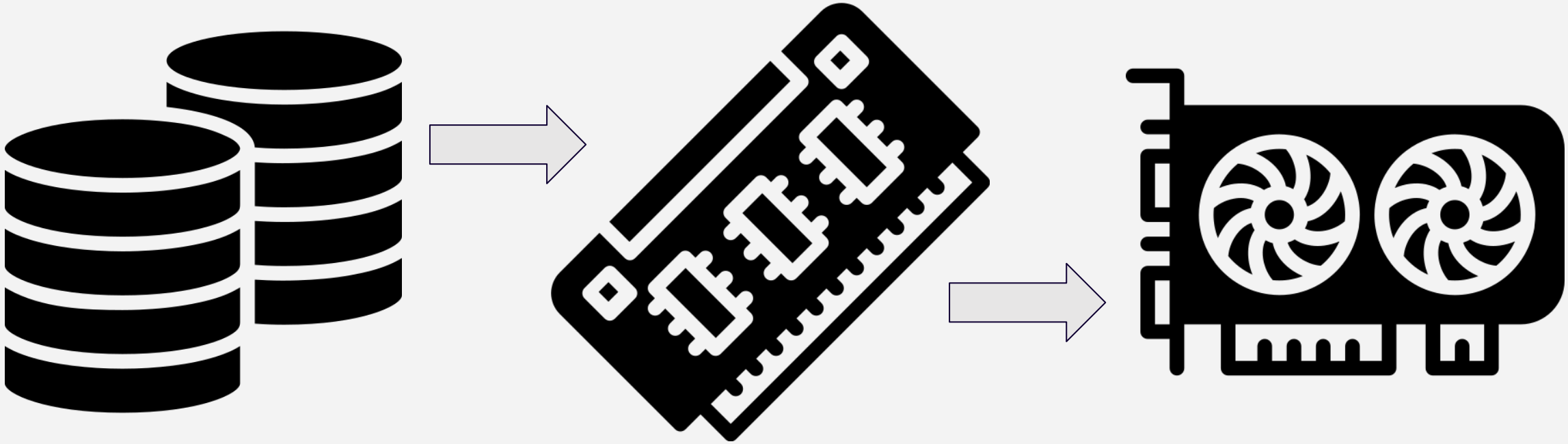
# Aprendizaje Automático Profundo (Deep Learning)

**Dr. Facundo Quiroga - Dr. Franco Ronchetti**

# Carga de datos

# ¿Cómo cargar conjuntos de datos?

- Dos enfoques principales
  - a) Cargar todo en memoria (si entra)
  - b) Cargar *batchs* a medida que se usan
    - Libero memoria de batches usados
    - Múltiples threads si la carga es lenta



# ¿Cómo cargar datos para entrenar?

- Cargar todo en memoria (si entra)
  - Es el enfoque que usamos hasta ahora con MNIST, CIFAR10 y todos los conjuntos de las prácticas 2 y 3

```
#cargar todos los datos en memoria
x,y=load_data()
print(x_shape[0]) # 10000 ejemplos


…
model= …
# Entrenar con 32 ejemplos por vez
model.fit(x,y,batch_size=32)
```

# Carga de datos con un **generator**

- Utilizando un *generator*

```
def generator(batch_size):
   n= 10000
   while True: # generador infinito
      for i in range(n//batch_size):
         index=i*batch_size
         x,y=load_samples(from=index,to=index+batch_size)
         yield x,y
model= …
# Entrenar con 32 ejemplos por batch
# Carga a memoria a medida que se usa
model.fit_generator(generator(32),
                    steps_per_epoch=10000/32,...)
```

**`load_samples`** carga las imágenes desde **`from`** hasta **`to`** desde disco, red, etc

`generator` es infinito => `steps_per_epoch` para saber cuantos batchs por época

# **fit, predict y evaluate** para generators

- Versiones de *fit*, *evaluate* y *predict* con generators
  - Implementadas en clase Model

```
#todos los datos en memoria
x,y=load_data()
…
model.fit(x,y,batch_size=32)
…
model.predict(x,y,batch_size=32)
…
model.evaluate(x,y,batch_size=32)
```

```
def generator(batch_size):
   …
g=generator(32)
model.fit_generator(g)
…
model.predict_generator(g)
…
model.evaluate_generator(g)
```

# Carga de datos en paralelo con **Sequence**

- Utilizando un *generator* subclasificando *Sequence*

```
class CustomSequence(Sequence):
   def __init__(self, batch_size):
      self.batch_size=batch_size
      self.n=10000
   def __len__(self): # cant de batches
      return self.n // self.batch_size
   def __getitem__(self, idx):
      index=idx*batch_size
      x,y=load_samples(from=index,to=index+batch_size)
      return x,y
…
model.fit_generator(CustomSequence(32))
```

- Carga de datos *multithreaded*
  - Carga y entrenamiento en paralelo

No requiere `steps_per_epoch`

# Eficiencia de la carga de datos en paralelo

- Eficiencia de **fit_generator**

- Sin multithreading
  - **def generator()**


- Con multithreading
  - clase **Sequence**

# Carga de imágenes para clasificación con Keras

- *Generators* para imágenes incluidos en Keras

```
datagen = ImageDataGenerator()
train_generator = datagen.flow_from_directory(
        'data/train', # dir de las imagenes
        target_size=(150, 150),# resize tam destino
        batch_size=32,
        class_mode='categorical')
model.fit_generator(train_generator,
        steps_per_epoch=len(train_generator)/32,...)
```

- Por cada clase, una subcarpeta con imágenes.
  - class_mode = ‘sparse’ => Codificar clase con *labels,* enteros
  - class_mode = ‘categorical’ => Codificar clase con vectores one_hot
  - class_mode = None => No cargar la clase