Exploration of Data Augmentation Techniques for Bush Detection in Blueberry Orchards

Supplementary Material

A. Appendix

A.1. Generative Model

In our study, we began with a standard Deep Convolutional Generative Adversarial Network (DCGAN) [33] framework as our foundation and expanded its layers to better accommodate images of size 3x128x128 instead of the conventional 3x64x64. This enhancement was designed to more accurately capture the intricacies and details of the higher-resolution images we seek to replicate, thereby improving the model's proficiency in generating synthetic images that closely mirror the real bush photographs quality and texture.

The architecture of our refined DCGAN includes two main components: a generator (G) and a discriminator (D). The generator's objective is to fabricate images that are indiscernible from real images, beginning from a latent space vector z, sampled from a standard normal distribution. Conversely, the discriminator evaluates images to determine their authenticity, real (from the dataset) or fake (generated by G), outputting a probability score.

Our generator is constructed with convolutional-transpose layers, batch normalization layers, and ReLU activations, designed to transform a latent vector into a 3x128x128 RGB image. In line with the DCGAN paper's recommendations, all model weights are initialized from a normal distribution with mean=0 and standard deviation=0.02, to promote effective training dynamics.

```
# input is ``(nc) x 128 x 128``
nn.Conv2d(nc, ndf // 2, 4, 2, 1, bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. ``(ndf//2) x 64 x 64``
nn.Conv2d(ndf // 2, ndf, 4, 2, 1, bias=False),
nn.BatchNorm2d(ndf),
nn.LeakyReLU(0.2, inplace=True),
# state size. ``(ndf) x 32 x 32``
nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm2d(ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. ``(ndf*2) x 16 x 16``
nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm2d(ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. ``(ndf*4) x 8 x 8``
nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm2d(ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. ``(ndf*8) x 4 x 4``
nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
nn.Sigmoid(),
```

Listing 1. Generator code

The discriminator is composed of strided convolution layers, batch normalization layers, and LeakyReLU activations. It processes input images of size 3x128x128, classifying them as real or fake. Importantly, to avoid the discriminator from becoming overly confident in its assessments—a scenario that could significantly impede the generator's learning gradient—we opted to employ soft labels rather than hard binary labels. Specifically, we adjusted the labels for the real images to 0.8 (instead of 1.0) and for the fake images to 0.2 (instead of 0.0). This modification ensures that the discriminator's output probabilities are never absolute (i.e., 100% real or fake), fostering a more nuanced and continuous learning environment for both components of the DCGAN. By implementing soft labels at these levels, we strike a balance that encourages the discriminator to maintain a level of uncertainty, thereby preventing it from dominating the generator too early in the training process and ensuring a healthier gradient flow for the generator's ongoing learning. The optimization of both the generator and discriminator is performed using the Adam optimizer, with a learning rate of 0.0002 and Beta1 set to 0.5.

```
# input is Z, going into a convolution
nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=
    False),
nn.BatchNorm2d(ngf * 8),
nn.ReLU(True),
# state size. ``(ngf*8) x 4 x 4``
nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1,
    bias=False),
nn.BatchNorm2d(ngf * 4),
nn.ReLU(True),
# state size. ``(ngf*4) x 8 x 8``
nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1,
    bias=False),
nn.BatchNorm2d(ngf * 2),
nn.ReLU(True),
# state size. ``(ngf*2) x 16 x 16``
nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=
    False),
nn.BatchNorm2d(ngf),
nn.ReLU(True),
# state size. ``(ngf) x 32 x 32``
nn.ConvTranspose2d(ngf, ngf // 2, 4, 2, 1, bias=
    False),
nn.BatchNorm2d(ngf // 2),
nn.ReLU(True),
# state size. ``(ngf//2) x 64 x 64``
nn.ConvTranspose2d(ngf // 2, nc, 4, 2, 1, bias=
    False),
nn.Tanh(),
```

Listing 2. Discriminator code

A crucial aspect of training involves the loss functions for both G and D, utilizing the Binary Cross Entropy (BCE) loss, expressed as:

$$\begin{aligned} \ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \\ l_n = -\left[y_n \cdot \log(x_n) + (1 - y_n) \cdot \log(1 - x_n)\right] \end{aligned} \tag{1}$$

This loss function calculates both components of the objective function, $\log(D(x))$ and $\log(1 - D(G(z)))$, allowing for specific parts of the BCE equation to be utilized depending on the input. The GAN loss function, pivotal to the training of both G and D in their min-max game, is given by [12]:

$$\begin{aligned} \min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] \\ + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \end{aligned} \tag{2}$$

In this min-max game, D aims to maximize the probability of correctly classifying real and fake images, while G strives to minimize the probability of D identifying its outputs as fake. The theoretical equilibrium of this game is achieved when $p_g = p_{\text{data}}$, and D classifies the inputs as real or fake with equal probability. However, in practice, achieving this equilibrium remains a complex challenge, with the convergence theory of GANs still under active investigation.