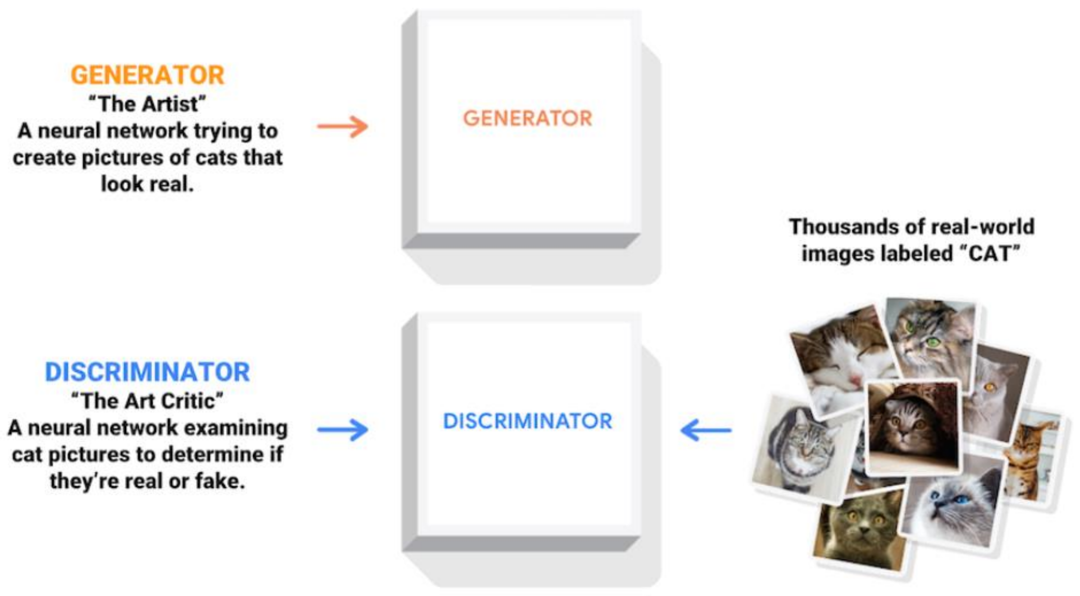
Generative Adversarial Network – Flower

This project demonstrates how to generate images of flowers using a Deep Convolutional Generative Adversarial Network (DCGAN). The code is written using the Keras Sequential API with a tf.GradientTape training loop.

What are GANs?

Generative Adversarial Networks (GANs) are one of the most interesting ideas in computer science today. Two models are trained simultaneously by an adversarial process. A *generator* ("the artist") learns to create images that look real, while a *discriminator* ("the art critic") learns to tell real images apart from fakes.


GENERATOR
"The Artist"
A neural network trying to create pictures of cats that look real.
GENERATOR
Thousands of real-world images labeled "CAT"
DISCRIMINATOR
"The Art Critic"
A neural network examining cat pictures to determine if they're real or fake.
DISCRIMINATOR

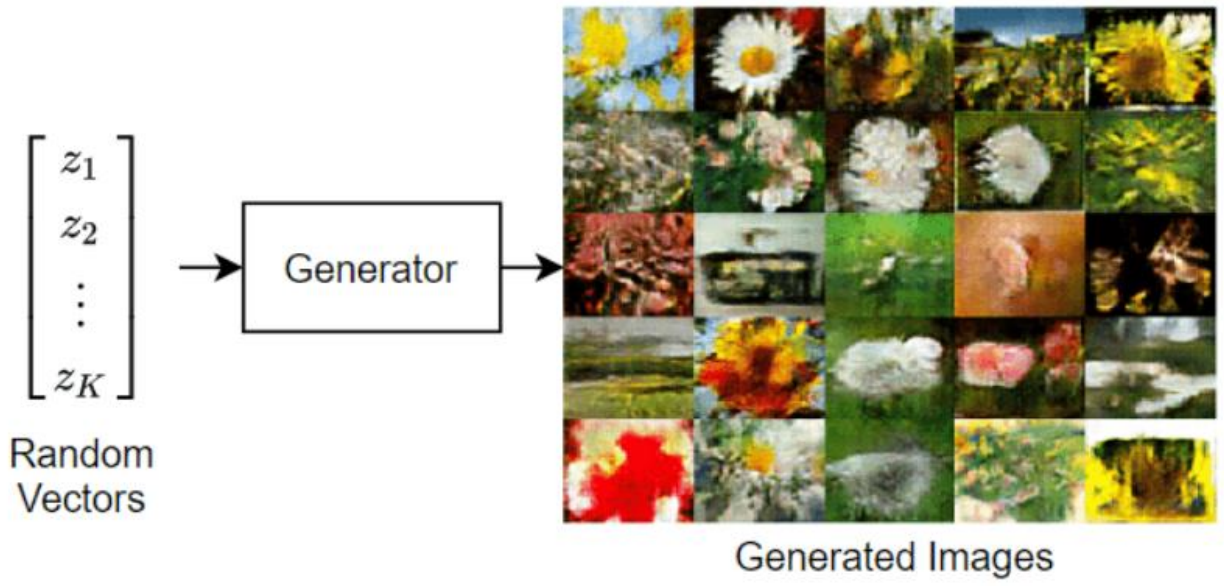
During training, the *generator* progressively becomes better at creating images that look real, while the *discriminator* becomes better at telling them apart. The process reaches equilibrium when the *discriminator* can no longer distinguish real images from fakes.

This notebook demonstrates this process on the Flowers dataset, which contains images of flowers. The following images produced by the *generator* as it was trained for 500 epochs. The images begin as random noise, and increasingly resemble flowers over time.

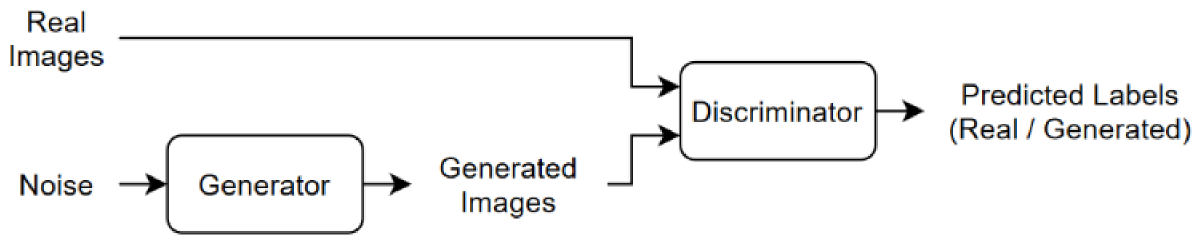
A GAN consists of two networks that train together:

1. Generator — Given a vector of random values (latent inputs) as input, this network generates data with the same structure as the training data.
2. Discriminator — Given batches of data containing observations from both the training data, and generated data from the generator, this network attempts to classify the observations as "real" or "generated".

This diagram illustrates the generator network of a GAN generating images from vectors of random inputs.



This diagram illustrates the structure of a GAN.



To train a GAN, train both networks simultaneously to maximize the performance of both:

- Train the generator to generate data that "fools" the discriminator.
- Train the discriminator to distinguish between real and generated data.

To optimize the performance of the generator, maximize the loss of the discriminator when given generated data. That is, the objective of the generator is to generate data that the discriminator classifies as "real".

To optimize the performance of the discriminator, minimize the loss of the discriminator when given batches of both real and generated data. That is, the objective of the discriminator is to not be "fooled" by the generator.

Ideally, these strategies result in a generator that generates convincingly realistic data and a discriminator that has learned strong feature representations that are characteristic of the training data.

Load and Preparing data

Download and extract the Flowers data set [1].

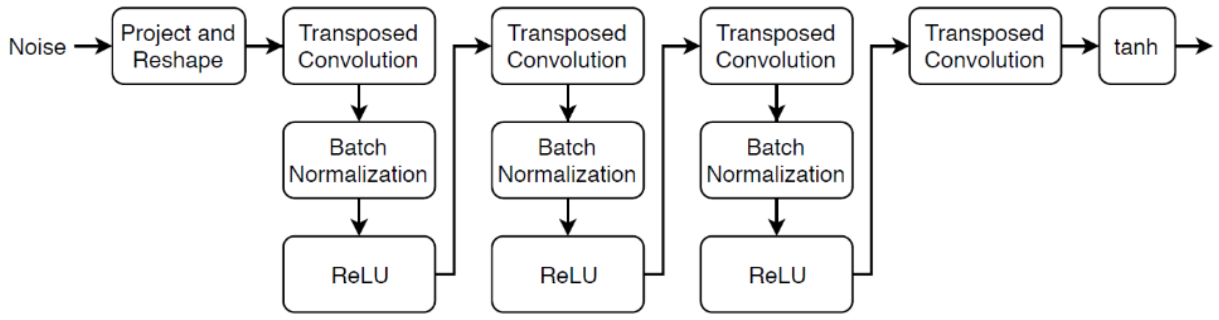
For data preparation, including pre-processing and data augmentation, the following steps were performed on the data:

- Resize
- Batch
- Normalization
- Shuffle
- Random Flip Horizontal

Define Generative Adversarial Network

Define Generator Network

Define the following network architecture, which generates images from random vectors.



This network:

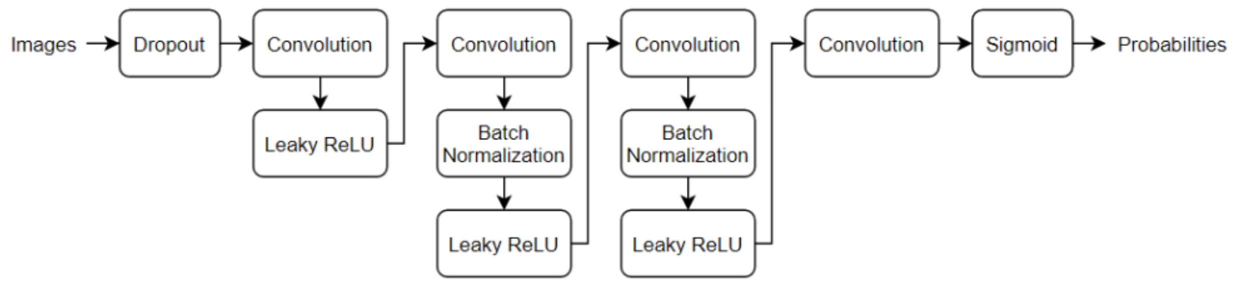
- Converts the random vectors of size 100 to 4-by-4-by-512 arrays using a project and reshape operation.
- Upscales the resulting arrays to 64-by-64-by-3 arrays using a series of transposed convolution layers with batch normalization and ReLU layers.

Specify the following network properties.

- For the transposed convolution layers, specify 5-by-5 filters with a decreasing number of filters for each layer and a stride of 2.
- For the final transposed convolution layer, specify three 5-by-5 filters corresponding to the 3 RGB channels of the generated images, and the output size of the previous layer.
- At the end of the network, include a tanh layer.

Define Discriminator Network

Define the following network, which classifies real and generated 64-by-64 images.



Note: In the above structure, a Conv, Batch Norm, Leaky ReLU block is not displayed incorrectly.

Create a network that takes 64-by-64-by-3 images and returns a scalar prediction score using a series of convolution layers with batch normalization and leaky ReLU layers. Add noise to the input images using dropout.

- For the dropout layer, specify a dropout probability of 0.5.
- For the convolution layers, specify 5-by-5 filters with an increasing number of filters for each layer. Also specify a stride of 2 and padding of the output.
- For the leaky ReLU layers, specify a scale of 0.2.
- To output the probabilities in the range [0, 1], specify a convolutional layer with one 4-by-4 filter followed by a sigmoid layer.

The rest of the settings related to optimization and loss parameters are in the notebook.

In this project, the GAN model produced flower images in the following 4 modes:

1. Batch size equal to 128 and use the dense layer as the last layer of the discriminator model

2. Batch size equal to 64 and use the dense layer as the last layer of the discriminator model

3. Batch size equal to 128 and not using the dense layer as the last layer of the discriminator model

4. Batch size equal to 64 and not using the dense layer as the last layer of the discriminator model

Some explanations about the folders in the project's GitHub repo:

In the model_checkpoints folder, there is a text file named “ckpt-10.txt”, inside which there is a link to download the model checkpoint in epoch 500. (GitHub does not allow uploading files larger than 100 MB!)

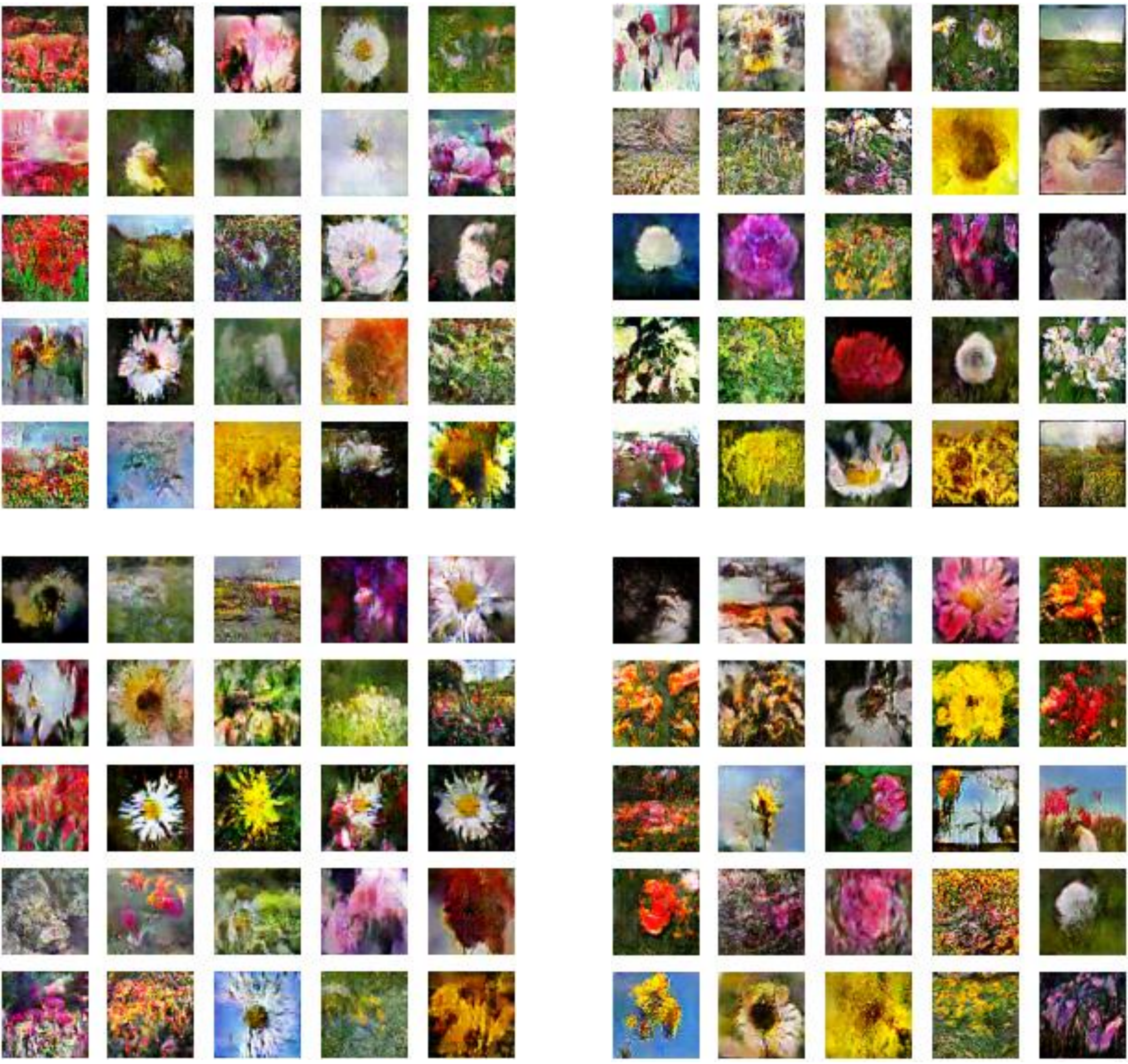
The results folder contains the last fifty photos produced by 4 models along with a gif file of the last 50 photos produced.

In the src folder, there is a Jupyternotebook related to the project codes, in which, in addition to the code, additional explanations for each section are given.

Results

It took about 1 hour and 50 minutes to run the model on the GoogleColab with GPU.

Generated flower Images in epoch equal to 500: (1. 64-with, 2. 64-without, 3. 128-with, 4. 128-without) (clockwise!)



We can see that some of the generated flower images are very close to reality and our model has generated them well.

Among the 4 different modes, it seems that the photos generated by the model with a batch-size of 128 and the use of the dense layer as the last layer of the discriminator model are slightly better.