GAN Implementation for Animal Images Generation

Objective: To build a Deep Learning model to generate new animal images using GAN

Data: The animal image dataset contains 14630 images of cats, dogs and wild animals

Data Preparation:

- Batch size of 128
- Image Resizing to (64, 64, 3)
- Cast image dtype to Float32
- Image Normalization to get the pixel intensity in [-1,1]

And we do shuffling to randomly reorder data before each epoch and use of buffer to prefetch inputs from input dataset before they are requested

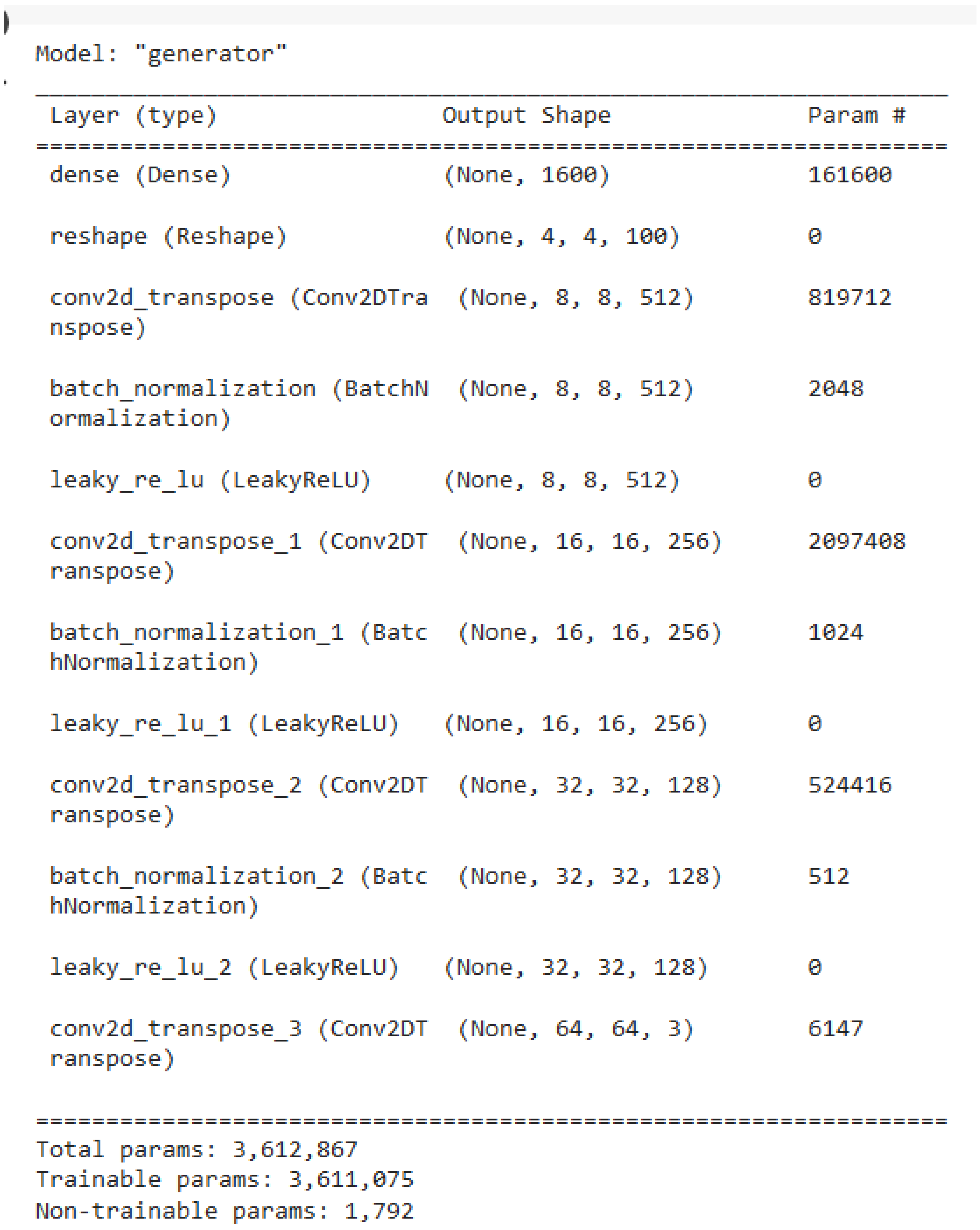
Model Design:

Generator’s purpose is to progressively become better at creating images that look real and generator’s purpose is to get better at telling them apart.

I have implemented **DCGAN** model to generate images.

Generator:

The generator uses Conv2DTranspose (up sampling) layers to produce images from random noise. We start with a Dense Layer that takes this input that is random noise (LATENT_DIM = 100). We keep up sampling with Conv2DTranspose, using Batch Normalization and LeakyReLU as activation function. Leaky ReLU is beneficial when the data has a lot of noise, because it can provide a non-zero output for negative inputs and avoid discarding potentially important information. And use tanh activation in last layer to get values between [-1,1]

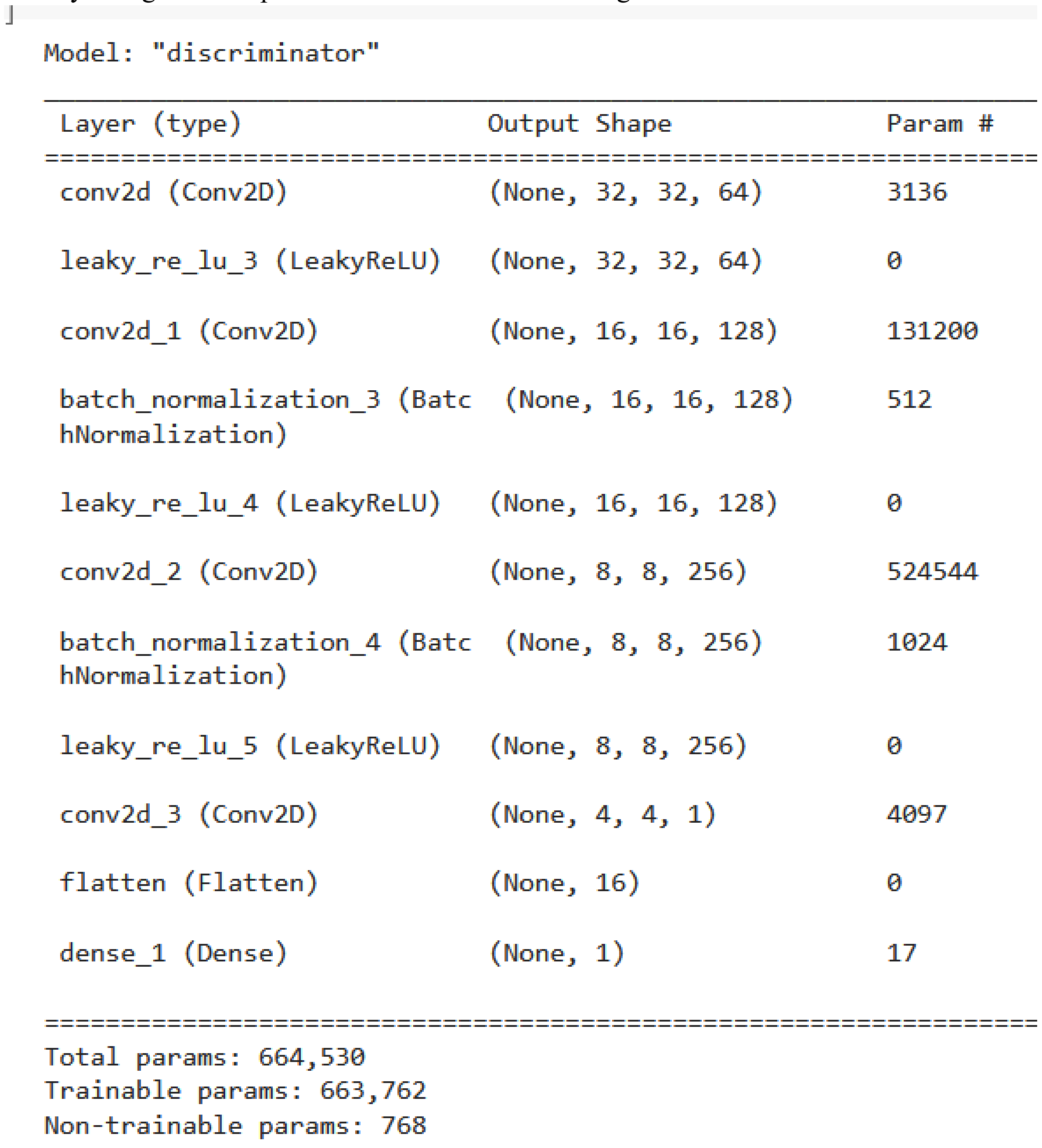


```
Model: "generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1600)              161600

 reshape (Reshape)           (None, 4, 4, 100)         0

 conv2d_transpose (Conv2DTra  (None, 8, 8, 512)        819712
 nspose)

 batch_normalization (BatchN  (None, 8, 8, 512)        2048
 ormalization)

 leaky_re_lu (LeakyReLU)     (None, 8, 8, 512)         0

 conv2d_transpose_1 (Conv2DT  (None, 16, 16, 256)      2097408
 ranspose)

 batch_normalization_1 (Batc  (None, 16, 16, 256)      1024
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 256)       0

 conv2d_transpose_2 (Conv2DT  (None, 32, 32, 128)      524416
 ranspose)

 batch_normalization_2 (Batc  (None, 32, 32, 128)      512
 hNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 32, 32, 128)       0

 conv2d_transpose_3 (Conv2DT  (None, 64, 64, 3)        6147
 ranspose)

=================================================================
Total params: 3,612,867
Trainable params: 3,611,075
Non-trainable params: 1,792
```

Discriminator:

The Discriminator is a CNN based image classifier.

We use the discriminator to classify the generated images as real or fake. The model gets trained to output positive values for real images and negative values for fake images. We use Conv2D layers to the images from the generator and flatten it and use sigmoid function in the last layer to get the output values for fake and real images.



```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        3136

 leaky_re_lu_3 (LeakyReLU)   (None, 32, 32, 64)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       131200

 batch_normalization_3 (Batc  (None, 16, 16, 128)      512
 hNormalization)

 leaky_re_lu_4 (LeakyReLU)   (None, 16, 16, 128)       0

 conv2d_2 (Conv2D)           (None, 8, 8, 256)         524544

 batch_normalization_4 (Batc  (None, 8, 8, 256)        1024
 hNormalization)

 leaky_re_lu_5 (LeakyReLU)   (None, 8, 8, 256)         0

 conv2d_3 (Conv2D)           (None, 4, 4, 1)           4097

 flatten (Flatten)           (None, 16)                0

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 664,530
Trainable params: 663,762
Non-trainable params: 768
```

Model Training:

We define loss functions and optimizers for both the models.

Discriminator Loss:

This method tells us how well the discriminator can differentiate between the real images and fakes. This compares the discriminators predictions on real images to an array of 1s and the discriminators predictions on fake (generated) images to 0's. We use BinaryCrossEntropy loss function for both the models

Generator Loss:

Generator loss tell us how well it was able to trick the discriminator. If the generator performs well it means the discriminator will classify the fake images as real. We compare the discriminators decisions on generated images to 1s.

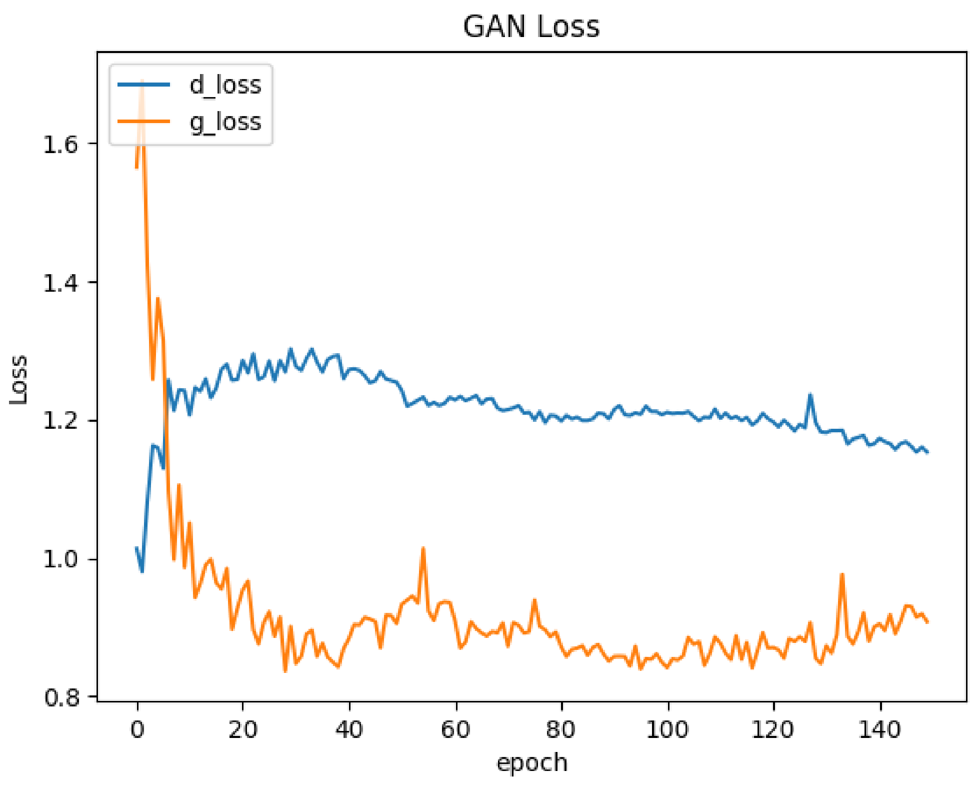
Optimizers:

Both Generator and discriminator use separate optimizers. We use Adam optimizer with learning rate of 0.0002.

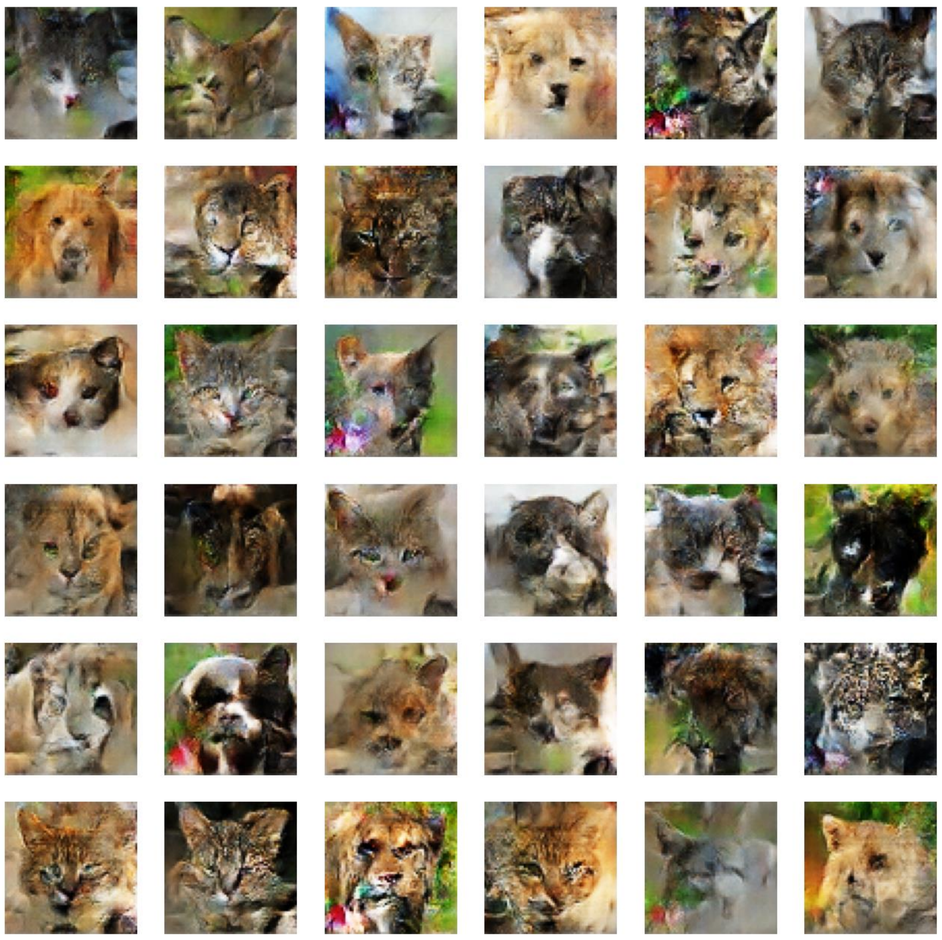
Training Loop:

Model trained on 150 epochs. Training starts with generator getting random noise as input. This noise is used to produce an image. The discriminator is then used to classify real images (drawn from the training set) and fake images (produced by the generator). Then we calculate the loss and use the gradients to update the generator and discriminator.

Generator vs Discriminator loss:



Sample Generated Images:



Model Deployment:

First save the trained the GAN model and its weights.

Upload the saved trained model files to Google Cloud Storage (GCS) for easy access during deployment.

We can use Google Cloud Functions for serverless deployment with minimal setup.

Deploy the GAN model using Google Cloud Functions and loading the model from Google Cloud Storage.

Implement an API to interact with deployed GAN model. We can use Flask or FAST API to set up the API. Implement code to handle incoming API requests, process inputs and generate images using GAN model.

Google Drive Link: (contains Dataset images, generated images, Colab notebook and model weights)
https://drive.google.com/drive/folders/1VvAGNeYiecgmoDuqUOFzDxrtrMJ1aWxM?usp=sharing

Github Link :

https://github.com/ujjawalsingh10/DCGAN-for-Image-Generation/