Clasificación de género usando Redes Neuronales Convolucionales

Colque Gabriela, Estrada Piero, Ferreyra Sebastián , Paredes Maxwel, Poma Soía

gabriela.colque.u@uni.edu.pe, sebastian.ferreyra.c@uni.edu.pe, piero.estrada.c@uni.pe, maxwel.paredes.l@uni.pe, sofia.poma.a@uni.pe

Escuela Profesional de Ciencias de la Computación

Facultad de Ciencias

Universidad Nacional de Ingeniería

Índice

1 Introducción

La detección y clasificación de género en tiempo real ha sido un tema de interés en el campo de la visión por computadora y el aprendizaje automático. En este informe, se presenta un proyecto que tiene como objetivo implementar un sistema capaz de detectar y clasificar el género de las personas en tiempo real utilizando la cámara.

El proyecto se basa en el uso de un modelo de red neuronal convolucional (CNN) entrenado previamente para realizar la clasificación de género. Se utilizó un conjunto de datos etiquetados que consiste en imágenes de rostros de hombres y mujeres, lo que permitió entrenar el modelo para reconocer y distinguir las características asociadas a cada género.

El desarrollo del proyecto involucró varias etapas, desde la preparación del conjunto de datos hasta la implementación del código para la detección y clasificación en tiempo real. Se utilizó el clasificador de cascada Haar para la detección de rostros en las imágenes capturadas por la cámara, y posteriormente se recortó la sección de la cara para su procesamiento.

Durante el entrenamiento del modelo, se ajustaron los pesos mediante iteraciones (epochs) utilizando un optimizador y una función de pérdida adecuada. Se evaluaron métricas como la pérdida y la precisión en conjuntos de datos de entrenamiento y validación para supervisar y mejorar el rendimiento del modelo.

Los resultados obtenidos durante la implementación del sistema son prometedores. El programa es capaz de detectar caras en tiempo real y clasificar el género con precisión. Sin embargo, se deben considerar posibles desafíos y limitaciones, como la variabilidad en la iluminación y las expresiones faciales, que pueden afectar el rendimiento del sistema.

En conclusión, este proyecto demuestra la viabilidad de implementar un sistema de detección y clasificación de género en tiempo real utilizando técnicas de visión por computadora y aprendizaje automático. Los resultados obtenidos son un paso hacia aplicaciones más amplias, como la identificación de grupos demográficos en imágenes y videos, y podrían ser útiles en áreas como el análisis de audiencia y la seguridad.

1.1. Definición del Problema

El problema consiste en desarrollar un sistema que pueda detectar y clasificar el género de una persona en tiempo real utilizando técnicas de visión por computadora y aprendizaje automático. El desafío radica en lograr una detección precisa de rostros y una clasificación confiable del género en diferentes situaciones y condiciones, considerando variables como iluminación, fondo y expresiones faciales.

1.2. Objetivos

- Desarrollar un sistema de detección y clasificación de género en tiempo real utilizando técnicas de visión por computadora y aprendizaje automático.
- Evaluar y validar el desempeño del sistema mediante pruebas exhaustivas utilizando conjuntos de datos de prueba, calculando la pérdida media y el accuracy (exactitud) del modelo, para garantizar la precisión y confiabilidad del sistema en la detección y clasificación de género.

1.3. Estado del Arte

La clasificación de género utilizando Redes Neuronales Convolucionales ha sido un tema de investigación importante en el campo de la visión computacional y el aprendizaje profundo. En los últimos años, se han realizado avances significativos en este campo, lo que ha permitido obtener resultados prometedores.

Una de las arquitecturas CNN más utilizadas para la clasificación de género es VGGFace, propuesta por Parkhi y sus colaboradores en 2015. VGGFace es una extensión de la red VGG16, preentrenada en un gran conjunto de datos de rostros, que ha demostrado excelentes capacidades de extracción de características. Al ajustar las últimas capas de VGGFace para la clasificación binaria de género, se ha logrado una precisión destacada en la identificación del género de un individuo a partir de su imagen facial.

Además de VGGFace, otras arquitecturas CNN como ResNet y MobileNet también se han aplicado con éxito en la clasificación de género. Estos modelos preentrenados en grandes conjuntos de datos, como ImageNet, pueden ser adaptados para la tarea específica de clasificación de género, lo que acelera el proceso de entrenamiento y mejora el rendimiento.

Para entrenar y evaluar estos modelos, se han utilizado diversos conjuntos de datos, como el conjunto de datos CelebA y el conjunto de datos Adience. Estos conjuntos de datos contienen imágenes faciales etiquetadas con el género correspondiente, lo que permite entrenar a los modelos de manera supervisada y evaluar su precisión en la clasificación.

Aunque las CNN han demostrado un rendimiento sólido en la clasificación de género, todavía existen desafíos en esta área. Por ejemplo, la clasificación puede verse afectada por variaciones en la iluminación, la expresión facial y las oclusiones. Además, la clasificación precisa puede ser más difícil en imágenes de baja resolución o con rostros parcialmente cubiertos.

En los últimos años, también se ha investigado enfoques basados en el aprendizaje por transferencia, donde modelos preentrenados en una tarea relacionada, como la detección de rostros o la clasificación de edad, se utilizan para mejorar el rendimiento de la clasificación de género. Estos enfoques han mostrado resultados prometedores y han contribuido a la mejora de la precisión en la clasificación.

A continuacion se muestran algunos de los estudios realizados en los útimos años:

En resumen, la clasificación de género utilizando Redes Neuronales Convolucionales ha experimentado avances significativos en los últimos años. Diversas arquitecturas CNN han demostrado un rendimiento destacado en esta tarea. A pesar de los avances, todavía existen desafíos por superar, lo que ofrece oportunidades para futuras investigaciones en el campo de la clasificación de género utilizando aprendizaje profundo.

Cuadro 1: Resumen de estudios sobre clasificación de género usando CNNs

Título del estudio	Año	Técnicas utilizadas
Deep Learning Face Attributes in the Wild (2014)	2014	- CNN profunda entrenada con un conjunto de datos grande y diverso. - Técnicas de aprendizaje profundo y representación de características. - Enfoque de transferencia de aprendizaje.
Deep Expectation of Real and Apparent Age from a Single Image Without Facial Landmarks (2015)	2015	- CNN profunda para la clasificación de género. - No se requieren puntos de referencia faciales explícitos. - Enfoque de transferencia de aprendizaje con conocimientos previos de un conjunto de datos grande.
Gender Classification from Facial Images Using CNNs (2015)	2015	- CNN para la clasificación de género. - Técnicas de preprocesamiento y extracción de características. - Ajuste de contraste y normalización de histograma adaptativo.
Gender Classification of Human Faces Using CNNs (2017)	2017	- Modelo de CNN para la clasificación de género. - Inicialización de pesos con aprendizaje previo (pretrained weights). - Ajuste fino (fine-tuning) de una red preentrenada en un conjunto de datos masivo.
Gender Recognition from Face Images with Occlusions (2018)	2018	- Enfoque robusto para la clasificación de género en presencia de occlusiones parciales en las imágenes faciales. - Técnicas de data augmentation para generar imágenes sintéticas con occlusiones. - Entrenamiento de una CNN con datos aumentados para mejorar la robustez del modelo ante occlusiones.

2 Fundamento Teórico

2.1. ¿Qué es la visión por computadora?

La visión por computadora es un campo de estudio que se enfoca en el desarrollo de algoritmos y técnicas para la interpretación de imágenes y videos por parte de las computadoras. Implica la extracción de información útil y significativa de las imágenes, como características visuales, objetos detectados y clasificación de patrones como se muestra en la figura 1.

Figura 1: Visión por computadora

2.2. ¿Qué es el aprendizaje automático?

El aprendizaje automático es una rama de la inteligencia artificial que se ocupa del desarrollo de algoritmos y modelos que permiten a las computadoras aprender y mejorar su rendimiento a partir de datos. En el contexto de este proyecto, se utiliza el aprendizaje supervisado para entrenar un modelo que pueda aprender a clasificar el género de una persona a partir de imágenes faciales.

- Aprendizaje supervisado

En la figura 2 se muestra el flujo de este tipo de aprendizaje, donde se proporciona al algoritmo de aprendizaje un conjunto de datos de entrenamiento que contiene tanto características como las etiquetas o respuestas correspondientes. El objetivo del algoritmo es aprender una función que mapee las características a las etiquetas, de modo que pueda realizar predicciones precisas en nuevos datos. Ejemplos de algoritmos de aprendizaje supervisado incluyen las redes neuronales, los árboles de decisión y las máquinas de vectores de soporte (SVM).

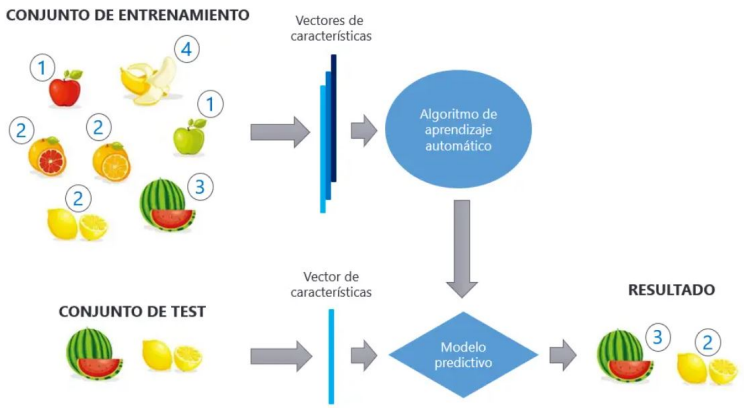

Figura 2: Ejemplo de aprendizaje supervisado

- Aprendizaje profundo

El aprendizaje profundo es una rama del aprendizaje automático que se enfoca en la construcción y entrenamiento de redes neuronales profundas. Estas redes neuronales tienen múltiples capas ocultas que les permiten aprender representaciones de datos cada vez más abstractas y de alto nivel. El aprendizaje profundo ha demostrado ser especialmente efectivo en tareas de visión por computadora, procesamiento del lenguaje natural y reconocimiento de voz, entre otros.

2.3. ¿ Qué son las Redes Neuronales Convolucionales?

Las redes neuronales convolucionales son un tipo de arquitectura de aprendizaje profundo especialmente diseñada para el procesamiento de imágenes. Estas redes han revolucionado el campo de la visión por computadora y han demostrado un rendimiento destacado en tareas como el reconocimiento de objetos, la detección de rostros y la segmentación de imágenes.

Como se puede observar en la figura 3, la principal característica de las CNN es su capacidad para aprender y extraer automáticamente características relevantes de las imágenes. Esto se logra mediante la aplicación de operaciones de convolución, que implican deslizar filtros sobre la imagen para detectar patrones visuales locales. Posteriormente, se utilizan capas de pooling para reducir la dimensionalidad y mejorar la eficiencia computacional.

Las CNN también están compuestas por capas de activación, como la función ReLU, y capas totalmente conectadas al final de la red para realizar la clasificación o regresión final.

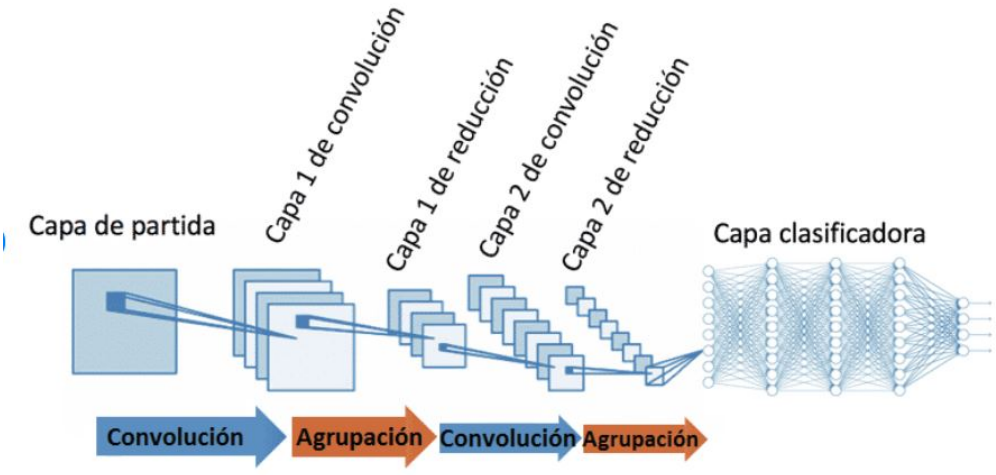

Figura 3: Esquema de Red Neuronal Convolucional.

2.3.1. Elementos

- Neuronas

  Son unidades fundamentales que reciben una entrada, aplican una función de activación y generan una salida. Cada neurona tiene una ponderación asociada a sus conexiones, que determina la influencia de la entrada en la salida.

- Capas

  Las neuronas se organizan en capas dentro de la red. Una red neuronal típica tiene una capa de entrada, una o varias capas ocultas y una capa de salida. Las capas ocultas permiten el aprendizaje de representaciones intermedias más complejas a medida que los datos se propagan a través de la red.

- Pesos

  Son parámetros ajustables que se asignan a las conexiones entre las neuronas. Estos pesos determinan la influencia de una neurona en la salida de las neuronas siguientes. Durante el entrenamiento de la red, los pesos se ajustan iterativamente para minimizar el error de predicción.

- Función de Activación

  Es una función no lineal que se aplica a la suma ponderada de las entradas de una neurona.Ejemplos comunes de funciones de activación incluyen la función sigmoide, la función ReLU (Rectified Linear Unit) y la función softmax.

- Función de Pérdida

  Es una medida que evalúa la discrepancia entre las predicciones de la red y los valores reales. La función de pérdida guía el proceso de aprendizaje al cuantificar el error cometido por la red y ajustar los pesos para reducir ese error.

- Algoritmo de Optimización

  Es el algoritmo utilizado para ajustar los pesos de la red neuronal durante el entrenamiento. El algoritmo busca minimizar la función de pérdida y encontrar los valores óptimos de los pesos para mejorar el rendimiento de la red.

- Algoritmo de Aprendizaje

  Es el método utilizado para ajustar las ponderaciones de las conexiones en función de los datos de entrenamiento. Uno de los algoritmos más comunes es el descenso del gradiente, que minimiza una función de pérdida calculando las derivadas parciales de las ponderaciones.

2.3.2. Construcción

La construcción de una red neuronal implica varios pasos esenciales que se enumeran a continuación:

- Definir arquitectura

  Consiste en definir la estructura de la red neuronal, que incluye el número de capas, el tipo de capas (capas convolucionales, capas de agrupación, capas completamente conectadas, etc.), el número de neuronas en cada capa y la función de activación utilizada.

- Preparar los datos

  Es importante preparar los datos de entrada para la red neuronal. Esto implica realizar una limpieza de datos, normalización, transformación de variables y división de los datos en conjuntos de entrenamiento, validación y prueba.

- Inicializar lo pesos y sesgos

  Los pesos y sesgos iniciales de las conexiones entre las neuronas se inicializan generalmente de forma aleatoria o utilizando algún método específico. Estos valores se irán ajustando durante el entrenamiento de la red.

- Calcular la función de pérdida

  Se compara con los valores reales y se calcula una medida de error utilizando una función de pérdida adecuada para el problema específico (como el error cuadrático medio o la entropía cruzada).

- Evaluación y Ajuste

  Una vez finalizado el entrenamiento, se evalúa el rendimiento de la red utilizando los datos de prueba. Si es necesario, se pueden realizar ajustes adicionales en la arquitectura de la red o en los parámetros de entrenamiento para mejorar el rendimiento.

La construcción de una red neuronal implica definir la arquitectura de la red, es decir, el número de capas, el número de neuronas en cada capa y las funciones de activación utilizadas. Luego, la red se entrena con un conjunto de datos de entrenamiento, ajustando las ponderaciones de las conexiones mediante el algoritmo de aprendizaje seleccionado. Una vez entrenada, la red se puede utilizar para realizar predicciones o clasificar nuevos datos.

2.3.3. Convolución

La operacion de convolución es un proceso que combina dos funciones para generar una tercera función que representa cómo una de las funciones influye en la otra a medida que se superponen.

La convolución de dos funciones $f(t)$ y $g(t)$ se denota como $(f * g)(t)$ y se define matemáticamente como:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau)\, d\tau$$

donde $f(\tau)$ y $g(t - \tau)$ son las funciones a convolucionar, y τ es la variable de integración.

Consideremos las funciones $f(t) = \cos(t)$ y $g(t) = t^2$. La convolución de estas funciones se denota como $(f * g)(t)$ y se calcula como:

$$(f * g)(t) = \int_{-\infty}^{\infty} \cos(\tau) \cdot (t - \tau)^2\, d\tau$$

Simplificando la expresión, podemos expandir el cuadrado y obtener:

$$(f * g)(t) = \int_{-\infty}^{\infty} \cos(\tau) \cdot (t^2 - 2t\tau + \tau^2)\, d\tau$$

Distribuyendo y agrupando términos, obtenemos:

$$(f * g)(t) = t^2 \int_{-\infty}^{\infty} \cos(\tau)\, d\tau - 2t \int_{-\infty}^{\infty} \tau \cdot \cos(\tau)\, d\tau + \int_{-\infty}^{\infty} \tau^2 \cdot \cos(\tau)\, d\tau$$

En el ámbito de las Redes Neuronales Convolucionales, la convolución es una operación que utiliza filtros para aplicar una transformación a la imagen de entrada, resaltando patrones locales y generando mapas de características que capturan información relevante para la tarea de clasificación de imágenes.

La convolución de imágenes tiene varias ventajas. En primer lugar, reduce la complejidad computacional al restringir el procesamiento a regiones locales de la imagen. Esto permite un procesamiento eficiente de imágenes de alta resolución. Además, la convolución ayuda a capturar características locales importantes, como bordes, texturas o patrones visuales, al aplicar diferentes filtros especializados.

2.3.4. Función de activación ReLU

RELU (Rectified Linear Unit) es una función de activación muy popular y ampliamente utilizada en las redes neuronales. Su característica principal es que mapea los valores de entrada negativos a cero y mantiene los valores positivos sin cambios.

La función de activación ReLU (Rectified Linear Unit) se define como:

$$\text{ReLU}(x) = \text{máx}(0, x)$$

donde x es la entrada a la función.

ReLU tiene varias ventajas. En primer lugar, es computacionalmente eficiente. Además, ayuda a prevenir el problema del desvanecimiento del gradiente, que es común en funciones de activación como la sigmoide. Además, la no linealidad de ReLU permite que la red aprenda relaciones complejas en los datos.

2.3.5. Max-Pooling

La capa de pooling es una operación común en las redes neuronales convolucionales (CNNs) que se utiliza para reducir el tamaño espacial de los mapas de características generados por la capa de convolución. Además, ayuda a controlar el sobreajuste al proporcionar una forma de generalización y resistencia a pequeñas variaciones en la posición de las características.

Esto se logra mediante la subdivisión de la imagen en regiones solapadas y la reducción de cada de estas regiones a un solo valor representativo.

Los tipos de Pooling más comúnes son Max-Pooling y Average Pooling :

- El objetivo de Max-Pooling es reducir el tamaño espacial manteniendo las características más importantes de cierta región.
- El objetivo de Average-Pooling es reducir el tamaño espacial manteniendo una representación más generalizada de la región.

Dentro de las capas Pooling se tienen 2 parámetros: tamaño del filtro y pasos.

El tamaño del filtro determina la dimensión de la región de la imagen a considerar, mientras que el paso controla el desplazamiento de la ventana durante el proceso.

Para este caso, se usara Max-Pooling, pues se sabe que una de las diferencias claves entre los rostros de hombres y mujeres es la acentuación de ciertos rasgos faciales.

2.3.6. Backpropagation

El algoritmo de backpropagation, también conocido como retropropagación, es un algoritmo de aprendizaje utilizado en redes neuronales artificiales para entrenar modelos de aprendizaje supervisado. Especifica cómo ajustar los pesos de las conexiones entre las neuronas para minimizar el error entre las salidas deseadas y las salidas producidas por la red.

El proceso de backpropagation se divide en dos fases: la propagación hacia adelante (forward propagation) y la propagación hacia atrás (backward propagation).

1. Propagación hacia adelante (Forward Propagation)
    - Se toma un conjunto de datos de entrenamiento y se introduce en la red neuronal.
    - Los datos se propagan a través de las capas ocultas hacia la capa de salida, calculando las salidas de cada neurona.
    - Se compara la salida obtenida con la salida esperada y se calcula el error utilizando una función de costo, como el error cuadrático medio.
2. Propagación hacia atrás (Backward Propagation)
    - El error se propaga hacia atrás a través de la red, calculando la contribución de cada peso al error total.
    - Se ajustan los pesos de las conexiones para reducir el error. Esto se logra utilizando el gradiente descendente, que indica la dirección en la que deben ajustarse los pesos para minimizar el error.
    - Se utiliza la regla de la cadena para calcular el gradiente de error con respecto a los pesos en cada capa.
    - Se actualizan los pesos utilizando una tasa de aprendizaje que controla la magnitud de los ajustes realizados en cada iteración.

El proceso de propagación hacia adelante y hacia atrás se repite varias veces (llamadas iteraciones o épocas) hasta que el error se minimiza o alcance un nivel aceptable. Esto implica pasar por todos los datos de entrenamiento en cada iteración.

El algoritmo de backpropagation es ampliamente utilizado debido a su eficiencia y capacidad para entrenar redes neuronales con múltiples capas ocultas. Sin embargo, es importante destacar que el algoritmo de backpropagation requiere una función de activación diferenciable y puede sufrir de problemas como el estancamiento en mínimos locales o el sobreajuste si no se utilizan técnicas adicionales, como la regularización o el descenso de gradiente estocástico.

2.3.7. Adam

Adam (Adaptive Moment Estimation) es un algoritmo de optimización utilizado comúnmente en el entrenamiento de redes neuronales. Combina los conceptos del método del gradiente descendente estocástico (SGD) y del método de RMSprop para adaptar las tasas de aprendizaje de forma dinámica para cada parámetro. A continuación se muestra el esquema básico del algoritmo Adam:

1. Inicialización de parámetros
    - Se inicializan los parámetros del modelo, como los pesos y los sesgos.
    - Se inicializan los momentos de primer orden (m) y de segundo orden (v) para cada parámetro a cero.
    - Se inicializa un contador (t) para llevar un seguimiento del número de pasos de optimización.
2. Cálculo del gradiente
    - Se selecciona un lote de datos de entrenamiento.
    - Se calcula el gradiente de la función de pérdida con respecto a los parámetros del modelo utilizando el lote de datos seleccionado.
3. Actualización de los momentos
    - Se actualiza el momento de primer orden (m) utilizando una combinación de decaimiento exponencial del gradiente actual y los momentos anteriores.
    - Se actualiza el momento de segundo orden (v) de manera similar, pero utilizando los cuadrados del gradiente.
    - Estos pasos se realizan para cada parámetro del modelo.
4. Corrección de sesgo
    - Los momentos (m) y (v) se corrigen para compensar su sesgo inicial hacia cero debido a la inicialización en cero.
    - Esto se hace dividiendo los momentos por una corrección basada en el contador (t).
5. Actualización de los parámetros
    - Se actualizan los parámetros del modelo utilizando los momentos adaptados y una tasa de aprendizaje global.
    - La tasa de aprendizaje determina el tamaño del paso de optimización y se multiplica por los momentos adaptados antes de aplicar las actualizaciones.

6. Actualización del contador
    - Se incrementa el contador (t) en uno.

Estos pasos se repiten para cada lote de datos de entrenamiento hasta que se complete una época de entrenamiento. El algoritmo Adam es conocido por su capacidad para adaptar las tasas de aprendizaje de forma automática y realizar ajustes más precisos en los parámetros del modelo. Esto lo logra al mantener estimaciones de momento de primer y segundo orden para cada parámetro y utilizando correcciones de sesgo para evitar sesgos iniciales.

2.3.8. Gradiente

El gradiente es una medida vectorial que indica la dirección y magnitud del cambio máximo de una función en un punto dado. En el contexto del aprendizaje automático y la optimización, el gradiente se refiere específicamente al vector de derivadas parciales de una función de pérdida con respecto a los parámetros de un modelo.

El cálculo del gradiente es fundamental para muchos algoritmos de optimización, como el descenso de gradiente, que se utiliza para ajustar los parámetros de un modelo de manera iterativa y reducir el error o la función de pérdida.

Formalmente, dado un modelo con parámetros θ y una función de pérdida $J(\theta)$, el gradiente se calcula como:

$$\nabla J(\theta) = \left(\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \ldots, \frac{\partial J}{\partial \theta_n} \right)$$

Donde $\nabla J(\theta)$ representa el gradiente de la función de pérdida J en el punto θ, y $\frac{\partial J}{\partial \theta_i}$ representa la derivada parcial de J con respecto al i-ésimo parámetro θ_i.

El gradiente indica la dirección en la que la función de pérdida aumenta más rápidamente, por lo que el descenso de gradiente utiliza esta información para actualizar los parámetros en la dirección opuesta al gradiente, lo que tiende a minimizar la función de pérdida.

En el aprendizaje automático, el cálculo del gradiente se realiza a menudo mediante el uso de técnicas de diferenciación automática, que permiten calcular de manera eficiente las derivadas parciales de una función con respecto a sus variables.

Es importante destacar que el gradiente puede ser positivo o negativo según la dirección del cambio máximo de la función. Además, el tamaño del gradiente indica la magnitud del cambio, de modo que un gradiente grande indica un cambio rápido en la función de pérdida, mientras que un gradiente pequeño indica un cambio más gradual.

2.3.9. Conjunto de Datos

Los conjuntos de datos se dividen típicamente en tres conjuntos principales: entrenamiento, validación y prueba. Estas divisiones se utilizan para entrenar, ajustar y evaluar un modelo de aprendizaje automático. A continuación, se describe cada uno de estos conjuntos:

1. Conjunto de entrenamiento
    - El conjunto de entrenamiento es utilizado para entrenar el modelo de aprendizaje automático.

- Consiste en un conjunto de ejemplos de entradas (características) y sus correspondientes salidas deseadas (etiquetas).
- Durante el entrenamiento, el modelo ajusta sus parámetros utilizando los ejemplos de entrada y las salidas deseadas para minimizar la función de pérdida.
- Cuanto más grande y representativo sea el conjunto de entrenamiento, mejor podrá generalizar el modelo.

2. Conjunto de validación

   - El conjunto de validación se utiliza para ajustar los hiper parámetros del modelo y evaluar su rendimiento durante el entrenamiento.
   - Se utiliza para medir el rendimiento del modelo en datos no vistos durante el entrenamiento y ayudar a evitar el sobreajuste.
   - Después de cada época o un número determinado de iteraciones, se evalúa el modelo utilizando el conjunto de validación y se ajustan los hiper parámetros (como la tasa de aprendizaje, el tamaño del lote, etc.) para mejorar el rendimiento.
   - Esto se repite hasta que el modelo alcance el rendimiento óptimo en el conjunto de validación.

3. Conjunto de prueba

   - El conjunto de prueba se utiliza para evaluar el rendimiento final del modelo después de haber sido entrenado y ajustado.
   - Es un conjunto de datos independiente que el modelo nunca ha visto durante el entrenamiento ni durante la validación.
   - Se utiliza para estimar el rendimiento del modelo en datos no vistos y proporcionar una medida imparcial de su capacidad de generalización.
   - El conjunto de prueba se utiliza típicamente solo una vez, después de que el modelo se haya finalizado, para evitar sesgos en la evaluación y comparación de modelos.

Es importante mencionar que es esencial mantener los conjuntos de entrenamiento, validación y prueba completamente independientes entre sí para evitar la filtración de información y garantizar una evaluación justa y precisa del modelo. Además, se deben tener en cuenta consideraciones como el tamaño de los conjuntos y la distribución de los datos para obtener resultados confiables.

2.3.10. Batch

El término "batch"se refiere a un grupo o conjunto de ejemplos de entrenamiento que se procesan simultáneamente en una pasada de entrenamiento. En lugar de actualizar los parámetros del modelo para cada ejemplo individualmente, se realiza una actualización basada en el promedio del gradiente calculado en el conjunto de ejemplos del batch. El tamaño del batch, también conocido como tamaño del lote (batch size), es un hiper parámetro que determina cuántos ejemplos se incluyen en cada batch. Algunos de los tamaños de batch más comunes son 32, 64, 128, 256, pero puede variar según el problema y la disponibilidad de recursos computacionales.

El uso de batches tiene varias ventajas:

1. Eficiencia computacional
    - Procesar un batch de ejemplos a la vez en lugar de cada ejemplo individualmente puede aprovechar mejor los recursos computacionales, ya que las operaciones se pueden realizar de manera paralela y se reducen las llamadas a las funciones de optimización.
2. Estabilidad del gradiente
    - Al calcular el gradiente en función de un conjunto de ejemplos, en lugar de uno solo, se reduce la variabilidad de los gradientes individuales, lo que puede ayudar a estabilizar y mejorar el proceso de optimización.
3. Uso eficiente de la memoria
    - Al procesar los ejemplos en batches, se puede reducir la cantidad de memoria necesaria para almacenar los gradientes y las variables intermedias en el cálculo del gradiente.

Es importante tener en cuenta que el tamaño del batch es un compromiso entre eficiencia y precisión. Un tamaño de batch más grande puede ser más eficiente computacionalmente, pero también puede requerir más memoria y potencialmente reducir la capacidad del modelo para generalizar. Por otro lado, un tamaño de batch más pequeño puede proporcionar actualizaciones de los parámetros más frecuentes, pero puede ser más lento y menos eficiente.

El tamaño del batch se elige en función de consideraciones prácticas, como la disponibilidad de recursos computacionales, la naturaleza del problema y las características del conjunto de datos. Es común realizar experimentos con diferentes tamaños de batch para encontrar el equilibrio adecuado entre eficiencia y precisión en un problema específico.

2.3.11. Taza de Aprendizaje

La tasa de aprendizaje (learning rate en inglés) es un hiperparámetro fundamental en los algoritmos de optimización utilizados en el entrenamiento de modelos de aprendizaje automático. Determina el tamaño del paso de actualización de los parámetros del modelo durante el proceso de optimización.

En el contexto del descenso de gradiente, la tasa de aprendizaje especifica cuánto se ajustan los parámetros en la dirección opuesta al gradiente calculado. Un valor alto de tasa de aprendizaje puede llevar a actualizaciones más grandes en cada paso, lo que puede hacer que el proceso de optimización sea más rápido, pero también puede llevar a oscilaciones y dificultades para converger hacia el mínimo global. Por otro lado, un valor bajo de tasa de aprendizaje puede hacer que el proceso sea más lento y puede quedarse estancado en mínimos locales.

Seleccionar una tasa de aprendizaje adecuada es crucial para obtener un buen rendimiento del modelo. Algunas consideraciones y técnicas comunes relacionadas con la tasa de aprendizaje son las siguientes:

1. Ajuste manual
    - Inicialmente, se puede realizar un ajuste manual de la tasa de aprendizaje y observar cómo se comporta el modelo. Se pueden probar diferentes valores y ajustarlos según la experiencia y el conocimiento del problema.
2. Decaimiento de la tasa de aprendizaje
    - Se puede reducir gradualmente la tasa de aprendizaje a medida que avanza el entrenamiento. Esto puede ser útil para permitir una exploración inicial más amplia y un ajuste más preciso a medida que se acerca al mínimo global.
3. Reglas de ajuste adaptativo
    - En lugar de fijar una tasa de aprendizaje constante, existen métodos que ajustan automáticamente la tasa de aprendizaje en función de la respuesta del modelo durante el entrenamiento. Algunos ejemplos populares son Adam, RMSprop y Adagrad.
4. Búsqueda de hiper parámetros
    - En algunos casos, se puede realizar una búsqueda sistemática de la tasa de aprendizaje junto con otros hiper parámetros para encontrar la combinación óptima que minimice la función de pérdida o mejore el rendimiento del modelo.

La elección de la tasa de aprendizaje depende de factores como la naturaleza del problema, el tamaño y la calidad del conjunto de datos, así como del algoritmo de optimización utilizado. Experimentar y ajustar la tasa de aprendizaje es una parte esencial del proceso de entrenamiento del modelo y puede requerir iteraciones y pruebas para obtener los mejores resultados.

2.3.12. Número de épocas

El número de épocas es otro hiper parámetro importante en el entrenamiento de modelos de aprendizaje automático. Una época se refiere a una pasada completa del conjunto de datos de entrenamiento durante el proceso de optimización. En cada época, se presentan todos los ejemplos del conjunto de entrenamiento al modelo.

El número de épocas especifica cuántas veces se repetirá el proceso de entrenamiento completo. Determinar el número adecuado de épocas es esencial para obtener un modelo bien ajustado sin caer en el sobreajuste (overfitting) o el subajuste (underfitting).

Aquí hay algunas consideraciones sobre el número de épocas:

1. Sobreajuste (Overfitting)
    - Si se entrena el modelo durante demasiadas épocas, existe el riesgo de que el modelo se ajuste demasiado a los datos de entrenamiento y no pueda generalizar bien a nuevos datos. Esto se conoce como sobreajuste. El modelo puede comenzar a memorizar los ejemplos de entrenamiento en lugar de aprender patrones y relaciones generales. En este caso, el rendimiento en el conjunto de validación o prueba puede empeorar a medida que aumenta el número de épocas.

2. Convergencia

   - En el caso ideal, el modelo comienza a converger a una solución óptima a medida que se realizan más épocas. Esto significa que la función de pérdida disminuye y el rendimiento mejora. Sin embargo, es importante monitorear tanto el rendimiento en el conjunto de entrenamiento como en el conjunto de validación para detectar signos de sobreajuste o estancamiento.

3. Tamaño del conjunto de datos

   - El número de épocas puede verse afectado por el tamaño del conjunto de datos. Si el conjunto de datos es grande, es posible que se necesiten menos épocas para que el modelo aprenda patrones y relaciones. Por otro lado, si el conjunto de datos es pequeño, puede ser necesario entrenar durante más épocas para que el modelo capture de manera efectiva las características y evite el sobreajuste.

4. Búsqueda de hiper parámetros

   - En algunos casos, se puede realizar una búsqueda sistemática del número de épocas junto con otros hiper parámetros para encontrar la combinación óptima que minimice la función de pérdida o mejore el rendimiento del modelo en el conjunto de validación.

Es importante encontrar un equilibrio entre el número de épocas suficientes para que el modelo aprenda y generalice bien, sin excederse y caer en el sobreajuste. Esto puede requerir experimentación y seguimiento del rendimiento del modelo en conjuntos de validación o prueba independientes a medida que se aumenta el número de épocas.

2.3.13. Clasificador de haarcascade

El clasificador de Haarcascade es un algoritmo utilizado para la detección de objetos en imágenes o vídeos. Fue propuesto por Paul Viola y Michael Jones en 2001 y se ha utilizado ampliamente para la detección de rostros.

El proceso de detección de rostros con Haarcascade consta de los siguientes pasos:

1. Extracción de características Haar

   - El algoritmo utiliza una técnica de características llamada características Haar, que se basa en la diferencia de intensidad de los píxeles en diferentes regiones de la imagen. Estas características son patrones rectangulares que representan variaciones en la intensidad de la imagen.

2. Entrenamiento del clasificador

   - Para entrenar el clasificador de Haarcascade, se necesitan ejemplos positivos y negativos. Los ejemplos positivos son imágenes que contienen rostros, mientras que los ejemplos negativos son imágenes que no contienen rostros. Durante el entrenamiento, el algoritmo ajusta los pesos y umbrales de las características Haar para clasificar correctamente los ejemplos positivos y negativos.

3. Creación del clasificador

   - Después del entrenamiento, se obtiene un clasificador de Haarcascade que consiste en una estructura en cascada de clasificadores débiles. Cada clasificador débil es un clasificador binario que utiliza una sola característica Haar y proporciona una respuesta binaria (rostro o no rostro).

4. Detección de rostros

   - Para detectar rostros en una imagen, se aplica el clasificador de Haarcascade en forma de ventana deslizante. La ventana deslizante es una pequeña región rectangular que se mueve por toda la imagen y se escala a diferentes tamaños. Para cada ubicación y escala de la ventana, se evalúa el clasificador de Haarcascade para determinar si hay un rostro presente en esa región. Se utilizan técnicas como el método de integral de imágenes para acelerar el proceso de evaluación.

5. Filtro de falsos positivos

   - Debido a la naturaleza del algoritmo de detección, puede haber falsos positivos, es decir, detecciones incorrectas de rostros en regiones que no son rostros reales. Para reducir estos falsos positivos, se utiliza un filtro de falsos positivos que aplica criterios adicionales, como la verificación de la forma y la posición del rostro.

El clasificador de Haarcascade para detección de rostros es ampliamente utilizado debido a su eficiencia y precisión. Sin embargo, es importante tener en cuenta que puede haber casos en los que el algoritmo no funcione correctamente, como en condiciones de iluminación o ángulos extremos, y puede requerir ajustes o complementarse con otros métodos de detección de rostros más avanzados.

2.3.14. Función de pérdida

La entropía cruzada (cross-entropy en inglés) es una función de pérdida comúnmente utilizada en problemas de clasificación en el aprendizaje automático. Esta función de pérdida mide la discrepancia entre la distribución de probabilidad predicha por un modelo y la distribución de probabilidad real de los datos.

En el contexto de la clasificación binaria, donde se tienen dos clases (por ejemplo, clase positiva y clase negativa), la entropía cruzada se define de la siguiente manera:

Entropía cruzada = -[y * log(p) + (1 - y) * log(1 - p)]

Donde:
- y es la etiqueta real de la muestra (0 o 1).
- p es la probabilidad predicha por el modelo de que la muestra pertenezca a la clase positiva.

En el caso de la clasificación multiclase, donde se tienen más de dos clases, la entropía cruzada se extiende para medir la discrepancia entre la distribución de probabilidad predicha (a menudo a través de una función de activación softmax) y la distribución de probabilidad real de los datos.

La función de pérdida de entropía cruzada se utiliza en problemas de clasificación porque penaliza fuertemente las predicciones incorrectas, es decir, cuando la probabilidad predicha

para la clase correcta es baja. Al minimizar la entropía cruzada durante el entrenamiento, el modelo se ajusta para hacer predicciones más precisas y se acerca más a la distribución de probabilidad real.

Es importante destacar que la entropía cruzada es una función continua y diferenciable, lo que la hace adecuada para el entrenamiento de modelos utilizando algoritmos de optimización basados en gradientes, como el descenso de gradiente estocástico (SGD) o Adam.

2.3.15. Función softmax

La función softmax es una función de activación utilizada comúnmente en problemas de clasificación multiclase. Toma un vector de valores reales y produce una distribución de probabilidad sobre las diferentes clases.

La función softmax se define de la siguiente manera para un vector de entrada x:

softmax(x) = [$e^(x_i)$ / sum($e^(x_j)$)] para todo i

Donde:
- e es la base del logaritmo natural (constante de Euler).
- x_i es el valor de la $i_{ésima}$ entrada del vector de entrada x.
- sum() representa la suma de todos los elementos en el vector.

La función softmax transforma cada elemento del vector de entrada en una probabilidad proporcional a su valor exponencial, normalizando el vector de salida para que la suma de todas las probabilidades sea igual a 1.

La salida de la función softmax es una distribución de probabilidad que asigna probabilidades a cada clase en un problema de clasificación multiclase. Cada elemento de salida representa la probabilidad de que la entrada pertenezca a la clase correspondiente.

La función softmax es útil porque proporciona una forma de representar las probabilidades de múltiples clases de manera coherente y permite que el modelo se enfoque en la clase más probable durante el entrenamiento y la inferencia.

Es importante tener en cuenta que la función softmax es diferenciable y se utiliza comúnmente en conjunción con la entropía cruzada como función de pérdida en problemas de clasificación multiclase. La combinación de softmax y entropía cruzada permite entrenar el modelo para que prediga la distribución de probabilidad correcta mientras minimiza la discrepancia entre la distribución predicha y la distribución real.

3 Procedimiento Experimental

Después de entender cada concepto acerca de la red neuronal convulcional, procederemos a realizar la implementación de ella, para la clasificación de género.

1. Preparación de los datos
   - Se recopiló un conjunto de datos que contiene imágenes etiquetadas con género (hombre/mujer) como se muestra en la figura 4.

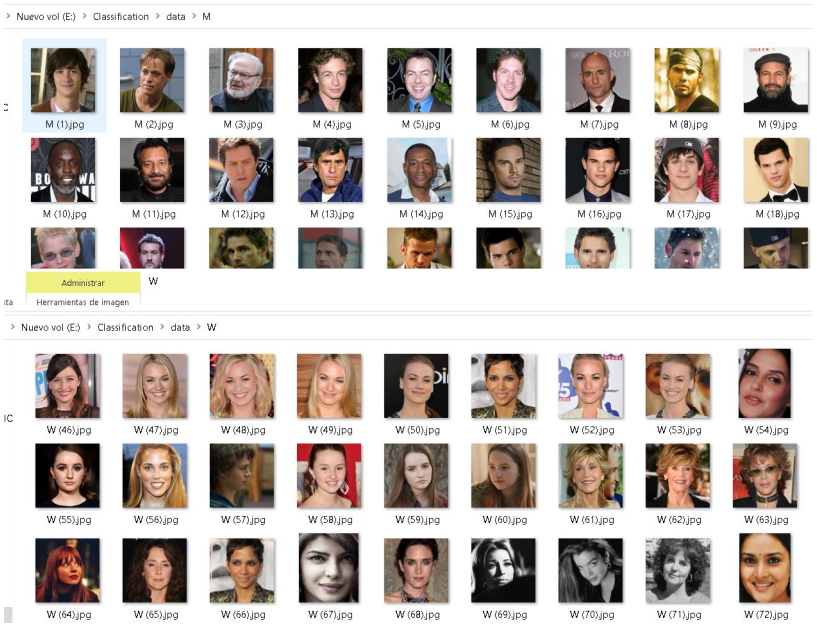

Figura 4: Data

   - Se usó el código **processed.py** para detectar los rostros, usando el haar casascade elegido, del conjunto de datos y de esa forma tener imágenes detectadas en la carpeta processed como se observa en la figura 5.

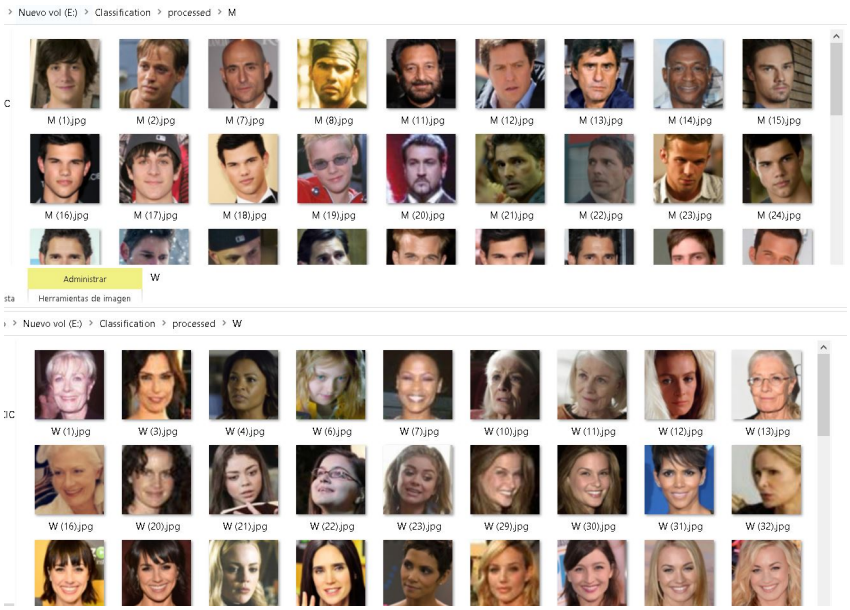

Figura 5: Processed

- Después se usó el código **validation.py** para separar al azar los datos de la carpeta processed en 20 % para el conjunto de validación y 80 % para el conjunto de entrenamiento (processed). Esto se oberva en la figura 6.

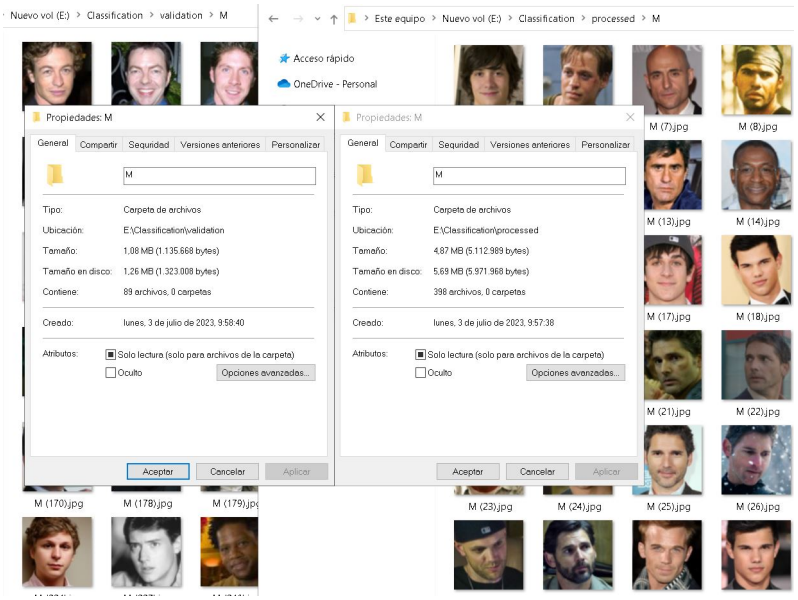

Figura 6: Processed

- Por último se usa el código **dataset.py** para preparar los conjuntos de datos de entrenamiento y validación utilizando la biblioteca torchvision y la clase DataLoader de PyTorch. Proporciona una serie de transformaciones de datos que se aplican a las imágenes como se ve en la figura 7, antes de ser utilizadas para entrenar y validar un modelo de aprendizaje automático.

```python
# Transformaciones de datos de entrenamiento
train_augs = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomRotation((0, 90)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Transformaciones de datos de validación
valid_augs = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Figura 7: Dataset

2. Configuración del modelo

   - En el código **model.py** definimos nuestro modelo de red neuronal convolucional utilizando la biblioteca PyTorch.
   - Aquí especificamos las capas y la estructura del modelo, incluyendo capas convolucionales, capas de activación y capas de clasificación como se puede observar en la figura 8.
   - Se proporciona métodos para realizar la propagación hacia adelante, capturar los gradientes de las activaciones y acceder a las activaciones del modelo.

     Estas funcionalidades son fundamentales en el entrenamiento y evaluación de redes neuronales convolucionales para tareas de clasificación

```python
class Model(nn.Module):
    def __init__(self, n_classes):
        super(Model, self).__init__()

        # Definición de la arquitectura del modelo
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=(5, 5), padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(4, 4), stride=2),

            nn.Conv2d(in_channels=16, out_channels=16, kernel_size=(5, 5), padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(4, 4), stride=2),

            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=(5, 5), padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(4, 4), stride=2),

            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(5, 5), padding=1),
            nn.ReLU(),
        )

        self.maxpool = nn.MaxPool2d(kernel_size=(4, 4), stride=2)

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(6400, 2048),
            nn.ReLU(),
            nn.Linear(2048, n_classes)
        )
```

Figura 8: Model

3. Entrenamiento del modelo

   - En el código **train.py** se realiza el entrenamiento y evaluación del modelo utilizando los conjuntos de datos de entrenamiento y validación
   - Se configuraron los parámetros de entrenamiento, como el tamaño del lote, la tasa de aprendizaje y el número de épocas.
   - Se define el optimizador (Adam) y la función de pérdida (CrossEntropyLoss) que se utilizarán durante el entrenamiento.
   - Se iteró sobre las épocas de entrenamiento, realizando el entrenamiento de la red neuronal utilizando los datos de entrenamiento y actualizando los pesos del modelo.
   - Se calculó la pérdida y el accuracy en cada época tanto para los datos de entrenamiento como para los datos de validación.

- Se guardaron los mejores pesos del modelo basados en la pérdida en el conjunto de validación.

  En la figura 9 se puede observar el entrenamiento del modelo en dos épocas, se aprecia lo que es el accurracy y el loss en cada conjunto de entrenamiento

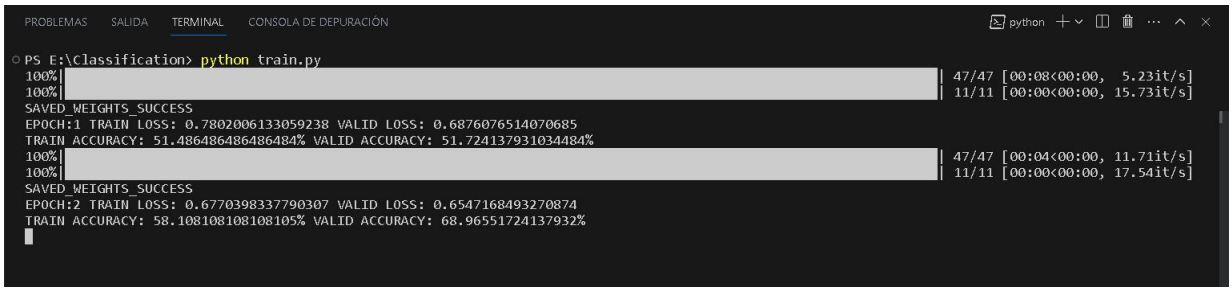

Figura 9: Train

En la figura 10 se puede observar la gráfica de las métricas respecto a las épocas, gracias a estas gráficas podemos escoger que mejor peso usar.

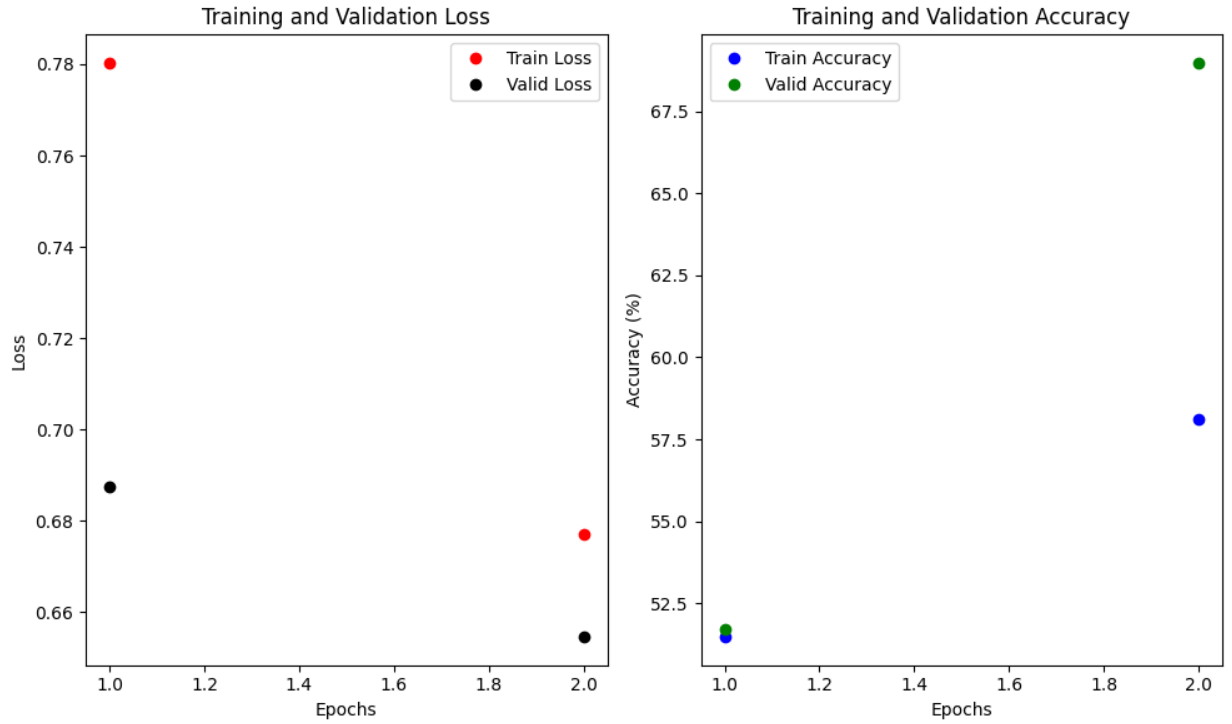

Figura 10: Gráfica de métricas de la época 2

4. Evaluación del modelo

- En el código **test.py** se carga el mejor peso del modelo entrenado, el cual fue en la época 50, ya que como se muestra en la figura 11 nuesto valid tiene un buen porcentaje de accurracy que supera al del train y tiene un menor loss respecto al train.

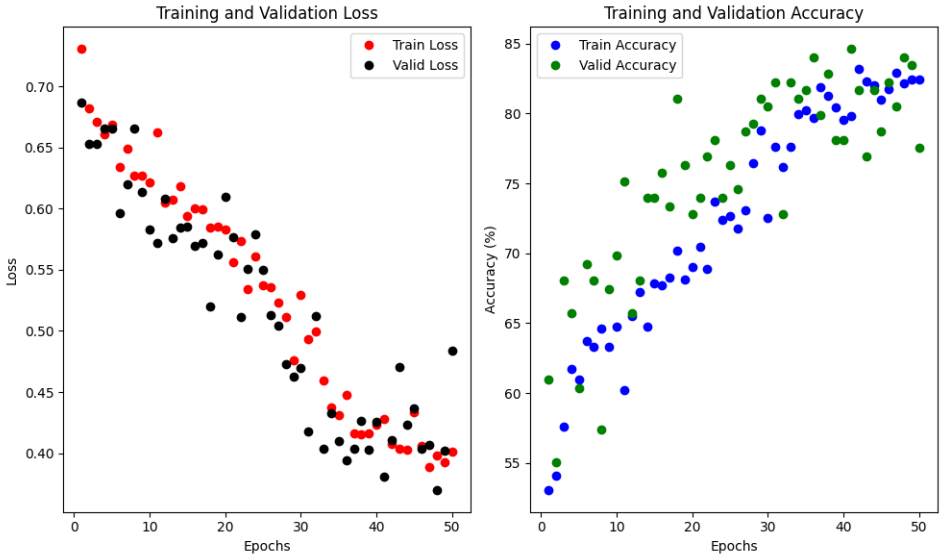

Figura 11: Gráfica de métricas la época 50

- Se realizó la evaluación del modelo utilizando un conjunto de pruebas independiente y se utilizó el clasificador de caras Haar Cascade para detectar las caras.
- Se realizó la predicción de género utilizando el modelo y se comparó con las etiquetas reales para evaluar la precisión del modelo como se observa en la figura 12.

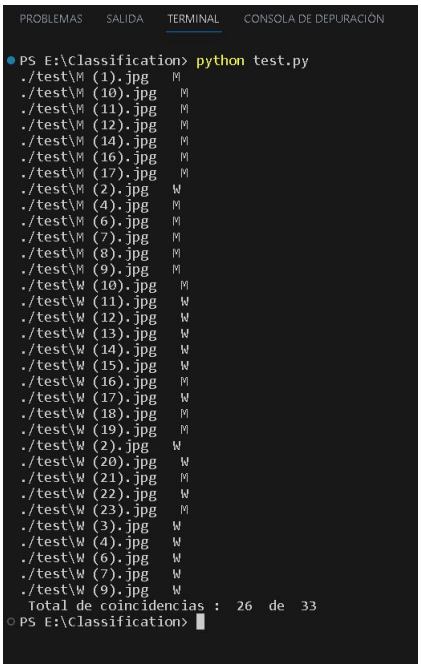

Figura 12: Evaluación usando el `best_weights_50.pt`

4 Discusiones y Conclusiones

4.1. Discusiones

En este informe, hemos abordado la tarea de clasificar si una imagen es de un hombre o una mujer utilizando una combinación de redes neuronales convolucionales (CNN) y Haar Cascade. A continuación, discutiremos algunos aspectos clave de nuestro proyecto.

Selección del enfoque: Optamos por combinar redes neuronales convolucionales con Haar Cascade debido a sus fortalezas complementarias. Las CNN son conocidas por su capacidad para extraer características relevantes de las imágenes, mientras que Haar Cascade es una técnica eficiente para la detección de objetos en imágenes. La combinación de ambos enfoques nos permitió aprovechar estas ventajas y mejorar la precisión de la clasificación.

Preprocesamiento de datos: Antes de alimentar las imágenes a nuestro modelo, realizamos un preprocesamiento para garantizar la calidad y la consistencia de los datos. Esto incluyó redimensionar todas las imágenes a un tamaño estándar, normalizar los valores de píxeles y aplicar técnicas de aumento de datos para aumentar la variabilidad y mejorar la capacidad del modelo para generalizar.

Arquitectura del modelo: La arquitectura esta compuesta por 4 capas, donde cada en una una se hace una convolucion, una activacion de tipo RELU y un Pooling de tipo Max-Pooling; y la entrenamos utilizando un conjunto de datos etiquetados que contenía imágenes de hombres y mujeres. Además, utilizamos Haar Cascade para detectar rostros en las imágenes y luego extraer esas regiones para su posterior clasificación por la CNN. Esta combinación permitió al modelo centrarse en características relevantes y mejorar la precisión de la clasificación.

Evaluación del rendimiento: Medimos el rendimiento de nuestro modelo utilizando métricas como precisión, recall y la matriz de confusión. Realizamos pruebas exhaustivas utilizando conjuntos de datos de prueba y validación para evaluar la precisión y la capacidad de generalización de nuestro modelo. Ademas de hacer pruebas con distintas epocas para comparar su eficiencia

4.2. Conclusiones

En conclusión, nuestro proyecto ha demostrado que la combinación de redes neuronales convolucionales y Haar Cascade es una estrategia efectiva para clasificar imágenes de hombres y mujeres. A través de un proceso de preprocesamiento adecuado y el diseño de una arquitectura de red sólida, logramos una alta precisión en la clasificación.

Nuestros resultados indican que las redes neuronales convolucionales son capaces de extraer características discriminativas de las imágenes para realizar una clasificación precisa. Al combinar esta técnica con Haar Cascade, pudimos mejorar aún más el rendimiento del modelo al enfocarnos en regiones relevantes de la imagen.

Sin embargo, también reconocemos algunas limitaciones de nuestro enfoque. La precisión de la clasificación puede verse afectada por diversos factores, como la calidad de las imágenes de entrada, la variabilidad en la apariencia de hombres y mujeres, y posibles sesgos en los datos de entrenamiento.

Para futuros trabajos, se pueden explorar algunas mejoras adicionales. Por ejemplo, se podrían utilizar arquitecturas de redes más avanzadas o técnicas de aprendizaje transferido para mejorar aún más la precisión del modelo.

5 Referencias

[1] https://www.inesdi.com/blog/que-son-las-redes-neuronales/

[2] https://www.ibm.com/es-es/topics/recurrent-neural-networks

[3] https://arxiv.org/pdf/1502.02766v3.pdf

[4] http://www.deeplearningbook.org/

[5] https://www.coursera.org/specializations/deep-learning

[6] https://dev.to/sandeepbalachandran/machine-learning-convolution-with-color-images-2p41

[7] https://www.youtube.com/watch?v=KuXjwB4LzSA

[8] https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[9] https://math.libretexts.org/Bookshelves/Differential_Equations/Introduction_to_Partial_Differential_Equations_(Herman)/09%3A_Transform_Techniques_in_Physics/9.06%3A_The_Convolution_Operation

[10] https://furkangulsen.medium.com/what-is-a-tensor-ce8e78835d08