```
import tensorflow as tf

#le code suivant divise le premier GPU (GPU 0) en 4 GPU -périphériques- virtuels. Chacun 2Gio de RAM.
#les instructions suivantes doivent être effectuées juste après l'importation du module tensorflow.

physical_gpus = tf.config.experimental.list_physical_devices("GPU")
tf.config.experimental.set_virtual_device_configuration(
    physical_gpus[0],
    [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=2048),
     tf.config.experimental.VirtualDeviceConfiguration(memory_limit=2048)])
```

```
# imports commun
import numpy as np
import os

# pour rendre stable l'exécution relativement aux nombres aléatoire générés.
np.random.seed(42)

# pour une meilleure visibilité des figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
import os

from tensorflow import keras
```

```
strategy = tf.distribute.MirroredStrategy()
print('Nombre de périphériques (GPU): {}'.format(strategy.num_replicas_in_sync))
```

```
     Nombre de périphériques (GPU): 2
```

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

```
     Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
     170498071/170498071 [==============================] - 4s 0us/step
```

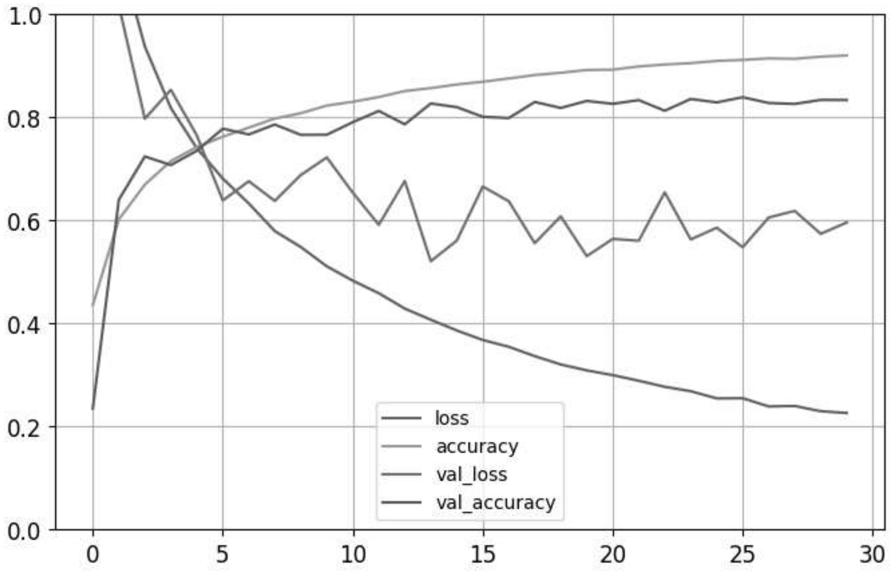
```
num_train_examples = x_train.shape[0]
num_test_examples = x_test[0]

BUFFER_SIZE = 1000

x_valid, x_train = x_train [:5000], x_train[5000:]
y_valid, y_train = y_train [:5000], y_train[5000:]

BATCH_SIZE_PER_REPLICA = 64
BATCH_SIZE = BATCH_SIZE_PER_REPLICA * strategy.num_replicas_in_sync

def scale(image, label):
  image = tf.cast(image, tf.float32)
  image /= 255

  return image, label

train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).map(scale).cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
valid_dataset = tf.data.Dataset.from_tensor_slices((x_valid, y_valid)).map(scale).batch(BATCH_SIZE)
eval_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).map(scale).batch(BATCH_SIZE)
```

```
with strategy.scope():
  model = tf.keras.Sequential([
      tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(32, 32, 3),padding='same'),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Conv2D(64, (3,3), activation='relu',padding='same'),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Conv2D(128, (3,3), activation='relu',padding='same'),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(128, activation='relu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(10,activation='softmax'),
  ])

  model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])


EPOCHS = 30

from time import time
t0 = time()

history = model.fit(train_dataset, epochs=EPOCHS,validation_data=valid_dataset)
tt = time() - t0
print("classifier trained in {} seconds".format(round(tt,3)))
```

```
Epoch 3/30
352/352 [==============================] - 11s 30ms/step - loss: 0.9377 - accuracy: 0.6694 - val_loss: 0.7973 - val_accuracy: 0.7240
Epoch 4/30
352/352 [==============================] - 10s 30ms/step - loss: 0.8182 - accuracy: 0.7151 - val_loss: 0.8534 - val_accuracy: 0.7074
Epoch 5/30
352/352 [==============================] - 10s 28ms/step - loss: 0.7400 - accuracy: 0.7423 - val_loss: 0.7649 - val_accuracy: 0.7350
Epoch 6/30
352/352 [==============================] - 10s 29ms/step - loss: 0.6811 - accuracy: 0.7625 - val_loss: 0.6387 - val_accuracy: 0.7780
Epoch 7/30
352/352 [==============================] - 11s 32ms/step - loss: 0.6328 - accuracy: 0.7800 - val_loss: 0.6760 - val_accuracy: 0.7668
Epoch 8/30
352/352 [==============================] - 10s 30ms/step - loss: 0.5792 - accuracy: 0.7976 - val_loss: 0.6378 - val_accuracy: 0.7860
Epoch 9/30
352/352 [==============================] - 10s 29ms/step - loss: 0.5483 - accuracy: 0.8080 - val_loss: 0.6881 - val_accuracy: 0.7660
Epoch 10/30
352/352 [==============================] - 10s 29ms/step - loss: 0.5109 - accuracy: 0.8228 - val_loss: 0.7221 - val_accuracy: 0.7662
Epoch 11/30
352/352 [==============================] - 11s 31ms/step - loss: 0.4829 - accuracy: 0.8302 - val_loss: 0.6538 - val_accuracy: 0.7906
Epoch 12/30
352/352 [==============================] - 11s 30ms/step - loss: 0.4586 - accuracy: 0.8394 - val_loss: 0.5914 - val_accuracy: 0.8124
Epoch 13/30
352/352 [==============================] - 11s 31ms/step - loss: 0.4288 - accuracy: 0.8510 - val_loss: 0.6764 - val_accuracy: 0.7864
Epoch 14/30
352/352 [==============================] - 10s 28ms/step - loss: 0.4072 - accuracy: 0.8567 - val_loss: 0.5207 - val_accuracy: 0.8268
Epoch 15/30
352/352 [==============================] - 10s 29ms/step - loss: 0.3863 - accuracy: 0.8637 - val_loss: 0.5607 - val_accuracy: 0.8198
Epoch 16/30
352/352 [==============================] - 11s 31ms/step - loss: 0.3678 - accuracy: 0.8690 - val_loss: 0.6657 - val_accuracy: 0.8010
Epoch 17/30
352/352 [==============================] - 11s 30ms/step - loss: 0.3546 - accuracy: 0.8754 - val_loss: 0.6371 - val_accuracy: 0.7986
Epoch 18/30
352/352 [==============================] - 10s 27ms/step - loss: 0.3365 - accuracy: 0.8821 - val_loss: 0.5557 - val_accuracy: 0.8298
Epoch 19/30
352/352 [==============================] - 11s 31ms/step - loss: 0.3203 - accuracy: 0.8865 - val_loss: 0.6078 - val_accuracy: 0.8182
Epoch 20/30
```

```
352/352 [==============================] - 10s 28ms/step - loss: 0.2770 - accuracy: 0.9025 - val_loss: 0.6344 - val_accuracy: 0.8128
Epoch 24/30
352/352 [==============================] - 11s 30ms/step - loss: 0.2684 - accuracy: 0.9052 - val_loss: 0.5630 - val_accuracy: 0.8356
Epoch 25/30
352/352 [==============================] - 11s 31ms/step - loss: 0.2545 - accuracy: 0.9095 - val_loss: 0.5857 - val_accuracy: 0.8290
Epoch 26/30
352/352 [==============================] - 11s 30ms/step - loss: 0.2548 - accuracy: 0.9114 - val_loss: 0.5479 - val_accuracy: 0.8390
Epoch 27/30
352/352 [==============================] - 10s 28ms/step - loss: 0.2389 - accuracy: 0.9144 - val_loss: 0.6056 - val_accuracy: 0.8278
Epoch 28/30
352/352 [==============================] - 10s 29ms/step - loss: 0.2398 - accuracy: 0.9136 - val_loss: 0.6184 - val_accuracy: 0.8260
Epoch 29/30
352/352 [==============================] - 11s 31ms/step - loss: 0.2296 - accuracy: 0.9178 - val_loss: 0.5739 - val_accuracy: 0.8338
Epoch 30/30
352/352 [==============================] - 11s 30ms/step - loss: 0.2261 - accuracy: 0.9200 - val_loss: 0.5958 - val_accuracy: 0.8336
classifier trained in 364.271 seconds
```

```
import pandas as pd

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```


loss
accuracy
val_loss
val_accuracy