



Institución
Universitaria
Reacreditada en Alta Calidad

MANUAL DE USUARIO

David Jiménez Murillo
Leonardo Duque Muñoz
Andrés Eduardo Castro Ospina

GENERADOR AUTOMÁTICO DE DATASETS DE PYTORCH

Índice

1. Introducción a Datasets y Dataloaders en Pytorch	2
1.1. Datasets preexistentes en Pytorch	2
1.2. Cargar un dataset preexistente	2
2. Generador Automático de Datasets de Pytorch	3
3. Prerrequisitos	3
4. Guía de Usuario	3
4.1. Organización de la base de datos	3
4.2. Ejecución del programa	4
4.2.1. Especificación de la ubicación de los datos	4
4.2.2. Cambio de resolución de las imágenes	4
4.2.3. Formatos de imagen soportados	5
4.2.4. Transformadas de torchvision	5
4.2.5. Generación de código fuente	6
4.3. Cómo importar la clase en el código fuente	7

1. Introducción a *Datasets* y *Dataloaders* en *Pytorch*

La idea detrás de los *datasets* en *Pytorch* es desacoplar el código para cargar y preprocesar los datos de un *dataset* del código de entrenamiento de un modelo con el fin de facilitar el mantenimiento y legibilidad del código. *Pytorch* dispone de dos clases para este propósito:

- `torch.utils.data.DataLoader`
- `torch.utils.data.Dataset`

que permiten cargar y usar los *datasets* preexistentes en la librería *torchvision* así como bases de datos propias. La clase *Dataset* almacena los datos y etiquetas en memoria, mientras que la clase *Dataloader* crea un iterable alrededor de una instancia cualquiera de la clase *Dataset* para acceder a las muestras.

1.1. *Datasets* preexistentes en *Pytorch*

Pytorch cuenta con algunas de las bases de datos más comunes en aprendizaje de máquinas. Entre los tipos de *datasets* preexistentes en *Pytorch* se encuentran: [imágenes](#), [texto](#), y [audio](#).

1.2. Cargar un *dataset* preexistente

Los *datasets* preexistentes de *pytorch* se encuentran en las librerías *torchvision*, *torchtext* y *torchaudio*. Aunque para cargar cada *dataset* preexistente en las librerías se tienen diferentes parámetros, la función de cada uno de ellos en general viene siendo similar. Por ejemplo, para cargar la base de datos “Fashion-MNIST” de acuerdo a la [documentación oficial](#), se tiene:

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor

training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

En este código fuente se crea instancia que hereda de la clase *dataset*, dónde:








- “root” es la ruta donde se encuentran los datos almacenados
- “train” especifica si se va a usar la partición de entrenamiento o de pruebas
- “download” especifica si se quiere descargar el dataset o no

- “transform” y “target_transform” especifican las transformaciones que se quieren hacer sobre los datos y las etiquetas

2. Generador Automático de *Datasets* de *Pytorch*

Este generador es una herramienta diseñada con el fin de facilitar el proceso de carga de datos para algoritmos de aprendizaje de máquina, escritos usando la librería *Pytorch*, cuando se tienen bases de datos propias. A continuación damos un guía de su funcionamiento.

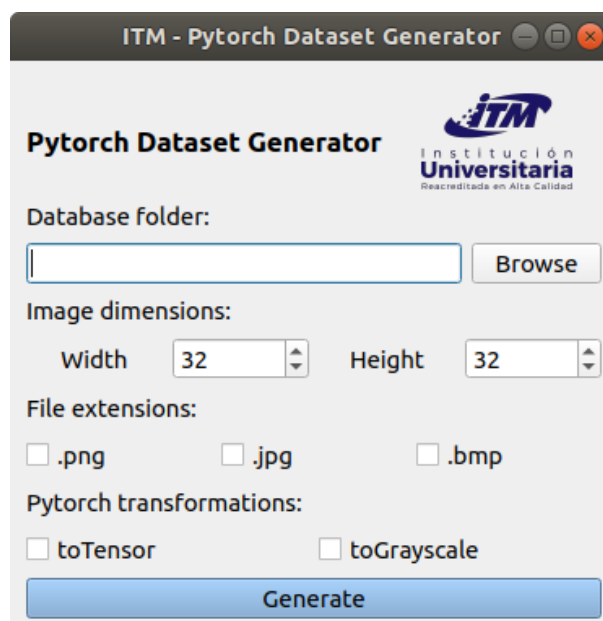
3. Prerrequisitos

-  **Operating system:**  Windows,  Linux
-  **Lenguaje de programación:**  Python >3.6
-  Cualquier editor de texto.
-  Un medio de almacenamiento para guardar el archivo generado.

4. Guía de Usuario

El generador automático consta de una sencilla interfaz gráfica desde la cual el usuario puede establecer los parámetros que requiere. La interfaz gráfica de usuario se muestra en la Figura 1:

Figura 1: Interfaz gráfica de usuario

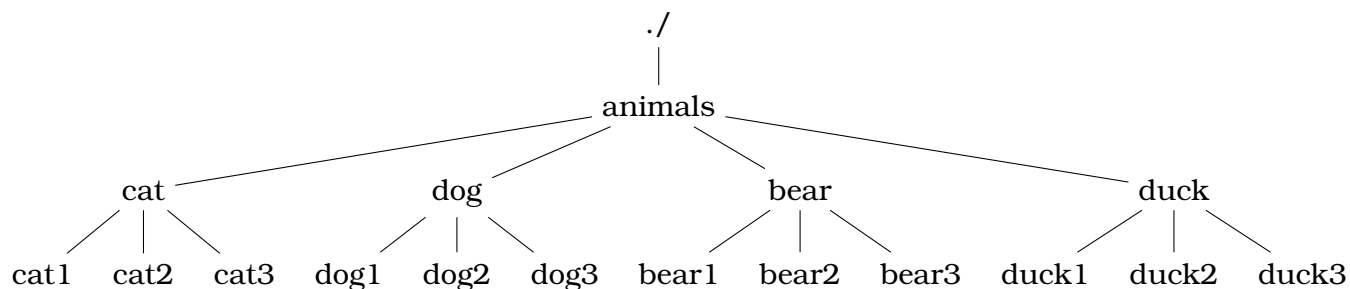


4.1. Organización de la base de datos

Previamente a usar el programa para generar el código fuente, se deben organizar los datos de tal forma que haya una carpeta principal y dentro de esta estén las imágenes separadas por clase dentro de diferentes subcarpetas. Cada subcarpeta aloja las imágenes de una determinada clase y debe tener el nombre de dicha clase. En la figura se muestra la estructura que debe tener el directorio.

4.2. Ejecución del programa

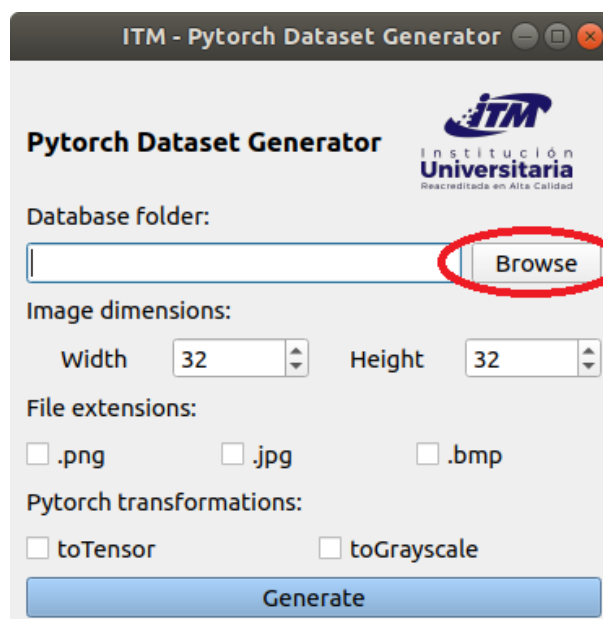
Luego de tener organizada la base de datos debe dirigirse a la carpeta donde se encuentra el archivo ejecutable y hacer doble clic en este para abrir el programa.



4.2.1. Especificación de la ubicación de los datos

En primer lugar se debe buscar la carpeta donde se encuentran los datos, para ello se debe hacer clic en el botón *Browse* para abrir el navegador de carpetas y seleccionar la indicada. Una vez seleccionada la carpeta, aparecerá su ruta en el cuadro de texto a la izquierda del botón, también puede introducirse manualmente la ruta por medio del teclado. En la Figura 2 se señala la ubicación del botón *Browse*:

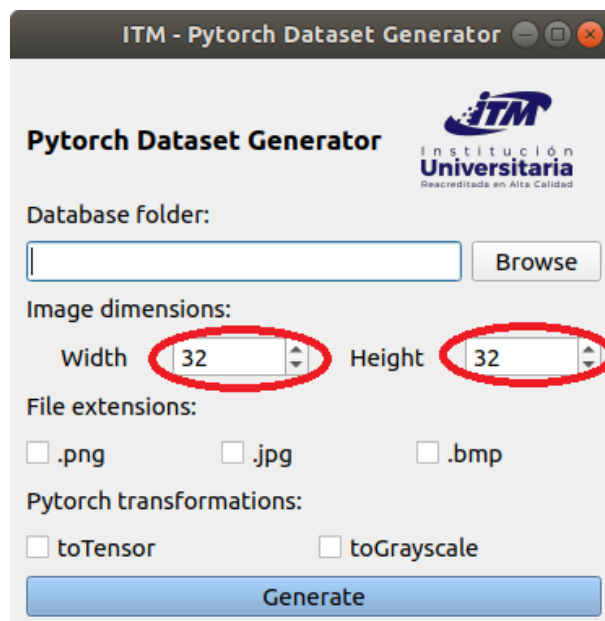
Figura 2: Ubicación del botón *Browse* en la interfaz




4.2.2. Cambio de resolución de las imágenes

Luego se debe indicar al programa la dimensión o resolución en píxeles a la que se desea cargar las imágenes para alimentar el algoritmo de aprendizaje. Por defecto se encuentra establecido con dimensiones de 32×32 como se muestra en la Figura 3. Por teclado puede asignarse cualquier tamaño o por medio de los botones incrementar o decrementar este valor de uno en uno.

Figura 3: Cuadro de texto y botones para establecer las dimensiones a las que se desea cargar las imágenes



ITM - Pytorch Dataset Generator

Pytorch Dataset Generator 
Institución Universitaria
Reacreditada en Alta Calidad

Database folder:

Image dimensions:
Width Height

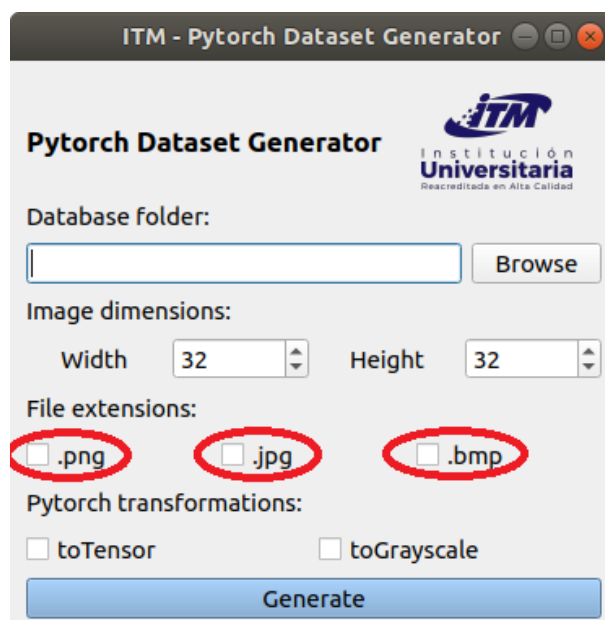
File extensions:
☐ .png ☐ .jpg ☐ .bmp

Pytorch transformations:
☐ toTensor ☐ toGrayscale


4.2.3. Formatos de imagen soportados

Después de especificar el tamaño debe indicarse los formatos de imagen con los que se desea trabajar. Los formatos incluidos son jpg, png y bmp como se muestra en la Figura 4. Puede elegirse varios al tiempo, para ello basta con señalar la casilla correspondiente al formato que se desea cargar.

Figura 4: Extensiones de archivo soportadas



ITM - Pytorch Dataset Generator

Pytorch Dataset Generator 
Institución Universitaria
Reacreditada en Alta Calidad

Database folder:

Image dimensions:
Width Height

File extensions:
☐ .png ☐ .jpg ☐ .bmp

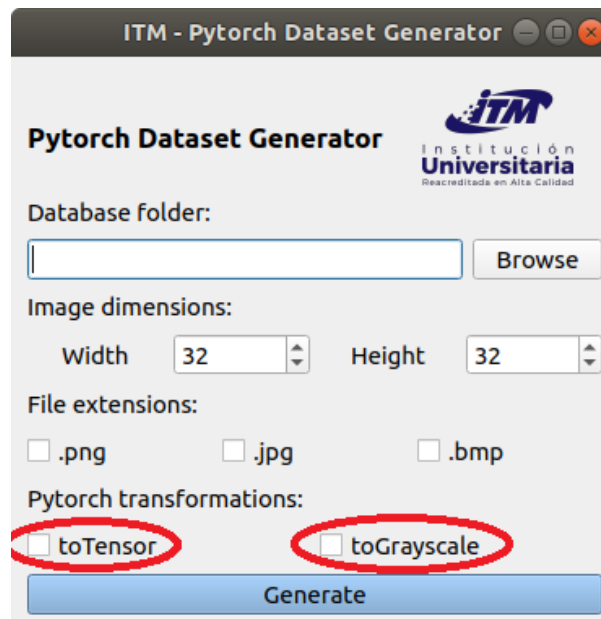
Pytorch transformations:
☐ toTensor ☐ toGrayscale

4.2.4. Transformadas de torchvision

Incluimos dos transformadas que son “toTensor” la cual convierte el arreglo de la imagen a tensor, y una segunda, “toGrayscale” para convertirlas a escala de grises, este par de transformadas son algunas de las más comunes en aprendizaje de máquina.

En la Figura 5 se señalan las transformadas de *torchvision* que se desea incluir en el *dataloader*.

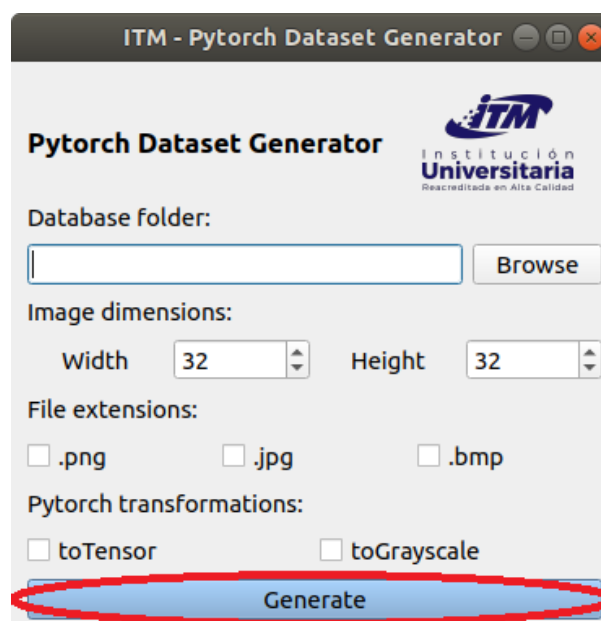
Figura 5: Transformadas de *torchvision*



4.2.5. Generación de código fuente

Por último, haciendo clic en el botón *Generate* como se muestra en la Figura 6, se creará un archivo de código fuente en *Python* llamado "dl.py"; este archivo contiene la clase *dataset* de acuerdo con la configuración establecida por el usuario en los pasos anteriores. El archivo con el código fuente se genera en el mismo directorio donde se encuentra ubicado el archivo ejecutable del programa. La clase *dataset* contenida en el archivo está lista para ser importada en el código de entrenamiento del algoritmo de aprendizaje.

Figura 6: Ubicación del botón *Generate*



4.3. Cómo importar la clase en el código fuente

El proceso de carga de la clase *dataset* es bastante sencillo, tan solo se necesita importar el archivo generado con esta línea de código:

```
import dl
```

Luego debe crearse una instancia de la clase “ImageDataset”, que es la que produce el código fuente y enviarla como argumento en una nueva instancia de la clase “Dataloader”, como en el siguiente fragmento de código:

```
animales = dl.ImageDataset()  
animales_ds = DataLoader(animales, batch_size=2, shuffle=True)
```

La estructura completa del código de carga es la siguiente:

```
from PIL import Image  
import torch  
from torch.utils.data import Dataset, DataLoader  
from torchvision import transforms  
import dl  
  
transformations = transforms.Compose([transforms.ToTensor()])  
animales = dl.ImageDataset()  
  
animales_ds = DataLoader(animales, batch_size=2, shuffle=True)  
images, labels = next(iter(animales_ds))
```