In this lab we will experiment with some trained GAN models. First we will download one trained on face data

```
import torch
use_gpu = True if torch.cuda.is_available() else False

# trained on high-quality celebrity faces "celebA" dataset
# this model outputs 512 x 512 pixel images
model = torch.hub.load('facebookresearch/pytorch_GAN_zoo:hub',
                       'PGAN', model_name='celebAHQ-512',
                       pretrained=True, useGPU=use_gpu)
```

```
Using cache found in /root/.cache/torch/hub/facebookresearch_pytorch_GAN_zoo_hub
Average network found !
```

```
print(model.netG)
```

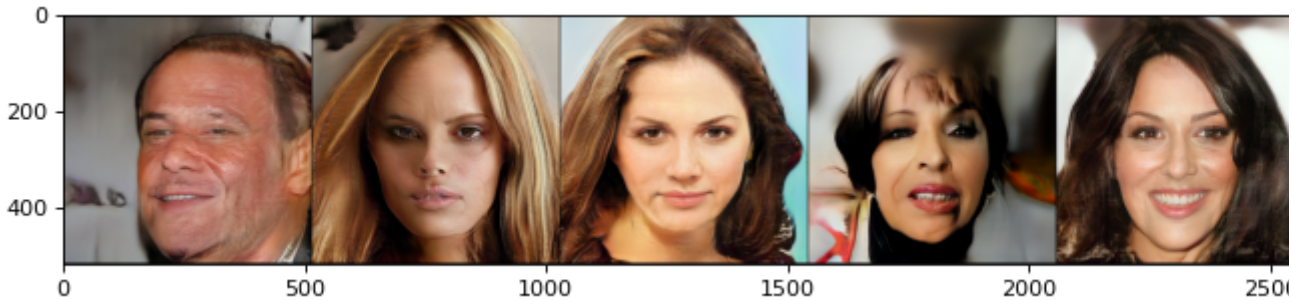
```
              (module): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1
            )
          )
          (5): ModuleList(
            (0): EqualizedConv2d(
              (module): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
            )
            (1): EqualizedConv2d(
              (module): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
            )
          )
          (6): ModuleList(
            (0): EqualizedConv2d(
              (module): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
            )
            (1): EqualizedConv2d(
              (module): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
            )
          )
        )
        (toRGBLayers): ModuleList(
          (0): EqualizedConv2d(
            (module): Conv2d(512, 3, kernel_size=(1, 1), stride=(1, 1))
          )
          (1): EqualizedConv2d(
            (module): Conv2d(512, 3, kernel_size=(1, 1), stride=(1, 1))
          )
          (2): EqualizedConv2d(
            (module): Conv2d(512, 3, kernel_size=(1, 1), stride=(1, 1))
          )
          (3): EqualizedConv2d(
            (module): Conv2d(512, 3, kernel_size=(1, 1), stride=(1, 1))
          )
          (4): EqualizedConv2d(
            (module): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
          )
          (5): EqualizedConv2d(
            (module): Conv2d(128, 3, kernel_size=(1, 1), stride=(1, 1))
          )
```

```
        )
        (6): EqualizedConv2d(
          (module): Conv2d(64, 3, kernel_size=(1, 1), stride=(1, 1))
        )
        (7): EqualizedConv2d(
          (module): Conv2d(32, 3, kernel_size=(1, 1), stride=(1, 1))
        )
      )
      (formatLayer): EqualizedLinear(
        (module): Linear(in_features=512, out_features=8192, bias=True)
      )
      (groupScale0): ModuleList(
        (0): EqualizedConv2d(
          (module): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
        )
      )
      (leakyRelu): LeakyReLU(negative_slope=0.2)
      (normalizationLayer): NormalizationLayer()
    )
  )
```

We will visualize some randomly generated faces. Run this cell a few times to generate new faces

```
num_images = 6
noise, _ = model.buildNoiseData(num_images)
with torch.no_grad():
    generated_images = model.netG(noise).detach()

# let's plot these images using torchvision and matplotlib
import matplotlib.pyplot as plt
import torchvision

grid = torchvision.utils.make_grid(generated_images.clamp(min=-1, max=1), scale_each=True,
plt.figure(figsize=(12, 6), dpi=80)

plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

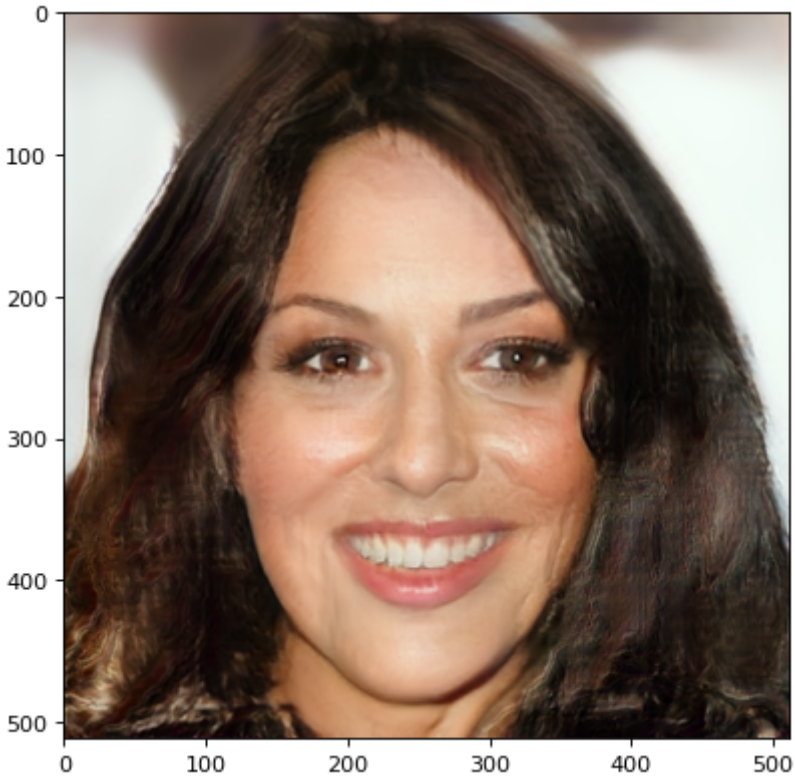
```
<matplotlib.image.AxesImage at 0x7fc204619bd0>
```




```
img1=generated_images[4]
noise1=noise[4]

grid = torchvision.utils.make_grid(img1.clamp(min=-1, max=1), scale_each=True, normalize=T
plt.figure(figsize=(12, 6), dpi=80)
```

```
plt.figure(figsize=(12, 6), dpi=80)

plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

```
<matplotlib.image.AxesImage at 0x7fc2043d5550>
```




```
img2=generated_images[2]
noise2=noise[2]

grid = torchvision.utils.make_grid(img2.clamp(min=-1, max=1), scale_each=True, normalize=T
plt.figure(figsize=(12, 6), dpi=80)

plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

```
<matplotlib.image.AxesImage at 0x7fc204339b90>
```


```
print(type(noise))
print(generated_images.shape)
```

```
<class 'torch.Tensor'>
torch.Size([6, 3, 512, 512])
```

(1) Find two random faces images that you prefer (e.g. the more realistic) and their noise. For example you may by rerunning the cell above a few times to find better faces. Write a function that will interpolate between two randomly generated faces. It will take a noise vector of size 2x512. Let's denote noise_1 and noise_2 the first and 2nd row. As above create 8 intermediate values that interpolate between them.

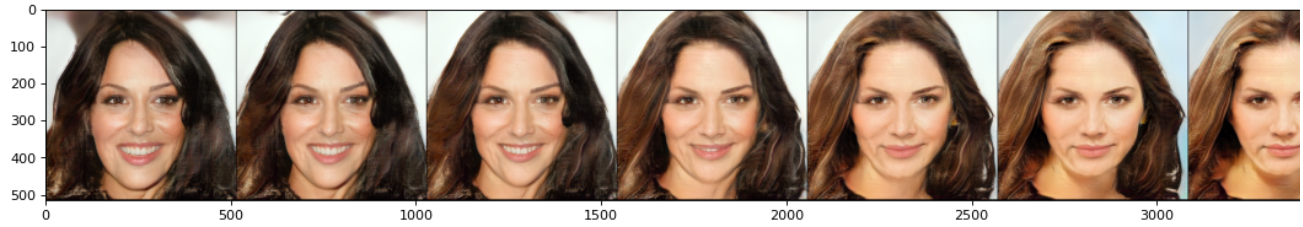
e.g. if we had just one intermediate value we would end up with noise_1, (noise_1+noise_2)/2, noise_2

pass these through the generator (e.g. by putting them in a 8x512 noise tensor) and visualize the interpolation

```
from numpy import asarray
from numpy.random import randn
from numpy.random import randint
from numpy import linspace
def interpolate_points(p1, p2, n_steps=6):
  diff=p2-p1
  vectors = [p1]
  for i in range(1,n_steps+1):
    vectors.append(p1+ diff*(i/n_steps))
  vectors.append(p2)
  return asarray(vectors)
```

```
res=interpolate_points(noise1,noise2)
tensors=torch.zeros([8, 512], dtype=torch.float)
for id in range(len(res)):
  tensors[id]=res[id]
# res=torch.from_numpy(res[0])
len(res[0])
with torch.no_grad():
    generated_images = model.netG(tensors).detach()

grid = torchvision.utils.make_grid(generated_images.clamp(min=-1, max=1), scale_each=True,
plt.figure(figsize=(20, 10), dpi=80)

plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

```
<matplotlib.image.AxesImage at 0x7fc2042a1e50>
```



We will now experiment with the bigGAN model trained on natural images. You can find the implementation and further documentation here https://github.com/huggingface/pytorch-pretrained-BigGAN. Run the cells below to download the model and generate some random images.

```
!pip install pytorch-pretrained-biggan
!pip install libsixel-python
import nltk
nltk.download('wordnet')
```

```
Collecting pytorch-pretrained-biggan
  Downloading https://files.pythonhosted.org/packages/21/05/cd567ad149d8e91080ee767dc
Collecting boto3
  Downloading https://files.pythonhosted.org/packages/fc/79/64c0815cbe8c6abd7fe5525ec
     |████████████████████████████████| 133kB 12.5MB/s
Requirement already satisfied: torch>=0.4.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from p
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Collecting jmespath<1.0.0,>=0.7.1
  Downloading https://files.pythonhosted.org/packages/07/cb/5f001272b6faeb23c1c9e0acc
Collecting botocore<1.21.0,>=1.20.49
  Downloading https://files.pythonhosted.org/packages/68/59/6e28ce58206039ad2592992b7
     |████████████████████████████████| 7.4MB 17.9MB/s
Collecting s3transfer<0.4.0,>=0.3.0
  Downloading https://files.pythonhosted.org/packages/98/14/0b4be62b65c52d6d1c442f24e
     |████████████████████████████████| 81kB 11.4MB/s
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
ERROR: botocore 1.20.49 has requirement urllib3<1.27,>=1.25.4, but you'll have urllib
Installing collected packages: jmespath, botocore, s3transfer, boto3, pytorch-pretrai
Successfully installed boto3-1.17.49 botocore-1.20.49 jmespath-0.10.0 pytorch-pretrai
Collecting libsixel-python
  Downloading https://files.pythonhosted.org/packages/06/0a/cd92860866ee7ef363a78f446
Building wheels for collected packages: libsixel-python
  Building wheel for libsixel-python (setup.py) ... done
  Created wheel for libsixel-python: filename=libsixel_python-0.5.0-cp37-none-any.whl
  Stored in directory: /root/.cache/pip/wheels/83/ba/62/a3c568b79bf35a7a79d44cf0ceb11
Successfully built libsixel-python
Installing collected packages: libsixel-python
```

```
Successfully installed libsixel-python-0.5.0
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
True
```

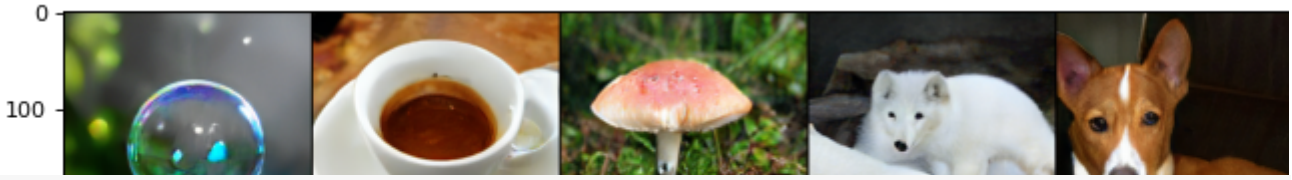
```
import torch
from pytorch_pretrained_biggan import (BigGAN, one_hot_from_names, truncated_noise_sample,
                                       save_as_images, display_in_terminal)
import matplotlib.pyplot as plt
import torchvision


# Load pre-trained model tokenizer (vocabulary)
model = BigGAN.from_pretrained('biggan-deep-256')

# Prepare a input
truncation = 0.4
class_vector = one_hot_from_names(['soap bubble', 'coffee', 'mushroom','fox','dog','lion']
noise_vector = truncated_noise_sample(truncation=truncation, batch_size=6)

# All in tensors
noise_vector = torch.from_numpy(noise_vector)
class_vector = torch.from_numpy(class_vector)

# If you have a GPU, put everything on cuda
noise_vector = noise_vector.to('cuda')
class_vector = class_vector.to('cuda')
model.to('cuda')

# Generate an image
with torch.no_grad():
    output = model(noise_vector, class_vector, truncation)

# If you have a GPU put back on CPU
output = output.to('cpu')
```

```
100%|██████████| 234411737/234411737 [00:06<00:00, 35419967.67B/s]
100%|██████████| 715/715 [00:00<00:00, 310834.10B/s]
```

We visualize the generated images

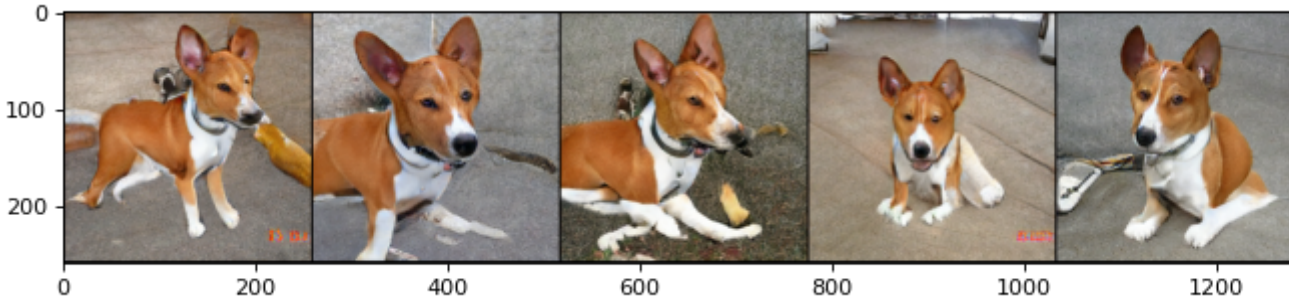
```
output.shape
print(noise_vector.shape)
print(class_vector.shape)
plt.figure(figsize=(12, 6), dpi=80)
grid = torchvision.utils.make_grid(output.clamp(min=-1, max=1), scale_each=True, normalize
plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

```
torch.Size([6, 128])
torch.Size([6, 1000])
<matplotlib.image.AxesImage at 0x7fc2049bee90>
```




```
# Load pre-trained model tokenizer (vocabulary)
model = BigGAN.from_pretrained('biggan-deep-256')

# Prepare a input
truncation = 0.4
class_vector = one_hot_from_names(['dog','dog','dog','dog','dog','dog'], batch_size=6)
noise_vector = truncated_noise_sample(truncation=truncation, batch_size=6)

# All in tensors
noise_vector = torch.from_numpy(noise_vector)
class_vector = torch.from_numpy(class_vector)

# If you have a GPU, put everything on cuda
noise_vector = noise_vector.to('cuda')
class_vector = class_vector.to('cuda')
model.to('cuda')

# Generate an image
with torch.no_grad():
    output = model(noise_vector, class_vector, truncation)

# If you have a GPU put back on CPU
output = output.to('cpu')

output.shape
print(noise_vector.shape)
print(class_vector.shape)
plt.figure(figsize=(12, 6), dpi=80)
grid = torchvision.utils.make_grid(output.clamp(min=-1, max=1), scale_each=True, normalize
plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

```
torch.Size([6, 128])
torch.Size([6, 1000])
<matplotlib.image.AxesImage at 0x7fc20038c650>
```



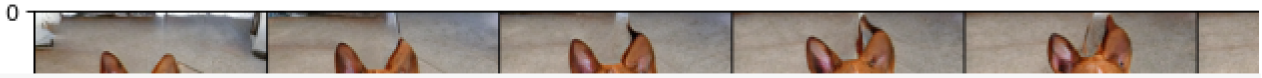
(2) Let's experiment with interpolating between different images in this model as we did in the face images. Note the BigGAN takes both a class vector and a random noise. (a) Sample two random images from the same category such as "dog" and interpolate between them with 8 intermediate steps and using the same class vector (b) Sample two random images from two diff classes (e.g. "dog" and "mushroom") and interpolate between them. For the class conditionin variable you may interpolate between these as well for best results.

Feel free to try other combinations and categories

2.A

```
steps=8
res=interpolate_points(noise_vector[3],noise_vector[0],steps-2)
print(res.shape)
tensors=torch.zeros([steps, 128], dtype=torch.float)
for id in range(len(res)):
  tensors[id]=res[id]
# res=torch.from_numpy(res[0])
len(res[0])
classes=[]
for i in range(steps):
  classes.append('dog')
class_vector = one_hot_from_names(classes, batch_size=steps)

# All in tensors
noise_vector = tensors
class_vector = torch.from_numpy(class_vector)

# If you have a GPU, put everything on cuda
noise_vector = noise_vector.to('cuda')
class_vector = class_vector.to('cuda')
model.to('cuda')

# Generate an image
with torch.no_grad():
    output = model(noise_vector, class_vector, truncation)

# If you have a GPU put back on CPU
output = output.to('cpu')

output.shape
print(noise_vector.shape)
print(class_vector.shape)
plt.figure(figsize=(15, 10), dpi=80)
grid = torchvision.utils.make_grid(output.clamp(min=-1, max=1), scale_each=True, normalize
plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

```
(8,)
torch.Size([8, 128])
torch.Size([8, 1000])
<matplotlib.image.AxesImage at 0x7fc2000b11d0>
```




```
import numpy as np
steps=32
dog_class_vector = one_hot_from_names(['mushroom'], batch_size=1)[0]
lion_class_vector = one_hot_from_names(['lion'], batch_size=1)[0]
class_inter_res=interpolate_points(dog_class_vector,lion_class_vector ,steps-2)
print(class_inter_res)
```

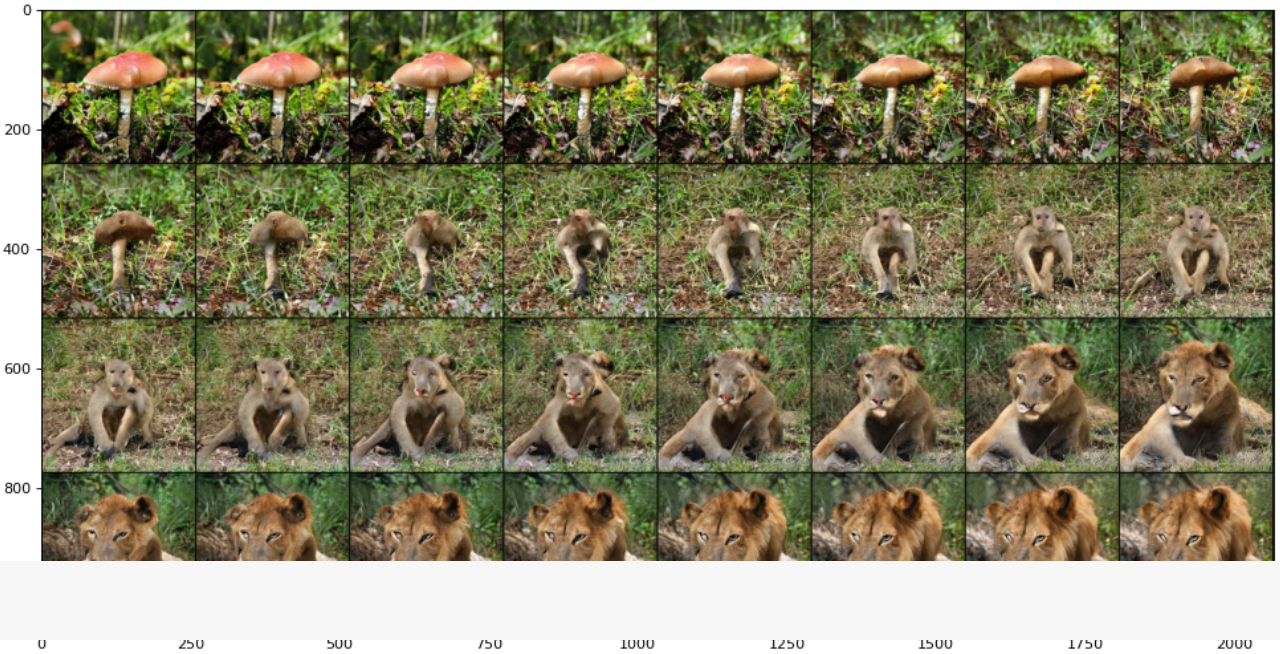
```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
for c in class_inter_res:
  print(np.nonzero(c))
```

```
(array([947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291, 947]),)
(array([291]),)
(array([291]),)
```

```
truncation = 0.4
class_vector = one_hot_from_names(['soap bubble', 'coffee', 'mushroom','fox','cat','lion']
noise_vector = truncated_noise_sample(truncation=truncation, batch_size=6)

res=interpolate_points(noise_vector[2],noise_vector[5],steps-2)
print(res.shape)
tensors=torch.zeros([steps, 128], dtype=torch.float)
for id in range(len(res)):
  tensors[id]=torch.from_numpy(res[id])
# res=torch.from_numpy(res[0])
len(res[0])
# classes=[]
# for i in range(steps):
#   classes.append('lion')

# class_vector = one_hot_from_names(classes, batch_size=steps)

# class_vector = one_hot_from_names(['mushroom','squirrel_monkey','cat','cat','tiger','tig

# All in tensors
noise_vector = tensors
class_vector = torch.from_numpy(class_inter_res)

# If you have a GPU, put everything on cuda
noise_vector = noise_vector.to('cuda')
class_vector = class_vector.to('cuda')
model.to('cuda')

# Generate an image
with torch.no_grad():
    output = model(noise_vector, class_vector, 0.4)

# If you have a GPU put back on CPU
output = output.to('cpu')

output.shape
print(noise_vector.shape)
print(class_vector.shape)
plt.figure(figsize=(15, 10), dpi=80)
grid = torchvision.utils.make_grid(output.clamp(min=-1, max=1), scale_each=True, normalize
plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
```

```
(32, 128)
torch.Size([32, 128])
torch.Size([32, 1000])
<matplotlib.image.AxesImage at 0x7fc293f41b10>
```



✓ 3s completed at 12:54 PM