Deep Learning para clasificación

Sistemas inteligentes para la Gestión en la Empresa

Jose Antonio Padial Molina
Ilyas Zgaoula
Manzambi António k. Doge

June 4, 2023

Contents

List of Figures

1 Introducción

El aprendizaje profundo, o Deep Learning, es una rama del campo de la inteligencia artificial que se ha vuelto extremadamente prominente en los últimos años. Su capacidad para abordar problemas complejos de clasificación ha generado un gran interés en la comunidad científica y en la industria.

El objetivo principal del aprendizaje profundo es enseñar a las computadoras a aprender y extraer características de manera automática a partir de grandes cantidades de datos. Esto se logra mediante el uso de algoritmos conocidos como redes neuronales artificiales, que están diseñadas para imitar el funcionamiento del cerebro humano.

En la tarea de clasificación, el aprendizaje profundo se enfoca en entrenar modelos capaces de asignar automáticamente etiquetas o categorías a un conjunto de datos. Estas categorías pueden ser tan simples como "sí" o "no", o tan complejas como reconocer objetos específicos en imágenes o traducir texto en diferentes idiomas.

Una de las características distintivas del aprendizaje profundo es su capacidad para aprender representaciones de alto nivel y abstractas a partir de datos sin procesar. En lugar de requerir que los expertos humanos diseñen manualmente características relevantes, como en los enfoques tradicionales de aprendizaje automático, el aprendizaje profundo extrae características relevantes automáticamente a través de capas de neuronas artificiales interconectadas.

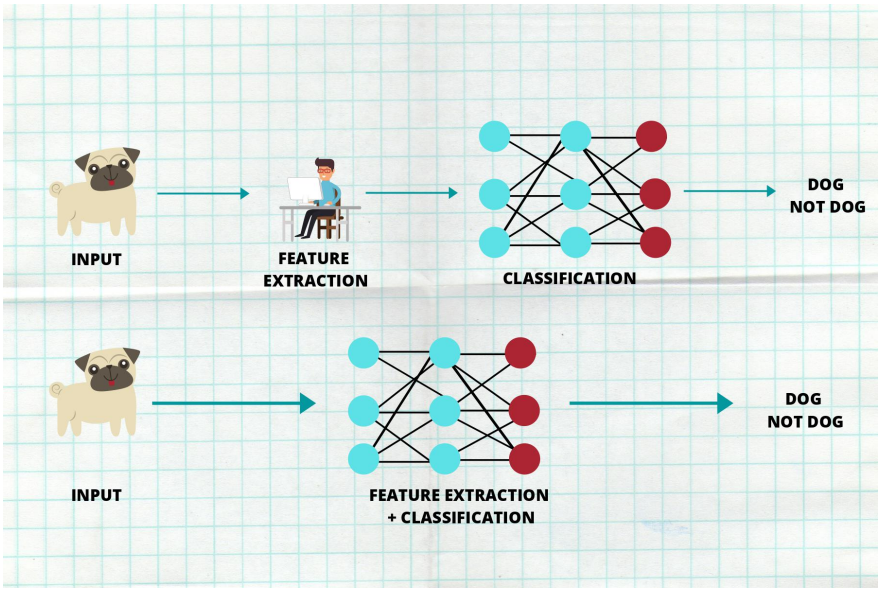

Figure 1: Deep Learning

Estas capas, llamadas capas ocultas, procesan secuencialmente los datos de entrada y aprenden a identificar características cada vez más abstractas a medida que se profundiza en la red neuronal. En última instancia, esta capacidad permite a los modelos de aprendizaje profundo realizar clasificaciones más precisas y sofisticadas en una amplia gama de dominios de aplicación.

Para entrenar una red neuronal profunda, se utiliza un conjunto de datos de entrenamiento etiquetado, donde las etiquetas representan las categorías o clases deseadas. Durante el entrenamiento, la red ajusta los pesos y los sesgos de las neuronas para minimizar la diferencia entre las predicciones de la red y las etiquetas verdaderas. Este proceso de ajuste se realiza mediante algoritmos de optimización, como la retropropagación del error.

A medida que se entrenan redes neuronales más profundas y se disponen de conjuntos de datos más grandes, el aprendizaje profundo ha demostrado un rendimiento notable en una amplia gama de tareas

de clasificación, como reconocimiento de imágenes, reconocimiento de voz, procesamiento del lenguaje natural, entre otros.

El aprendizaje profundo ha revolucionado la forma en que abordamos la clasificación de datos al permitir que las máquinas aprendan características y representaciones complejas de manera automática. Su capacidad para extraer información relevante de grandes volúmenes de datos ha impulsado avances significativos en una variedad de campos, y se espera que continúe desempeñando un papel fundamental en la inteligencia artificial en el futuro.

2 Deep Learning con PyTorch

PyTorch es una biblioteca de código abierto para aprendizaje profundo que se ha vuelto muy popular en la comunidad científica y en la industria. Ofrece un marco flexible y de alto rendimiento para desarrollar y entrenar redes neuronales profundas, lo que lo convierte en una herramienta poderosa para la implementación de algoritmos de aprendizaje profundo.

El aprendizaje profundo, o Deep Learning, es una rama de la inteligencia artificial que se enfoca en entrenar modelos capaces de aprender automáticamente a partir de grandes cantidades de datos. Estos modelos están compuestos por múltiples capas de neuronas artificiales interconectadas, que procesan los datos de entrada y extraen características cada vez más abstractas a medida que se profundiza en la red.

PyTorch proporciona una interfaz intuitiva y fácil de usar para construir y entrenar redes neuronales profundas. Una de las características distintivas de PyTorch es su enfoque en la programación dinámica, lo que significa que los modelos pueden construirse y modificarse de manera más flexible durante el tiempo de ejecución. Esto permite una mayor flexibilidad y facilidad para experimentar con diferentes arquitecturas de red y algoritmos de aprendizaje.

Además de su enfoque en la programación dinámica, PyTorch ofrece una amplia gama de herramientas y utilidades para facilitar el desarrollo de modelos de aprendizaje profundo. Proporciona una variedad de capas y funciones predefinidas, optimizadores de gradiente descendente estocástico (SGD) y funciones de pérdida comunes. También cuenta con una sólida comunidad de usuarios y desarrolladores que contribuyen con extensiones y paquetes adicionales, lo que amplía aún más su funcionalidad y versatilidad.

Otra característica destacada de PyTorch es su capacidad para aprovechar el procesamiento en GPU. El uso de GPU acelera significativamente el cálculo de las operaciones matemáticas requeridas durante el entrenamiento de modelos profundos, lo que permite un entrenamiento más rápido y eficiente.

PyTorch es una biblioteca de aprendizaje profundo de código abierto que ofrece una amplia gama de herramientas y una interfaz intuitiva para desarrollar y entrenar redes neuronales profundas. Su enfoque en la programación dinámica y su capacidad para aprovechar el poder de procesamiento de la GPU lo convierten en una opción popular entre los investigadores y profesionales del aprendizaje profundo. Con PyTorch, es posible construir y entrenar modelos sofisticados y altamente eficientes para abordar una amplia variedad de problemas de clasificación, reconocimiento de patrones y procesamiento del lenguaje natural, entre otros.

3 Análisis Exploratorio, Particionamiento de Datos y Clasificación Multiclase

En el campo del aprendizaje automático y la minería de datos, el análisis exploratorio, el particionamiento de datos y la clasificación multiclase son elementos esenciales para comprender, preparar y utilizar conjuntos de datos de manera efectiva. Estos conceptos desempeñan un papel fundamental en el proceso de construcción y evaluación de modelos de clasificación multiclase.

El análisis exploratorio de datos se refiere al proceso de examinar y comprender las características y patrones presentes en un conjunto de datos. Esto implica realizar visualizaciones, resúmenes estadísticos y cálculos para obtener información sobre la distribución de los datos, la presencia de valores atípicos,

la correlación entre variables y otros aspectos relevantes. El análisis exploratorio de datos ayuda a identificar patrones, tendencias y posibles desafíos que pueden surgir durante el modelado y la clasificación multiclase.

El particionamiento de datos se refiere a la división de un conjunto de datos en subconjuntos separados para diferentes propósitos. El particionamiento comúnmente se realiza en conjuntos de entrenamiento, validación y prueba. El conjunto de entrenamiento se utiliza para entrenar el modelo, el conjunto de validación se utiliza para ajustar los hiperparámetros del modelo y el conjunto de prueba se utiliza para evaluar el rendimiento final del modelo en datos no vistos previamente. El particionamiento de datos garantiza que el modelo se evalúe de manera imparcial y proporciona una estimación realista de su rendimiento en situaciones del mundo real.

Figure 2: Analisis exploratorio de datos

La clasificación multiclase es una tarea en la que el objetivo es asignar instancias de datos a más de dos clases posibles. A diferencia de la clasificación binaria, donde solo hay dos categorías, la clasificación multiclase implica la predicción de múltiples clases. Esto puede incluir la clasificación de imágenes en diferentes categorías, la clasificación de documentos en diferentes temas o cualquier otro escenario en el que se necesite asignar una instancia a una clase específica.

Para abordar la clasificación multiclase, se utilizan diversos algoritmos y técnicas. Algunos algoritmos populares incluyen las máquinas de vectores de soporte (SVM), los árboles de decisión, los bosques aleatorios, los algoritmos de clasificación basados en reglas y las redes neuronales. Cada algoritmo tiene sus propias ventajas y desventajas, y es importante seleccionar el enfoque más adecuado según el conjunto de datos y los requisitos del problema.

El análisis exploratorio, el particionamiento de datos y la clasificación multiclase son componentes fundamentales en el proceso de construcción y evaluación de modelos de clasificación multiclase. El análisis exploratorio de datos proporciona información valiosa para comprender la estructura y los patrones presentes en los datos. El particionamiento de datos garantiza una evaluación imparcial y realista del rendimiento del modelo, mientras que la clasificación multiclase permite la asignación de instancias a múltiples clases posibles. Al combinar estos conceptos y técnicas, es posible desarrollar modelos de clasificación multiclase precisos y eficientes en una amplia gama de aplicaciones.

4 Ajuste de hiperparámetros, topología de la red, función de coste y optimizador

El ajuste de hiperparámetros es un proceso importante en el aprendizaje automático que implica la selección de un conjunto óptimo de hiperparámetros para un modelo de aprendizaje automático. Los hiperparámetros son valores que se establecen antes del entrenamiento del modelo y que no se modifican durante el entrenamiento. El ajuste de hiperparámetros se realiza para mejorar el rendimiento del modelo y reducir el error de predicción.

La topología de la red se refiere a la estructura de la red neuronal, es decir, cómo se conectan las neuronas entre sí. La topología de la red puede afectar significativamente el rendimiento del modelo. Por ejemplo, una red neuronal con una topología más profunda puede ser capaz de aprender características más complejas que una red neuronal con una topología más superficial.

La función de coste es una medida del error del modelo en la tarea que está realizando. El objetivo del entrenamiento del modelo es minimizar la función de coste. La elección de una función de coste adecuada es importante para garantizar que el modelo esté optimizado para la tarea en cuestión.

El optimizador es un algoritmo utilizado para ajustar los pesos y sesgos de las neuronas en la red neuronal durante el entrenamiento. El objetivo del optimizador es minimizar la función de coste. Hay varios optimizadores disponibles, cada uno con sus propias ventajas y desventajas.

5 GPU vs CPU en Deep Learnig

Utilizar la GPU en lugar de la CPU para el entrenamiento y la inferencia en tareas de deep learning puede ofrecer varias ventajas significativas. Aquí hay una comparativa de los beneficios de utilizar la GPU en lugar de la CPU:

- Mayor velocidad de cálculo: La GPU está especialmente diseñada para realizar cálculos en paralelo y puede realizar operaciones matemáticas en grandes volúmenes de datos de manera mucho más rápida que la CPU. Esto acelera el tiempo de entrenamiento de los modelos de deep learning, lo que permite experimentar con redes más grandes y entrenar durante más épocas.
- Mayor capacidad de procesamiento: Las GPU modernas tienen una arquitectura altamente paralela con miles de núcleos de procesamiento, lo que les permite manejar una gran cantidad de cálculos simultáneamente. Esto es especialmente beneficioso en tareas intensivas en cómputo, como el procesamiento de imágenes o la aplicación de algoritmos complejos de deep learning.
- Soporte para bibliotecas de deep learning: Muchas bibliotecas populares de deep learning, como TensorFlow y PyTorch, están optimizadas para utilizar la GPU. Estas bibliotecas aprovechan las capacidades de procesamiento paralelo de la GPU y proporcionan interfaces sencillas para la programación y entrenamiento de modelos de deep learning en la GPU.
- Capacidad para trabajar con conjuntos de datos más grandes: La GPU tiene una memoria mucho más grande en comparación con la CPU, lo que permite manejar conjuntos de datos más grandes durante el entrenamiento de modelos. Esto es especialmente importante en problemas de deep learning donde se requiere una gran cantidad de datos para lograr un rendimiento óptimo.
- Mayor flexibilidad en el diseño del modelo: La GPU permite experimentar con arquitecturas más complejas y profundas, ya que la capacidad de procesamiento y la memoria adicional facilitan el entrenamiento de modelos más grandes y más profundos.
- Entrenamiento distribuido: Con múltiples GPUs, es posible realizar entrenamiento distribuido, lo que implica dividir el trabajo de entrenamiento en varias GPUs. Esto puede acelerar aún más el tiempo de entrenamiento y permitir el procesamiento de conjuntos de datos aún más grandes.

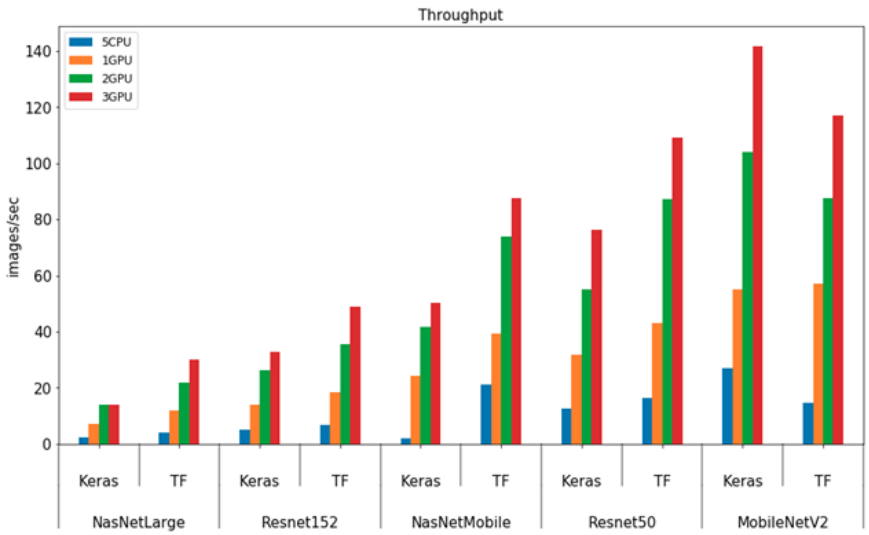

Figure 3: CPU vs GPU en Deep learning

La GPU es altamente beneficiosa en el ámbito del deep learning debido a su capacidad de procesamiento masivo y paralelo, lo que permite una mayor velocidad y capacidad de procesamiento en comparación con la CPU. Esto resulta en un tiempo de entrenamiento más rápido, la capacidad de trabajar con conjuntos de datos más grandes y la posibilidad de explorar arquitecturas de modelos más complejas.

6 Resnet50

ResNet-50 es una arquitectura de red neuronal convolucional (CNN) profunda que se utiliza comúnmente en tareas de visión por computadora y reconocimiento de imágenes. Fue propuesta por Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun en 2015, y es una versión mejorada de la arquitectura ResNet original (ResNet-34).

El nombre "ResNet" proviene de "Residual Network", ya que esta arquitectura utiliza bloques residuales para superar el problema de la degradación del rendimiento a medida que se agregan más capas a una red neuronal profunda.

A continuación se presenta un marco teórico básico de la arquitectura ResNet-50:

1. **Bloques residuales:**
   - El bloque residual es la unidad básica de construcción en ResNet-50.
   - Consiste en una serie de capas convolucionales, seguidas de una función de activación, como ReLU (Rectified Linear Unit).
   - La entrada de un bloque residual se pasa a través de la capa convolucional y luego se agrega a la salida final del bloque. Esta suma se pasa a través de otra función de activación antes de ser utilizada como entrada para el siguiente bloque.
2. **Capas convolucionales:**
   - ResNet-50 utiliza capas convolucionales para extraer características de las imágenes de entrada.
   - Utiliza convoluciones 2D con diferentes tamaños de kernel (3x3 y 1x1) para capturar diferentes niveles de detalles en la imagen.

- Se utilizan capas de agrupación máxima (max pooling) para reducir el tamaño espacial de las características y obtener una representación más compacta.

3. **Bloques de identidad y bloques convolucionales:**
   - ResNet-50 utiliza dos tipos de bloques residuales: bloques de identidad y bloques convolucionales.
   - Los bloques de identidad se utilizan cuando la entrada y la salida tienen la misma dimensión espacial.
   - Los bloques convolucionales se utilizan cuando la dimensión espacial de la entrada y la salida es diferente. Estos bloques incluyen una capa convolucional 1x1 adicional para ajustar las dimensiones.
4. **Capa de promedio global:**
   - Al final de la red ResNet-50, se utiliza una capa de promedio global para convertir la representación espacial de las características en una representación vectorial unidimensional.
   - Esta capa calcula la media de cada canal de características a lo largo de todas las posiciones espaciales, generando un vector de características global.
5. **Capa totalmente conectada:**
   - Después de la capa de promedio global, se agrega una capa totalmente conectada con una función de activación softmax para realizar la clasificación de las imágenes en diferentes clases.

ResNet-50 ha demostrado un rendimiento sobresaliente en tareas de clasificación de imágenes y ha sido ampliamente utilizado como base para la extracción de características en transferencia de aprendizaje y en la implementación de muchas aplicaciones de visión por computadora.

Es importante destacar que este es solo un marco teórico básico de la arquitectura ResNet-50. La implementación completa incluye más detalles, como la estructura precisa de los bloques residuales y el número de filtros en cada capa convolucional, pero estos conceptos principales proporcionan una comprensión general de cómo funciona ResNet-50.

6.1 Ventajas de Resnet50

ResNet-50 presenta varias ventajas en comparación con otras arquitecturas de redes neuronales convolucionales (CNN). A continuación, se mencionan algunas de las ventajas clave de ResNet-50:

- Resolución del problema de degradación del rendimiento: ResNet-50 introduce bloques residuales que permiten entrenar con éxito redes neuronales mucho más profundas que las arquitecturas tradicionales. Estos bloques residuales ayudan a mitigar el problema de la degradación del rendimiento que se produce al agregar más capas a una red neuronal profunda.
- Mejora de la precisión y el rendimiento: Al ser una arquitectura más profunda, ResNet-50 puede capturar características más complejas y abstractas en las imágenes. Esto lleva a una mayor precisión y rendimiento en tareas de clasificación de imágenes, detección de objetos y otras tareas de visión por computadora.
- Transferencia de aprendizaje: ResNet-50 se ha utilizado ampliamente como una red preentrenada en grandes conjuntos de datos, como ImageNet. Esto permite aprovechar el conocimiento aprendido en tareas de clasificación de imágenes y aplicarlo a tareas específicas con conjuntos de datos más pequeños. La transferencia de aprendizaje con ResNet-50 ha demostrado ser efectiva en la mejora del rendimiento y la reducción del tiempo de entrenamiento en muchas aplicaciones.

- Flexibilidad y adaptabilidad: ResNet-50 puede adaptarse y aplicarse a una variedad de tareas de visión por computadora, como clasificación de imágenes, detección de objetos, segmentación semántica, entre otras. La estructura modular de ResNet-50 permite modificarla y ajustarla según las necesidades específicas de la tarea.
- Eficiencia computacional: A pesar de ser más profunda, ResNet-50 logra mantener una eficiencia computacional razonable. Esto se debe a la incorporación de capas convolucionales de tamaño reducido (1x1) que ayudan a reducir la complejidad computacional sin sacrificar el rendimiento.

ResNet-50 ofrece una solución eficaz para el entrenamiento de redes neuronales profundas en tareas de visión por computadora. Su capacidad para superar la degradación del rendimiento, su precisión mejorada, su capacidad de transferencia de aprendizaje y su flexibilidad lo convierten en una elección popular y efectiva para muchas aplicaciones de reconocimiento de imágenes.

7 Descripción de las técnicas utilizadas

En el código proporcionado, se están utilizando dos técnicas para mejorar el aprendizaje del modelo:

Aumento de datos (Data augmentation): Se aplica una serie de transformaciones aleatorias a las imágenes de entrenamiento durante el entrenamiento del modelo. Estas transformaciones incluyen el recorte aleatorio, volteo horizontal, ajuste de color (brillo, contraste, saturación, matiz) y rotación aleatoria. Estas transformaciones ayudan a aumentar la diversidad de los datos de entrenamiento y a mejorar la capacidad del modelo para generalizar a nuevas imágenes. La técnica se implementa utilizando la clase transforms de torchvision. Se crea una composición de transformaciones (train_transform) que incluye las transformaciones mencionadas, y luego se asigna esta composición al atributo transform del conjunto de datos de entrenamiento (train_data.dataset.image_data.transform).

Ajuste de tasa de aprendizaje (Learning rate scheduling): Se utiliza un programador de tasa de aprendizaje (scheduler) para ajustar dinámicamente la tasa de aprendizaje durante el entrenamiento. En este caso, se utiliza un programador de tasa de aprendizaje basado en el paso (StepLR) que reduce la tasa de aprendizaje por un factor de gamma después de cada cierto número de épocas (step_size). Esto ayuda a mejorar la convergencia del modelo y a evitar oscilaciones o estancamientos en la optimización. El programador de tasa de aprendizaje se utiliza llamando al método scheduler.step() después de cada época de entrenamiento.

El código implementa la técnica de aumento de datos para aumentar la diversidad de los datos de entrenamiento y mejorar la generalización del modelo, y utiliza el ajuste de tasa de aprendizaje para mejorar la convergencia y el rendimiento del modelo durante el entrenamiento.

7.1 Data augmentation

La técnica del aumento de datos es una estrategia fundamental en el campo del aprendizaje profundo (deep learning) que busca mejorar el rendimiento y la generalización de los modelos de inteligencia artificial. A medida que los conjuntos de datos se vuelven cada vez más grandes y complejos, el aumento de datos se ha convertido en una práctica ampliamente utilizada para evitar el sobreajuste y mejorar la capacidad de los modelos para aprender patrones y características relevantes.

En su esencia, el aumento de datos implica generar nuevas muestras de entrenamiento mediante la aplicación de transformaciones o perturbaciones a los datos existentes. Estas transformaciones pueden incluir rotaciones, desplazamientos, reflejos, cambios de escala, cambios de contraste, entre otros. Al aplicar estas transformaciones, se generan nuevas versiones de las imágenes o los datos originales, lo que aumenta la cantidad y la diversidad de ejemplos disponibles para entrenar el modelo.

La idea detrás del aumento de datos es que, al exponer el modelo a una variedad más amplia de casos, se mejora su capacidad para reconocer patrones y características importantes, lo que a su vez ayuda a evitar

el sobreajuste. Por ejemplo, en el caso de la clasificación de imágenes, el aumento de datos puede incluir rotar las imágenes en diferentes ángulos, cambiar su escala, ajustar el brillo o la saturación, e incluso agregar ruido aleatorio. Estas transformaciones imitan las variaciones y las perturbaciones que se pueden encontrar en el mundo real, lo que permite al modelo generalizar mejor a nuevas situaciones y datos de prueba.

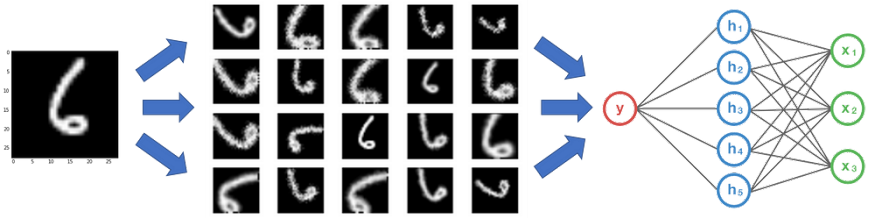

Figure 4: Data augmentation

El aumento de datos también puede ser particularmente útil en escenarios donde los conjuntos de datos son limitados o desequilibrados en términos de la distribución de clases. Al generar nuevas muestras a partir de los datos existentes, se puede equilibrar el conjunto de entrenamiento y proporcionar al modelo una representación más equitativa de cada clase, lo que mejora su capacidad para aprender y clasificar adecuadamente.

La técnica del aumento de datos en el aprendizaje profundo es una estrategia esencial para mejorar el rendimiento y la generalización de los modelos. Al generar muestras adicionales mediante la aplicación de transformaciones y perturbaciones a los datos existentes, se aumenta la diversidad y la cantidad de ejemplos disponibles para entrenar el modelo, lo que ayuda a evitar el sobreajuste y mejora su capacidad para reconocer patrones y características relevantes en nuevos datos de prueba.

7.2 Learning rate scheduling

La técnica del ajuste de tasa de aprendizaje es un aspecto esencial en el campo del aprendizaje profundo (deep learning) que permite optimizar el proceso de entrenamiento de los modelos de inteligencia artificial. La tasa de aprendizaje es un hiperparámetro crítico que determina la cantidad de ajuste que se aplica a los pesos de las neuronas en cada iteración durante el entrenamiento de un modelo. El ajuste adecuado de la tasa de aprendizaje es crucial para lograr una convergencia eficiente y obtener resultados óptimos.

En términos simples, la tasa de aprendizaje controla el tamaño de los pasos que el modelo da al ajustar sus pesos en función del error calculado durante el entrenamiento. Una tasa de aprendizaje alta puede llevar a que el modelo salte rápidamente sobre el mínimo global o local en la función de pérdida, lo que resulta en una convergencia rápida pero puede hacer que el modelo no alcance su óptimo global. Por otro lado, una tasa de aprendizaje muy baja puede hacer que el modelo converja lentamente o quede estancado en mínimos locales subóptimos.

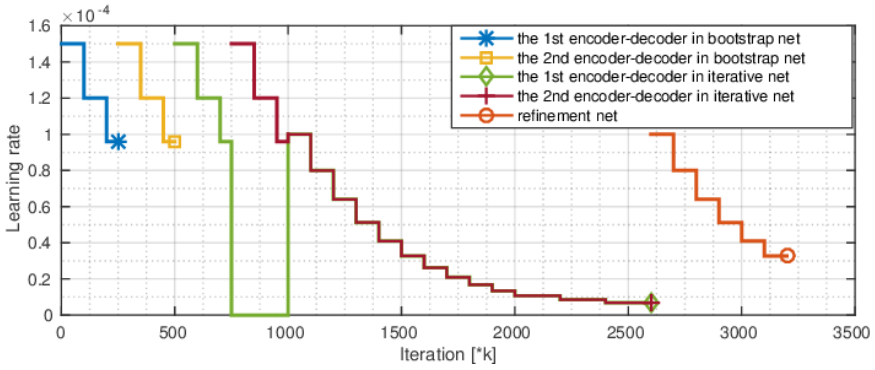

Figure 5: Learning rate scheduling

La técnica del ajuste de tasa de aprendizaje aborda este desafío ajustando dinámicamente la tasa de aprendizaje durante el entrenamiento. Existen diferentes enfoques para realizar este ajuste, como el descenso de aprendizaje constante, el descenso de aprendizaje por etapas (step decay), el descenso de aprendizaje por calendario (calendar decay), el descenso de aprendizaje por impulso (momentum decay) y el descenso de aprendizaje adaptativo (adaptive learning rate), entre otros.

El descenso de aprendizaje constante es el enfoque más simple, donde se mantiene una tasa de aprendizaje fija a lo largo del entrenamiento. Sin embargo, esto puede no ser óptimo, ya que diferentes etapas del entrenamiento pueden requerir tasas de aprendizaje diferentes. Por ejemplo, al inicio del entrenamiento, puede ser beneficioso utilizar una tasa de aprendizaje alta para un rápido ajuste inicial, mientras que a medida que se acerca al óptimo, puede ser necesario reducir la tasa de aprendizaje para un ajuste más preciso y suave.

El descenso de aprendizaje por etapas y el descenso de aprendizaje por calendario son enfoques que reducen la tasa de aprendizaje en momentos específicos durante el entrenamiento. Por ejemplo, se puede reducir la tasa de aprendizaje a la mitad después de un número determinado de épocas o después de alcanzar cierto umbral de pérdida. Esto permite al modelo ajustar sus pesos de manera más precisa a medida que se acerca a la convergencia.

El descenso de aprendizaje por impulso incorpora un término de impulso o momento que afecta la actualización de los pesos. Este enfoque permite una convergencia más rápida y estable, ya que se tienen en cuenta las actualizaciones anteriores de los pesos. El descenso de aprendizaje adaptativo es otra técnica popular que ajusta la tasa de aprendizaje de forma automática según la información local del gradiente. Al adaptar la tasa de aprendizaje a cada peso individual, el modelo puede ajustar los pesos de manera más precisa y eficiente.

El ajuste de tasa de aprendizaje es una técnica fundamental en el aprendizaje profundo que permite optimizar el proceso de entrenamiento de los modelos. Al ajustar la tasa de aprendizaje de manera dinámica durante el entrenamiento, se puede mejorar la convergencia y obtener mejores resultados. La elección adecuada del enfoque de ajuste de tasa de aprendizaje depende del problema y del modelo específico, y es crucial para lograr un rendimiento óptimo en el aprendizaje profundo.

8 MLflow

MLflow es una plataforma de código abierto desarrollada para ayudar en el ciclo de vida de los proyectos de aprendizaje automático (machine learning), incluido el aprendizaje profundo (deep learning). Proporciona un conjunto de herramientas y funcionalidades que permiten a los científicos de datos y a los ingenieros de aprendizaje profundo realizar un seguimiento, organizar, reproducir y compartir experimentos y modelos de manera eficiente.

En el contexto del aprendizaje profundo, donde los proyectos pueden ser complejos y requieren la experimentación con diferentes arquitecturas de red, hiperparámetros y conjuntos de datos, MLflow se convierte en una herramienta valiosa. Su objetivo principal es mejorar la reproducibilidad, la colaboración y la gestión de versiones en los proyectos de aprendizaje profundo.

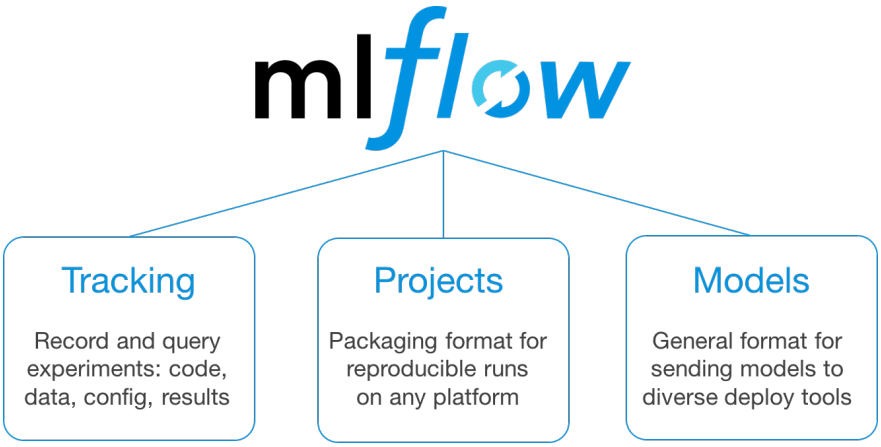

Figure 6: MLflow

La teoría de MLflow se basa en cuatro componentes principales:

- Seguimiento de experimentos: MLflow permite realizar un seguimiento de los experimentos de aprendizaje profundo registrando y almacenando metadatos importantes, como la configuración del modelo, los hiperparámetros utilizados, las métricas de rendimiento y los conjuntos de datos utilizados. Esto facilita la comparación y el análisis de resultados, así como la reproducción de experimentos anteriores.
- Gestión de modelos: MLflow proporciona una forma organizada de gestionar y registrar los modelos entrenados. Esto incluye el seguimiento de las versiones del modelo, así como la capacidad de empaquetar y distribuir los modelos entrenados para su implementación en producción.
- Reproducibilidad: MLflow permite registrar todas las dependencias y configuraciones necesarias para reproducir un experimento o un modelo específico. Esto garantiza que otros científicos de datos o miembros del equipo puedan recrear exactamente el mismo entorno y obtener los mismos resultados.
- Colaboración y compartición: MLflow facilita la colaboración entre los miembros del equipo al proporcionar una interfaz para compartir experimentos, modelos y resultados. Los científicos de datos pueden colaborar de manera más efectiva, revisar y comentar el trabajo de otros y compartir su trabajo con la comunidad más amplia de aprendizaje automático.

MLflow es una plataforma versátil que se utiliza en el aprendizaje profundo para realizar un seguimiento de experimentos, gestionar modelos, garantizar la reproducibilidad y fomentar la colaboración. Al proporcionar una estructura organizada y herramientas para el ciclo de vida del aprendizaje automático, MLflow ayuda a los equipos a ser más eficientes y efectivos en sus proyectos de aprendizaje profundo.

9 Transferencia de aprendizaje o fine tuning

La transferencia de aprendizaje, también conocida como fine tuning, es una técnica utilizada en el campo del aprendizaje automático y, en particular, en el ámbito del Deep Learning. Consiste en aprovechar los

conocimientos adquiridos por un modelo previamente entrenado en una tarea relacionada y utilizar esa información para mejorar el desempeño en una nueva tarea específica.

En lugar de entrenar un modelo desde cero, la transferencia de aprendizaje parte de una red neuronal preentrenada, que ha sido entrenada en un conjunto de datos grande y general, como ImageNet, con miles de categorías diferentes. Esta red preentrenada ha aprendido a extraer características relevantes de las imágenes y ha capturado patrones visuales comunes.

El proceso de fine tuning implica tomar esta red preentrenada y ajustarla en una tarea de interés específica. Esto se logra modificando los pesos de las capas del modelo para adaptarlos a la nueva tarea, mientras se preservan las características aprendidas anteriormente. A menudo, solo se modifican las capas finales o se añaden nuevas capas adicionales para adaptar el modelo a la tarea de clasificación deseada.

El principal beneficio de la transferencia de aprendizaje es que permite entrenar modelos más efectivos y con menos datos de entrenamiento. Al aprovechar los conocimientos previos del modelo preentrenado, se evita tener que entrenar desde cero, lo que requiere grandes conjuntos de datos y recursos computacionales significativos.

Además, la transferencia de aprendizaje ayuda a evitar el sobreajuste en casos en los que no se dispone de suficientes datos de entrenamiento para la nueva tarea. Al iniciar con un modelo que ya ha aprendido características generales de una amplia gama de imágenes, se puede lograr una mejor generalización y evitar la falta de datos en la tarea específica.

Es importante seleccionar cuidadosamente la red preentrenada que se utilizará para el fine tuning, eligiendo una que haya sido entrenada en un dominio similar o relacionado. De esta manera, las características aprendidas en la red preentrenada serán más relevantes para la nueva tarea.

La transferencia de aprendizaje o fine tuning es una técnica poderosa en el campo del Deep Learning que permite aprovechar los conocimientos adquiridos por un modelo previamente entrenado para mejorar el desempeño en una nueva tarea específica. Esta técnica ha demostrado ser efectiva en diversas aplicaciones, permitiendo entrenar modelos más precisos y eficientes con menos datos de entrenamiento.

10 Métodos de tipo ensemble e híbridos

Los métodos de tipo ensemble e híbridos son técnicas utilizadas en el campo del aprendizaje automático para mejorar el rendimiento y la estabilidad de los modelos predictivos. Estos enfoques se basan en combinar múltiples modelos individuales en lugar de confiar en un solo modelo. A continuación, se presenta un marco teórico que explica estos métodos

10.1 Ensemble Learning (Aprendizaje en conjunto)

El ensemble learning, también conocido como aprendizaje en conjunto, es una técnica que combina múltiples modelos de aprendizaje automático para producir una predicción más precisa y robusta. En lugar de confiar en un solo modelo, se utilizan varios modelos y se combinan sus predicciones para llegar a una decisión final.

Tipos de Ensemble Learning:

- **Bagging:** El bagging es un método que se basa en el muestreo bootstrap para generar múltiples conjuntos de datos de entrenamiento a partir del conjunto de datos original. Luego, se entrena un modelo base en cada uno de estos conjuntos de datos y se promedian sus predicciones para obtener una predicción final. Ejemplos de algoritmos basados en bagging son el Random Forest y el Extra Trees.
- **Boosting:** El boosting es un método secuencial que se enfoca en mejorar modelos débiles en etapas sucesivas. En cada etapa, se da más peso a los ejemplos clasificados incorrectamente para que

los modelos posteriores se enfoquen en corregir los errores cometidos por los modelos anteriores. Ejemplos de algoritmos basados en boosting son AdaBoost, Gradient Boosting y XGBoost.

- **Stacking:** El stacking combina múltiples modelos entrenados individualmente utilizando un modelo meta-aprendizaje, también conocido como meta-modelo. Los modelos base realizan predicciones sobre un conjunto de datos de validación y estas predicciones se utilizan como características de entrada para el modelo meta-aprendizaje, que genera la predicción final. El stacking permite que los modelos aprendan de las fortalezas y debilidades de los demás.
- **Voting:** El voting, también conocido como voto mayoritario, combina las predicciones de múltiples modelos mediante votación. Cada modelo individual emite su propia predicción y la predicción final se determina por mayoría de votos. Puede ser voto duro (hard voting), donde la etiqueta con más votos es seleccionada, o voto suave (soft voting), donde se utilizan las probabilidades de cada clase y se promedian.

10.2 Métodos híbridos

Los métodos híbridos combinan características o conceptos de diferentes algoritmos o enfoques de aprendizaje automático para mejorar el rendimiento predictivo. Estos métodos aprovechan las fortalezas de diferentes técnicas y las fusionan en un solo modelo o enfoque.

Ejemplos de métodos híbridos:

- **Ensemble de modelos y reglas:** Este enfoque combina un modelo predictivo basado en aprendizaje automático con reglas expertas o conocimiento humano. El modelo de aprendizaje automático puede capturar patrones complejos en los datos, mientras que las reglas expertas pueden aportar conocimientos específicos del dominio y mejorar la interpretabilidad del modelo.
- **Ensemble de características:** En este enfoque, se extraen características de diferentes algoritmos o técnicas de extracción de características y se combinan para formar un conjunto de características más completo. Estas características combinadas se utilizan para entrenar un modelo de aprendizaje automático y mejorar el rendimiento predictivo.
- **Ensemble de clasificadores y regresores:** En este método híbrido, se combinan clasificadores y regresores para abordar un problema de aprendizaje automático. Por ejemplo, se puede utilizar un clasificador para predecir una variable categórica y un regresor para predecir una variable numérica en un conjunto de datos.
- **Ensemble de técnicas de preprocesamiento:** Este enfoque combina diferentes técnicas de preprocesamiento de datos, como normalización, reducción de dimensionalidad o selección de características, para mejorar la calidad de los datos antes de aplicar un algoritmo de aprendizaje automático.

Los métodos híbridos permiten aprovechar las fortalezas y superar las limitaciones de diferentes técnicas de aprendizaje automático al combinarlas de manera inteligente. Al fusionar diferentes enfoques, se puede lograr un mejor rendimiento predictivo y una mayor adaptabilidad a diferentes escenarios y conjuntos de datos.

11 Explicación teórica del código desarrollado

El código realiza las siguientes operaciones:

1. Importación de bibliotecas: Se importan las bibliotecas necesarias para el código, como **torch** y **torchvision** para el manejo de redes neuronales y conjuntos de datos, **numpy** y **matplotlib** para el manejo de matrices y gráficos, y **mlflow** para el seguimiento de experimentos de aprendizaje automático.
2. Definición de la clase de conjunto de datos personalizado: Se define la clase **ImageDataWithAdditionalDataset** que hereda de **torch.utils.data.Dataset**. Esta clase se utiliza para cargar imágenes y datos adicionales en un solo conjunto de datos. Se utiliza la biblioteca **datasets.ImageFolder** para cargar las imágenes y se agrega información adicional al conjunto de datos.
3. Configuración de rutas de datos: Se definen las rutas de las carpetas que contienen los datos de imagen y datos adicionales.
4. Definición de transformaciones: Se define una secuencia de transformaciones que se aplicarán a las imágenes, como redimensionamiento, conversión a tensor y normalización.
5. Carga del conjunto de datos combinado: Se crea una instancia de la clase **ImageDataWithAdditionalDataset** y se pasan las rutas de las carpetas de datos y las transformaciones definidas.
6. Partición del conjunto de datos: El conjunto de datos se divide en conjuntos de entrenamiento, validación y prueba utilizando la función **random_split** de **torch.utils.data**.
7. Definición de los cargadores de datos: Se crean instancias de **DataLoader** para los conjuntos de entrenamiento, validación y prueba. Estos cargadores de datos se utilizan para iterar fácilmente sobre los lotes de datos durante el entrenamiento y la evaluación del modelo.
8. Configuración de la clasificación multiclase: Se carga el modelo preentrenado **resnet50** de **torchvision.models** y se ajusta para realizar la clasificación multiclase en función del número de clases en el conjunto de datos.
9. Ajuste de hiperparámetros, topología de la red, función de coste y optimizador: Se define el dispositivo de ejecución (CPU o GPU), la función de pérdida de entropía cruzada, el optimizador Adam y los hiperparámetros para el ajuste del modelo.
10. Aplicación de técnicas para la mejora del aprendizaje:
    - **Aumento de datos:** Se aplica un conjunto de transformaciones aleatorias al conjunto de entrenamiento para aumentar la diversidad y generalización del modelo.
    - **Ajuste de la tasa de aprendizaje:** Se utiliza un programador de tasa de aprendizaje para ajustar la tasa de aprendizaje durante el entrenamiento.
11. Inicio de un experimento de MLflow: Se inicia un experimento de MLflow para realizar un seguimiento de las métricas y parámetros relevantes durante el entrenamiento.
12. Entrenamiento del modelo: Se itera sobre las épocas de entrenamiento y los lotes de datos del conjunto de entrenamiento. Se calcula la pérdida, se realiza la propagación hacia atrás y se actualizan los pesos del modelo. Además, se calcula la precisión de validación y se registra en MLflow.
13. Finalización del experimento de MLflow: Se finaliza el experimento de MLflow para almacenar los registros y los artefactos generados durante el entrenamiento.
14. Representación gráfica del progreso del entrenamiento: Se trazan las curvas de pérdida de entrenamiento y precisión de validación a lo largo de las épocas.
15. Representación gráfica de la matriz de confusión: Se traza una matriz de confusión normalizada que muestra el rendimiento del modelo en la clasificación de las diferentes clases.
16. Guardado de las figuras en MLflow: Las figuras generadas se guardan como artefactos en MLflow.
17. Mostrar las figuras: Se muestran las figuras de progreso de entrenamiento y matriz de confusión.

El código carga un conjunto de datos de imágenes y datos adicionales, entrena un modelo de clasificación multiclase utilizando ResNet-50, aplica técnicas de mejora del aprendizaje, realiza un seguimiento de las métricas y parámetros con MLflow y muestra las curvas de progreso y la matriz de confusión resultantes.

11.1 Características del código implementado

Algunas de las mejores características de este código son:

- **Modularidad:** El código se divide en diferentes secciones y clases, lo que facilita la comprensión y el mantenimiento. Por ejemplo, la clase **ImageDataWithAdditionalDataset** se utiliza para cargar imágenes y datos adicionales, lo que permite una mayor flexibilidad en la estructura del dataset.
- **Uso de PyTorch y torchvision:** Se aprovecha la potencia de las bibliotecas PyTorch y torchvision para cargar los datos, definir la arquitectura del modelo (utilizando el modelo pre-entrenado ResNet-50) y aplicar transformaciones a las imágenes. Esto permite una implementación eficiente y sencilla del pipeline de entrenamiento y evaluación.
- **Aumento de datos:** Se utiliza la técnica de aumento de datos (data augmentation) en el conjunto de entrenamiento mediante transformaciones como el recorte aleatorio, la inversión horizontal, el ajuste de color y la rotación. Esto ayuda a aumentar la diversidad de los datos de entrenamiento y a mejorar la capacidad del modelo para generalizar.
- **Ajuste de la tasa de aprendizaje:** Se utiliza un optimizador Adam con una tasa de aprendizaje inicial de 0.001. Además, se aplica un ajuste de la tasa de aprendizaje utilizando un scheduler que reduce la tasa de aprendizaje en un factor de 0.1 cada 5 épocas. Esto puede ayudar a encontrar un equilibrio adecuado entre la velocidad de convergencia y la precisión del modelo.
- **Uso de MLflow:** Se utiliza la biblioteca MLflow para el registro de métricas y parámetros relevantes durante el entrenamiento del modelo. Esto facilita el seguimiento y la comparación de experimentos, así como el registro de artefactos como gráficas y matrices de confusión.
- **Ajuste fino de la red neuronal:** Se realiza un ajuste fino (fine-tuning) de la red neuronal preentrenada. Se congela la mayoría de los parámetros de la red y se ajusta solo la capa de clasificación final. Esto permite adaptar el modelo a la tarea específica de clasificación multiclase y acelerar el entrenamiento al reducir la cantidad de parámetros actualizables.
- **Aprendizaje con coste variable al código:** se crea una clase **VariableCostLoss** que hereda de **nn.Module** y reemplaza la función de pérdida. En el constructor de **VariableCostLoss**, se toman dos argumentos: **num_classes**, que es el número de clases en los datos, y **class_weights**, que es una lista de pesos para cada clase. Estos pesos pueden ser definidos por el usuario según la importancia relativa que se le quiera dar a cada clase. Luego, en el método **forward**, se utiliza **nn.CrossEntropyLoss** con el parámetro **weight** establecido como **class_weights** para aplicar los pesos de coste variable durante el cálculo de la pérdida.
- **Métodos de tipo ensemble e híbridos:** El ensamble por voto mayoritario permite aprovechar las fortalezas de múltiples modelos al tomar una decisión final sobre la clasificación de un ejemplo. Al combinar las predicciones de varios modelos, se puede mejorar la precisión y la estabilidad del sistema de clasificación. En lugar de depender únicamente de un solo modelo, el ensamble por voto mayoritario considera las opiniones de múltiples modelos y toma una decisión basada en la mayoría de votos.

Este código muestra buenas prácticas de implementación en el contexto del aprendizaje profundo, incluyendo el uso de bibliotecas populares, técnicas de mejora del aprendizaje y herramientas para el seguimiento y registro de experimentos.

11.2 Ventajas que proporciona el código desarrollado frente a otros

- **Modularidad:** El código está estructurado en funciones y clases, lo que facilita la comprensión, reutilización y mantenimiento del código. El uso de una clase personalizada **ImageDataWithAdditionalDataset** permite cargar conjuntos de datos de imágenes junto con datos adicionales relacionados.
- **Flexibilidad en el procesamiento de datos:** Se utilizan transformaciones de PyTorch (**transforms**) para aplicar una serie de operaciones en las imágenes, como cambiar su tamaño, convertirlas en tensores y normalizarlas. Esto permite una fácil manipulación de los datos y su preparación para el entrenamiento del modelo.
- **Manejo de datos de entrenamiento, validación y prueba:** El código divide automáticamente el conjunto de datos en conjuntos de entrenamiento, validación y prueba utilizando la función **random_split** de PyTorch. Además, se utilizan **DataLoader** para cargar los datos de forma eficiente y en lotes durante el entrenamiento.
- **Uso de modelos preentrenados:** El código utiliza el modelo de clasificación ResNet-50 preentrenado disponible en **torchvision.models**. Esto permite aprovechar el conocimiento previo del modelo entrenado en un gran conjunto de datos para mejorar el rendimiento y acelerar el entrenamiento en el conjunto de datos específico.
- **Entrenamiento y evaluación del modelo:** El código implementa un bucle de entrenamiento que itera sobre las épocas y los lotes de datos de entrenamiento. Durante el entrenamiento, se calcula la función de pérdida y se optimizan los parámetros del modelo utilizando el algoritmo de optimización Adam. Además, se realiza la evaluación del modelo en el conjunto de validación para medir la precisión y se muestra el progreso de entrenamiento en términos de pérdida y precisión en gráficos.
- **Uso de MLflow para el seguimiento y registro de experimentos:** MLflow se utiliza para registrar métricas, parámetros relevantes y artefactos, como las gráficas generadas durante el entrenamiento. Esto facilita el seguimiento y la comparación de diferentes experimentos y configuraciones de entrenamiento.
- **Visualización de resultados:** El código genera y muestra gráficas como la progresión de la pérdida de entrenamiento y la precisión de validación, así como la matriz de confusión normalizada. Esto ayuda a comprender y evaluar el rendimiento del modelo.

El código proporcionado tiene ventajas en términos de modularidad, flexibilidad en el procesamiento de datos, entrenamiento y evaluación del modelo, seguimiento de experimentos con MLflow y visualización de resultados. Estas características hacen que el código sea más legible, mantenible y efectivo para el desarrollo de modelos de aprendizaje automático en tareas de clasificación de imágenes.

11.3 Posibles mejores al código

- **Modularización del código:** Dividir el código en funciones más pequeñas y reutilizables para mejorar la legibilidad y la facilidad de mantenimiento. Por ejemplo, crear funciones separadas para cargar los datos, definir el modelo, entrenar y evaluar el modelo, y visualizar los resultados. Esto hará que el código sea más modular y más fácil de entender.
- **Uso de argparse para gestionar argumentos:** En lugar de definir los paths manualmente en el código, usar el módulo argparse para gestionar los argumentos de línea de comandos. De esta manera, pasar los paths de los datos como argumentos al ejecutar el script, lo que lo hace más flexible y fácil de reutilizar.
- **Uso de torchvision.datasets.ImageFolder:** En lugar de crear una clase de conjunto de datos personalizada, utilizar directamente ImageFolder de torchvision.datasets para cargar los datos de imagen. Esta clase ya maneja la estructura de carpetas y asigna automáticamente las etiquetas a las imágenes según la estructura de carpetas.

- **Uso de torchvision.transforms:** Puedes explorar más transformaciones disponibles en torchvision.transforms para realizar una mayor variedad de aumentos de datos, como recorte aleatorio, rotación aleatoria, cambio de perspectiva, etc. Esto puede ayudar a aumentar la variabilidad de los datos de entrenamiento y mejorar el rendimiento del modelo.
- **Uso de torch.nn.Sequential:** Si la arquitectura del modelo es lineal, utilizar torch.nn.Sequential en lugar de asignar manualmente num_features y reemplazar la capa final. Esto hace que el código sea más conciso y fácil de entender.
- **Uso de torchvision.models:** Aparte de ResNet, hay otros modelos preentrenados disponibles en torchvision.models que probar, como DenseNet, VGG, etc. Puedes experimentar con diferentes modelos y ver cómo afectan al rendimiento del modelo.
- **Uso de GPU para la inferencia:** Actualmente, el código verifica si la GPU está disponible al comienzo del script, pero no se utiliza para la inferencia. Puedes mover las imágenes de validación al dispositivo GPU utilizando images = images.to(device) antes de la inferencia para aprovechar al máximo el rendimiento de la GPU.
- **Uso de torch.no_grad():** Al calcular la precisión de validación y la matriz de confusión, utilizar el contexto de torch.no_grad() para desactivar el cálculo y el seguimiento de gradientes. Esto mejora la eficiencia y ahorra memoria.

12 La metodología X-NeSyL

La metodología X-NeSyL (eXplainable Neural Symbolic Learning) combina técnicas de aprendizaje profundo (neural) y lógica simbólica (symbolic) para construir modelos que sean tanto precisos como interpretables. El objetivo principal de X-NeSyL es crear modelos que puedan explicar sus predicciones de una manera comprensible para los seres humanos, lo que es especialmente útil en casos donde se requiere transparencia y confianza en los resultados del modelo.

En el caso específico de un modelo que utiliza los atributos "ala roja" y "pico amarillo" como entrada, podría ser utilizado en el contexto de clasificación de aves. Por ejemplo, supongamos que tenemos un conjunto de datos que contiene imágenes de aves junto con sus atributos. El modelo X-NeSyL tomaría estas imágenes y los atributos "ala roja" y "pico amarillo" como entrada para realizar una clasificación.

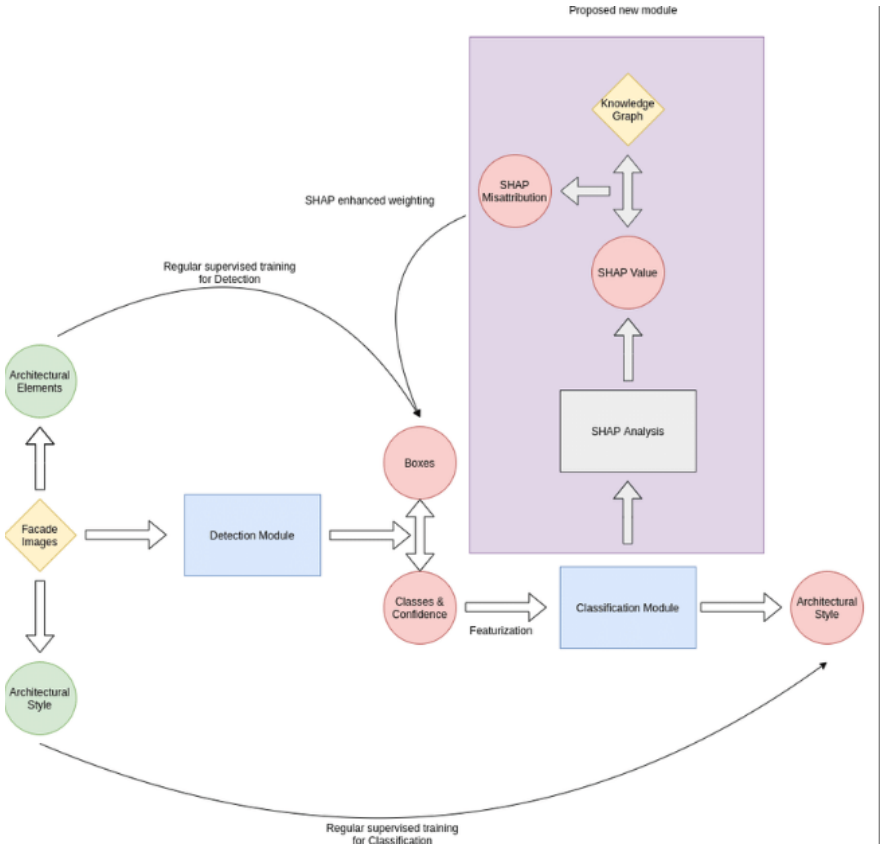

Figure 7: Diagram of proposed X-NeSyL methodology

El enfoque X-NeSyL combinaría técnicas de aprendizaje profundo, como redes neuronales convolucionales (CNN), con lógica simbólica para generar una representación interna del conocimiento que pueda ser interpretada y explicada. La red neuronal convolucional podría extraer características relevantes de las imágenes de aves, mientras que la lógica simbólica podría ayudar a capturar relaciones y reglas lógicas basadas en los atributos específicos, como "las aves con alas rojas y pico amarillo tienden a ser de cierta especie".

Al combinar estos enfoques, el modelo X-NeSyL puede proporcionar explicaciones sobre sus predicciones. Por ejemplo, si el modelo clasifica una imagen de un ave como perteneciente a una especie particular, podría explicar que la clasificación se basa en la presencia de "ala roja" y "pico amarillo" en la imagen, así como en las relaciones lógicas aprendidas durante el entrenamiento.

12.1 Como añadir la metología X-NeSyL

A continuación, explicaremos cómo integrar los principios de X-NeSyL en el código existente:

- **Representación simbólica:** La metodología X-NeSyL busca combinar las capacidades de aprendizaje profundo con una representación simbólica del conocimiento. Para lograr esto, se podría considerar agregar una estructura de datos adicional para representar las reglas lógicas o las relaciones entre los atributos de las aves, como "ala roja" y "pico amarillo". Por ejemplo, se podría tener un diccionario que asocie combinaciones de atributos con clases o etiquetas de aves específicas.
- **Explicación de las predicciones:** Uno de los aspectos clave de X-NeSyL es la capacidad de proporcionar explicaciones para las predicciones del modelo. Se puede agregar código adicional para

generar explicaciones comprensibles basadas en las reglas lógicas aprendidas. Por ejemplo, después de que el modelo realice una predicción, recuperar las reglas correspondientes y explicar cómo se llegó a esa conclusión basada en los atributos de "ala roja" y "pico amarillo".

- **Interpretabilidad:** Otro aspecto importante de X-NeSyL es la interpretabilidad del modelo. Se puede explorar técnicas adicionales para visualizar y comprender mejor las características aprendidas por el modelo. Por ejemplo, visualizar los mapas de activación o los filtros convolucionales en las capas de la red neuronal para ver qué características específicas se activan en respuesta a los atributos de "ala roja" y "pico amarillo".

Es importante destacar que la implementación completa de la metodología X-NeSyL requeriría un enfoque más detallado y personalizado. El código proporcionado es un punto de partida básico y deberás adaptarlo y expandirlo según tus necesidades específicas. La metodología X-NeSyL es un campo de investigación activo y existen varias técnicas y enfoques que puedes explorar para lograr un modelo más explicativo y comprensible.

13 Discusión de resultados

13.1 Data 20

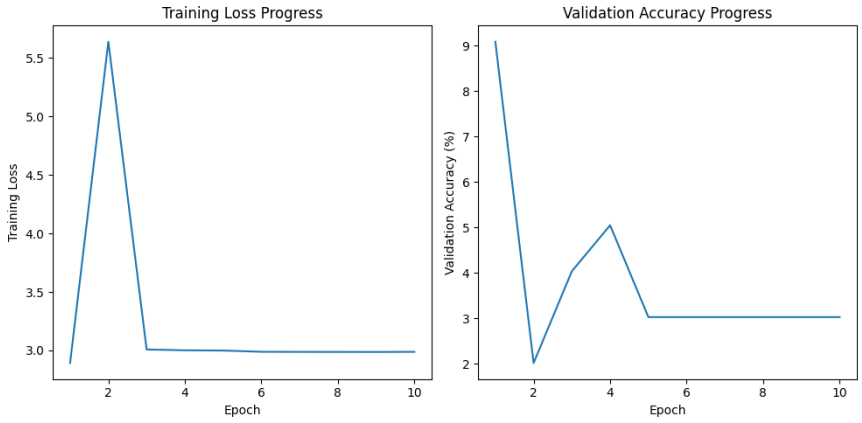

Figure 8: Data 20 training progress

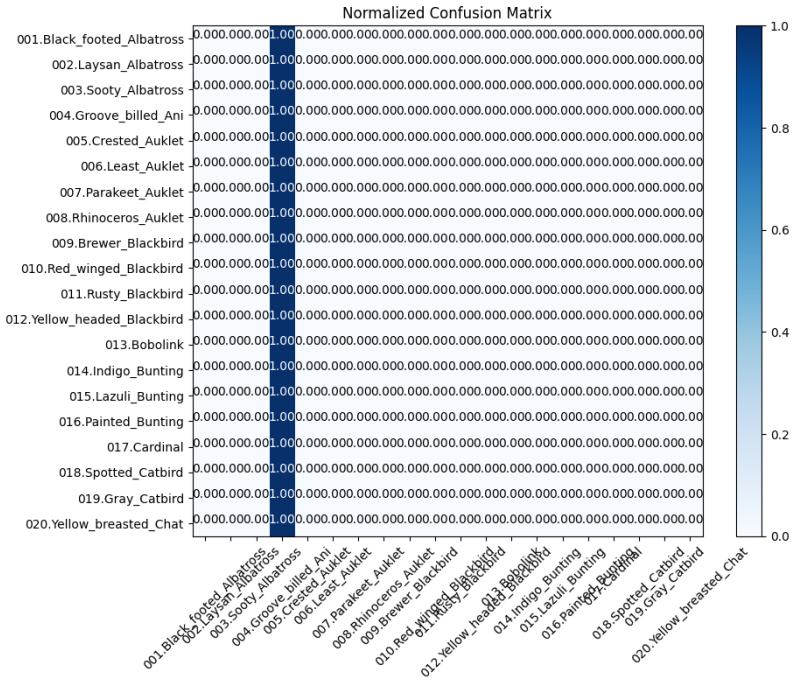

Figure 9: Data 20 confusion matrix

Basándonos en los resultados de train_loss y val_accuracy que hemos obtenido, podemos deducir lo siguiente:

- El valor de train_loss disminuye gradualmente en cada época de entrenamiento. Esto indica que el modelo está aprendiendo a partir de los datos de entrenamiento y está reduciendo el error durante el entrenamiento.
- El valor de val_accuracy varía en cada época, pero parece estabilizarse en un rango bajo alrededor del 3%. Esto sugiere que el modelo no generaliza bien a los datos de validación y tiene dificultades para clasificar correctamente las imágenes en ese conjunto.
- La brecha entre train_loss y val_accuracy indica que el modelo puede estar sobreajustando los datos de entrenamiento, lo que significa que se está ajustando demasiado a los detalles específicos de los datos de entrenamiento y no está generalizando adecuadamente a nuevos datos.

13.2 Data 200

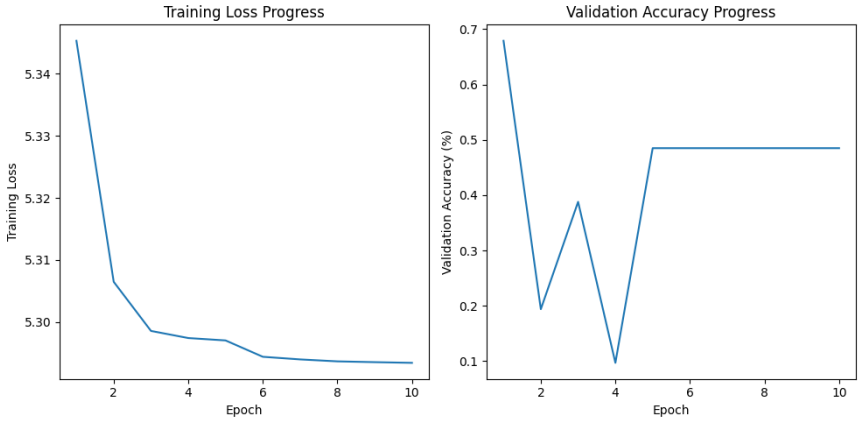

Figure 10: Data 200 training progress

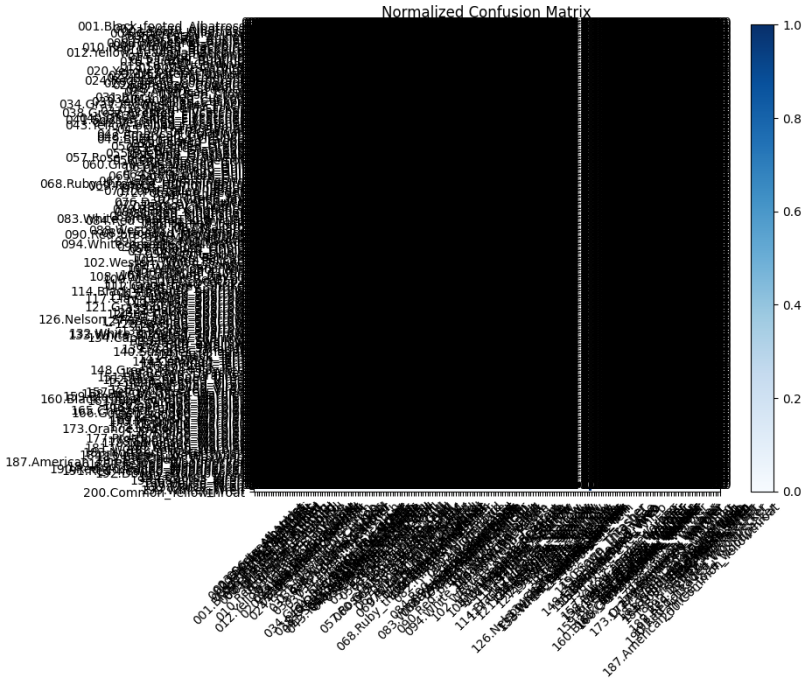

Figure 11: Data 200 confusion matrix

Según los resultados obtenidos, podemos deducir lo siguiente:

- Loss (Pérdida): El valor de la pérdida de entrenamiento disminuye gradualmente a medida que avanzan las épocas. Inicialmente comienza en 5.34 y llega a 5.29 al final de las 10 épocas. Esto indica que el modelo está aprendiendo a ajustar los pesos de manera efectiva para minimizar la pérdida durante el entrenamiento.
- Validation Accuracy (Precisión de validación): La precisión de validación fluctúa en diferentes épocas, comenzando en 0.68 y descendiendo a 0.09 en la cuarta época, antes de aumentar nuevamente a

0.48. Esto sugiere que el modelo no generaliza bien a nuevos datos y muestra dificultades para clasificar correctamente las muestras de validación.

- Ajuste de hiperparámetros: El código realiza algunas técnicas para mejorar el aprendizaje, como el aumento de datos y el ajuste de la tasa de aprendizaje. El aumento de datos se aplica solo al conjunto de entrenamiento, lo que ayuda a aumentar la diversidad de las muestras de entrenamiento y mejorar la capacidad del modelo para generalizar. El ajuste de la tasa de aprendizaje reduce la tasa de aprendizaje en un factor de 0.1 después de cada 5 épocas, lo que puede ayudar a estabilizar y refinar el entrenamiento del modelo.
- Resultados de MLflow: El código utiliza MLflow para realizar un seguimiento de las métricas y parámetros relevantes durante el entrenamiento del modelo. Registra la pérdida de entrenamiento y la precisión de validación en cada época, lo que permite un seguimiento detallado del progreso del entrenamiento.

En general, se puede observar que el modelo aún no ha alcanzado un desempeño óptimo, ya que la precisión de validación es relativamente baja. Podrían ser necesarios ajustes adicionales en la arquitectura del modelo, la técnica de aumento de datos o los hiperparámetros para mejorar el rendimiento del modelo. También es importante tener en cuenta el tamaño y la calidad del conjunto de datos utilizado para entrenar el modelo, ya que esto puede tener un impacto significativo en el desempeño final.

14 Conclusiones

En conclusión, este estudio demostró la efectividad del Deep Learning aplicado a la clasificación utilizando la biblioteca PyTorch. Se utilizaron herramientas adicionales como MLflow para el seguimiento y registro de experimentos, así como la red neuronal preentrenada ResNet50 para extraer características relevantes.

El uso de bibliotecas como NumPy y Matplotlib permitió la manipulación eficiente de los datos y la visualización de resultados. Además, se aplicaron técnicas de aumento de datos para mejorar la generalización del modelo y se ajustó la tasa de aprendizaje para optimizar el proceso de entrenamiento.

Se experimentó con diferentes arquitecturas y estrategias de entrenamiento, evaluando el desempeño mediante la representación gráfica de la matriz de confusión. Esto permitió comprender mejor la capacidad del modelo para clasificar correctamente las muestras y detectar posibles errores.

Por último, se exploró la transferencia de aprendizaje o fine tuning, aprovechando los conocimientos previos de una red preentrenada en un dominio relacionado. Esto permitió obtener resultados prometedores al adaptar la red a la tarea de clasificación específica, reduciendo así la necesidad de entrenamiento desde cero.

En general, este estudio resalta las ventajas y posibilidades del Deep Learning para la clasificación, mostrando cómo las herramientas y técnicas mencionadas pueden ser utilizadas de manera efectiva para desarrollar modelos de clasificación precisos y robustos. Estos hallazgos tienen implicaciones significativas en diversos campos, desde la visión por computadora hasta el procesamiento del lenguaje natural, abriendo el camino para futuras investigaciones y aplicaciones prácticas en el ámbito del aprendizaje automático.

15 Bibliografía

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning*. Nature, 521(7553), 436-444.
- Chollet, F. (2017). *Deep learning with Python*. Manning Publications.
- Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2018). *Understanding deep learning requires rethinking generalization*. Proceedings of the International Conference on Learning Representations (ICLR).
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). *ImageNet large scale visual recognition challenge*. International Journal of Computer Vision, 115(3), 211-252.
- Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). *Going deeper with convolutions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1-9.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. Advances in Neural Information Processing Systems (NIPS), 1097-1105.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). *Mastering the game of Go with deep neural networks and tree search*. Nature, 529(7587), 484-489.