FruitGAN: Generating Realistic Synthetic Fruit Images

Manuel Scionti SCIONTI.MANUEL@HOTMAIL.IT

1. Model Description

A **Conditional DCGAN (Deep Convolutional Generative Adversarial Network)** is an advanced model derived from standard DCGANs (Mirza and Osindero, 2014). It enhances both the generator and discriminator networks by incorporating conditional information. In addition to the random noise vector, a Conditional DCGAN takes extra inputs like class labels or other conditional data. This inclusion allows for the creation of more controlled and specific outputs. By conditioning the generator, it becomes capable of producing samples that align with particular classes or features, resulting in more focused and diverse output generation. The discriminator network is also modified to consider this conditional information when making discrimination decisions. The primary objective of this model is to generate (64x64) resolution images that are well-aligned with given image labels. Following training, the model's performance is assessed using metrics such as the Inception Score and FID. Furthermore, the model serves the purpose of constructing a synthetic dataset, effectively augmenting data for training a classifier based on the conditioning labels. This process aims to determine whether it contributes positively to classification performance.
For the classification task in this study, a simple convolutional neural network (CNN) was used both before (using the original dataset) and after creating synthetic images. This phase is essential to determine whether CNN can correctly categorize fake images and to provide a comparison with the classifier's performance with the original dataset.

1.1. The Generative Adversarial Network (GAN)

The used **Generator**, as can be observed in Figure 1, is organized into three distinct blocks. Notably, the *Embedding layer* converts class labels into continuous embeddings for conditional generation. *Linear layers* handle data transformation, while *LeakyReLU* activations introduce non-linearity. The third block employs transposed convolutional layers for upsampling and includes *Batch Normalization* for stability. A final *Tanh* activation scales output to [-1, 1] for image generation.

Figure 2 offers a detailed overview of the **Discriminator** architecture. It consists of two sequential blocks: the first handles class label embeddings and linear transformations, enhancing conditional discrimination. The second block features *Conv2d layers* with *LeakyReLU* activations, introducing non-linearity and spatial dimension reduction, followed by *Batch Normalization* for stability and a flatten layer for data preparation. Regularization is applied via a *Dropout layer*, and a final Linear layer with *Sigmoid* activation performs binary classification.

Layers	**Param #**
Generator	–
Sequential: 1-1	–
Embedding: 2-1	500
Linear: 2-2	1,616
Sequential: 1-2	–
Linear: 2-3	827,392
LeakyReLU: 2-4	–
Sequential: 1-3	–
ConvTranspose2d: 2-5	4,202,496
BatchNorm2d: 2-6	1,024
ReLU: 2-7	–
ConvTranspose2d: 2-8	2,097,152
BatchNorm2d: 2-9	512
ReLU: 2-10	–
ConvTranspose2d: 2-11	524,288
BatchNorm2d: 2-12	256
ReLU: 2-13	–
ConvTranspose2d: 2-14	6,144
Tanh: 2-15	–
Total params:	7,661,380
Trainable params:	7,661,380
Non-trainable params:	0

Figure 1: Generator Architecture

Layers	**Param #**
Discriminator	–
Sequential: 1-1	–
Embedding: 2-1	500
Linear: 2-2	1,241,088
Sequential: 1-2	–
Conv2d: 2-3	6,144
LeakyReLU: 2-4	–
Conv2d: 2-5	131,072
BatchNorm2d: 2-6	256
LeakyReLU: 2-7	–
Conv2d: 2-8	524,288
BatchNorm2d: 2-9	512
LeakyReLU: 2-10	–
Flatten: 2-11	–
Dropout: 2-12	–
Linear: 2-13	16,385
Sigmoid: 2-14	–
Total params:	1,920,245
Trainable params:	1,920,245
Non-trainable params:	0

Figure 2: Discriminator Architecture

1.2. The Convolutional Neural Network

Table 1: CNN Model Layers

Layer	Details
Convolutional	Conv2d (in_channels=3, out_channels=3, kernel_size=9, padding=0, stride=1)
Activation	ReLU
Convolutional	Conv2d (in_channels=3, out_channels=3, kernel_size=9, padding=0, stride=1)
Activation	ReLU
Max Pooling	MaxPool2d (kernel_size=2, stride=2)
Fully Connected	Linear (in_features=10, out_features=10)
Activation	ReLU
Fully Connected	Linear (in_features=10, out_features=5)

The presented **Convolutional Neural Network (CNN)** architecture, as depicted in Table 1, consists of several key layers. It begins with a pair of convolutional layers, each with a 3x3 kernel size and *ReLU* activation functions, aimed at capturing complex patterns in the input data. Following these convolutional layers, a max-pooling operation with a 2x2 kernel size is applied to downsample the features, reducing computational complexity. The subsequent fully connected layers, comprising one with 10 neurons and another with 5 neurons, introduce non-linearity into the network and facilitate classification tasks.

2. Dataset

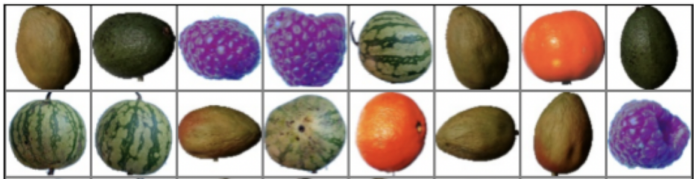

Figure 3: Selected fruit classes

For the generation task, a dataset based on fruit images was chosen. To overcome some computation limits, only 5 over 33 original classes of fruit were selected: avocado, clementine, papaya, raspberry, watermelon. The initial image size was 100x100 pixels. To facilitate training, the dataset was divided into three subsets: 70% for training, 15% for validation, and 15% for testing. This resulted in 1661 images for training, 356 for validation, and 357 for testing. Additionally, all images were resized to a 64x64 resolution and normalized within the range ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)). Papaya and avocado were intentionally selected as classes because of their similarity. So as to generate an extra difficulty for the model.

3. Training procedure

3.1. Hyperparameters and Optimization Techniques

In this section, we outline the key hyperparameters and optimization techniques employed during the training of the GAN. The image size is fixed at 64x64 pixels, and there are 5 classes in the dataset. An embedding dimension of 100 is utilized, along with a latent space dimension (z) also set to 100. Image preprocessing involves resizing and normalization. Custom weight initialization is applied to convolutional and batch normalization layers. *Adam optimizer* with a learning rate of 0.0002 and betas (0.5, 0.999) is employed for both the Generator and Discriminator. Furthermore, *Binary Cross-Entropy (BCE) loss* functions are used for training the Generator and Discriminator.

3.2. Loss Function

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot \log\left(\hat{y}_i\right) + \left(1 - y_i\right) \cdot \log\left(1 - \hat{y}_i\right) \tag{1}$$

The loss function employed for training the GAN is the *Binary Cross-Entropy (BCE) loss*. It is used to calculate the losses for both the Generator and Discriminator during the training process. The BCE loss is a common choice for GANs and plays a crucial role in optimizing the models. Here in 1, the y_i represents the ground truth value, instead $\hat{y}_i$ represents the output value.

3.3. Quality Metrics

Two quality metrics are utilized to evaluate the performance of the trained GAN:

- **Inception Score (IS)**: The Inception Score measures the diversity and quality of generated images. It calculates the mean and standard deviation of the scores for generated images. A higher IS indicates better image quality and diversity.
- **Frechet Inception Distance (FID)**: FID assesses the similarity between the distribution of real and generated images. A lower FID signifies a closer match between the two distributions, indicating better GAN performance.

4. Experimental Results

The GAN model's performance was assessed using two key metrics, *Inception Score (IS)* and *Frechet Inception Distance (FID)*, across different training epochs. Initially, at Epoch *1/1000*, the model had an IS of 1.32 (mean) with a 0.15 standard deviation, indicating moderate diversity in generated samples. However, the FID score was relatively high at 338.92, indicating dissimilarity with real images. By *Epoch 500/500*, IS was 2.25 (mean) with a 0.60 standard deviation, and FID remained low at 194.65. *After 1000 epochs*, IS further increased to 2.46 (mean) with a 0.68 standard deviation, and FID decreased to 188.43. In summary, the GAN displayed exceptional performance, with increasing IS signifying better diversity and quality and decreasing FID indicating closer similarity to real images.

Epoch	Inception Score (mean, std)	Frechet Inception Distance
1	1.32 - 0.15	338.92
500	2.25 - 0.60	194.65
1000	2.46 - 0.68	188.43

Table 2: GAN Performance Metrics at Different Epochs

4.1. Image Generation

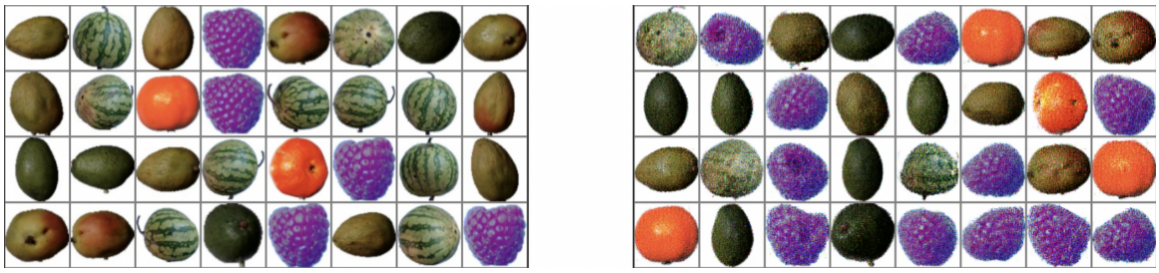

Figure 4: Selected fruit classes

Although the model has only been trained for 1000 epochs, we can see from Figure 4 how the generated images are quite similar to the originals. This result, confirms the goodness of the model.

4.2. Classification task

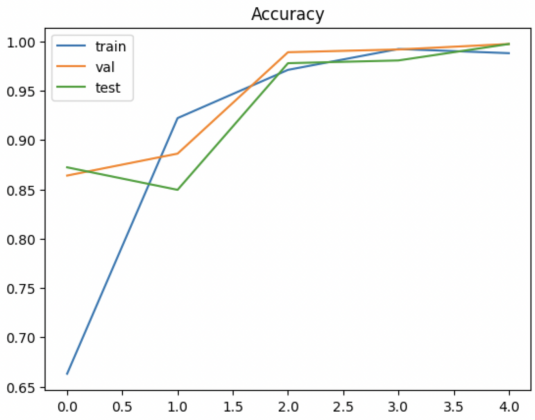

Figure 5: Classifier performance with original images

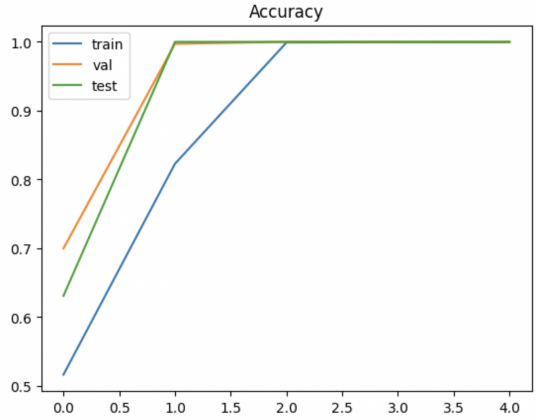

Figure 6: Classifier performance with original and generated images

As we can see from Figures 5 and 6, the classifier performed very well both on the dataset with real images and with the dataset concatenated with real images and generated by our GANs. These results demonstrate the goodness of our GAN and how the false images created are of excellent quality

References

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.