GenPer: Software de tomografía por microondas

Año 2021

Autor: Estanislao María Ledesma

Tutor: Dr. César Federico Caiafa

Índice

Introducción

GenPer ("Generador de permitividades") es un software capaz de reconstruir imágenes de permitividades de dieléctricos a partir de sus datos de dispersión obtenidos de tomografías por microondas (problema de dispersión inversa o inverse scattering problem - ISP). Este software está basado en la red neuronal desarrollada por Zhun Wei en [1].

En este documento se detallan todas las funcionalidades de GenPer, cómo usarlo, sus entradas y salidas. También se detallan los contenidos de cada carpeta y cómo monitorear la correcta ejecución de cada comando.

Estructura general del proyecto

El proyecto está dividido en carpetas, donde cada una tiene un propósito diferente. A continuación se detallan los contenidos de las mismas.

configs

Contiene todos los hiperparámetros, rutas de archivos y cualquier cosa configurable del proyecto, como el logger. Acá se definen variables como número de imágenes, Tamaño del batch, intervalo de permitividades, etc.

data

Contiene todos los archivos generados durante el procesamiento del software. Posee una carpeta para cada tipo de ejecución: generación de imágenes, preprocesamiento, entrenamiento, etc.

dataloader

Contiene todas las clases para carga y preprocesamiento de datos. Acá se define la forma de las imágenes, cómo se generan, cómo se preprocesan y las formas que tienen.

docs

Contiene toda la documentación del proyecto.

evaluation

Contiene todas las clases encargadas de evaluar el rendimiento y exactitud del modelo. Acá está la clase encargada de testear el modelo y la encargada de validar el modelo con la versión de [1].

executor

Contiene todas las clases encargadas de entrenar el modelo, tanto en CPU como GPU.

logs

Contiene los logs de cada ejecución junto con imágenes generadas durante la misma. Posee una carpeta para cada tipo de ejecución: generación de imágenes, preprocesamiento, entrenamiento y, a su vez, subcarpetas con imágenes de muestra de cada ejecución.

model

Contiene todas las clases que conforman la red neuronal en sí. Esto es, las capas del modelo y la red compuesta por todas las capas.

scripts

Contiene los scripts para ejecutar cada funcionalidad del código por separado. Los mismos se ejecutan a través de:

```
python <opciones> <script.py>
```

tests

Contiene todos los archivos de tests unitarios y funcionales.

utils

Contiene todas las clases que pueden ser reutilizadas a lo largo de todo el proyecto. Como las clases que manejan archivos, clases que grafican, etc.

Funcionalidades

GenPer puede ser ejecutado para una variedad de funcionalidades, desde la básica, que procesa imágenes con círculos, hasta validación del modelo y ejecución con el set de datos MNIST [2].

Ejecución básica

Esta consiste de cuatro comandos: el primero, que genera las imágenes, el segundo, que las preprocesa, el tercero, que entrena el modelo, y el cuarto, que testea el modelo.

Generación de imágenes

En este paso se genera un número customizable de imágenes con círculos en posiciones aleatorias de las mismas, de las cuales se obtiene su respectivo campo eléctrico (dado por las antenas emisoras y receptoras). Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python <opciones> generate_images.py
```

Dentro de sus *<opciones>* se encuentran:

- `-t` o `--test` para generar 10 imágenes predeterminadas que se utilizan para los tests unitarios y funcionales.
- `-r` o `--rectangles` para que en lugar de círculos, las imágenes se generen con rectángulos.

A su vez, se puede configurar en el archivo *configs/basic_parameters.json*:

- El número de imágenes (no_of_images)
- El tipo incidencia de la onda (wave_type)
- El número de antenas emisoras (no_of_transmitters)
- El número de antenas receptoras (no_of_receivers)

- La permitividad mínima de los círculos (min_permittivity)
- La permitividad máxima de los círculos (max_permittivity)
- El radio del círculo donde se encuentran las antenas emisoras (transmitter_radius)
- El radio del círculo donde se encuentran las antenas receptoras (receiver_radius)
- El largo del dominio de la imagen (max_diameter)
- El número de píxeles de las imágenes - ancho y alto (no_of_pixels)
- El mínimo radio de los círculos (min_radius)
- El máximo radio de los círculos (max_radius)
- El largo mínimo de los lados de los rectángulos (min_side)
- El largo máximo de los lados de los rectángulos (max_side)

Como salida se genera un archivo images.pkl en la carpeta *data/image_generator*. En el caso de ser una ejecución de test, el archivo se genera en *data/image_generator/test*.

Los logs de la ejecución se separan por fecha de ejecución y se encuentran en la carpeta *logs/image_generator*. A su vez, se generan gráficos de algunas de las imágenes en *logs/image_generator/images* o *logs/image_generator/images/test* en el caso de ejecución de test.

Preprocesamiento de imágenes

En este paso se preprocesan las imágenes generadas en el punto anterior. Se toma la salida de la generación de imágenes y se procesa hasta generar un conjunto de imágenes preliminares para utilizar a la entrada de la red neuronal. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python <opciones> preprocess_images.py
```

Dentro de sus *<opciones>* se encuentran:

- `-t` o `--test` para preprocesar las 10 imágenes predeterminadas que se utilizan para los tests unitarios y funcionales.

En este caso no hay gran variedad de parámetros customizables, ya que la mayoría se utiliza en el primer paso y si se cambia en este, es probable que se obtengan resultados

erróneos. No obstante, se puede cambiar en el archivo *configs/basic_parameters.json* el siguiente parámetro:

- Nivel de ruido aplicado en el preprocesamiento (*noise_level*)

Como salida se genera un archivo *preprocessed_images.pkl* en la carpeta *data/preprocessor.* En el caso de ser una ejecución de test, el archivo se genera en *data/preprocessor/test*.

Los logs de la ejecución se separan por fecha de ejecución y se encuentran en la carpeta *logs/preprocessor*. A su vez, se generan gráficos de algunas de las imágenes en *logs/preprocessor/preprocesed_images* o *logs/preprocessor/preprocesed_images/test* en el caso de ejecución de test.

Entrenamiento del modelo

En este paso se crea la red neuronal junto con todas sus capas, se divide el set de datos generado anteriormente en 3 (entrenamiento, validación y testing) y se entrena/valida la red con los dos primeros, respectivamente. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python <opciones> train_model.py
```

Dentro de sus *<opciones>* se encuentran:

- `-t` o `--test` para entrenar el modelo con las 10 imágenes predeterminadas que se utilizan para los tests unitarios y funcionales.
- `-l` o `--load` para retomar la ejecución anteriormente interrumpida. Si no se ingresa esta opción y hay una ejecución interrumpida, entonces esta información se borrará.

A su vez, se puede configurar en el archivo *configs/basic_parameters.json*:

- Tamaño del batch (*batch_size*)
- Acumulación de pasos en los que no se optimiza el modelo (*accumulation_steps*)
- Semilla para aleatorizar el set de datos (*manual_seed*)
- Número de workers en paralelo para cargar el set de datos (*num_workers*)
- Número de iteraciones (*num_epochs*)
- Learning rate (*learning_rate*)

- Weight decay (*weight_decay*)
- Proporción del set de validación (*val_proportion*)
- Proporción del set de testing (*test_proportion*)

Como salida se genera un archivo *trained_model.pt* en la carpeta *data/trainer.* En el caso de ser una ejecución de test, el archivo se genera en data/trainer/test.

Los logs de la ejecución, como la duración del entrenamiento o el avance del error, se separan por fecha de ejecución y se encuentran en la carpeta *logs/trainer*. Allí también se puede encontrar un gráfico del error de *training/validación* en función del número de iteración. A su vez, se generan gráficos de algunas de las imágenes en *logs/trainer/training_images* para imágenes de entrenamiento o *logs/trainer/validation_images*, para imágenes de validación. Las carpetas *logs/trainer/training_images/test* y *logs/trainer/validation_images/test* son las análogas en el caso de ser una ejecución de test.

Testing del modelo

En este paso se testea el modelo entrenado en el paso anterior, con el set de testing también obtenido durante el entrenamiento. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python <opciones> test_model.py
```

Dentro de sus *<opciones>* se encuentran:

- `-t` o `--test` para testear el modelo con las imágenes predeterminadas que se utilizan para los tests unitarios y funcionales.

En este caso tampoco es posible customizar parámetros ya que se la gran mayoría se comparten con el entrenamiento del modelo y podría resultar en errores.

En este caso no hay salida ya que solo se testea el modelo entrenado y se obtienen métricas.

Los logs de la ejecución, como el error total/medio de entrenamiento, validación y testing o el desvío estándar de los errores de testing, se separan por fecha de ejecución y se encuentran en la carpeta *logs/tester.* A su vez, se generan gráficos de algunas de las imágenes en *logs/tester/testing_images* o *logs/tester/testing_images/test* en el caso de ser una ejecución de test.

Ejecución con set de datos MNIST

Esta consiste de cuatro comandos: el primero, que genera las imágenes a partir del set de datos MNIST, el segundo, que las preprocesa, el tercero, que entrena el modelo, y el cuarto, que testea el modelo.

Generación del set de datos MNIST

En este paso se descarga, si no se descargó anteriormente, el set de datos MNIST en la carpeta *data/mnist_dataset_generator/MNIST.* Se lo procesa, generando un set con los mismos parámetros que las imágenes de círculos, obteniendo imágenes con un único dígito o con dos superpuestos. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python generate_mnist_dataset.py
```

A su vez, se puede configurar en el archivo *configs/basic_parameters.json*:

- El tipo incidencia de la onda (*wave_type*)
- El número de antenas emisoras (*no_of_transmitters*)
- El número de antenas receptoras (*no_of_receivers*)
- La permitividad mínima de los círculos (*min_permittivity*)
- La permitividad máxima de los círculos (*max_permittivity*)
- El largo del dominio de la imagen (*max_diameter*)
- El número de píxeles de las imágenes - ancho y alto (*no_of_pixels*)

Como salida se genera un archivo *images.pkl* en la carpeta *data/ mnist_dataset_generator/mnist_training_images* para entrenamiento y validación, y otro en *data/mnist_dataset_generator/mnist_testing_images* para testing.

Los logs de la ejecución se separan por fecha de ejecución y se encuentran en la carpeta *logs/mnist_dataset_generator*. A su vez, se generan gráficos de algunas de las imágenes en *logs/mnist_dataset_generator/mnist_training_images* para imágenes de entrenamiento o *logs/mnist_dataset_generator/mnist_testing_images*, para imágenes de testing.

Preprocesamiento del set de datos MNIST

Este paso es análogo al preprocesamiento básico, solo que utiliza los archivos generados en el paso anterior de MNIST para entrenamiento y testing. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python preprocess_mnist_dataset.py
```

Como salida se genera un archivo *preprocessed_images.pkl* en la carpeta *data/preprocessor/mnist_training_images* para entrenamiento y validación, y otro en *data/preprocessor/mnist_testing_images* para testing.

Los logs de la ejecución se separan por fecha de ejecución y se encuentran en la carpeta *logs/preprocessor.* A su vez, se generan gráficos de algunas de las imágenes en *logs/preprocessor/mnist_preprocessed_images/mnist_training_images* para imágenes de entrenamiento o *logs/preprocessor/mnist_preprocessed_images/mnist_testing_images*, para imágenes de testing.

Entrenamiento del modelo con set de datos MNIST

Este paso toma las imágenes preprocesadas del set de datos MNIST para entrenamiento, lo divide en dos y entrena/valida el modelo. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python <opciones> train_model_with_mnist_dataset.py
```

Dentro de sus <*opciones*> se encuentran:

- `-l` o `--load` para retomar la ejecución anteriormente interrumpida. Si no se ingresa esta opción y hay una ejecución interrumpida, entonces esta información se borrará.

A su vez, se puede configurar en el archivo *configs/basic_parameters.json*:

- Tamaño del batch (*mnist_batch_size*)
- Acumulación de pasos en los que no se optimiza el modelo (*accumulation_steps*)
- Semilla para aleatorizar el set de datos (*manual_seed*)
- Número de workers en paralelo para cargar el set de datos (*num_workers*)
- Número de iteraciones (*num_epochs*)
- Learning rate (*learning_rate*)

- Weight decay (*weight_decay*)
- Proporción del set de validación (*val_proportion*)
- Proporción del set de testing (*test_proportion*)

Como salida se genera un archivo *trained_model.pt* en la carpeta *data/trainer/mnist*.

Los logs de la ejecución, con la duración del entrenamiento por ejemplo o el avance de los errores, se separan por fecha de ejecución y se encuentran en la carpeta *logs/ trainer*, en *logs/trainer/mnist* también se guarda un gráfico de los errores de entrenamiento/validación en función del número de iteración. A su vez, se generan gráficos de algunas de las imágenes en *logs/trainer/mnist/training_images* para imágenes de entrenamiento o *logs/trainer/mnist/validation_images*, para imágenes de validación.

Testing del modelo con set de datos MNIST

En este paso se testea el modelo entrenado anteriormente, utilizando las imágenes de testing del set de datos MNIST. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python test_model.py
```

En este caso no hay salida ya que solo se testea el modelo entrenado y se obtienen métricas.

Los logs de la ejecución, como el error total/medio de entrenamiento, validación y testing o el desvío estándar de los errores de testing, se separan por fecha de ejecución y se encuentran en la carpeta *logs/tester*. A su vez, se generan gráficos de algunas de las imágenes en *logs/tester/mnist/testing_images*.

Ejecución de validación con datos de MATLAB [1]

Esta ejecución sirve para comparar GenPer con el software desarrollado en MATLAB en [1]. Para esto, se necesita haber realizado una ejecución completa del software en MATLAB, agregando las siguientes línea en el archivo Display_Results_all_Results.m, luego de terminar la iteración de las imágenes de testing (se muestran en rojo las líneas preexistentes para ver el contexto):

```
err_bp(t)=norm(reshape(epsil_bp,[],1)-reshape(epsil_exa,[],1))/norm(reshape(epsil_exa,[]
```

```
,1));
err_rec(t)=norm(reshape(epsil_rec,[],1)-reshape(epsil_exa,[],1))/norm(reshape(epsil_exa,[
],1));
end
err_rec
training_results = info.train;
validation_results = info.val;
save('network_results.mat', 'training_results', 'validation_results', 'err_rec');
mean(err_rec)
```

Una vez que se haya ejecutado el código matlab y ya se tenga el archivo *Forward_Circ1.mat* generado en el primer paso (*data_generate_Circle_Es.m*) y el archivo *network_results.mat,* generado en el último al agregar las líneas de código en *Display_Results_all_Results.m*, se deben guardar en la carpeta de GenPer: el primero en *data/matlab_image_generator* y el segundo en *data/matlab_validator*. Una vez que se tenga esto, se comienza con la ejecución.

Generación de imágenes de MATLAB

En este paso se genera el mismo conjunto de imágenes que el que se generó previamente en la ejecución de MATLAB. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python -f <archivo 1 de matlab> generate_matlab_images.py
```

"Archivo 1 de matlab" debe ser el nombre del archivo previamente guardado en data/matlab_image_generator.

A su vez, se puede configurar en el archivo *configs/basic_parameters.json*:

- El número de imágenes (*no_of_images*)
- El tipo incidencia de la onda (*wave_type*)
- El número de antenas emisoras (*no_of_transmitters*)
- El número de antenas receptoras (*no_of_receivers*)
- La permitividad mínima de los círculos (*min_permittivity*)
- La permitividad máxima de los círculos (*max_permittivity*)

- El radio del círculo donde se encuentran las antenas emisoras (*transmitter_radius*)
- El radio del círculo donde se encuentran las antenas receptoras (*receiver_radius*)
- El largo del dominio de la imagen (*max_diameter*)
- El número de píxeles de las imágenes - ancho y alto (*no_of_pixels*)
- El mínimo radio de los círculos (*min_radius*)
- El máximo radio de los círculos (*max_radius*)

Es importante que estos parámetros sean iguales a los utilizados en la ejecución de MATLAB.

Como salida se genera un archivo *images.pkl* en la carpeta *data/ matlab_image_generator.*

Los logs de la ejecución se separan por fecha de ejecución y se encuentran en la carpeta *logs/matlab_image_generator*. A su vez, se generan gráficos de algunas de las imágenes en *logs/matlab_image_generator/images*.

Preprocesamiento de imágenes de MATLAB

En este paso se preprocesan las imágenes generadas en el paso anterior. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python preprocess_matlab_images.py
```

Como salida se genera un archivo *preprocessed_images.pkl* en la carpeta *data/preprocessor/matlab.*

Los logs de la ejecución se separan por fecha de ejecución y se encuentran en la carpeta *logs/preprocessor.* A su vez, se generan gráficos de algunas de las imágenes en *logs/preprocessor/matlab_preprocesed_images*.

Entrenamiento del modelo con imágenes de MATLAB

En este paso se crea la red neuronal junto con todas sus capas, se divide el set de datos de MATLAB generado anteriormente en 3 (entrenamiento, validación y testing) y se entrena/valida la red con los dos primeros, respectivamente. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python <opciones> train_model_with_matlab_images.py
```

Dentro de sus *<opciones>* se encuentran:

- `-l` o `--load` para retomar la ejecución anteriormente interrumpida. Si no se ingresa esta opción y hay una ejecución interrumpida, entonces esta información se borrará.

A su vez, se puede configurar en el archivo *configs/basic_parameters.json*:

- Tamaño del batch (*batch_size*)
- Acumulación de pasos en los que no se optimiza el modelo (*accumulation_steps*)
- Semilla para aleatorizar el set de datos (*manual_seed*)
- Número de workers en paralelo para cargar el set de datos (*num_workers*)
- Número de iteraciones (*num_epochs*)
- Learning rate (*learning_rate*)
- Weight decay (*weight_decay*)
- Proporción del set de validación (*val_proportion*)
- Proporción del set de testing (*test_proportion*)

Es importante que estos parámetros sean iguales a los utilizados en la ejecución de MATLAB.

Como salida se genera un archivo trained_model.pt en la carpeta *data /trainer/matlab*.

Los logs de la ejecución, como la duración del entrenamiento o el avance del error, se separan por fecha de ejecución y se encuentran en la carpeta *logs/trainer*. En l*ogs/trainer/matlab* también se puede encontrar un gráfico del error de training/validación en función del número de iteración. A su vez, se generan gráficos de algunas de las imágenes en *logs/trainer/matlab/training_images* para imágenes de entrenamiento o *logs/trainer/matlab/validation_images*, para imágenes de validación.

Testing del modelo con imágenes de MATLAB y validación

En este paso se testea el modelo entrenado anteriormente, utilizando las imágenes de testing del set de datos MNIST. Luego, se valida el mismo contra el archivo con métricas

obtenidas de la ejecución MATLAB. Para ejecutarlo se debe ingresar el siguiente comando en la carpeta *scripts*:

```
python -f <archivo 2 de matlab> test_model_with_matlab_images.py
```

"Archivo 2 de matlab" debe ser el nombre del archivo guardado en data/matlab_validator.

En este caso no hay salida ya que solo se testea el modelo entrenado y se obtienen métricas.

Los logs de la ejecución, como el error total/medio de entrenamiento, validación y testing o el desvío estándar de los errores de testing, se separan por fecha de ejecución y se encuentran en la carpeta *logs/tester*. A su vez, se generan gráficos de algunas de las imágenes en *logs/tester/matlab/testing_images*. Además, en la carpeta *logs/ matlab_validator,* se generan logs comparando las métricas de los errores. A su vez, se generan gráficos comparando errores de entrenamiento y validación en función del número de iteración y otro comparando errores de cada imagen de testing.

Contacto

Ante cualquier duda contactarse con Estanislao Ledesma en el correo: estanislaomledesma@gmail.com

Referencias

(1) Z. Wei and X. Chen, “Deep learning schemes for full-wave nonlinear inverse scattering problems,” IEEE Transactions on Geoscience and Remote Sensing, 57 (4), pp. 1849-1860, 2019. URL: https://www.ece.nus.edu.sg/stfpage/elechenx/Papers/TGRS_Learning.pdf

(2) LeCun, Yann and Cortes, Corinna. "MNIST handwritten digit database." (2010). URL: http://yann.lecun.com/exdb/mnist/