Open in Colab

T81-558: Applications of Deep Neural Networks

Module 7: Generative Adversarial Networks

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 7 Material

- **Part 7.1: Introduction to GANs for Image and Data Generation** [Video] [Notebook]
- Part 7.2: Train StyleGAN3 with your Own Images [Video] [Notebook]
- Part 7.3: Exploring the StyleGAN Latent Vector [Video] [Notebook]
- Part 7.4: GANs to Enhance Old Photographs Deoldify [Video] [Notebook]
- Part 7.5: GANs for Tabular Synthetic Data Generation [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    from google.colab import drive
    %tensorflow_version 2.x
    drive.mount('/content/drive', force_remount=True)
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Mounted at /content/drive
Note: using Google CoLab
```

Part 7.1: Introduction to GANS for Image and Data Generation

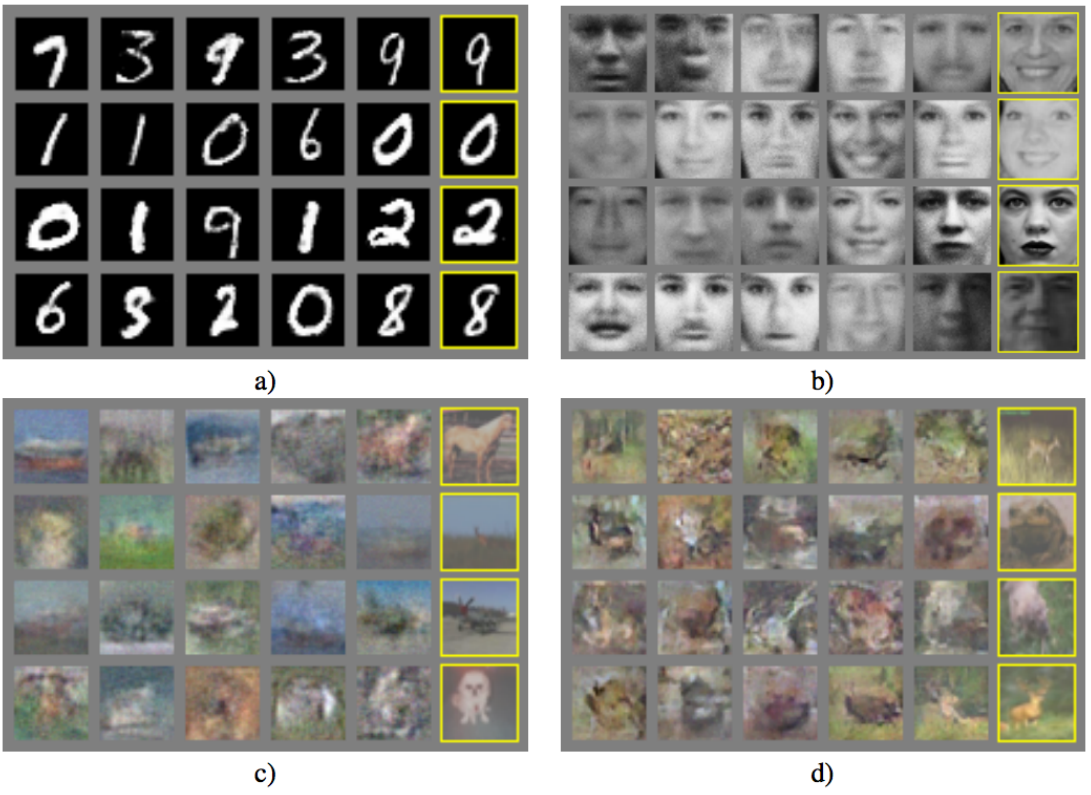
A generative adversarial network (GAN) is a class of machine learning systems invented by Ian Goodfellow in 2014. [Cite:goodfellow2014generative] Two neural networks compete with each other in a game. The GAN training algorithm starts

with a training set and learns to generate new data with the same distributions as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics.

This chapter makes use of the PyTorch framework rather than Keras/TensorFlow. While there are versions of StyleGAN2-ADA that work with TensorFlow 1.0, NVIDIA has switched to PyTorch for StyleGAN. Running this notebook in this notebook in Google CoLab is the most straightforward means of completing this chapter. Because of this, I designed this notebook to run in Google CoLab. It will take some modifications if you wish to run it locally.

This original StyleGAN paper used neural networks to automatically generate images for several previously seen datasets: MINST and CIFAR. However, it also included the Toronto Face Dataset (a private dataset used by some researchers). You can see some of these images in Figure 7.GANS.

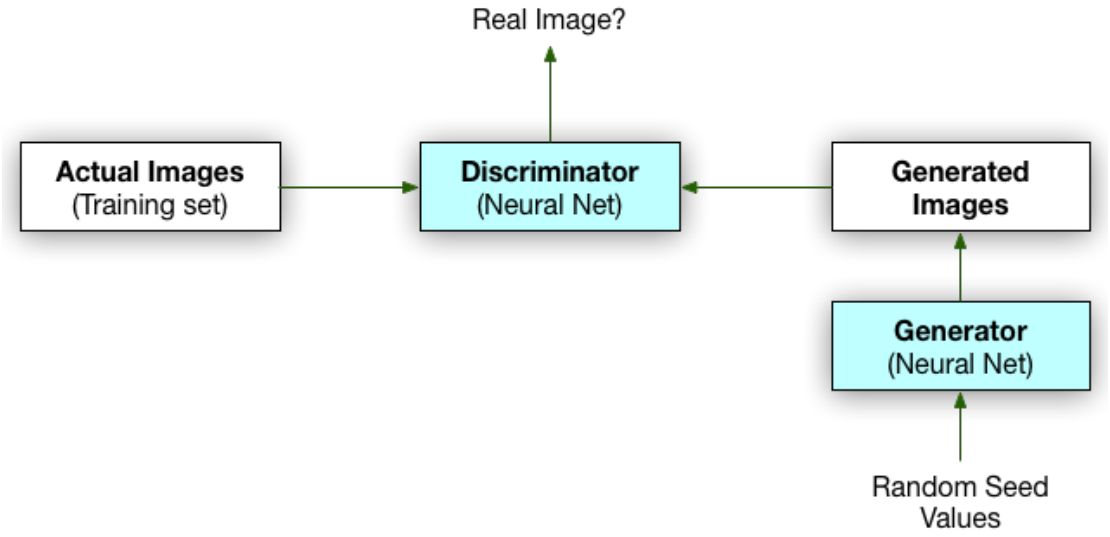
Figure 7.GANS: GAN Generated Images



Only sub-figure D made use of convolutional neural networks. Figures A-C make use of fully connected neural networks. As we will see in this module, the researchers significantly increased the role of convolutional neural networks for GANs.

We call a GAN a generative model because it generates new data. You can see the overall process in Figure 7.GAN-FLOW.

Figure 7.GAN-FLOW: GAN Structure



Face Generation with StyleGAN and Python

GANs have appeared frequently in the media, showcasing their ability to generate highly photorealistic faces. One significant step forward for realistic face generation was the NVIDIA StyleGAN series. NVIDIA introduced the origional StyleGAN in 2018. [Cite:karras2019style] StyleGAN was followed by StyleGAN2 in 2019, which improved the quality of StyleGAN by removing certian artifacts. [Cite:karras2019analyzing] Most recently, in 2020, NVIDIA released StyleGAN2 adaptive discriminator augmentation (ADA), which will be the focus of this module. [Cite:karras2020training] We will see both how to train StyleGAN2 ADA on any arbitray set of images; as well as use pretrained weights provided by NVIDIA. The NVIDIA weights allow us to generate high resolution photorealistic looking faces, such seen in Figure 7.STY-GAN.

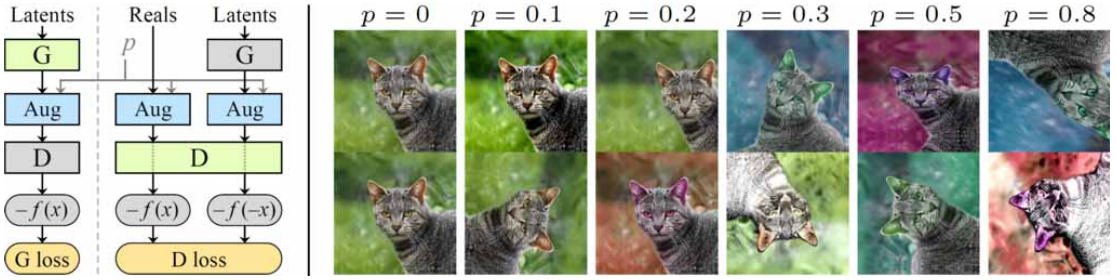
Figure 7.STY-GAN: StyleGAN2 Generated Faces

The above images were generated with StyleGAN2, using Google CoLab. Following the instructions in this section, you will be able to create faces like this of your own. StyleGAN2 images are usually 1,024 x 1,024 in resolution. An example of a full-resolution StyleGAN image can be found here.

The primary advancement introduced by the adaptive discriminator augmentation is that the algorithm augments the training images in real-time. Image augmentation is a common technique in many convolution neural network applications. Augmentation has the effect of increasing the size of the training set. Where StyleGAN2 previously required over 30K images for an effective to develop an effective neural network; now much fewer are needed. I used 2K images to train

the fish generating GAN for this section. Figure 7.STY-GAN-ADA demonstrates the ADA process.

Figure 7.STY-GAN-ADA: StyleGAN2 ADA Training

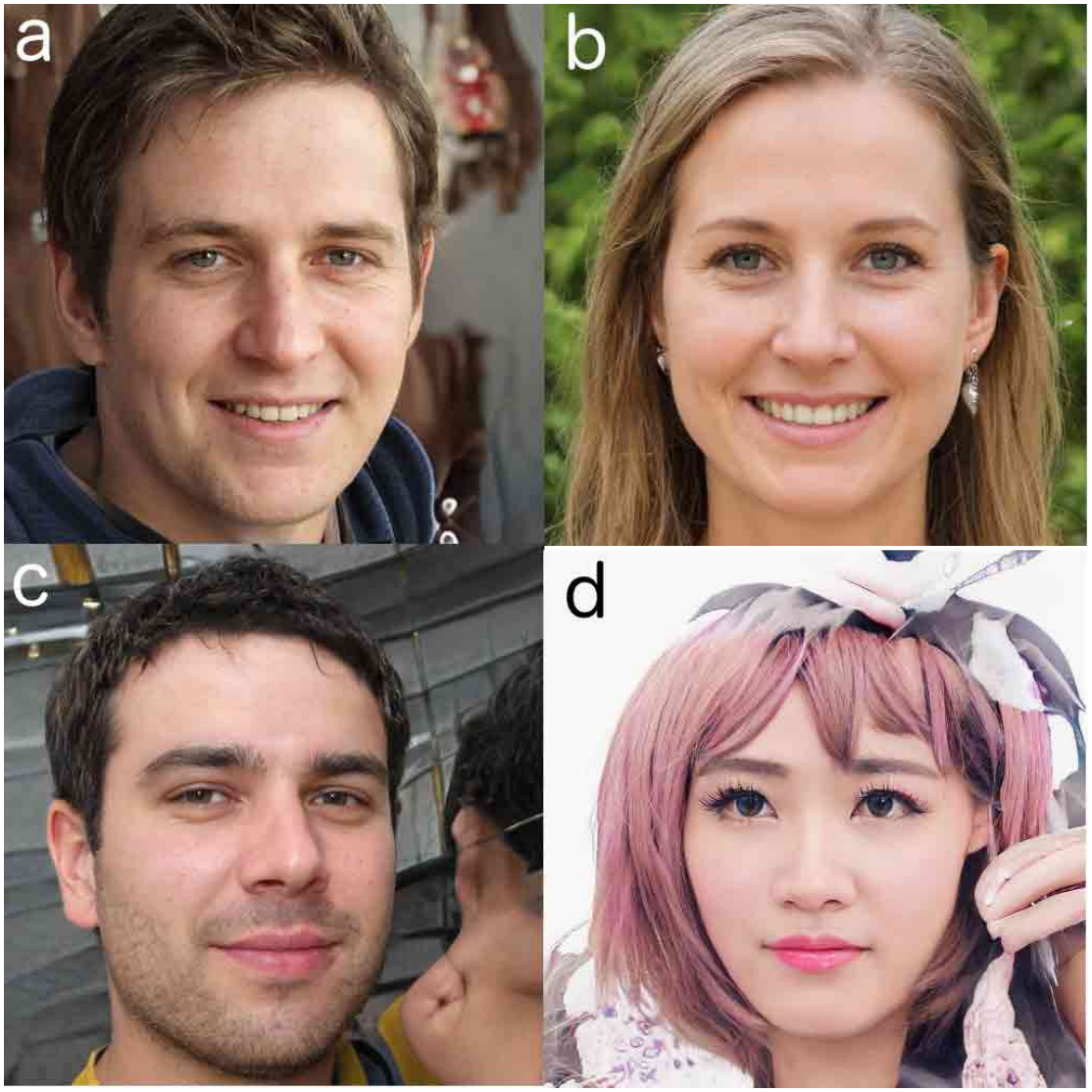


The figure shows the increasing probability of augmentation as p increases. For small image sets, the discriminator will generally memorize the image set unless the training algorithm makes use of augmentation. Once this memorization occurs, the discriminator is no longer providing useful information to the training of the generator.

While the above images look much more realistic than images generated earlier in this course, they are not perfect. Look at Figure 7.STYLEGAN2. There are usually several tell-tail signs that you are looking at a computer-generated image. One of the most obvious is usually the surreal, dream-like backgrounds. The background does not look obviously fake at first glance; however, upon closer inspection, you usually can't quite discern what a GAN-generated background is. Also, look at the image character's left eye. It is slightly unrealistic looking, especially near the eyelashes.

Look at the following GAN face. Can you spot any imperfections?

Figure 7.STYLEGAN2: StyleGAN2 Face



- Image A demonstrates the abstract backgrounds usually associated with a GAN-generated image.
- Image B exhibits issues that earrings often present for GANs. GANs sometimes have problems with symmetry, particularly earrings.
- Image C contains an abstract background and a highly distorted secondary image.
- Image D also contains a highly distorted secondary image that might be a hand.

Several websites allow you to generate GANs of your own without any software.

- This Person Does not Exist
- Which Face is Real

The first site generates high-resolution images of human faces. The second site presents a quiz to see if you can detect the difference between a real and fake human face image.

In this chapter, you will learn to create your own StyleGAN pictures using Python.

Generating High Rez GAN Faces with Google

CoLab

This notebook demonstrates how to run NVidia StyleGAN2 ADA inside a Google CoLab notebook. I suggest you use this to generate GAN faces from a pretrained model. If you try to train your own, you will run into compute limitations of Google CoLab. Make sure to run this code on a GPU instance. GPU is assumed.

First, we clone StyleGAN3 from GitHub.

In [2]:
```
# HIDE OUTPUT
!git clone https://github.com/NVlabs/stylegan3.git
!pip install ninja
```

```
Cloning into 'stylegan3'...
remote: Enumerating objects: 193, done.
remote: Counting objects: 100% (91/91), done.
remote: Compressing objects: 100% (39/39), done.
remote: Total 193 (delta 55), reused 52 (delta 52), pack-reused 102
Receiving objects: 100% (193/193), 4.17 MiB | 11.50 MiB/s, done.
Resolving deltas: 100% (89/89), done.
Collecting ninja
  Downloading ninja-1.10.2.3-py2.py3-none-manylinux_2_5_x86_64.manylinux1_
x86_64.whl (108 kB)
     |████████████████████████████████| 108 kB 4.3 MB/s
Installing collected packages: ninja
Successfully installed ninja-1.10.2.3
```

Verify that StyleGAN has been cloned.

In [3]:
```
!ls /content/stylegan3
```

```
avg_spectra.py   Dockerfile       gen_video.py  metrics      train.py
calc_metrics.py  docs             gui_utils     README.md    visualizer.py
dataset_tool.py  environment.yml  legacy.py     torch_utils  viz
dnnlib           gen_images.py    LICENSE.txt   training
```

Run StyleGan From Command Line

Add the StyleGAN folder to Python so that you can import it. I based this code below on code from NVidia for the original StyleGAN paper. When you use StyleGAN you will generally create a GAN from a seed number. This seed is an integer, such as 6600, that will generate a unique image. The seed generates a latent vector containing 512 floating-point values. The GAN code uses the seed to generate these 512 values. The seed value is easier to represent in code than a 512 value vector; however, while a small change to the latent vector results in a slight change to the image, even a small change to the integer seed value will produce a radically different image.

In [4]:
```
# HIDE OUTPUT
URL = "https://api.ngc.nvidia.com/v2/models/nvidia/research/"\
      "stylegan3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"

!python /content/stylegan3/gen_images.py \
```

```
    --network={URL} \
  --outdir=/content/results --seeds=6600-6625
```

```
Loading networks from "https://api.ngc.nvidia.com/v2/models/nvidia/researc
h/stylegan3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"...
Downloading https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan
3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl ... done
Generating image for seed 6600 (0/26) ...
Setting up PyTorch plugin "bias_act_plugin"... Done.
Setting up PyTorch plugin "filtered_lrelu_plugin"... Done.
Generating image for seed 6601 (1/26) ...
Generating image for seed 6602 (2/26) ...
Generating image for seed 6603 (3/26) ...
Generating image for seed 6604 (4/26) ...
Generating image for seed 6605 (5/26) ...
Generating image for seed 6606 (6/26) ...
Generating image for seed 6607 (7/26) ...
Generating image for seed 6608 (8/26) ...
Generating image for seed 6609 (9/26) ...
Generating image for seed 6610 (10/26) ...
Generating image for seed 6611 (11/26) ...
Generating image for seed 6612 (12/26) ...
Generating image for seed 6613 (13/26) ...
Generating image for seed 6614 (14/26) ...
Generating image for seed 6615 (15/26) ...
Generating image for seed 6616 (16/26) ...
Generating image for seed 6617 (17/26) ...
Generating image for seed 6618 (18/26) ...
Generating image for seed 6619 (19/26) ...
Generating image for seed 6620 (20/26) ...
Generating image for seed 6621 (21/26) ...
Generating image for seed 6622 (22/26) ...
Generating image for seed 6623 (23/26) ...
Generating image for seed 6624 (24/26) ...
Generating image for seed 6625 (25/26) ...
```

We can now display the images created.

In [5]:
```
!ls /content/results
```

```
seed6600.png  seed6606.png  seed6612.png  seed6618.png  seed6624.png
seed6601.png  seed6607.png  seed6613.png  seed6619.png  seed6625.png
seed6602.png  seed6608.png  seed6614.png  seed6620.png
seed6603.png  seed6609.png  seed6615.png  seed6621.png
seed6604.png  seed6610.png  seed6616.png  seed6622.png
seed6605.png  seed6611.png  seed6617.png  seed6623.png
```

Next, copy the images to a folder of your choice on GDrive.

In [6]:
```
!cp /content/results/* \
    /content/drive/My\ Drive/projects/stylegan3
```

Run StyleGAN From Python Code

Add the StyleGAN folder to Python so that you can import it.

In [7]:
```
import sys
```

```
sys.path.insert(0, "/content/stylegan3")
import pickle
import os
import numpy as np
import PIL.Image
from IPython.display import Image
import matplotlib.pyplot as plt
import IPython.display
import torch
import dnnlib
import legacy

def seed2vec(G, seed):
  return np.random.RandomState(seed).randn(1, G.z_dim)

def display_image(image):
  plt.axis('off')
  plt.imshow(image)
  plt.show()

def generate_image(G, z, truncation_psi):
    # Render images for dlatents initialized from random seeds.
    Gs_kwargs = {
        'output_transform': dict(func=tflib.convert_images_to_uint8,
         nchw_to_nhwc=True),
        'randomize_noise': False
    }
    if truncation_psi is not None:
        Gs_kwargs['truncation_psi'] = truncation_psi

    label = np.zeros([1] + G.input_shapes[1][1:])
    # [minibatch, height, width, channel]
    images = G.run(z, label, **G_kwargs)
    return images[0]

def get_label(G, device, class_idx):
  label = torch.zeros([1, G.c_dim], device=device)
  if G.c_dim != 0:
      if class_idx is None:
          ctx.fail("Must specify class label with --class when using "\
            "a conditional network")
      label[:, class_idx] = 1
  else:
      if class_idx is not None:
          print ("warn: --class=lbl ignored when running on "\
            "an unconditional network")
  return label

def generate_image(device, G, z, truncation_psi=1.0, noise_mode='const',
                   class_idx=None):
  z = torch.from_numpy(z).to(device)
  label = get_label(G, device, class_idx)
  img = G(z, label, truncation_psi=truncation_psi, noise_mode=noise_mode
  img = (img.permute(0, 2, 3, 1) * 127.5 + 128).clamp(0, 255).to(\
      torch.uint8)
  return PIL.Image.fromarray(img[0].cpu().numpy(), 'RGB')
```

In [8]:

```
#URL = "https://github.com/jeffheaton/pretrained-gan-fish/releases/"\
#  "download/1.0.0/fish-gan-2020-12-09.pkl"
#URL = "https://github.com/jeffheaton/pretrained-merry-gan-mas/releases/
#  "download/v1/christmas-gan-2020-12-03.pkl"
```

```
URL = "https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/"\
  "versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"

print(f'Loading networks from "{URL}"...')
device = torch.device('cuda')
with dnnlib.util.open_url(URL) as f:
    G = legacy.load_network_pkl(f)['G_ema'].to(device) # type: ignore
```

```
Loading networks from "https://api.ngc.nvidia.com/v2/models/nvidia/researc
h/stylegan3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"...
```

We can now generate images from integer seed codes in Python.

In [9]:

```
# Choose your own starting and ending seed.
SEED_FROM = 1000
SEED_TO = 1003

# Generate the images for the seeds.
for i in range(SEED_FROM, SEED_TO):
  print(f"Seed {i}")
  z = seed2vec(G, i)
  img = generate_image(device, G, z)
  display_image(img)
```

```
Seed 1000
Setting up PyTorch plugin "bias_act_plugin"... Done.
Setting up PyTorch plugin "filtered_lrelu_plugin"... Done.
```

```
Seed 1001
```

```
Seed 1002
```

Examining the Latent Vector

Figure 7.LVEC shows the effects of transforming the latent vector between two images. We accomplish this transformation by slowly moving one 512-value latent vector to another 512 vector. A high-dimension point between two latent vectors will appear similar to both of the two endpoint latent vectors. Images that have similar latent vectors will appear similar to each other.

Figure 7.LVEC: Transforming the Latent Vector

In [10]:
```
def expand_seed(seeds, vector_size):
  result = []

  for seed in seeds:
    rnd = np.random.RandomState(seed)
    result.append( rnd.randn(1, vector_size) )
  return result

#URL = "https://github.com/jeffheaton/pretrained-gan-fish/releases/"\
#  "download/1.0.0/fish-gan-2020-12-09.pkl"
#URL = "https://github.com/jeffheaton/pretrained-merry-gan-mas/releases/
#  "download/v1/christmas-gan-2020-12-03.pkl"
#URL = "https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada/pretrained/ffhq.pk
URL = "https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/"\
  "versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"

print(f'Loading networks from "{URL}"...')
device = torch.device('cuda')
with dnnlib.util.open_url(URL) as f:
    G = legacy.load_network_pkl(f)['G_ema'].to(device) # type: ignore

vector_size = G.z_dim
# range(8192,8300)
seeds = expand_seed( [8192+1,8192+9], vector_size)
#generate_images(Gs, seeds,truncation_psi=0.5)
print(seeds[0].shape)
```

```
Loading networks from "https://api.ngc.nvidia.com/v2/models/nvidia/researc
h/stylegan3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"...
(1, 512)
```

The following code will move between the provided seeds. The constant STEPS specify how many frames there should be between each seed.

In [11]:
```
# HIDE OUTPUT
# Choose your seeds to morph through and the number of steps to
# take to get to each.

SEEDS = [6624,6618,6616] # Better for faces
#SEEDS = [1000,1003,1001] # Better for fish
STEPS = 100

# Remove any prior results
!rm /content/results/*

from tqdm.notebook import tqdm

os.makedirs("./results/", exist_ok=True)

# Generate the images for the video.
idx = 0
for i in range(len(SEEDS)-1):
  v1 = seed2vec(G, SEEDS[i])
  v2 = seed2vec(G, SEEDS[i+1])

  diff = v2 - v1
  step = diff / STEPS
  current = v1.copy()

  for j in tqdm(range(STEPS), desc=f"Seed {SEEDS[i]}"):
```

```
    current = current + step
    img = generate_image(device, G, current)
    img.save(f'./results/frame-{idx}.png')
    idx+=1

# Link the images into a video.
!ffmpeg -r 30 -i /content/results/frame-%d.png -vcodec mpeg4 -y movie.mp
```

```
Seed 6624:   0%|          | 0/100 [00:00<?, ?it/s]
Seed 6618:   0%|          | 0/100 [00:00<?, ?it/s]
ffmpeg version 3.4.8-0ubuntu0.2 Copyright (c) 2000-2020 the FFmpeg develop
ers
  built with gcc 7 (Ubuntu 7.5.0-3ubuntu1~18.04)
  configuration: --prefix=/usr --extra-version=0ubuntu0.2 --toolchain=hard
ened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linu
x-gnu --enable-gpl --disable-stripping --enable-avresample --enable-avisyn
th --enable-gnutls --enable-ladspa --enable-libass --enable-libbluray --en
able-libbs2b --enable-libcaca --enable-libcdio --enable-libflite --enable-
libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --e
nable-libgsm --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg
--enable-libopenmpt --enable-libopus --enable-libpulse --enable-librubberb
and --enable-librsvg --enable-libshine --enable-libsnappy --enable-libsoxr
--enable-libspeex --enable-libssh --enable-libtheora --enable-libtwolame
--enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --
enable-libx265 --enable-libxml2 --enable-libxvid --enable-libzmq --enable-
libzvbi --enable-omx --enable-openal --enable-opengl --enable-sdl2 --enabl
e-libdc1394 --enable-libdrm --enable-libiec61883 --enable-chromaprint --en
able-frei0r --enable-libopencv --enable-libx264 --enable-shared
  libavutil      55. 78.100 / 55. 78.100
  libavcodec     57.107.100 / 57.107.100
  libavformat    57. 83.100 / 57. 83.100
  libavdevice    57. 10.100 / 57. 10.100
  libavfilter     6.107.100 /  6.107.100
  libavresample   3.  7.  0 /  3.  7.  0
  libswscale      4.  8.100 /  4.  8.100
  libswresample   2.  9.100 /  2.  9.100
  libpostproc    54.  7.100 / 54.  7.100
Input #0, image2, from '/content/results/frame-%d.png':
  Duration: 00:00:08.00, start: 0.000000, bitrate: N/A
    Stream #0:0: Video: png, rgb24(pc), 1024x1024, 25 fps, 25 tbr, 25 tbn,
25 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> mpeg4 (native))
Press [q] to stop, [?] for help
Output #0, mp4, to 'movie.mp4':
  Metadata:
    encoder         : Lavf57.83.100
    Stream #0:0: Video: mpeg4 (mp4v / 0x7634706D), yuv420p, 1024x1024, q=
2-31, 200 kb/s, 30 fps, 15360 tbn, 30 tbc
    Metadata:
      encoder         : Lavc57.107.100 mpeg4
    Side data:
      cpb: bitrate max/min/avg: 0/0/200000 buffer size: 0 vbv_delay: -1
frame=  200 fps= 43 q=31.0 Lsize=    1161kB time=00:00:06.63 bitrate=143
3.4kbits/s speed=1.43x
video:1159kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB m
uxing overhead: 0.146784%
```

You can now download the generated video.

In [12]:
```
from google.colab import files
files.download('movie.mp4')
```

Module 7 Assignment

You can find the first assignment here: assignment 7

Open in Colab

T81-558: Applications of Deep Neural Networks

Module 7: Generative Adversarial Networks

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 7 Material

- Part 7.1: Introduction to GANs for Image and Data Generation [Video] [Notebook]
- **Part 7.2: Train StyleGAN3 with your Own Images** [Video] [Notebook]
- Part 7.3: Exploring the StyleGAN Latent Vector [Video] [Notebook]
- Part 7.4: GANs to Enhance Old Photographs Deoldify [Video] [Notebook]
- Part 7.5: GANs for Tabular Synthetic Data Generation [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [ ]:
```
try:
    from google.colab import drive
    drive.mount('/content/drive', force_remount=True)
    COLAB = True
    print("Note: using Google CoLab")
    %tensorflow_version 2.x
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

Part 7.2: Train StyleGAN3 with your Images

Training GANs with StyleGAN is resource-intensive. The NVIDA StyleGAN researchers used computers with eight high-end GPUs for the high-resolution face GANs trained by NVIDIA. The GPU used by NVIDIA is an A100, which has more memory and cores than the P100 or V100 offered by even Colab Pro+. In this part,

we will use StyleGAN2 to train rather than StyleGAN3. You can use networks trained with StyleGAN2 from StyleGAN3; however, StyleGAN3 usually is more effective at training than StyleGAN2.

Unfortunately, StyleGAN3 is compute-intensive and will perform slowly on any GPU that is not the latest Ampere technology. Because Colab does not provide such technology, I am keeping the training guide at the StyleGAN2 level. Switching to StyleGAN3 is relatively easy, as will be pointed out later.

Make sure that you are running this notebook with a GPU runtime. You can train GANs with either Google Colab Free or Pro. I recommend at least the Pro version due to better GPU instances, longer runtimes, and timeouts. Additionally, the capability of Google Colab Pro to run in the background is valuable when training GANs, as you can close your browser or reboot your laptop while training continues.

You will store your training data and trained neural networks to GDRIVE. For GANs, I lay out my GDRIVE like this:

- ./data/gan/images - RAW images I wish to train on.
- ./data/gan/datasets - Actual training datasets that I convert from the raw images.
- ./data/gan/experiments - The output from StyleGAN2, my image previews, and saved network snapshots.

You will mount the drive at the following location.

```
/content/drive/MyDrive/data
```

What Sort of GPU do you Have?

The type of GPU assigned to you by Colab will significantly affect your training time. Some sample times that I achieved with Colab are given here. I've found that Colab Pro generally starts you with a V100, however, if you run scripts non-stop for 24hrs straight for a few days in a row, you will generally be throttled back to a P100.

- 1024x1024 - V100 - 566 sec/tick (CoLab Pro)
- 1024x1024 - P100 - 1819 sec/tick (CoLab Pro)
- 1024x1024 - T4 - 2188 sec/tick (CoLab Free)

By comparison, a 1024x1024 GAN trained with StyleGAN3 on a V100 is 3087 sec/tick.

If you use Google CoLab Pro, generally, it will not disconnect before 24 hours, even if you (but not your script) are inactive. Free CoLab WILL disconnect a perfectly good running script if you do not interact for a few hours. The following describes how to circumvent this issue.

- How to prevent Google Colab from disconnecting?

Set Up New Environment

You will likely need to train for >24 hours. Colab will disconnect you. You must be prepared to restart training when this eventually happens. Training is divided into ticks, every so many ticks (50 by default), your neural network is evaluated, and a snapshot is saved. When CoLab shuts down, all training after the last snapshot is lost. It might seem desirable to snapshot after each tick; however, this snapshotting process itself takes nearly an hour. Learning an optimal snapshot size for your resolution and training data is important.

We will mount GDRIVE so that you will save your snapshots there. You must also place your training images in GDRIVE.

You must also install NVIDIA StyleGAN2 ADA PyTorch. We also need to downgrade PyTorch to a version that supports StyleGAN.

In [ ]:
```
!pip uninstall jax jaxlib -y
!pip install "jax[cuda11_cudnn805]==0.3.10" -f https://storage.googleapi
!pip install torch==1.8.1 torchvision==0.9.1
!git clone https://github.com/NVlabs/stylegan2-ada-pytorch.git
!pip install ninja
```

Find Your Files

The drive is mounted to the following location.

```
/content/drive/MyDrive/data
```

It might be helpful to use an `ls` command to establish the exact path for your images.

In [ ]:
```
!ls /content/drive/MyDrive/data/gan/images
```

Convert Your Images

You must convert your images into a data set form that PyTorch can directly utilize. The following command converts your images and writes the resulting data set to another directory.

In [ ]:
```
CMD = "python /content/stylegan2-ada-pytorch/dataset_tool.py "\
  "--source /content/drive/MyDrive/data/gan/images/circuit "\
  "--dest /content/drive/MyDrive/data/gan/dataset/circuit"

!{CMD}
```

You can use the following command to clear out the newly created dataset. If

something goes wrong and you need to clean up your images and rerun the above command, you should delete your partially completed dataset directory.

In [ ]:
```
#!rm -R /content/drive/MyDrive/data/gan/dataset/circuit/*
```

Clean Up your Images

All images must have the same dimensions and color depth. This code can identify images that have issues.

In [ ]:
```
from os import listdir
from os.path import isfile, join
import os
from PIL import Image
from tqdm.notebook import tqdm

IMAGE_PATH = '/content/drive/MyDrive/data/gan/images/fish'
files = [f for f in listdir(IMAGE_PATH) if isfile(join(IMAGE_PATH, f))]

base_size = None
for file in tqdm(files):
  file2 = os.path.join(IMAGE_PATH,file)
  img = Image.open(file2)
  sz = img.size
  if base_size and sz!=base_size:
    print(f"Inconsistant size: {file2}")
  elif img.mode!='RGB':
    print(f"Inconsistant color format: {file2}")
  else:
    base_size = sz
```

Perform Initial Training

This code performs the initial training. Set SNAP low enough to get a snapshot before Colab forces you to quit.

In [ ]:
```
import os

# Modify these to suit your needs
EXPERIMENTS = "/content/drive/MyDrive/data/gan/experiments"
DATA = "/content/drive/MyDrive/data/gan/dataset/circuit"
SNAP = 10

# Build the command and run it
cmd = f"/usr/bin/python3 /content/stylegan2-ada-pytorch/train.py "\
  f"--snap {SNAP} --outdir {EXPERIMENTS} --data {DATA}"
!{cmd}
```

Resume Training

You can now resume training after you are interrupted by something in the

pervious step.

In [ ]:

```
import os

# Modify these to suit your needs
EXPERIMENTS = "/content/drive/MyDrive/data/gan/experiments"
NETWORK = "network-snapshot-000100.pkl"
RESUME = os.path.join(EXPERIMENTS, \
                "00008-circuit-auto1-resumecustom", NETWORK)
DATA = "/content/drive/MyDrive/data/gan/dataset/circuit"
SNAP = 10

# Build the command and run it
cmd = f"/usr/bin/python3 /content/stylegan2-ada-pytorch/train.py "\
  f"--snap {SNAP} --resume {RESUME} --outdir {EXPERIMENTS} --data {DATA}
!{cmd}
```

Open in Colab

T81-558: Applications of Deep Neural Networks

Module 1: Python Preliminaries

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 7 Material

- Part 7.1: Introduction to GANs for Image and Data Generation [Video] [Notebook]
- Part 7.2: Train StyleGAN3 with your Own Images [Video] [Notebook]
- **Part 7.3: Exploring the StyleGAN Latent Vector** [Video] [Notebook]
- Part 7.4: GANs to Enhance Old Photographs Deoldify [Video] [Notebook]
- Part 7.5: GANs for Tabular Synthetic Data Generation [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

Part 7.3: Exploring the StyleGAN Latent Vector

StyleGAN seeds, such as 3000, are only random number seeds used to generate much longer 512-length latent vectors, which create the GAN image. If you make a small change to the seed, for example, change 3000 to 3001, StyleGAN will create an entirely different picture. However, if you make a small change to a few latent vector values, the image will only change slightly. In this part, we will see how we

can fine-tune the latent vector to control, to some degree, the resulting GAN image appearance.

Installing Needed Software

We begin by installing StyleGAN.

In [2]:
```
# HIDE OUTPUT
!git clone https://github.com/NVlabs/stylegan3.git
!pip install ninja
```

```
Cloning into 'stylegan3'...
remote: Enumerating objects: 193, done.
remote: Counting objects: 100% (193/193), done.
remote: Compressing objects: 100% (104/104), done.
remote: Total 193 (delta 90), reused 153 (delta 86), pack-reused 0
Receiving objects: 100% (193/193), 4.17 MiB | 5.24 MiB/s, done.
Resolving deltas: 100% (90/90), done.
Collecting ninja
  Downloading ninja-1.10.2.3-py2.py3-none-manylinux_2_5_x86_64.manylinux1_
x86_64.whl (108 kB)
     |████████████████████████████████| 108 kB 13.5 MB/s
Installing collected packages: ninja
Successfully installed ninja-1.10.2.3
```

We will use the same functions introduced in the previous part to generate GAN seeds and images.

In [3]:
```
import sys
sys.path.insert(0, "/content/stylegan3")
import pickle
import os
import numpy as np
import PIL.Image
from IPython.display import Image
import matplotlib.pyplot as plt
import IPython.display
import torch
import dnnlib
import legacy

def seed2vec(G, seed):
  return np.random.RandomState(seed).randn(1, G.z_dim)

def display_image(image):
  plt.axis('off')
  plt.imshow(image)
  plt.show()

def generate_image(G, z, truncation_psi):
    # Render images for dlatents initialized from random seeds.
    Gs_kwargs = {
        'output_transform': dict(func=tflib.convert_images_to_uint8,
        nchw_to_nhwc=True),
        'randomize_noise': False
    }
    if truncation_psi is not None:
        Gs_kwargs['truncation_psi'] = truncation_psi
```

```
    label = np.zeros([1] + G.input_shapes[1][1:])
    # [minibatch, height, width, channel]
    images = G.run(z, label, **G_kwargs)
    return images[0]

def get_label(G, device, class_idx):
  label = torch.zeros([1, G.c_dim], device=device)
  if G.c_dim != 0:
      if class_idx is None:
          ctx.fail('Must specify class label with --class'\
                   'when using a conditional network')
      label[:, class_idx] = 1
  else:
      if class_idx is not None:
          print ('warn: --class=lbl ignored when running '\
            'on an unconditional network')
  return label

def generate_image(device, G, z, truncation_psi=1.0,
                   noise_mode='const', class_idx=None):
  z = torch.from_numpy(z).to(device)
  label = get_label(G, device, class_idx)
  img = G(z, label, truncation_psi=truncation_psi,
          noise_mode=noise_mode)
  img = (img.permute(0, 2, 3, 1) * 127.5 + 128)\
    .clamp(0, 255).to(torch.uint8)
  return PIL.Image.fromarray(img[0].cpu().numpy(), 'RGB')
```

Next, we load the NVIDIA FFHQ (faces) GAN. We could use any StyleGAN pretrained GAN network here.

In [4]:

```
# HIDE CODE

URL = "https://api.ngc.nvidia.com/v2/models/nvidia/research/"\
  "stylegan3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"

print('Loading networks from "%s"...' % URL)
device = torch.device('cuda')
with dnnlib.util.open_url(URL) as fp:
    G = legacy.load_network_pkl(fp)['G_ema']\
      .requires_grad_(False).to(device)
```

```
Loading networks from "https://api.ngc.nvidia.com/v2/models/nvidia/researc
h/stylegan3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl"...
Downloading https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan
3/versions/1/files/stylegan3-r-ffhq-1024x1024.pkl ... done
```

Generate and View GANS from Seeds

We will begin by generating a few seeds to evaluate potential starting points for our fine-tuning. Try out different seeds ranges until you have a seed that looks close to what you wish to fine-tune.

In [5]:

```
# HIDE OUTPUT 1
# Choose your own starting and ending seed.
SEED_FROM = 4020
```

```
SEED_TO = 4023

# Generate the images for the seeds.
for i in range(SEED_FROM, SEED_TO):
  print(f"Seed {i}")
  z = seed2vec(G, i)
  img = generate_image(device, G, z)
  display_image(img)
```

```
Seed 4020
Setting up PyTorch plugin "bias_act_plugin"... Done.
Setting up PyTorch plugin "filtered_lrelu_plugin"... Done.
```

```
Seed 4021
```

```
Seed 4022
```

Fine-tune an Image

If you find a seed you like, you can fine-tune it by directly adjusting the latent vector.

First, choose the seed to fine-tune.

In [6]:
```
START_SEED = 4022

current = seed2vec(G, START_SEED)
```

Next, generate and display the current vector. You will return to this point for each iteration of the finetuning.

In [7]:
```
img = generate_image(device, G, current)

SCALE = 0.5
display_image(img)
```

Choose an explore size; this is the number of different potential images chosen by moving in 10 different directions. Run this code once and then again anytime you wish to change the ten directions you are exploring. You might change the ten directions if you are no longer seeing improvements.

In [8]:
```
EXPLORE_SIZE = 25

explore = []
for i in range(EXPLORE_SIZE):
  explore.append( np.random.rand(1, 512) - 0.5 )
```

Each image displayed from running this code shows a potential direction that we can move in the latent vector. Choose one image that you like and change MOVE_DIRECTION to indicate this decision. Once you rerun the code, the code will give you a new set of potential directions. Continue this process until you have a latent vector that you like.

In [9]:
```
# HIDE OUTPUT 1
# Choose the direction to move.  Choose -1 for the initial iteration.
MOVE_DIRECTION = -1
SCALE = 0.5

if MOVE_DIRECTION >=0:
  current = current + explore[MOVE_DIRECTION]
```

```
for i, mv in enumerate(explore):
  print(f"Direction {i}")
  z = current + mv
  img = generate_image(device, G, z)
  display_image(img)
```

```
Direction 0
```

```
Direction 1
```

```
Direction 2
```

```
Direction 3
```

Direction 4

Direction 5

Direction 6

Direction 7

Direction 8

Direction 9

Direction 10

Direction 11

Direction 12

Direction 13

Direction 14

Direction 15

Direction 16

Direction 17

Direction 18

Direction 19

Direction 20

Direction 21

Direction 22

Direction 23

```
Direction 24
```

Open in Colab

T81-558: Applications of Deep Neural Networks

Module 7: Generative Adversarial Networks

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 7 Material

- Part 7.1: Introduction to GANs for Image and Data Generation [Video] [Notebook]
- Part 7.2: Train StyleGAN3 with your Own Images [Video] [Notebook]
- Part 7.3: Exploring the StyleGAN Latent Vector [Video] [Notebook]
- **Part 7.4: GANs to Enhance Old Photographs Deoldify** [Video] [Notebook]
- Part 7.5: GANs for Tabular Synthetic Data Generation [Video] [Notebook]

Part 7.4: GANS to Enhance Old Photographs Deoldify

For the last two parts of this module, we will examine two applications of GANs. The first application is named deoldify, which uses a PyTorche-based GAN to transform old photographs into more modern-looking images. The complete source code to Deoldify is provided, along with several examples notebooks upon which I based this part.

Install Needed Software

We begin by cloning the deoldify repository.

In [1]:
```
# HIDE OUTPUT
!git clone https://github.com/jantic/DeOldify.git DeOldify
%cd DeOldify
```

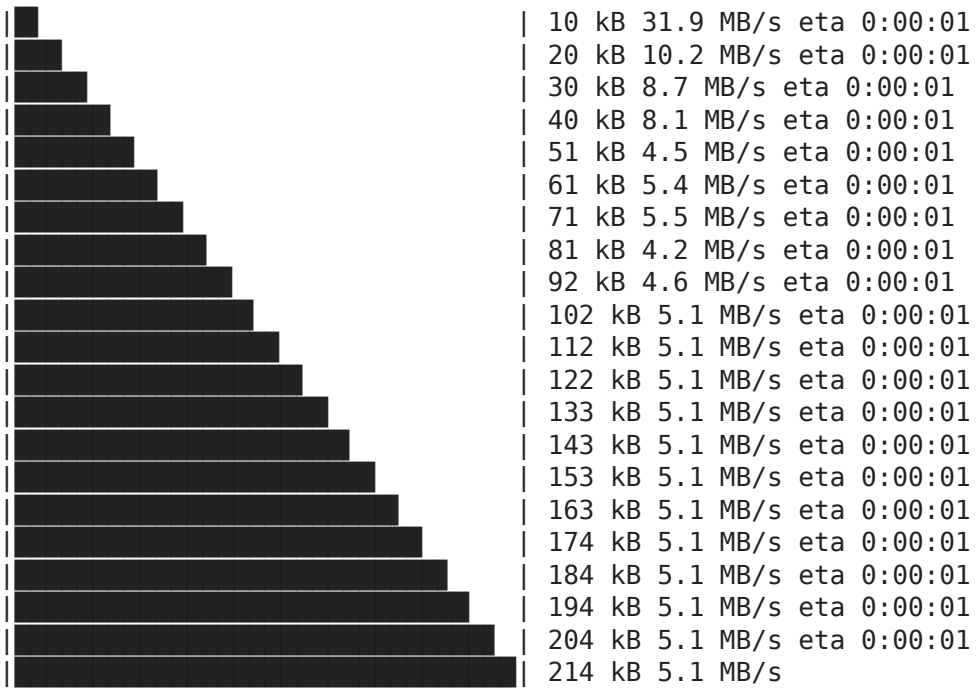
```
Cloning into 'DeOldify'...
remote: Enumerating objects: 2344, done.
remote: Counting objects: 100% (116/116), done.
remote: Compressing objects: 100% (107/107), done.
remote: Total 2344 (delta 57), reused 29 (delta 9), pack-reused 2228
Receiving objects: 100% (2344/2344), 69.46 MiB | 41.02 MiB/s, done.
Resolving deltas: 100% (1064/1064), done.
/content/DeOldify
```

Install any additional Python packages needed.

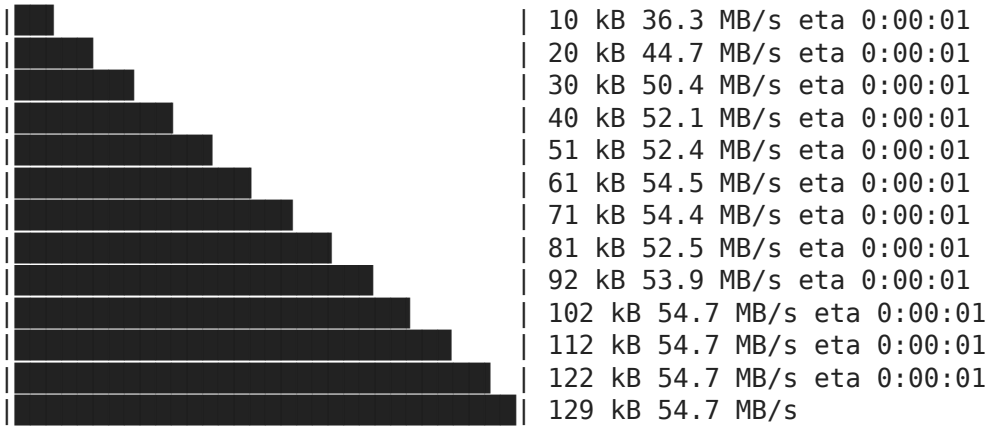
In [2]:

```
# HIDE OUTPUT
!pip install -r colab_requirements.txt
```

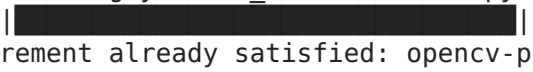
```
Collecting fastai==1.0.51
  Downloading fastai-1.0.51-py3-none-any.whl (214 kB)
```



```
Collecting tensorboardX==1.6
  Downloading tensorboardX-1.6-py2.py3-none-any.whl (129 kB)
```



```
Collecting ffmpeg-python
  Downloading ffmpeg_python-0.2.0-py3-none-any.whl (25 kB)
Collecting youtube-dl>=2019.4.17
  Downloading youtube_dl-2021.12.17-py2.py3-none-any.whl (1.9 MB)
```



```
Requirement already satisfied: opencv-python>=3.3.0.10 in /usr/local/lib/p
ython3.7/dist-packages (from -r colab_requirements.txt (line 5)) (4.1.2.3
0)
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-pac
kages (from -r colab_requirements.txt (line 6)) (7.1.2)
Requirement already satisfied: tornado~=5.1.0 in /usr/local/lib/python3.7/
dist-packages (from -r colab_requirements.txt (line 7)) (5.1.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-pack
ages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (1.4.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-pac
kages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (1.3.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dis
t-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1))
(3.2.2)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-pac
kages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (3.13)
Requirement already satisfied: spacy>=2.0.18 in /usr/local/lib/python3.7/d
ist-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1))
```

```
(2.2.4)
Requirement already satisfied: nvidia-ml-py3 in /usr/local/lib/python3.7/d
ist-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1))
(7.352.0)
Requirement already satisfied: numexpr in /usr/local/lib/python3.7/dist-pa
ckages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (2.8.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.7/
dist-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1))
(4.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-p
ackages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (2.23.0)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dis
t-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (0.1
1.1+cu111)
Requirement already satisfied: torch>=1.0.0 in /usr/local/lib/python3.7/di
st-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (1.1
0.0+cu111)
Collecting typing
  Downloading typing-3.7.4.3.tar.gz (78 kB)
     |████████████████████████████████| 78 kB 6.8 MB/s
Requirement already satisfied: fastprogress>=0.1.19 in /usr/local/lib/pyth
on3.7/dist-packages (from fastai==1.0.51->-r colab_requirements.txt (line
1)) (1.0.2)
Requirement already satisfied: bottleneck in /usr/local/lib/python3.7/dis
t-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1))
(1.3.4)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dis
t-packages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (1.2
1.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-
packages (from fastai==1.0.51->-r colab_requirements.txt (line 1)) (21.3)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packag
es (from tensorboardX==1.6->-r colab_requirements.txt (line 2)) (1.15.0)
Requirement already satisfied: protobuf>=3.2.0 in /usr/local/lib/python
3.7/dist-packages (from tensorboardX==1.6->-r colab_requirements.txt (line
2)) (3.17.3)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/python
3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requiremen
ts.txt (line 1)) (1.1.3)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/lib/p
ython3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requi
rements.txt (line 1)) (1.0.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/pyt
hon3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_require
ments.txt (line 1)) (3.0.6)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/pytho
n3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requireme
nts.txt (line 1)) (2.0.6)
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.7/di
st-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requirements.txt
(line 1)) (7.4.0)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib/pyth
on3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requirem
ents.txt (line 1)) (0.9.0)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/pytho
n3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requireme
nts.txt (line 1)) (1.0.5)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dis
t-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requirements.txt
(line 1)) (57.4.0)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/li
b/python3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_re
quirements.txt (line 1)) (1.0.6)
```

```
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/python
3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requiremen
ts.txt (line 1)) (0.4.1)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/pytho
n3.7/dist-packages (from spacy>=2.0.18->fastai==1.0.51->-r colab_requireme
nts.txt (line 1)) (4.63.0)
Requirement already satisfied: importlib-metadata>=0.20 in /usr/local/lib/
python3.7/dist-packages (from catalogue<1.1.0,>=0.0.7->spacy>=2.0.18->fast
ai==1.0.51->-r colab_requirements.txt (line 1)) (4.11.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=
2.0.18->fastai==1.0.51->-r colab_requirements.txt (line 1)) (3.7.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/
python3.7/dist-packages (from importlib-metadata>=0.20->catalogue<1.1.0,>=
0.0.7->spacy>=2.0.18->fastai==1.0.51->-r colab_requirements.txt (line 1))
(3.10.0.2)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python
3.7/dist-packages (from requests->fastai==1.0.51->-r colab_requirements.tx
t (line 1)) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
3.7/dist-packages (from requests->fastai==1.0.51->-r colab_requirements.tx
t (line 1)) (2021.10.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->fastai==1.0.51->-r
colab_requirements.txt (line 1)) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/di
st-packages (from requests->fastai==1.0.51->-r colab_requirements.txt (lin
e 1)) (2.10)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-pac
kages (from ffmpeg-python->-r colab_requirements.txt (line 3)) (0.16.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/di
st-packages (from matplotlib->fastai==1.0.51->-r colab_requirements.txt (l
ine 1)) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/pyth
on3.7/dist-packages (from matplotlib->fastai==1.0.51->-r colab_requirement
s.txt (line 1)) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib->fastai==1.0.51->-
r colab_requirements.txt (line 1)) (3.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python
3.7/dist-packages (from matplotlib->fastai==1.0.51->-r colab_requirements.
txt (line 1)) (1.4.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/di
st-packages (from pandas->fastai==1.0.51->-r colab_requirements.txt (line
1)) (2018.9)
Building wheels for collected packages: typing
  Building wheel for typing (setup.py) ... done
  Created wheel for typing: filename=typing-3.7.4.3-py3-none-any.whl size=
26325 sha256=59cd22666e09f1de026cc1027fafc482d06516afe68f1f390a5815150309d
329
  Stored in directory: /root/.cache/pip/wheels/35/f3/15/01aa6571f0a72ee6ae
7b827c1491c37a1f72d686fd22b43b0e
Successfully built typing
Installing collected packages: typing, youtube-dl, tensorboardX, ffmpeg-py
thon, fastai
  Attempting uninstall: fastai
    Found existing installation: fastai 1.0.61
    Uninstalling fastai-1.0.61:
      Successfully uninstalled fastai-1.0.61
Successfully installed fastai-1.0.51 ffmpeg-python-0.2.0 tensorboardX-1.6
typing-3.7.4.3 youtube-dl-2021.12.17
```

Install the pretrained weights for deoldify.

In [3]:
```
# HIDE OUTPUT
!mkdir './models/'
CMD = "wget https://data.deepai.org/deoldify/ColorizeArtistic_gen.pth"\
  " -O ./models/ColorizeArtistic_gen.pth"
!{CMD}
```

```
--2022-04-03 18:32:26--  https://data.deepai.org/deoldify/ColorizeArtistic
_gen.pth
Resolving data.deepai.org (data.deepai.org)... 138.201.36.183
Connecting to data.deepai.org (data.deepai.org)|138.201.36.183|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 255144681 (243M) [application/octet-stream]
Saving to: ‘./models/ColorizeArtistic_gen.pth’

./models/ColorizeAr 100%[===================>] 243.32M  29.3MB/s    in 8.9
s

2022-04-03 18:32:36 (27.4 MB/s) - ‘./models/ColorizeArtistic_gen.pth’ save
d [255144681/255144681]
```

The authors of deoldify suggest that you might wish to include a watermark to let others know that AI-enhanced this picture. The following code downloads this standard watermark. The authors describe the watermark as follows:

"This places a watermark icon of a palette at the bottom left corner of the image. The authors intend this practice to be a standard way to convey to others viewing the image that AI colorizes it. We want to help promote this as a standard, especially as the technology continues to improve and the distinction between real and fake becomes harder to discern. This palette watermark practice was initiated and led by the MyHeritage in the MyHeritage In Color feature (which uses a newer version of DeOldify than what you're using here)."

In [4]:
```
# HIDE OUTPUT
CMD = "wget https://media.githubusercontent.com/media/jantic/"\
  "DeOldify/master/resource_images/watermark.png "\
  "-O /content/DeOldify/resource_images/watermark.png"
!{CMD}
```

```
--2022-04-03 18:32:36--  https://media.githubusercontent.com/media/jantic/
DeOldify/master/resource_images/watermark.png
Resolving media.githubusercontent.com (media.githubusercontent.com)... 185
.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to media.githubusercontent.com (media.githubusercontent.com)|18
5.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9210 (9.0K) [image/png]
Saving to: '/content/DeOldify/resource_images/watermark.png'

/content/DeOldify/r 100%[===================>]   8.99K  --.-KB/s    in 0s

2022-04-03 18:32:36 (84.3 MB/s) - '/content/DeOldify/resource_images/water
mark.png' saved [9210/9210]
```

Initialize Torch Device

First, we must initialize a Torch device. If we have a GPU available, we will detect it here. I assume that you will run this code from Google CoLab, with a GPU. It is possible to run this code from a local GPU; however, some modification will be necessary.

In [5]:
```
import sys

#NOTE:  This must be the first call in order to work properly!
from deoldify import device
from deoldify.device_id import DeviceId
#choices:  CPU, GPU0...GPU7
device.set(device=DeviceId.GPU0)

import torch

if not torch.cuda.is_available():
  print('GPU not available.')
else:
  print('Using GPU.')
```

```
Using GPU.
```

We can now call the model. I will enhance an image from my childhood, probably taken in the late 1970s. The picture shows three miniature schnauzers. My childhood dog (Scooby) is on the left, followed by his mom and sister. Overall, a stunning improvement. However, the red in the fire engine riding toy is lost, and the red color of the picnic table where the three dogs were sitting.

In [6]:
```
# HIDE OUTPUT
import fastai
from deoldify.visualize import *
```

```
import warnings
from urllib.parse import urlparse
import os

warnings.filterwarnings("ignore", category=UserWarning,
          message=".*?Your .*? set is empty.*?")

URL = 'https://raw.githubusercontent.com/jeffheaton/'\
  't81_558_deep_learning/master/photos/scooby_family.jpg'

!wget {URL}

a = urlparse(URL)
before_file = os.path.basename(a.path)

RENDER_FACTOR = 35
WATERMARK = False

colorizer = get_image_colorizer(artistic=True)

after_image = colorizer.get_transformed_image(
    before_file, render_factor=RENDER_FACTOR,
    watermarked=WATERMARK)
#print("Starting image:")
```

```
--2022-04-03 18:32:43--  https://raw.githubusercontent.com/jeffheaton/t81_
558_deep_learning/master/photos/scooby_family.jpg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199
.109.133, 185.199.111.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 43719 (43K) [image/jpeg]
Saving to: ‘scooby_family.jpg’


scooby_family.jpg     0%[                    ]       0  --.-KB/s
scooby_family.jpg   100%[===================>]  42.69K  --.-KB/s    in 0.0
1s

2022-04-03 18:32:43 (4.19 MB/s) - ‘scooby_family.jpg’ saved [43719/43719]


Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" t
o /root/.cache/torch/hub/checkpoints/resnet34-b627a593.pth
  0%|          | 0.00/83.3M [00:00<?, ?B/s]
```

You can see the starting image here.

In [7]:

```
from IPython import display
display.Image(URL)
```

Out[7]:

You can see the deoldify version here. Please note that these two images will look similar in a black and white book. To see it in color, visit this link.

In [8]:
```
after_image
```

Out[8]:

Open in Colab

T81-558: Applications of Deep Neural Networks

Module 7: Generative Adversarial Networks

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 7 Material

- Part 7.1: Introduction to GANs for Image and Data Generation [Video] [Notebook]
- Part 7.2: Train StyleGAN3 with your Own Images [Video] [Notebook]
- Part 7.3: Exploring the StyleGAN Latent Vector [Video] [Notebook]
- Part 7.4: GANs to Enhance Old Photographs Deoldify [Video] [Notebook]
- **Part 7.5: GANs for Tabular Synthetic Data Generation** [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow. Running the following code will map your GDrive to `/content/drive`.

In [1]:
```
try:
    from google.colab import drive
    COLAB = True
    print("Note: using Google CoLab")
    %tensorflow_version 2.x
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

Part 7.5: GANs for Tabular Synthetic Data Generation

Typically GANs are used to generate images. However, we can also generate tabular data from a GAN. In this part, we will use the Python tabgan utility to create fake data from tabular data. Specifically, we will use the Auto MPG dataset to train a GAN

to generate fake cars. Cite:ashrapov2020tabular

Installing Tabgan

Pytorch is the foundation of the tabgan neural network utility. The following code installs the needed software to run tabgan in Google Colab.

In [2]:

```
# HIDE OUTPUT
CMD = "wget https://raw.githubusercontent.com/Diyago/"\
  "GAN-for-tabular-data/master/requirements.txt"

!{CMD}
!pip install -r requirements.txt
!pip install tabgan
```

```
--2022-04-03 18:53:04--  https://raw.githubusercontent.com/Diyago/GAN-for-
tabular-data/master/requirements.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199
.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 197 [text/plain]
Saving to: ‘requirements.txt.1’

requirements.txt.1  100%[===================>]     197  --.-KB/s    in 0s

2022-04-03 18:53:04 (8.18 MB/s) - ‘requirements.txt.1’ saved [197/197]

Requirement already satisfied: scipy==1.4.1 in /usr/local/lib/python3.7/di
st-packages (from -r requirements.txt (line 1)) (1.4.1)
Requirement already satisfied: category_encoders==2.1.0 in /usr/local/lib/
python3.7/dist-packages (from -r requirements.txt (line 2)) (2.1.0)
Requirement already satisfied: numpy==1.18.1 in /usr/local/lib/python3.7/d
ist-packages (from -r requirements.txt (line 3)) (1.18.1)
Requirement already satisfied: torch==1.6.0 in /usr/local/lib/python3.7/di
st-packages (from -r requirements.txt (line 4)) (1.6.0)
Requirement already satisfied: pandas==1.2.2 in /usr/local/lib/python3.7/d
ist-packages (from -r requirements.txt (line 5)) (1.2.2)
Requirement already satisfied: lightgbm==2.3.1 in /usr/local/lib/python
3.7/dist-packages (from -r requirements.txt (line 6)) (2.3.1)
Requirement already satisfied: scikit_learn==0.23.2 in /usr/local/lib/pyth
on3.7/dist-packages (from -r requirements.txt (line 7)) (0.23.2)
Requirement already satisfied: torchvision>=0.4.2 in /usr/local/lib/python
3.7/dist-packages (from -r requirements.txt (line 8)) (0.7.0)
Requirement already satisfied: python-dateutil==2.8.1 in /usr/local/lib/py
thon3.7/dist-packages (from -r requirements.txt (line 11)) (2.8.1)
Requirement already satisfied: tqdm==4.61.1 in /usr/local/lib/python3.7/di
st-packages (from -r requirements.txt (line 12)) (4.61.1)
Requirement already satisfied: patsy>=0.4.1 in /usr/local/lib/python3.7/di
st-packages (from category_encoders==2.1.0->-r requirements.txt (line 2))
(0.5.2)
Requirement already satisfied: statsmodels>=0.6.1 in /usr/local/lib/python
3.7/dist-packages (from category_encoders==2.1.0->-r requirements.txt (lin
e 2)) (0.10.2)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-pac
kages (from torch==1.6.0->-r requirements.txt (line 4)) (0.16.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/di
st-packages (from pandas==1.2.2->-r requirements.txt (line 5)) (2018.9)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pyth
on3.7/dist-packages (from scikit_learn==0.23.2->-r requirements.txt (line
7)) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/di
st-packages (from scikit_learn==0.23.2->-r requirements.txt (line 7))
(1.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-p
ackages (from python-dateutil==2.8.1->-r requirements.txt (line 11)) (1.1
5.0)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.7/d
ist-packages (from torchvision>=0.4.2->-r requirements.txt (line 8))
(7.1.2)
Requirement already satisfied: tabgan in /usr/local/lib/python3.7/dist-pac
kages (1.2.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-pack
ages (from tabgan) (1.18.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-pac
kages (from tabgan) (1.2.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packa
```

```
ges (from tabgan) (4.61.1)
Requirement already satisfied: scikit-learn==0.23.2 in /usr/local/lib/pyth
on3.7/dist-packages (from tabgan) (0.23.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python
3.7/dist-packages (from tabgan) (2.8.1)
Requirement already satisfied: category-encoders in /usr/local/lib/python
3.7/dist-packages (from tabgan) (2.1.0)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.7/dist-p
ackages (from tabgan) (2.3.1)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dis
t-packages (from tabgan) (0.7.0)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-pack
ages (from tabgan) (1.6.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/di
st-packages (from scikit-learn==0.23.2->tabgan) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pyth
on3.7/dist-packages (from scikit-learn==0.23.2->tabgan) (3.1.0)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/d
ist-packages (from scikit-learn==0.23.2->tabgan) (1.4.1)
Requirement already satisfied: patsy>=0.4.1 in /usr/local/lib/python3.7/di
st-packages (from category-encoders->tabgan) (0.5.2)
Requirement already satisfied: statsmodels>=0.6.1 in /usr/local/lib/python
3.7/dist-packages (from category-encoders->tabgan) (0.10.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/di
st-packages (from pandas->tabgan) (2018.9)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packag
es (from patsy>=0.4.1->category-encoders->tabgan) (1.15.0)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-pac
kages (from torch->tabgan) (0.16.0)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.7/d
ist-packages (from torchvision->tabgan) (7.1.2)
```

Note, after installing; you may see this message:

- You must restart the runtime in order to use newly installed versions.

If so, click the "restart runtime" button just under the message. Then rerun this notebook, and you should not receive further issues.

Loading the Auto MPG Data and Training a Neural Network

We will begin by generating fake data for the Auto MPG dataset we have previously seen. The tabgan library can generate categorical (textual) and continuous (numeric) data. However, it cannot generate unstructured data, such as the name of the automobile. Car names, such as "AMC Rebel SST" cannot be replicated by the GAN, because every row has a different car name; it is a textual but non-categorical value.

The following code is similar to what we have seen before. We load the AutoMPG dataset. The tabgan library requires Pandas dataframe to train. Because of this, we keep both the Pandas and Numpy values.

In [3]:
```
# HIDE OUTPUT
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
```

```
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import pandas as pd
import io
import os
import requests
import numpy as np
from sklearn import metrics

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv",
    na_values=['NA', '?'])

COLS_USED = ['cylinders', 'displacement', 'horsepower', 'weight',
          'acceleration', 'year', 'origin','mpg']
COLS_TRAIN = ['cylinders', 'displacement', 'horsepower', 'weight',
          'acceleration', 'year', 'origin']

df = df[COLS_USED]

# Handle missing value
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median())


# Split into training and test sets
df_x_train, df_x_test, df_y_train, df_y_test = train_test_split(
    df.drop("mpg", axis=1),
    df["mpg"],
    test_size=0.20,
    #shuffle=False,
    random_state=42,
)

# Create dataframe versions for tabular GAN
df_x_test, df_y_test = df_x_test.reset_index(drop=True), \
  df_y_test.reset_index(drop=True)
df_y_train = pd.DataFrame(df_y_train)
df_y_test = pd.DataFrame(df_y_test)

# Pandas to Numpy
x_train = df_x_train.values
x_test = df_x_test.values
y_train = df_y_train.values
y_test = df_y_test.values

# Build the neural network
model = Sequential()
# Hidden 1
model.add(Dense(50, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(25, activation='relu')) # Hidden 2
model.add(Dense(12, activation='relu')) # Hidden 2
model.add(Dense(1)) # Output
model.compile(loss='mean_squared_error', optimizer='adam')

monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=5, verbose=1, mode='auto',
        restore_best_weights=True)
model.fit(x_train,y_train,validation_data=(x_test,y_test),
        callbacks=[monitor], verbose=2,epochs=1000)
```

```
Epoch 1/1000
10/10 - 1s - loss: 139176.5625 - val_loss: 40689.0703 - 1s/epoch - 148ms/s
tep
Epoch 2/1000
10/10 - 0s - loss: 19372.2285 - val_loss: 3346.7378 - 108ms/epoch - 11ms/s
tep
Epoch 3/1000
10/10 - 0s - loss: 873.7932 - val_loss: 769.1017 - 109ms/epoch - 11ms/step
Epoch 4/1000
10/10 - 0s - loss: 1485.8730 - val_loss: 1525.9556 - 136ms/epoch - 14ms/st
ep
Epoch 5/1000
10/10 - 0s - loss: 866.6918 - val_loss: 195.6039 - 155ms/epoch - 15ms/step
Epoch 6/1000
10/10 - 0s - loss: 142.9136 - val_loss: 177.2400 - 96ms/epoch - 10ms/step
Epoch 7/1000
10/10 - 0s - loss: 193.9373 - val_loss: 142.7312 - 113ms/epoch - 11ms/step
Epoch 8/1000
10/10 - 0s - loss: 116.1862 - val_loss: 89.0451 - 79ms/epoch - 8ms/step
Epoch 9/1000
10/10 - 0s - loss: 106.6868 - val_loss: 95.9191 - 174ms/epoch - 17ms/step
Epoch 10/1000
10/10 - 0s - loss: 104.5894 - val_loss: 87.7888 - 111ms/epoch - 11ms/step
Epoch 11/1000
10/10 - 0s - loss: 100.0589 - val_loss: 88.2749 - 96ms/epoch - 10ms/step
Epoch 12/1000
10/10 - 0s - loss: 99.6257 - val_loss: 87.3040 - 115ms/epoch - 11ms/step
Epoch 13/1000
10/10 - 0s - loss: 99.4177 - val_loss: 86.6027 - 103ms/epoch - 10ms/step
Epoch 14/1000
10/10 - 0s - loss: 98.5141 - val_loss: 86.1316 - 97ms/epoch - 10ms/step
Epoch 15/1000
10/10 - 0s - loss: 98.0732 - val_loss: 85.9742 - 108ms/epoch - 11ms/step
Epoch 16/1000
10/10 - 0s - loss: 97.4856 - val_loss: 85.1393 - 76ms/epoch - 8ms/step
Epoch 17/1000
10/10 - 0s - loss: 97.1630 - val_loss: 84.7279 - 85ms/epoch - 8ms/step
Epoch 18/1000
10/10 - 0s - loss: 96.7964 - val_loss: 84.2021 - 160ms/epoch - 16ms/step
Epoch 19/1000
10/10 - 0s - loss: 96.3048 - val_loss: 83.9871 - 92ms/epoch - 9ms/step
Epoch 20/1000
10/10 - 0s - loss: 95.4462 - val_loss: 83.0952 - 121ms/epoch - 12ms/step
Epoch 21/1000
10/10 - 0s - loss: 94.9444 - val_loss: 82.6417 - 91ms/epoch - 9ms/step
Epoch 22/1000
10/10 - 0s - loss: 94.3362 - val_loss: 82.0355 - 132ms/epoch - 13ms/step
Epoch 23/1000
10/10 - 0s - loss: 93.8082 - val_loss: 81.6199 - 89ms/epoch - 9ms/step
Epoch 24/1000
10/10 - 0s - loss: 93.2513 - val_loss: 81.1020 - 82ms/epoch - 8ms/step
Epoch 25/1000
10/10 - 0s - loss: 92.6264 - val_loss: 80.3359 - 121ms/epoch - 12ms/step
Epoch 26/1000
10/10 - 0s - loss: 92.2328 - val_loss: 79.8444 - 121ms/epoch - 12ms/step
Epoch 27/1000
10/10 - 0s - loss: 91.4926 - val_loss: 79.1404 - 109ms/epoch - 11ms/step
Epoch 28/1000
10/10 - 0s - loss: 90.7999 - val_loss: 78.6531 - 118ms/epoch - 12ms/step
Epoch 29/1000
10/10 - 0s - loss: 90.1882 - val_loss: 78.1106 - 112ms/epoch - 11ms/step
Epoch 30/1000
10/10 - 0s - loss: 89.8745 - val_loss: 77.8685 - 116ms/epoch - 12ms/step
```

```
Epoch 31/1000
10/10 - 0s - loss: 89.4765 - val_loss: 76.8118 - 94ms/epoch - 9ms/step
Epoch 32/1000
10/10 - 0s - loss: 88.4912 - val_loss: 76.5078 - 87ms/epoch - 9ms/step
Epoch 33/1000
10/10 - 0s - loss: 88.0864 - val_loss: 75.5026 - 102ms/epoch - 10ms/step
Epoch 34/1000
10/10 - 0s - loss: 86.9415 - val_loss: 75.0887 - 90ms/epoch - 9ms/step
Epoch 35/1000
10/10 - 0s - loss: 86.7026 - val_loss: 74.8265 - 129ms/epoch - 13ms/step
Epoch 36/1000
10/10 - 0s - loss: 86.5384 - val_loss: 73.6916 - 97ms/epoch - 10ms/step
Epoch 37/1000
10/10 - 0s - loss: 85.6226 - val_loss: 73.5788 - 105ms/epoch - 11ms/step
Epoch 38/1000
10/10 - 0s - loss: 84.6683 - val_loss: 72.4751 - 91ms/epoch - 9ms/step
Epoch 39/1000
10/10 - 0s - loss: 83.8491 - val_loss: 71.7716 - 90ms/epoch - 9ms/step
Epoch 40/1000
10/10 - 0s - loss: 83.1613 - val_loss: 71.0936 - 119ms/epoch - 12ms/step
Epoch 41/1000
10/10 - 0s - loss: 82.5631 - val_loss: 70.6658 - 89ms/epoch - 9ms/step
Epoch 42/1000
10/10 - 0s - loss: 81.8695 - val_loss: 69.8167 - 163ms/epoch - 16ms/step
Epoch 43/1000
10/10 - 0s - loss: 81.1869 - val_loss: 69.2964 - 91ms/epoch - 9ms/step
Epoch 44/1000
10/10 - 0s - loss: 80.8101 - val_loss: 68.9843 - 88ms/epoch - 9ms/step
Epoch 45/1000
10/10 - 0s - loss: 80.6469 - val_loss: 67.9292 - 143ms/epoch - 14ms/step
Epoch 46/1000
10/10 - 0s - loss: 79.4096 - val_loss: 67.6057 - 102ms/epoch - 10ms/step
Epoch 47/1000
10/10 - 0s - loss: 78.5745 - val_loss: 66.5229 - 69ms/epoch - 7ms/step
Epoch 48/1000
10/10 - 0s - loss: 78.8939 - val_loss: 66.6657 - 95ms/epoch - 10ms/step
Epoch 49/1000
10/10 - 0s - loss: 76.9754 - val_loss: 65.2553 - 106ms/epoch - 11ms/step
Epoch 50/1000
10/10 - 0s - loss: 76.6228 - val_loss: 64.6849 - 81ms/epoch - 8ms/step
Epoch 51/1000
10/10 - 0s - loss: 76.0204 - val_loss: 64.7692 - 120ms/epoch - 12ms/step
Epoch 52/1000
10/10 - 0s - loss: 74.8868 - val_loss: 63.3094 - 119ms/epoch - 12ms/step
Epoch 53/1000
10/10 - 0s - loss: 74.4092 - val_loss: 62.8904 - 136ms/epoch - 14ms/step
Epoch 54/1000
10/10 - 0s - loss: 73.6486 - val_loss: 62.5721 - 98ms/epoch - 10ms/step
Epoch 55/1000
10/10 - 0s - loss: 72.7242 - val_loss: 61.3689 - 131ms/epoch - 13ms/step
Epoch 56/1000
10/10 - 0s - loss: 72.2849 - val_loss: 61.0335 - 120ms/epoch - 12ms/step
Epoch 57/1000
10/10 - 0s - loss: 72.1777 - val_loss: 60.2657 - 163ms/epoch - 16ms/step
Epoch 58/1000
10/10 - 0s - loss: 71.7879 - val_loss: 59.4650 - 127ms/epoch - 13ms/step
Epoch 59/1000
10/10 - 0s - loss: 71.8203 - val_loss: 60.6488 - 83ms/epoch - 8ms/step
Epoch 60/1000
10/10 - 0s - loss: 69.9323 - val_loss: 58.5242 - 135ms/epoch - 13ms/step
Epoch 61/1000
10/10 - 0s - loss: 70.4658 - val_loss: 58.6250 - 153ms/epoch - 15ms/step
Epoch 62/1000
```

```
10/10 - 0s - loss: 68.6058 - val_loss: 57.0953 - 202ms/epoch - 20ms/step
Epoch 63/1000
10/10 - 0s - loss: 67.7657 - val_loss: 56.8579 - 110ms/epoch - 11ms/step
Epoch 64/1000
10/10 - 0s - loss: 67.2709 - val_loss: 56.0743 - 134ms/epoch - 13ms/step
Epoch 65/1000
10/10 - 0s - loss: 66.5735 - val_loss: 55.5872 - 115ms/epoch - 12ms/step
Epoch 66/1000
10/10 - 0s - loss: 66.1336 - val_loss: 54.8934 - 84ms/epoch - 8ms/step
Epoch 67/1000
10/10 - 0s - loss: 65.7582 - val_loss: 54.4984 - 142ms/epoch - 14ms/step
Epoch 68/1000
10/10 - 0s - loss: 65.1338 - val_loss: 53.6615 - 151ms/epoch - 15ms/step
Epoch 69/1000
10/10 - 0s - loss: 63.7764 - val_loss: 54.1908 - 107ms/epoch - 11ms/step
Epoch 70/1000
10/10 - 0s - loss: 63.6102 - val_loss: 52.6200 - 88ms/epoch - 9ms/step
Epoch 71/1000
10/10 - 0s - loss: 62.9163 - val_loss: 52.3956 - 92ms/epoch - 9ms/step
Epoch 72/1000
10/10 - 0s - loss: 62.3272 - val_loss: 51.6602 - 99ms/epoch - 10ms/step
Epoch 73/1000
10/10 - 0s - loss: 63.4992 - val_loss: 51.2628 - 161ms/epoch - 16ms/step
Epoch 74/1000
10/10 - 0s - loss: 62.8709 - val_loss: 50.4873 - 111ms/epoch - 11ms/step
Epoch 75/1000
10/10 - 0s - loss: 60.9686 - val_loss: 51.4177 - 82ms/epoch - 8ms/step
Epoch 76/1000
10/10 - 0s - loss: 59.7037 - val_loss: 49.4762 - 89ms/epoch - 9ms/step
Epoch 77/1000
10/10 - 0s - loss: 59.5827 - val_loss: 49.6083 - 121ms/epoch - 12ms/step
Epoch 78/1000
10/10 - 0s - loss: 59.7795 - val_loss: 48.5477 - 115ms/epoch - 11ms/step
Epoch 79/1000
10/10 - 0s - loss: 58.4787 - val_loss: 48.2142 - 113ms/epoch - 11ms/step
Epoch 80/1000
10/10 - 0s - loss: 58.1287 - val_loss: 48.3080 - 93ms/epoch - 9ms/step
Epoch 81/1000
10/10 - 0s - loss: 57.5325 - val_loss: 47.2556 - 77ms/epoch - 8ms/step
Epoch 82/1000
10/10 - 0s - loss: 56.7754 - val_loss: 47.1473 - 109ms/epoch - 11ms/step
Epoch 83/1000
10/10 - 0s - loss: 56.4003 - val_loss: 46.8065 - 101ms/epoch - 10ms/step
Epoch 84/1000
10/10 - 0s - loss: 56.7061 - val_loss: 45.9990 - 185ms/epoch - 18ms/step
Epoch 85/1000
10/10 - 0s - loss: 56.1603 - val_loss: 45.8791 - 197ms/epoch - 20ms/step
Epoch 86/1000
10/10 - 0s - loss: 55.1062 - val_loss: 45.0677 - 126ms/epoch - 13ms/step
Epoch 87/1000
10/10 - 0s - loss: 54.8889 - val_loss: 44.7583 - 120ms/epoch - 12ms/step
Epoch 88/1000
10/10 - 0s - loss: 54.1313 - val_loss: 44.8168 - 82ms/epoch - 8ms/step
Epoch 89/1000
10/10 - 0s - loss: 53.5392 - val_loss: 44.2517 - 76ms/epoch - 8ms/step
Epoch 90/1000
10/10 - 0s - loss: 53.6703 - val_loss: 44.3647 - 86ms/epoch - 9ms/step
Epoch 91/1000
10/10 - 0s - loss: 53.1194 - val_loss: 43.2339 - 164ms/epoch - 16ms/step
Epoch 92/1000
10/10 - 0s - loss: 52.4162 - val_loss: 43.0597 - 115ms/epoch - 12ms/step
Epoch 93/1000
10/10 - 0s - loss: 52.0441 - val_loss: 42.6480 - 83ms/epoch - 8ms/step
```

```
Epoch 94/1000
10/10 - 0s - loss: 51.5989 - val_loss: 42.4377 - 42ms/epoch - 4ms/step
Epoch 95/1000
10/10 - 0s - loss: 51.3697 - val_loss: 41.9103 - 48ms/epoch - 5ms/step
Epoch 96/1000
10/10 - 0s - loss: 51.3203 - val_loss: 41.6717 - 55ms/epoch - 6ms/step
Epoch 97/1000
10/10 - 0s - loss: 51.1632 - val_loss: 41.1382 - 65ms/epoch - 7ms/step
Epoch 98/1000
10/10 - 0s - loss: 50.2788 - val_loss: 40.7239 - 47ms/epoch - 5ms/step
Epoch 99/1000
10/10 - 0s - loss: 49.7021 - val_loss: 40.8461 - 50ms/epoch - 5ms/step
Epoch 100/1000
10/10 - 0s - loss: 50.6249 - val_loss: 40.8916 - 67ms/epoch - 7ms/step
Epoch 101/1000
10/10 - 0s - loss: 49.9470 - val_loss: 40.2612 - 47ms/epoch - 5ms/step
Epoch 102/1000
10/10 - 0s - loss: 49.3327 - val_loss: 39.3642 - 53ms/epoch - 5ms/step
Epoch 103/1000
10/10 - 0s - loss: 49.4299 - val_loss: 39.4563 - 67ms/epoch - 7ms/step
Epoch 104/1000
10/10 - 0s - loss: 48.4410 - val_loss: 39.5185 - 44ms/epoch - 4ms/step
Epoch 105/1000
10/10 - 0s - loss: 47.7434 - val_loss: 39.1029 - 49ms/epoch - 5ms/step
Epoch 106/1000
10/10 - 0s - loss: 47.3096 - val_loss: 38.3037 - 64ms/epoch - 6ms/step
Epoch 107/1000
10/10 - 0s - loss: 47.3403 - val_loss: 38.1661 - 50ms/epoch - 5ms/step
Epoch 108/1000
10/10 - 0s - loss: 47.3158 - val_loss: 39.3938 - 44ms/epoch - 4ms/step
Epoch 109/1000
10/10 - 0s - loss: 47.2465 - val_loss: 37.4724 - 63ms/epoch - 6ms/step
Epoch 110/1000
10/10 - 0s - loss: 46.1793 - val_loss: 38.2548 - 46ms/epoch - 5ms/step
Epoch 111/1000
10/10 - 0s - loss: 45.9742 - val_loss: 37.9052 - 48ms/epoch - 5ms/step
Epoch 112/1000
10/10 - 0s - loss: 46.8534 - val_loss: 36.6737 - 51ms/epoch - 5ms/step
Epoch 113/1000
10/10 - 0s - loss: 45.6568 - val_loss: 37.2436 - 43ms/epoch - 4ms/step
Epoch 114/1000
10/10 - 0s - loss: 46.1722 - val_loss: 37.9826 - 59ms/epoch - 6ms/step
Epoch 115/1000
10/10 - 0s - loss: 45.0864 - val_loss: 36.1506 - 45ms/epoch - 5ms/step
Epoch 116/1000
10/10 - 0s - loss: 44.5590 - val_loss: 36.2634 - 42ms/epoch - 4ms/step
Epoch 117/1000
10/10 - 0s - loss: 44.0101 - val_loss: 36.1932 - 50ms/epoch - 5ms/step
Epoch 118/1000
10/10 - 0s - loss: 44.5253 - val_loss: 36.1185 - 55ms/epoch - 6ms/step
Epoch 119/1000
10/10 - 0s - loss: 43.6802 - val_loss: 35.3576 - 49ms/epoch - 5ms/step
Epoch 120/1000
10/10 - 0s - loss: 43.8521 - val_loss: 35.2081 - 63ms/epoch - 6ms/step
Epoch 121/1000
10/10 - 0s - loss: 42.8944 - val_loss: 35.2362 - 63ms/epoch - 6ms/step
Epoch 122/1000
10/10 - 0s - loss: 43.0618 - val_loss: 34.6546 - 46ms/epoch - 5ms/step
Epoch 123/1000
10/10 - 0s - loss: 42.5577 - val_loss: 34.5727 - 46ms/epoch - 5ms/step
Epoch 124/1000
10/10 - 0s - loss: 42.0112 - val_loss: 35.2444 - 47ms/epoch - 5ms/step
Epoch 125/1000
```

```
10/10 - 0s - loss: 41.5351 - val_loss: 33.8780 - 50ms/epoch - 5ms/step
Epoch 126/1000
10/10 - 0s - loss: 43.1731 - val_loss: 36.7196 - 42ms/epoch - 4ms/step
Epoch 127/1000
10/10 - 0s - loss: 44.9588 - val_loss: 34.4649 - 67ms/epoch - 7ms/step
Epoch 128/1000
10/10 - 0s - loss: 41.6290 - val_loss: 35.5199 - 74ms/epoch - 7ms/step
Epoch 129/1000
10/10 - 0s - loss: 40.7516 - val_loss: 33.2187 - 43ms/epoch - 4ms/step
Epoch 130/1000
10/10 - 0s - loss: 42.1431 - val_loss: 35.9299 - 65ms/epoch - 6ms/step
Epoch 131/1000
10/10 - 0s - loss: 40.5715 - val_loss: 32.7406 - 45ms/epoch - 4ms/step
Epoch 132/1000
10/10 - 0s - loss: 40.3439 - val_loss: 32.6067 - 71ms/epoch - 7ms/step
Epoch 133/1000
10/10 - 0s - loss: 39.9940 - val_loss: 32.7292 - 47ms/epoch - 5ms/step
Epoch 134/1000
10/10 - 0s - loss: 40.0634 - val_loss: 32.1410 - 48ms/epoch - 5ms/step
Epoch 135/1000
10/10 - 0s - loss: 40.4516 - val_loss: 32.3407 - 69ms/epoch - 7ms/step
Epoch 136/1000
10/10 - 0s - loss: 39.1197 - val_loss: 32.0680 - 61ms/epoch - 6ms/step
Epoch 137/1000
10/10 - 0s - loss: 38.6692 - val_loss: 31.8778 - 51ms/epoch - 5ms/step
Epoch 138/1000
10/10 - 0s - loss: 38.7935 - val_loss: 32.9095 - 59ms/epoch - 6ms/step
Epoch 139/1000
10/10 - 0s - loss: 38.6567 - val_loss: 31.1871 - 54ms/epoch - 5ms/step
Epoch 140/1000
10/10 - 0s - loss: 38.1735 - val_loss: 31.1331 - 63ms/epoch - 6ms/step
Epoch 141/1000
10/10 - 0s - loss: 37.6601 - val_loss: 31.2389 - 44ms/epoch - 4ms/step
Epoch 142/1000
10/10 - 0s - loss: 37.5095 - val_loss: 31.1678 - 62ms/epoch - 6ms/step
Epoch 143/1000
10/10 - 0s - loss: 37.3672 - val_loss: 30.4313 - 61ms/epoch - 6ms/step
Epoch 144/1000
10/10 - 0s - loss: 37.9671 - val_loss: 30.2334 - 50ms/epoch - 5ms/step
Epoch 145/1000
10/10 - 0s - loss: 37.7869 - val_loss: 32.6676 - 61ms/epoch - 6ms/step
Epoch 146/1000
10/10 - 0s - loss: 37.3247 - val_loss: 30.0956 - 49ms/epoch - 5ms/step
Epoch 147/1000
10/10 - 0s - loss: 37.0411 - val_loss: 29.7544 - 63ms/epoch - 6ms/step
Epoch 148/1000
10/10 - 0s - loss: 37.8974 - val_loss: 34.1898 - 55ms/epoch - 6ms/step
Epoch 149/1000
10/10 - 0s - loss: 35.6341 - val_loss: 31.0651 - 50ms/epoch - 5ms/step
Epoch 150/1000
10/10 - 0s - loss: 38.9956 - val_loss: 32.1005 - 52ms/epoch - 5ms/step
Epoch 151/1000
10/10 - 0s - loss: 35.7875 - val_loss: 28.9734 - 65ms/epoch - 7ms/step
Epoch 152/1000
10/10 - 0s - loss: 35.7318 - val_loss: 29.1119 - 48ms/epoch - 5ms/step
Epoch 153/1000
10/10 - 0s - loss: 35.2600 - val_loss: 28.6848 - 65ms/epoch - 6ms/step
Epoch 154/1000
10/10 - 0s - loss: 35.9957 - val_loss: 29.1977 - 42ms/epoch - 4ms/step
Epoch 155/1000
10/10 - 0s - loss: 35.7540 - val_loss: 29.7204 - 72ms/epoch - 7ms/step
Epoch 156/1000
10/10 - 0s - loss: 34.8676 - val_loss: 28.1050 - 44ms/epoch - 4ms/step
```

```
Epoch 157/1000
10/10 - 0s - loss: 34.6044 - val_loss: 29.6049 - 49ms/epoch - 5ms/step
Epoch 158/1000
10/10 - 0s - loss: 34.8734 - val_loss: 27.8684 - 49ms/epoch - 5ms/step
Epoch 159/1000
10/10 - 0s - loss: 34.2168 - val_loss: 27.5564 - 61ms/epoch - 6ms/step
Epoch 160/1000
10/10 - 0s - loss: 34.3384 - val_loss: 27.3708 - 64ms/epoch - 6ms/step
Epoch 161/1000
10/10 - 0s - loss: 33.9496 - val_loss: 28.5652 - 47ms/epoch - 5ms/step
Epoch 162/1000
10/10 - 0s - loss: 33.4599 - val_loss: 27.3174 - 58ms/epoch - 6ms/step
Epoch 163/1000
10/10 - 0s - loss: 33.8438 - val_loss: 27.6048 - 45ms/epoch - 5ms/step
Epoch 164/1000
10/10 - 0s - loss: 33.1440 - val_loss: 26.6754 - 71ms/epoch - 7ms/step
Epoch 165/1000
10/10 - 0s - loss: 33.6024 - val_loss: 28.4600 - 45ms/epoch - 4ms/step
Epoch 166/1000
10/10 - 0s - loss: 33.0155 - val_loss: 26.3480 - 75ms/epoch - 8ms/step
Epoch 167/1000
10/10 - 0s - loss: 33.6239 - val_loss: 27.4919 - 43ms/epoch - 4ms/step
Epoch 168/1000
10/10 - 0s - loss: 33.7240 - val_loss: 28.3199 - 51ms/epoch - 5ms/step
Epoch 169/1000
10/10 - 0s - loss: 33.5670 - val_loss: 25.8991 - 62ms/epoch - 6ms/step
Epoch 170/1000
10/10 - 0s - loss: 31.7415 - val_loss: 26.0897 - 62ms/epoch - 6ms/step
Epoch 171/1000
10/10 - 0s - loss: 31.5303 - val_loss: 25.4196 - 49ms/epoch - 5ms/step
Epoch 172/1000
10/10 - 0s - loss: 31.8498 - val_loss: 26.9220 - 50ms/epoch - 5ms/step
Epoch 173/1000
10/10 - 0s - loss: 31.6775 - val_loss: 25.7945 - 62ms/epoch - 6ms/step
Epoch 174/1000
10/10 - 0s - loss: 31.3026 - val_loss: 25.3169 - 53ms/epoch - 5ms/step
Epoch 175/1000
10/10 - 0s - loss: 30.8672 - val_loss: 25.7407 - 69ms/epoch - 7ms/step
Epoch 176/1000
10/10 - 0s - loss: 31.5805 - val_loss: 24.5653 - 72ms/epoch - 7ms/step
Epoch 177/1000
10/10 - 0s - loss: 31.6575 - val_loss: 24.7597 - 53ms/epoch - 5ms/step
Epoch 178/1000
10/10 - 0s - loss: 31.0366 - val_loss: 26.0440 - 61ms/epoch - 6ms/step
Epoch 179/1000
10/10 - 0s - loss: 30.4223 - val_loss: 24.2567 - 65ms/epoch - 7ms/step
Epoch 180/1000
10/10 - 0s - loss: 29.6591 - val_loss: 24.0481 - 62ms/epoch - 6ms/step
Epoch 181/1000
10/10 - 0s - loss: 29.5555 - val_loss: 23.9970 - 65ms/epoch - 7ms/step
Epoch 182/1000
10/10 - 0s - loss: 29.4170 - val_loss: 23.5796 - 58ms/epoch - 6ms/step
Epoch 183/1000
10/10 - 0s - loss: 29.2324 - val_loss: 24.1796 - 52ms/epoch - 5ms/step
Epoch 184/1000
10/10 - 0s - loss: 28.8220 - val_loss: 23.2315 - 46ms/epoch - 5ms/step
Epoch 185/1000
10/10 - 0s - loss: 29.0582 - val_loss: 24.1037 - 61ms/epoch - 6ms/step
Epoch 186/1000
10/10 - 0s - loss: 28.7244 - val_loss: 22.9213 - 64ms/epoch - 6ms/step
Epoch 187/1000
10/10 - 0s - loss: 28.4226 - val_loss: 23.6453 - 52ms/epoch - 5ms/step
Epoch 188/1000
```

```
10/10 - 0s - loss: 30.4988 - val_loss: 22.3928 - 67ms/epoch - 7ms/step
Epoch 189/1000
10/10 - 0s - loss: 29.2073 - val_loss: 22.8565 - 46ms/epoch - 5ms/step
Epoch 190/1000
10/10 - 0s - loss: 27.8428 - val_loss: 24.4894 - 61ms/epoch - 6ms/step
Epoch 191/1000
10/10 - 0s - loss: 28.7934 - val_loss: 21.9677 - 46ms/epoch - 5ms/step
Epoch 192/1000
10/10 - 0s - loss: 29.0026 - val_loss: 23.3321 - 68ms/epoch - 7ms/step
Epoch 193/1000
10/10 - 0s - loss: 28.7747 - val_loss: 21.6816 - 60ms/epoch - 6ms/step
Epoch 194/1000
10/10 - 0s - loss: 27.4620 - val_loss: 21.3625 - 48ms/epoch - 5ms/step
Epoch 195/1000
10/10 - 0s - loss: 26.9269 - val_loss: 21.5949 - 46ms/epoch - 5ms/step
Epoch 196/1000
10/10 - 0s - loss: 27.0653 - val_loss: 21.2440 - 66ms/epoch - 7ms/step
Epoch 197/1000
10/10 - 0s - loss: 26.5345 - val_loss: 21.7376 - 60ms/epoch - 6ms/step
Epoch 198/1000
10/10 - 0s - loss: 26.6457 - val_loss: 20.7840 - 43ms/epoch - 4ms/step
Epoch 199/1000
10/10 - 0s - loss: 26.1900 - val_loss: 20.5610 - 63ms/epoch - 6ms/step
Epoch 200/1000
10/10 - 0s - loss: 26.4989 - val_loss: 21.2801 - 73ms/epoch - 7ms/step
Epoch 201/1000
10/10 - 0s - loss: 25.7045 - val_loss: 20.3552 - 51ms/epoch - 5ms/step
Epoch 202/1000
10/10 - 0s - loss: 26.2127 - val_loss: 20.3249 - 47ms/epoch - 5ms/step
Epoch 203/1000
10/10 - 0s - loss: 26.5411 - val_loss: 23.3969 - 43ms/epoch - 4ms/step
Epoch 204/1000
10/10 - 0s - loss: 26.0485 - val_loss: 19.6083 - 52ms/epoch - 5ms/step
Epoch 205/1000
10/10 - 0s - loss: 25.7512 - val_loss: 21.4974 - 61ms/epoch - 6ms/step
Epoch 206/1000
10/10 - 0s - loss: 27.9661 - val_loss: 24.6909 - 73ms/epoch - 7ms/step
Epoch 207/1000
10/10 - 0s - loss: 27.1431 - val_loss: 20.3076 - 41ms/epoch - 4ms/step
Epoch 208/1000
10/10 - 0s - loss: 25.5981 - val_loss: 20.7630 - 43ms/epoch - 4ms/step
Epoch 209/1000
10/10 - 0s - loss: 25.4804 - val_loss: 18.8038 - 65ms/epoch - 7ms/step
Epoch 210/1000
10/10 - 0s - loss: 26.6374 - val_loss: 26.9133 - 63ms/epoch - 6ms/step
Epoch 211/1000
10/10 - 0s - loss: 26.2150 - val_loss: 18.8805 - 48ms/epoch - 5ms/step
Epoch 212/1000
10/10 - 0s - loss: 25.7188 - val_loss: 20.6097 - 50ms/epoch - 5ms/step
Epoch 213/1000
10/10 - 0s - loss: 24.9249 - val_loss: 18.8219 - 50ms/epoch - 5ms/step
Epoch 214/1000
Restoring model weights from the end of the best epoch: 209.
10/10 - 0s - loss: 24.0144 - val_loss: 19.2638 - 50ms/epoch - 5ms/step
Epoch 214: early stopping
```

Out[3]: <keras.callbacks.History at 0x7f126e090b90>

We now evaluate the trained neural network to see the RMSE. We will use this trained neural network to compare the accuracy between the original data and the GAN-generated data. We will later see that you can use such comparisons for anomaly detection. We can use this technique can be used for security systems. If a

neural network trained on original data does not perform well on new data, then the new data may be suspect or fake.

In [4]:
```
pred = model.predict(x_test)
score = np.sqrt(metrics.mean_squared_error(pred,y_test))
print("Final score (RMSE): {}".format(score))
```

```
Final score (RMSE): 4.33633936452545
```

Training a GAN for Auto MPG

Next, we will train the GAN to generate fake data from the original MPG data. There are quite a few options that you can fine-tune for the GAN. The example presented here uses most of the default values. These are the usual hyperparameters that must be tuned for any model and require some experimentation for optimal results. To learn more about tabgab refer to its paper or this Medium article, written by the creator of tabgan.

In [5]:
```
from tabgan.sampler import GANGenerator
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

gen_x, gen_y = GANGenerator(gen_x_times=1.1, cat_cols=None,
           bot_filter_quantile=0.001, top_filter_quantile=0.999, \
              is_post_process=True,
           adversarial_model_params={
               "metrics": "rmse", "max_depth": 2, "max_bin": 100,
               "learning_rate": 0.02, "random_state": \
                42, "n_estimators": 500,
           }, pregeneration_frac=2, only_generated_data=False,\
           gan_params = {"batch_size": 500, "patience": 25, \
          "epochs" : 500,}).generate_data_pipe(df_x_train, df_y_train,\
          df_x_test, deep_copy=True, only_adversarial=False, \
          use_adversarial=True)
```

```
Fitting CTGAN transformers for each column:   0%|          | 0/8 [00:00<?,
?it/s]
Training CTGAN, epochs::   0%|          | 0/500 [00:00<?, ?it/s]
```

Note: if you receive an error running the above code, you likely need to restart the runtime. You should have a "restart runtime" button in the output from the second cell. Once you restart the runtime, rerun all of the cells. This step is necessary as tabgan requires specific versions of some packages.

Evaluating the GAN Results

If we display the results, we can see that the GAN-generated data looks similar to the original. Some values, typically whole numbers in the original data, have fractional values in the synthetic data.

In [6]:
```
gen_x
```

Out[6]:

	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	5	296.949632	106.872450	2133	18.323035	73	2
1	5	247.744505	97.532052	2233	19.490136	75	2
2	4	259.648421	108.111921	2424	19.898952	79	3
3	5	319.208637	93.764364	2054	19.420225	78	3
4	4	386.237667	129.837418	1951	20.989091	82	2
...	...	...	...	...	...	...	...
542	8	304.000000	150.000000	3672	11.500000	72	1
543	8	304.000000	150.000000	3433	12.000000	70	1
544	4	98.000000	80.000000	2164	15.000000	72	1
545	4	97.500000	80.000000	2126	17.000000	72	1
546	5	138.526374	68.958515	2497	13.495784	71	1

547 rows × 7 columns

Finally, we present the synthetic data to the previously trained neural network to see how accurately we can predict the synthetic targets. As we can see, you lose some RMSE accuracy by going to synthetic data.

In [7]:
```
# Predict
pred = model.predict(gen_x.values)
score = np.sqrt(metrics.mean_squared_error(pred,gen_y.values))
print("Final score (RMSE): {}".format(score))
```

```
Final score (RMSE): 9.083745225633098
```