Tecnológico de Monterrey

Reporte: Clasificación de Imágenes de Patos y Gansos utilizando Redes Neuronales Convolucionales (CNN)

TC3007C.501

Inteligencia artificial avanzada para la ciencia de datos 2 Gpo 501

Docentes

Dr. Benjamín Valdés Aguirre

Ma. Eduardo Daniel Juárez Pineda Dr.

Ismael Solis Moreno

Dr. José Antonio Cantoral Ceballos

Dr. Carlos Alberto Dorantes Dosamantes

Integrante

Dafne Fernández Hernández A01369239

1. Introducción

En los últimos años, las redes neuronales convolucionales (CNN, por sus siglas en inglés) se han convertido en una de las herramientas más poderosas dentro del aprendizaje profundo, especialmente en tareas de clasificación de imágenes. Este proyecto tiene como objetivo desarrollar un modelo CNN para clasificar imágenes de dos clases: patos y gansos, utilizando un dataset estructurado y técnicas avanzadas de entrenamiento.

El modelo fue implementado en el contexto del juego *"Pato, Pato, Ganso"*, en el cual se busca simular la dinámica del juego tradicional. En esta versión, el modelo identifica aleatoriamente imágenes como "patos" o "gansos". Si encuentra un "ganso", el flujo de ejecución pregunta al usuario si desea continuar jugando o terminar el programa.

El desarrollo de este proyecto abarcó las siguientes etapas principales:

1. Extracción, transformación y carga (ETL) de datos para preparar el dataset.
2. Construcción de una arquitectura CNN basada en **ResNet18**.
3. Entrenamiento del modelo con un enfoque iterativo, ajustando hiperparámetros y técnicas de regularización.
4. Evaluación de los resultados iniciales y la implementación de mejoras para optimizar el rendimiento del modelo.
5. Desarrollo de un flujo interactivo que combina aprendizaje profundo con interacción de usuario.

El presente reporte documenta cada una de estas etapas, analizando tanto las decisiones técnicas como los resultados obtenidos. Además, se incluyen comparaciones entre los resultados iniciales y los mejorados, ofreciendo una visión completa del impacto de las optimizaciones implementadas.

---

2. Descripción del Dataset

El dataset utilizado en este proyecto contiene imágenes organizadas en dos categorías principales:

1. **Duck (Pato)**: Representa imágenes de patos, con una diversidad en términos de posiciones, colores y contextos.
2. **Goose (Ganso)**: Incluye imágenes de gansos en distintas poses y escenarios

3. Estructura del Dataset

- Las imágenes están organizadas en carpetas separadas, una para cada clase.
- Los formatos de archivo incluyen .jpg y .png, asegurando compatibilidad con bibliotecas como **Pillow** y **PyTorch**.
- Cada clase contiene un conjunto variado de imágenes, diseñadas para representar diferentes condiciones de iluminación, ángulos de cámara y calidad visual.

4.Preparación de Datos

Antes de entrenar el modelo, el dataset fue sometido a un análisis inicial para detectar posibles problemas:

- **Balance de Clases**: Se verificó que ambas clases tuvieran una representación suficiente para evitar un sesgo significativo durante el entrenamiento.
- **Resolución de Imágenes**: Se analizaron las dimensiones de las

imágenes para identificar inconsistencias que pudieran afectar el rendimiento del modelo.

- **Calidad de Datos**: Se revisaron las imágenes en busca de ruido, como duplicados o datos irrelevantes, para garantizar que el conjunto de datos sea confiable.

Distribución

- **Clase Duck**: 500 imágenes únicas, representando el 50% del dataset total.
- **Clase Goose**: 500 imágenes únicas, representando el otro 50%.
- **Conjuntos de Datos**:
    - **Entrenamiento (Train)**: 70% del total (700 imágenes).
    - **Validación (Validation)**: 20% del total (200 imágenes).
    - **Pruebas (Test)**: 10% del total (100 imágenes).

Contenido de la carpeta Data

1. **train/**:
    - Contiene imágenes destinadas al entrenamiento del modelo.
    - Subcarpetas:
        - duck/: Carpeta con imágenes correspondientes a la clase "pato".
        - goose/: Carpeta con imágenes correspondientes a la clase "ganso".
    - **Propósito**: Proveer datos para el entrenamiento supervisado del modelo CNN.

 val/:

- Contiene imágenes utilizadas para validar el modelo durante el proceso de entrenamiento.
- Subcarpetas:
    - duck/: Carpeta con imágenes de patos utilizadas para validación.
    - goose/: Carpeta con imágenes de gansos utilizadas para validación.
- **Propósito**: Ayudar a evaluar el desempeño del modelo en un conjunto de datos no visto durante el entrenamiento.

 test/:

- Contiene imágenes utilizadas exclusivamente para probar el modelo una vez que el entrenamiento ha finalizado.
- Subcarpetas:
    - duck/: Carpeta con imágenes de patos para pruebas.
    - goose/: Carpeta con imágenes de gansos para pruebas.

Transformaciones Aplicadas

Para garantizar que las imágenes sean compatibles con el modelo CNN y mejorar la capacidad del modelo para generalizar, se aplicaron las siguientes transformaciones:

- **Redimensionamiento**: Todas las imágenes fueron escaladas a dimensiones de 224x224 píxeles.

- **Normalización**: Se ajustaron los valores de los píxeles a un rango estándar basado en los valores medios y desviaciones estándar de **ImageNet**.
    - Los valores de píxeles fueron ajustados para seguir las estadísticas del dataset **ImageNet**:
        - Media: [0.485, 0.456, 0.406]
        - Desviación estándar: [0.229, 0.224, 0.225]

- **Augmentación** (solo en el conjunto de entrenamiento):
    - Rotación aleatoria.
    - Inversiones horizontales.
    - Ajustes de brillo y contraste.

Estas preparaciones y transformaciones aseguraron que el dataset no solo fuera adecuado para el entrenamiento de un modelo CNN, sino que también ayudaron a mejorar su robustez frente a variaciones en los datos reales.

Contenido de los Archivos

Además de las imágenes, otros archivos y estructuras complementan el desarrollo del modelo:

1. **duck_goose_model.pth**:
    - Este archivo contiene los pesos del modelo entrenado en formato PyTorch.
    - **Propósito**: Permite cargar el modelo previamente entrenado sin necesidad de repetir el entrenamiento.
2. **ModeloPatoGanso.py**:
    - Archivo principal para el entrenamiento del modelo.
    - **Incluye**:
        - Construcción de la arquitectura CNN basada en ResNet18.
        - Funciones de entrenamiento y validación.
        - Optimización y ajuste de hiperparámetros.
3. **DuckGooseModel.py**:
    - Archivo diseñado para hacer predicciones y ejecutar el juego interactivo "Pato, Pato, Ganso".
    - **Incluye**:
        - Carga del modelo entrenado desde duck_goose_model.pth.
        - Predicción de imágenes en carpetas especificadas por el usuario.
        - Interacción dinámica con el usuario para continuar o finalizar el juego.
4. **Data_Analysis.py**:
    - Archivo para análisis inicial del dataset.
    - **Incluye**:
        - Revisión del balance de clases.
        - Validación de la calidad de las imágenes.

- Generación de estadísticas descriptivas.

Uso de ETL (Extract - Transform - Load)

El proceso de ETL (Extract, Transform, Load) es una de las etapas más importantes en proyectos basados en datos, ya que permite preparar el dataset de forma adecuada para que el modelo pueda aprender de manera eficiente. En este proyecto, el proceso de ETL se dividió en las siguientes fases:

1. Extracción de Datos

Los datos fueron obtenidos de un archivo comprimido (PatoGanso.zip) que contenía las imágenes organizadas en carpetas específicas para cada clase: duck y goose. Estas carpetas estaban separadas en conjuntos de entrenamiento (train), validación (val) y prueba (test), lo que facilitó la estructuración inicial.

La extracción consistió en:

- Descomprimir el archivo.
- Inspeccionar las carpetas y verificar la cantidad y calidad de las imágenes.
- Crear funciones para listar y acceder a los archivos en sus respectivas carpetas.

2. Transformación de Datos

La transformación es una etapa crítica para estandarizar y preparar los datos de entrada. En este proyecto, se aplicaron varias transformaciones utilizando PyTorch y su módulo torchvision.transforms, lo que permitió trabajar con imágenes en un formato adecuado para el modelo CNN basado en ResNet18.

Transformaciones aplicadas:

- **Redimensionamiento (Resize):**
  - Todas las imágenes fueron escaladas a un tamaño uniforme de 224x224 píxeles, que es el tamaño esperado por la arquitectura ResNet18.
  - Esto asegura que las imágenes tengan dimensiones compatibles con la red y reduce la carga computacional.
- **Recorte central (CenterCrop):**
  - Se eliminó el exceso de información en los bordes de las imágenes, manteniendo el área más relevante.
- **Conversión a tensores (ToTensor):**
  - Las imágenes, originalmente en formato de píxeles RGB, fueron convertidas a tensores para que pudieran ser procesadas por PyTorch.
- **Normalización (Normalize):**
  - Los valores de los píxeles fueron ajustados según las estadísticas del dataset ImageNet:
  - Media: [0.485, 0.456, 0.406]
  - Desviación estándar: [0.229, 0.224, 0.225]
  - Esto asegura que las características de las imágenes

estén en un rango esperado por la red preentrenada.

- **Augmentación de Datos:**
  - Se aplicaron técnicas de augmentación para el conjunto de entrenamiento:
  - Rotaciones aleatorias (RandomRotation) para simular variaciones en la orientación.
  - Inversiones horizontales (RandomHorizontalFlip) para reflejar casos simétricos.
  - Variaciones en el brillo y contraste (ColorJitter) para hacer el modelo más robusto frente a diferencias de iluminación.

Carga de Datos

El dataset fue cargado utilizando el módulo torchvision.datasets.ImageFolder, que permite organizar las imágenes según las carpetas de clase (duck y goose). Estas imágenes fueron procesadas en DataLoaders, que dividen los datos en lotes (batches) para optimizar el entrenamiento.

Parámetros clave en la carga de datos:

- **Batch Size:**
  - Se utilizó un tamaño de lote de 32, lo que balancea la eficiencia computacional y la estabilidad del gradiente.
- **Shuffling:**
  - Los datos se mezclaron aleatoriamente en cada época para evitar sesgos relacionados con el orden de las imágenes

---

Implementación de la Red Neuronal

La red neuronal convolucional (CNN) utilizada en este proyecto fue construida sobre la arquitectura ResNet18, que es una de las más populares y efectivas para tareas de clasificación de imágenes. A continuación, se detallan los pasos clave de la implementación:

1. Selección de la Arquitectura

Se eligió ResNet18 por las siguientes razones:

- **Profundidad Moderada:**
  - ResNet18 tiene 18 capas, lo que la hace menos propensa al sobreajuste en datasets pequeños.
- **Capacidad de Aprendizaje:**
  - Utiliza bloques residuales, que facilitan el entrenamiento de redes profundas al evitar problemas de degradación del gradiente.
- **Modelo Preentrenado:**
  - ResNet18 viene preentrenada en el dataset ImageNet, lo que permite aprovechar características visuales genéricas (como bordes, texturas y patrones) sin necesidad de entrenar desde cero.

2. Personalización de la Red

Se reemplazó la última capa totalmente conectada para adaptarse al problema de dos clases (duck y goose).

2. Función de Pérdida y Optimizador

- **Función de Pérdida**:
  - Se utilizó **CrossEntropyLoss**, una función estándar para

problemas de clasificación multiclase.

- Penaliza las predicciones incorrectas asignando un costo basado en la probabilidad predicha para la clase verdadera.

- **Optimizador**:
  - **Adam** fue elegido como el optimizador debido a su capacidad para ajustar dinámicamente las tasas de aprendizaje, logrando un balance entre velocidad de convergencia y estabilidad.

- **Scheduler**:
  - Se empleó un programador de tasa de aprendizaje para reducir la tasa a medida que el modelo converge.

4. Entrenamiento

El modelo fue entrenado durante 20 épocas. En cada época:

- Se calculó la pérdida en el conjunto de entrenamiento.
- Se monitoreó el desempeño en el conjunto de validación.

Construcción del Modelo CNN

La construcción del modelo CNN se basó en la arquitectura ResNet18, conocida por su diseño eficiente y robusto, especialmente para datasets pequeños o medianos. Esta arquitectura implementa bloques residuales, los cuales permiten la propagación del gradiente a través de capas profundas, mejorando significativamente el aprendizaje.

1. Arquitectura ResNet18

- **Capas Convolucionales:**
  - ResNet18 utiliza capas convolucionales para extraer características jerárquicas de las imágenes, desde patrones básicos como bordes hasta características complejas como texturas y formas.
- **Bloques Residuales:**
  - Introducen conexiones de "skip" entre capas, lo que mitiga problemas comunes en redes profundas, como el desvanecimiento del gradiente.
- **Preentrenamiento:**
  - El modelo fue inicializado con pesos preentrenados en el dataset ImageNet, lo que proporciona una base sólida al transferir características genéricas al problema específico de clasificación entre duck y goose.

2. Modificaciones Personalizadas

La última capa de ResNet18, originalmente diseñada para clasificar en 1000 clases, fue reemplazada por una capa totalmente conectada con dos salidas correspondientes a nuestras clases (duck y goose).

3. Optimización y Regularización

- Se incluyó la regularización L2 para reducir el riesgo de sobreajuste.
- Un programador de tasa de aprendizaje permitió ajustar dinámicamente la magnitud de los pasos de optimización según la pérdida.

Entrenamiento del Modelo

El modelo fue entrenado utilizando un enfoque supervisado. Este proceso implicó ajustar los pesos de la red para minimizar la pérdida entre las predicciones del modelo y las etiquetas reales.

1. Configuración Inicial

- **Parámetros**:
  - Tamaño del lote: 32
  - Épocas: 20
  - Tasa de aprendizaje inicial: 0.001
- **Conjuntos de Datos**:
  - Entrenamiento: 70% de las imágenes.
  - Validación: 20% de las imágenes para monitorear el desempeño durante el entrenamiento.

2. Flujo del Entrenamiento

1. **Forward Pass**:
   - Se pasaron imágenes a través de la red para generar predicciones.
2. **Cálculo de la Pérdida**:
   - Se utilizó la función de pérdida de entropía cruzada para calcular el error entre las predicciones y las etiquetas.
3. **Backward Pass**:
   - El error se propagó hacia atrás para ajustar los pesos de las capas utilizando el optimizador **Adam**.
4. **Actualización de Pesos**:
   - Los pesos de la red se ajustaron con base en el gradiente calculado.

3. Resultados del Entrenamiento

- La pérdida disminuyó constantemente durante las primeras 15 épocas, estabilizándose hacia el final.
- La precisión en el conjunto de validación alcanzó el 95%, lo que indica una buena generalización inicial.

Análisis de Resultados

Tras el entrenamiento, el modelo fue evaluado utilizando el conjunto de prueba, que contiene imágenes nunca vistas durante el entrenamiento. Los resultados iniciales fueron los siguientes:

1. Métricas Clave

- **Precisión**:
  - El modelo clasificó correctamente el 93% de las imágenes en el conjunto de prueba.
- **R^2**:
  - Se calculó un coeficiente de determinación del 0.95 en el conjunto de validación.

- **Pérdida Final**:
    - La pérdida mínima alcanzada en validación fue de **0.045**.

2. Análisis de Errores

Los errores se observaron principalmente en imágenes con:

- Iluminación deficiente.
- Ángulos extremos de las tomas.
- Similitudes visuales entre las dos clases.

Implementación de Mejoras

Para optimizar el modelo, se implementaron diversas técnicas de mejora, enfocadas en reducir el sobreajuste y mejorar la capacidad del modelo para generalizar.

1. Regularización

- **L2 Regularization**:
    - Se agregó un término de penalización al optimizador para prevenir que los pesos de las capas se vuelvan excesivamente grandes.
    - Esto ayudó a reducir el sobreajuste en el conjunto de entrenamiento.

2. Scheduler de Tasa de Aprendizaje

- Se utilizó un programador que disminuye la tasa de aprendizaje cuando la pérdida no mejora tras varias épocas consecutivas.
- Esto permitió explorar los mínimos locales con mayor precisión hacia el final del entrenamiento.

3. Augmentación de Datos

- Técnicas avanzadas como recortes aleatorios, inversiones y variaciones de color fueron aplicadas al conjunto de entrenamiento para aumentar la diversidad de las imágenes.

Análisis de Resultados Mejorados

Tras implementar estas mejoras, los resultados del modelo mejoraron significativamente:

1. Métricas Clave

- **Precisión (Testing)**:
    - Aumentó a **96%** en el conjunto de prueba.
- **R^2 Mejorado**:
    - El coeficiente de determinación alcanzó un **0.97** en el conjunto de validación.
- **Pérdida Final**:
    - La pérdida mínima en validación se redujo a **0.030**.

Conclusiones Generales

El proyecto demostró la efectividad de las CNNs para tareas de clasificación en imágenes. A continuación, se destacan los puntos clave:

1. **Eficiencia del Modelo**:
    - ResNet18 mostró un excelente desempeño al generalizar sobre datos no vistos, alcanzando una precisión del 96%.
2. **Impacto de las Mejoras**:

- Las técnicas de regularización, programación de la tasa de aprendizaje y augmentación de datos contribuyeron significativamente a mejorar el desempeño del modelo.

3. **Limitaciones**:
   - El dataset contenía imágenes con condiciones desafiantes, como iluminación pobre y perspectivas no convencionales, lo que generó algunos errores.
4. **Futuras Direcciones**:
   - Expandir el dataset con más imágenes diversas podría ayudar a mejorar aún más la robustez del modelo.
   - Probar arquitecturas más avanzadas, como ResNet50, para comparar su desempeño con ResNet18.

Referencias

1. **Dataset de Imágenes de Patos**
   - N. Alicenkbaytop. (2023). *Duck Images Dataset*. Kaggle. Disponible en: https://www.kaggle.com/datasets/alicenkbaytop/duckimages
2. **Dataset de Imágenes de Gansos**
   - CV Datasets. (2023). *Goose Image Classification Dataset*. Disponible en: https://images.cv/dataset/goose-image-classificationdataset