GAN Augmentation: use generative models for data augmentation.

Orazio Pontorno OPONTORNO@GMAIL.COM
Giuseppe Pulino PULINO1998@GMAIL.COM

1. Introduction

The aim of this project is to show how the use of generative models, such as Generative Adversarial Networks (GANs) and Diffusion Models, in the Data Augmentation phase can lead to an effective improvement in the performance of a classification model.
In carrying out this work, we used the 'New Plant Diseases Dataset' available on the kaggle platform. The dataset contains 25.800 images of plants divided into 38 classes, which represent diseases of particular types of plants.
Below are some examples of images used in the construction of the model.

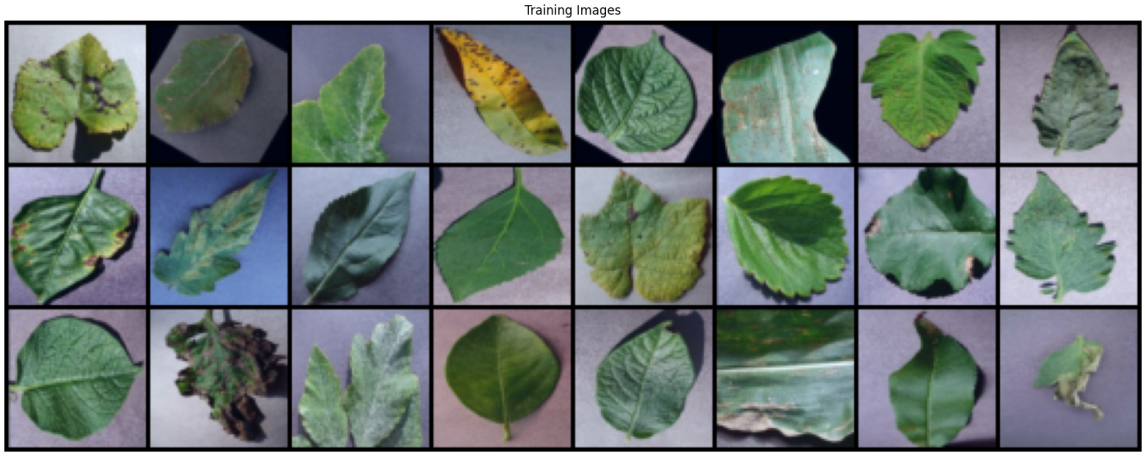


Figure 1: Training images

Initially, an *resnet18* network was trained, using the *fine-tuning* technique, in classifying different diseases using the image dataset described above. Subsequently, using the same set of images, a Deep Convolutional Generative Adversarial Network (Conditional DCGAN) was trained to reproduce the same images. The latter were subsequently added to the former by expanding the image dataset. At this point, the classifier was again trained using this new set of images. Finally, both the images and the results obtained from the classifications of the two models were compared in order to assess whether enlarging the image dataset by using the generative models led to improvements.

2. Model Description

As mentioned in the previous section, two types of neural networks were used: the ResNet18 and the *Conditional Deep Convolutional Generative Adversarial Network*.

2.1. ResNet18

ResNet18 is a convolutional neural network (CNN) architecture that belongs to the ResNet (Residual Network) family of models. It was introduced by He et al. in 2015 and is one of the smaller variants of the ResNet models. ResNet-18 has gained popularity due to its simplicity, effectiveness, and competitive performance on various computer vision tasks, including image classification.
The key characteristic of *ResNet18* is the use of residual connections, also known as skip connections or shortcut connections. These connections allow the model to learn residual mappings instead of directly learning the desired underlying mappings. This idea helps overcome the problem of vanishing gradients in deep neural networks and enables the training of very deep architectures.
The following figure shows the entire network architecture.

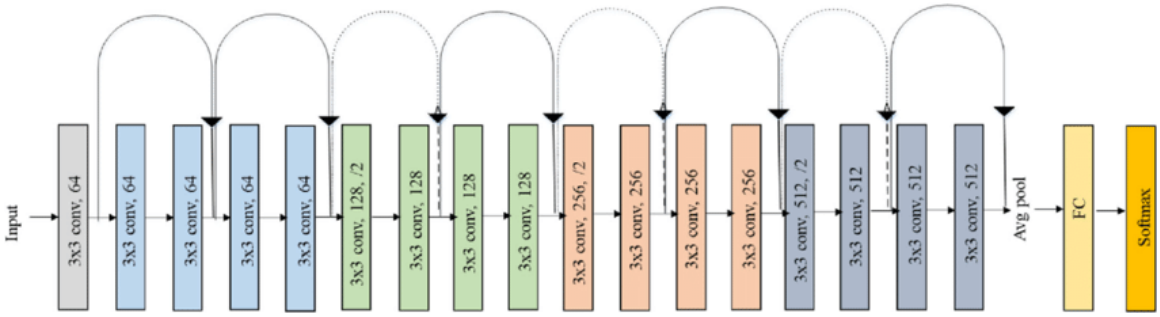


Figure 2: Architecture of ResNet18

- Basic Building Block: The basic building block of *ResNet18* consists of two convolutional layers with batch normalization and ReLU activation functions. Each convolutional layer has 3x3 filters, and the number of filters increases as we go deeper into the network.
- Residual Connections: The residual connections in *ResNet18* are introduced to address the degradation problem in deep networks. The skip connections allow the input to be directly added to the output of the convolutional layers, creating a shortcut path. This skip connection ensures that the gradient can flow directly through the network without being significantly attenuated, facilitating the training of deeper networks.

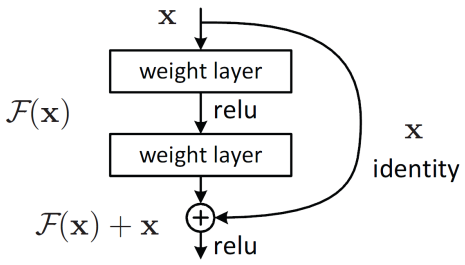


Figure 3: Resudual connection

- Shortcut Projection: In *ResNet18*, when the number of input and output channels of the residual block differs, a projection shortcut is introduced. This projection is performed using a 1x1 convolutional layer to match the dimensions of the input and output feature maps. This technique helps preserve the spatial information and ensures compatibility for the element-wise addition operation.

- Downsampling: *ResNet18* incorporates downsampling in the form of strided convolutions and max pooling. This downsampling operation reduces the spatial dimensions of the feature maps, enabling the network to capture larger receptive fields and more global features as the network depth increases.
- Pre-activation Residual Blocks: *ResNet18* uses pre-activation residual blocks, which means the batch normalization and ReLU activation are applied before the convolutional layers. This ordering has shown to improve the overall performance and convergence of the network.
- Global Average Pooling and Fully Connected Layer: Instead of using fully connected layers at the end of the network, *ResNet18* employs global average pooling. This operation reduces the spatial dimensions of the feature maps to 1x1 and averages the values across each channel. The resulting vector is then fed into a fully connected layer for classification.

ResNet18 has gained popularity due to its effectiveness in image classification tasks, especially in scenarios with limited labeled data. It has been widely used and adapted in various computer vision applications and serves as a foundation for more advanced architectures in the ResNet family.

2.2. Conditional DCGAN

A conditional DCGAN (Conditional Deep Convolutional Generative Adversarial Network) is an extension of the DCGAN (Deep Convolutional Generative Adversarial Network) framework that allows for controlled and targeted generation of images by conditioning the generator on additional input information. In a traditional DCGAN, the generator generates images from random noise. However, in a conditional DCGAN, the generator is conditioned on additional input data, such as class labels or other attributes, to generate images that correspond to specific conditions.

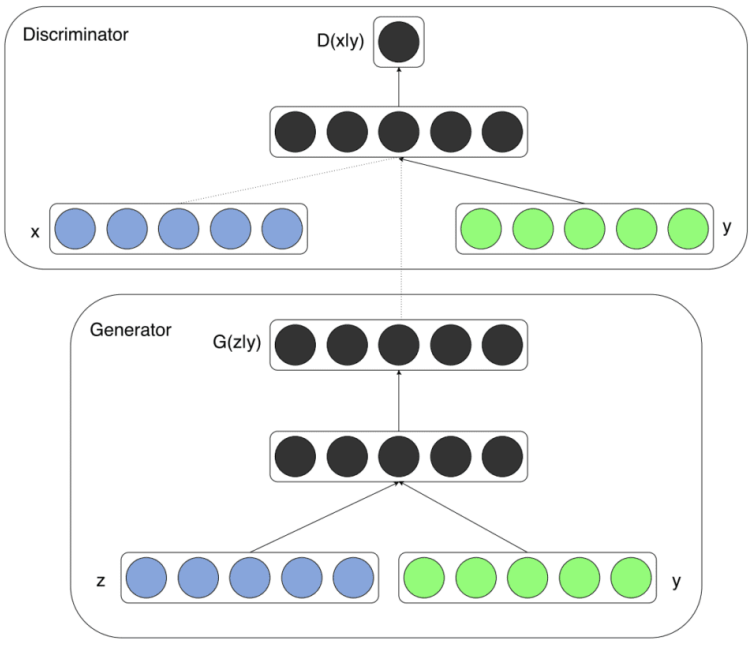


Figure 4: Architecture of cDCGAN

- Conditioning Input: In addition to random noise, the generator of a conditional DCGAN takes additional conditioning input, such as class labels or attributes. This conditioning input serves as a guidance signal for the generator, allowing it to generate images that align with the specified conditions. For example, if the class labels represent different object categories, the generator can be conditioned to generate images of specific objects based on the provided label.
- Generator Network: The generator network in a conditional DCGAN takes both the random noise and the conditioning input as inputs. These inputs are typically concatenated or combined in some way before being fed through the network. The generator then generates images that are conditioned on the provided input, resulting in controlled image synthesis.
- Discriminator Network: The discriminator network in a conditional DCGAN is responsible for distinguishing between real and generated images, as well as classifying the generated images into the appropriate categories. It takes both the image and the conditioning input as inputs and predicts the probability of the input being real or fake. The conditioning input helps the discriminator in classifying the generated images based on the specified conditions.
- Loss Functions: The loss functions used in a conditional DCGAN include the adversarial loss, which encourages the generator to generate realistic images that fool the discriminator, and the conditional loss, which ensures that the generated images align with the provided conditions. The conditional loss can be computed using various techniques, such as cross-entropy loss or mean squared error, depending on the specific problem and conditioning input.
- Training Procedure: The training procedure for a conditional DCGAN is similar to that of a traditional DCGAN. It involves iteratively training the generator and discriminator networks in an adversarial manner. The generator aims to generate realistic images that align with the conditioning input, while the discriminator aims to accurately classify real and generated images based on their authenticity and conditions.

The main advantage of conditional DCGANs is their ability to generate images that correspond to specific conditions or attributes, providing fine-grained control over the image synthesis process. This makes them useful in various applications, such as image-to-image translation, style transfer, and semantic image editing, where generating images based on specific conditions is desired. Conditional DCGANs have been widely adopted and extended to cater to different types of conditioning inputs and tasks, making them a powerful tool in the field of generative modeling.

3. Training procedure

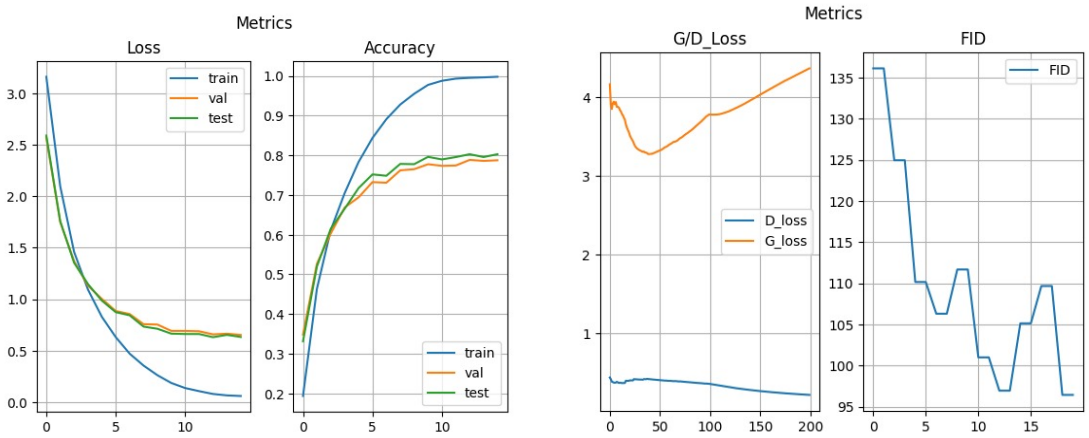
The training procedure in deep learning is the process of training a neural network model to learn from data and make accurate predictions. It involves several key steps that are repeated iteratively to optimise the performance of the model. This is referred to as the *Training loop*, i.e. iterating the training dataset in batches. For each batch, perform the following steps:

1. Forward pass: Feed the batch of input data through the network to obtain the predictions.
2. Loss calculation: Compares the model predictions with the actual labels and calculates the loss using the selected loss function.
3. Backpropagation: Calculates loss gradients with respect to model parameters. This is done by propagating the backward error gradients through the network, layer by layer.
4. Parameter update: Use the optimization algorithm to update the model parameters based on the calculated gradients. This step adjusts the weights and biases of the network, with the goal of reducing loss.

The final performance of the model depends very much on the choice of *Hyperparameters* in the training procedure. In this session, we will present the configurations of the hyperparameters used in the training phase of both the *ResNet18* network and the networks (generator and discriminator) of the *cDCGAN*. The following table summarises the hyper-parameters used in the training phase of the three networks.

	ResNet18	**Generator**	**Discriminator**
Optimizer	SGD	Adam	Adam
Loss Function	CrossEntropyLoss()	CrossEntropyLoss()	CrossEntropyLoss()
Epochs	16	200	200
Initial Learning Rate	0.001	0.001	0.001
beta1	-	0.5	0.999
beta2	-	0.5	0.999
Momentum	0.9	-	-
LRScheduler	-	step=100, gamma=0.1	step=100, gamma=0.1

The train images were passed to the networks in 64 *batch*. The learning curves drawn during the training procedure are shown below.



(*a*) ResNet18 (*b*) cDCGAN

Figure 5: Learning curves of Networks

As can be seen from the graphs, the learning curves follow an optimal trend and there do not seem to be any overfitting problems. From the training of the *ResNet18* we obtained an accuracy of 0.99 in the train set, 78 in the validation and 79 the test set; while loss values of 0.004 in the train set, 0.66 in the validation set and 0.65 in the test set were obtained. From the training of the *cDCGAN*, loss values of 4.35 for the generator and 0.42 for the discriminator were obtained. In addition, during the training phase of the generator, every 20 epochs the value of the *FID* was calculated by generating 12,000 fakes and comparing them with the same number of real images, finally obtaining a total of 97.

4. Experimental Results

We have now reached the stage where we assess whether the Data Augmentation procedure carried out using a generative network such as cDCGAN has led to any increase in the performance of the

classifier.
To do this, we again trained the network *ResNet18* by adding the images synthesised by the generative model to the original training dataset. Some of these generated images are shown below.

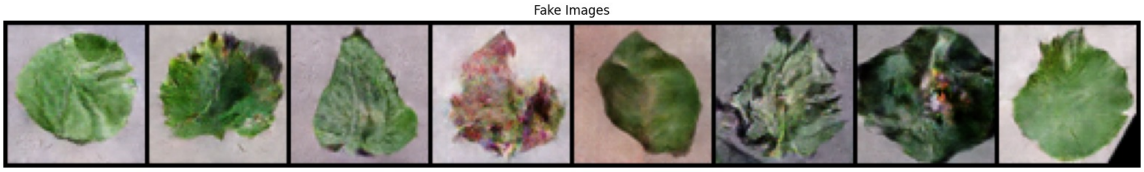


Figure 6: Images generated by cDCGAN

We specify that the classifier was trained using the same configuration of hyperparameters. Below are the results obtained from the new training procedure:

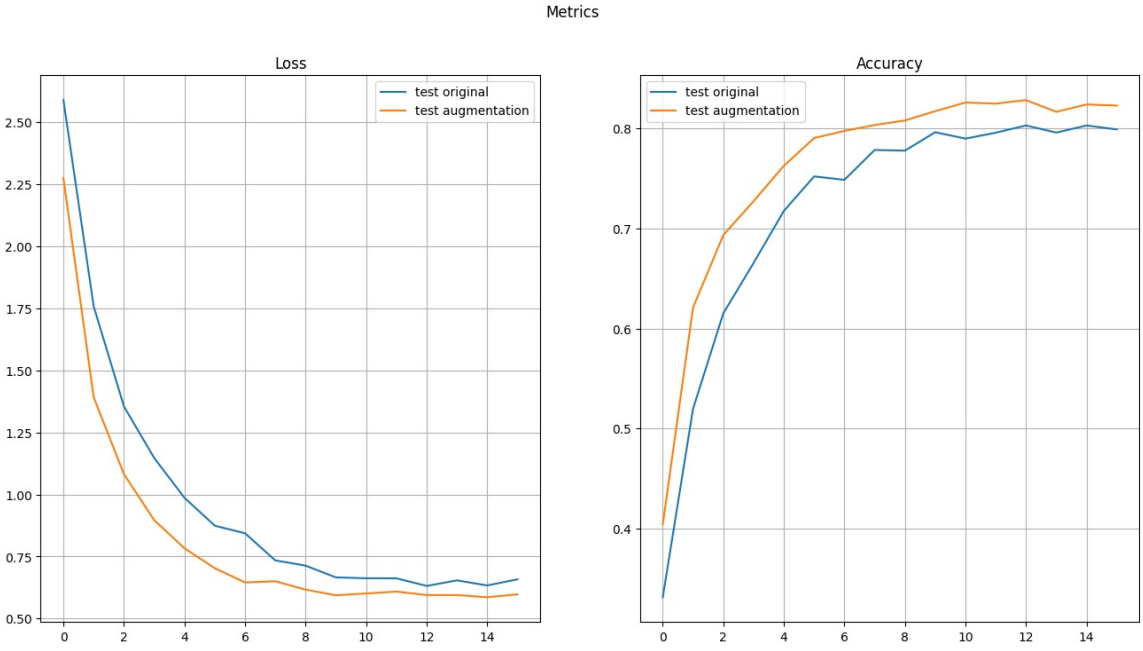


Figure 7: Comparison of the accuracies obtained before and after the addition of GAN-generated images.

As can be seen from the graphs, the addition of images generated by the *cDCGAN* resulted in an increase in accuracy in the test dataset of 4%. In fact, it went from an accuracy of 79% to an accuracy of 83%.