```python
import os

gen_dog_imgs = '/kaggle/working/generative-dog-images'
if not os.path.exists(gen_dog_imgs):
    os.makedirs(gen_dog_imgs)

dogs_dir = '/kaggle/working/dogs'
if not os.path.exists(dogs_dir):
    os.makedirs(dogs_dir)

# !mkdir /kaggle/working/generative-dog-images
!unzip /kaggle/input/generative-dog-images/all-dogs.zip -d /kaggle/working/generative-dog-images > /dev/null 2>&1
!unzip /kaggle/input/generative-dog-images/Annotation.zip -d /kaggle/working/generative-dog-images > /dev/null 2>&1
```

Generative Adversarial Network

Generative Adversarial Network (GAN) is a class of machine learning frameworks and a prominent framework for approaching generative AI. In a GAN, two neural networks, generator and discriminator, contest with each other in the form of a zero-sum game, where one agent's gain is another agent's loss. The core idea of a GAN is to train the generator to "fool" the discriminator rather than directly minimize the individual image distances, and the discriminator is indirectly trained to tell how realistic the generated images may seem. This way, both the generator and the discriminator are updated dynamically against each other to achieve realistic imitation to the original images.

GAN aims to learn to generate new data with the same statistics as the provided training set. A GAN trained on photographs can generate new iamges superficially authentic to human observers. While GAN is originally intended to be implemented in unsupervised learning, variations of GANs are developed into models suitable for semi-supervised and supervised learning purposes.

Dataset Overview

The Stanford Dogs dataset contains images of 120 breeds of dogs worldwide. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization. There are 20,580 images, out of which 12,000 are used for training and 8580 for testing. Class labels and bounding box annotations are provided for all the 12,000 images. In this study, training and testing images are not distinguished as they are repurposed into image generation.

```python
#%matplotlib inline
import torch
import torch.nn as nn
from torchvision import transforms
from torchvision.utils import make_grid
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
import random
from PIL import Image
import xml.etree.ElementTree as ET
```

```python
# Set random seed for reproducibility
manualSeed = 999
print("Random Seed: ", manualSeed)
random.seed(manualSeed)
torch.manual_seed(manualSeed)
torch.set_deterministic(True) # Needed for reproducible results
```

```
Random Seed:  999
```

```python
all_dogs_dir = '/kaggle/working/generative-dog-images/all-dogs'
annotation_dir = '/kaggle/working/generative-dog-images/Annotation'
```

```python
def bndbox_extraction(filename, square=False):

    root = ET.parse(filename).getroot()
    box = root.find('object/bndbox')
    xmin, ymin, xmax, ymax = map(int, [box.find(tag).text for tag in ['xmin', 'ymin', 'xmax', 'ymax']])

    if square:
        center_x, center_y = (xmin + xmax) // 2, (ymin + ymax) // 2
        size = max(xmax - xmin, ymax - ymin)
        xmin, xmax = center_x - size // 2, center_x + size // 2
        ymin, ymax = center_y - size // 2, center_y + size // 2

    return xmin, ymin, xmax, ymax
```

The purpose of excluding the "intruders" images in a Dog Image Generation project using DCGAN (Deep Convolutional Generative Adversarial Network) can be explained as follows:

Consistency and Purity of Dataset: By excluding images labeled as "intruders," you ensure that your training data consists only of clear, relevant examples of what you want your model to learn - in this case, dogs. This helps in maintaining the purity of the dataset, focusing solely on the intended subject matter.

Reducing Noise and Improving Model Learning: "Intruders" might include images that are not dogs or are of poor quality, mislabeled, or in some way not representative of the target class (dogs). Including these could introduce noise, making it harder for the model to learn the specific features of dogs. This exclusion helps in reducing such noise, allowing the model to learn more accurate and detailed features of dogs.

Enhancing Model Performance: A cleaner dataset leads to better performance. By removing irrelevant or ambiguous images, you increase the likelihood that the generator and discriminator in the DCGAN will learn to produce and identify dog images with higher fidelity and accuracy. This also potentially reduces the training time as the model doesn't have to waste computational resources on learning from or distinguishing irrelevant images.

Ref: https://www.kaggle.com/korovai/dogs-images-intruders-extraction

```python
intruders = [
    'n02088238_10870_0',
    'n02088466_6901_1', 'n02088466_6963_0', 'n02088466_9167_0',
    'n02088466_9167_1', 'n02088466_9167_2',
    'n02089867_2221_0', 'n02089867_2227_1',
    'n02089973_1132_3', 'n02089973_1352_3', 'n02089973_1458_1',
    'n02089973_1799_2', 'n02089973_2791_3', 'n02089973_4055_0',
    'n02089973_4185_1', 'n02089973_4185_2',
    'n02090379_4673_1', 'n02090379_4875_1',
    'n02090622_7705_1', 'n02090622_9358_1', 'n02090622_9883_1',
    'n02090721_209_1', 'n02090721_1222_1', 'n02090721_1534_1',
    'n02090721_1835_1', 'n02090721_3999_1', 'n02090721_4089_1',
    'n02090721_4276_2',
    'n02091032_722_1', 'n02091032_745_1', 'n02091032_1773_0',
    'n02091032_9592_0',
    'n02091134_2349_1', 'n02091134_14246_2',
    'n02091244_583_1', 'n02091244_2407_0', 'n02091244_3438_1',
    'n02091244_5639_1', 'n02091244_5639_2',
    'n02091467_473_0', 'n02091467_4386_1', 'n02091467_4427_1',
    'n02091467_4558_1', 'n02091467_4560_1',
    'n02091635_1192_1', 'n02091635_4422_0',
    'n02091831_1594_1', 'n02091831_2880_0', 'n02091831_7237_1',
    'n02092002_1551_1', 'n02092002_1937_1', 'n02092002_4218_0',
    'n02092002_4596_0', 'n02092002_5246_1', 'n02092002_6518_0',
    'n02093256_1826_1', 'n02093256_4997_0', 'n02093256_14914_0',
    'n02093428_5662_0', 'n02093428_6949_1'
]
intruders = [filename + '.jpg' if not filename.endswith('.jpg') else filename for filename in intruders]
len(intruders)
```

```
60
```

Compare the number of images in the dataset before and after the intruder exclusion

```python
dogs_count = 0
for breed in os.listdir(annotation_dir):
    for dog in os.listdir(os.path.join(annotation_dir, breed)):
        #print(dog)
        bndbox = bndbox_extraction(os.path.join(annotation_dir, breed, dog), square=True)
        jpg_name = os.path.join(all_dogs_dir, dog+'.jpg')
        intruders_join = '\t'.join(intruders)
        if os.path.exists(jpg_name):
            if dog not in intruders_join:
                img = Image.open(jpg_name).crop(bndbox)
                img.save(os.path.join(dogs_dir, dog+'.jpg'))
                dogs_count+=1
print('number of dogs in the original dataset:', len(os.listdir(dogs_dir))+len(intruders))
print('number of dogs in the dataset excluding intruders:',dogs_count)
```

```
number of dogs in the original dataset: 20580
number of dogs in the dataset excluding intruders: 20520
```

```python
class LoadDataset(Dataset):
    def __init__(self, data_dir, transforms=None):
        self.files = [os.path.join(data_dir, file) for file in os.listdir(data_dir)]
        self.transforms = transforms

    def __len__(self):
        return len(self.files)

    def __getitem__(self, index):
        img = Image.open(self.files[index])
        if self.transforms is not None:
            img = self.transforms(img)
        return img
```

```python
img_size = (64, 64)
image_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(img_size),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

batch_size = 128
trainloader = DataLoader(
    LoadDataset(data_dir=dogs_dir, transforms=image_transforms),
    batch_size = batch_size,
    shuffle = True,
    num_workers = 3,
)
```

DCGAN

DCGAN (Deep Convolutional GAN) is a generative adversarial network architecture using deep convolutional neural networks. It is specialized for generating realistic images, mainly square images, in computer vision field. DCGAN can learn and capture detailed features in images of the original training dataset to generate realistic fake images hardly distinguishable by human eyes.

A great DCGAN tutorial can be founded in: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

```python
class Generator(nn.Module):
    def __init__(self, z_channels, out_channels=3):
        super(Generator, self).__init__()

        convs = []
        channels = [z_channels, 1024, 512, 256, 128, 64]
        for i in range(1, len(channels)):
            convs.append(nn.ConvTranspose2d(channels[i-1], channels[i], 2, stride=2, bias=False))
            convs.append(nn.BatchNorm2d(channels[i]))
            convs.append(nn.LeakyReLU(0.1, inplace=True))
        convs.append(nn.ConvTranspose2d(channels[-1], out_channels, 2, stride=2, bias=False))
        convs.append(nn.Tanh())

        self.convs = nn.Sequential(*convs)

    def forward(self, x):
        return self.convs(x)
```

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        channels = [3, 64, 128, 256, 512]
        convs = []
        for i in range(1, len(channels)):
            convs.append(nn.Conv2d(channels[i-1], channels[i], 3, padding=1, stride=2, bias=False))
            if i != 1:
                convs.append(nn.BatchNorm2d(channels[i]))
            convs.append(nn.LeakyReLU(0.2, inplace=True))

        convs.append(nn.Conv2d(channels[-1], 1, 4, bias=False))
        convs.append(nn.Sigmoid())

        self.convs = nn.Sequential(*convs)

    def forward(self, x):
        x = self.convs(x)
        return x.view(-1)
```

```python
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

```python
z_channels = 100
G = Generator(z_channels, 3)
G.apply(weights_init)
D = Discriminator()
D.apply(weights_init)
criterion = nn.BCELoss()

cuda = torch.cuda.is_available()
if cuda:
    print('Use GPU')
    G = G.cuda()
    D = D.cuda()
    criterion = criterion.cuda()
else:
    print('No GPU')
```

```
Use GPU
```

```python
lr = 0.0004
b1,b2 = 0.6,0.999

optimizerG = torch.optim.AdamW(G.parameters(), lr=lr, betas=(b1, b2))
optimizerD = torch.optim.AdamW(D.parameters(), lr=lr, betas=(b1, b2))
```

```python
fixed_noise = torch.normal(0, 0.1, size=(64, z_channels, 1, 1))
if cuda:
    fixed_noise = fixed_noise.cuda()

epoches = 20
generate_imgs = []
G_losses, D_losses = [],[]
for epoch in range(epoches):
    for i, img in enumerate(trainloader):
        z = torch.normal(0, 0.1, size=(img.size(0), z_channels, 1, 1))
        real = torch.ones(img.size(0))
        fake = torch.zeros(img.size(0))
        if cuda:
            img, z = img.cuda(), z.cuda()
            real, fake = real.cuda(), fake.cuda()

        # train D
        D.zero_grad()
        loss_real = criterion(D(img), real)
        loss_real.backward()

        fake_img = G(z)
        loss_fake = criterion(D(fake_img.detach()), fake)
        loss_fake.backward()

        loss_D = (loss_real + loss_fake) / 2

        optimizerD.step()

        # train G
        G.zero_grad()
        loss_G = criterion(D(fake_img), real)
        loss_G.backward()
        optimizerG.step()

    with torch.no_grad():
        noise_img = G(fixed_noise)
        generate_imgs.append(noise_img)
        print(f'[Epoch {epoch+1}/{epoches}] [G loss: {loss_G.item()}] [D loss: {loss_D.item()} | loss_real: {loss_real.item()} loss_fake: {loss_fake.item()}]')
        G_losses.append(loss_G.item())
        D_losses.append(loss_D.item())
```

```
[Epoch 1/20] [G loss: 10.865691184997559] [D loss: 0.2619589567184448 | loss_real: 0.054818179458379745 loss_fake: 0.46909973025321 96]
[Epoch 2/20] [G loss: 6.498304843902588] [D loss: 0.09412280470132828 | loss_real: 0.055279213935136795 loss_fake: 0.13296639919281006]
[Epoch 3/20] [G loss: 4.126104354858398] [D loss: 0.2906491458415985 | loss_real: 0.3250446021556854 loss_fake: 0.2562536895275116]
[Epoch 4/20] [G loss: 5.790511131286621] [D loss: 0.13597875833511353 | loss_real: 0.053666938096284866 loss_fake: 0.2182905673980713]
[Epoch 5/20] [G loss: 4.9497151374816895] [D loss: 0.046063609421253204 | loss_real: 0.05700443312525749 loss_fake: 0.03512278199195862]
[Epoch 6/20] [G loss: 6.157364845275879] [D loss: 0.071864552795887 | loss_real: 0.131789430975914 loss_fake: 0.011939677409827 71]
[Epoch 7/20] [G loss: 7.788684368133545] [D loss: 0.18688279390335083 | loss_real: 0.07670976221561432 loss_fake: 0.29705584049224854]
[Epoch 8/20] [G loss: 4.8096537590026855] [D loss: 0.1546897292137146 | loss_real: 0.13383172452449799 loss_fake: 0.1755477488040924]
[Epoch 9/20] [G loss: 6.225575923919678] [D loss: 0.09683665633201599 | loss_real: 0.12571518123149872 loss_fake: 0.06795813143253326]
[Epoch 10/20] [G loss: 7.191747188568115] [D loss: 0.04842983931303024 | loss_real: 0.041421569883823395 loss_fake: 0.05543811246752739]
[Epoch 11/20] [G loss: 9.025453567504883] [D loss: 0.2669317126274109 | loss_real: 0.08311474323272705 loss_fake: 0.45074865221977234]
[Epoch 12/20] [G loss: 3.9974205493927] [D loss: 0.06524422019720078 | loss_real: 0.08067264407873154 loss_fake: 0.04981579631567001]
[Epoch 13/20] [G loss: 4.709574222564697] [D loss: 0.23730894923210144 | loss_real: 0.3895694315433502 loss_fake: 0.08504845947027206]
[Epoch 14/20] [G loss: 7.0576677322387695] [D loss: 0.08900472521781921 | loss_real: 0.03240731731057167 loss_fake: 0.14560213685035706]
[Epoch 15/20] [G loss: 12.919687271118164] [D loss: 0.5925876498222351 | loss_real: 0.006958964746445417 loss_fake: 1.1782163381576538]
[Epoch 16/20] [G loss: 10.298261642456055] [D loss: 0.32368263602256775 | loss_real: 0.0043887184001505375 loss_fake: 0.6429765820503235]
[Epoch 17/20] [G loss: 4.768966197967529] [D loss: 0.11061170697212219 | loss_real: 0.05123958736658096 loss_fake: 0.16998383402824402]
[Epoch 18/20] [G loss: 8.470938682556152] [D loss: 0.17758509516716003 | loss_real: 0.013689598999917507 loss_fake: 0.34148058295249 94]
[Epoch 19/20] [G loss: 7.244987487792969] [D loss: 0.07223927229642868 | loss_real: 0.038334932178258896 loss_fake: 0.10614361613988876]
[Epoch 20/20] [G loss: 8.073629379272461] [D loss: 0.13738255202770233 | loss_real: 0.03298019990324974 loss_fake: 0.24178491532802582]
```

```python
fig = plt.figure(figsize=(15,10))
x_epoches = list(range(1,epoches+1))
plt.plot(x_epoches, G_losses, 'b-o', x_epoches, D_losses, 'r-x')
plt.legend(['Generator Loss','Discriminator Loss'])
plt.title('Optimizer Loss vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

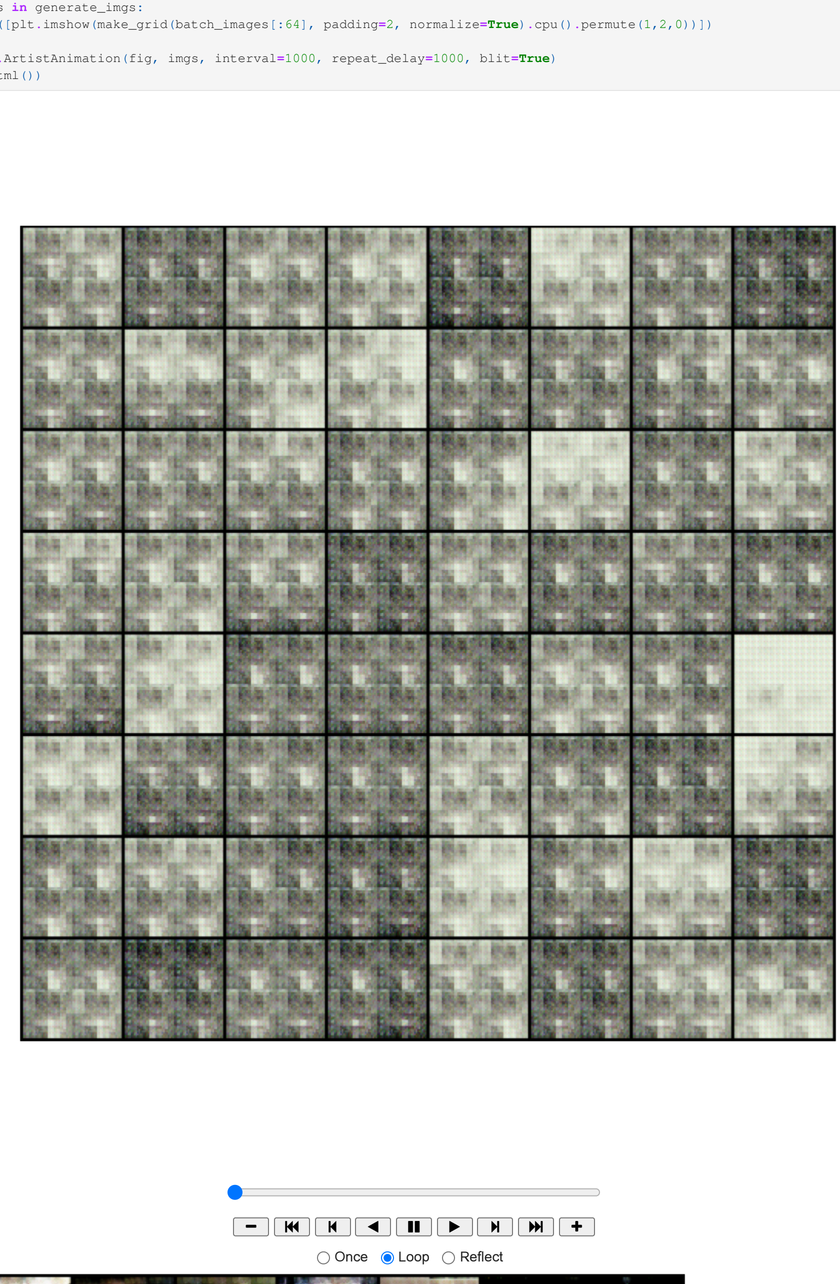
```
Text(0, 0.5, 'Loss')
```

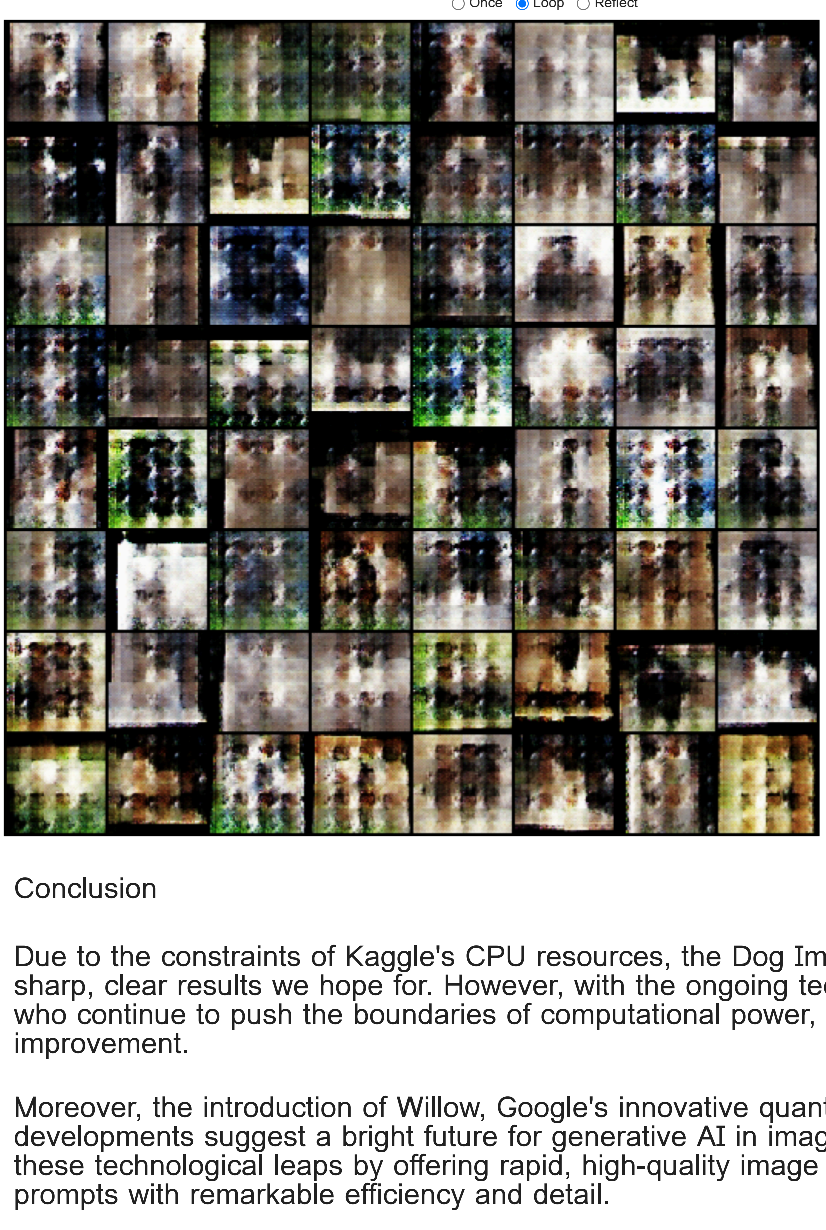

```python
import matplotlib.animation as animation
from IPython.display import HTML

fig = plt.figure(figsize=(15,15))
plt.axis("off")

imgs = []
for batch_images in generate_imgs:
    imgs.append([plt.imshow(make_grid(batch_images[:64], padding=2, normalize=True).cpu().permute(1,2,0))])

ani = animation.ArtistAnimation(fig, imgs, interval=1000, repeat_delay=1000, blit=True)
HTML(ani.to_jshtml())
```





Conclusion

Due to the constraints of Kaggle's CPU resources, the Dog Image Generation project using DCGAN has not yet yielded the sharp, clear results we hope for. However, with the ongoing technological advancements from GPU manufacturers like NVIDIA, who continue to push the boundaries of computational power, the landscape of image generation is set for significant improvement.

Moreover, the introduction of Willow, Google's innovative quantum chip, adds an exciting dimension to this field. These developments suggest a bright future for generative AI in image creation. Indeed, companies like xAI are already capitalizing on these technological leaps by offering rapid, high-quality image generation services to the public at no cost, responding to user prompts with remarkable efficiency and detail.