Primera Práctica: Distinguir entre caras Reales y Falsas

Alejandro Jesús González Santana, Joaquín Ibáñez Peñalva

January 2024

Contents

Github Repository:

---

1 Introducción

En este documento se tratará de crear una red neuronal, lo suficientemente potente como para distinguir en un conjunto de imágenes de caras que contienen imágenes falsas y verdaderas.[1].

2 Métodología

2.1 Elección de la red

Primero, es crucial examinar la arquitectura de la red utilizada. Después de realizar diversas pruebas, observamos que las redes neuronales diseñadas manualmente no lograban resultados satisfactorios para nuestro problema. Por esta

[1] Conjunto de datos: https://www.kaggle.com/datasets/ciplab/real-and-fake-face-detection

razón, optamos por comenzar con un modelo base, la resnet50 implementada en PyTorch [2]. Resnet-50 es una red preentrenada con el conjunto de datos ImageNet, que contiene más de un millón de imágenes etiquetadas, convirtiéndola en una excelente elección para problemas de clasificación de imágenes como el que estamos abordando. Pertenece a la familia de redes neuronales residuales profundas y es una variante de 50 capas.

Esta red acepta entradas de tamaño 224x224 píxeles, lo que implica que debemos redimensionar nuestras imágenes previamente. En nuestra implementación, utilizaremos esta red preentrenada y congelaremos la actualización de parámetros para las 3 capas más profundas. Esto tiene varios beneficios, como la reducción del tiempo de entrenamiento y la prevención del sobreajuste, ya que las capas más profundas permanecen más estables al captar detalles más generales y aceleran la convergencia.

Es crucial ajustar la última capa completamente conectada para adaptarla a nuestro problema de clasificación binaria (imagen real o falsa). Creamos una capa lineal con 512 neuronas, seguida de una función de activación ReLU y una capa de dropout que apaga cerca de la mitad de las neuronas durante el entrenamiento. Finalmente, la salida se determina mediante una capa con 512 entradas y 2 salidas, definiendo así la clase a la que pertenece la imagen.

2.2 Calcular métricas

En nuestra función de cálculo de métricas, llevamos a cabo una evaluación exhaustiva de nuestro modelo. Esta evaluación nos proporciona las etiquetas predichas para un conjunto de datos específico mediante el uso de nuestra red neuronal. Después de obtener todas las etiquetas reales y predichas, procedemos a calcular la matriz de confusión, que ofrece una representación visual de los resultados de la clasificación, donde los valores en la diagonal principal corresponden a las predicciones correctas.

Posteriormente, procedemos al cálculo del F1 score, una métrica que se fundamenta en los valores de la matriz de confusión y abarca tanto la precisión (que cuantifica las muestras correctamente clasificadas) como la exhaustividad (que evalúa la capacidad del modelo para capturar muestras positivas, en este caso caras falsas). El F1 score alcanza su valor máximo de 1 cuando existe un equilibrio perfecto entre precisión y exhaustividad, indicando la ausencia de falsos positivos y falsos negativos. Por otro lado, un valor de 0 señala la presencia de numerosos falsos positivos y falsos negativos.

Adicionalmente, realizamos el cálculo del recall, previamente discutido, que representa el número de muestras verdaderas positivas entre el total de muestras asignadas a la clase positiva, ya sean correctas o incorrectas.

[2] Resnet50: https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html

Finalmente, determinamos el porcentaje de muestras correctamente clasificadas, brindando así una medida adicional de la eficacia global del modelo.

2.3 Proceso de entrenamiento

Transformamos las imágenes a una resolución uniforme de 256 por 256 píxeles, incorporando variabilidad mediante rotación horizontal con un ángulo máximo de 15 grados. Para adaptar nuestros datos, aplicamos recortes aleatorios, reduciendo su tamaño a 224x224 píxeles y diversificando así las muestras. Además, introducimos ajustes en la saturación, brillo, tono y contraste para ampliar las características visuales. Posteriormente, convertimos las imágenes en tensores y normalizamos sus valores de píxeles. Este proceso de transformación es comúnmente utilizado en tareas de aprendizaje profundo para preparar conjuntos de datos de imágenes, mejorando la capacidad del modelo para generalizar y realizar predicciones precisas en diversas condiciones. Estas transformaciones forman parte de las técnicas de aumento de datos.

Luego, cargamos los datasets y destinamos un 80% de las muestras para entrenamiento y el resto para prueba utilizando la función random_split del paquete torch. Creamos un dataloader para cargar el conjunto de datos de entrenamiento con lotes de tamaño 32, y barajamos los datos en cada época para evitar que el modelo memorice el orden. No es necesario barajar los datos de prueba, ya que no se utilizan para el entrenamiento.

A continuación, trasladamos la red al dispositivo y utilizamos la función de pérdida CrossEntropyLoss junto con el optimizador Adam. Configuramos diez épocas de entrenamiento. Para cada muestra, antes de calcular el gradiente, aseguramos que los gradientes de los parámetros del modelo estén a cero, avanzamos por la red para obtener una predicción y calculamos la función de pérdida. Posteriormente, aplicamos la retropropagación con la función backward sobre la pérdida para actualizar los pesos. Utilizamos optimizer.step() para indicar que se utilizará el optimizador, y acumulamos la pérdida.

En cada época, evaluamos la red con net.eval(), obtenemos métricas como precisión, matriz de confusión, f1 y recall para los conjuntos de datos de entrenamiento y prueba. Luego, calculamos la media de la pérdida para el conjunto de datos de entrenamiento.

Después de completar todas las épocas, mostramos los resultados de nuestra red.

3 Resultados

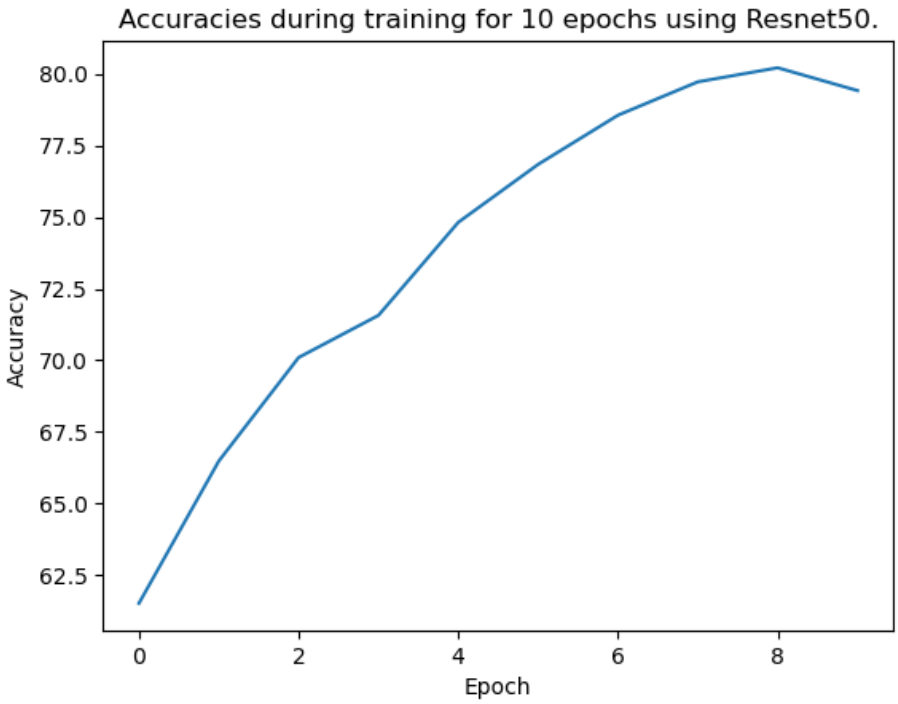

Figure 1: Accuracy a lo largo del entrenamiento

Como se evidencia, la red arroja resultados aceptables; alcanza un valor cercano al 80% en la octava iteración. No obstante, a partir de este punto, se observa un declive en los valores, sugiriendo posibles escenarios de sobreajuste o una tasa de aprendizaje demasiado elevada. Es crucial recordar que estos resultados corresponden al conjunto de entrenamiento, por lo que es necesario confirmar con el conjunto de datos de prueba.

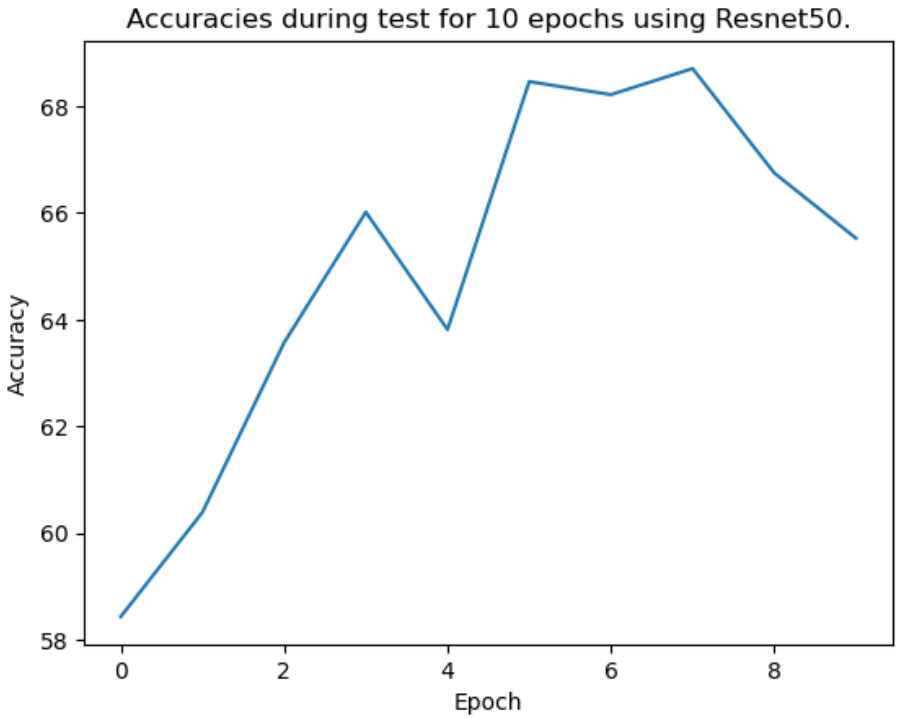

Figure 2: Accuracy a lo largo de la prueba

En el análisis de rendimiento, observamos que, en términos generales, los resultados para el conjunto de prueba son algo inferiores, alcanzando un valor máximo del 68%. A pesar de esta ligera disminución, aún podemos considerar estos resultados como aceptables, considerando que hay imágenes que son difíciles de diferenciar para el propio ojo humano.

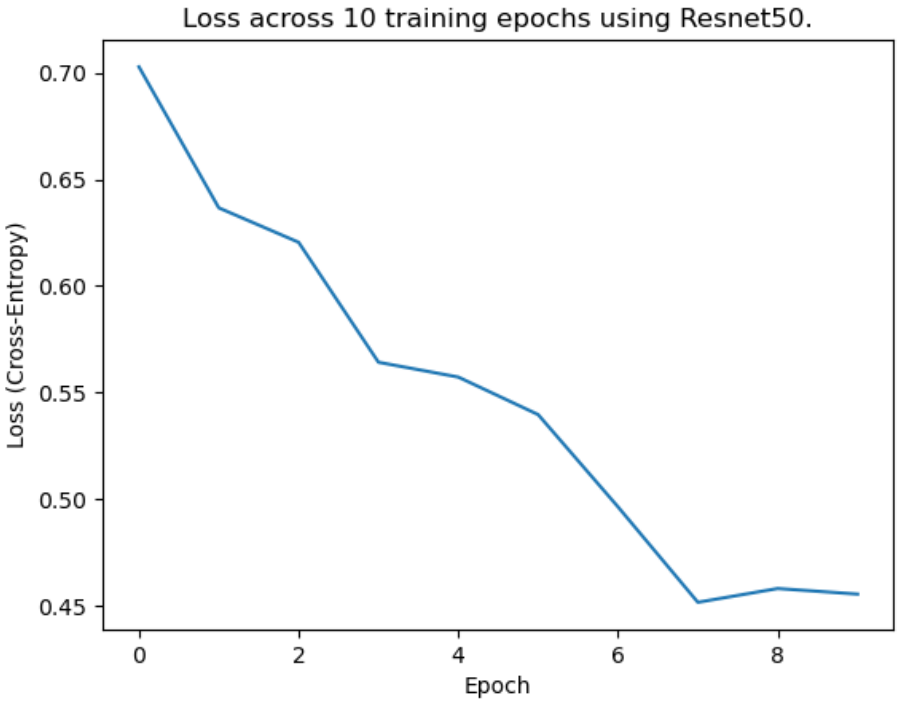

Figure 3: Accuracy a lo largo del entrenamiento

Observamos que la función de pérdida durante el entrenamiento disminuye rápidamente y se estabiliza cerca de 0.45, indicando que el modelo está clasificando de manera acertada, aunque no perfecta.

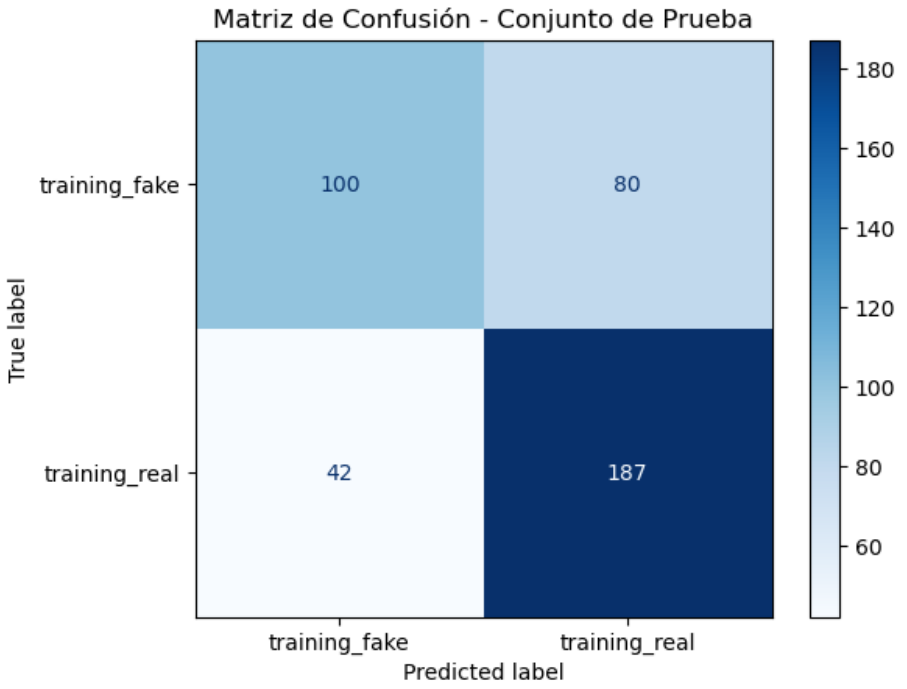

Figure 4: Matriz de confusión para test

En la matriz de confusión, observamos que se clasificaron correctamente un total de 287 muestras, mientras que 122 fueron clasificadas incorrectamente. El F1 score, que es 0.6225, sugiere un equilibrio razonable entre la clasificación de muestras y la detección de caras falsas. En este caso, nuestro recall (o exhaustividad) es del 70.42%, indicando que del total de muestras clasificadas como caras falsas, el 70.42% eran realmente caras falsas. Esto podría sugerir que el modelo tiende más a clasificar las muestras como reales, dado que hay una mayor proporción de estas en comparación con las muestras falsas, y la mayoría de los fallos ocurren en este caso.