# Descripción General

Este apartado documenta el funcionamiento detallado del script de entrenamiento (train.py) desarrollado para entrenar el modelo StrainNet. El objetivo principal es entrenar una red neuronal para predecir mapas de desplazamiento a partir de pares de imágenes de referencia (Ref) y deformadas (Def). Estos desplazamientos son esenciales en aplicaciones como la reconstrucción de información de fase en imágenes de contraste por moteo de rayos X.

## 1. Propósito General del Código

El script train.py tiene como objetivo entrenar el modelo StrainNet utilizando un conjunto de datos de pares de imágenes (Ref y Def) y sus correspondientes mapas de desplazamiento (Dispx, Dispy). Durante el entrenamiento, el modelo aprende a predecir los desplazamientos entre las imágenes, lo que permite reconstruir las deformaciones sufridas por los patrones de moteo.

## 2. Argumentos de Entrada

El script acepta varios parámetros configurables desde la línea de comandos mediante la biblioteca argparse. Estos incluyen:

* Arquitectura del modelo (--arch): Permite seleccionar entre diferentes variantes de StrainNet (StrainNet\_f, StrainNet\_h).  
  Optimizador (--solver): Selecciona el algoritmo de optimización (adam o sgd).  
  Número de trabajadores (--workers): Define el número de procesos en paralelo para cargar los datos.  
  Número de épocas (--epochs): Especifica el número total de épocas de entrenamiento.  
  Tamaño del batch (--batch-size): Define el tamaño del mini-batch para el entrenamiento.  
  Tasa de aprendizaje (--lr): Especifica la tasa de aprendizaje inicial.  
  Pesos multiescala (--multiscale-weights): Define los pesos de entrenamiento para cada escala de resolución.  
  Modelo preentrenado (--pretrained): Ruta del archivo .pth que contiene los pesos de un modelo previamente entrenado (útil para reanudar entrenamientos).

## 3. Flujo General del Script

### 3.1. Inicialización

Se configuran las rutas de entrada y salida, asegurando que existan directorios donde se guardarán los resultados y los checkpoints del modelo.  
Se cargan los conjuntos de datos de entrenamiento y validación desde archivos CSV que contienen las rutas de las imágenes Ref, Def, Dispx y Dispy.

### 3.2. Configuración del Dispositivo

Se detecta automáticamente si hay una GPU disponible (torch.cuda.is\_available()) y se utiliza para acelerar el entrenamiento. Si no hay GPU disponible, se ejecuta en la CPU.

### 3.3. Carga del Modelo

Se inicializa el modelo seleccionado (StrainNet\_f o StrainNet\_h) y se carga en el dispositivo correspondiente (GPU o CPU).Si se proporciona un modelo preentrenado (--pretrained), se cargan los pesos desde el archivo .pth.

### 3.4. Configuración del Optimizador y Scheduler

Se configura el optimizador (Adam o SGD) con los parámetros proporcionados (tasa de aprendizaje, momento, etc.).  
Se configura un scheduler (MultiStepLR) para reducir la tasa de aprendizaje en épocas específicas (milestones).

### 3.5. Entrenamiento

El modelo se entrena durante un número específico de épocas (--epochs).  
En cada época, se procesan los datos en mini-batches, se calcula la pérdida utilizando la función multiscaleEPE y se actualizan los pesos del modelo mediante retropropagación.  
Se registran métricas como la pérdida y el error EPE (End Point Error) en TensorBoard.

### 3.6. Validación

Después de cada época, se evalúa el modelo en el conjunto de validación utilizando la función validate.  
Se calcula el error EPE en el conjunto de validación y se registra en TensorBoard.

### 3.7. Guardado de Checkpoints

Se guarda un checkpoint del modelo después de cada época si se obtiene un mejor error EPE en el conjunto de validación.  
El checkpoint incluye el estado del modelo, el optimizador y las métricas relevantes.

### 3.8. Early Stopping

Si el error EPE no mejora durante un número específico de épocas (early\_stopping\_patience), se detiene el entrenamiento para evitar sobreajuste.

## 4. Manejo de Errores y Advertencias

El script incluye múltiples niveles de manejo de errores para garantizar una ejecución robusta:

* Verificación de GPU: Si no hay GPU disponible, se muestra una advertencia y se utiliza la CPU.  
  Carga de datos: Si no se encuentran archivos Ref o Def correspondientes, se muestra un error.  
  Guardado de checkpoints: Si no se puede guardar un checkpoint, se muestra un error.

## 5. Resultados y Salida

Durante el entrenamiento, el script genera los siguientes resultados:

* Checkpoints del modelo: Archivos .pth que contienen los pesos del modelo en diferentes épocas.  
  Métricas en TensorBoard: Gráficas de la pérdida y el error EPE durante el entrenamiento y la validación.  
  Mejor modelo: El modelo con el menor error EPE en el conjunto de validación se guarda como el mejor modelo.

## 6. Aplicaciones Prácticas

El script de entrenamiento es fundamental para aplicaciones donde se necesitan mapas de desplazamiento precisos, como:

* Reconstrucción de fase en imágenes de rayos X.  
  Monitorización de deformaciones en materiales.  
  Análisis de estructuras internas en tejidos blandos.

## Explicación de las Funciones de Pérdida Multiescala

El script utiliza funciones de pérdida multiescala (multiscaleEPE y realEPE) para evaluar el rendimiento del modelo durante el entrenamiento. Estas funciones son esenciales para garantizar que el modelo aprenda a predecir desplazamientos en diferentes escalas de resolución.

## Conclusión

El script train.py es una herramienta esencial para entrenar el modelo StrainNet en tareas de predicción de desplazamientos. Su estructura modular y su capacidad para manejar múltiples escalas de resolución lo hacen adecuado para aplicaciones en imágenes de moteo y análisis de deformaciones.