Supplementary Materials

1. Implementation details

For implementing PlantPlotGAN and the other two GAN networks (i.e., DCGAN and WGAN), we utilized the TensorFlow library, setting the Adam optimizer with the learning rate to 2e-4, $\beta1 = 0.5$, and $\beta2 = 0.99$. The layered architecture for each model is the following:

The careful reader will note that DCGAN and PlantPlotGAN architectures are equal, which demonstrates the importance of using a second discriminator to achieve the multispectral evaluation. On the other hand, WGAN has a more complex architecture, but it didn't result in improved accuracy.

The PlantPlotGAN model begins with a dense layer that takes random noise as input and produces a high-dimensional output. This output is then passed through a LeakyReLU activation function, which introduces a small amount of non-linearity to prevent the "dying ReLU" problem. The subsequent reshaped layer rearranges the output into a 4D tensor, preparing it for the transposed convolutional layers. Transposed convolutional layers, also known

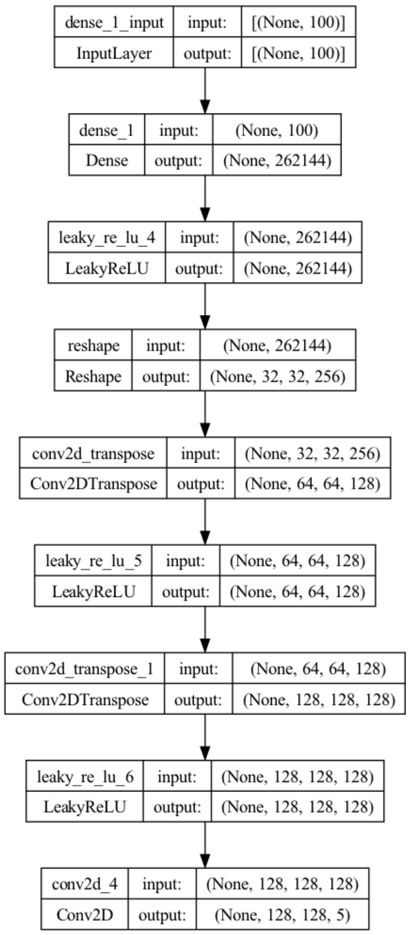
dense_1_input | input: | [(None, 100)]
InputLayer | output: | [(None, 100)]
dense_1 | input: | (None, 100)
Dense | output: | (None, 262144)
leaky_re_lu_4 | input: | (None, 262144)
LeakyReLU | output: | (None, 262144)
reshape | input: | (None, 262144)
Reshape | output: | (None, 32, 32, 256)
conv2d_transpose | input: | (None, 32, 32, 256)
Conv2DTranspose | output: | (None, 64, 64, 128)
leaky_re_lu_5 | input: | (None, 64, 64, 128)
LeakyReLU | output: | (None, 64, 64, 128)
conv2d_transpose_1 | input: | (None, 64, 64, 128)
Conv2DTranspose | output: | (None, 128, 128, 128)
leaky_re_lu_6 | input: | (None, 128, 128, 128)
LeakyReLU | output: | (None, 128, 128, 128)
conv2d_4 | input: | (None, 128, 128, 128)
Conv2D | output: | (None, 128, 128, 5)

Figure 1. Base architecture for implementing the DCGAN model.

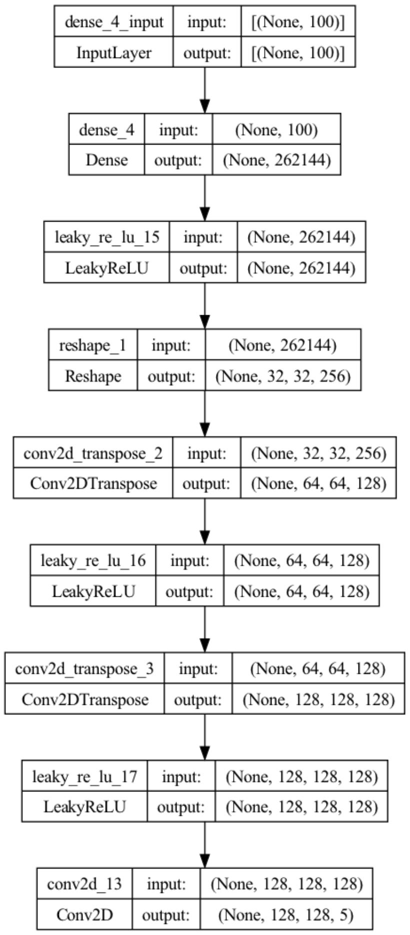
dense_4_input | input: | [(None, 100)]
InputLayer | output: | [(None, 100)]
dense_4 | input: | (None, 100)
Dense | output: | (None, 262144)
leaky_re_lu_15 | input: | (None, 262144)
LeakyReLU | output: | (None, 262144)
reshape_1 | input: | (None, 262144)
Reshape | output: | (None, 32, 32, 256)
conv2d_transpose_2 | input: | (None, 32, 32, 256)
Conv2DTranspose | output: | (None, 64, 64, 128)
leaky_re_lu_16 | input: | (None, 64, 64, 128)
LeakyReLU | output: | (None, 64, 64, 128)
conv2d_transpose_3 | input: | (None, 64, 64, 128)
Conv2DTranspose | output: | (None, 128, 128, 128)
leaky_re_lu_17 | input: | (None, 128, 128, 128)
LeakyReLU | output: | (None, 128, 128, 128)
conv2d_13 | input: | (None, 128, 128, 128)
Conv2D | output: | (None, 128, 128, 5)

Figure 2. Base architecture for implementing the PlantPlotGAN model.

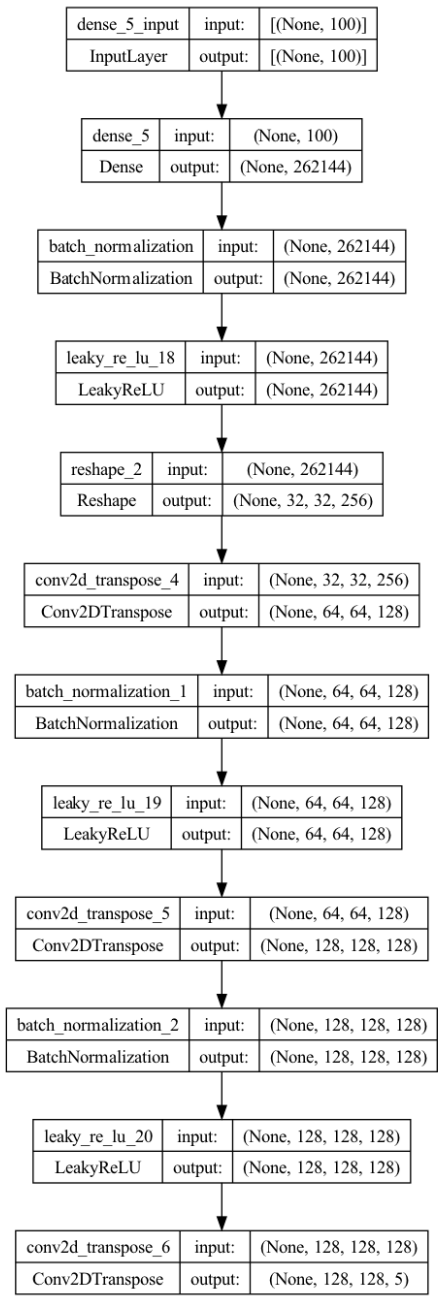
dense_5_input | input: | [(None, 100)]
InputLayer | output: | [(None, 100)]
dense_5 | input: | (None, 100)
Dense | output: | (None, 262144)
batch_normalization | input: | (None, 262144)
BatchNormalization | output: | (None, 262144)
leaky_re_lu_18 | input: | (None, 262144)
LeakyReLU | output: | (None, 262144)
reshape_2 | input: | (None, 262144)
Reshape | output: | (None, 32, 32, 256)
conv2d_transpose_4 | input: | (None, 32, 32, 256)
Conv2DTranspose | output: | (None, 64, 64, 128)
batch_normalization_1 | input: | (None, 64, 64, 128)
BatchNormalization | output: | (None, 64, 64, 128)
leaky_re_lu_19 | input: | (None, 64, 64, 128)
LeakyReLU | output: | (None, 64, 64, 128)
conv2d_transpose_5 | input: | (None, 64, 64, 128)
Conv2DTranspose | output: | (None, 128, 128, 128)
batch_normalization_2 | input: | (None, 128, 128, 128)
BatchNormalization | output: | (None, 128, 128, 128)
leaky_re_lu_20 | input: | (None, 128, 128, 128)
LeakyReLU | output: | (None, 128, 128, 128)
conv2d_transpose_6 | input: | (None, 128, 128, 128)
Conv2DTranspose | output: | (None, 128, 128, 5)

Figure 3. Base architecture for implementing the WGAN model.

as deconvolutional layers, are used for upsampling the data and increasing its spatial dimensions. The first transposed convolutional layer applies 128 filters with a kernel size of 4x4 and a stride of 2. Another LeakyReLU activation follows to introduce non-linearity. With the same specifications, the second transposed convolutional layer further increases the spatial dimensions. Another LeakyReLU activation is applied. Finally, a convolutional layer with a number of filters equal to the desired number of output chan-

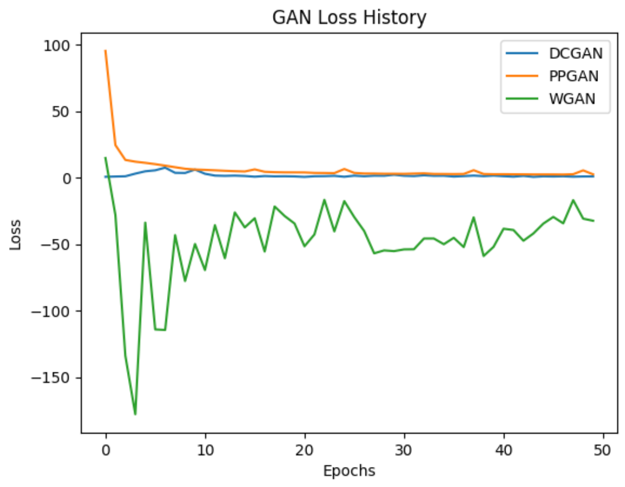

Figure 4. Observed loss function for each GAN selected for evaluation.

nels (e.g., multispectral channels) and a kernel size of 4x4 is added. The activation function used here is tanh, which scales the output values between -1 and 1. This layer generates the final synthetic image output. Overall, the layers in the generator progressively transform the input noise into a high-dimensional tensor representing a synthetic image.

At the last layer of each model, there is a Conv2D layer of shape (128, 128, 5) just for generating the multispectral imagery, which generally has 5 channels.

Due to size limitations, we could not attach the trained models with this supplementary material. However, our work's code and trained models are available at the following GitHub repository: https://github.com/felipealencar/plant-disease-prediction.

2. Observing the loss functions of the generators

Another complementary experiment was observing each loss function during the training process. Surprisingly, PlantPlotGAN's loss function achieved a stable increasing learning, similar to DCGAN. However, its associated learning process starts in a shape that resembles the WGAN loss function initially, but with improved metrics.

Conversely, although WGAN utilizes a distance-based loss function as PlantPlotGAN, it still did not achieve suitable convergence. As discussed in the paper, the reason for that is the spectral regularizer integrated into PlantPlotGAN.