Implementación de un modelo de deep learning para clasificar escenarios reales y editados.

Luis Ángel Cruz García[1]

Correo institucional: A01736345@tec.mx

[1] *Técnologico de Monterrey, Escuela de Ingeniería y Ciencias, Epigmenio González 500 Fracc. San Pablo 76130 Querétaro, Qro, México.*

Primera revisión 2 de noviembre del 2024

En este reporte se describirán los procesos llevados a cabo para implementar modelo de deep learning con el fin de comprender cómo es que dicho modelo funciona y con el objetivo de lograr un resultado eficaz que brinde predicciones acerca de escenarios reales o editados , pasando desde la etapa de la elección de la base de datos hasta la estimación de la eficacia de dicho modelo.

1. Introducción

El aprendizaje profundo o deep learning, es una técnica de machine learning, en la que se implementan redes neuronales profundas, las cuales están formadas por varias capas de nodos interconectados, donde cada uno se basa en la capa anterior para optimizar la predicción. El principal diferenciador entre una red neuronal profunda y una red neuronal tradicional, es el número de capas y la complejidad de las mismas, si bien no existe un número exacto para definir cuándo se considera deep learning, normalmente un modelo con esta característica suele tener desde cientos de capas [1].

En cuanto a la situación planteada, es importante mencionar que actualmente nos encontramos en un auge del machine learning, es por eso que cada vez más personas tienen acceso a estas tecnologías, como modelos que generan texto, imágenes, sonido, etc. Esto agregado a que de por sí ya existían formas de editar imágenes de manera convencional, nos deja en una situación en la que ya no podemos estar totalmente seguro de que lo que vemos en una imágen es real, es por eso que en este reporte se describe el proceso llevado a cabo para lograr clasificar entre imágenes reales e imágenes editadas, más específicamente en entornos de carreteras, y en donde los elementos editados son los automóviles que aparecen en la imágen. Para esto se irán modificando diferentes elementos de las imágenes y del dataset en general para alcanzar el mejor resultado posible utilizando PyTorch para la parte de implementación.

2. Obtención del dataset

La base de datos fue obtenida de la plataforma Kaggle, en la cual podemos encontrar diferentes tipos de elementos relacionados con el aprendizaje máquina, en este caso utilizaremos los recursos encontrados en la competencia llamada CIDAUT AI Fake Scene Classification 2024 [2] la cual permite el uso de repositorios públicos de GitHub, modelos preentrenados y otros conjuntos de datos disponibles públicamente. Ahora bien, durante esta fase en un proceso más complejo, es donde se podría tomar la información mediante diferentes técnicas, además de comprobar que la información obtenida es verídica y/o proviene de fuentes confiables. En este caso específico se considerará que la información del dataset cuenta con estas características sin hacer un énfasis muy específico en esta etapa ya que no forma parte de los principales objetivos del proyecto.

Uno de los elementos a destacar es que también se debe de identificar si existe documentación que ayude a entender mejor la información, en este caso se nos brinda como información inicial en Kaggle que las imágenes con las que se cuentan están en formato RGB comprimidas en JPEG. Se cuenta con un archivo train.csv con las etiquetas a modo de caracteres (real y editada) del dataset de entrenamiento, sin embargo, el dataset de prueba no se encuentra etiquetado y está más destinado a la entrega de resultados de la competencia de Kaggle, en dichos resultados se espera que las etiquetas indiquen si la imagen es real (1) o falsa (0). Puedes realizar tu propia división de los datos en conjuntos de entrenamiento y validación.

3. Análisis del dataset

A. Cantidad de datos

Utilizando de base el archivo train.csv, que contiene los labels, se puede ver que se cuenta con 720 imágenes.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 720 entries, 0 to 719
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   image   720 non-null    object
 1   label   720 non-null    object
dtypes: object(2)
memory usage: 11.4+ KB
```

Fig. 1. Información de train.csv dada por Python

B. Transformaciones iniciales

En la figura 1 se puede observar que los datos del archivo son de tipo object, esto es debido a que el label no se encuentra codificado, sino que aún se encuentran como caracteres, por lo que lo que dentro de una clase personalizada se cambiaron los valores, siendo 1 para real y 0 para editada.

4. Eligiendo pruebas

Lo siguiente que se hizo fue utilizar la técnica de split para entrenar y estimar el rendimiento del modelo utilizando datos distintos para cada uno de estos procesos. En este caso utilicé el

80% de los datos para mi conjunto de entrenamiento y el otro 20% para el conjunto de prueba, a su vez, el 80% de entrenamiento se dividió en 70% para entrenamiento y 10% para validación.

```
train data:  (504, 2)
test data:  (144, 2)
validation data:  (72, 2)
```

Fig. 2. Cantidad de datos destinados a cada conjunto

Análisis del modelo

Las redes neuronales tienen un uso muy amplio por la flexibilidad que tienen y la complejidad que pueden alcanzar, a esto se agrega que existen modelos preentrenados que pueden brindar un resultado aún mejor al ser adaptados a una problemática.

5. Composición de la red

A. Función de costo

Los modelos implementados utilizan la función de entropía cruzada (Cross-Entropy loss) como función de costo debido a que es un problema de clasificación.

B. Métricas

Las métricas permiten evaluar la calidad del modelo, las utilizadas serán el Accuracy, el F1-score y la matriz de confusión.

6. Hiperparámetros

Debido a la complejidad de las redes utilizadas, comencé con una base de 40 épocas, además de un learning rate de 0.0001 que funciona en conjunto con el algoritmo Adam como optimizador, el cual es un algoritmo para el descenso de gradiente. Este método es muy eficiente al trabajar con problemas grandes que implican muchos datos o parámetros. Requiere menos memoria y es más eficaz. Intuitivamente, combina el algoritmo de "descenso de gradiente con momento" y el algoritmo "RMSProp" [3].

7. Prueba de modelos

A. Modelo 1

El primer modelo que utilicé fue EfficientNet-B2 con los pesos preentranados de ImageNet.
Además de contar con las siguientes aumentaciones:

- Redimensión de imagen a un tamaño de 260x260 píxeles.
- Ajuste en el brillo y contraste
- Ajuste en la saturación y tono
- Normalización

Obteniendo los siguientes resultados para el conjunto de validación respecto al de entrenamiento.

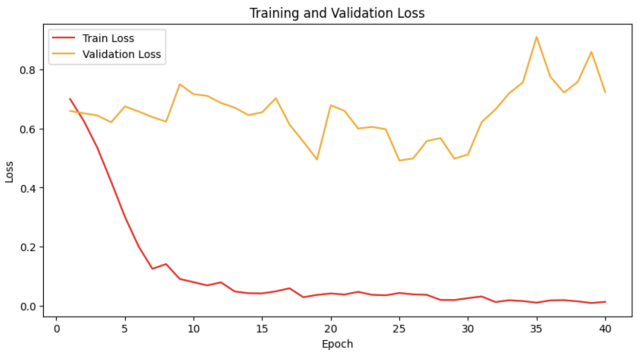

Fig. 3. Gráfica de pérdida modelo 1

En la gráfica 3 se puede observar que la pérdida en los datos de entrenamiento se comporta de manera correcta, y se puede notar la tendencia que al incrementar las épocas ya no disminuirá mucho más, si no que se mantendrá similar. Por otra parte, la pérdida en el conjunto de validación no resulta como se espera, dando un primer indicio de overfitting que tampoco parece corregirse con el pasar de las épocas.

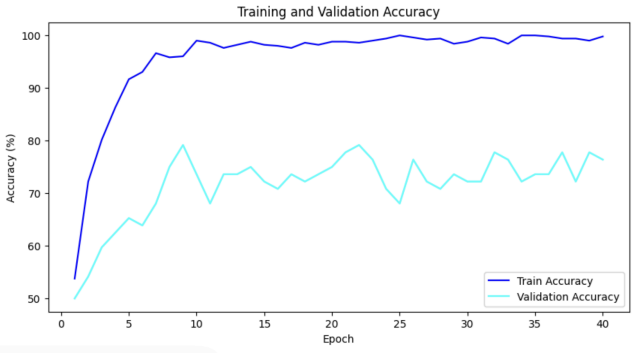

Fig. 4. Gráfica de Accuracy modelo 1

En la figura 4 se puede observar el mismo problema que en la figura 3 pero que nos ayuda a definir el problema de manera más segura, y es que se cuenta con overfitting.

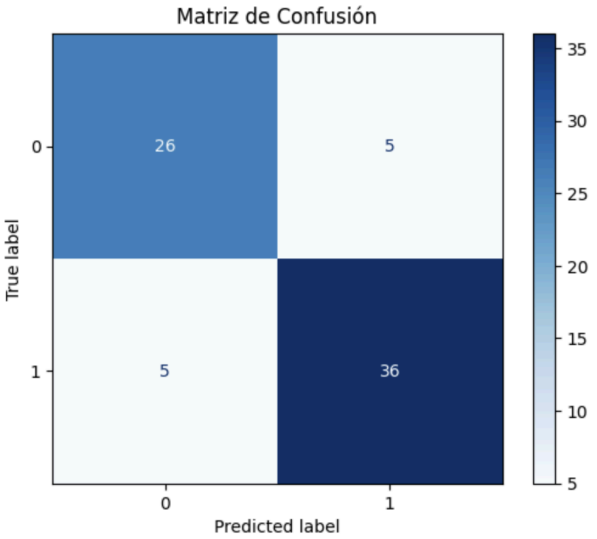

Fig. 5. Matriz de confusión de validación modelo 1

Por otra parte, en la matriz de confusión vemos que realmente los valores obtenidos son buenos en cuanto a la tendencia de predecir una clase, ya que no se nota que se elija significativamente una más que otra.

Al hacer la evaluación con el conjunto de prueba y comparar los resultados podemos ver que el overfitting es notable, aunque aún con eso, el modelo cuenta con un accuracy que es mejorable pero que llega a cerca del 80% para test.

Conjunto de datos	**Accuracy**
Train	99.81%
Validation	76.39%
Test	78.08%

B. Modelo 2

En este caso fue el mismo modelo, sin embargo, agregué más aumentaciones con el fin de reducir el overfitting, quedando de la siguiente manera:

.

- Redimensión de imagen a un tamaño de 260x260 píxeles.
- Ajuste en el brillo y contraste
- Ajuste en la saturación y tono
- Volteo horizontal aleatorio (0.5)
- Rotación aleatoria hasta 15°
- Desplazamiento corto aleatorio
- Desenfoque gaussiano
- Normalización

Obteniendo los siguientes resultados para el conjunto de validación respecto al de entrenamiento.

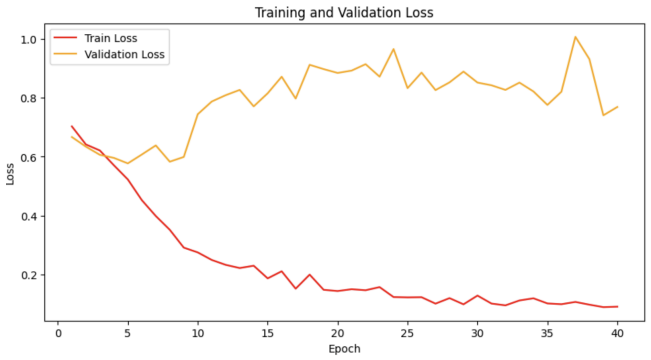

Fig. 6. Gráfica de pérdida modelo 2

En la gráfica 6 se puede observar se mantiene una tendencia similar a la que se veía con las aumentaciones pasadas.

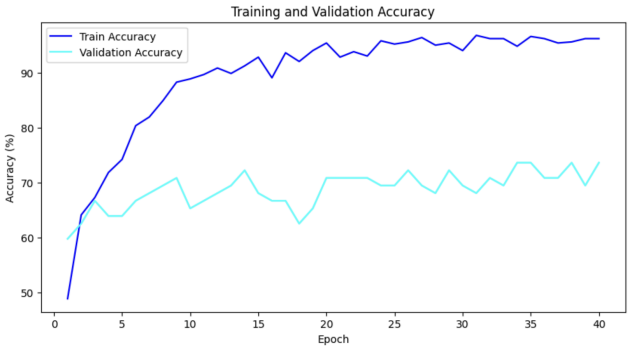

Fig. 7. Gráfica de Accuracy modelo 2

Tampoco se muestra una mejora notable en el accuracy de validación con el pasar de las épocas, al contrario, inclusive da una primera impresión de que es inclusive menor al de la iteración pasada..

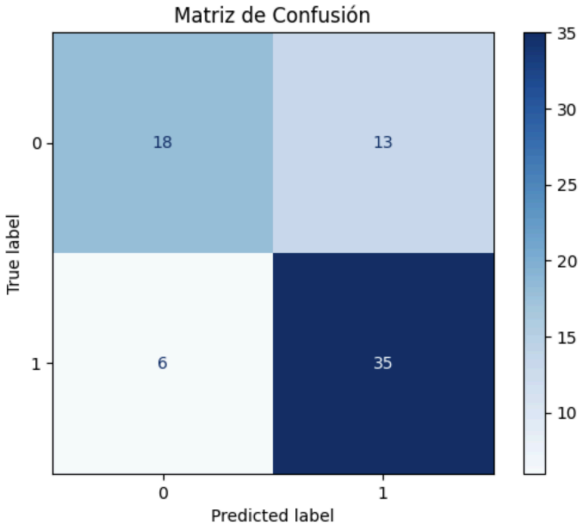

Fig. 8. Matriz de confusión de validación modelo 2

La matriz de confusión también muestra peores resultados.

Al hacer la evaluación con el conjunto de prueba y comparar los resultados podemos ver que además de existir overfitting, los valores de accuracy son aún más bajos, además de que no parecían mejorar con las épocas, por lo que descarté esas aumentaciones y regresé a las primeras.

Conjunto de datos	**Accuracy**
Train	97.82%
Validation	73.61%
Test	70.83%

C. Modelo 3

Considerando el número de imágenes con las que cuento y que el error constante ha sido el overfitting, pensé que el modelo podría estar influyendo, por lo que opté por utilizar una versión menos compleja, siendo EfficientNet-B0.

Obteniendo los siguientes resultados para el conjunto de validación respecto al de entrenamiento.

Fig. 9. Gráfica de pérdida modelo 3

Se puede observar a primera instancia que el error de validación decayó en mayor medida.

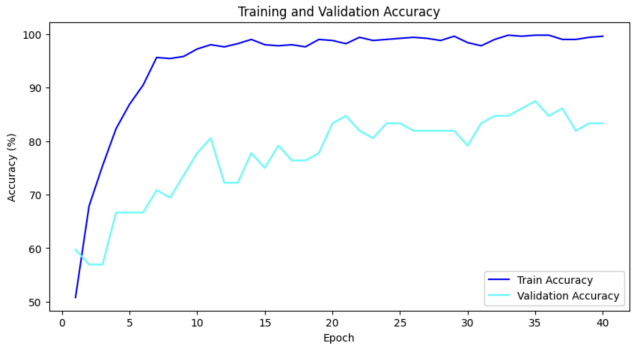

Fig. 10. Gráfica de Accuracy modelo 3

Se puede observar en la figura 10 que si bien sigue existiendo overfitting, este ya se encuentra en una menor cantidad.

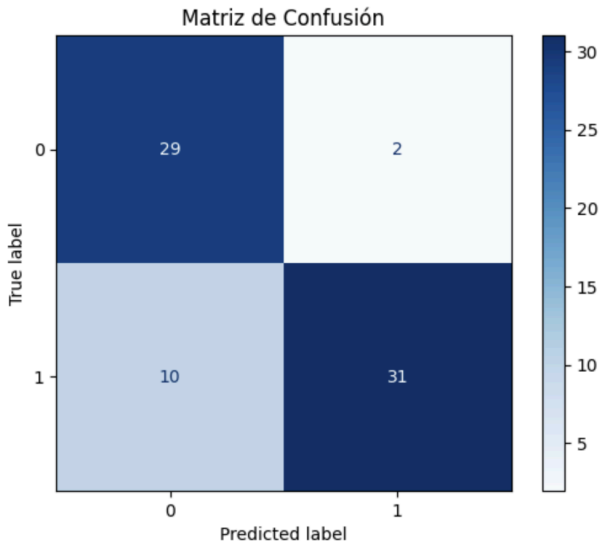

Fig. 11. Matriz de confusión de validación modelo 3

Por otra parte, en la matriz de confusión de validación se nota una mejora notable respecto a la comparación anterior, sin dejar una tendencia notable de solo elegir una clase.

Al comparar con el conjunto de prueba, vemos que al igual que en el primer modelo, casi se llega al 80%, la diferencia es que para el conjunto de validación si se superó ese límite, además, ahora si pareciera que con más épocas se podría obtener un mejor resultado. Por lo que decidí continuar con este modelo.

Conjunto de datos	**Accuracy**
Train	99.40%
Validation	83.61%
Test	77.78%

D. Modelo 4

Continuando con EfficientNet-B0, para lograr disminuir el overfitting, busqué implementar capas de dropout, por lo que cree una clase en la que se encuentra este modelo modificado, en donde se cambia la capa de clasificación final por más capas densas y dropouts. Quedando de la siguiente manera:

- EfficientNet-B0
- Capa Densa (nn.Linear con 128 neuronas)
- Dropout (p=0.5)
- Capa Intermedia (nn.Linear con 2 neuronas)
- Dropout (p=0.5)
- Output (Logits para 2 clases)

Obteniendo los siguientes resultados para el conjunto de validación respecto al de entrenamiento.

Fig. 12. Gráfica de pérdida modelo 4

En la figura 12 se aprecia que los valores de la función de pérdida cuentan con una distancia menor entre ambos conjuntos de datos.

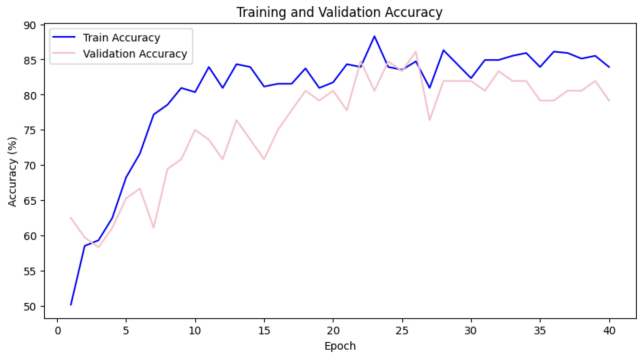

Fig. 13. Gráfica de Accuracy modelo 4

En la gráfica de la figura 13 ya se puede observar una mejora considerable en el aspecto del overfitting, sin embargo, ahora no se llegó a un valor muy alto para los valores de train, como se había estado logrando antes.

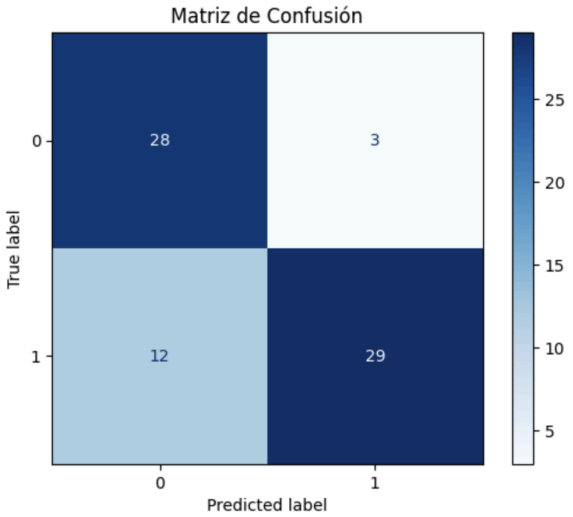

Fig. 14. Matriz de confusión de validación modelo 4

La matriz de confusión de validación se aprecia similar a los resultados anteriores.

Al comparar con el conjunto de prueba, vemos que el overfitting se redujo considerablemente, sin embargo, ahora parece empezarse a notar cierto nivel de underfitting.

Conjunto de datos	**Accuracy**
Train	88.60%
Validation	79.17%
Test	78.08%

Sin embargo, en las gráficas de pérdida y de accuracy, se nota que sí existe cierta tendencia a mejorar, por lo que aumenté el número de épocas a 100.

Obteniendo los siguientes resultados:

Fig. 15. Gráfica de pérdida modelo 4 a 100 épocas

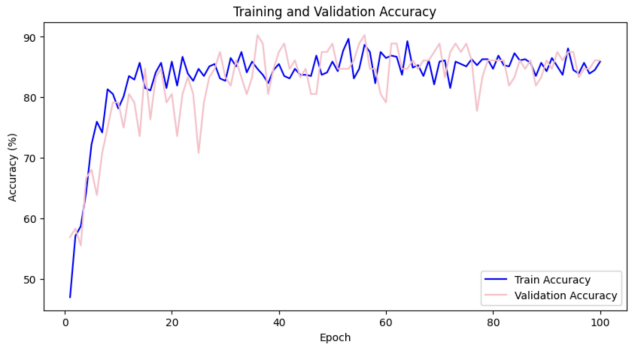

Fig. 16. Gráfica de pérdida modelo 4 a 100 épocas

Conjunto de datos	**Accuracy**
Train	92.82%
Validation	86.11%
Test	79.20%

Si bien se redujo la cantidad de overfitting y las épocas si ayudaron a obtener un mejor accuracy, lo cierto es que no resultaron en una mejora significativa respecto a las 40 épocas y teniendo en mente los datos de prueba, no se puede considerar como una solución relevante.

E. Modelo 5

Con el fin de evitar que se presente underfitting, reduje el porcentaje de dropout del 50% al 30% para las dos capas agregadas, además, pensando en los detalles que se pudieran estar perdiendo cambié la redimensión a 512x512.

Además, continué con las 100 épocas para poder observar de mejor manera la tendencia de los valores en el entrenamiento. Este modelo tardó un poco más de tiempo en su entrenamiento y se obtuvieron los siguientes resultados.

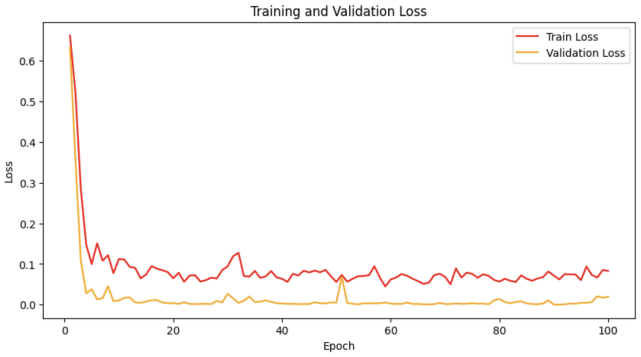

Fig. 17. Gráfica de pérdida modelo 5

Se puede notar una gráfica de pérdida mucho más acorde a lo que estaríamos buscando observar en un modelo preciso.

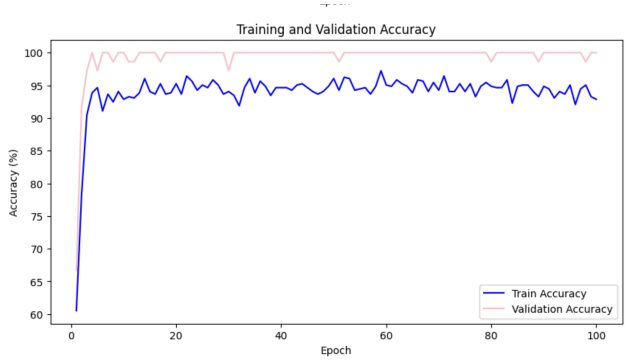

Fig. 18. Gráfica de Accuracy modelo 5

En la figura 18 se puede notar que el accuracy mejoró mucho en comparación con los otros modelos, ya que el valor que está más cerca del 100% corresponde al conjunto de validación, mientras que el de entrenamiento se encuentra rondando el 95%, por lo que no existe un overfitting considerable ni un underfitting que represente un porcentaje de error considerable.

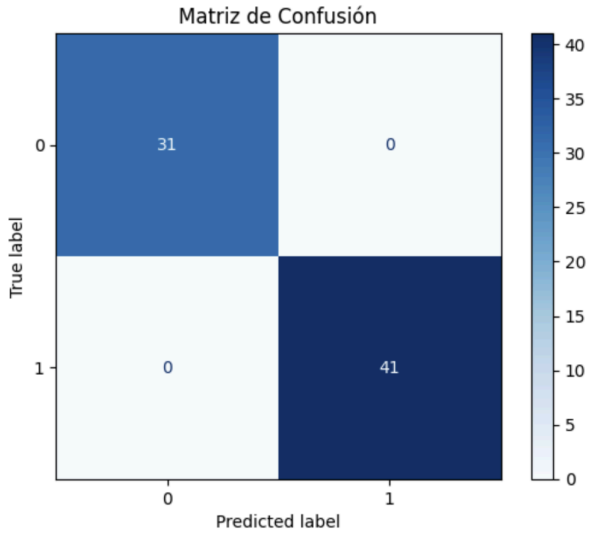

Fig. 19. Matriz de confusión de validación modelo 5

En la matriz de confusión de validación se muestra algo muy interesante y es que todos los valores se predijeron de manera correcta.

Además, al utilizar las funciones que nos brindan las métricas al evaluar los modelos, podemos ver que fue demasiado eficiente para todos los conjuntos de datos, por lo que será considerado como el modelo final.

Conjunto de datos	**Accuracy**
Train	99.81%
Validation	100%
Test	97.92%

8. Evaluación de los resultados

El modelo final resulta eficiente y preciso para la gran mayoría de casos, inclusive alcanzando el 100% de accuracy en el conjunto de validación.
Una de sus desventajas es que tarda un tiempo considerable en ser entrenado.
Aunque se puede considerar al modelo como fitting en su ajuste, la principal forma de asegurar su eficiencia sería agregando más imágenes al dataset, por otra parte, técnicas como K-fold cross validation podrían ayudar a evitar overfitting en los modelos iniciales, aunque aumentarían el tiempo de entrenamiento .

Referencias

1. IBM. (s. f.). *Deep learning*. IBM. https://www.ibm.com/mx-es/topics/deep-learning
2. Kaggle. (2024). *Cidaut AI Fake Scene Classification 2024*. https://www.kaggle.com/competitions/cidaut-ai-fake-scene-classification-2024/data
3. GeeksforGeeks. (2023). *Adam optimizer*. https://www.geeksforgeeks.org/adam-optimizer/