Bài 32 - Kĩ thuật tensorflow Dataset

09 Apr 2020 - phamdinhkhanh

Menu

1. Vai trò của tensorflow Dataset

Chắc hẳn các bạn từng thắc mắc vì sao trong deep learning các bộ dữ liệu bigdata có kích thước rất lớn mà các máy tính có RAM nhỏ hơn vẫn có thể huấn luyện được?

Xuất phát từ lý do đó, bài này mình sẽ lý giải các cách thức dữ liệu có thể được truyền vào mô hình để huấn luyện theo cách tiếp cận dễ hiểu nhất. Các bạn chuyên gia và giàu kinh nghiệm huấn luyện mô hình có thể bỏ qua bài viết này vì nó khá cơ bản.

Vì sao có thể truyền các bộ dữ liệu lớn vào mô hình huấn luyện?

Các bộ dữ liệu deep learning thường có kích thước rất lớn. Trong quá trình huấn luyện các model deep learning chúng ta không thể truyền toàn bộ dữ liệu vào mô hình cùng một lúc bởi dữ liệu thường có kích thước lớn hơn RAM máy tính. Xuất phát từ lý do này, các framework deep learning đều hỗ trợ các hàm huấn luyện mô hình theo generator. Dữ liệu sẽ không được khởi tạo ngay toàn bộ từ đầu mà sẽ huấn luyện đến đâu sẽ được khởi tạo đến đó theo từng phần nhỏ gọi là batch.

Tùy theo định dạng dữ liệu là text, image, data frame, numpy array,… mà chúng ta sẽ sử dụng những module tạo dữ liệu huấn luyện khác nhau.

Vậy thì với từng kiểu dữ liệu khác nhau sẽ có phương pháp xử lý như thế nào để đưa vào huấn luyện mô hình? Có những kĩ thuật khởi tạo dataset trong tensorflow nào? Bài viết này mình sẽ giới thiệu tới các bạn.

2. Định nghĩa generator

generator có thể coi là một người vay nợ, được quyền sử dụng tiền của người khác mà không trả ngay. Nếu chúng ta coi tiền là dữ liệu thì ta có thể hình dung generator sẽ sử dụng và biến đổi dữ liệu như cách người vay nợ sử dụng tiền vào các mục đích của mình. Tuy nhiên dữ liệu sau biến đổi không được trả về như các hàm return thông thường của python.

Để đơn giản hóa mình lấy ví dụ một hàm tính lãi suất phải trả theo năm như sau:

Giả sử một người vay n món nợ với cùng lãi suất là 1%/tháng. Để tính lãi suất phải trả của các khoản vay chúng ta có thể sử dụng vòng for và tính để tính kết quả trong 1 lần.

Top

Note: Bạn đọc có thể mở google colab để cùng thực hành tensorflow Dataset - khanh blog (https://colab.research.google.com/drive/1mVwq7Py4Rv2MCDp1lOD8mQ1FXQWJDLlp)

```
import numpy as np
from datetime import datetime

def _interest_rate(month):
  return (1+0.01)**month - 1


periods = [1, 3, 6, 9, 12]
scales = [_interest_rate(month) for month in periods]
print('scales of origin balance: ', scales)
```

```
scales of origin balance:  [0.010000000000000009, 0.030301000000000133
```

Nếu sử dụng generator thì chúng ta chỉ việc thay `return` bằng `yield` .

```
def _gen_interest_rate(month):
  yield (1+0.01)**month - 1


periods = [1, 3, 6, 9, 12]
scales = [_gen_interest_rate(month) for month in periods]
print('scales of origin balance: ', scales)
```

```
scales of origin balance:  [<generator object _gen_interest_rate at 0x
```

Ta thấy generator sẽ không trả về kết quả ngay mà chỉ tạo sẵn các ô nhớ lưu hàm generator mô tả cách tính lãi suất. Do đó chúng ta sẽ không tốn chi phí thời gian để thực hiện các phép tính. Thực tế là chúng ta đang nợ máy tính kết quả trả về. Chỉ khi nào được chủ nợ gọi tên bằng cách kích hoạt trong hàm `next()` thì mới tính kết quả.

```
[next(_gen_interest_rate(0.01, n)) for n in periods]
```

```
[1.01,
 1.0303010000000001,
 1.0615201506010001,
 1.0936852726843609,
 1.1268250301319698]
```

Chúng ta có thể thấy generator có lợi thế là:

- Không sinh toàn bộ dữ liệu cùng một lúc, do đó sẽ nâng cao hiệu suất vì sử dụng ít bộ nhớ hơn.
- Không phải chờ toàn bộ các vòng lặp được xử lý xong thì mới xử lý tiếp nên tiết kiệm thời gian tính toán.

Đó chính là lý do generator chính là giải pháp được lựa chọn cho huấn luyện mô hình deep learning với dữ liệu lớn.

Top

3. Các cách khởi tạo một Dataset

Dataset là một class của tensorflow được sử dụng để wrap dữ liệu trước khi truyền vào mô hình để huấn luyện. Bạn hình dung dữ liệu của bạn có input là ma trận X và output là Y. Ban đầu X và Y chỉ là các dữ liệu thô định dạng numpy. Tất nhiên chúng ta có thể truyền trực tiếp chúng vào hàm `fit()` của mô hình. Nhưng để kiểm soát được X và Y chẳng hạn như fit vào với batch size bằng bao nhiêu? có shuffle dữ liệu hay không thì chúng ta nên wrap chúng trong `tf.Dataset` .

Có 2 phương pháp chính để khởi tạo một tf.Dataset trong tensorflow:

- In memory Dataset: Khởi tạo các dataset ngay từ đầu và dữ liệu được lưu trữ trên memory.
- Generator Dataset: Dữ liệu được sinh ra theo từng batch và xen kẽ với quá trình huấn luyện từ các hàm khởi tạo generator.

Phương pháp `In memory Dataset` sẽ phù hợp với các bộ dữ liệu kích thước nhỏ mà RAM có thể load được. Quá trình huấn luyện theo cách này thì nhanh hơn so với phương pháp `Generator Dataset` vì dữ liệu đã được chuẩn bị sẵn mà không tốn thời gian chờ khởi tạo batch. Tuy nhiên dễ xảy ra `out of memory` trong quá trình huấn luyện.

Theo cách `Generator Dataset` chúng ta sẽ qui định cách mà dữ liệu được tạo ra như thế nào thông qua một hàm `generator` . Quá trình huấn luyện đến đâu sẽ tạo batch đến đó. Do đó các bộ dữ liệu big data có thể được load theo từng batch sao cho kích thước vừa được dung lượng RAM. Theo cách huấn luyện này chúng ta có thể huấn luyện được các bộ dữ liệu có kích thước lớn hơn nhiều so với RAM bằng cách chia nhỏ chúng theo batch. Đồng thời có thể áp dụng thêm các step preprocessing data trước khi dữ liệu được đưa vào huấn luyện. Do đó đây thường là phương pháp được ưa chuộng khi huấn luyện các model deep learning.

3.1. In Memory Dataset

Bên dưới chúng ta sẽ thử nghiệm khởi tạo một dtaset trên tensorflow theo phuwong pháp `In Memory Dataset` . Bộ dữ liệu được lựa chọn là chữ số viết tay mnist với kích thước của tập train và validation lần lượt là `60000` và `10000` .

```
%tensorflow_version 2.x

from google.colab import drive
import os

drive.mount("/content/gdrive")
path = 'gdrive/My Drive/Colab Notebooks/TensorflowData'
os.chdir(path)
os.listdir()
```

```
Enter your authorization code:
··········
Mounted at /content/gdrive

['Dog-Cat-Classifier', 'TensorflowDataset.ipynb']
```

Top

```
from tensorflow.keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-kera
11493376/11490434 [==============================] - 0s 0us/step
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

Như vậy các dữ liệu train và test của bộ dữ liệu mnist đã được load vào bộ nhớ. Tiếp theo chúng ta sẽ khởi tạo Dataset cho những dữ liệu in memory này bằng hàm `tf.data.Dataset.from_tensor_slices()`. Hàm này sẽ khai báo dữ liệu đầu vào cho mô hình huấn luyện.

```
import tensorflow as tf
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
valid_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
```

Khi đó chúng ta đã có thể fit vào mô hình huấn luyện các dữ liệu được truyền vào `tf.Dataset` là `(X_train, y_train)`.

Chúng ta cũng có thể áp dụng các phép biến đổi bằng các hàm như `Dataset.map()` hoặc `Dataset.batch()` để biến đổi dữ liệu trước khi fit vào model. Các bạn xem thêm tại tf.Dataset (https://www.tensorflow.org/api_docs/python/tf/data/Dataset). Chẳng hạn trước khi truyền batch vào huấn luyện tôi sẽ thực hiện chuẩn hóa batch theo phân phối chuẩn.

Top

```
import numpy as np
from tensorflow.keras.backend import std, mean
from tensorflow.math import reduce_std, reduce_mean

def _normalize(X_batch, y_batch):
  '''
  X_batch: matrix digit images, shape batch_size x 28 x 28
  y_batch: labels of digit.
  '''
  X_batch = tf.cast(X_batch, dtype = tf.float32)
  # Padding về 2 chiều các giá trị 0 để được shape là 32 x 32
  pad = tf.constant([[0, 0], [2, 2], [2, 2]])
  X_batch = tf.pad(X_batch, paddings=pad, mode='CONSTANT', constant_v
  X_batch = tf.expand_dims(X_batch, axis=-1)
  mean = reduce_mean(X_batch)
  std = reduce_std(X_batch)
  X_norm = (X_batch-mean)/std
  return X_norm, y_batch

train_dataset = train_dataset.batch(32).map(_normalize)
valid_dataset = valid_dataset.batch(32).map(_normalize)
```

`train_dataset` và `valid_dataset` lần lượt thực hiện các bước xử lý dữ liệu sau:

- Hàm `.batch(32)` : Trích xuất ra từ list `(X_train, y_train)` các batch_size có kích thước là 32.
- Hàm `.map(_normalize)` : Mapping đầu vào là các batch `(X_batch, y_batch)` kích thước 32 vào hàm số `_normalize()` . Kết quả trả về là giá trị đã chuẩn hóa theo batch của `X_batch` và `y_batch` . Dữ liệu này sẽ được sử dụng để huấn luyện model.

Huấn luyện và kiểm định model

```
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam

base_extractor = MobileNet(input_shape = (32, 32, 1), include_top = F
flat = Flatten()
den = Dense(10, activation='softmax')
model = Sequential([base_extractor,
                    flat,
                    den])
model.summary()
```

Top

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenet_1.00_32 (Model)    (None, 1, 1, 1024)        3228288
_________________________________________________________________
flatten_5 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                10250
=================================================================
Total params: 3,238,538
Trainable params: 3,216,650
Non-trainable params: 21,888
_________________________________________________________________
```

```
model.compile(Adam(), loss='sparse_categorical_crossentropy', metrics =
model.fit(train_dataset,
          validation_dataset = valid_dataset,
          epochs = 5)
```

```
Epoch 1/5
1875/1875 [==============================] - 687s 366ms/step - loss: 0
Epoch 2/5
1875/1875 [==============================] - 686s 366ms/step - loss: 0
Epoch 3/5
1505/1875 [=======================>......] - ETA: 2:15 - loss: 0.1432
```

3.2. Generator Dataset

Theo cách khởi tạo từ generator chúng ta sẽ không phải ghi nhớ toàn bộ dữ liệu vào RAM. Thay vào đó có thể tạo dữ liệu trong quá trình huấn luyện ở mỗi lượt fit từng batch.

Giả sử bên dưới chúng ta có tên các món ăn được chia thành hai nhóm thuộc các địa phương ‘hà nội’ và ‘hồ chí minh’. Chúng ta sẽ khởi tạo data generator để sinh dữ liệu cho mô hình phân loại món ăn theo địa phương.

```
import pandas as pd

hanoi = ['bún chả hà nội', 'chả cá lã vọng hà nội', 'cháo lòng hà nội',
hochiminh = ['bánh canh sài gòn', 'hủ tiếu nam vang sài gòn', 'hủ tiếu
city = ['hanoi'] * len(hanoi) + ['hochiminh'] * len(hochiminh)
corpus = hanoi+hochiminh

data = pd.DataFrame({'city': city, 'food': corpus})
data.sample(5)
```

	city	food
8	hochiminh	hủ tiếu nam vang sài gòn
1	hanoi	chả cá lã vọng hà nội
7	hochiminh	bánh canh sài gòn

Top

	city	food
0	hanoi	bún chả hà nội
1	hochiminh	bánh phở

```
class Voc(object):
  def __init__(self, corpus):
    self.corpus = corpus
    self.dictionary = {'unk': 0}
    self._initialize_dict(corpus)

  def _add_dict_sentence(self, sentence):
    words = sentence.split(' ')
    for word in words:
      if word not in self.dictionary.keys():
        max_indice = max(self.dictionary.values())
        self.dictionary[word] = (max_indice + 1)

  def _initialize_dict(self, sentences):
    for sentence in sentences:
      self._add_dict_sentence(sentence)

  def _tokenize(self, sentence):
    words = sentence.split(' ')
    token_seq = [self.dictionary[word] for word in words]
    return np.array(token_seq)

voc = Voc(corpus = corpus)
```

corpus là list toàn bộ tên các món ăn. Class Voc có tác dụng khởi tạo index từ điển cho toàn bộ corpus (bộ văn bản).

```
voc.dictionary
```

Tiếp theo chúng ta sẽ khởi tạo một `random_generator` có tác dụng lựa chọn ngẫu nhiên một tên món ăn trong corpus và tokenize chúng.

Top

```
import tensorflow as tf

cat_indices = {
    'hanoi': 0,
    'hochiminh': 1
}

def generators():
  i = 0
  while True:
    i = np.random.choice(data.shape[0])
    sentence = data.iloc[i, 1]
    x_indice = voc._tokenize(sentence)
    label = data.iloc[i, 0]
    y_indice = cat_indices[label]
    yield x_indice, y_indice
    i += 1

random_generator = tf.data.Dataset.from_generator(
    generators,
    output_types = (tf.float16, tf.float16),
    output_shapes = ((None,), ())
)

random_generator
```

```
import numpy as np

random_generator_batch = random_generator.shuffle(20).padded_batch(20,
sequence_batch, label = next(iter(random_generator_batch))

print(sequence_batch)
print(label)
```

Top

```
tf.Tensor(
[[ 8.  9.  3.  4.  0.  0.]
 [10. 11.  0.  0.  0.  0.]
 [17. 18. 19. 20. 15. 16.]
 [22. 23. 15. 16.  0.  0.]
 [13. 14. 15. 16.  0.  0.]
 [13. 23.  0.  0.  0.  0.]
 [17. 18. 19. 20. 15. 16.]
 [ 1.  2.  3.  4.  0.  0.]
 [10. 11.  0.  0.  0.  0.]
 [17. 18. 21. 15. 16.  0.]
 [ 8.  9.  0.  0.  0.  0.]
 [22. 23. 15. 16.  0.  0.]
 [ 2.  5.  6.  7.  3.  4.]
 [13. 14. 15. 16.  0.  0.]
 [ 1.  2.  3.  4.  0.  0.]
 [13. 14. 15. 16.  0.  0.]
 [ 8.  9.  0.  0.  0.  0.]
 [13. 14. 15. 16.  0.  0.]
 [13. 23.  0.  0.  0.  0.]
 [10. 11. 12.  3.  4.  0.]], shape=(20, 6), dtype=float16)
tf.Tensor([0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0
```

hàm `shuffle(20)` có tác dụng trộn lẫn ngẫu nhiên dữ liệu. Sau đó dữ liệu được chia thành những batch có kích thước là 10 và padding giá trị 0 sao cho bằng với độ dài của câu dài nhất bằng hàm `padded_batch()`.

3.2.1. Sử dụng ImageGenerator

ImageGenerator cũng là một dạng data generator được xây dựng trên framework keras và dành riêng cho dữ liệu ảnh.

Đây là một high level function nên cú pháp đơn giản, rất dễ sử dụng nhưng khả năng tùy biến và can thiệp sâu vào dữ liệu kém.

Khi khởi tạo ImageGenerator chúng ta sẽ khai báo các thủ tục preprocessing image trước khi đưa vào huấn luyện. Mình sẽ không quá đi sâu vào các kĩ thuật preprocessing data này. Bạn đọc quan tâm có thể xem thêm tại ImageDataGenerator (https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator).

```
image_gen = tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rescale = 1./255,
    rotation_range = 20,
    horizontal_flip = True
)
```

Tiếp theo chúng ta sẽ truyền dữ liệu vào mô hình thông qua một hàm là `flow_from_directory()`.

Top

```
import glob2

root_folder = 'Dog-Cat-Classifier/Data/Train_Data/'
images, labels = next(image_gen.flow_from_directory(root_folder))
```

```
Found 1399 images belonging to 2 classes.
```

Hàm `flow_from_directory()` sẽ có tác dụng đọc các ảnh từ `root_folder` và lấy ra những thông tin bao gồm ma trận ảnh sau biến đổi và nhãn tương ứng. Cấu trúc cây thư mục của `root_folder` có dạng như sau:

```
root-folder

sub-folder-class-1

sub-folder-class-2

...

sub-folder-class-C
```

Trong đó bên trong các `sub-folder-class-i` là list toàn bộ các ảnh thuộc về một class. Hàm `flow_from_directory()` sẽ tự động xác định các file dữ liệu nào là ảnh để load vào quá trình huấn luyện mô hình.

```
!ls {root_folder}
```

```
cat  dog
```

Ở đây trong root_folder chúng ta có 2 `sub-folders` tương ứng với 2 classes là `dog, cat` .

Tiếp theo ta sẽ khởi tạo một `tf.Dataset` từ generator thông qua hàm `from_generator()` .

```
image_gen_dataset = tf.data.Dataset.from_generator(
    image_gen.flow_from_directory,
    args = ([root_folder]),
    output_types=(tf.float32, tf.float32),
    output_shapes=([32,256,256,3], [32, 1])
)
```

Trong hàm `from_generator()` chúng ta phải khai báo bắt buộc định dạng dữ liệu input và output thông qua tham số `output_types` và output shape thông qua tham số `output_shapes` .

Như vậy kết quả trả ra sẽ là những batch có kích thước 32 và ảnh có kích thước `256 x 256` và nhãn tương ứng của ảnh.

3.2.2. Customize ImageGenerator

Giả sử bạn có một bộ dữ liệu ảnh mà kích thước các ảnh là khác biệt nhau. Đồng thời bạn cũng muốn can thiệp sâu hơn vào bức ảnh trước khi đưa vào huấn luyện như giảm nhiễu bằng bộ lọc Gausianblur (https://phamdinhkhanh.github.io/2020/01/06/ImagePreprocessing.html#222-l%C3%A0m-m%E1%BB%9D-%E1%BA%A3nh-image-blurring), rotate ảnh, crop, zoom ảnh,.... Nếu sử dụng các hàm mặc định của image preprocessing trong ImageGenerator thì sẽ gặp hạn

Top

chế đó là bị giới hạn bởi một số phép biến đổi mà hàm này hỗ trợ. Sử dụng high level framework tiện thì rất tiện nhưng khi muốn can thiệp sâu thì rất khó. Muốn can thiệp được sâu vào bên trong các biến đổi chúng ta phải customize lại một chút ImageGenerator.

Cách thức customize như thế nào. Mình sẽ giới thiệu với các bạn qua chương này.

Đầu tiên chúng ta sẽ download bộ dữ liệu `Dog & Cat` đã được thu nhỏ số lượng ảnh về.

```
1     !git clone https://github.com/ardamavi/Dog-Cat-Classifier.git
```

```
Cloning into 'Dog-Cat-Classifier'... remote: Enumerating objects: 1654, done.
remote: Total 1654 (delta 0), reused 0 (delta 0), pack-reused 1654 Receiving
objects: 100% (1654/1654), 34.83 MiB | 16.60 MiB/s, done. Resolving deltas: 100%
(147/147), done. Checking out files: 100% (1672/1672), done.
```

Tiếp theo ta sẽ khởi tạo một DataGenerator cho bộ dữ liệu ảnh kế thừa class Sequence của keras. Mình sẽ giải thích các phương thức trong DataGenerator này bên dưới.

Top

```
import numpy as np
from tensorflow.keras.utils import Sequence, to_categorical
import cv2

class DataGenerator(Sequence):
    'Generates data for Keras'
    def __init__(self,
                 all_filenames,
                 labels,
                 batch_size,
                 index2class,
                 input_dim,
                 n_channels,
                 n_classes=2,
                 shuffle=True):
        '''
        all_filenames: list toàn bộ các filename
        labels: nhãn của toàn bộ các file
        batch_size: kích thước của 1 batch
        index2class: index của các class
        input_dim: (width, height) đầu vào của ảnh
        n_channels: số lượng channels của ảnh
        n_classes: số lượng các class
        shuffle: có shuffle dữ liệu sau mỗi epoch hay không?
        '''
        self.all_filenames = all_filenames
        self.labels = labels
        self.batch_size = batch_size
        self.index2class = index2class
        self.input_dim = input_dim
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        '''
        return:
          Trả về số lượng batch/1 epoch
        '''
        return int(np.floor(len(self.all_filenames) / self.batch_size

    def __getitem__(self, index):
        '''
        params:
          index: index của batch
        return:
          X, y cho batch thứ index
        '''
        # Lấy ra indexes của batch thứ index
        indexes = self.indexes[index*self.batch_size:(index+1)*self.b

        # List all_filenames trong một batch
        all_filenames_temp = [self.all_filenames[k] for k in indexes]

        # Khởi tạo data
        X, y = self.__data_generation(all_filenames_temp)
```

Top

```
58
59            return X, y
60
61        def on_epoch_end(self):
62            '''
63            Shuffle dữ liệu khi epochs end hoặc start.
64            '''
65            self.indexes = np.arange(len(self.all_filenames))
66            if self.shuffle == True:
67                np.random.shuffle(self.indexes)
68
69        def __data_generation(self, all_filenames_temp):
70            '''
71            params:
72              all_filenames_temp: list các filenames trong 1 batch
73            return:
74              Trả về giá trị cho một batch.
75            '''
76            X = np.empty((self.batch_size, *self.input_dim, self.n_channe
77            y = np.empty((self.batch_size), dtype=int)
78
79            # Khởi tạo dữ liệu
80            for i, fn in enumerate(all_filenames_temp):
81                # Đọc file từ folder name
82                img = cv2.imread(fn)
83                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
84                img = cv2.resize(img, self.input_dim)
85                label = fn.split('/')[3]
86                label = self.index2class[label]
87
88                X[i,] = img
89
90                # Lưu class
91                y[i] = label
92            return X, y
```

Một DataGenerator sẽ cần xác định kích thước của một batch, số lượt steps huấn luyện.

- Hàm **len**(): Như chúng ta đã biết, `__len__()` là một built in function trong python. Bất kì một object nào của python cũng sẽ có hàm `__len__()`. Đối với Datagenerator thì chúng ta sẽ qui định

$$\text{len} = \frac{\#\text{ Obs}}{\text{batch size}}$$

Đây chính là số lượng step trong một epoch.

- Hàm `__getitem__()`: Trong quá trình huấn luyện chúng ta cần phải access vào từng batch trong bộ dữ liệu. Hàm `__getitem__()` sẽ khởi tạo batch theo thứ tự của batch được truyền vào hàm.
- Hàm `on_epoch_end()`: Đây là hàm được tự động run mỗi khi một epoch huấn luyện bắt đầu và kết thúc. Tại hàm này chúng ta sẽ xác định các hành động khi bắt đầu hoặc kết thúc một epoch như có shuffle dữ liệu hay không. Điều chỉnh lại tỷ lệ các class tước khi fit vào model,….

Top

- Hàm `__data_generation()` : Hàm này sẽ được gọi trong `__getitem__()` . `__data_generation()` sẽ trực tiếp biến đổi dữ liệu và quyết định các kết quả dữ liệu trả về cho người dùng. Tại hàm này ta có thể thực hiện các phép preprocessing image.

```
import cv2
import glob2

dict_labels = {
    'dog': 0,
    'cat': 1
}

root_folder = 'Dog-Cat-Classifier/Data/Train_Data/*/*'
fns = glob2.glob(root_folder)
print(len(fns))

image_generator = DataGenerator(
    all_filenames = fns,
    labels = None,
    batch_size = 32,
    index2class = dict_labels,
    input_dim = (224, 224),
    n_channels = 3,
    n_classes = 2,
    shuffle = True
)
```

```
X, y = image_generator.__getitem__(1)

print(X.shape)
print(y.shape)
```

```
(32, 224, 224, 3)
(32,)
```

Như vậy ta có thể thấy, tại mỗi lượt huấn luyện model lấy ra một batch có kích thước là 32. Mặc dù ảnh của chúng ta có kích thước khác nhau nhưng đã được resize về chung một kích thước là `width x height = 224 x 224` .

Chúng ta sẽ thử nghiệm huấn luyện model với generator. Đầu tiên là khởi tạo model.

```
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam

base_extractor = MobileNet(input_shape = (224, 224, 3), include_top =
flat = Flatten()
den = Dense(1, activation='sigmoid')
model = Sequential([base_extractor,
                    flat,
                    den])
model.summary()
```

Top

```
Downloading data from https://storage.googleapis.com/tensorflow/keras
17227776/17225924 [==============================] - 0s 0us/step
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenet_1.00_224 (Model)   (None, 7, 7, 1024)        3228864
_________________________________________________________________
flatten_3 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 50177
=================================================================
Total params: 3,279,041
Trainable params: 3,257,153
Non-trainable params: 21,888
_________________________________________________________________
```

Tiếp theo để huấn luyện model chúng ta chỉ cần thay generator vào vị trí của train data trong hàm `fit()`.

```
model.compile(Adam(), loss='binary_crossentropy', metrics = ['accuracy
model.fit(image_generator,
          epochs = 5)
```

```
Epoch 1/5
43/43 [==============================] - 247s 6s/step - loss: 1.1607
Epoch 2/5
43/43 [==============================] - 248s 6s/step - loss: 0.5528
Epoch 3/5
43/43 [==============================] - 244s 6s/step - loss: 0.2020
Epoch 4/5
43/43 [==============================] - 248s 6s/step - loss: 0.2046
Epoch 5/5
43/43 [==============================] - 245s 6s/step - loss: 0.0494



<tensorflow.python.keras.callbacks.History at 0x7fd031252fd0>
```

Chỉ với khoảng 5 epochs nhưng kết quả đã đạt 98.4% độ chính xác. Đây là một kết quả khá ấn tượng.

4. Tổng kết

Như vậy qua bài viết này tôi đã giới thiệu tới các bạn các phương pháp chính để khởi tạo một Dataset trong tensorflow, ưu nhược điểm và trường hợp sử dụng của từng phương pháp.

Khi nắm vững được kiến thức này, các bạn sẽ không còn phải lo lắng nếu phải đối mặt với những bộ dữ liệu rất lớn mà không biết cách truyền vào mô hình huấn luyện.

Top

Chúc các bạn thành công với những mô hình sắp tới. Cuối cùng không thể thiếu là các tài liệu mà tôi đã tham khảo để viết bài này.

5. Tài liệu tham khảo

1. tensorflow data - tensorflow (https://www.tensorflow.org/guide/data)
2. tensorflow ImageDataGenerator - tensorflow (https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator)
3. how to generate data on the fly - standford.edu (https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly)
4. generator python - wiki (https://wiki.python.org/moin/Generators)

Top