Detección de Caras Retocadas: Implementación en Python con PyTorch

Carlos Santana Esplá y Raquel Almeida Quesada

18-01-2024

1 Objetivo del Código

El código tiene como objetivo implementar un modelo de detección de caras retocadas utilizando técnicas de aprendizaje profundo, específicamente utilizando la biblioteca PyTorch.

2 Preprocesamiento de Imágenes

2.1 Conversiones y Redimensionamiento

Las imágenes en el directorio especificado se abren y convierten a formato RGB. Se redimensionan a un tamaño de 256x256 píxeles para asegurar uniformidad.

2.2 Transformaciones para Entrenamiento

Se emplean diversas transformaciones para aumentar la variabilidad del conjunto de datos de entrenamiento:

- `RandomRotation(15)`: Rotación aleatoria de hasta 15 grados para simular diferentes ángulos de captura de imágenes.
- `RandomResizedCrop(224)`: Recorte aleatorio redimensionado a 224x224 píxeles, permitiendo al modelo centrarse en diferentes regiones de la imagen.
- `RandomHorizontalFlip()`: Volteo horizontal aleatorio para simular diferentes orientaciones faciales.
- `ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2)`: Ajuste de brillo, contraste, saturación y tono para mejorar la capacidad del modelo para manejar variaciones en la iluminación y el color.

2.3 Transformaciones para Validación

- `Resize((224, 224))`: Redimensionamiento fijo a 224x224 píxeles para mantener la consistencia durante la validación.
- `ToTensor()`: Convierte la imagen a un tensor, un formato compatible con PyTorch.
- `Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])`: Normalización de los valores de píxeles de la imagen utilizando la media y desviación estándar precalculadas de ImageNet. Esto ayuda a que el modelo preentrenado trabaje de manera efectiva con los datos.

3 Modelo de Red Neuronal

3.1 Arquitectura ResNet18

Se utiliza la arquitectura ResNet18 preentrenada, una red neuronal profunda y eficiente que ha demostrado un buen rendimiento en tareas de clasificación de imágenes. La capa final se ajusta para el problema específico, seguido de una función de activación ELU, una capa de dropout para regularización y finalmente, una capa lineal de salida con una sola neurona para la clasificación binaria.

4 Entrenamiento con Validación Cruzada

4.1 Número de Divisiones y Épocas

Se realiza una validación cruzada con 5 divisiones para evaluar la robustez del modelo. Se configuran 2 épocas de entrenamiento para cada división.

4.2 Cargador de Datos Personalizado para Validación Cruzada

En cada época, se itera sobre las divisiones de entrenamiento y validación utilizando un cargador de datos personalizado. Se crea un conjunto de entrenamiento y validación diferente en cada iteración.

4.3 Bucle de Entrenamiento y Evaluación

Para cada división de validación cruzada, se realiza el entrenamiento del modelo y se evalúan los resultados. Durante el entrenamiento, se calcula la pérdida y la precisión en el conjunto de entrenamiento. Se evalúa la precisión en el conjunto de validación para monitorear el rendimiento.

4.4 Promedio de Resultados

Se calcula el promedio de la pérdida y la precisión en el conjunto de entrenamiento durante las épocas para cada división. Los resultados se imprimen para cada división, proporcionando información detallada sobre el rendimiento del modelo en cada pliegue de la validación cruzada.

4.5 Resultados del entrenamiento

El modelo parece mejorar en la segunda época, ya que tanto la pérdida de entrenamiento como la precisión de entrenamiento muestran mejoras. La validación cruzada proporciona una evaluación robusta del modelo en diferentes conjuntos de datos de entrenamiento y validación. La precisión de validación muestra variabilidad entre los pliegues, indicando que el modelo podría beneficiarse de más ajustes o técnicas de regularización. El entrenamiento finaliza después de dos épocas con resultados prometedores, pero la interpretación definitiva dependerá de un análisis más detallado y visualización de los resultados finales.

5 Evaluación del Modelo

5.1 Evaluación Final y Análisis

El modelo muestra un rendimiento aceptable con una precisión de validación de entre 74.02 y 84, dependiendo del número de épocas que le pongamos, a más épocas mejor rendimiento. Aunque la tasa de aciertos es buena, el análisis de las muestras incorrectas es crucial para entender las áreas en las que el modelo podría beneficiarse de mejoras.

6 Librerías Utilizadas

6.1 Librerías Utilizadas

- PyTorch: Para la construcción y entrenamiento del modelo.
- scikit-learn: Implementación de la validación cruzada.
- Matplotlib: Visualización de resultados.

7 Conclusión

El código presenta una solución completa para el problema de detección de caras retocadas, aprovechando modelos preentrenados y aplicando técnicas de aumento de datos para mejorar la generalización del modelo. El uso de la validación cruzada proporciona una evaluación más robusta de la capacidad del modelo para generalizar a nuevos datos. Sin embargo, el principal problema del código es el tiempo en el que tarda en ejecutarse, ya que si lo entrenamos

con 5 épocas el código llega a tardar más de una hora de entrenamiento para una precisión del 85 por ciento. A pesar de haber buscado formas de mejorar el tiempo de ejecución, cambiando el batch size y más parámetros, no conseguimos hacer una reducción significante del tiempo, sin perder precisión.