Modelo para detectar el uso correcto de mascarillas en tiempo real utilizando redes neuronales convolucionales

Model for correct use of face mask in real time using convolutional neural networks

Hugo Andrade Carrera
Escuela Politécnica Nacional, Quito, Ecuador
hugo.andrade@epn.edu.ec

Soraya Sinche Maita
Escuela Politécnica Nacional, Quito, Ecuador
soraya.sinche@epn.edu.ec
ORCID: 0000-0001-5308-2255

Pablo Hidalgo Lascano
Escuela Politécnica Nacional, Quito, Ecuador
pablo.hidalgo@epn.edu.ec

doi: https://doi.org/10.36825/RITI.09.17.011

Recibido: Noviembre 12, 2020
Aceptado: Enero 21, 2021

Resumen: A partir de la aparición del Covid-19, el mundo ha entrado en una nueva etapa, en la que se pretende mitigar los efectos del virus. Una de las principales medidas adoptadas por muchos países, entre ellos Ecuador, es el uso obligatorio de mascarillas en lugares públicos, y al mantener contacto con personas ajenas al círculo familiar. Es por esto que la finalidad de este artículo es desarrollar un modelo de red neuronal convolucional utilizando Tensorflow basado en MobileNetV2, que permita realizar la detección de mascarillas en tiempo real, en el que su principal aporte es el determinar si la persona está utilizando la mascarilla de forma apropiada. Posteriormente se prueba el modelo utilizando OpenCV y una red neuronal preentrenada para la detección de rostros. Adicionalmente, se analizan las métricas del desempeño de la red neuronal como son: precisión, exactitud *(accuracy)*, exhaustividad (*recall*) y el valor F1. Todas las métricas se analizan en función del número de iteraciones para el entrenamiento del modelo, obteniendo como resultado un modelo que establece tres clasificaciones: rostros sin mascarilla, rostros con mascarilla mal colocada y rostros con mascarilla colocada correctamente. Sus resultados presentan valores de precisión, exhaustividad y F1 superiores al 85% y la exactitud que oscila entre el 93% para 5 iteraciones y 95% para 25 iteraciones.

Palabras clave: *Redes Neuronales Convolucionales, Clasificación Multiclase, Detección de Mascarillas, TensorFlow, MobileNetV2.*

Abstract: Since Covid-19 appeared, the world has entered into a new stage, in which everybody is trying to mitigate the effects of the virus. The mandatory use of face masks in public places and when maintaining contact

with people outside the family circle is one of mandatory measures that many countries have implemented, such as Ecuador, thus, the purpose of this article is to develop a convolutional neural network model using TensorFlow based on MobileNetV2, that allows to perform mask detection in real time video with the key feature of determining if the person is using the face mask properly or if it is not wearing a mask, in order to use the model with OpenCV and a pretrained neural network that detects faces. In addition, the performance metrics of the neural network are analyzed, including precision, accuracy, recall and the F1 score. All performance metrics consider the number of epochs for the training process, obtaining as a result a model that classifies between three groups: faces without face mask, faces wearing a face mask improperly and faces wearing a mask properly. with a great performance in all metrics; The results show values greater than 85% for precision, recall and F1 score, and accuracy values between 93% for 5 epochs and 95% for 25 epochs.

Keywords: *Convolutional Neural Networks, Multiclass Classification, Face Mask Detection, TensorFlow, MobilNetV2.*

1. Introducción

La Organización Mundial de la Salud (OMS) o también conocida como *World Health Organization* (WHO), ha declarado al Covid-19 como una pandemia debido a la rápida propagación del virus. Desde su aparición hasta el mes de octubre del presente año, en todo el mundo se han reportado cerca de cincuenta millones de personas infectadas y alrededor de un millón de muertes [1]. Solamente en Ecuador, se han confirmado 170 mil casos de Covid-19 hasta finales de octubre de 2020, y aunque el sistema de salud ha estado trabajando arduamente para controlar la situación y brindar ayuda a las personas que la necesiten, todos los intentos han sido insuficientes y la cifra de contagios sigue aumentando cada día [2].

Según estudios realizados por la OMS, el uso de mascarillas fuera del domicilio y al mantener contacto con otras personas, es un aspecto fundamental para disminuir el número de contagios y en ciertos casos prevenirlos, por lo que el uso de mascarillas se ha tomado como una medida obligatoria en muchos países [3]. A pesar de que el uso de mascarillas no es suficiente para brindar una protección adecuada en contra del Covid-19, utilizar una mascarilla permite reducir el riesgo de contagio, puesto que protege al portador de una exposición directa con otras personas, sea que éstas se encuentren infectadas o no, y disminuye el esparcimiento del virus proveniente de personas infectadas [4]. No obstante, es necesario mencionar la importancia de utilizar correctamente la mascarilla, pues de nada sirve utilizarla si no se encuentra colocada adecuadamente.

La contribución de este artículo es desarrollar un modelo que contenga una red neuronal para detectar el uso de mascarillas en tiempo real utilizando OpenCV [5], por medio de una red neuronal convolucional construida empleando TensorFlow [6] y MobileNetV2 [7], y teniendo como aspecto fundamental el determinar si las personas se encuentran utilizando la mascarilla de forma correcta o si no llevan colocada una mascarilla, de tal forma que pueda ser utilizado como un método de control de acceso en lugares con gran afluencia de personas y a la vez, facilite detectar posibles focos de contagio del Covid-19. Se pretende también analizar las métricas precisión, exactitud, exhaustividad y el valor F1, resultantes de la implementación de la red neuronal y cómo éstas varían en función del número de iteraciones con las que se entrene a la red neuronal.

El documento se encuentra estructurado de la siguiente manera: en la Sección 2 se revisa el estado del arte, mencionando algunas contribuciones y el estado actual de los temas relacionados en el campo de la investigación; en la Sección 3 se revisa la metodología utilizada para desarrollar el modelo que contiene a la red neuronal, incluyendo los materiales; en la Sección 4 se publican los resultados obtenidos y por último, en la Sección 5, se mencionan algunas conclusiones y una orientación hacia trabajos futuros que puedan complementar este artículo.

2. Estado del arte

Paralelamente a los sucesos ocurridos durante la pandemia, se han realizado algunas contribuciones que permiten reducir la propagación del Covid-19 referentes al área de *Computer Vision* [8] y *Machine Learning* [9], generalmente empleando redes neuronales que permitan reconocer patrones y puedan ser aplicados para detección de objetos.

Hasta el momento, se han desarrollado varios trabajos referentes a *Machine Learning* en los que se emplea Tensorflow y se trabaja con CNNs (*Convolutional Neural Networks*) para realizar la detección de objetos. Los

principales componentes de una CNN son las capas convolucionales, las capas de *pooling* y las capas totalmente conectadas, como se observa en la Figura 1.

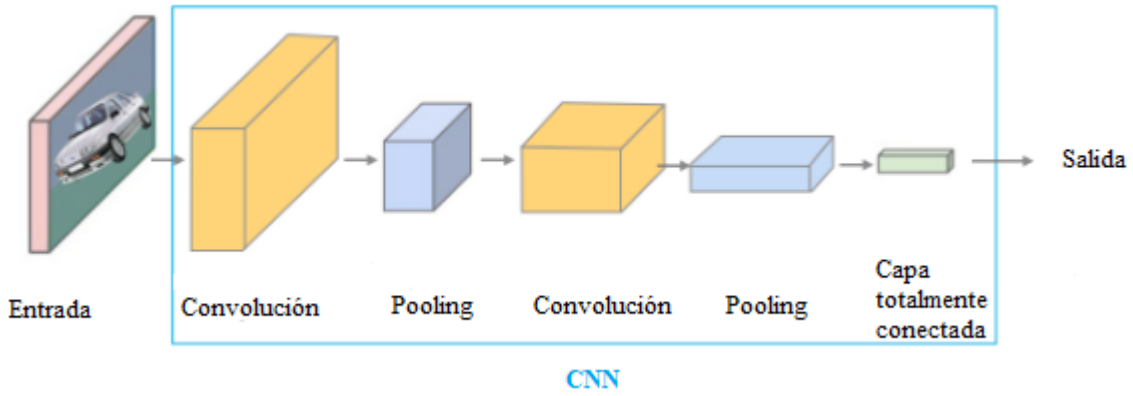

Figura 1. Descripción del funcionamiento de una red neuronal convolucional (CNN) [10].

Un campo relacionado es la detección de rostros, pues es el precursor para realizar la detección de mascarillas en los rostros de las personas; en este campo, se puede destacar el trabajo realizado por Goyal, Agarwal y Kumar, donde se realiza una detección de rostros utilizando OpenCV [11]. En este trabajo se implementa la detección de rostros en tiempo real con seguimiento de los movimientos de la cabeza utilizando un clasificador Haar en un Raspberry Pi BCM2835; también se utiliza la librería OpenCV con una resolución de 1080 pixeles y 30 cuadros por segundo.

Khan, Chakraborty, Astya, y Khepra [12] realizan la detección de rostros mediante un sistema que obtiene datos de entrada a través de una cámara y se trabaja en tiempo real con el algoritmo PCA, el cual, permite reducir la cantidad de datos de procesamiento al enfocarse solamente en las componentes principales.

Específicamente, existen algunas investigaciones donde se realiza la detección de mascarillas utilizando las herramientas mencionadas anteriormente. Un ejemplo es una implementación de la detección de mascarillas desarrollado por Ansor, Ritzkal y Afrianto [13], que utiliza TensorFlow y una CNN pre-entrenada en un Raspberry Pi y se analizan las métricas de la red neuronal construida para el sistema considerando la exactitud, la precisión y la exhaustividad.

Se puede mencionar también el trabajo de Cakiroglu, Ozer y Gunsel [14] en el que se entrena un detector de objetos existente para realizar la detección de mascarillas empleando TensorFlow con una CNN basada en regiones y se analizan los resultados del rendimiento de la misma. El principal aporte del modelo es el entrenamiento con una pequeña cantidad de datos, debido a que en total se dispone de 2695 imágenes.

Joshi, Kanahasabai, Kapil y Gupta [15] realizaron la detección de mascarillas en videos empleando Deep Learning. En este estudio se emplea una CNN multitarea en cascada o también conocida como MTCNN [16] para identificar rostros y sus correspondientes puntos para el reconocimiento en el video, posteriormente se procesan las imágenes faciales mediante un clasificador que utiliza la arquitectura MobileNetV2 y permite reconocer las mascarillas, presentando un buen desempeño en cuanto a la exactitud, la precisión y la exhaustividad. Este modelo fue entrenado con un conjunto de videos tomados en lugares públicos, mientras se irrespetaban los protocolos de bioseguridad en contra del Covid-19.

Militante y Dionisio [17] han empleado técnicas de *Deep Learning* en Python, utilizando TensorFlow y OpenCV para identificar si una persona se encuentra utilizando una mascarilla y si se mantiene un distanciamiento físico entre cada persona. Su *dataset* de entrada se encuentra conformado por veinte mil muestras de imágenes de 224x224 pixeles y como resultado se ha obtenido un 97% de exactitud en el modelo. Esta implementación también permite tomar una captura de pantalla en caso de que la persona que no utilice una mascarilla o que no mantenga un distanciamiento físico con otras personas.

Sin embargo, hasta el momento existen muy pocas investigaciones relacionadas con la detección de mascarillas que contemplen el uso correcto de las mismas, teniendo en cuenta que, como producto final, se obtenga un modelo que pueda trabajar en tiempo real y pueda ser implementado en un sistema embebido, para que pueda

funcionar como un método de control de acceso y facilitar la identificación de personas que están haciendo mal uso de la mascarilla.

3. Materiales y metodología

Para implementar un modelo de red neuronal que permita realizar la detección de mascarillas en tiempo real y a su vez, permita determinar si la persona se encuentra usando la mascarilla de forma adecuada, es necesario disponer de un IDE y tener instalados algunos *Frameworks* compatibles con Python como TensorFlow y OpenCV; así como también disponer de algunas librerías entre las que se encuentran Numpy, Keras, Sklearn y Matplotlib.

El proceso de obtención del modelo se ve reflejado primeramente en la construcción de un dataset, el cual, debe formarse a partir de imágenes de tres clases: rostros de personas que porten una mascarilla de forma correcta (*with_mask*), rostros de personas que porten una mascarilla de forma incorrecta (*bad_mask*) y rostros de personas que no porten una mascarilla (*without_mask*). En este caso, se debe crear una carpeta contenedora del dataset y colocar las imágenes de cada clase en carpetas separadas. Cabe mencionar que las imágenes pueden ser obtenidas en la web mediante cualquier motor de búsqueda; en este caso, se utilizó el buscador de Google, debiendo disponer de un número muy similar de muestras para cada clase, en este caso se utilizaron mil muestras para cada una.

Una vez construido el *dataset*, se procede a abrir un nuevo archivo en IDE de Python y se importan todas las librerías necesarias para el funcionamiento del modelo. Posteriormente se debe definir la tasa de aprendizaje, el número de iteraciones y el tamaño del lote, que se define como el número de muestras de entrenamiento utilizadas en cada iteración. Para este caso específicamente se utilizó una tasa de aprendizaje de 0.0001, un número de iteraciones variable (5, 10, 15, 20 y 25) y un tamaño de lote de 32.

Es necesario cargar al modelo las imágenes que se encuentran alojadas en el *dataset*, para esto, se debe leer el directorio de la carpeta contenedora, de la cual se extrae la etiqueta que identifica la clase y el nombre del archivo de imagen. Para realizar el preprocesamiento a las imágenes, en primer lugar, se debe ajustar el tamaño de las imágenes a 224 x 224 pixeles, puesto que es el tamaño con el que trabaja MobileNetV2 e ImageNet [18]; se debe convertir las imágenes a un arreglo (*array*) y utilizar el método *preprocess_input*. Finalmente se agregan las imágenes y la etiqueta de la clase a la que pertenece cada una, en dos listas diferentes.

En cuanto a las etiquetas, es necesario transformarlas a binario mediante el método *LabelBinarizer* y luego transformarlas en variables categóricas, con la finalidad de obtener una matriz *one-hot*, en la que cada fila representa una muestra y posee un uno (1) solamente en la columna de la clase correspondiente y en las otras columnas se tiene ceros (0). El siguiente paso es separar el total de las muestras de entrada en un conjunto de datos para entrenamiento y en otro conjunto de datos para prueba; se lo realiza mediante el método *train_test_split,* teniendo en cuenta que el 80% de las muestras serán para entrenamiento, el 20% para prueba y se estratifican de acuerdo a las etiquetas.

Para aumentar la cantidad de muestras durante el proceso de entrenamiento y así obtener mejores resultados, se puede utilizar el método *ImageDataGenerator*, indicando sus parámetros correspondientes como son: rotación de la imagen, zoom, desplazamiento en ancho y alto de la imagen, corte de una parte y giro horizontal.

El modelo base se crea utilizando MobileNetV2, obteniendo los pesos de la red neuronal ImageNet y se incluye el formato de las entradas de 224x224x3 pixeles. En la salida del modelo base se tiene la cabeza del modelo, es decir, en las capas más externas de la red neuronal. Aquí se implementa una capa *average-pooling* de 7x7, una capa para convertir la matriz resultante en un vector mediante el método *flatten*, una capa totalmente conectada con 128 neuronas y una función de activación ReLu, una capa regularizadora que implementa *dropout* para excluir el 50% de las neuronas y reducir el sobreajuste, y por último una capa de salida con tres neuronas y función de activación Softmax para realizar la clasificación entre las tres clases. De esta forma, el modelo completo tendrá como entrada al modelo base y como salida a la cabeza del modelo, como se observa en la Figura 2.

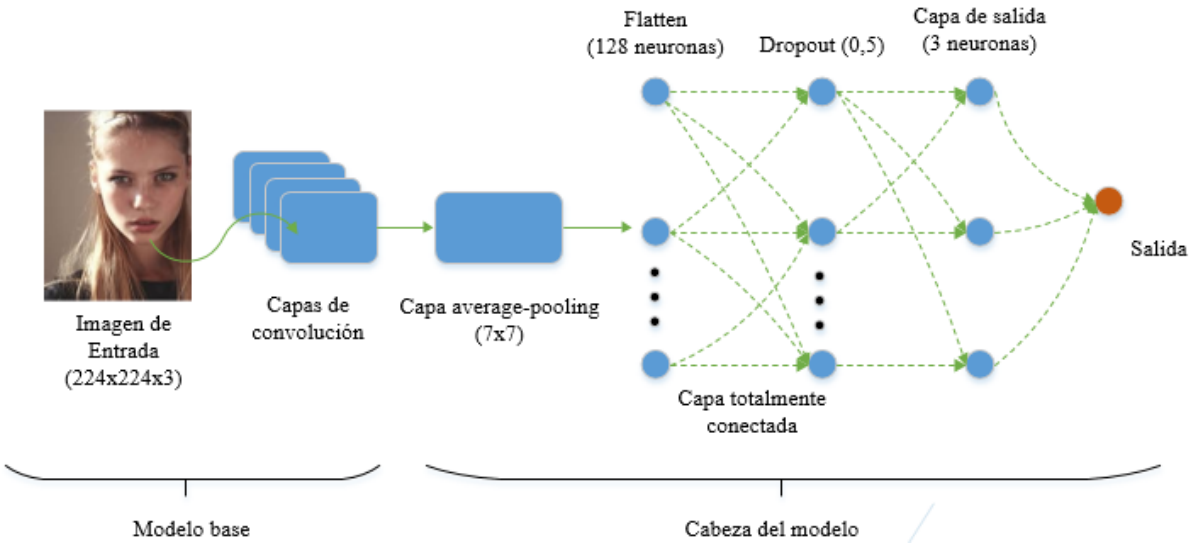

Figura 2. Representación de la red neuronal convolucional utilizada en el modelo. Fuente: Elaboración propia.

El modelo se compila utilizando el método *model.compile* y se selecciona una función de pérdida *categorical_crossentropy*, puesto que esta aplicación corresponde a una clasificación multiclase. El proceso de entrenamiento se ejecuta mediante el método *model.fit*, indicando el número de pasos por iteración, el conjunto de datos de validación y el número de pasos para la validación. Finalmente se evalúa el modelo mediante el método *model.predict* indicando el conjunto de datos de prueba y el tamaño del lote.

Para obtener las métricas del rendimiento del modelo se utiliza un reporte de clasificación, construido a partir del método *classification_report*; en este caso, se analiza la exactitud del modelo, la precisión, la exhaustividad y el valor F1 para cada una de las tres clases. También es posible graficar la exactitud y el valor del coste durante el proceso de entrenamiento en función del número de iteraciones, utilizando la librería Matplotlib y las métricas obtenidas anteriormente.

Por último, es necesario guardar el modelo para poder utilizarlo en la detección de mascarilla en imágenes o en videos en tiempo real en un archivo *model*; para esto se utiliza el método *model.save* indicando el nombre del archivo y el formato en el que se quiere guardar, en este caso, se selecciona el formato h5.

Una vez obtenido el modelo, se procede integrarlo en un video en tiempo real utilizando OpenCV, en este caso se utiliza el modelo que presente mejor desempeño. Se requiere cargar dos redes neuronales, una que detecta rostros de personas y otra que detecta mascarillas. Para detectar los rostros se utilizó una red neuronal profunda preentrenada disponible en Github [19] y las mascarillas se detectan en base al modelo construido anteriormente.

Es necesario iniciar la captura de tramas utilizando el método *VideoCapture* y especificando la fuente de video; para este artículo, se empleó la cámara que viene integrada en el computador. El algoritmo trabaja detectando un blob dentro de cada cuadro del video, que sirve como entradas para la red neuronal de detección de rostros y a la salida se obtiene la ubicación de los rostros; en cuanto a la detección de mascarillas, se obtienen las predicciones de las tres clases utilizando el método *model.predict*. Finalmente se muestran los resultados de las predicciones en cada cuadro de video utilizando la ubicación proporcionada de la detección de rostros.

4. Resultados

De la implementación del modelo para detectar mascarillas y su correcto uso se obtuvo un archivo que puede ser utilizado posteriormente con OpenCV o con cualquier otra aplicación que permita manejar video en tiempo real; también se obtuvieron las métricas relacionadas al desempeño de la red neuronal construida y, por último, un gráfico de la exactitud y el valor del coste durante el proceso de entrenamiento en función del número de iteraciones.

Los resultados de las métricas obtenidas en función del número de iteraciones se muestran en las Tablas 1, 2, 3, 4 y 5.

Tabla 1. Métricas del desempeño del modelo con cinco iteraciones.

Clase	Exactitud	Precisión	Exhaustividad	Valor F1
With mask	0.93	0.94	0.94	0.94
Without mask	0.93	0.94	0.85	0.90
Bad mask	0.93	0.91	0.99	0.95

Fuente: Elaboración propia.

Tabla 2. Métricas del desempeño del modelo con diez iteraciones.

Clase	Exactitud	Precisión	Exhaustividad	Valor F1
With mask	0.95	0.96	0.96	0.96
Without mask	0.95	0.96	0.89	0.92
Bad mask	0.95	0.93	0.99	0.96

Fuente: Elaboración propia.

Tabla 3. Métricas del desempeño del modelo con quince iteraciones.

Clase	Exactitud	Precisión	Exhaustividad	Valor F1
With mask	0.94	0.96	0.96	0.96
Without mask	0.94	0.95	0.85	0.90
Bad mask	0.94	0.89	0.99	0.94

Fuente: Elaboración propia.

Tabla 4. Métricas del desempeño del modelo con veinte iteraciones.

Clase	Exactitud	Precisión	Exhaustividad	Valor F1
With mask	0.95	0.97	0.97	0.97
Without mask	0.95	0.97	0.88	0.92
Bad mask	0.95	0.91	0.99	0.95

Fuente: Elaboración propia.

Tabla 5. Métricas del desempeño del modelo con veinticinco iteraciones.

Clase	Exactitud	Precisión	Exhaustividad	Valor F1
With mask	0.94	0.96	0.98	0.97
Without mask	0.94	0.97	0.87	0.92
Bad mask	0.94	0.91	0.98	0.94

Fuente: Elaboración propia.

Los resultados muestran un incremento en la precisión, la exhaustividad y el valor F1 para las distintas clases, a medida que se entrena el modelo con mayor cantidad de iteraciones. El modelo presenta una exactitud que tiene un valor del 93% para cinco iteraciones, 94% para quince y veinticinco iteraciones, mientras que su mayor valor es del 95% para diez y veinte iteraciones.

La clase que menor valor de precisión presenta es *bad mask*, mientras que para las clases *with mask* y *without mask* se obtienen valores mayores muy similares. El menor valor de precisión es de 0.89 y se obtiene en la clase *bad mask* en el modelo entrenado con quince iteraciones y el valor más alto es de 0.97 que se obtiene en el modelo entrenado con veinte iteraciones.

En cuanto a la exhaustividad, el menor valor es de 0.85 para la clase *without mask* en el modelo entrenado con quince iteraciones, mientras que se obtiene un valor de 0.99 para la clase *bad mask* en los todos los modelos, con excepción del modelo entrenado con veinticinco iteraciones.

El valor F1 es muy representativo y útil al momento de cuantificar el desempeño de una red neuronal, pues permite combinar la precisión y la exhaustividad en un solo valor. Se obtienen los valores más altos de F1 para el modelo entrenado con 20 iteraciones y el valor más bajo de F1 se obtiene en el modelo entrenado con cinco iteraciones.

Los gráficos del valor del coste y la exactitud obtenidas del modelo en función del número de iteraciones se muestran en la Figura 3.

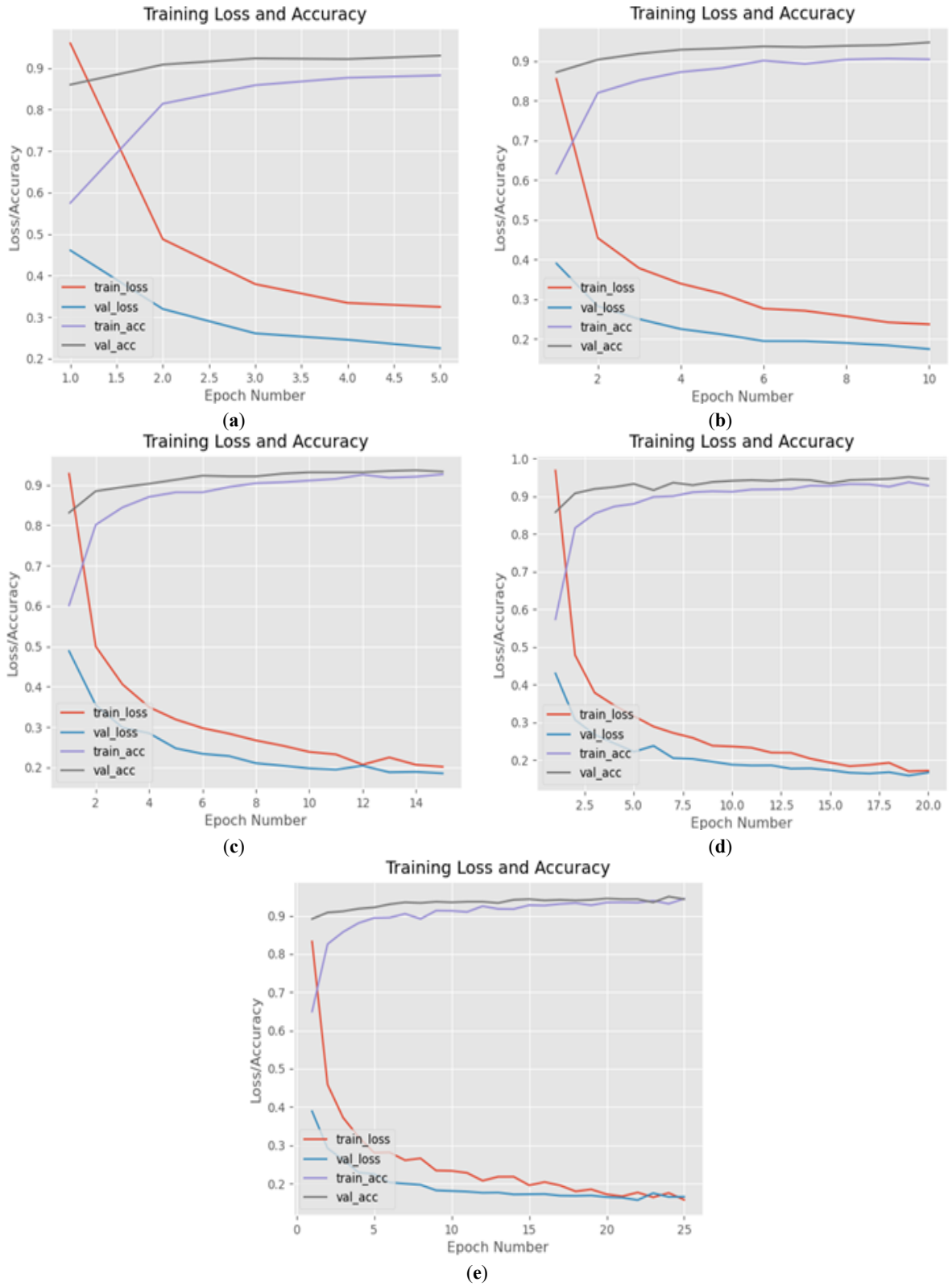

Figura 3. Gráficos del valor del coste y exactitud en función del número de iteraciones en el proceso de entrenamiento del modelo: (a) Para 5 iteraciones; (b) Para 10 iteraciones; (c) Para 15 iteraciones; (d) Para 20 iteraciones; (e) Para 25 iteraciones.

Los resultados obtenidos al momento de probar el modelo construido para la detección de mascarillas en tiempo real se muestran en la Figura 4.

(a) (b) (c)

Figura 4. Resultados del desarrollo del modelo de detección del correcto uso de mascarillas utilizando OpenCV en tiempo real: (a) Rostros sin mascarilla; (b) Rostros con mascarilla mal colocada; (c) Rostros con mascarilla colocada correctamente.

Los resultados muestran que el modelo es capaz de clasificar correctamente entre rostros con mascarilla, rostros con mascarilla mal utilizada y rostros sin mascarilla de diferentes personas, en tiempo real utilizando OpenCV y la cámara integrada en el computador. También cabe mencionar que el tiempo de procesamiento que toma capturar las tramas desde la cámara web del computador, detectar el rostro y posteriormente detectar el correcto uso de la mascarilla, no genera un retardo perceptible**.**

5. Conclusiones y Trabajo Futuro

Basándose en los resultados obtenidos del desarrollo del modelo, es posible construir un modelo que contenga una red neuronal que clasifica en tres clases: rostros que porten una mascarilla de forma incorrecta, rostros que porten una mascarilla de forma correcta y rostros que no porten una mascarilla. El modelo puede ser construido utilizando imágenes obtenidas en la web como datos de entrada; como estructura del modelo se puede utilizar la red neuronal convolucional MobileNetV2 y las salidas corresponden a la clasificación entre las tres clases de imágenes.

Se puede realizar la clasificación de más de dos clases utilizando TensorFlow y MobileNetV2, empleando el método *train_test_split* para separar las muestras de entrada en el conjunto de datos de entrenamiento y en el conjunto de datos de prueba, siempre y cuando la matriz *one-hot* se haya construido de forma adecuada, tomando en cuenta el orden de las etiquetas que representan a cada clase.

Para realizar una clasificación multiclase se debe emplear una función de coste de entropía cruzada categórica y no utilizar una función de coste de entropía cruzada binaria, puesto que ésta se emplea en clasificación binaria o en clasificación multietiqueta. Además, se concluye que se deben tener tantas neuronas en la capa de salida de la red neuronal con una función de activación Softmax como categorías para clasificar.

De las métricas obtenidas, se concluye que el número de iteraciones influye en el desempeño del modelo de red neuronal. A medida que el número de iteraciones aumenta, se observa un ligero incremento en la precisión, la exhaustividad y el valor F1 para las distintas clases. El modelo presenta una exactitud que tiene un valor del 93% para cinco iteraciones y del 95% para diez y veinte iteraciones. El menor valor de precisión se obtiene para la clase *bad mask*, mientras que para las clases *with mask* y *without mask* se obtienen valores mayores muy similares. En cuanto a la exhaustividad, se obtiene el valor más bajo para la clase *without mask*, mientras que el mayor valor se obtiene para la clase *bad mask* y lo mismo sucede para el valor F1.

De los resultados obtenidos, se puede inferir que, con diez iteraciones, el modelo obtiene un rendimiento óptimo y presenta mejoras leves en las métricas de desempeño al entrenar el modelo con un mayor número de iteraciones, por lo que es posible optimizar el tiempo que toma entrenar el modelo. Se concluye también que el modelo entrenado con veinte iteraciones presentó los valores más altos en cuanto a exactitud, precisión, exhaustividad y valor F1, por lo tanto, se utilizó este modelo en las pruebas en tiempo real.

Se puede concluir que el sistema de detección de mascarillas en tiempo real funciona correctamente, clasificando con gran exactitud cada una de las tres clases y no presenta retardo debido al procesamiento, con respecto al flujo de video obtenido de la cámara del computador.

Como direcciones de investigación futuras se plantea utilizar el modelo entrenado para la detección de mascarillas y su correcto uso en videos en tiempo real, orientado a sistemas de control de acceso, automatizando el ingreso a lugares con gran afluencia de personas, en donde podría resultar difícil realizar el control manual del uso correcto de mascarillas. Se propone también aprovechar el hecho de que el modelo es multiplataforma y puede ser trasladado a sistemas embebidos, de tal forma que pueda ser utilizado en cualquier lugar y así ayudar a reducir focos de contagio y a combatir el Covid-19.

6. Referencias

[1] World Health Organization. (2020). *WHO Coronavirus Disease (COVID-19) Dashboard*. Recuperado de: https://covid19.who.int/

[2] Ministerio de Salud Pública. (s/f). *Actualización de casos de coronavirus en Ecuador – Ministerio de Salud Pública.* Recuperado de: https://www.salud.gob.ec/actualizacion-de-casos-de-coronavirus-en-ecuador/

[3] World Health Organization. (2020). *Coronavirus disease (COVID-19): Masks*. Recuperado de: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/q-a-on-covid-19-and-masks

[4] World Health Organization. (2020). *When and how to use masks*. Recuperado de: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/advice-for-public/when-and-how-to-use-masks

[5] Zelinsky, A. (2009). Learning OpenCV---Computer Vision with the OpenCV. *IEEE Robotics & Automation Magazine*, 16 (3), 100-100. doi: https://doi.org/10.1109/MRA.2009.933612

[6] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. Trabajo presentado en *12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah, USA. doi: https://doi.org/10.1016/0076-6879(83)01039-3

[7] Jingbo, H., Yang, T. (2020). Surface Water Quality Classification Based on MobileNetV2 Surface Water Quality Classification Based on MobileNetV2. *Journal of Physics: Conference Series*, 1646, 1–4. doi: https://doi.org/10.1088/1742-6596/1646/1/012049

[8] Mohammed Solyman, A. M. (2019). *Introduction to Computer Vision*. Egyptian Atomic Energy Authority. Cairo, Egypt.

[9] Başţanlar, Y., Özuysal, M. (2014). Introduction to machine learning. *Methods in Molecular Biology*, 1107, 105–128. doi: https://doi.org/10.1007/978-1-62703-748-8_7

[10] Camacho, C. (2018). *Convolutional Neural Networks – Machine and deep learning educator*. Recuperado de: https://cezannec.github.io/Convolutional_Neural_Networks/

[11] Goyal, K., Agarwal, K., Kumar, R. (2017). Face detection and tracking: Using OpenCV. Trabajo presentado *en International Conference on Electronics, Communication and Aerospace Technology (ICECA*), Coimbatore, India. doi: https://doi.org/10.1109/ICECA.2017.8203730

[12] Khan, M., Chakraborty, S., Astya, R., Khepra, S. (2019). Face Detection and Recognition Using OpenCV. Trabajo presentado en *International Conference on Computing, Communication, and Intelligent Systems, (ICCCIS),* Greater Noida, India. doi: https://doi.org/10.1109/ICCCIS48478.2019.8974493

[13] Ansor, A., Ritzkal, R., Afrianto, Y. (2020). Mask Detection Using Framework Tensorflow and Pre-Trained CNN Model Based on Raspberry Pi. *Journal Mantik*, 4 (3), 1539–1545.

[14] Cakiroglu, O., Ozer, C., Gunsel, B. (2019). Design of a deep face detector by mask R-CNN. Trabajo presentado en *27th Signal Processing and Communications Applications Conference*. Sivas, Turkey. doi: https://doi.org/10.1109/SIU.2019.8806447

[15] Joshi, A. S., Joshi, S. S., Kanahasabai, G., Kapil, R., Gupta, S. (2020). Deep Learning Framework to Detect Face Masks from Video Footage. Trabajo presentado en *12th International Conference on Computational Intelligence and Communication Networks (CICN)*, Bhimtal, India. doi: https://doi.org/10.1109/CICN49253.2020.9242625

[16] Qin, W., Wang, L., Luo, W. (2017). Face Recognition Based On Gabor Local Feature and Convolutional Neural Network. Trabajo presentado en *2nd International Conference on Computer Engineering, Information Science & Application Technology),* Paris, France. doi: https://doi.org/10.2991/iccia-17.2017.94

[17] Militante, S. V., Dionisio, N. V. (2020). Deep Learning Implementation of Facemask and Physical Distancing Detection with Alarm Systems. Trabajo presentado en *Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, Surabayam, Indonesia. doi: https://doi.org/10.1109/ICVEE50212.2020.9243183

[18] Deng, J., Dong, W., Socher, R., Li, L.-J., Li., K., Fei-Fei, L. (2010). ImageNet: A large-scale hierarchical image database. Trabajo presentado en *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA. doi: https://doi.org/10.1109/cvpr.2009.5206848

[19] Saaduddin, M. (2019). *GitHub - simplesaad/FaceDetection_Realtime: This is a Python 3 based project to perform fast & accurate face detection with OpenCV face detection to videos, video streams, and webcams using a pre-trained deep learning face detector model shipped with the library*. Recuperado de: https://github.com/simplesaad/FaceDetection_Realtime