In [1]:
```
# DATASET LOADING AND PROCESSING (DO NEED TO MODIFY)
import os
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from torch.optim import Adam
import torch.nn.functional as F


def plot_airfoils(airfoil_x, airfoil_y):
    '''
    plot airfoils: no need to modify
    '''
    idx = 0
    fig, ax = plt.subplots(nrows=4, ncols=4)
    for row in ax:
        for col in row:
            col.scatter(airfoil_x, airfoil_y[idx, :], s=0.6, c='black')
            col.axis('off')
            col.axis('equal')
            idx += 1
    plt.show()


class AirfoilDataset(Dataset):
    '''
    airfoil dataset: no need to modify
    '''
    def __init__(self, path='/content/drive/MyDrive/Colab Notebooks/airfoils'):
        super(AirfoilDataset, self).__init__()
        self._X = []    # x coordinates of all airfoils (shared)
        self._Y = []    # y coordinates of all airfoils
        self.names = [] # name of all airfoils
        self.norm_coeff = 0     # normalization coeff to scale y to [-1, 1]
        airfoil_fn = [afn for afn in os.listdir(path) if afn.endswith('.dat')]

        # get x coordinates of all airfoils
        with open(os.path.join(path, airfoil_fn[0]), 'r', encoding="utf8", errors='ignore') as f:
            raw_data = f.readlines()
            for idx in range(len(raw_data)):
                raw_xy = raw_data[idx].split(' ')
```

```
                while "" in raw_xy:
                    raw_xy.remove("")
                self._X.append(float(raw_xy[0]))
        self._X = np.array(self._X)

        # get y coordinates of each airfoils
        for idx, fn in enumerate(airfoil_fn):
            with open(os.path.join(path, fn), 'r', encoding="utf8", errors='ignore') as f:
                self.names.append(fn[:-10])
                raw_data = f.readlines()
                airfoil = np.empty(self._X.shape[0])
                for i in range(len(raw_data)):
                    raw_xy = raw_data[i].split(' ')
                    while "" in raw_xy:
                        raw_xy.remove("")
                    curr_y = float(raw_xy[1])
                    airfoil[i] = curr_y
                    self.norm_coeff = max(self.norm_coeff, np.abs(curr_y))
                self._Y.append(airfoil)

        self._Y = np.array([airfoil / self.norm_coeff for airfoil in self._Y], dtype=np.float32)

    def get_x(self):
        '''
        get shared x coordinates
        '''
        return self._X

    def get_y(self):
        '''
        get y coordinates of all airfoils
        '''
        return self._Y

    def __getitem__(self, idx):
        return self._Y[idx], self.names[idx]

    def __len__(self):
        return len(self._Y)
```

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

VAE

In [14]:
```
class Encoder(nn.Module):
    def __init__(self, input_dim, latent_dim):
        super(Encoder, self).__init__()

        self.fc1 = nn.Linear(input_dim, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128,64)

        self.fc_mean = nn.Linear(64, latent_dim)
        self.fc_std = nn.Linear(64, latent_dim)

    def forward(self, x):

        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))

        mean = self.fc_mean(x)
        log_var = self.fc_std(x)

        return mean, log_var


class Decoder(nn.Module):
    def __init__(self, latent_dim, output_dim):
        super(Decoder, self).__init__()

        self.fc1 = nn.Linear(latent_dim, 64)
        self.fc2 = nn.Linear(64, 128)
        self.fc3 = nn.Linear(128, 256)
        self.fc4 = nn.Linear(256, output_dim)

    def forward(self, x):

        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = torch.tanh(self.fc4(x))

        return x
```

```python
class VAE(nn.Module):
    def __init__(self, airfoil_dim, latent_dim):
        super(VAE, self).__init__()
        self.enc = Encoder(airfoil_dim, latent_dim)
        self.dec = Decoder(latent_dim, airfoil_dim)

    def reparameterize(self, mean, log_var):

        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)

        return mean + eps * std

    def forward(self, x):

        mean, log_var = self.enc(x)
        z = self.reparameterize(mean, log_var)
        recon_x = self.dec(z)

        return recon_x, mean, log_var

    def decode(self, z):

        return self.dec(z)
```

In [32]:

```python
# check if cuda available
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

# define dataset and dataloader
dataset = AirfoilDataset()
airfoil_x = dataset.get_x()
airfoil_dim = airfoil_x.shape[0]
airfoil_dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

# hyperparameters
latent_dim = 16 # please do not change latent dimension
lr = 0.001      # learning rate
num_epochs = 30


# build the model
vae = VAE(airfoil_dim=airfoil_dim, latent_dim=latent_dim).to(device)
print("VAE model:\n", vae)
```

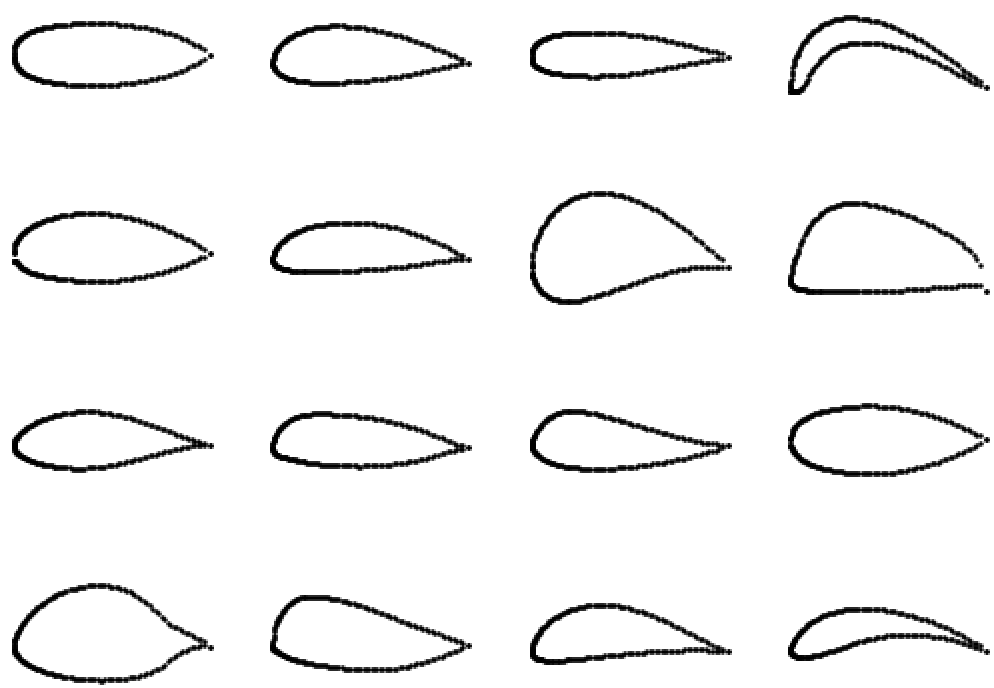
```python
# define your loss function here
# loss = ?

def vae_loss(recon_x, x, mu, logvar):
    recon_loss = F.mse_loss(recon_x, x, reduction='sum') / x.size(0)
    kl_loss = torch.mean(-0.5*torch.sum(1+ logvar- mu**2 - logvar.exp(),dim=1), dim=0)
    loss = recon_loss + 0.2*kl_loss

    return loss

# define optimizer for discriminator and generator separately
optim = Adam(vae.parameters(), lr=lr)

losses = []

# train the VAE model
for epoch in range(num_epochs):
    epoch_losses = []
    for n_batch, (local_batch, __) in enumerate(airfoil_dataloader):
        y_real = local_batch.to(device)

        # train VAE

        # calculate customized VAE loss
        # loss = your_loss_func(...)

        recon_batch, mu, logvar = vae(y_real)
        loss = vae_loss(recon_batch, y_real, mu, logvar)

        optim.zero_grad()
        loss.backward()
        optim.step()

        epoch_losses.append(loss.item())

        # print loss while training
        if (n_batch + 1) % 30 == 0:
            print("Epoch: [{}/{}], Batch: {}, loss: {}".format(
                epoch, num_epochs, n_batch, loss.item()))

    epoch_loss = sum(epoch_losses) / len(epoch_losses)
    losses.append(epoch_loss)

# test trained VAE model
```

```
num_samples = 100

# reconstuct airfoils
real_airfoils = dataset.get_y()[:num_samples]
recon_airfoils, __, __ = vae(torch.from_numpy(real_airfoils).to(device))
if 'cuda' in device:
    recon_airfoils = recon_airfoils.detach().cpu().numpy()
else:
    recon_airfoils = recon_airfoils.detach().numpy()

# randomly synthesize airfoils
noise = torch.randn((num_samples, latent_dim)).to(device)   # create random noise
gen_airfoils = vae.decode(noise)
if 'cuda' in device:
    gen_airfoils = gen_airfoils.detach().cpu().numpy()
else:
    gen_airfoils = gen_airfoils.detach().numpy()

# plot real/reconstructed/synthesized airfoils
print("real airfoils")
plot_airfoils(airfoil_x, real_airfoils)
print("reconstructed airfoils")
plot_airfoils(airfoil_x, recon_airfoils)
print("synthesized airfoils")
plot_airfoils(airfoil_x, gen_airfoils)

print(" ")
plt.figure()
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs Epoch')
plt.show()

torch.save(vae.state_dict(), '/content/drive/MyDrive/Colab Notebooks/vae.pth')
```

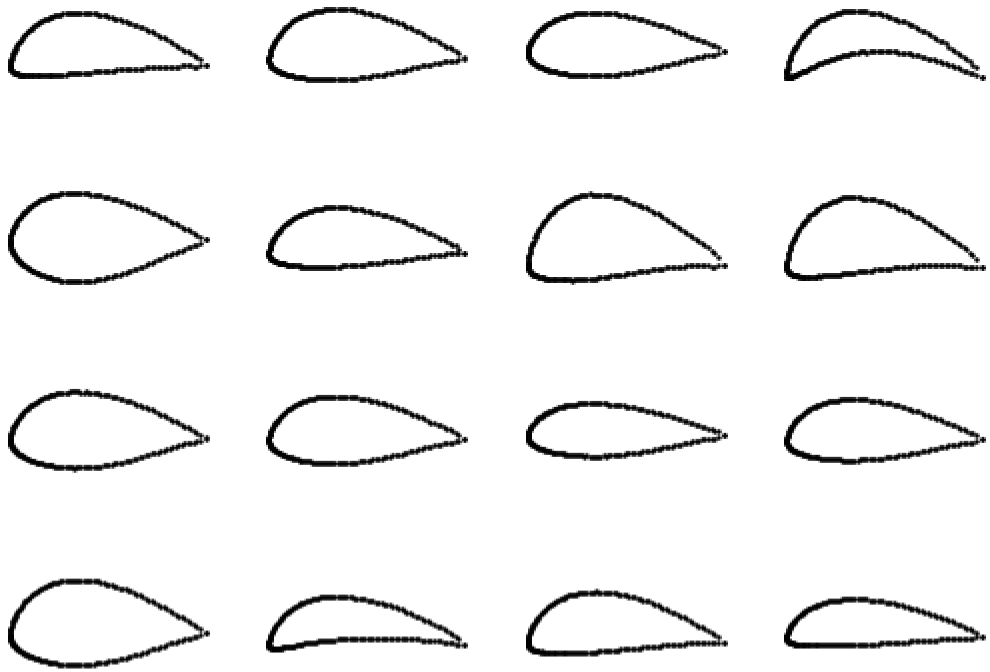
```
VAE model:
 VAE(
  (enc): Encoder(
    (fc1): Linear(in_features=200, out_features=256, bias=True)
    (fc2): Linear(in_features=256, out_features=128, bias=True)
    (fc3): Linear(in_features=128, out_features=64, bias=True)
    (fc_mean): Linear(in_features=64, out_features=16, bias=True)
    (fc_std): Linear(in_features=64, out_features=16, bias=True)
  )
  (dec): Decoder(
    (fc1): Linear(in_features=16, out_features=64, bias=True)
    (fc2): Linear(in_features=64, out_features=128, bias=True)
    (fc3): Linear(in_features=128, out_features=256, bias=True)
    (fc4): Linear(in_features=256, out_features=200, bias=True)
  )
)
Epoch: [0/30], Batch: 29, loss: 1.2568283081054688
Epoch: [0/30], Batch: 59, loss: 0.967051088809967
Epoch: [0/30], Batch: 89, loss: 1.1113934516906738
Epoch: [1/30], Batch: 29, loss: 1.2306588888168335
Epoch: [1/30], Batch: 59, loss: 0.7745790481567383
Epoch: [1/30], Batch: 89, loss: 0.5744256377220154
Epoch: [2/30], Batch: 29, loss: 0.795558512210846
Epoch: [2/30], Batch: 59, loss: 0.612313985824585
Epoch: [2/30], Batch: 89, loss: 0.7001008987426758
Epoch: [3/30], Batch: 29, loss: 0.8261765241622925
Epoch: [3/30], Batch: 59, loss: 0.8821898698806763
Epoch: [3/30], Batch: 89, loss: 1.505903720855713
Epoch: [4/30], Batch: 29, loss: 0.5241866707801819
Epoch: [4/30], Batch: 59, loss: 0.7192654609680176
Epoch: [4/30], Batch: 89, loss: 0.6514350175857544
Epoch: [5/30], Batch: 29, loss: 0.964148998260498
Epoch: [5/30], Batch: 59, loss: 0.6244620084762573
Epoch: [5/30], Batch: 89, loss: 0.5171216726303101
Epoch: [6/30], Batch: 29, loss: 0.6415860056877136
Epoch: [6/30], Batch: 59, loss: 0.5762733221054077
Epoch: [6/30], Batch: 89, loss: 0.6678453087806702
Epoch: [7/30], Batch: 29, loss: 0.7610674500465393
Epoch: [7/30], Batch: 59, loss: 0.7712926268577576
Epoch: [7/30], Batch: 89, loss: 0.6050438284873962
Epoch: [8/30], Batch: 29, loss: 0.6862906813621521
Epoch: [8/30], Batch: 59, loss: 0.508882999420166
Epoch: [8/30], Batch: 89, loss: 0.6168999671936035
Epoch: [9/30], Batch: 29, loss: 0.5428687930107117
Epoch: [9/30], Batch: 59, loss: 0.4210575819015503
```

```
Epoch: [9/30], Batch: 89, loss: 0.5457625985145569
Epoch: [10/30], Batch: 29, loss: 0.4545830488204956
Epoch: [10/30], Batch: 59, loss: 0.5822412967681885
Epoch: [10/30], Batch: 89, loss: 0.5887246131896973
Epoch: [11/30], Batch: 29, loss: 0.3662900924682617
Epoch: [11/30], Batch: 59, loss: 0.5479189157485962
Epoch: [11/30], Batch: 89, loss: 0.6750072836875916
Epoch: [12/30], Batch: 29, loss: 0.6728198528289795
Epoch: [12/30], Batch: 59, loss: 0.5002647638320923
Epoch: [12/30], Batch: 89, loss: 0.44942396879196167
Epoch: [13/30], Batch: 29, loss: 0.5729416012763977
Epoch: [13/30], Batch: 59, loss: 0.6276856660842896
Epoch: [13/30], Batch: 89, loss: 0.46042656898498535
Epoch: [14/30], Batch: 29, loss: 0.5128128528594971
Epoch: [14/30], Batch: 59, loss: 0.507204532623291
Epoch: [14/30], Batch: 89, loss: 0.6028555631637573
Epoch: [15/30], Batch: 29, loss: 0.5595499277114868
Epoch: [15/30], Batch: 59, loss: 0.8116866946220398
Epoch: [15/30], Batch: 89, loss: 0.4068645238876343
Epoch: [16/30], Batch: 29, loss: 0.6456145644187927
Epoch: [16/30], Batch: 59, loss: 0.5736095905303955
Epoch: [16/30], Batch: 89, loss: 0.4680807590484619
Epoch: [17/30], Batch: 29, loss: 0.5523345470428467
Epoch: [17/30], Batch: 59, loss: 0.5720200538635254
Epoch: [17/30], Batch: 89, loss: 0.6506773233413696
Epoch: [18/30], Batch: 29, loss: 0.48725438117980957
Epoch: [18/30], Batch: 59, loss: 0.6139523386955261
Epoch: [18/30], Batch: 89, loss: 0.5469404458999634
Epoch: [19/30], Batch: 29, loss: 0.43494850397109985
Epoch: [19/30], Batch: 59, loss: 0.6223950386047363
Epoch: [19/30], Batch: 89, loss: 0.41155803203582764
Epoch: [20/30], Batch: 29, loss: 0.3579903542995453
Epoch: [20/30], Batch: 59, loss: 0.5147667527198792
Epoch: [20/30], Batch: 89, loss: 0.4679056406021118
Epoch: [21/30], Batch: 29, loss: 0.5972059369087219
Epoch: [21/30], Batch: 59, loss: 0.6776161193847656
Epoch: [21/30], Batch: 89, loss: 0.4607918858528137
Epoch: [22/30], Batch: 29, loss: 0.5408642292022705
Epoch: [22/30], Batch: 59, loss: 0.6033877730369568
Epoch: [22/30], Batch: 89, loss: 0.493122398853302
Epoch: [23/30], Batch: 29, loss: 0.4994364380836487
Epoch: [23/30], Batch: 59, loss: 0.45746976137161255
Epoch: [23/30], Batch: 89, loss: 0.5262988209724426
Epoch: [24/30], Batch: 29, loss: 0.6348119378089905
Epoch: [24/30], Batch: 59, loss: 0.5356062650680542
```

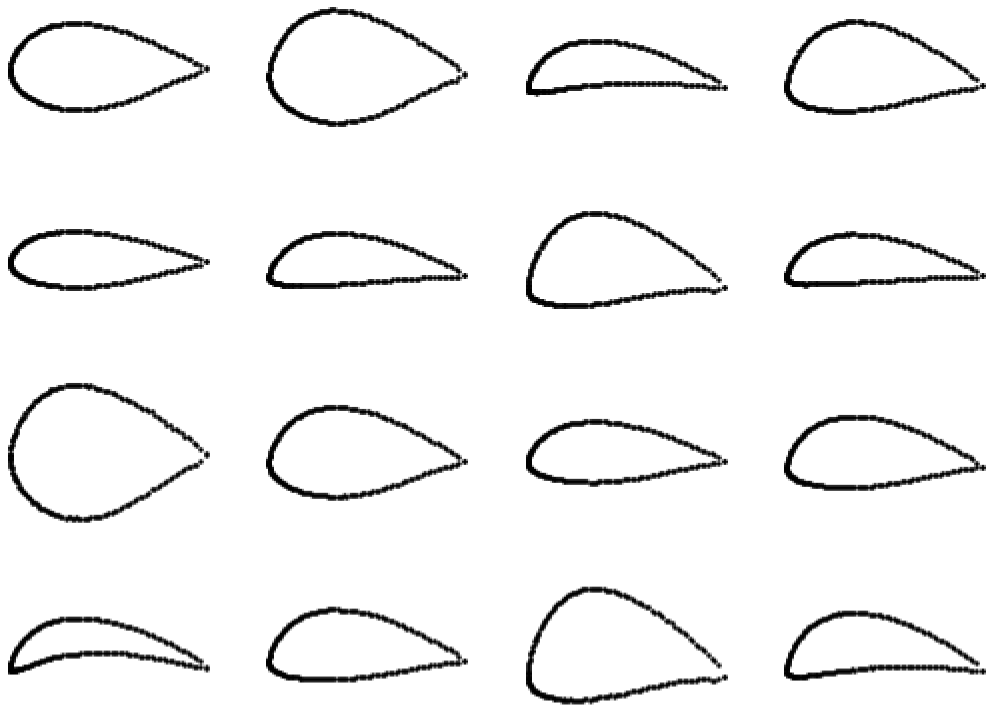
```
Epoch: [24/30], Batch: 89, loss: 0.6589794754981995
Epoch: [25/30], Batch: 29, loss: 0.5460250973701477
Epoch: [25/30], Batch: 59, loss: 0.895304799079895
Epoch: [25/30], Batch: 89, loss: 0.6378650665283203
Epoch: [26/30], Batch: 29, loss: 0.7089762687683105
Epoch: [26/30], Batch: 59, loss: 0.41426604986190796
Epoch: [26/30], Batch: 89, loss: 0.4573306441307068
Epoch: [27/30], Batch: 29, loss: 0.49216634035110474
Epoch: [27/30], Batch: 59, loss: 0.4741140604019165
Epoch: [27/30], Batch: 89, loss: 0.5339498519897461
Epoch: [28/30], Batch: 29, loss: 0.46695953607559204
Epoch: [28/30], Batch: 59, loss: 0.38788992166519165
Epoch: [28/30], Batch: 89, loss: 0.662644624710083
Epoch: [29/30], Batch: 29, loss: 0.629335343837738
Epoch: [29/30], Batch: 59, loss: 0.6217707395553589
Epoch: [29/30], Batch: 89, loss: 0.5966886281967163
real airfoils
```

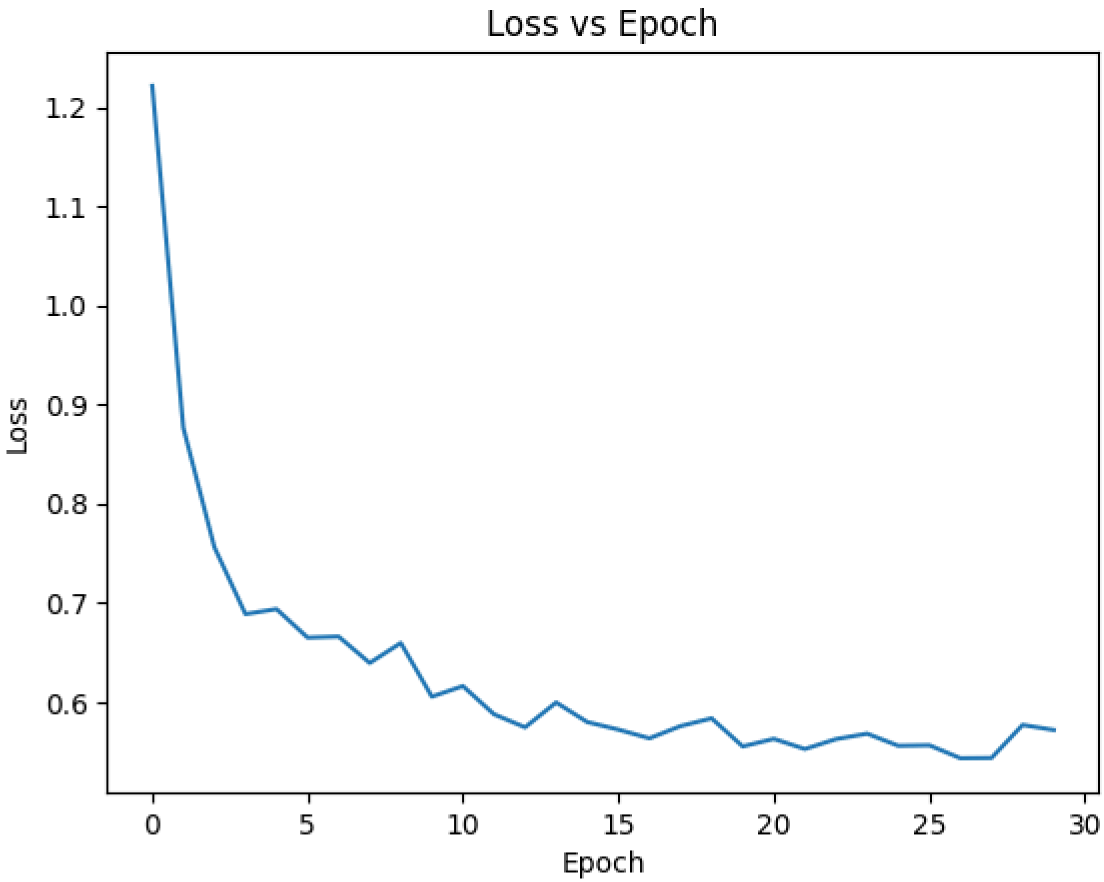


```
reconstructed airfoils
```



synthesized airfoils





GAN

In [28]:
```
class Discriminator(nn.Module):
    def __init__(self, input_dim):
        super(Discriminator, self).__init__()
        # build your model here
        # your output should be of dim (batch_size, 1)
        # since discriminator is a binary classifier
        self.model = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 64),
            nn.LeakyReLU(0.2),
            nn.Linear(64, 1),
```

```
            nn.Sigmoid()
        )

    def forward(self, x):
        # define your feedforward pass
        return self.model(x)


class Generator(nn.Module):
    def __init__(self, latent_dim, airfoil_dim):
        super(Generator, self).__init__()
        # build your model here
        # your output should be of dim (batch_size, airfoil_dim)
        # you can use tanh() as the activation for the last layer
        # since y coord of airfoils range from -1 to 1
        self.model = nn.Sequential(
            nn.Linear(latent_dim, 64),
            nn.LeakyReLU(0.2),
            nn.Linear(64, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, airfoil_dim),
            nn.Tanh()
        )

    def forward(self, x):
        # define your feedforward pass
        return self.model(x)
```

In [34]:

```
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

# define dataset and dataloader
dataset = AirfoilDataset()
airfoil_x = dataset.get_x()
airfoil_dim = airfoil_x.shape[0]
airfoil_dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

# hyperparameters
latent_dim = 16 # please do not change latent dimension
lr_dis = 0.0005 # discriminator learning rate
lr_gen = 0.0005 # generator learning rate
num_epochs = 60

# build the model
dis = Discriminator(input_dim=airfoil_dim).to(device)
gen = Generator(latent_dim=latent_dim, airfoil_dim=airfoil_dim).to(device)
```

```
print("Distrminator model:\n", dis)
print("Generator model:\n", gen)

# define your GAN loss function here
# you may need to define your own GAN loss function/class
# loss = ?
adversarial_loss = nn.BCELoss()

# define optimizer for discriminator and generator separately
optim_dis = Adam(dis.parameters(), lr=lr_dis)
optim_gen = Adam(gen.parameters(), lr=lr_gen)

dis_losses = []
gen_losses = []

# train the GAN model
for epoch in range(num_epochs):
    epoch_dis_losses = []
    epoch_gen_losses = []
    for n_batch, (local_batch, __) in enumerate(airfoil_dataloader):
        y_real = local_batch.to(device)

        # train discriminator
        optim_dis.zero_grad()

        # calculate customized GAN loss for discriminator
        # enc_loss = loss(...)
        y_pred_real = dis(y_real)
        real_loss = adversarial_loss(y_pred_real, torch.ones_like(y_pred_real).to(device))

        noise = torch.randn((local_batch.size(0), latent_dim)).to(device)
        gen_airfoils = gen(noise)
        y_pred_gen = dis(gen_airfoils.detach())
        fake_loss = adversarial_loss(y_pred_gen, torch.zeros_like(y_pred_gen).to(device))

        loss_dis = (real_loss + fake_loss) / 2

        loss_dis.backward()
        optim_dis.step()

        # train generator

        # calculate customized GAN loss for generator
        # enc_loss = loss(...)
```

```
        optim_gen.zero_grad()

        gen_airfoils = gen(noise)
        y_pred_gen = dis(gen_airfoils)
        loss_gen = adversarial_loss(y_pred_gen, torch.ones_like(y_pred_gen).to(device))

        loss_gen.backward()
        optim_gen.step()

        epoch_dis_losses.append(loss_dis.item())
        epoch_gen_losses.append(loss_gen.item())

        # print loss while training
        if (n_batch + 1) % 30 == 0:
            print("Epoch: [{}/{}], Batch: {}, Discriminator loss: {}, Generator loss: {}".format(
                epoch, num_epochs, n_batch, loss_dis.item(), loss_gen.item()))

    avg_dis_loss = sum(epoch_dis_losses) / len(epoch_dis_losses)
    avg_gen_loss = sum(epoch_gen_losses) / len(epoch_gen_losses)
    dis_losses.append(avg_dis_loss)
    gen_losses.append(avg_gen_loss)

# test trained GAN model
num_samples = 100
# create random noise
noise = torch.randn((num_samples, latent_dim)).to(device)
# generate airfoils
gen_airfoils = gen(noise)
if 'cuda' in device:
    gen_airfoils = gen_airfoils.detach().cpu().numpy()
else:
    gen_airfoils = gen_airfoils.detach().numpy()

# plot generated airfoils
print("generated airfoils")
plot_airfoils(airfoil_x, gen_airfoils)

print(" ")
plt.plot(dis_losses, label='Discriminator Loss')
plt.plot(gen_losses, label='Generator Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('GAN Losses vs Epoch')
plt.legend()
plt.show()
```

```
torch.save(dis.state_dict(), '/content/drive/MyDrive/Colab Notebooks/dis.pth')
torch.save(gen.state_dict(), '/content/drive/MyDrive/Colab Notebooks/gen.pth')
```

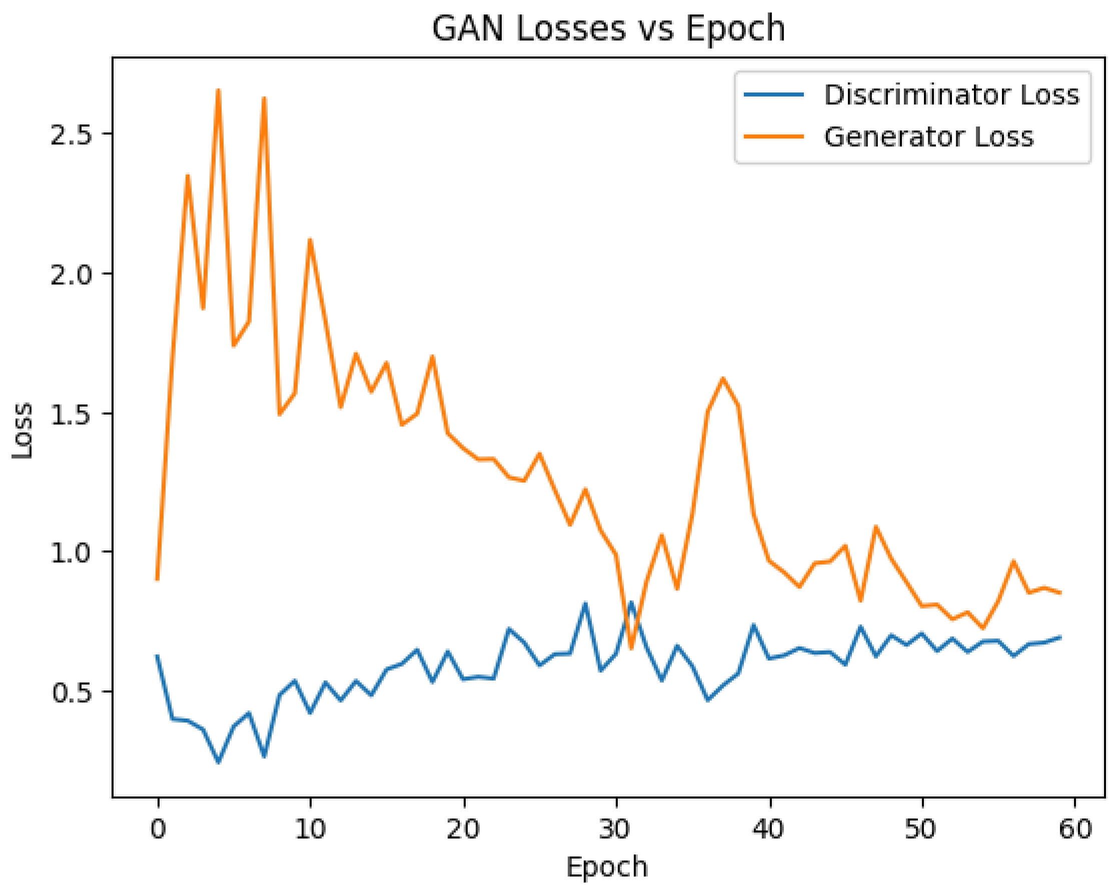
```
Distrminator model:
 Discriminator(
  (model): Sequential(
    (0): Linear(in_features=200, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
    (4): Linear(in_features=64, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
Generator model:
 Generator(
  (model): Sequential(
    (0): Linear(in_features=16, out_features=64, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=64, out_features=128, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
    (4): Linear(in_features=128, out_features=200, bias=True)
    (5): Tanh()
  )
)
Epoch: [0/60], Batch: 29, Discriminator loss: 0.6550744771957397, Generator loss: 0.6155168414115906
Epoch: [0/60], Batch: 59, Discriminator loss: 0.79807049036026, Generator loss: 0.831487238407135
Epoch: [0/60], Batch: 89, Discriminator loss: 0.4181658923625946, Generator loss: 1.153719425201416
Epoch: [1/60], Batch: 29, Discriminator loss: 0.46174147725105286, Generator loss: 1.1566137075424194
Epoch: [1/60], Batch: 59, Discriminator loss: 0.3682071566581726, Generator loss: 1.724064826965332
Epoch: [1/60], Batch: 89, Discriminator loss: 0.21780501306056976, Generator loss: 2.2987823486328125
Epoch: [2/60], Batch: 29, Discriminator loss: 0.25270870327949524, Generator loss: 3.244699239730835
Epoch: [2/60], Batch: 59, Discriminator loss: 0.34200167655944824, Generator loss: 2.384166955947876
Epoch: [2/60], Batch: 89, Discriminator loss: 0.43785256147384644, Generator loss: 1.7672066688537598
Epoch: [3/60], Batch: 29, Discriminator loss: 0.2967348098754883, Generator loss: 1.7337510585784912
Epoch: [3/60], Batch: 59, Discriminator loss: 0.5267454385757446, Generator loss: 1.4155938625335693
Epoch: [3/60], Batch: 89, Discriminator loss: 0.2983630299568176, Generator loss: 2.2027039527893066
Epoch: [4/60], Batch: 29, Discriminator loss: 0.14755883812904358, Generator loss: 3.0049171447753906
Epoch: [4/60], Batch: 59, Discriminator loss: 0.20020607113838196, Generator loss: 2.671449899673462
Epoch: [4/60], Batch: 89, Discriminator loss: 0.22030708193778992, Generator loss: 2.386460781097412
Epoch: [5/60], Batch: 29, Discriminator loss: 0.30570393800735474, Generator loss: 1.923386573791504
Epoch: [5/60], Batch: 59, Discriminator loss: 0.3364807963371277, Generator loss: 2.173663854598999
Epoch: [5/60], Batch: 89, Discriminator loss: 0.43400177359580994, Generator loss: 1.2687697410583496
Epoch: [6/60], Batch: 29, Discriminator loss: 0.5224012732505798, Generator loss: 1.1397604942321777
Epoch: [6/60], Batch: 59, Discriminator loss: 0.34007155895233154, Generator loss: 2.047773838043213
Epoch: [6/60], Batch: 89, Discriminator loss: 0.32929712533950806, Generator loss: 3.330932140350342
Epoch: [7/60], Batch: 29, Discriminator loss: 0.22194524109363556, Generator loss: 2.970999002456665
Epoch: [7/60], Batch: 59, Discriminator loss: 0.24797102808952332, Generator loss: 2.0921883583068848
```

```
Epoch: [7/60], Batch: 89, Discriminator loss: 0.3005181849002838, Generator loss: 1.9180964231491089
Epoch: [8/60], Batch: 29, Discriminator loss: 0.6067341566085815, Generator loss: 1.2839103937149048
Epoch: [8/60], Batch: 59, Discriminator loss: 0.3172677755355835, Generator loss: 1.4541579484939575
Epoch: [8/60], Batch: 89, Discriminator loss: 0.46880441904067993, Generator loss: 1.6380051374435425
Epoch: [9/60], Batch: 29, Discriminator loss: 0.8842848539352417, Generator loss: 0.7257320880889893
Epoch: [9/60], Batch: 59, Discriminator loss: 0.38572508096694946, Generator loss: 1.668159008026123
Epoch: [9/60], Batch: 89, Discriminator loss: 0.49421876668930054, Generator loss: 1.7136098146438599
Epoch: [10/60], Batch: 29, Discriminator loss: 0.4835028052330017, Generator loss: 1.6844137907028198
Epoch: [10/60], Batch: 59, Discriminator loss: 0.4390072226524353, Generator loss: 1.9456424713134766
Epoch: [10/60], Batch: 89, Discriminator loss: 0.3860313296318054, Generator loss: 2.02215313911438
Epoch: [11/60], Batch: 29, Discriminator loss: 1.0142091512680054, Generator loss: 1.1940572261810303
Epoch: [11/60], Batch: 59, Discriminator loss: 0.8681546449661255, Generator loss: 1.4249837398529053
Epoch: [11/60], Batch: 89, Discriminator loss: 0.5403953790664673, Generator loss: 1.5099586248397827
Epoch: [12/60], Batch: 29, Discriminator loss: 0.49638545513153076, Generator loss: 1.3146133422851562
Epoch: [12/60], Batch: 59, Discriminator loss: 0.457496702671051, Generator loss: 1.5028020143508911
Epoch: [12/60], Batch: 89, Discriminator loss: 0.5323343276977539, Generator loss: 1.2125569581985474
Epoch: [13/60], Batch: 29, Discriminator loss: 0.4789835214614868, Generator loss: 1.731103539466858
Epoch: [13/60], Batch: 59, Discriminator loss: 0.7653356194496155, Generator loss: 1.8054252862930298
Epoch: [13/60], Batch: 89, Discriminator loss: 0.3440096378326416, Generator loss: 1.7626421451568604
Epoch: [14/60], Batch: 29, Discriminator loss: 0.6798123121261597, Generator loss: 1.0716384649276733
Epoch: [14/60], Batch: 59, Discriminator loss: 0.3068241477012634, Generator loss: 1.7259663343429565
Epoch: [14/60], Batch: 89, Discriminator loss: 0.34006908535957336, Generator loss: 1.5672533512115479
Epoch: [15/60], Batch: 29, Discriminator loss: 0.4785253405570984, Generator loss: 1.9566551446914673
Epoch: [15/60], Batch: 59, Discriminator loss: 0.18439318239688873, Generator loss: 3.1212124824523926
Epoch: [15/60], Batch: 89, Discriminator loss: 0.6002306342124939, Generator loss: 2.00795578956604
Epoch: [16/60], Batch: 29, Discriminator loss: 0.42849200963974, Generator loss: 1.6339290142059326
Epoch: [16/60], Batch: 59, Discriminator loss: 0.8413727283477783, Generator loss: 1.3055349588394165
Epoch: [16/60], Batch: 89, Discriminator loss: 0.5671584010124207, Generator loss: 1.029969334602356
Epoch: [17/60], Batch: 29, Discriminator loss: 0.5220879316329956, Generator loss: 1.7177839279174805
Epoch: [17/60], Batch: 59, Discriminator loss: 0.41616034507751465, Generator loss: 1.8091237545013428
Epoch: [17/60], Batch: 89, Discriminator loss: 0.8541027307510376, Generator loss: 1.3087342977523804
Epoch: [18/60], Batch: 29, Discriminator loss: 0.33928823471069336, Generator loss: 1.821062445640564
Epoch: [18/60], Batch: 59, Discriminator loss: 0.8028422594070435, Generator loss: 1.1756054162979126
Epoch: [18/60], Batch: 89, Discriminator loss: 0.15898394584655762, Generator loss: 3.337599754333496
Epoch: [19/60], Batch: 29, Discriminator loss: 0.3965265452861786, Generator loss: 1.8631064891815186
Epoch: [19/60], Batch: 59, Discriminator loss: 0.7182698249816895, Generator loss: 1.2186071872711182
Epoch: [19/60], Batch: 89, Discriminator loss: 0.7422538995742798, Generator loss: 1.3206108808517456
Epoch: [20/60], Batch: 29, Discriminator loss: 0.3653023838996887, Generator loss: 1.4876093864440918
Epoch: [20/60], Batch: 59, Discriminator loss: 0.5067502856254578, Generator loss: 1.347588300704956
Epoch: [20/60], Batch: 89, Discriminator loss: 0.4327831566333771, Generator loss: 1.3175835609436035
Epoch: [21/60], Batch: 29, Discriminator loss: 0.8869737386703491, Generator loss: 0.6824265122413635
Epoch: [21/60], Batch: 59, Discriminator loss: 0.2506749927997589, Generator loss: 1.9018216133117676
Epoch: [21/60], Batch: 89, Discriminator loss: 0.5091173648834229, Generator loss: 1.3391380310058594
Epoch: [22/60], Batch: 29, Discriminator loss: 0.7604482173919678, Generator loss: 1.1425578594207764
Epoch: [22/60], Batch: 59, Discriminator loss: 0.6245137453079224, Generator loss: 1.041046142578125
```

```
Epoch: [22/60], Batch: 89, Discriminator loss: 0.39898306131362915, Generator loss: 1.7080806493759155
Epoch: [23/60], Batch: 29, Discriminator loss: 0.367755651473999, Generator loss: 2.1869661808013916
Epoch: [23/60], Batch: 59, Discriminator loss: 0.6855685710906982, Generator loss: 1.0529059171676636
Epoch: [23/60], Batch: 89, Discriminator loss: 0.5897995829582214, Generator loss: 1.4049118757247925
Epoch: [24/60], Batch: 29, Discriminator loss: 0.7133272886276245, Generator loss: 1.05250084400177
Epoch: [24/60], Batch: 59, Discriminator loss: 0.5364795327186584, Generator loss: 1.2549179792404175
Epoch: [24/60], Batch: 89, Discriminator loss: 0.6670167446136475, Generator loss: 1.1230891942977905
Epoch: [25/60], Batch: 29, Discriminator loss: 0.833859920501709, Generator loss: 0.9770385026931763
Epoch: [25/60], Batch: 59, Discriminator loss: 0.6272759437561035, Generator loss: 1.2504569292068481
Epoch: [25/60], Batch: 89, Discriminator loss: 0.4370332360267639, Generator loss: 1.49563729763031
Epoch: [26/60], Batch: 29, Discriminator loss: 0.5266097784042358, Generator loss: 1.2164533138275146
Epoch: [26/60], Batch: 59, Discriminator loss: 0.8596965074539185, Generator loss: 1.1119816303253174
Epoch: [26/60], Batch: 89, Discriminator loss: 0.40002697706222534, Generator loss: 1.4983091354370117
Epoch: [27/60], Batch: 29, Discriminator loss: 0.9480269551277161, Generator loss: 0.8282008171081543
Epoch: [27/60], Batch: 59, Discriminator loss: 0.6581426858901978, Generator loss: 0.8950518369674683
Epoch: [27/60], Batch: 89, Discriminator loss: 0.4915812015533447, Generator loss: 1.4077057838439941
Epoch: [28/60], Batch: 29, Discriminator loss: 0.8851500153541565, Generator loss: 0.9962475299835205
Epoch: [28/60], Batch: 59, Discriminator loss: 0.3059955835342407, Generator loss: 1.9167571067810059
Epoch: [28/60], Batch: 89, Discriminator loss: 1.4116498231887817, Generator loss: 0.8815492987632751
Epoch: [29/60], Batch: 29, Discriminator loss: 0.5315619707107544, Generator loss: 1.0880274772644043
Epoch: [29/60], Batch: 59, Discriminator loss: 0.5203179717063904, Generator loss: 1.2397940158843994
Epoch: [29/60], Batch: 89, Discriminator loss: 0.3790057897567749, Generator loss: 1.3712213039398193
Epoch: [30/60], Batch: 29, Discriminator loss: 0.6317315697669983, Generator loss: 0.8245289325714111
Epoch: [30/60], Batch: 59, Discriminator loss: 0.3423987627029419, Generator loss: 1.3142080307006836
Epoch: [30/60], Batch: 89, Discriminator loss: 0.6688573360443115, Generator loss: 0.705692708492279
Epoch: [31/60], Batch: 29, Discriminator loss: 0.9661539793014526, Generator loss: 0.4662729501724243
Epoch: [31/60], Batch: 59, Discriminator loss: 0.9182649254798889, Generator loss: 0.5824178457260132
Epoch: [31/60], Batch: 89, Discriminator loss: 0.5749976634979248, Generator loss: 0.9410746693611145
Epoch: [32/60], Batch: 29, Discriminator loss: 0.45524904131889343, Generator loss: 1.2379218339920044
Epoch: [32/60], Batch: 59, Discriminator loss: 0.7716741561889648, Generator loss: 0.7345198392868042
Epoch: [32/60], Batch: 89, Discriminator loss: 0.8410378694534302, Generator loss: 0.6266323328018188
Epoch: [33/60], Batch: 29, Discriminator loss: 0.4983392059803009, Generator loss: 1.114750862121582
Epoch: [33/60], Batch: 59, Discriminator loss: 0.6466550827026367, Generator loss: 0.9364255666732788
Epoch: [33/60], Batch: 89, Discriminator loss: 0.33675622940063477, Generator loss: 1.4265449047088623
Epoch: [34/60], Batch: 29, Discriminator loss: 0.8425878286361694, Generator loss: 0.6343092918395996
Epoch: [34/60], Batch: 59, Discriminator loss: 0.5901330709457397, Generator loss: 0.9578167200088501
Epoch: [34/60], Batch: 89, Discriminator loss: 0.4834311008453369, Generator loss: 1.0596208572387695
Epoch: [35/60], Batch: 29, Discriminator loss: 0.40505972504615784, Generator loss: 1.3555068969726562
Epoch: [35/60], Batch: 59, Discriminator loss: 0.4687652587890625, Generator loss: 1.44296395778656
Epoch: [35/60], Batch: 89, Discriminator loss: 0.49450552463531494, Generator loss: 1.477943778038025
Epoch: [36/60], Batch: 29, Discriminator loss: 0.31306004524230957, Generator loss: 1.8560172319412231
Epoch: [36/60], Batch: 59, Discriminator loss: 0.614395260810852, Generator loss: 1.2210888862609863
Epoch: [36/60], Batch: 89, Discriminator loss: 0.4303891360759735, Generator loss: 1.649458885192871
Epoch: [37/60], Batch: 29, Discriminator loss: 0.398321270942688, Generator loss: 1.6341736316680908
Epoch: [37/60], Batch: 59, Discriminator loss: 0.4449957013130188, Generator loss: 1.8853516578674316
```

```
Epoch: [37/60], Batch: 89, Discriminator loss: 0.29372739791870117, Generator loss: 2.4962849617004395
Epoch: [38/60], Batch: 29, Discriminator loss: 0.5761655569076538, Generator loss: 1.3850969076156616
Epoch: [38/60], Batch: 59, Discriminator loss: 0.3512575030326843, Generator loss: 2.029810905456543
Epoch: [38/60], Batch: 89, Discriminator loss: 0.6089514493942261, Generator loss: 1.2272071838378906
Epoch: [39/60], Batch: 29, Discriminator loss: 0.24254441261291504, Generator loss: 2.0032308101654053
Epoch: [39/60], Batch: 59, Discriminator loss: 0.7419211864471436, Generator loss: 1.059429407119751
Epoch: [39/60], Batch: 89, Discriminator loss: 0.6946442127227783, Generator loss: 0.8261737823486328
Epoch: [40/60], Batch: 29, Discriminator loss: 0.5927780866622925, Generator loss: 0.9059246778488159
Epoch: [40/60], Batch: 59, Discriminator loss: 0.4606635272502899, Generator loss: 1.3043543100357056
Epoch: [40/60], Batch: 89, Discriminator loss: 0.6805889010429382, Generator loss: 0.9090443849563599
Epoch: [41/60], Batch: 29, Discriminator loss: 0.4865620732307434, Generator loss: 1.111356496810913
Epoch: [41/60], Batch: 59, Discriminator loss: 0.5597919225692749, Generator loss: 0.8459228873252869
Epoch: [41/60], Batch: 89, Discriminator loss: 0.705072283744812, Generator loss: 1.1493266820907593
Epoch: [42/60], Batch: 29, Discriminator loss: 0.6850334405899048, Generator loss: 0.818731963634491
Epoch: [42/60], Batch: 59, Discriminator loss: 0.6261404156684875, Generator loss: 0.8158000111579895
Epoch: [42/60], Batch: 89, Discriminator loss: 0.6271350383758545, Generator loss: 0.7644550800323486
Epoch: [43/60], Batch: 29, Discriminator loss: 0.5014773607254028, Generator loss: 1.2114092111587524
Epoch: [43/60], Batch: 59, Discriminator loss: 0.7913365960121155, Generator loss: 0.8646181225776672
Epoch: [43/60], Batch: 89, Discriminator loss: 0.5850721597671509, Generator loss: 1.0449743270874023
Epoch: [44/60], Batch: 29, Discriminator loss: 0.6637716293334961, Generator loss: 0.9551294445991516
Epoch: [44/60], Batch: 59, Discriminator loss: 0.5480940341949463, Generator loss: 1.0901297330856323
Epoch: [44/60], Batch: 89, Discriminator loss: 0.5976502299308777, Generator loss: 1.031406283378601
Epoch: [45/60], Batch: 29, Discriminator loss: 0.5492858290672302, Generator loss: 1.1886842250823975
Epoch: [45/60], Batch: 59, Discriminator loss: 0.63437420129776, Generator loss: 0.8836265206336975
Epoch: [45/60], Batch: 89, Discriminator loss: 0.4586101472377777, Generator loss: 1.4498043060302734
Epoch: [46/60], Batch: 29, Discriminator loss: 0.877993643283844, Generator loss: 0.5585631728172302
Epoch: [46/60], Batch: 59, Discriminator loss: 0.7689322829246521, Generator loss: 0.6868237257003784
Epoch: [46/60], Batch: 89, Discriminator loss: 0.4747493267059326, Generator loss: 1.2780448198318481
Epoch: [47/60], Batch: 29, Discriminator loss: 0.8922815322875977, Generator loss: 0.7054301500320435
Epoch: [47/60], Batch: 59, Discriminator loss: 0.8492335081100464, Generator loss: 0.6467353105545044
Epoch: [47/60], Batch: 89, Discriminator loss: 0.45494431257247925, Generator loss: 1.4828734397888184
Epoch: [48/60], Batch: 29, Discriminator loss: 0.6468321084976196, Generator loss: 1.04784095287323
Epoch: [48/60], Batch: 59, Discriminator loss: 0.5225121378898621, Generator loss: 1.0246562957763672
Epoch: [48/60], Batch: 89, Discriminator loss: 0.6194901466369629, Generator loss: 0.9672044515609741
Epoch: [49/60], Batch: 29, Discriminator loss: 0.8165537118911743, Generator loss: 0.8387085199356079
Epoch: [49/60], Batch: 59, Discriminator loss: 0.6847240328788757, Generator loss: 0.8032385110855103
Epoch: [49/60], Batch: 89, Discriminator loss: 0.6200693249702454, Generator loss: 0.9391229748725891
Epoch: [50/60], Batch: 29, Discriminator loss: 0.6744126677513123, Generator loss: 0.8406569361686707
Epoch: [50/60], Batch: 59, Discriminator loss: 0.8327611684799194, Generator loss: 0.5747285485267639
Epoch: [50/60], Batch: 89, Discriminator loss: 0.631398618221283, Generator loss: 0.922844648361206
Epoch: [51/60], Batch: 29, Discriminator loss: 0.5753737688064575, Generator loss: 0.8741309642791748
Epoch: [51/60], Batch: 59, Discriminator loss: 0.6163042783737183, Generator loss: 0.8699458837509155
Epoch: [51/60], Batch: 89, Discriminator loss: 0.6535316705703735, Generator loss: 0.7641764283180237
Epoch: [52/60], Batch: 29, Discriminator loss: 0.6994298696517944, Generator loss: 0.7291172742843628
Epoch: [52/60], Batch: 59, Discriminator loss: 0.6293007135391235, Generator loss: 0.8901227116584778
```

```
Epoch: [52/60], Batch: 89, Discriminator loss: 0.6804361343383789, Generator loss: 0.6589197516441345
Epoch: [53/60], Batch: 29, Discriminator loss: 0.6949654817581177, Generator loss: 0.6709092855453491
Epoch: [53/60], Batch: 59, Discriminator loss: 0.6522687673568726, Generator loss: 0.8338349461555481
Epoch: [53/60], Batch: 89, Discriminator loss: 0.7168437242507935, Generator loss: 0.7443930506706238
Epoch: [54/60], Batch: 29, Discriminator loss: 0.7844477891921997, Generator loss: 0.6269591450691223
Epoch: [54/60], Batch: 59, Discriminator loss: 0.6245450973510742, Generator loss: 0.8415483832359314
Epoch: [54/60], Batch: 89, Discriminator loss: 0.6183096170425415, Generator loss: 0.8732079267501831
Epoch: [55/60], Batch: 29, Discriminator loss: 0.5993813872337341, Generator loss: 0.9122262001037598
Epoch: [55/60], Batch: 59, Discriminator loss: 0.6694711446762085, Generator loss: 0.8146799206733704
Epoch: [55/60], Batch: 89, Discriminator loss: 0.7070192098617554, Generator loss: 0.7326091527938843
Epoch: [56/60], Batch: 29, Discriminator loss: 0.5011726021766663, Generator loss: 1.1044608354568481
Epoch: [56/60], Batch: 59, Discriminator loss: 0.6959010362625122, Generator loss: 0.8460350036621094
Epoch: [56/60], Batch: 89, Discriminator loss: 0.6619138717651367, Generator loss: 0.9028451442718506
Epoch: [57/60], Batch: 29, Discriminator loss: 0.5675066113471985, Generator loss: 0.8606228828430176
Epoch: [57/60], Batch: 59, Discriminator loss: 0.6036743521690369, Generator loss: 0.8372207880020142
Epoch: [57/60], Batch: 89, Discriminator loss: 0.6648803949356079, Generator loss: 0.8735774159431458
Epoch: [58/60], Batch: 29, Discriminator loss: 0.5546207427978516, Generator loss: 0.9859463572502136
Epoch: [58/60], Batch: 59, Discriminator loss: 0.7616094350814819, Generator loss: 0.7070373296737671
Epoch: [58/60], Batch: 89, Discriminator loss: 0.6576980352401733, Generator loss: 1.0522966384887695
Epoch: [59/60], Batch: 29, Discriminator loss: 0.714821994304657, Generator loss: 0.8552855253219604
Epoch: [59/60], Batch: 59, Discriminator loss: 0.7011092305183411, Generator loss: 0.6713104248046875
Epoch: [59/60], Batch: 89, Discriminator loss: 0.6015176177024841, Generator loss: 0.8367698788642883
generated airfoils
```



VAE

For VAE, the model chosen was a simple linear neural network with 4 linear layers in total. The neurons used were 256, 128, 64 as hidden laer neurons and the latent dim and airfoil dim for the input and output layers. There is an encoder and a decoder. The encoder goes from higher neuron to lower neuron and the decoder goes from lower neuron to higher neuron i.e., in the reverse order. With the encoder the mean and the log std were calculated and with decoder the final values were found. ReLu was used for the activation.

The hyper parameters used were learning rate is 0.001 the epochsa are 30 and for reconstruction loss the mse loss was used. For KL loss, it was calculated using the formula.

GAN

For GAN, the model chosen was a simple neural network with 3 layers. The neurons were 128, 64 as hidden layer neurons and the latent dim and airfoil dim was used for the input and output layers. There is a discriminator and generator in the model. activation functions used were Leaky Relu.

The hyper parameters used were learning rate is 0.0005 for both discrimator and generator the epochs used were 60. The adversarial loss function was BCELoss.

Compare the synthesized airfoils from VAE and GAN, describe your observation and give a brief explanation.

In VAE, the synthesized airfoils exhibit smoother transitions and fewer artifacts due to the reconstruction loss incorporated in the VAE. Whereas, the GAN images capture finer details and are sharper therefore producing high quality samples. This can be because VAE focuses on latent representation of the data while GAN directly generate realistic samples. The reconstruction loss term in VAE encourages the model to reconstruct input samples accurately. This leads to smoother but less detailed output. VAE produces more diverse samples as they learn from the underlying distribution. As the synthesized airfoils from VAEs might cover a wider range of variations present in the training data. Whereas in GAN there can be mode collapse there showing less diverse generations. Along with that, VAE, produce realistic samples that might struggle to capture details and GANs are known for highly realistic samples therefore, the airfoils generated by GANs exhibit high level of realism capturing both global structures and local details present in the training data. As Gan have generator-discriminator framework to learn the data distribution implicitly therefore helping them to produce high quality realistic samples.

In [ ]:

```
```