
PyTorch pretrained BigGAN

An op-for-op PyTorch reimplementation of DeepMind's BigGAN model with the pre-trained weights from DeepMind.

Introduction

This repository contains an op-for-op PyTorch reimplementation of DeepMind's BigGAN that was released with the paper Large Scale GAN Training for High Fidelity Natural Image Synthesis by Andrew Brock, Jeff Donahue and Karen Simonyan.

This PyTorch implementation of BigGAN is provided with the pretrained 128x128, 256x256 and 512x512 models by DeepMind. We also provide the scripts used to download and convert these models from the TensorFlow Hub models.

This reimplementation was done from the raw computation graph of the Tensorflow version and behave similarly to the TensorFlow version (variance of the output difference of the order of $1e-5$).

This implementation currently only contains the generator as the weights of the discriminator were not released (although the structure of the discriminator is very similar to the generator so it could be added pretty easily. Tell me if you want to do a PR on that, I would be happy to help.)

Installation

This repo was tested on Python 3.6 and PyTorch 1.0.1

PyTorch pretrained BigGAN can be installed from pip as follows:

```
1 pip install pytorch-pretrained-biggan
```

If you simply want to play with the GAN this should be enough.

If you want to use the conversion scripts and the imagenet utilities, additional requirements are needed, in particular TensorFlow and NLTK. To install all the requirements please use the `full_requirements.txt` file:

```
1 git clone https://github.com/huggingface/pytorch-pretrained-BigGAN.git
2 cd pytorch-pretrained-BigGAN
3 pip install -r full_requirements.txt
```

Models

This repository provide direct and simple access to the pretrained “deep” versions of BigGAN for 128, 256 and 512 pixels resolutions as described in the associated publication. Here are some details on the models:

- [BigGAN-deep-128](#): a 50.4M parameters model generating 128x128 pixels images, the model dump weights 201 MB,
- [BigGAN-deep-256](#): a 55.9M parameters model generating 256x256 pixels images, the model dump weights 224 MB,
- [BigGAN-deep-512](#): a 56.2M parameters model generating 512x512 pixels images, the model dump weights 225 MB.

Please refer to Appendix B of the paper for details on the architectures.

All models comprise pre-computed batch norm statistics for 51 truncation values between 0 and 1 (see Appendix C.1 in the paper for details).

Usage

Here is a quick-start example using [BigGAN](#) with a pre-trained model.

See the doc section below for details on these classes and methods.

```
1 import torch
2 from pytorch_pretrained_biggan import (BigGAN, one_hot_from_names,
3                                     truncated_noise_sample,
4                                     save_as_images,
5                                     display_in_terminal)
6
7 # OPTIONAL: if you want to have more information on what's happening,
8 # activate the logger as follows
9 import logging
10 logging.basicConfig(level=logging.INFO)
11
12 # Load pre-trained model tokenizer (vocabulary)
13 model = BigGAN.from_pretrained('biggan-deep-256')
14
15 # Prepare a input
16 truncation = 0.4
17 class_vector = one_hot_from_names(['soap bubble', 'coffee', 'mushroom'
18                                   ], batch_size=3)
19 noise_vector = truncated_noise_sample(truncation=truncation, batch_size
20                                     =3)
21
22 # All in tensors
```

```
18 noise_vector = torch.from_numpy(noise_vector)
19 class_vector = torch.from_numpy(class_vector)
20
21 # If you have a GPU, put everything on cuda
22 noise_vector = noise_vector.to('cuda')
23 class_vector = class_vector.to('cuda')
24 model.to('cuda')
25
26 # Generate an image
27 with torch.no_grad():
28     output = model(noise_vector, class_vector, truncation)
29
30 # If you have a GPU put back on CPU
31 output = output.to('cpu')
32
33 # If you have a sixtel compatible terminal you can display the images
34 #   in the terminal
35 # (see https://github.com/saitoha/libsixel for details)
36 display_in_terminal(output)
37
38 # Save results as png images
39 save_as_images(output)
```





Doc

Loading DeepMind's pre-trained weights

To load one of DeepMind's pre-trained models, instantiate a `BigGAN` model with `from_pretrained()` as:

```
1 model = BigGAN.from_pretrained(PRE_TRAINED_MODEL_NAME_OR_PATH,  
                                cache_dir=None)
```

where

- `PRE_TRAINED_MODEL_NAME_OR_PATH` is either:
 - the shortcut name of a Google AI's or OpenAI's pre-trained model selected in the list:
 - * `biggan-deep-128`: 12-layer, 768-hidden, 12-heads, 110M parameters
 - * `biggan-deep-256`: 24-layer, 1024-hidden, 16-heads, 340M parameters
 - * `biggan-deep-512`: 12-layer, 768-hidden, 12-heads, 110M parameters
 - a path or url to a pretrained model archive containing:
 - * `config.json`: a configuration file for the model, and
 - * `pytorch_model.bin` a PyTorch dump of a pre-trained instance of `BigGAN` (saved with the usual `torch.save()`).

If `PRE_TRAINED_MODEL_NAME_OR_PATH` is a shortcut name, the pre-trained weights will be downloaded from AWS S3 (see the links here) and stored in a cache folder to avoid future download (the cache folder can be found at `~/ .pytorch_pretrained_biggan/`).

- `cache_dir` can be an optional path to a specific directory to download and cache the pre-trained model weights.

Configuration

`BigGANConfig` is a class to store and load BigGAN configurations. It's defined in `config.py`.

Here are some details on the attributes:

- `output_dim`: output resolution of the GAN (128, 256 or 512) for the pre-trained models,
- `z_dim`: size of the noise vector (128 for the pre-trained models).
- `class_embed_dim`: size of the class embedding vectors (128 for the pre-trained models).
- `channel_width`: size of each channel (128 for the pre-trained models).
- `num_classes`: number of classes in the training dataset, like imagenet (1000 for the pre-trained models).
- `layers`: A list of layers definition. Each definition for a layer is a triple of [up-sample in the layer? (bool), number of input channels (int), number of output channels (int)]
- `attention_layer_position`: Position of the self-attention layer in the layer hierarchy (8 for the pre-trained models).
- `eps`: epsilon value to use for spectral and batch normalization layers (1e-4 for the pre-trained models).
- `n_stats`: number of pre-computed statistics for the batch normalization layers associated to various truncation values between 0 and 1 (51 for the pre-trained models).

Model

`BigGAN` is a PyTorch model (`torch.nn.Module`) of BigGAN defined in `model.py`. This model comprises the class embeddings (a linear layer) and the generator with a series of convolutions and conditional batch norms. The discriminator is currently not implemented since pre-trained weights have not been released for it.

The inputs and output are **identical to the TensorFlow model inputs and outputs**.

We detail them here.

`BigGAN` takes as *inputs*:

-
- `z`: a `torch.FloatTensor` of shape `[batch_size, config.z_dim]` with noise sampled from a truncated normal distribution, and
 - `class_label`: an optional `torch.LongTensor` of shape `[batch_size, sequence_length]` with the token types indices selected in `[0, 1]`. Type 0 corresponds to a `sentence A` and type 1 corresponds to a `sentence B` token (see BERT paper for more details).
 - `truncation`: a float between 0 (not comprised) and 1. The truncation of the truncated normal used for creating the noise vector. This truncation value is used to select between a set of pre-computed statistics (means and variances) for the batch norm layers.

`BigGAN` outputs an array of shape `[batch_size, 3, resolution, resolution]` where resolution is 128, 256 or 512 depending of the model:

Utilities: Images, Noise, Imagenet classes

We provide a few utility method to use the model. They are defined in `utils.py`.

Here are some details on these methods:

- `truncated_noise_sample(batch_size=1, dim_z=128, truncation=1., seed=None)`:

Create a truncated noise vector.

- Params:
 - * `batch_size`: batch size.
 - * `dim_z`: dimension of `z`
 - * `truncation`: truncation value to use
 - * `seed`: seed for the random generator
- Output: array of shape `(batch_size, dim_z)`

- `convert_to_images(obj)`:

Convert an output tensor from BigGAN in a list of images.

- Params:
 - * `obj`: tensor or numpy array of shape `(batch_size, channels, height, width)`
- Output:
 - * list of Pillow Images of size `(height, width)`

- `save_as_images(obj, file_name='output')`:

Convert and save an output tensor from BigGAN in a list of saved images.

– Params:

- * obj: tensor or numpy array of shape (batch_size, channels, height, width)
- * file_name: path and beginning of filename to save. Images will be saved as `file_name_{image_number}.png`

- `display_in_terminal(obj)`:

Convert and display an output tensor from BigGAN in the terminal. This function uses `libsixel` and will only work in a `libsixel`-compatible terminal. Please refer to <https://github.com/saitoha/libsixel> for more details.

– Params:

- * obj: tensor or numpy array of shape (batch_size, channels, height, width)
- * file_name: path and beginning of filename to save. Images will be saved as `file_name_{image_number}.png`

- `one_hot_from_int(int_or_list, batch_size=1)`:

Create a one-hot vector from a class index or a list of class indices.

– Params:

- * int_or_list: int, or list of int, of the imagenet classes (between 0 and 999)
- * batch_size: batch size.
 - If int_or_list is an int create a batch of identical classes.
 - If int_or_list is a list, we should have `len(int_or_list) == batch_size`

– Output:

- * array of shape (batch_size, 1000)

- `one_hot_from_names(class_name, batch_size=1)`:

Create a one-hot vector from the name of an imagenet class ('tennis ball', 'daisy', ...). We use NLTK's wordnet search to try to find the relevant synset of ImageNet and take the first one. If we can't find it directly, we look at the hyponyms and hypernyms of the class name.

– Params:

- * class_name: string containing the name of an imagenet object.

– Output:

- * array of shape (batch_size, 1000)

Download and conversion scripts

Scripts to download and convert the TensorFlow models from TensorFlow Hub are provided in `./scripts`.

The scripts can be used directly as:

```
1 ./scripts/download_tf_hub_models.sh
2 ./scripts/convert_tf_hub_models.sh
```