




PhillipChin

Keras Models as Datasets test


6
voters

last run 10 months ago · Python notebook · 816 views
using data from [multiple data sources](#) ·  Public

[Notebook](#) [Code](#) [Data \(2\)](#) [Comments \(0\)](#) [Log](#) [Versions \(5\)](#) [Forks \(5\)](#) [Fork Notebook](#)

Tags

multiple data sources

Notebook

Keras is installed on the Kaggle Kernels but if you want to use pretrained Imagenet models, Keras will try to download some additional files like the pretrained weights. The Kernel won't let you go download files off the Internet though. To get around this problem, I've uploaded the model files for VGG16 and VGG19 as a Data Source. Data Sources have file size limits (<500 megs). This mainly affects the full model files. To get around this, I've split up the hdf5 files into multiple parts. This is just an example of how to load the split model files and how to create your own.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # Linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.
```

```
dogs-vs-cats-redux-kernels-edition
keras-models
```

You're probably going to be using some other data set with the Keras models one so just remember that the data will be in additional subdirectories under "../input".

In [2]:

```
import h5py
import numpy as np
import json
import os
from random import randint
```

In [3]:

```
from keras import applications
from keras.engine import topology
```

Using TensorFlow backend.

In [4]:

```
from keras.preprocessing import image
from keras.applications.imagenet_utils import preprocess_input
```

Loading a split model

```
In [5]:
def load_split_weights(model, model_path_pattern='model_%d.h5', memb_size=102400000):
    """Loads weights from split hdf5 files.

    Parameters
    -----
    model : keras.models.Model
        Your model.
    model_path_pattern : str
        The path name should have a "%d" wild card in it. For "model_%d.h5", the following
        files will be expected:
        model_0.h5
        model_1.h5
        model_2.h5
        ...
    memb_size : int
        The number of bytes per hdf5 file.
    """
    model_f = h5py.File(model_path_pattern, "r", driver="family", memb_size=memb_size)
    topology.load_weights_from_hdf5_group_by_name(model_f, model.layers)

    return model
```

Create a full VGG19 model with no weights loaded.

```
In [6]:
model = applications.VGG19(include_top=True, weights=None)
```

Load the weights.

```
In [7]:
keras_models_dir = '../input/keras-models'
model_path_pattern = keras_models_dir + "/vgg19_weights_tf_dim_ordering_tf_kernels_%d.h5"
model = load_split_weights(model, model_path_pattern = model_path_pattern)
```

```
In [8]:
def load_img_to_np(img_path, target_size=(224, 224)):
    """Loads an image file into a numpy array for preprocess_image.

    Parameters
    -----
    img_path : str
        Path for image.
    target_size : (int, int)
        Height and width for the image to be resized to.
```

```

Returns
-----
numpy.ndarray (len(shape)=4)

"""
img = image.load_img(img_path, target_size=target_size)

# RGB -> BGR
img_np = np.asarray(img)[...::-1]

# reshape for preprocess_input
return img_np.reshape(1, img_np.shape[0], img_np.shape[1], img_np.shape[2]).copy().astype(np.float32)

```

The original `decode_predictions()` tries to download `imagenet_class_index.json`. I grabbed the code from here and made some modifications to load it from Keras models.

In [9]:

```

def decode_predictions(preds, top=5):
    """Decodes the prediction of an ImageNet model.

    # Arguments
        preds: Numpy tensor encoding a batch of predictions.
        top: integer, how many top-guesses to return.

    # Returns
        A list of lists of top class prediction tuples
        `(class_name, class_description, score)`.
        One list of tuples per sample in batch input.

    # Raises
        ValueError: in case of invalid shape of the `pred` array
                    (must be 2D).

    """
    if len(preds.shape) != 2 or preds.shape[1] != 1000:
        raise ValueError("`decode_predictions` expects 'a batch of predictions '
                        '(i.e. a 2D array of shape (samples, 1000)). '
                        'Found array with shape: ' + str(preds.shape))
    fpath = '../input/keras-models/imagenet_class_index.json'
    CLASS_INDEX = json.load(open(fpath))
    results = []
    for pred in preds:
        top_indices = pred.argsort()[-top:][::-1]
        result = [tuple(CLASS_INDEX[str(i)]) + (pred[i],) for i in top_indices]
        result.sort(key=lambda x: x[2], reverse=True)
        results.append(result)
    return results

```

In [10]:

```

def make_prediction(model, img_np):
    """Make predictions for an image.

    Parameters
    -----

```

```

model : Keras.models.Model
    Your model.
img_np : numpy.ndarray (len(shape)=4)
    Array of images as numpy arrays.

Returns
-----
List
"""

preds = model.predict(preprocess_input(img_np))
return decode_predictions(preds)

```

```

In [11]: kitten_file = '%s/images/kitten.jpg' % keras_models_dir
kitten_img = image.load_img(kitten_file, target_size=(224, 224))
kitten_img

```

Out[11]:



```

In [12]: kitten_np = load_img_to_np(kitten_file)
print(make_predition(model, kitten_np))

```

```

[(['n02123045', 'tabby', 0.4385581), ('n02123394', 'Persian_cat', 0.12586932), ('n02127052', 'lynx', 0.086984731), ('n02124075', 'Egyptian_cat', 0.069369592), ('n03887697', 'paper_towel', 0.051091887)]]

```

```

In [13]: dogs_cats_dir = "../input/dogs-vs-cats-redux-kernels-edition"
img_files = [file for file in os.listdir("%s/train/" % dogs_cats_dir) if file.endswith(".jpg")]
cat_files = ['%s/train/%s' % (dogs_cats_dir, file) for file in img_files if file.startswith("cat")]
dog_files = ['%s/train/%s' % (dogs_cats_dir, file) for file in img_files if file.startswith("dog")]

```

```

In [14]: dog_idx = randint(0, len(dog_files))

dog_file = dog_files[dog_idx]
dog_img = image.load_img(dog_file, target_size=(224, 224))

```

```
dog_img
```

Out[14]:



In [15]:

```
dog_np = load_img_to_np(dog_file)
print(make_prediction(model, dog_np))
```

```
[(['n02111500', 'Great_Pyrenees', 0.48836046), ('n02099712', 'Labrador_retriever', 0.188287
45), ('n02104029', 'kuvasz', 0.15693837), ('n02090622', 'borzoi', 0.066495448), ('n0209960
1', 'golden_retriever', 0.029401585)]]
```

In [16]:

```
cat_idx = randint(0, len(dog_files))
cat_file = cat_files[cat_idx]
cat_img = image.load_img(cat_file, target_size=(224, 224))
cat_img
```

Out[16]:



In [17]:

```
cat_np = load_img_to_np(cat_file)
print(make_prediction(model, cat_np))
```

```
[(['n02106166', 'Border_collie', 0.17368713), ('n02133161', 'American_black_bear', 0.126518
88), ('n02104365', 'schipperke', 0.093967155), ('n02111277', 'Newfoundland', 0.054639738),
('n03887697', 'paper_towel', 0.039618474)]]
```

Splitting hdf5 model files

Here's my function for splitting up hdf5 model files. You can change the memb_size but the number of bytes must match when you call load_split_weights() to load the weights back into your model.

In [18]:

```
def split_h5_file(src_path, dest_path_pattern='model_%d.h5', memb_size=102400000):
    """Takes an hdf5 file and makes a copy of it split into multiple files.

    Parameters
    -----
    src_path : str
        The path of the source hdf5 file.
    dest_path_pattern : str
        The path pattern of the destination hdf5 files. The path pattern should have a "%d"
    wild card in it.
        For "model_%d.h5", the following
        files will be expected:
            model_0.h5
            model_1.h5
            model_2.h5
    memb_size : int
        Max number of bytes for each split file.
    """
    src_f = h5py.File(src_path, 'r+')
    dest_f = h5py.File(dest_path_pattern, driver="family", memb_size=memb_size)

    # copy items
    for (name, _) in src_f.items():
        src_f.copy(name, dest_f)

    # copy attribs
    for (name, value) in src_f.attrs.items():
        dest_f.attrs.create(name, value)

    dest_f.flush()
```

Did you find this Kernel useful?
Show your appreciation with an upvote

6



Comments (0)

Sort by Select...



Click here to enter a comment...