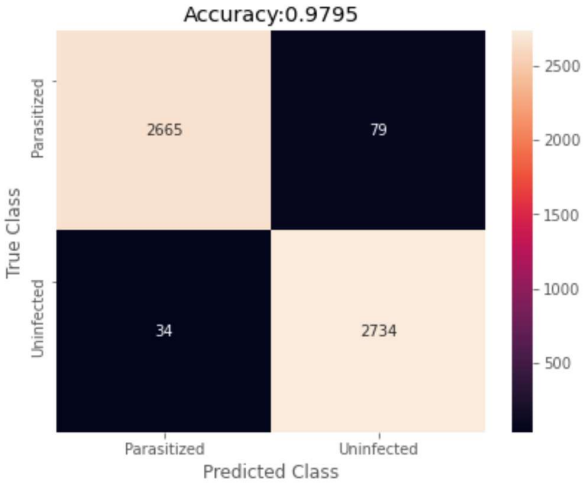


Fig. 6. Confusion Matrix for 4-fold

4.3 Experimental results of the ensemble model

Finally, an ensemble model is proposed. The usual way to carry out an ensembled model is by using different models. However, the proposed ensemble model is composed of 10 trained models in each one of the 10-fold cross-validation. Therefore, there has not been necessary to conduct any additional training to build the ensemble. The results presented by the ensemble model obtained has been improved, despite the use of the same model, trained with almost the same set of images. The accuracy has risen to 98.29% from an averaged value of accuracy of 97.70%. This value represents a 25.75% error decrease of ensemble values over the averaged values of individual methods.

Fig. 7 shows the Receiver Operating Curve of the ensemble model, calculated with the validation values; the AUC value is 99.76%.

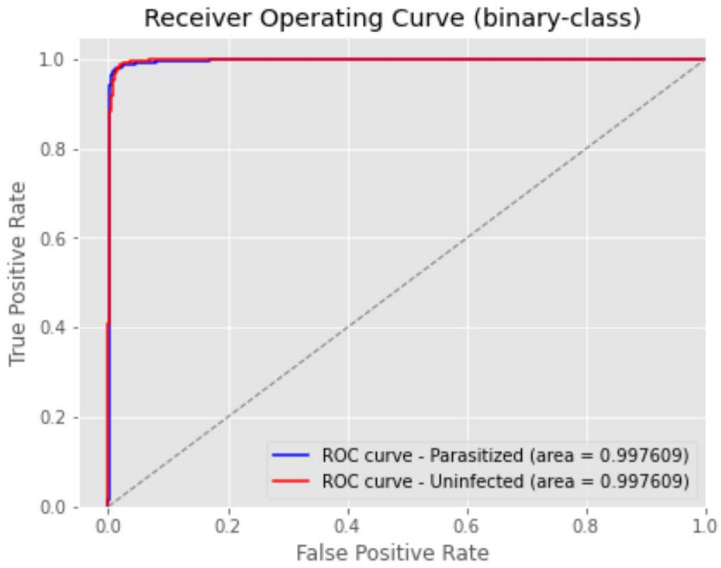


Fig. 7. Roc for ensembled results

Finally, Fig. 8 presents the confusion matrix of this final experiment. False-positive classification almost doubles the quantity of false-negative ones, which seems to indicate that further tunning could be made in this direction as to future work. However, the impact of a false positive is critical when compared with the cost of a false negative. The patient detected with a false positive will be followed by the clinical team who will ensure the absence of disease.

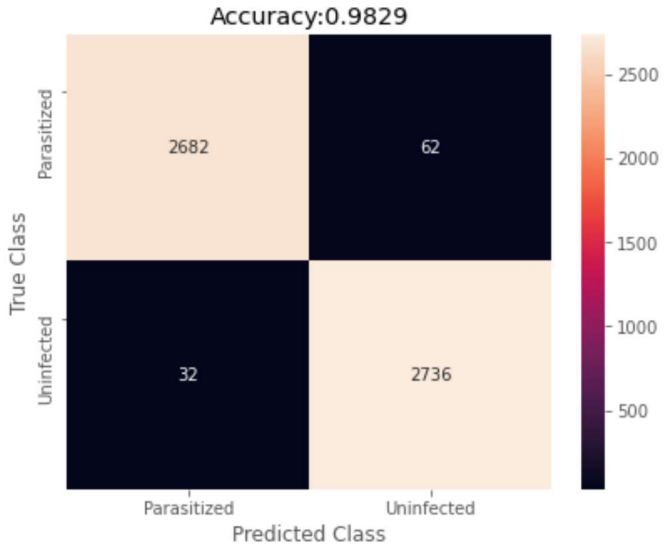


Fig. 8. Confusion matrix for ensembled results

The ensemble model presents an F1-score of 0.9828, a recall of 0.9882, a precision of 0.9774, a false positive rate of 0.0222 and a false negative rate of 0.0118.

5 Discussion

The results are compared with the state of the art. However, it must be noted that the comparison is limited since the proposals use different datasets and parameters. Moreover, as most of the papers do not include the software, it is not feasible to evaluate their techniques to provide a more reliable comparison.

Several CNN methods applied to malaria diagnosis can be found in the literature and they increase every day [32]. The focus on these techniques is to help health professionals. The authors propose various architectures such as ResNet, DenseNet, DPN92 or ADCN [43] for automated medical diagnosis of malaria. We compare our model results with other similar studies that can be found in the literature. Table 7 presents the related work.

Table 7. Comparison results of the state-of-art models for automated medical diagnosis of malaria

Reference	Architecture	Accuracy	Recall	Precision	F1-Score
Liang [39]	Transfer Learning	0.9199	0.8900	0.9512	0.9024
Liang [39]	Own CNN	0.9737	0.9775	0.9699	0.9736
Rajaraman [40]	Own CNN	0.9400	0.9310	0.9512	0.9410
Rajaraman [40]	ResNet50	0.9570	0.9450	0.9690	0.9570
Rahman [41]	Own CNN	0.9629	0.9234	0.9804	0.9495
Rahman [41]	VGG16	0.9777	0.9720	0.9719	0.9709
Shah [42]	Own CNN	0.9477	0.9526	0.9437	0.9481
Quan [43] [44]	DenseNet121	0.9094	0.9251	0.8960	0.9103
Quan [43] [45]	DPN92	0.8788	0.8681	0.8892	0.8785
Quan [43]	ADCN	0.9747	0.9520	0.9350	0.9434
Yang [46]	VGG19	0.9372	0.8731	0.5299	0.6595
Yang [46]	AlexNet	0.9633	0.8215	0.7023	0.7573
Yang [46]	Own CNN	0.9726	0.8273	0.7898	0.8081
Proposed	***EfficientNetB0***	***0.9829***	***0.9882***	***0.9774***	***0.9828***

Accuracy, Recall, Precision and F1-score has been chosen as the evaluation criteria as these performance metrics are provided by most of the literature [39] [40] [41] [42] [43] [46]. Our method outperforms all the related works concerning automated medical diagnosis. The results shown in Table 7 suggest that large and recognized CNN seem not to work optimally in this application. This kind of networks mostly has large and heavy architectures, with a huge number of trainable parameters. Although, simpler customized CNN architectures, with fewer parameters, outperform them. The results indicate that the increase in CNN complexity does not always result in better performance. This could be explained since microscopic peripheral blood cells images have low variability and complexity.

The accuracy of the state-of-art models for automated medical diagnosis of malaria values of Table 7 varies from 87,88% to 97,47%. The proposed method outperforms them. Therefore, the use of the EfficientNet model shows encouraging results for malaria disease. Table 7 contains the accuracy, precision, recall, and F1-score of each fold for the binary classification using the validation data set. Then, an ensemble model, with each one of the 10 different models obtained from the 10-fold validation processes has been developed

which outperforms the individual results of its components. The results show an accuracy of 98,29%, a recall of 98,82%, a precision of 97.74% and an F1-score of 98.28%, which significantly improve the averaged and individual results, shown in Table 6. The AUC is 99,76% and the ROC curve can be shown in Figure 7; Finally, the confusion matrix is displayed in Fig. 8.

As far as the authors of this paper know, no similar work for an automated system for malaria diagnosis can be found in the literature, including a set of characteristics as the followings:

1. The model is based on EfficientNet and uses a pre-trained model for transfer learning.
2. 10-fold stratified cross-validation is applied for validation. This technique selects different datasets of samples for training and testing. It reduces bias and each one of the images is used 9 times in the training process and only 1 in the testing phase.
3. The method involves validation using a separated data set of images, not previously used in the training/testing phase. The validation process used 5.512 samples not previously used. No sign of overfitting has been detected in the results.
4. An ensemble model produced from each of the models obtained in the 10-fold cross-validation process.
5. We have used the Albumentation library to decrease the possibility of overfitting, improve transfer learning execution, expand the volume of the dataset, and reduce execution time.
6. The source code is accessible as a supplementary file to let the research community replicate the results obtained in this paper.

Following the same approach of [43], we present 3 samples of incorrectly classified images for both classes. Fig. 9 shows three infected cells that were categorized as normal. The reason for this behaviour could be that the symptoms are not still obvious, or the pathogen has not been appropriately stained.

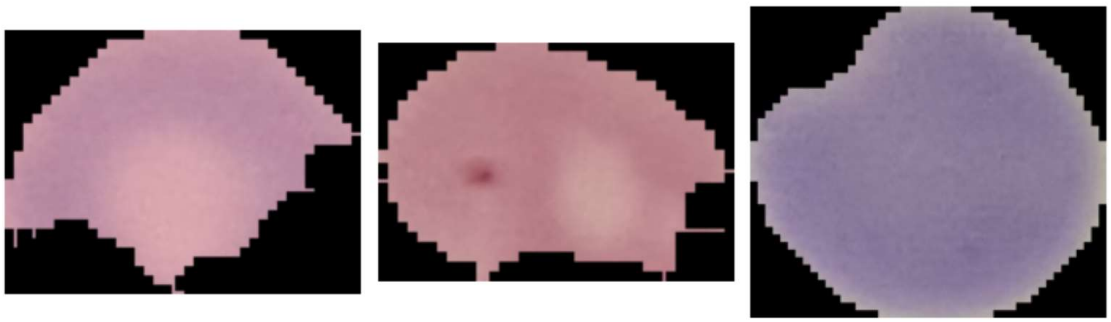


Fig. 9. Three samples of malaria class images misclassified as normal.

On the other hand, Fig. 10 shows three normal cell images that were classified as infected. It may be produced by impurities or staining artefacts, which are very similar to the appearance of the stained pathogen, in these uninfected cell images.

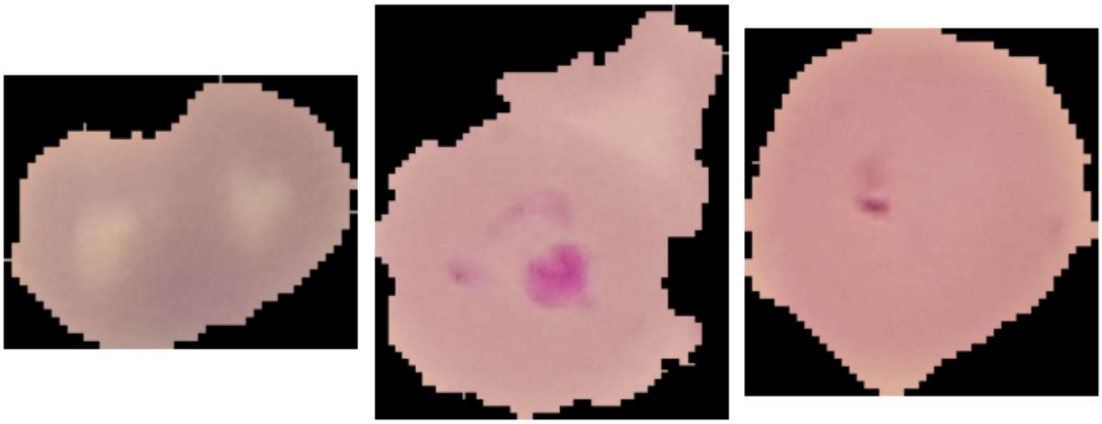


Fig. 10. Three samples of normal class images misclassified as malaria.

The authors state the promising results of the EfficientNet method to support the diagnosis of malaria. Additionally, this study proposes the use of Albumentation and ImageDataAugmentator libraries. This work supports the current body of knowledge because it gives an effective answer for the automated diagnosis of malaria. Automated decision algorithms do not intend to supplant medical experts. Instead, these methods will support medical teams, lower their detection time of malaria and improve the treatment.

6 Conclusion

This study proposes an automated system based on an EfficientNet to assist the identification of malaria in blood cells. The proposed technique employs the EfficientNetB0 method, and it has been assessed using 10-fold cross-validation. An ensembled algorithm composed of the 10 trained model previously calculated is proposed. The results are promising with 98.29% accuracy, 98.82% recall, 97.74% precision, 98.28% F1-score and 99.76% AUC. The authors could not find a similar study that uses EfficientNet for automated detection of malaria.

Nevertheless, this work has limitations common to all the methods found in the literature. Considering the high number of people who have been infected by malaria, the image collections accessible are not yet robust enough. Even so, the number of images available can be continuously collected to train CNN algorithms states and increase their performance. Furthermore, it is necessary to thoroughly study the behaviour of this type of techniques taking into account the progression of the illness in the patient.

All the source code used in this experiment is provided as a supplemental file. Therefore, readers can reproduce the results. Consequently, future research activities can reproduce, update, revise, and modify different parameters to adjust the outcomes of the proposed work.

References

[1] WHO, World Malaria Report 2020, Geneva: World Health Organization, 2020, p. 299.

[2] H. Carballo and K. King, “Emergency department management of mosquito-borne illness: malaria, dengue and west nile virus,” *Emergency medicine practice, ,* vol. 16, no. 5, pp. 1-23, 2014.

[3] WHO, Guidelines for the Treatment of Malaria, 3 ed., Geneva: World Health Organization, 2015.

[4] K. Makhija, S. Maloney and R. Norton, “The utility of serial blood film testing for the diagnosis of malaria,” *Pathology,* vol. 47, no. 1, pp. 68-70, 2015.

[5] K. Mitiku, G. Mengitsu and B. Gelaw, "The reliability of blood film examination for malaria at the peripheral health unit," *Ethiopian Journal of Health Development,* vol. 17, no. 3, pp. 149-246, 2003.

[6] T. Tokumasu, R. Fairhurst and G. Ostera, "Band 3 modifications in Plasmodium falciparum-infected AA and CC erythrocycles assayed by autocorrelation analysis using quantum dots," *Journal of Cell Science,* vol. 118, no. 5, pp. 1091-1098, 2005.

[7] C. Dong, C. Loy and X. Tang, "Accelerating the super-resolution convolutional neural network," in *Compuer Vision - ECC*, Cham, Springer International Publishing, 2016, pp. 391-407.

[8] S. Lawrence and C. C. T. A. Giles, "Face recognition: a convolutional neurl-network aproach," *IEEE Transactions on Neural Networks,* vol. 8, no. 1, pp. 98-113, 1997.

[9] D. Das, S. Koley, S. Bose, A. Maiti, B. Mitra, G. Mukherjee and P. Dutta, "Computer aided tool for automatic detection and delineation of nucleus from oral histopathology images for OSCC screening," vol. 8, 2019.

[10] X. Ji, Q. Yu, Y. Liu and S. Kong, "A recognition method for Italian alphabet gestures based on convolutional neural network," in *Intelligent Computing Theories and Application*, Springer International Publishinf, 2019, pp. 663-664.

[11] A. Tavanei, M. Ghodrati, S. Kheradpiesheh, T. Masquelier and A. Maida, "Deep learning in spiking neural networks," *Neural Networks,* pp. 47-63, 2019.

[12] R. Karthik, M. Hariharan, S. Ananda, P. Mathikshara, A. Johnson and R. Menaka, "Attention embedded residual CNN for disease detection in tomato leaves," *Applied Soft Computing,* vol. 86, no. 105933, 2020.

[13] Y. Wang, X. Wei, H. Shen, L. Ding and J. Wan, "Robust fusion for RGB-D tracking using CNN features," *Applied Soft Computing,* vol. 92, no. 106302, 2020.

[14] J. Rangel, J. Martínez-Gómez, C. Romero-González, J. García-Varea and M. Cazorla, "Semi-supervised 3D object recognition through CNN labeling," *Applied Soft Computing,* vol. 65, pp. 603-613, 2018.

[15] C. Wang, Z. Zhao, Y. Xu and Y. Yu, "A novel multi-focus image fusion by combining simplified very deep convolutional networks and patch-based sequential reconstruction strategy," *Applied Soft Computing,* vol. 91, no. 106253, 2020.

[16] O. Sagi and L. Rochard, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews Data Mining and Knowledge Discovery,* vol. 8, no. 4, 2018.

[17] G. Brown, "Ensemble Learning," in *Encyclopedia of Machine Learning*, Boston, MA: Springer, 2011.

[18] A. Sarwar, M. Ali, M. Jatinder and V. Sharma, "Diagnosis of diabetes type-II using hybrid machine learning based ensemble model," *International Journal of Information Technology,* vol. 12, pp. 419-428, 2020.

[19] D. Yadav and S. Pal, "To Generate an Ensemble Model for Women Thyroid Prediction Using Data Mining Techniques," *Asian Pacific Journal of Cancer Prevention,* vol. 20, no. 4, pp. 1275-1281, 2019.

[20] M. Kaur, A. Malhi and H. Pannu, "Machine learning ensemble for neurological disorders," *Neural Computing and Applications,* vol. 32, no. 8, pp. 12697-12714, 2020.

[21] Y. Xiao, J. Wu, Z. Lin and X. Zhao, "A deep learning-based multi-model ensemble method for cancer prediction," *Computer Methods and Programs in Biomedicine,* vol. 153, no. 1, pp. 1-9, 2018.

[22] M. Re and G. Valentini, "Ensemble methods: A review," in *Advances in Machine Learning and Data Mining for Astronomy*, London, Chapman & Hall, 2012, pp. 563-594.

[23] F. Kimura and M. Shridhar, "Handwritten numerical recognition based on multiple algorithms," *Pattern Recognition,* vol. 24, no. 10, pp. 969-983, 1991.

[24] L. Lam and S. Suen, "Application of majority voting to pattern recognition: an analysis of its behavior and performance," *IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems and Humans,* vol. 27, no. 5, p. 553–568, 1997.

[25] M. Perrone and L. Cooper, "When networks disagree: Ensemble methods for hybrid neural networks," in *How We Learn; How We Remember: Toward an Understanding of Brain and Neural Systems*, New Jersey, World Scientific, 1995, pp. 342-358.

[26] L. Duong, P. Nguyen, C. Di Sipio and D. Di Ruscio, "Automated fruit recognition using efficientnet and mixnet," *Artículos académicos para comput. electron. agric.,* vol. 171, 2020.

[27] M. Tan and Q. Le, "Efficcientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, California, 2019.

[28] M. Tan, "Efficientnet: Improving accuracy and efficiency through automl and model scaling," 29 05 2019. [Online]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Accessed 24 01 2021].

[29] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv 1409.1556,* vol. 9, 2015.

[30] Y. Tai, J. Yang and L. X., "Image Super-Resolution via Deep Recursive Residual Network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[31] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab and H. Radha, "Deep learning algorithm for autonomous driving using GoogLeNet," in *IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, 2017.

[32] M. Poostchi, K. Silamut, R. Maude, S. Jaeger and G. Thoma, "Image analysis and machine learning for detecting malaria," *Translational Research,* vol. 194, no. 4, pp. 36-55, 2018.

[33] Y. Purwar, S. Shah, G. Clarke, A. Almugairi and A. Muehlenbachs, "Automated and unsupervised detection of malarial parasites in microscopic images," *Malaria Journal,* vol. 10, no. 364, 2011.

[34] D. Das, C. Chakraborty, B. Mitra, A. Maiti and A. Ray, "Quantitative microscopy approach for shape-based erythrocytes characterization in anaemia," *Journal of Microscopy,* vol. 249, no. 2, pp. 136-149, 2013.

[35] J. Vink, M. Laubscher, R. Vlutters, K. Silamut, R. Maude, M. Hasan and G. Haan, "An automatic vision-based malaria diagnosis system," *Journal of Microscopy,* vol. 250, no. 3, pp. 166-178, 2013.

[36] S. Sio, W. Sun, S. Kumar, W. Bin, S. Tan, S. Ong, H. Kikuchi, Y. Oshima and K. Tan, "MalariaCount: an image analysis-based program for the accurate determination of parasitemia," *Journal of Microbiological Methods,* vol. 68, no. 1, pp. 11-18, 2007.

[37] S. Savkare and S. Narote, "Automated system for malaria parasite identification," in *015 International Conference on Communication, Information & Computing Technology (ICCICT)*, Mumbai (India), 2015.

[38] L. Malihi, K. Ansari-Asl and A. Behbahani, "Malaria parasite detection in giemsa-stained blood cell images," in *2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP)*, Zanjan (Iran), 2013.

[39] Z. Liang, A. Powell, I. Ersoy, M. Poostchi, K. Silamu, K. Palaniappan, P. Guo, M. Hossain, A. Sameer, R. Maude and e. al., "CNN-Based Image Analysis for Malaria Diagnosis," in *International Conference on bioinformatics and biomedicine*, Shenzhen, 2016.

[40] S. Rajaraman, S. Antani, M. Pootschi, K. Silamut and M. Hossain, "Pre-trained convolutional networks as feature extractors toward improved malaria parasite detection in thin blood smear images," *PeerJ,* vol. 6, no. 4, p. 4578, 2018.

[41] A. Rahman, H. Zunair, M. Y. J. Rahman, S. Biswas, A. Alam, N. Alam and M. Mahdy, "Improving malaria parasite detection from red blood cell using deep convolutional neural networks," 23 07 2019. [Online]. Available: https://arxiv.org/abs/1907.10418. [Accessed 26 2 2021].

[42] D. Shah, K. Kawale, M. Shah, S. Randive and R. Mapari, "Malaria Parasite Detection Using Deep Learning (Beneficial to humankind)," in *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2020.

[43] Q. Quan, J. Wang and L. Liu, "An Effective Convolutional Neural Network for Classifying Red Blood Cells in Malaria Diseases," *Interdisciplinary Sciences: Computational Life Sciences,* vol. 12, pp. 217-225, May 2020.

[44] G. Huang, Z. Liu, L. Maaten and K. Weinberger, "Densely Connected Convolutional Networks," 25 8 2016. [Online]. Available: https://arxiv.org/abs/1608.06993?source=post_page. [Accessed 26 02 2021].

[45] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan and J. Feng, "Dual Path Networks," 6 07 2017. [Online]. Available: https://arxiv.org/abs/1707.01629. [Accessed 26 02 2021].

[46] F. Yang, M. Poostchi, H. Yu, Z. Zhou, K. Silamut, Y. J., M. R.J., S. Jaeger and S. Antani, "Deep Learning for Smartphone-based Malaria Parasite Detection in Thick Blood Smears," *IEEE J Biomed Health Inform,* vol. 24, no. 5, pp. 1247-1438, May 2020 .

[47] I. Ersoy, F. Bunyak, J. M. Higgins and K. Palaniappan, "Coupled edge profile active contours for red blood cell flow analysis," in *9th IEEE International Symposium on Biomedical Imaging (ISBI)*, Barcelona, 2012.

[48] Z. Jan and B. Verma, "Balanced Image Data Based Ensemble of Convolutional Neural Networks," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, Xiamen, China, 219.

[49] P. Yakubovskiy, "Implementation of EfficientNet model. Keras and TensorFlow Keras," [Online]. Available: https://github.com/qubvel/efficientnet. [Accessed 30 01 2021].

[50] A. Parinov, "albumentations," albumentations-team, [Online]. Available: https://github.com/albumentations-team/albumentations. [Accessed 30 01 2021].

[51] G. Marques, D. Agarwal and I. de la Torre, "Automated medical diagnosis of COVID-19 through EfficientNet convolutional neural network," *Applied Soft Computing Journal,* vol. 96, no. 106691, 2020.

[52] A. Buslaev, V. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin and A. Kalinin, "Albumentations: Fast and Flexible Image Augmentations," *Information,* vol. 11, no. 2, p. 125, 2020.

[53] M. Tukiainen, "ImageDataAugmentor," [Online]. Available: https://github.com/mjkvaak/ImageDataAugmentor. [Accessed 18 1 2021].

[54] I. Mohd Jais, A. Ismail and N. S.Q., "Adam Optimization Algorithm for Wide and Deep Neural Network," *Knowledge Engineering and Data Science (KEDS),* vol. 2, no. 1, pp. 41-46, 2019.