MRI-STYLE-TRANSFER-USING-GAN

To build a Generative adversarial model(modified U-Net) which can generate artificial MRI images of different contrast levels from existing MRI scans.

Data understanding:

The dataset used for this task (`MRI+T1_T2+Dataset.RAR`) contains T1 and T2 MRI images in RGB format with a base size of 217 x 181 pixels. To ensure reliable results, the images need to be preprocessed before further analysis.

Since the T1 and T2 images in the dataset are unpaired (i.e., they don’t correspond to the same anatomical regions or patients), we will employ a CycleGAN model. CycleGAN is well-suited for this type of problem as it allows for image-to-image translation even without paired datasets.

The model will learn to generate T2-weighted images from T1-weighted images and vice-versa, enhancing the dataset’s diagnostic potential by creating synthetic contrast variations from the given scans.

Project Description:

-Misdiagnosis in healthcare is a significant concern with serious consequences for patients. As radiologists face increasing workloads, there is an urgent need for Artificial Intelligence to support them in making accurate and timely decisions.

- Magnetic Resonance Imaging (MRI) is a key imaging technology which offers superb soft tissue contrast with different contrast mechanisms such as T1 weighted and T2 weighted. A radiologist often needs multi contrast images to arrive at the right decision but is often cost prohibitive.
- Magnetic Resonance Imaging (MRI) is a vital medical imaging technique that provides excellent soft tissue contrast through various mechanisms, such as T1-weighted and T2-weighted scans. However, radiologists often require multiple contrast images for accurate diagnosis, which can be challenging and expensive to obtain.
- CycleGAN can be utilized to translate the style of one MRI image into another, enhancing the interpretation of scanned images. This approach allows the generation of T2-weighted images from T1-weighted scans and vice versa, providing additional perspectives for more accurate diagnosis.

Project Pipeline:

In this notebook, we will create T2 weighted images from T1 weighted MRI image and vice-versa.

Below sequence is followed for given problem statement:

1. Importing Libraries

2. Data Understanding

- Basic EDA

- Data loading
- Sample data visualization

3. Image Processing

- Data normalization
- Data resizing
- Data reshaping
- Batch and shuffling of data
- Data Augmentation

4. Model Building

- Instance Normalization
- Downsampling, Upsampling and Unet
- Generator Building using Unet
- Discriminator Building

5. Model Training

- Declare Loss type
- Calculate Discriminator Loss
- Calculate Generator Loss
- Cycle Loss
    - Identity Loss
- Optimizer
- Checkpoint Initialization
- Training Flow

6. Testing Model

7. Generating a GIF

weighted T1 dataset

http://biomedic.doc.ic.ac.uk/brain-development/downloads/IXI/IXI-T1.tar

weighted T2 dataset

http://biomedic.doc.ic.ac.uk/brain-development/downloads/IXI/IXI-T2.tar

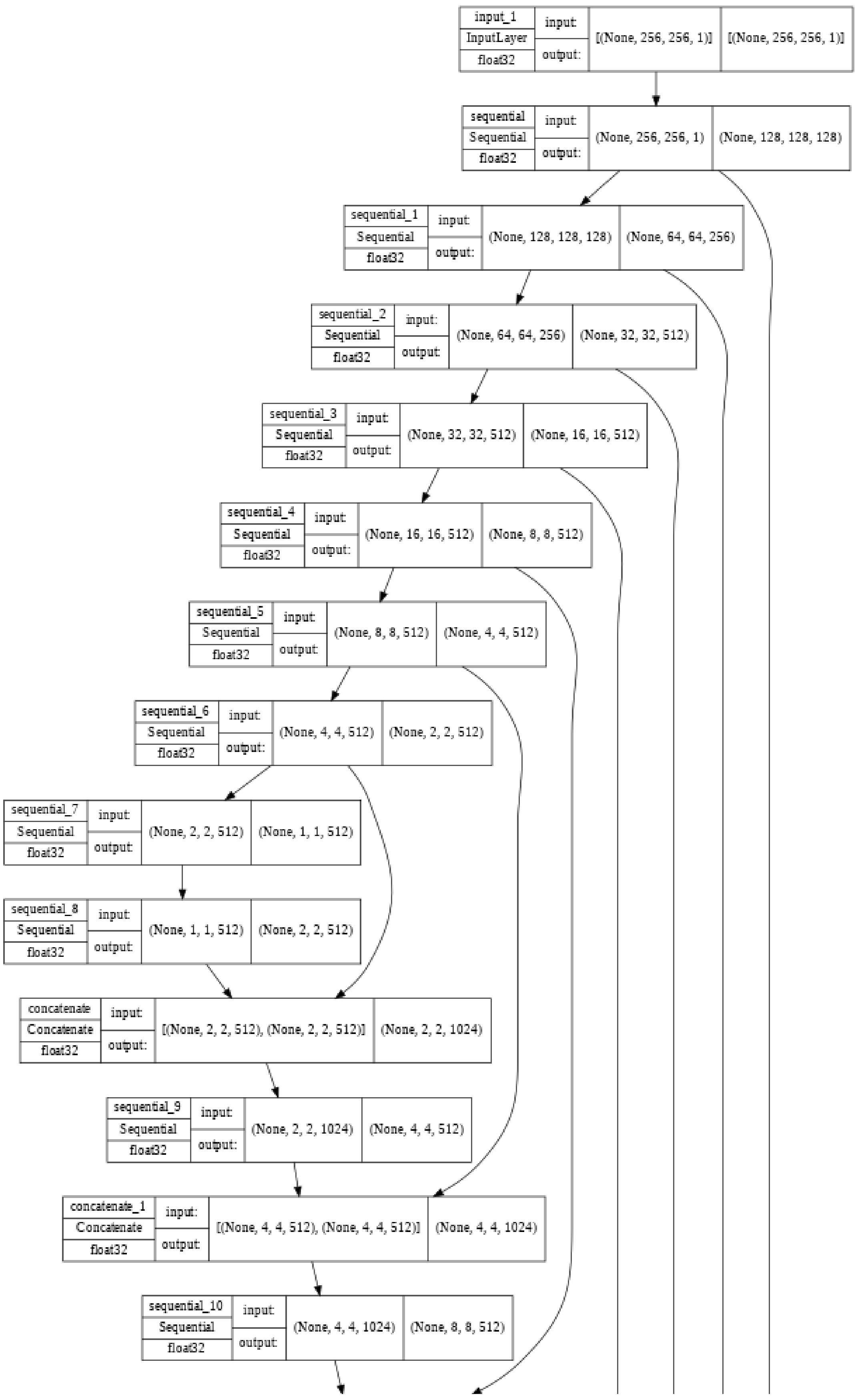
Generator

Generator follows *U-net architecture* which is fairly simple architecture; however, to create the skip connections between the encoder and decoder, we will need to concatenate some layers. So the Keras Functional API is most appropriate for this purpose.

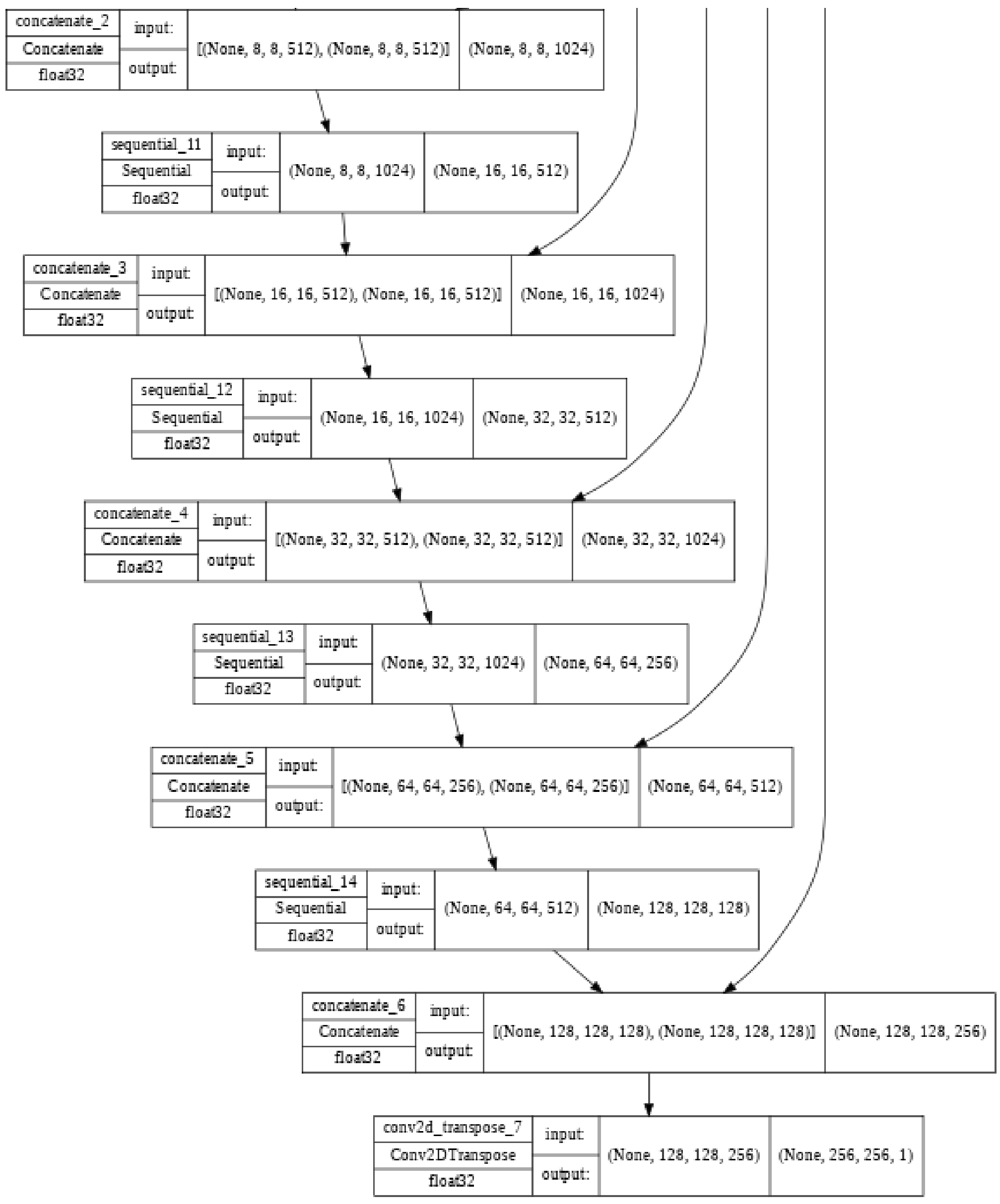
U-Net is able to localize and distinguish borders by classifying on every pixel, with both input and output images being of same size

Here, Generator generates synthetic images(T1 to T2 and vice versa) using given random normal initializer

To summarize the U-net generator the input image (256 x 256 x 1) is increased to (1 x 1 x 512) and brought back to (256 x 256 x 1) using encoder [which learns the image irrespective of its loaction] and decoder [details are captured here] using skip connections to get precise locations, feature maps from corresponding encoder level is concatenated with output of transposed convolutional layers



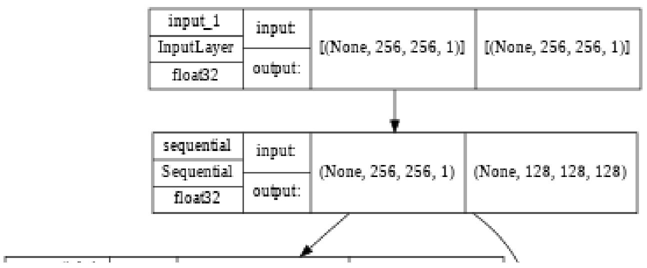
input_1 | InputLayer | float32 | input: | output: | [(None, 256, 256, 1)] | [(None, 256, 256, 1)]
sequential | Sequential | float32 | input: | output: | (None, 256, 256, 1) | (None, 128, 128, 128)
sequential_1 | Sequential | float32 | input: | output: | (None, 128, 128, 128) | (None, 64, 64, 256)
sequential_2 | Sequential | float32 | input: | output: | (None, 64, 64, 256) | (None, 32, 32, 512)
sequential_3 | Sequential | float32 | input: | output: | (None, 32, 32, 512) | (None, 16, 16, 512)
sequential_4 | Sequential | float32 | input: | output: | (None, 16, 16, 512) | (None, 8, 8, 512)
sequential_5 | Sequential | float32 | input: | output: | (None, 8, 8, 512) | (None, 4, 4, 512)
sequential_6 | Sequential | float32 | input: | output: | (None, 4, 4, 512) | (None, 2, 2, 512)
sequential_7 | Sequential | float32 | input: | output: | (None, 2, 2, 512) | (None, 1, 1, 512)
sequential_8 | Sequential | float32 | input: | output: | (None, 1, 1, 512) | (None, 2, 2, 512)
concatenate | Concatenate | float32 | input: | output: | [(None, 2, 2, 512), (None, 2, 2, 512)] | (None, 2, 2, 1024)
sequential_9 | Sequential | float32 | input: | output: | (None, 2, 2, 1024) | (None, 4, 4, 512)
concatenate_1 | Concatenate | float32 | input: | output: | [(None, 4, 4, 512), (None, 4, 4, 512)] | (None, 4, 4, 1024)
sequential_10 | Sequential | float32 | input: | output: | (None, 4, 4, 1024) | (None, 8, 8, 512)

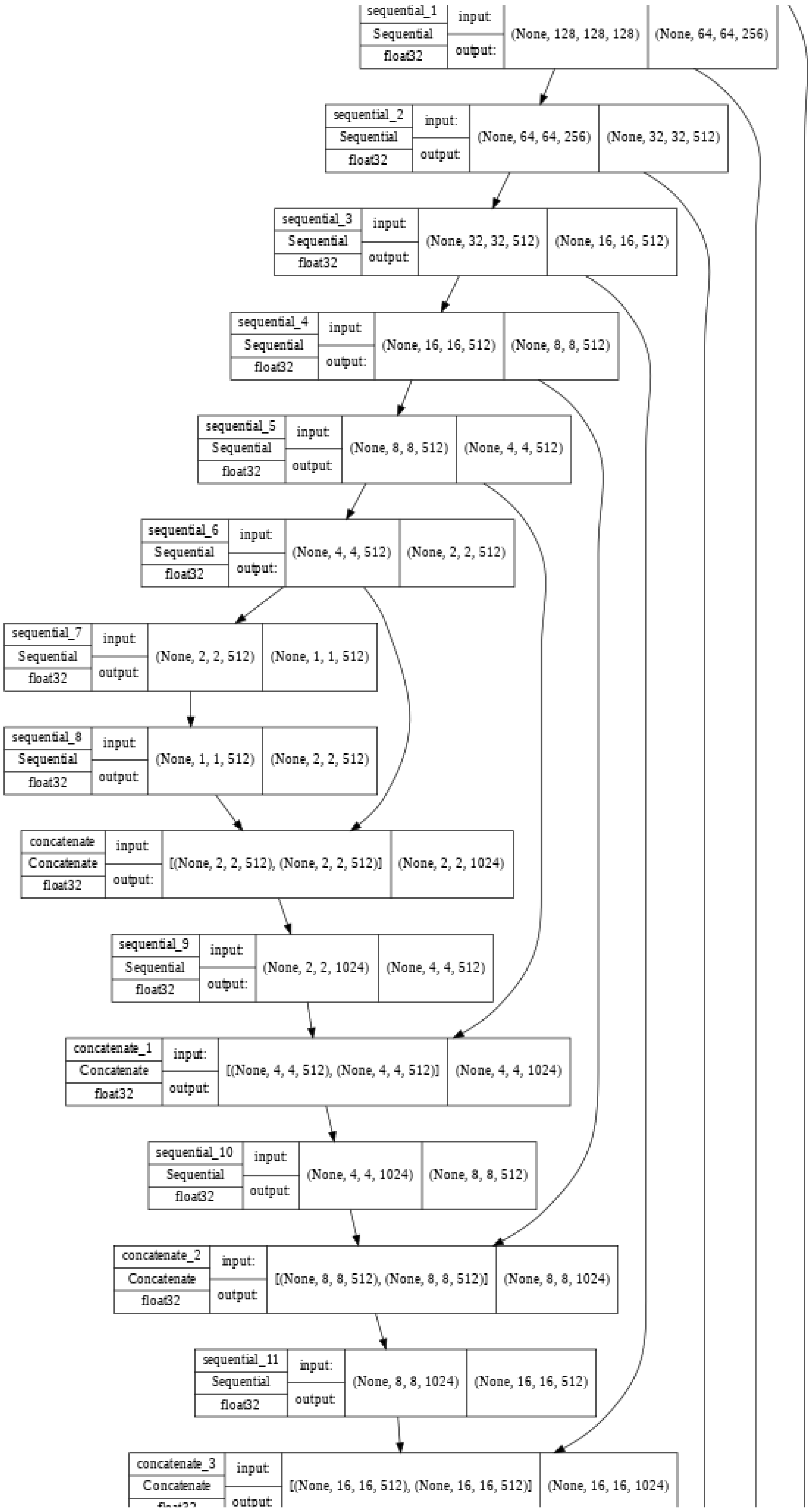

concatenate_2 | Concatenate | float32 | input: | output: | [(None, 8, 8, 512), (None, 8, 8, 512)] | (None, 8, 8, 1024)
sequential_11 | Sequential | float32 | input: | output: | (None, 8, 8, 1024) | (None, 16, 16, 512)
concatenate_3 | Concatenate | float32 | input: | output: | [(None, 16, 16, 512), (None, 16, 16, 512)] | (None, 16, 16, 1024)
sequential_12 | Sequential | float32 | input: | output: | (None, 16, 16, 1024) | (None, 32, 32, 512)
concatenate_4 | Concatenate | float32 | input: | output: | [(None, 32, 32, 512), (None, 32, 32, 512)] | (None, 32, 32, 1024)
sequential_13 | Sequential | float32 | input: | output: | (None, 32, 32, 1024) | (None, 64, 64, 256)
concatenate_5 | Concatenate | float32 | input: | output: | [(None, 64, 64, 256), (None, 64, 64, 256)] | (None, 64, 64, 512)
sequential_14 | Sequential | float32 | input: | output: | (None, 64, 64, 512) | (None, 128, 128, 128)
concatenate_6 | Concatenate | float32 | input: | output: | [(None, 128, 128, 128), (None, 128, 128, 128)] | (None, 128, 128, 256)
conv2d_transpose_7 | Conv2DTranspose | float32 | input: | output: | (None, 128, 128, 256) | (None, 256, 256, 1)

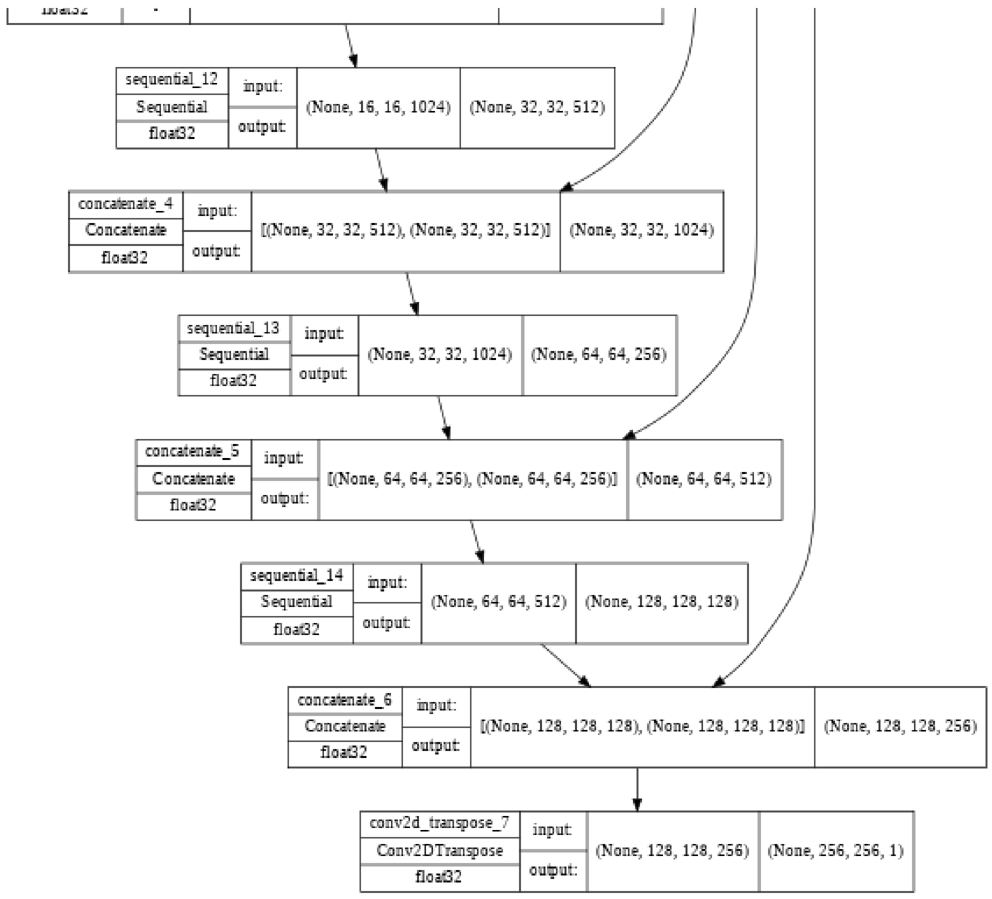
Discriminator

Discriminator is a traditional CNN, which we use to classify the Images. It only uses Downsampling hence. Discriminator shall classify a portion of the image generated from the generator (input image) as fake or real

Discriminator uses an image patch [patchGAN model] of (30 x 30 x 1) from the generator to verify the synthetic image as real or fake


input_1 | InputLayer | float32 | input: | output: | [(None, 256, 256, 1)] | [(None, 256, 256, 1)]
sequential | Sequential | float32 | input: | output: | (None, 256, 256, 1) | (None, 128, 128, 128)


sequential_1 | Sequential | float32 | input: | output: | (None, 128, 128, 128) | (None, 64, 64, 256)
sequential_2 | Sequential | float32 | input: | output: | (None, 64, 64, 256) | (None, 32, 32, 512)
sequential_3 | Sequential | float32 | input: | output: | (None, 32, 32, 512) | (None, 16, 16, 512)
sequential_4 | Sequential | float32 | input: | output: | (None, 16, 16, 512) | (None, 8, 8, 512)
sequential_5 | Sequential | float32 | input: | output: | (None, 8, 8, 512) | (None, 4, 4, 512)
sequential_6 | Sequential | float32 | input: | output: | (None, 4, 4, 512) | (None, 2, 2, 512)
sequential_7 | Sequential | float32 | input: | output: | (None, 2, 2, 512) | (None, 1, 1, 512)
sequential_8 | Sequential | float32 | input: | output: | (None, 1, 1, 512) | (None, 2, 2, 512)
concatenate | Concatenate | float32 | input: | output: | [(None, 2, 2, 512), (None, 2, 2, 512)] | (None, 2, 2, 1024)
sequential_9 | Sequential | float32 | input: | output: | (None, 2, 2, 1024) | (None, 4, 4, 512)
concatenate_1 | Concatenate | float32 | input: | output: | [(None, 4, 4, 512), (None, 4, 4, 512)] | (None, 4, 4, 1024)
sequential_10 | Sequential | float32 | input: | output: | (None, 4, 4, 1024) | (None, 8, 8, 512)
concatenate_2 | Concatenate | float32 | input: | output: | [(None, 8, 8, 512), (None, 8, 8, 512)] | (None, 8, 8, 1024)
sequential_11 | Sequential | float32 | input: | output: | (None, 8, 8, 1024) | (None, 16, 16, 512)
concatenate_3 | Concatenate | float32 | input: | output: | [(None, 16, 16, 512), (None, 16, 16, 512)] | (None, 16, 16, 1024)



)

Loss Functions

Binary Cross Entropy loss function is used in this example. Learning Rate (Lambda) of 10 is used as recommended in the original paper Several Loss functions are defined

- Generator loss : It is a binary cross entropy loss of the generated images and an array of ones
- Discriminator loss: Discrimantor loss Consists of 2 inputs

— real_loss is a binary cross entropy loss of the real images and an array of ones(since these are the real images)

— generated_loss is a sigmoid cross entropy loss of the generated images and an array of zeros(since these are the fake images)

— total_loss is the sum of real_loss and the generated_loss

- Cycle Consistency Loss — Cycle consistency means the result should be close to the original input. If T1 image is translated to T2 image, and then translates it back from T2 to T1, then the resulting image should be the same as the original image
- Identity Loss — If the image is fed to the generator, it should yield the real image or something close to image.
- Optimizer — Adam optimizer is used for both generators and discriminators

Training

The training consists of four broad steps

Get the predictions — The generator and discriminator modules are used to get the predictions

Calculate the loss — Various loss functions defined above are used to calculate the losses

Calculate the gradients using backpropagation

Apply the gradients to the optimizer — Adam optimizer as defined in the original paper is used for optimization

Below is the animated picture of the training process. The Predicted image is plotted after every epoch. (Plotted only few images for brevity)

Result of MRI Style Transfer Using GAN (T1 to T2 <-> T2 to T1)

(https://github.com/nishitgoyal17/gan-mri/blob/main/DSC43_animated_cycleGAN_100_epochs.gif