# Project - First Report. Deep Learning Model

## Super resolution

**Estudiante 1:** Helman Andrés Merchán Quevedo

**Código:** 2182693

**Estudiante 2:** Cesar Enrique Rojas Hernandez

**Código:** 2191952

**Grupo:** C2

*Escuela de ingeniería de sistemas e informática*

*Universidad Industrial de Santander*

7 de mayo de 2024

## 1 Introducción

EL modelo de aprendizaje profundo es un método de inteligencia artificial para procesar datos, para este proyecto se utiliza este modelo con la finalidad de procesar unas imágenes que tiene baja calidad y mejorar su resolución, esto se hace mediante el entrenamiento de una red la cual utiliza diferentes parámetros para hacer que se lo mas eficaz posible

## 2 Desarrollo

### 2.1 Entendiendo el código

Explicar los pasos secuenciales seguidos por el código proporcionado.
Cargar modelo y definir hiperparámetros, optimizador y función de pérdida
El código proporcionado realiza los siguientes pasos:

1. Importa el optimizador SGD, Adam, RMSprop, AdamW y Adagrad desde el módulo `torch.optim`.

2. Define un factor de decimación espacial de 4, que no se puede modificar.

3. Define hiper parámetros para el modelo:
   - `EPOCHS`: número de épocas de entrenamiento (50).
   - `BATCH_SIZE`: tamaño del lote de datos para el entrenamiento (4).
   - `base_filter`: número de filtros base para las capas convolucionales (8).
   - `num_residuals`: número de bloques residuales en el modelo (3).

4. Instancia un modelo DeepNeuralNetwork con los hiperparámetros `base_filter`, `num_residuals` y `scale_factor` (factor de decimación espacial).

5. Define el nombre del optimizador (`SGD`, `Adam` o `RMSprop`) que se utilizará.

6. Define la función de costo (`MSE` o `MAE`) que se utilizará.

7. Define la tasa de aprendizaje (learning rate) del optimizador (1e-3).

8. Instancia el optimizador correspondiente al nombre seleccionado, pasando los parámetros del modelo y la tasa de aprendizaje. También se configuran otros hiperparámetros como el momentum y el weight_decay.

9. Instancia la función de costo correspondiente a la selección (`MSELoss` o `L1Loss`).

Se podría decir que este código configura los hiperparámetros, define el modelo, el optimizador y la función de costo que se utilizarán para el entrenamiento de una red neuronal profunda (Deep Neural Network).

**Load Dataset and visualize samples**

En esta parte del código se realiza lo siguiente:

1. Define la ruta del conjunto de datos (`data_path`) como "/content/BSDS500".

2. Carga los conjuntos de datos de entrenamiento y prueba utilizando la función `load_data`. Esta función toma como argumentos la ruta del conjunto de datos (`data_path`), el factor de decimación espacial (`spatial_decimation_factor`) y el tamaño del lote (`batch_size`). Los conjuntos de datos cargados se almacenan en las variables `trainset` y `testset`.

3. Define el número de imágenes (`num_imgs`) que se visualizarán, establecido en 4.

4. Crea una figura con un tamaño de 15x7 pulgadas utilizando `plt.figure`.

5. Obtiene una muestra del conjunto de datos de entrenamiento utilizando `next(iter(trainset))`. Esta muestra contiene un lote de imágenes de alta resolución (`x`) y sus respectivas versiones de baja resolución (`y`).

6. Itera sobre las primeras `num_imgs` imágenes de la muestra y realiza lo siguiente para cada una:
   - Separa las imágenes de alta resolución (`x[i]`) y baja resolución (`y[i]`) correspondientes.
   - Crea dos subplots, uno para la imagen de alta resolución y otro para la de baja resolución.
   - Muestra la imagen de alta resolución en el primer subplot utilizando `plt.imshow`, permutando las dimensiones de la imagen con `permute(1, 2, 0)` y convirtiéndola a un arreglo NumPy con `.numpy()`.
   - Muestra la imagen de baja resolución en el segundo subplot de manera similar.
   - Establece los títulos de las filas de subplots como "High Resolution Images" y "Low Resolution Images", respectivamente.

7. Muestra la figura con todas las imágenes utilizando `plt.show()`.

En esta parte se carga los conjuntos de datos de entrenamiento y prueba, obtiene una muestra del conjunto de entrenamiento y visualiza las primeras `num_imgs` imágenes de alta y baja resolución correspondientes en una figura con subplots. Esto permite inspeccionar visualmente las muestras del conjunto de datos.

**Training process**

En la parte del entrenamiento el modelo de súper resolución de imágenes utiliza PyTorch. A continuación, se explican los pasos secuenciales:

1. Inicialización de métricas: Se crean instancias de las métricas `PeakSignalNoiseRatio` (PSNR) y `StructuralSimilarityIndexMeasure` (SSIM) para medir la calidad de las imágenes reconstruidas. Estas métricas se envían al dispositivo apropiado (`device`).

2. Transferencia del modelo al dispositivo: El modelo se transfiere al dispositivo apropiado (`device`) utilizando `model.to(device)`.

3. Bucle de entrenamiento: Se inicia un bucle que itera sobre el número de épocas (`EPOCHS`) definido previamente.

4. Inicialización de medidores: Se inicializan tres instancias de `AverageMeter` para monitorear la pérdida promedio (`losses`), el PSNR promedio (`psnr`) y el SSIM promedio (`ssim`) durante el entrenamiento. Además, se inicializan tres medidores adicionales (`val_losses`, `val_psnr` y `val_ssim`) para el conjunto de validación.

5. Modo de entrenamiento: El modelo se establece en modo de entrenamiento (`model.train()`).

6. Iteración sobre el conjunto de entrenamiento: Se itera sobre el conjunto de datos de entrenamiento (`trainset`), obteniendo un lote de imágenes de alta resolución (`x`) y baja resolución (`y`).

7. Transferencia de datos al dispositivo: Las imágenes `x` e `y` se transfieren al dispositivo apropiado (`device`).

8. Reinicio del gradiente: Se reinicia el gradiente del optimizador (`optimizer.zero_grad()`).

9. Predicción del modelo: Se obtiene la predicción del modelo (`x_hat`) para las imágenes de baja resolución (`y`).

10. Cálculo de la pérdida: Se calcula la pérdida (`loss`) comparando la predicción `x_hat` con las imágenes de alta resolución `x` utilizando la función de costo definida previamente (`criterion`).

11. Retropropagación del error: Se realiza la retropropagación del error (`loss.backward()`).

12. Actualización de los parámetros del modelo: Se actualizan los parámetros del modelo utilizando el optimizador (`optimizer.step()`).

13. Cálculo de métricas: Se calculan los valores de PSNR (`psnr_val`) y SSIM (`ssim_val`) comparando las predicciones `x_hat` con las imágenes de alta resolución `x`.

14. Actualización de medidores: Se actualizan los medidores de pérdida, PSNR y SSIM para el conjunto de entrenamiento (`losses.update()`, `psnr.update()` y `ssim.update()`).

15. Modo de evaluación: El modelo se establece en modo de evaluación (`model.eval()`).

16. Iteración sobre el conjunto de validación: Se itera sobre el conjunto de datos de validación (`testset`) de manera similar al conjunto de entrenamiento.

17. Actualización de medidores de validación: Se actualizan los medidores de pérdida, PSNR y SSIM para el conjunto de validación (`val_losses.update()`, `val_psnr.update()` y `val_ssim.update()`).

18. Impresión de resultados:: Se imprimen los resultados de la época actual, incluyendo la pérdida, el PSNR y el SSIM para los conjuntos de entrenamiento y validación.

Este proceso se repite para cada época de entrenamiento, permitiendo al modelo aprender a reconstruir imágenes de alta resolución a partir de imágenes de baja resolución, mientras se monitorean las métricas de pérdida, PSNR y SSIM.

**Inference process**

Para finalizar implementa el proceso de inferencia, es decir, la predicción de imágenes de alta resolución a partir de imágenes de baja resolución utilizando el modelo entrenado. A continuación, se explican los pasos secuenciales:

1. Definición del número de imágenes: Se establece el número de imágenes (`num_images`) que se mostrarán, en este caso, 4.

2. Iterador del conjunto de prueba: Se crea un iterador (`iter_testset`) sobre el conjunto de datos de prueba (`testset`).

3. Creación de la figura: Se crea una figura (`fig`) con tres filas y `num_images + 1` columnas, con un tamaño de 16x10 pulgadas.

4. Bucle para mostrar imágenes: Se inicia un bucle que itera sobre `num_images`.

5. Obtención de muestras del conjunto de prueba: En cada iteración, se obtiene una muestra (`x`, `y`) del conjunto de prueba utilizando `next(iter_testset)`.

6. Transferencia de datos al dispositivo: La imagen de baja resolución (`y`) se transfiere al dispositivo apropiado (`device`).

7. Predicción del modelo: Se obtiene la predicción del modelo (`x_hat`) para la imagen de baja resolución (`y`), y se transfiere de vuelta a la CPU (`detach().cpu()`).

8. Transferencia de imágenes a la CPU: Las imágenes `y` y `x` (alta resolución) también se transfieren de vuelta a la CPU.

9. Visualización de imágenes: Se muestran las imágenes de baja resolución (`y`), las imágenes reconstruidas (`x_hat`) y las imágenes de alta resolución (`x`) en las filas correspondientes de la figura.

10. Configuración de títulos: Se establecen los títulos de las filas de la figura: "Low Resolution Images", "Restored Images" y "High Resolution Images".

11. Obtención de una muestra aleatoria del conjunto de prueba: Se genera un índice aleatorio (`iter_n`) dentro del rango del conjunto de prueba, excluyendo las primeras `num_images` muestras.

12. Iteración sobre la muestra aleatoria: Se itera sobre la muestra aleatoria del conjunto de prueba.

13. Visualización de la muestra aleatoria: Se repiten los pasos 6-8 para la muestra aleatoria y se muestran las imágenes en la última columna de la figura.

14. Visualización de la figura: Se muestra la figura completa utilizando `plt.show()`.

Se puede llegar a la conclusión de que este código crea una figura con tres filas y `num_images + 1` columnas. Las primeras `num_images` columnas muestran las imágenes de baja resolución, las imágenes reconstruidas por el modelo y las imágenes de alta resolución correspondientes. La última columna muestra una muestra aleatoria del conjunto de prueba. Esto permite visualizar y comparar las imágenes de baja resolución, las predicciones del modelo y las imágenes de alta resolución de referencia.

• Explain what a dataset is. Describe the dataset used in the code.

Un conjunto de datos es una colección de datos organizados y estructurados que se utiliza para entrenar, evaluar y probar modelos de aprendizaje automático o realizar análisis estadísticos. Los conjuntos de datos suelen estar compuestos por instancias individuales, también llamadas muestras o ejemplos, que contienen características o atributos relevantes para el problema que se está abordando.

En el código proporcionado, el conjunto de datos utilizado es el BSDS500 (Berkeley Segmentation Dataset and Benchmarks 500), que es un conjunto de datos ampliamente utilizado en tareas relacionadas con el procesamiento de imágenes, como la súper resolución.

El conjunto de datos BSDS500 contiene 500 imágenes de alta resolución, junto con anotaciones de segmentación de objetos realizadas por expertos humanos. Estas imágenes de alta resolución se utilizan para generar versiones de baja resolución, simulando el efecto de reducir la resolución.

El objetivo de utilizar este conjunto de datos en el contexto del código es entrenar un modelo de aprendizaje profundo para aprender a reconstruir imágenes de alta resolución a partir de sus versiones de baja resolución. El conjunto de datos se divide en dos partes:

1. Conjunto de entrenamiento (trainset): Este conjunto se utiliza para entrenar el modelo, mostrándole pares de imágenes de baja y alta resolución, con el fin de que el modelo aprenda a mapear las imágenes de baja resolución a sus correspondientes versiones de alta resolución.

2. Conjunto de prueba (testset): Este conjunto se utiliza para evaluar el rendimiento del modelo entrenado en datos que no ha visto durante el entrenamiento. Permite medir la capacidad de generalización del modelo a nuevos datos.

El código carga estos conjuntos de datos utilizando la función `load_data`, que toma como entrada la ruta del conjunto de datos (`data_path`), el factor de decimación espacial (`spatial_decimation_factor`) y el tamaño del lote (`batch_size`). Luego, se visualizan algunas muestras de las imágenes de alta y baja resolución del conjunto de entrenamiento para inspección visual.

Ell conjunto de datos BSDS500 es un conjunto de referencia ampliamente utilizado en tareas de procesamiento de imágenes, y en este caso específico, se utiliza para entrenar un modelo de súper resolución de imágenes.

• What is dataset preprocessing? What kind of preprocessing can you apply to the selected problem?

El preprocesamiento de conjuntos de datos se refiere a las técnicas y transformaciones que se aplican a los datos antes de utilizarlos para entrenar un modelo de aprendizaje automático o realizar análisis. El objetivo principal del preprocesamiento es preparar y transformar los datos en un formato adecuado para el modelo, mejorar la calidad de los datos y, en última instancia, mejorar el rendimiento y la precisión del modelo.

En el caso específico del problema de súper resolución de imágenes, donde se busca reconstruir imágenes de alta resolución a partir de imágenes de baja resolución, se pueden aplicar varios tipos de preprocesamiento a los conjuntos de datos. A continuación, se mencionan algunas técnicas de preprocesamiento comunes:

1. Normalización de datos: Las imágenes suelen tener valores de píxeles en el rango de 0 a 255. Es común normalizar estos valores a un rango más pequeño, como 0 a 1, para facilitar el entrenamiento del modelo.

2. Aumento de datos (Data Augmentation): Esta técnica consiste en aplicar transformaciones aleatorias a las imágenes, como rotaciones, volteos horizontales o verticales, recortes, etc. Esto ayuda a aumentar la diversidad de los datos de entrenamiento y mejorar la capacidad de generalización del modelo.

3. Eliminación de ruido: Algunas imágenes pueden contener ruido o artefactos no deseados. Se pueden aplicar técnicas de filtrado o procesamiento de imágenes para eliminar este ruido antes del entrenamiento.

4. Recorte y redimensionamiento: Es común recortar las imágenes a un tamaño específico y redimensionarlas a una resolución fija para que todas las imágenes tengan el mismo tamaño y facilitar el procesamiento por el modelo.

5. Separación en parches: En lugar de procesar imágenes completas, a veces es más eficiente dividir las imágenes en parches o bloques más pequeños y procesarlos individualmente. Esto puede mejorar el rendimiento y reducir los requisitos de memoria.

6. Generación de imágenes de baja resolución: Como el objetivo es reconstruir imágenes de alta resolución a partir de imágenes de baja resolución, es necesario generar las imágenes de baja resolución a partir de las imágenes de alta resolución originales. Esto se puede hacer aplicando técnicas de submuestreo o filtrado pasa-bajos.

7. División en conjuntos de entrenamiento, validación y prueba: Es importante dividir el conjunto de datos en subconjuntos de entrenamiento, validación y prueba para evaluar adecuadamente el rendimiento del modelo y evitar el sobreajuste.

Estas son solo algunas de las técnicas de preprocesamiento que se pueden aplicar al problema de súper resolución de imágenes.

• What is a deep neural network? Make a detailed description of the architecture used in the code.

Una red neuronal profunda (Deep Neural Network, DNN) es un tipo de red neuronal artificial que tiene una arquitectura con múltiples capas ocultas entre la capa de entrada y la capa de salida. Estas redes tienen una profundidad mayor que las redes neuronales tradicionales, lo que les permite aprender representaciones más complejas y jerárquicas de los datos.

En el código proporcionado, se utiliza una arquitectura de red neuronal profunda específica para el problema de súper resolución de imágenes. Aunque no se muestra la implementación completa de la arquitectura, podemos deducir algunos detalles de su estructura basándonos en los hiperparámetros mencionados y la nomenclatura utilizada.

La arquitectura se denomina `DeepNeuralNetwork` y se instancia con los siguientes hiperparámetros:

- `base_filter`: Este hiperparámetro determina el número de filtros en las primeras capas convolucionales de la red. Un valor típico podría ser 8 o 16.
- `num_residuals`: Este hiperparámetro indica el número de bloques residuales en la arquitectura. Los bloques residuales son un concepto introducido por las redes residuales (ResNet), que permiten un flujo de información más efectivo a través de la red, lo que facilita el entrenamiento de redes muy profundas.
- `scale_factor`: Este hiperparámetro se refiere al factor de escala o aumento de resolución deseado. En el código, se establece en `spatial_decimation_factor`, que tiene un valor de 4.

Basándonos en esta información y en arquitecturas típicas para tareas de súper resolución, podemos suponer que la red neuronal profunda utilizada sigue una estructura similar a la siguiente:

1. Capa de entrada: Recibe la imagen de baja resolución como entrada.

2. Bloque de convoluciones y submuestreo: Compuesto por capas convolucionales y capas de submuestreo (como max-pooling) que extraen características de la imagen y reducen su resolución espacial.

3. Bloques residuales: Una secuencia de `num_residuals` bloques residuales, cada uno compuesto por capas convolucionales, operaciones de normalización por lotes (batch normalization) y conexiones residuales. Estos bloques extraen características de alto nivel de la imagen.

4. Bloque de deconvoluciones y super-resolución: Compuesto por capas de deconvolución (también conocidas como convoluciones transpuestas) y capas convolucionales que aumentan progresivamente la resolución espacial de las características hasta alcanzar el factor de escala deseado (`scale_factor`).

5. Capa de salida: Produce la imagen de alta resolución reconstruida como salida.

Además, es común que la arquitectura utilice técnicas como el aprendizaje residual (donde el modelo aprende a predecir la diferencia entre la imagen de entrada y la salida deseada) y funciones de activación como ReLU o PReLU en las capas intermedias.

Esta arquitectura de red neuronal profunda, con bloques residuales y capas de deconvolución, es capaz de aprender patrones complejos y mapear eficazmente las imágenes de baja resolución a sus versiones de alta resolución, aprovechando la capacidad de las redes neuronales profundas para aprender representaciones jerárquicas de los datos.

• What is the cost or loss function? Describe the loss function employed in the code.

En el código proporcionado, se define una variable llamada `cost_function` o `criterion` que determina la función de costo o pérdida que se utilizará durante el entrenamiento del modelo de red neuronal profunda.

Específicamente, se definen dos opciones posibles para la función de costo:

1. MSE (Mean Squared Error: Si `cost_function = 'MSE'`, entonces se utiliza la función de pérdida `nn.MSELoss()` de PyTorch.

2. MAE (Mean Absolute Error): Si `cost_function = 'MAE'`, entonces se utiliza la función de pérdida `nn.L1Loss()` de PyTorch.

La función de pérdida MSE (Mean Squared Error) se define como:

```
MSE = (1/n) * Σ(y_pred - y_true)^2
```

Donde `n` es el número total de muestras, `y_pred` es la salida predicha por el modelo y `y_true` es la salida real o esperada.

La función de pérdida MAE (Mean Absolute Error) se define como:

```
```

MAE = (1/n) * Σ|y_pred - y_true|
```

Donde `n` es el número total de muestras, `y_pred` es la salida predicha por el modelo y `y_true` es la salida real o esperada.

Ambas funciones de pérdida miden la diferencia entre las predicciones del modelo y las salidas reales, pero difieren en cómo se calcula esta diferencia. La MSE penaliza más fuertemente los errores grandes al elevar las diferencias al cuadrado, mientras que la MAE trata todos los errores de manera lineal.

Durante el entrenamiento, el objetivo es minimizar la función de pérdida seleccionada ajustando los pesos y sesgos del modelo de red neuronal profunda. El optimizador seleccionado (SGD, Adam o RMSprop) actualiza los parámetros del modelo en la dirección que reduce el valor de la función de pérdida.

La elección entre MSE y MAE como función de pérdida puede depender del problema específico y de las características deseadas. En general, la MSE tiende a ser más sensible a valores atípicos (outliers), mientras que la MAE es más robusta a ellos. En el caso de la súper resolución de imágenes, ambas funciones de pérdida son comúnmente utilizadas.

• What are the metrics? Describe the metrics employed in the code.

En el código proporcionado, se utilizan dos métricas principales para evaluar el rendimiento del modelo de súper resolución de imágenes: PSNR (Peak Signal-to-Noise Ratio) y SSIM (Structural Similarity Index Measure).

1. PSNR (Peak Signal-to-Noise Ratio):
   - El PSNR es una métrica ampliamente utilizada para medir la calidad de la reconstrucción de imágenes en tareas de procesamiento de imagen como la súper resolución.
   - Representa la relación entre la máxima señal posible y el ruido que afecta la representación de la señal. Un valor más alto de PSNR indica una mejor calidad de la imagen reconstruida.
   - En el código, se define una instancia `psnr_fn` de la clase `PeakSignalNoiseRatio` de PyTorch, con un rango de datos de 1.0 y una reducción de tipo `''elementwise_mean''`.
   - Durante el entrenamiento y la evaluación, se calcula el PSNR comparando las imágenes reconstruidas (`x_hat`) con las imágenes de alta resolución de referencia (`x`).

2. SSIM (Structural Similarity Index Measure):
   - El SSIM es otra métrica utilizada para evaluar la similitud estructural entre dos imágenes.
   - A diferencia del PSNR, que se basa únicamente en errores de intensidad de píxeles, el SSIM considera características adicionales como la luminancia, el contraste y la estructura de las imágenes.
   - Un valor de SSIM más cercano a 1 indica una mayor similitud estructural entre las imágenes comparadas.
   - En el código, se define una instancia `ssim_fn` de la clase `StructuralSimilarityIndexMeasure` de PyTorch, con un rango de datos de 1.0 y una reducción de tipo `''elementwise_mean''`.
   - Durante el entrenamiento y la evaluación, se calcula el SSIM comparando las imágenes reconstruidas (`x_hat`) con las imágenes de alta resolución de referencia (`x`).
```

Tanto el PSNR como el SSIM se calculan durante el entrenamiento y la evaluación del modelo. Los valores de estas métricas se monitorean y se imprimen en cada época de entrenamiento, lo que permite realizar un seguimiento del progreso del modelo.

Utilizar múltiples métricas brinda una evaluación más completa del rendimiento del modelo, ya que cada métrica captura diferentes aspectos de la calidad de la imagen reconstruida. Mientras que el PSNR se enfoca en la diferencia de intensidad de píxeles, el SSIM considera características estructurales más complejas.

Estas métricas son comúnmente utilizadas en tareas de súper resolución de imágenes, ya que brindan una evaluación cuantitativa de la calidad de las imágenes reconstruidas en comparación con las imágenes de alta resolución de referencia.

• Describe the training process (gradient descent algorithm) and how it works.

El proceso de entrenamiento utilizado en el código se basa en el algoritmo de descenso de gradiente, que es un enfoque ampliamente utilizado para optimizar los parámetros (pesos y sesgos) de un modelo de red neuronal profunda.

El algoritmo de descenso de gradiente funciona de la siguiente manera:

1. Inicialización de parámetros: Al comienzo del entrenamiento, los parámetros del modelo (pesos y sesgos) se inicializan con valores aleatorios pequeños.

2. Propagación hacia adelante (forward pass): En cada iteración de entrenamiento, se pasa un lote (batch) de datos de entrada a través de la red neuronal. El modelo realiza cálculos de propagación hacia adelante para obtener las predicciones o salidas correspondientes.

3. Cálculo de la función de pérdida: Se compara la salida predicha por el modelo con la salida real o esperada utilizando una función de pérdida o costo (en este caso, MSE o MAE). Esta función de pérdida cuantifica la diferencia entre las predicciones y las salidas reales.

4. Retropropagación (backward pass): Utilizando el cálculo de derivadas parciales, se calcula el gradiente de la función de pérdida con respecto a cada uno de los parámetros del modelo. Este gradiente indica la dirección en la que se deben ajustar los parámetros para reducir la pérdida.

5. Actualización de parámetros: Los parámetros del modelo se actualizan en la dirección opuesta al gradiente, utilizando un optimizador como SGD (Descenso de Gradiente Estocástico), Adam o RMSprop. El optimizador determina la magnitud del ajuste aplicado a los parámetros, controlado por una tasa de aprendizaje (learning rate).

6. Iteración del proceso: Los pasos 2 a 5 se repiten para cada lote de datos del conjunto de entrenamiento durante una época. Después de completar una época, el proceso se repite para todas las épocas restantes especificadas.

Durante el entrenamiento, el objetivo es minimizar la función de pérdida ajustando los parámetros del modelo en la dirección opuesta al gradiente. A medida que se procesan más datos de entrenamiento, el modelo aprende patrones y características relevantes para mapear las imágenes de baja resolución a sus versiones de alta resolución.

Además, el código monitorea métricas adicionales como PSNR y SSIM durante el entrenamiento y la evaluación. Estas métricas brindan una evaluación cuantitativa de la calidad de las imágenes reconstruidas por el modelo.

El proceso de entrenamiento continúa hasta que se alcanza un número predefinido de épocas o hasta que se cumple algún otro criterio de convergencia o parada. Al final del entrenamiento, se espera que el modelo haya aprendido a reconstruir imágenes de alta resolución de manera efectiva a partir de imágenes de baja resolución.

• Describe the inference process (how to restore unobserved corrupt images) and how it works.

El proceso de inferencia en el contexto de la súper resolución de imágenes se refiere a cómo el modelo entrenado restaura o reconstruye imágenes de alta resolución a partir de imágenes de baja resolución no observadas durante el entrenamiento. Este proceso se lleva a cabo después de que el modelo ha sido entrenado utilizando el algoritmo de descenso de gradiente y los conjuntos de datos de entrenamiento y validación.

El código proporcionado implementa el proceso de inferencia de la siguiente manera:

1. Preparación del modelo: Primero, se establece el modelo en modo de evaluación (`model.eval()`). Esto es importante porque desactiva operaciones como la normalización por lotes y la aplicación de dropout, que se utilizan durante el entrenamiento pero no son necesarias durante la inferencia.

2. Creación de una figura: Se crea una figura con subplots utilizando `plt.subplots()`. Esta figura tendrá tres filas y `num_images + 1` columnas, donde `num_images` es el número de imágenes de muestra que se mostrarán.

3. Iteración sobre muestras del conjunto de prueba: Se itera sobre `num_images` muestras del conjunto de prueba (`testset`) utilizando un iterador (`iter_testset`).

4. Obtención de muestras: Para cada iteración, se obtiene una muestra (`x`, `y`) del conjunto de prueba, donde `x` es la imagen de alta resolución y `y` es la imagen de baja resolución correspondiente.

5. Transferencia de datos al dispositivo: La imagen de baja resolución (`y`) se transfiere al dispositivo adecuado (CPU o GPU) utilizando `y.to(device)`.

6. Inferencia del modelo: Se realiza la inferencia del modelo pasando la imagen de baja resolución (`y`) a través del modelo utilizando `model(y)`. Esto produce la predicción del modelo, que es la imagen reconstruida de alta resolución (`x_hat`).

7. Transferencia de datos a la CPU: Las imágenes reconstruidas (`x_hat`) se transfieren de vuelta a la CPU utilizando `detach().cpu()`.

8. Visualización de imágenes: Se muestran las imágenes de baja resolución (`y`), las imágenes reconstruidas (`x_hat`) y las imágenes de alta resolución originales (`x`) en las filas correspondientes de la figura.

9. Muestra aleatoria adicional: Se genera un índice aleatorio (`iter_n`) dentro del rango del conjunto de prueba, excluyendo las primeras `num_images` muestras. Se itera sobre esta muestra aleatoria y se visualiza en la última columna de la figura.

10. Visualización de la figura: Finalmente, se muestra la figura completa utilizando `plt.show()`.

Este proceso de inferencia permite evaluar visualmente cómo el modelo entrenado restaura imágenes de alta resolución a partir de imágenes de baja resolución no observadas durante el entrenamiento. Comparando las imágenes reconstruidas (`x_hat`) con las imágenes de alta resolución originales (`x`), se puede evaluar la calidad de la reconstrucción y el rendimiento del modelo en datos nuevos.

**2.2 Improving baseline results**

Cambios realizados

2.2.1 En los cambios realizados se encuentra la implementación de super resolución se encuentran los siguientes:

- Primero, se agrega una capa de activación no lineal Softmax2d. Esta capa se aplica a los ejes bidimensionales generados por las operaciones de convolución en la red neuronal. La función Softmax2d permite normalizar los valores en la dimensión del canal de las imágenes, convirtiéndolos en distribuciones de probabilidad. Esta normalización fue fundamental para interpretar y comparar características extraídas de imágenes de redes neuronales.

  A continuación, se introdujo una capa de activación sigmoidea al final de la red neuronal. La función sigmoidea tiene la capacidad de trazar valores de entrada en el rango entre 0 y 1, convirtiéndolos en valores de probabilidad. Esta capa sigmoidea está diseñada para proporcionar una mejora en los datos procesados por la red.
- Luego se agregaron 2 optimizadores para hacer diferentes pruebas con esto, los resultados vaian según la distribución de la red y otros factores, pero los que se agregaron fueron el RMSprop y Adagrad que se encargan de adaptar la tasa de aprendizaje de cada parámetro.
- Por ultimo se agrogo una función de perdida y aunque no fue utilizada, se trataba de la función Smooth.

2.2.2 Ajustes realizados a la red neuronal profunda.

- Como ya se nombro se agrego a la red neuronal 2 capas, una softmax2d y una sigmoid que ayudo a mejorar la estructura de la imagen un poco en comparación a la red original, esto aunque no permite que la imagen sea bastante similares a la original si permite la mejora de la pixelada y se ve relativamente mejor con los coles, se hace antes del bloque residual y en la salida final, estos cambios se implementaron basados en la documentación de pytorch.

2.2.3 Modificaciones adicionales.

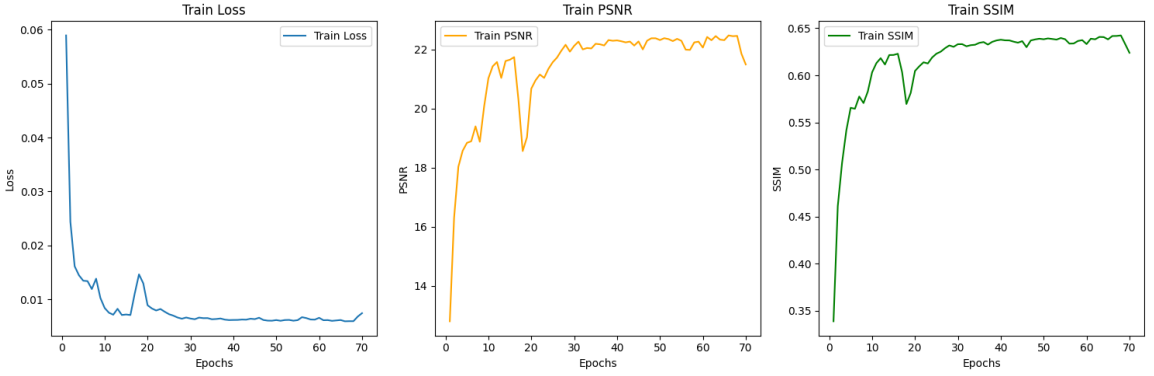
- En las modificaciones adicioneales que se realizaron principalmente se encuentra los hiperparametros ya que aunque se agregaron funciones de perdida y otros optimizadores no se usaron los hiperparametros lo que se busco fue encontrar un numero promedio de épocas en las cuales el modelo fuera entrenado lo suficiente sin ver que aunque se realizaran mas épocas no mejoraran por eso 60 epocas resulto un numero suficiente para que esto se cumpliera, el tamaño del lote se hizo de 5 para tratar de que se generalizara y así poder obtener mejores resultados luego. Para el base filters se usaron 32 para revisar que la red pudiera mejorar lo suficiente desde el inicio y obtener buenos resultados, y el numero de residuales también es 25, eso para mejorar el aprendizaje, y por ultimo el leaming rate que se dejo en 1e^-3 que resulta ser un numero bastante estándar a utilizar y resulta bastante bueno.

**2.3 Analysis of the obtained results**

2.3.1 Informe de cuantitativos y cualitativos

- Inicialmente el MSE que nos es una medida de la diferencia entre dos imágenes. Calcula el promedio de las diferencias cuadráticas entre los valores de píxeles de una imagen de referencia y los valores de píxeles de una imagen procesada. Entre más bajo este valor, indica que la imagen se parece mas a la original, esta esta por debajo de 0.01 lo que inica buenos resultados, mientras tanto el PSNR y el SSIM no tenemos tan buenos resultados con valores entre 21y y 23 cuando la red ya esta entrenada, estos entre mas altos mas similitud indican por lo que indica que aun después de restaurada la imagen faltas varios detalles, lo que se logra notar en la imagen restaurada que aunque suele conservar los mismos colores y estructura en comparación con la imagen pixelada lo que hace es suavizarla y los pixeles no se marcan tanto como lo hacen al inicio, pero en comparación a la imagen original le hace falta bastantes detalles.

2.3.1 Grafico de convergencia




2.3.1 Conclusiones

- Los resultados obtenidos después del entrenamiento evidencian una mejora en la imagen pixelada, dependiendo de los parámetros utilizados se pueden notar varios cambios en esta como lo puede ser la definición y el color, siendo este ultimo el mas notable cuando la red no esta bien entrenada.
- Con los datos representado en los gráficos se nota el mejoramiento de la capacidad de la red profunda para mejorar imágenes, aunque si se mira de manera mas amplia los datos se nota que aun se podrían mejorar mas o que al menos la calidad tampoco resulta ser la mejor en comparación a la imagen original, lo que indica que se pueden hacer varios ajustes para tratar de mejorar los datos.

**Referencias**

https://pytorch.org/docs/stable/nn.html#module-torch.nn.utils