Advanced GAN training

Renato Cardoso

Supervisor: Sofia Vallecorsa

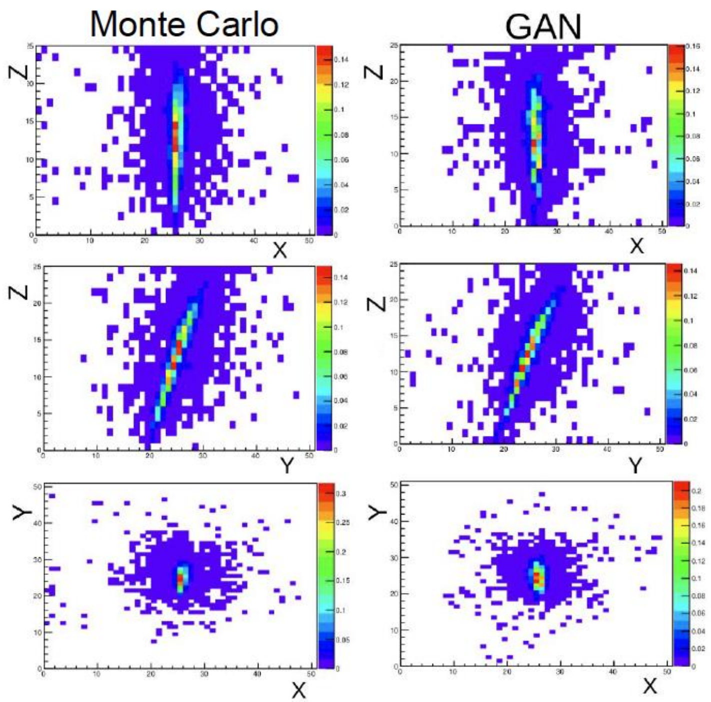
22/03/2022

What is this Presentation about?

- Distributed training on Cloud
- Conditional Progressive GAN for satellite images

Model from *G. Khattak, ICMLA 2019*

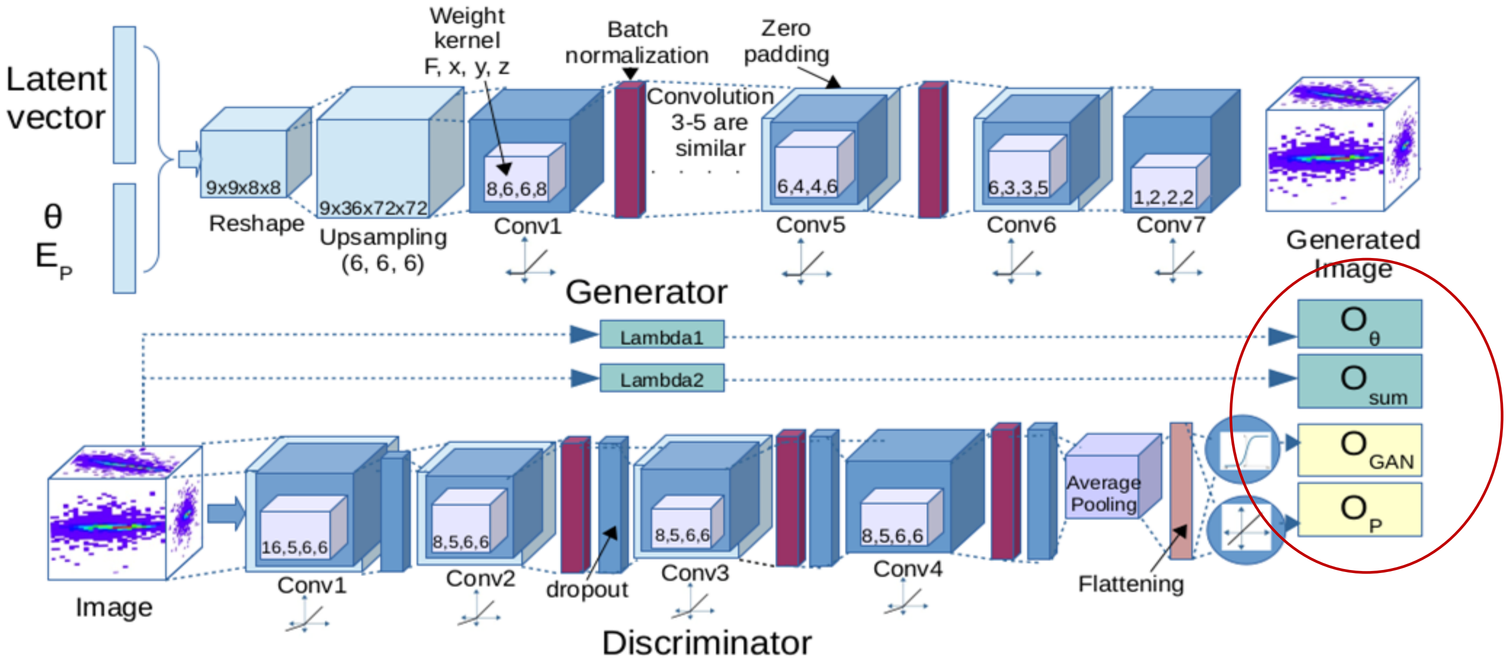
The 3DGAN prototype



3D convolutional layers.

51x51x25 pixels image: **sparse, large dynamic range**

Custom losses, including **physics constraints**



Physics constraints

3DGAN trains in about 1 week on a single NVIDIA V100

Training on TPUs

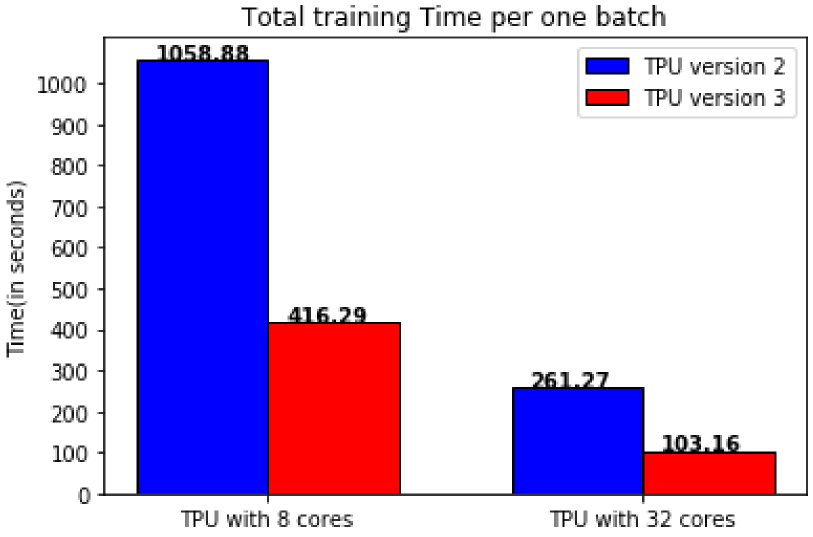
TensorFlow

- Distribute training using Tensorflow data parallel strategies:
  - Customized techniques to adapt the **Tensorflow MirroredStrategy and TPUStrategy**
  - Common setup to run on TPUs and multiple GPUs

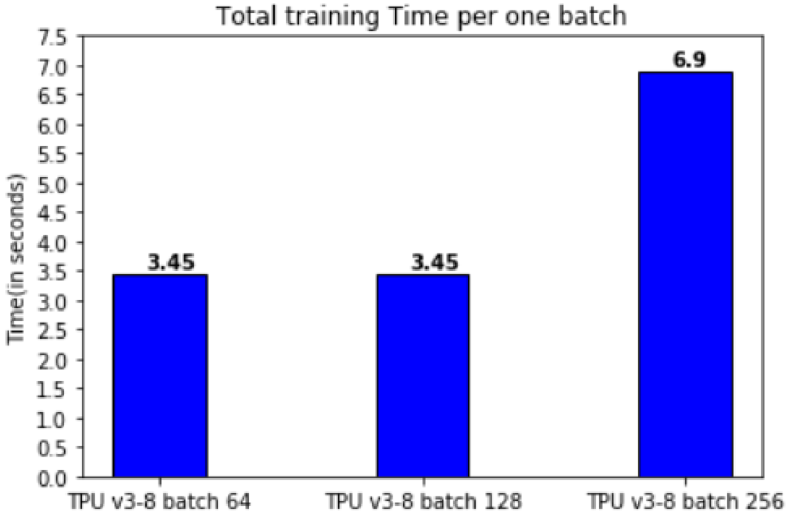
Google Cloud

CloudBank

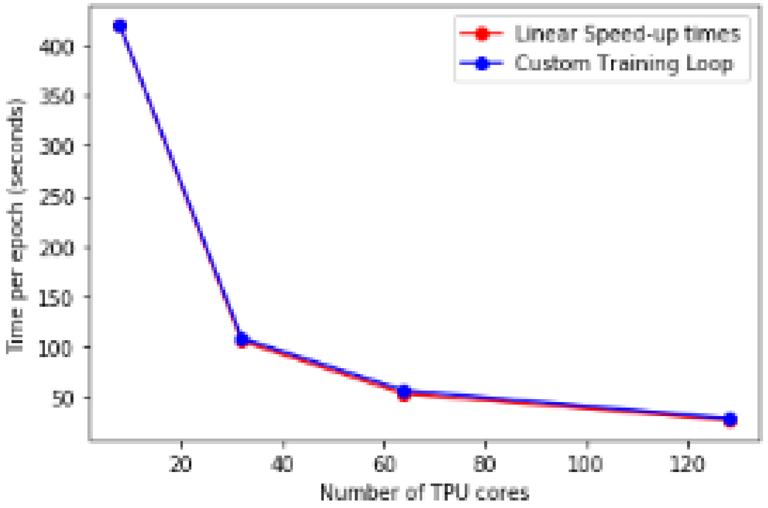
Deplyment made possible due to the colaboration with CloudBank EU and Google Cloud



TPUs versions comparison



Batch size optimisation



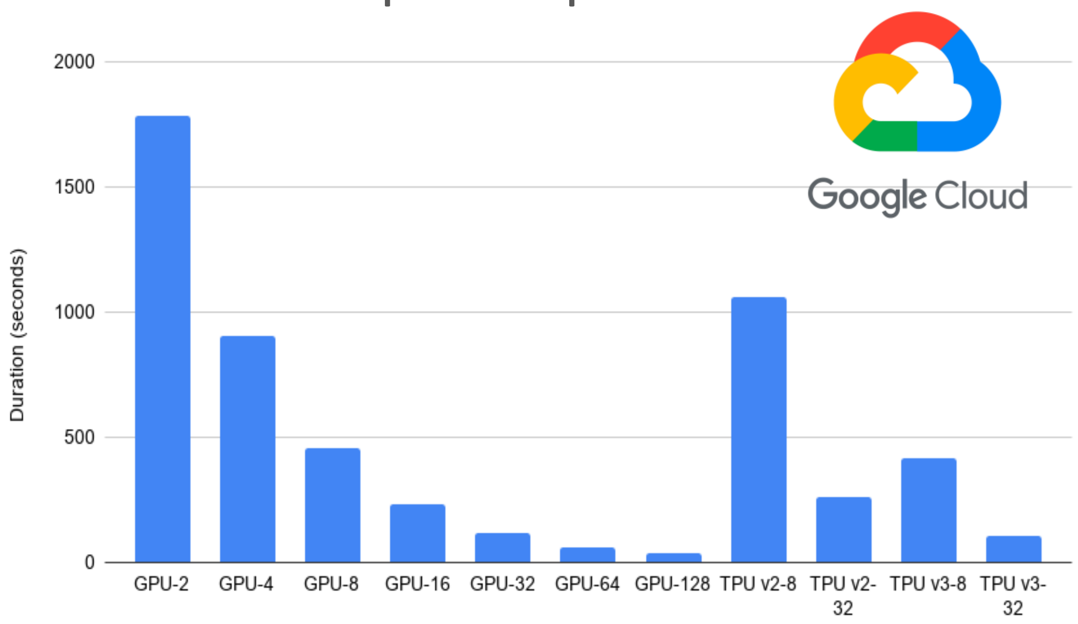
Speed-up

Multi GPU setups

Kubeflow based deployment on **GCP**.

From 1 to 128 (V100) GPUs

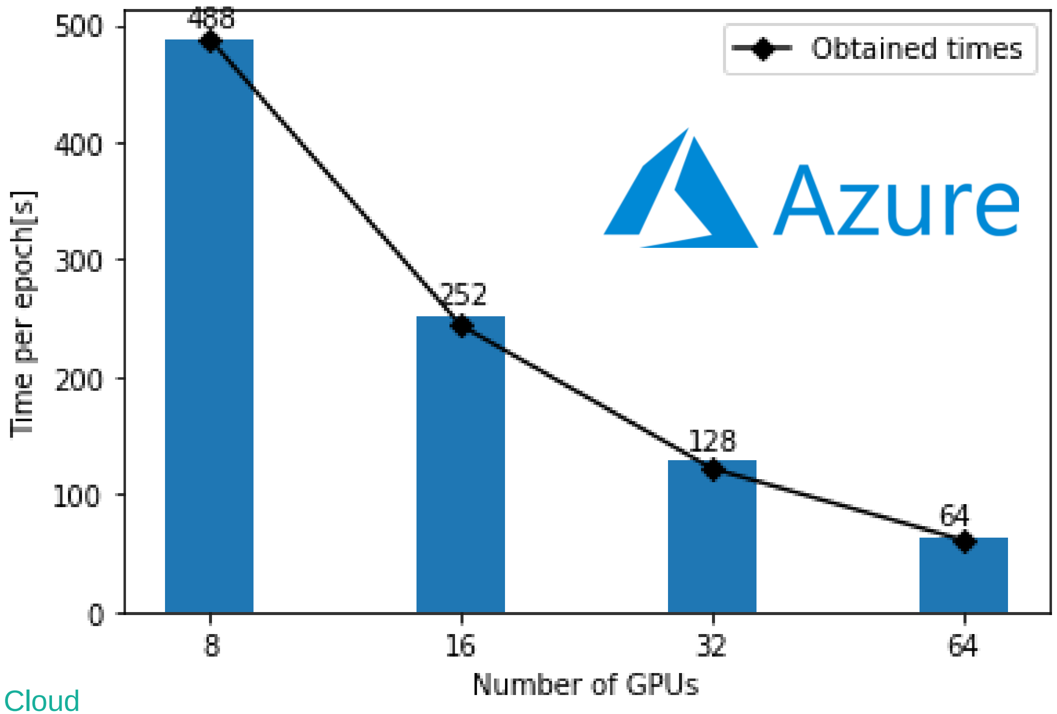
X100 near linear speed-up



Cluster provisioning through the **Azure** Machine Learning Service

24 vCPU cores VMs with 448 GiB memory and 4 V100 GPUs.

Azure ML automatically optimizes the data set management



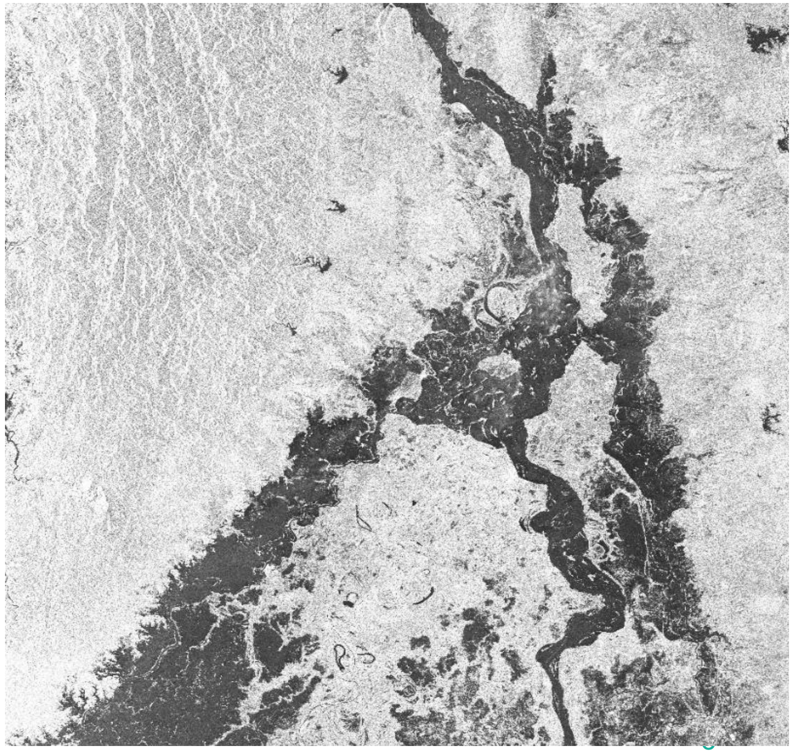
Progressive GAN

An improved logic to training for better performance on high resolution images:

Example satellite images from UNOSAT

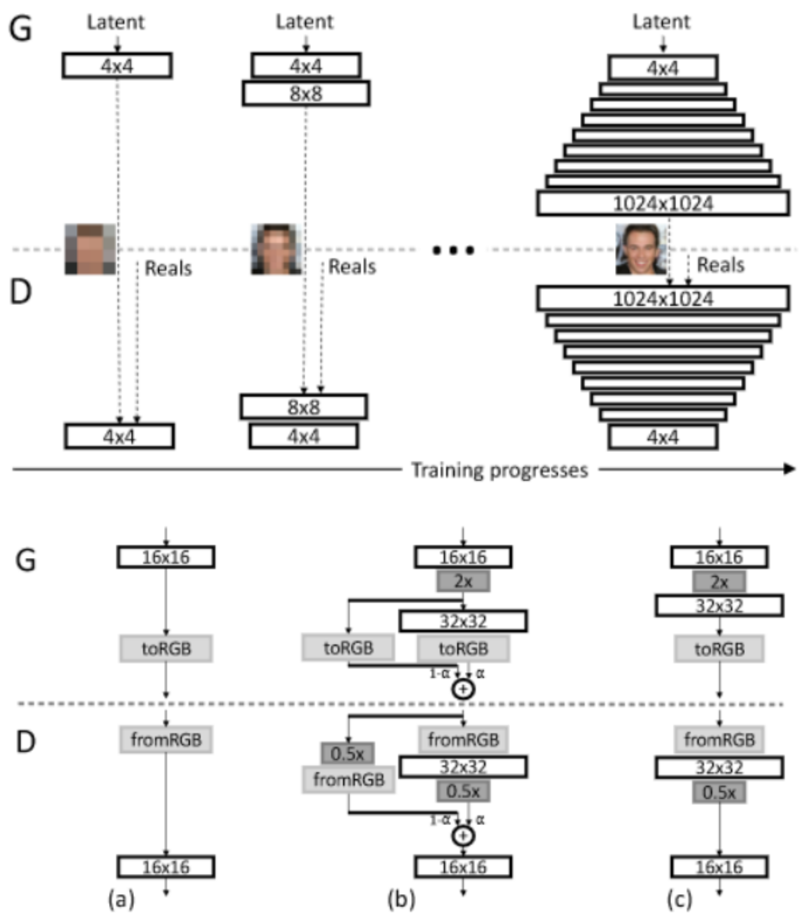
United Nations Institute for Training and Research

Myanmar Flood dataset



Progressive GAN

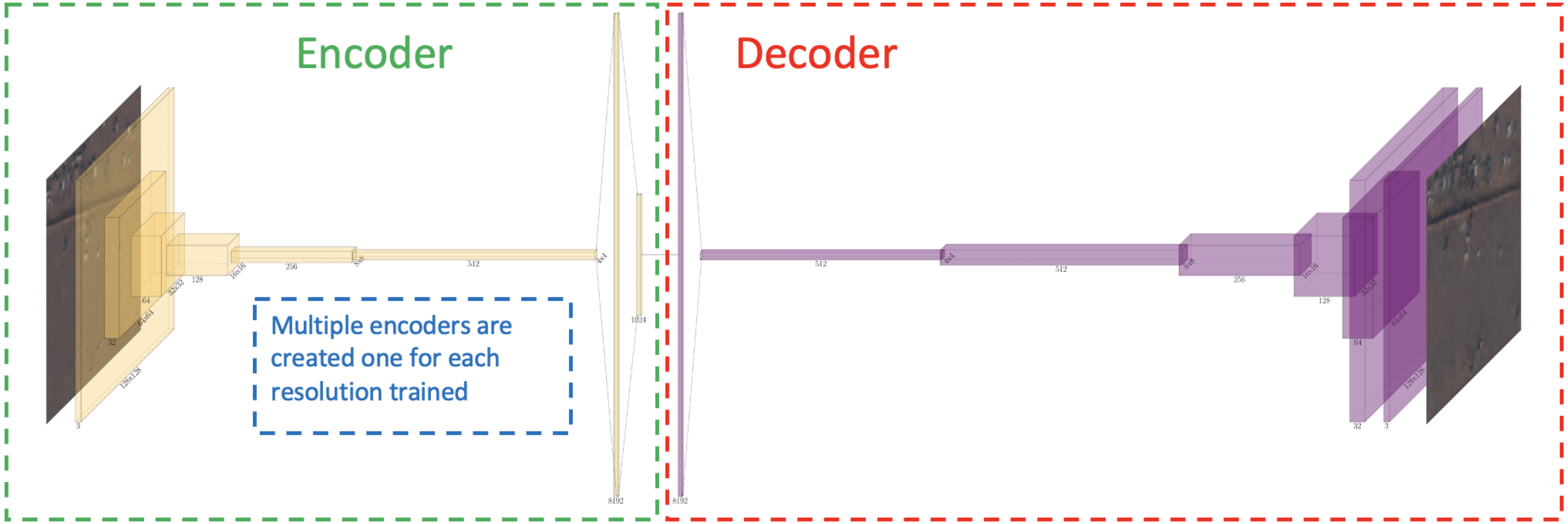
- Starts with low-resolution
- Progressively increase it by adding layers.
  - first discover large-scale structure
  - shift attention to increasingly finer scale detail
- Generator and Discriminator are mirror images of each other
- grow in synchrony.
  - All layers remain trainable throughout the training process.
  - Smoothly fade in new layers.
    - Avoids sudden shocks to previously trained, smaller-resolution layers.



Our Progressive GAN

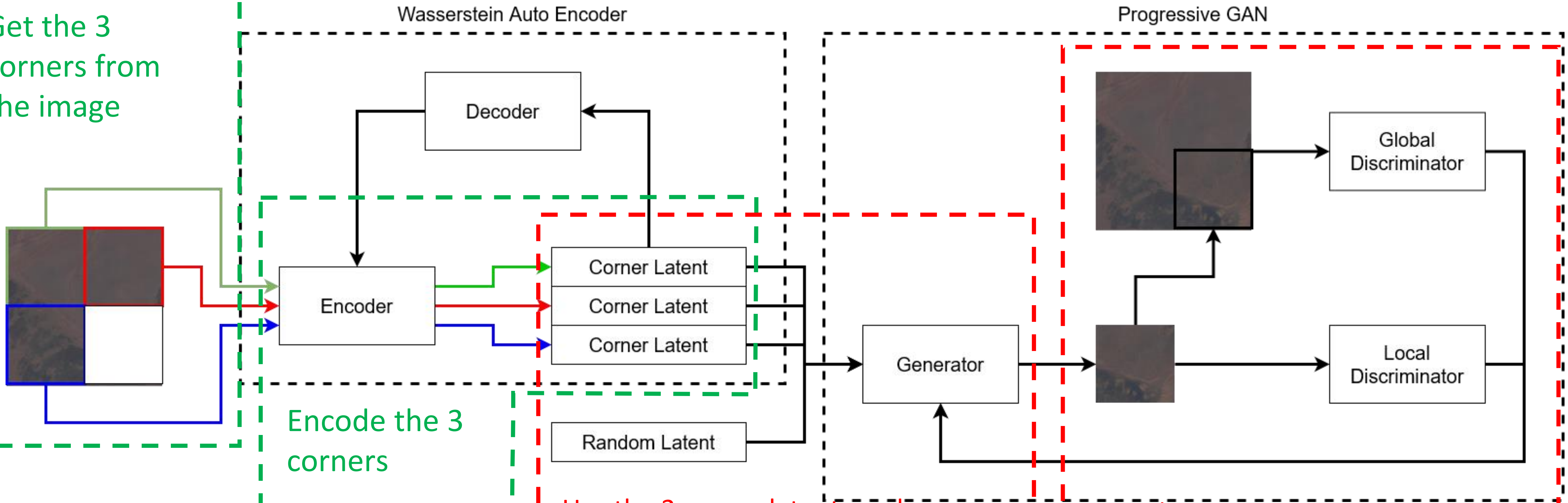
- Objective:
  - Image completion of 2D satellite images
  - Image in-painting and image extension
- Method: **Conditional** Progressive GAN
  - Use 3 corners of the image to infer the 4th
  - Introduce an **encoder** to decrease the dimensionality of the 3 given corners
  - It can generate a corner of an image and, iteratively, be used to produce larger images.

Use a Wasserstein AutoEncoder as input to conditioning



VAE + proGAN Architecture

Get the 3 corners from the image

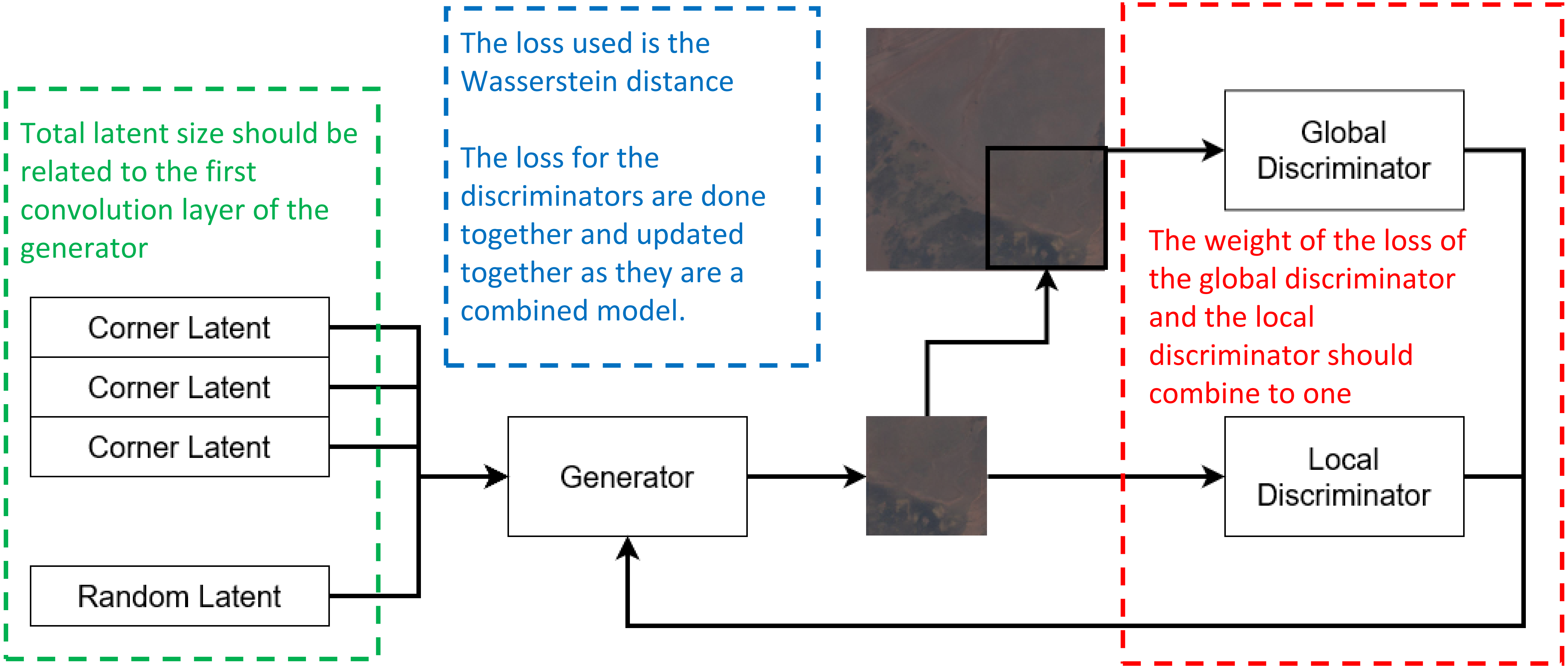


Encode the 3 corners

Use the 3 corner latents and a random latent to generate the last corner image

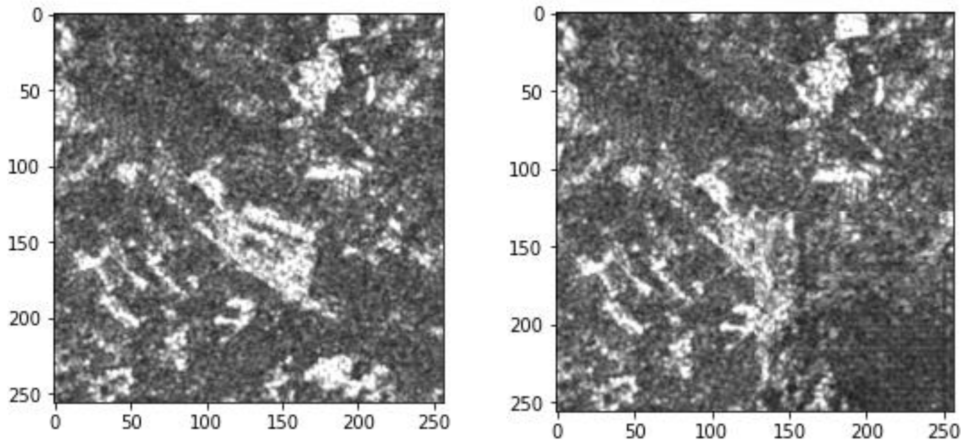
Use 2 discriminators to train:
- Global Discriminator to train with the whole image
- Local Discriminator to train with only the generated corner

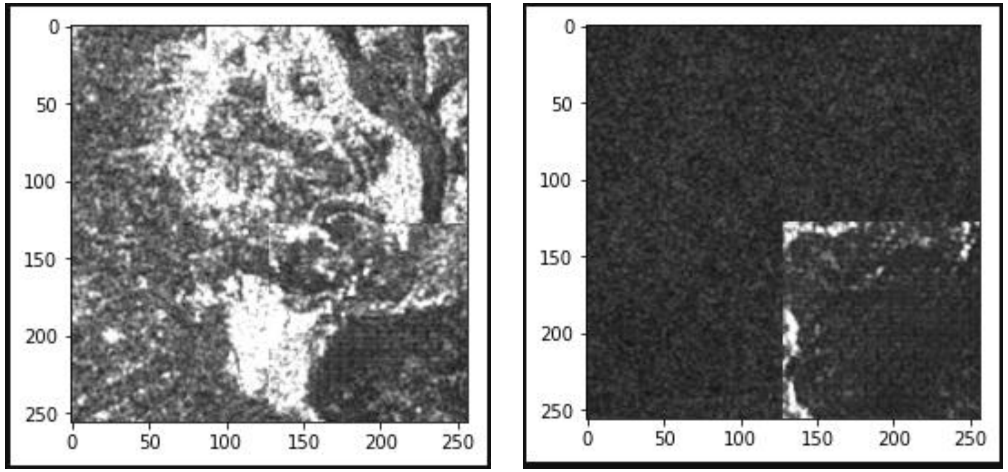
Progressive GAN



Results

Original Generated





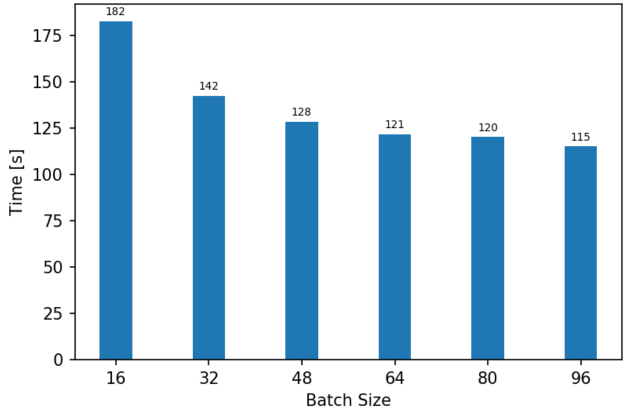
- Training:
  - 56472 different images from UNOSAT Myanmar flood dataset (a total of 12 million steps)
- Results:
  - The generated corner is consistent with the features present in the original image
  - **Similar but no identical**
- Problems:
  - Limitations in replicating multiple details and monochromatic tiles
- Next Steps:
  - Reduce training time: **1 month with 2 V100 GPUs**
  - Hyper-parameter optimization (role of the global vs local loss)
  - Test with RGB datasets

QUESTIONS?

renato.cardoso@cern.ch

Google Cloud Platform with Kubeflow

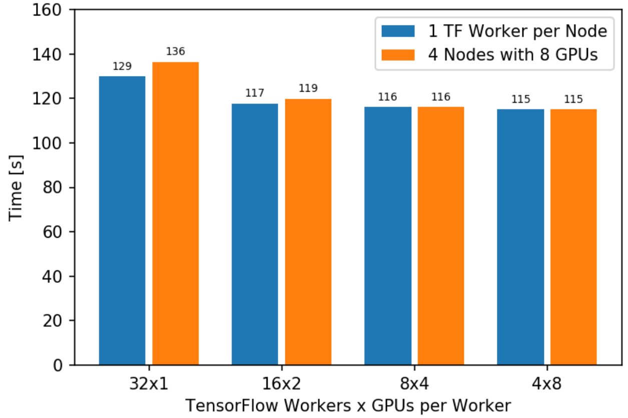
Batch Size Tests



- better generalization performance

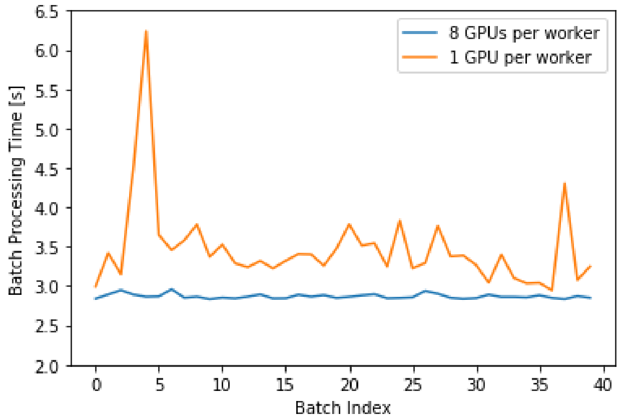
- less steps to complete
- faster training

Cluster Configuration



- optimal configuration
  - more GPUs per node
  - more GPUs per worker
- best results
  - number of workers = number of nodes
  - number of GPUs per worker = the number of GPUs per node

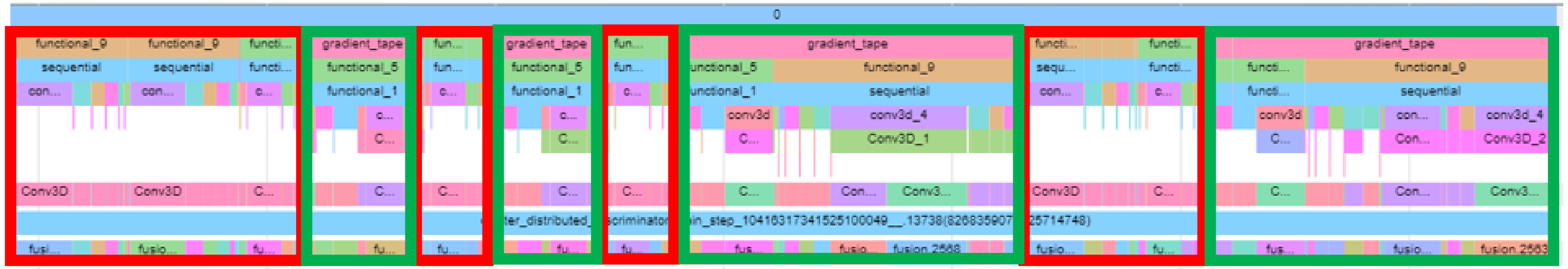
Stability Test



- Sub-optimal configuration makes training time instable and overall longer
- Equal number of GPUs per worker and GPUs per node keeps instability to a minimum

Further Analysis

- TPUs:
  - idle time is 0.7%, with 28.5% for All-Reduce operations
  - 38% for **forward-pass**
  - 61% **for back-propagation**
- GPUs:
  - idle time is 2.9%, mostly for All-Reduce
  - Similar percentages for forward and backward propagation as the TPUs.
- Program is not input bound, 0% of the training step time was spent waiting for input
- With this profile it is possible to verify that the model is compute bound

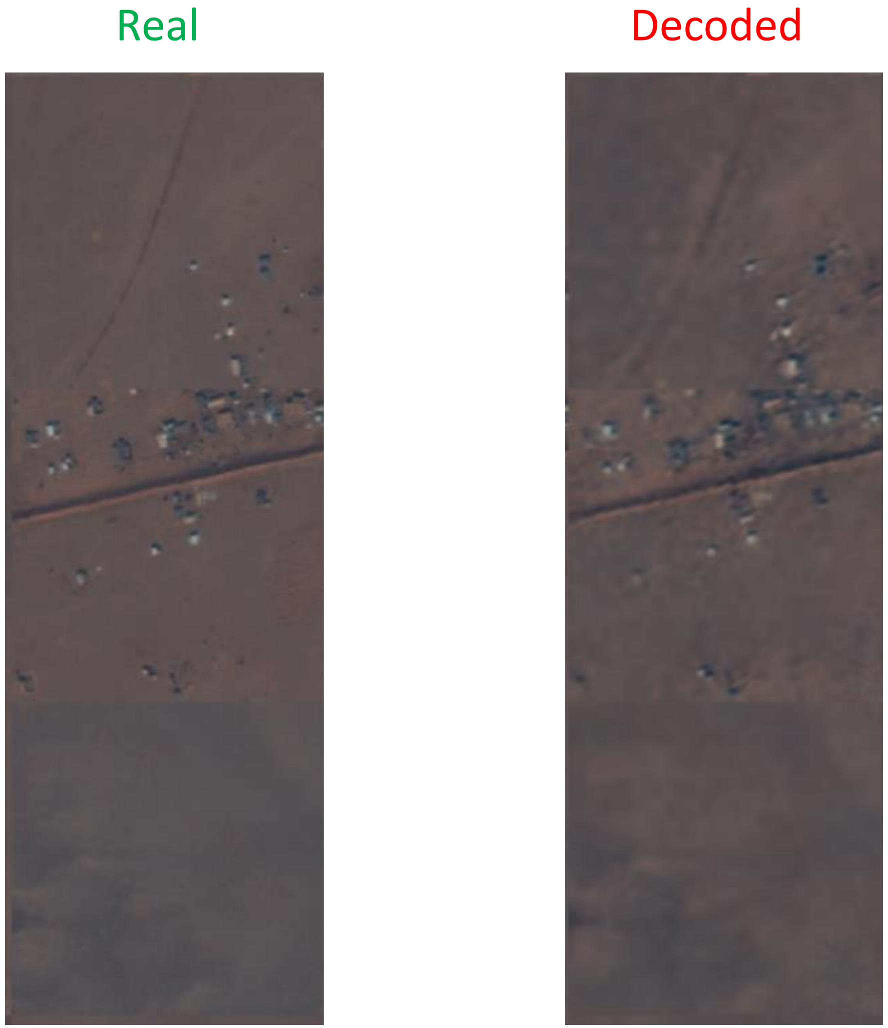


4 gradient tapes
- After each gradient tape there is one all reduce
- Followed by a RMSprop and the corresponding updates

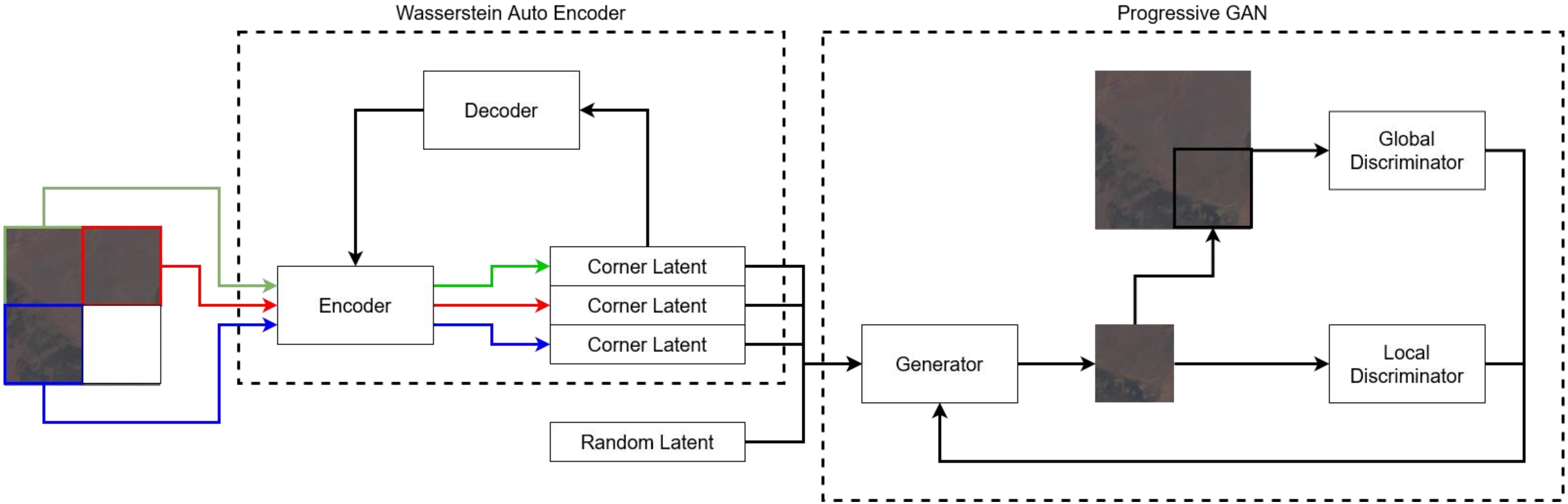
Wasserstein autoencoder

Results

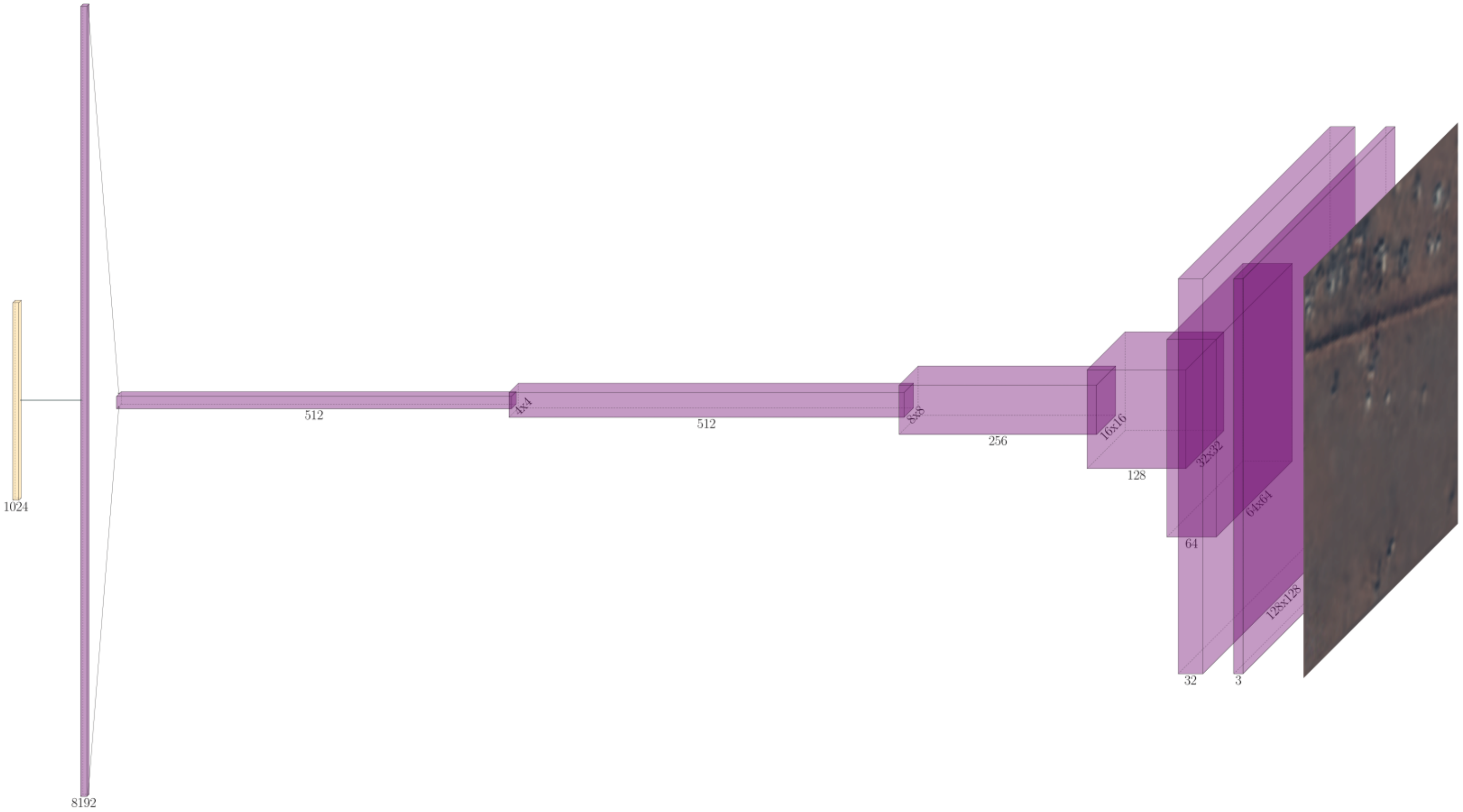
- There are two problem with the decoded images:
  - The blurry effect
  - The difference in color from the original
- The difference in color is something that persist to the Progressive GAN



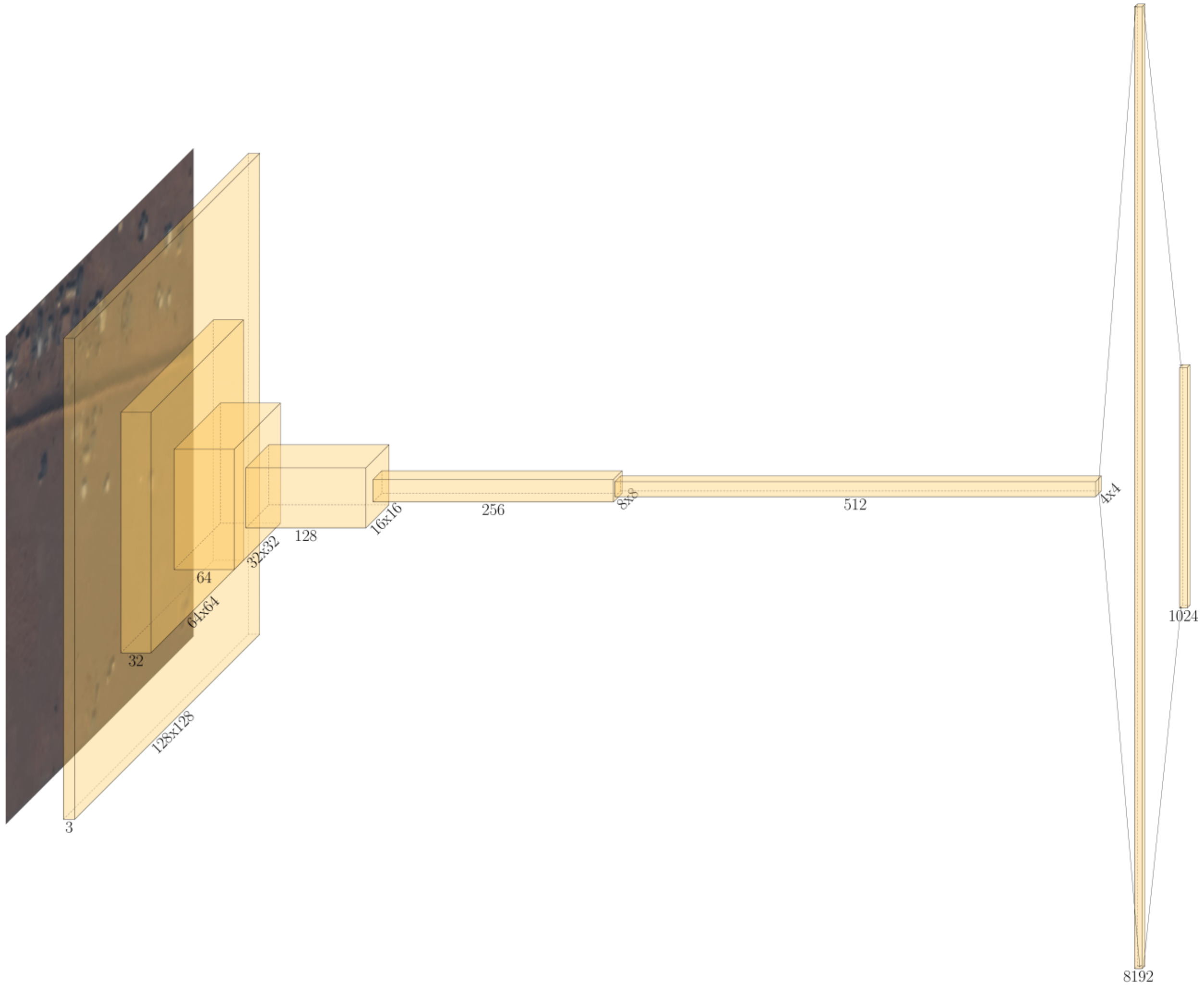
Backup - Architecture



Backup - Decoder



Backup - Encoder



Progressive GAN

- Architecture:
  - Both the Generator and the discriminator follow the usual Progressive Gan structure including pixel normalization and weight scalling
  - Generator:
    - The input is the concatenation of the 3 corner latents obtained from the encoder and a random latent
    - The start with a dimension is 32x32 and goes up to 128x128
    - Each generator block is composed of an UpSampling layer followed by 2 2D convolution layers with a leaku relu activation
  - Discriminators (Combined model)
    - Each discriminator block is composed by 2 2D convolution layers with a leaku relu activation followed by an average pooling layer
    - The global discriminator receives as an input the generated image with the 3 original corners while the local discriminator receives only the generated image

Wasserstein autoencoder

- Architecture:
  - Encoder (8 Layers):
    - Input is a 128x128 image
    - 1st layer is a 2D convolution layer used to extract the color of the image
    - The next 5 layers are 2D convolution layers (strides of 2) with batch normalization and leaky relu activation
    - The last 2 layers are a flatten and a Dense layer, respectively, used to compose the latent
  - Decoder (7 layers):
    - Input is a latent
    - 1st layer is a Dense layer followed by a reshape
    - The next 5 layers are 2D Transpose convolution layers (strides of 2) with batch normalization and leaky relu activation
    - The last layer is a 2D convolution layer used to convert to RGB, obtaining the final image

Wasserstein autoencoder

- Loss
  - Loss calculation uses 2 components a reconstruction loss and a mmd loss
  - Reconstruction loss
    - Mean squared error between the decoded image and the original image
  - MMD loss:
    - Maximum mean discrepancy (MMD) with a radial basis function between a random normal and the latent obtained from the encoder

The encoder is trained first using the decoder and doesn't change when being used for the progressive GAN training

More Results