République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
ECOLE SUPÉRIEURE EN INFORMATIQUE
8 Mai 1945 - Sidi-Bel-Abbès

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
المدرسة العليا للإعلام الآلي
8 ماي 1945 - سيدي بلعباس

Neural Style Transfer With GAN

MINI PROJECT DEEP LEARNING

Team Members:

- Benghenima Hafsa
- Ghandouz Amina

Professor:

- Mrs Dif Nassima

Contents

List of Figures

1 Introduction:

In recent years, the intersection of computer vision and artistic creativity has produced fascinating developments, one of which is neural style transfer, which means the ability of deep learning models to manipulate or recreate artistic styles in images. Neural style transfer techniques aim to apply the visual appearance (style) of an image, typically an artwork, to the content of another image, such as a real photograph. While earlier approaches were based on optimization techniques (e.g Gatys), recent advancements leverage **Generative Adversarial Networks (GANs)** to achieve style transfer in a more realistic, efficient, and scalable way.

GANs, introduced by **Ian Goodfellow and his colleagues** in **June 2014**, are composed of two competing neural networks, a generator and a discriminator, they learn to produce increasingly realistic data through adversarial training. In the context of image-to-image translation, GANs have proven powerful in generating photorealistic outputs from sketches, semantic labels, or artworks. However, traditional GAN-based translation methods often require paired datasets which are scarce or unavailable in many real-world scenarios.

To overcome this limitation, **CycleGAN** introduced a framework that allows unpaired image-to-image translation using cycle-consistency loss. This enables learning a mapping between two domains even when corresponding image pairs do not exist.

In this project, we apply neural style transfer using **CycleGAN** to the **Monet2Photo** dataset that includes two distinct domains:

- Monet paintings (representing the impressionist artistic style of **Claude Monet**)
- Real-world photographs of landscapes and scenes similar in content

The objective is to learn two mappings:

- From Monet-style paintings to realistic photos
- From photos to Monet-style artworks

This task not only showcases the capabilities of unpaired style transfer models but also highlights how deep learning can bridge the gap between art and reality. Moreover, the Monet2Photo dataset provides a unique opportunity to experiment with the visual translation of impressionist textures, brushstrokes, and color palettes into the domain of modern photography.

2 Background and Related Work:

2.1 What is Style Transfer?:

Style Transfer is the process of merging the content of one image with the style of another, resulting in a new and recognizable image. In more technical terms, it is a method that allows to take the structure and layout of one image (*content*) and paint it in the unique style, textures, and patterns of another image (*style*), it's exactly like telling a machine, *"Make this photograph look like it was painted by Van Gogh"*.

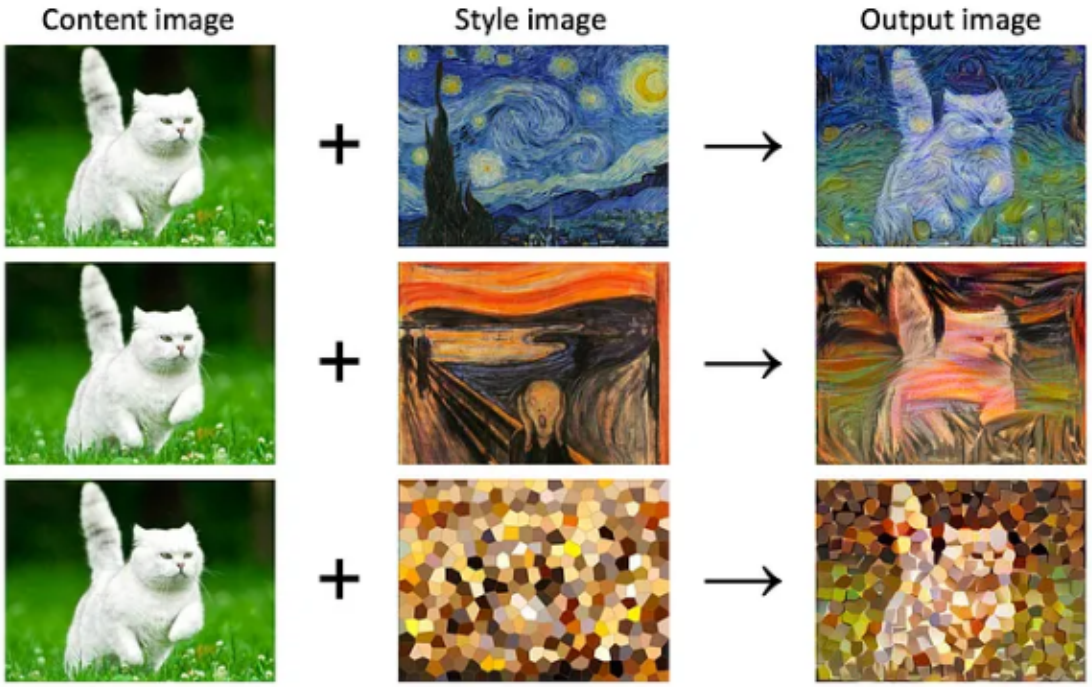

Figure 1: Style Transfer in Computer Vision.

Style transfer can be broadly categorized into:

- **Traditional Style Transfer:** Uses techniques like histogram matching, edge detection, and texture synthesis. These are often rule-based and work well only for simple cases.
- **Neural Style Transfer (NST):** In **2015**, **Gatys et al.** introduced **NST**, revolutionizing image generation by using CNNs to separate and recombine image content and style. Their method allowed computers to blend objects and artistic textures convincingly.

In the context of **style transfer**, *content* refers to the higher-level features of the image, like shapes, objects, and their spatial arrangement, it's the structure or the *what* of the image. On the other hand, style represents the lower-level features like textures, colors, and patterns, the style is the *how* or the feel of the image. For example, if we have a photo of a mountain range, the content would be the actual layout of the mountains, sky, and ground. If we apply the style of a *Van Gogh* painting, vibrant color palette would be transferred, but the mountains would still be recognizable.

In **Style Transfer**, the network needs to balance two things: *retaining the content* and *applying the style* so we use *loss functions* to measure how well it's doing this balancing act. NST optimizes for

two losses:

- **Content loss:** measures how close the generated image is to the original content.
- **Style loss:** measures how similar its textures and patterns are to the target style.

At the heart of style transfer is **feature extraction**, this is how the algorithm gets a sense of what's in the image. CNN will be as seasoned art critic that knows how to separate content from style. When an image is passed through a CNN, its early layers detect textures and colors, while deeper layers identify complex shapes and structures. This layered understanding helps distinguish between the image's content (overall layout) and style (visual patterns and textures).

2.2 What is CNN?:

Convolutional Neural Networks (CNNs) are a type of deep learning neural network architecture that is particularly well suited to image classification and object recognition tasks. A CNN starts by taking an input image, which is then transformed into a feature map through a series of convolutional and pooling layers, the convolutional layer applies a set of filters to the input image where each filter will produce a feature map that highlights a specific aspect of the input image, the pooling layer then downsamples the feature map to reduce its size, while retaining the most important information. The feature map produced by the convolutional layer is then passed through multiple additional convolutional and pooling layers, each layer learning increasingly complex features of the input image and the final output of the network is a predicted class label or probability score for each class for multi-classifiifcation tasks.

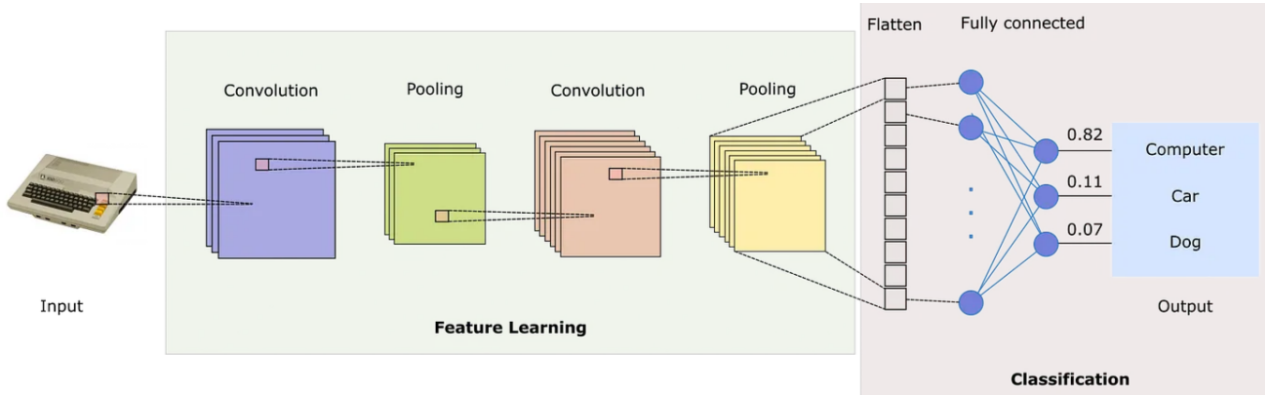

Figure 2: CNN architecture.

2.3 How do CNNs work?:

CNNs are composed of layers of artificial neurons, each responsible for transforming the input image in a specific way.

1. **Neurons:** The most basic unit in a neural network, they are composed of a sum of linear/non-linear function (the activation function).

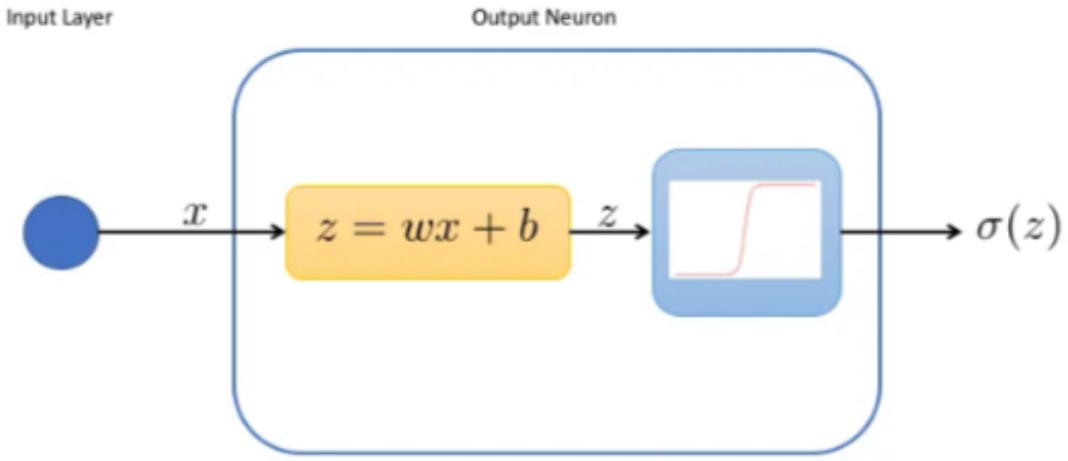

Figure 3: Neuron representation

2. **Input Layer:** each neuron in this layer corresponds to one of the input features, for example in an image classification task where the input is 32x32, the input layer would have 1024 neurons which means one neuron for each pixel.
3. **Hidden Layer:** is the layer between input and output ones, they may be more, each neuron in the hidden layer is summed by the result of the neurons in the previous layers then multiplied by non-linear function.
4. **Output Layer:** in this layer, the number of neurons corresponds to the number of the classes, for example in multi-classification task with digits 0-9, we would have 9 neurons in the output layer.

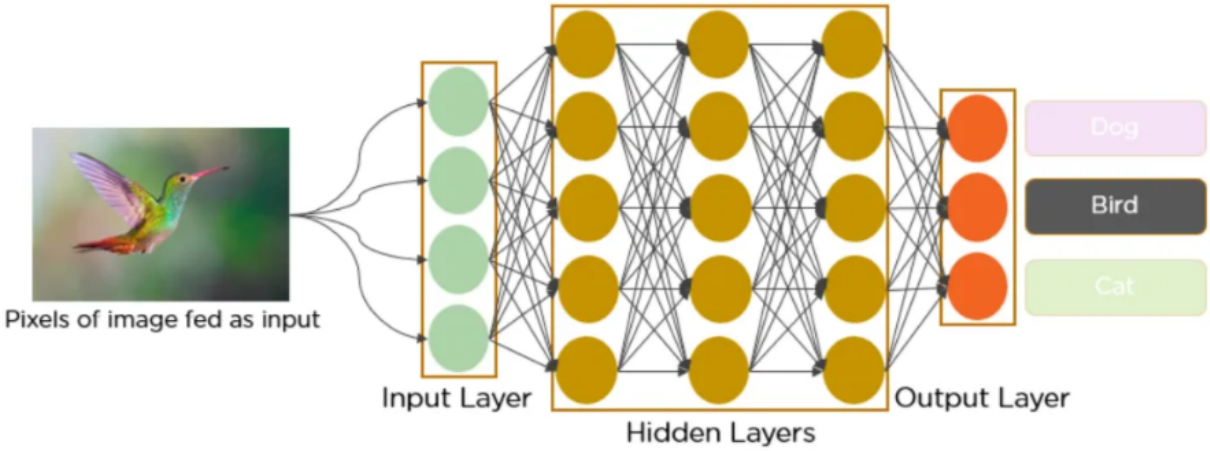

Figure 4: CNN example

Convolutional layer are what makes a CNN different from a basic neural network, they are the fundamental building blocks of CNNs, these layers perform a critical mathematical operation known as convolution. We can classify the layers of CNN into:

- **Convolutional Layer:** is responsible for extracting features from the input image, it performs a convolution operation on the input image, where a filter or kernel is applied to the image to identify and extract specific features.

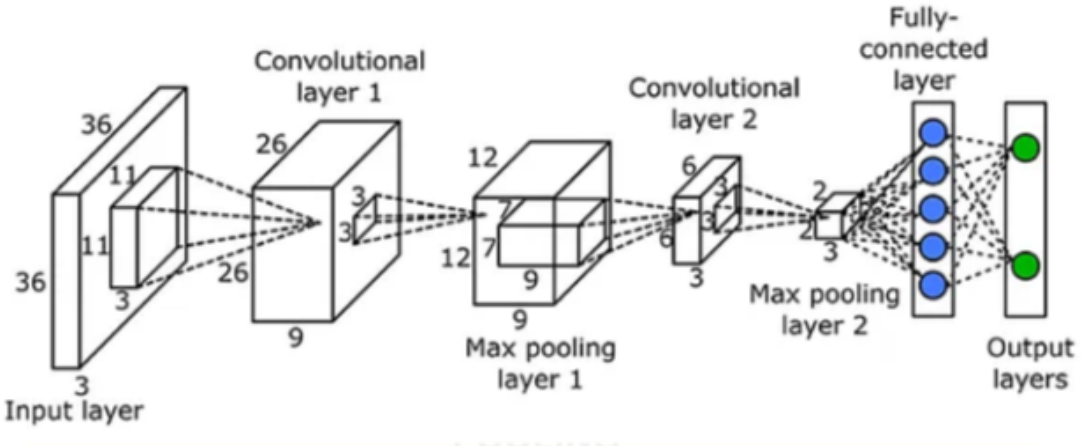

Figure 5: Convolutional Layer

- **Pooling Layer:** is responsible for reducing the spatial dimensions of the feature maps produced by the convolutional layer, it performs a down-sampling operation to reduce the size of the feature maps and reduce computational complexity.

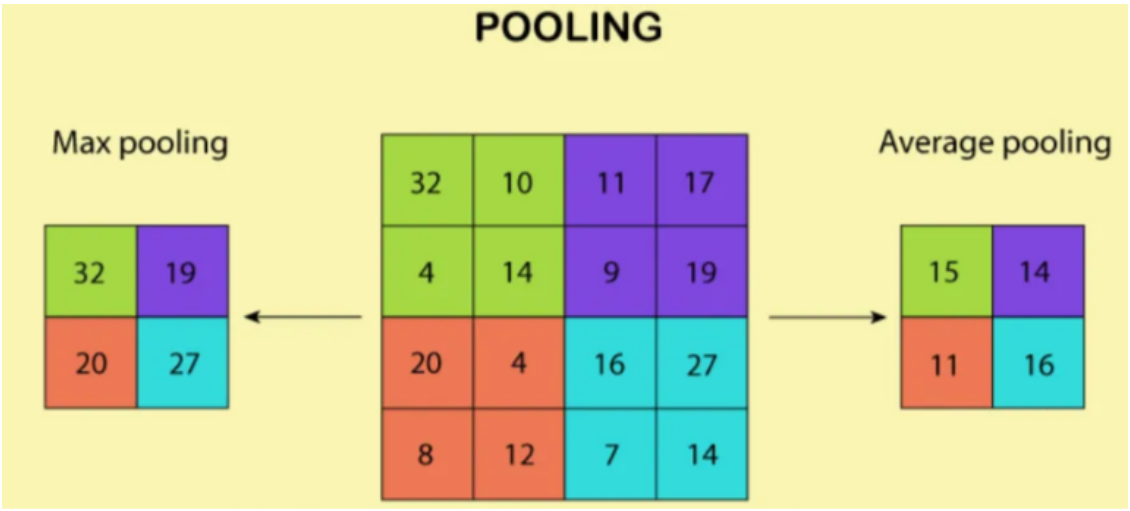

Figure 6: MaxPooling Layer

- **Activation Layer:** applies a non-linear activation function, such as the ReLU to the output of the pooling layer, this function helps to introduce non-linearity into the model, allowing it to learn more complex representations of the input data.

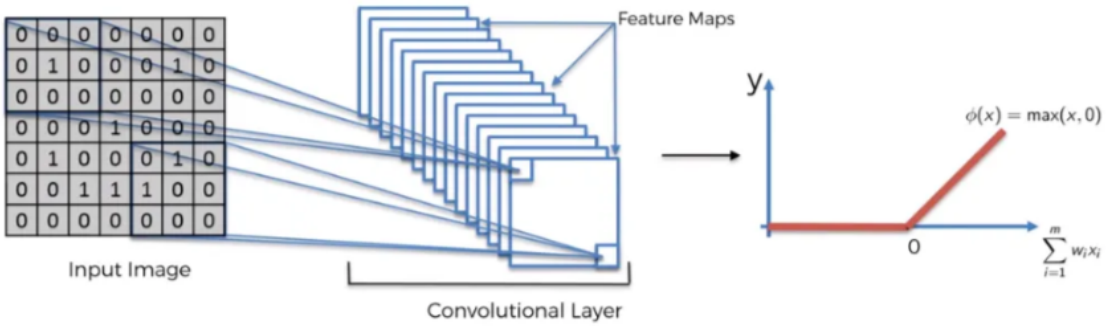

Figure 7: Activation Layer

- **Fully Connected Layer:** is a traditional neural network layer that connects all the neurons in the previous layer to all the neurons in the next layer, it's responsible for combining the features learned by the convolutional and pooling layers to make a prediction.

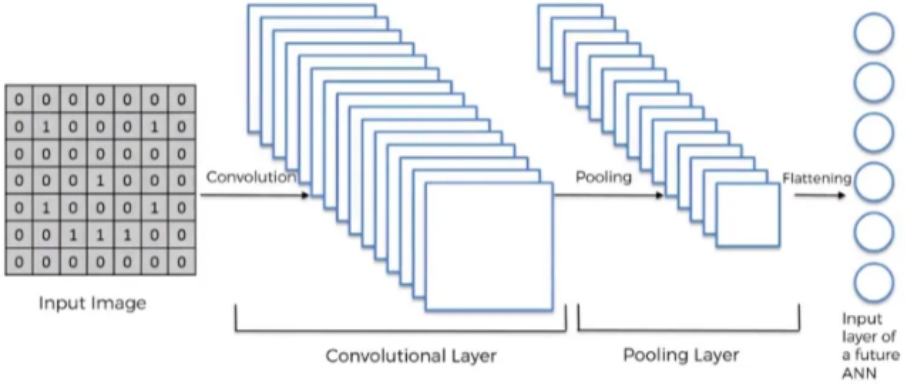

Figure 8: Fully Connected Layer

- **Normalization Layer:** performs normalization operations such as batch normalization to ensure that the activations of each layer are well-conditioned and prevent overfitting.
- **Dropout Layer:** is used to prevent overfitting by randomly dropping out neurons during training which helps to ensure that the model does not memorize the training data but instead generalizes to new, unseen data.
- **Dense Layer:** after the convolutional and pooling layers have extracted features from the input image, this layer can then be used to combine those features and make a final prediction. In a CNN, the dense layer is usually the final layer and is used to produce the output predictions. The activations from the previous layers are flattened and passed as inputs to the dense layer which performs a weighted sum of the inputs and applies an activation function to produce the final output.

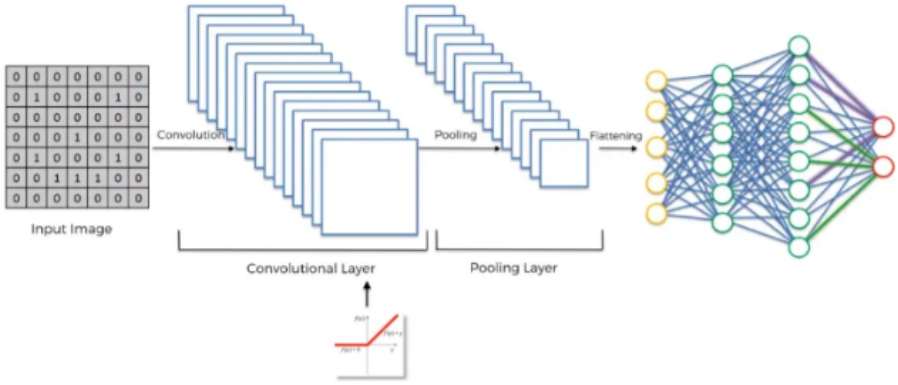

Figure 9: Dense layer

CNNs are a powerful deep learning architecture well-suited to image classification and object recognition tasks with its ability to automatically extract relevant features, handle noisy images, and leverage pre-trained models.

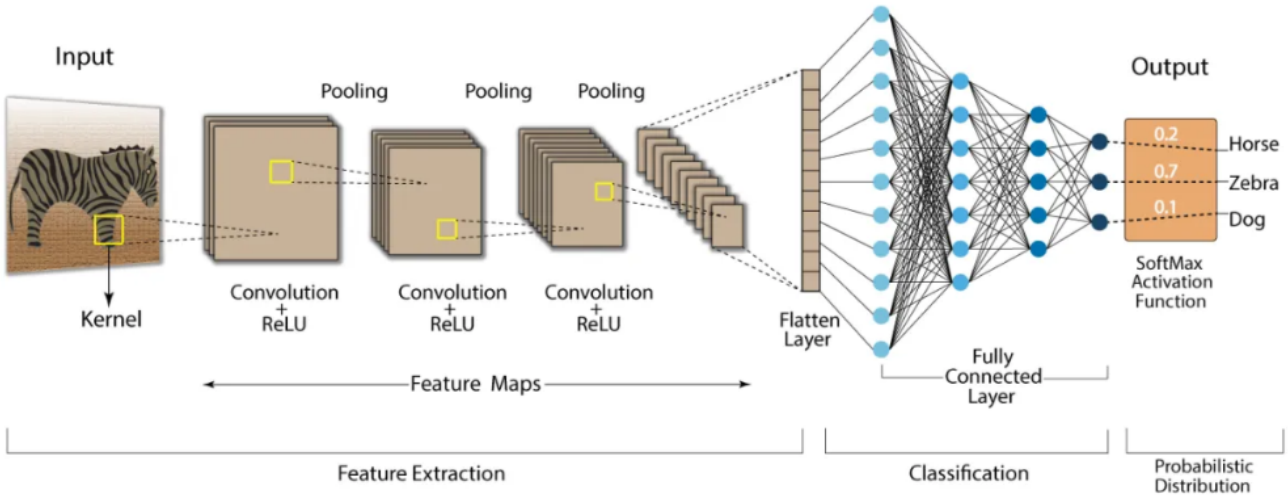

Figure 10: CNN representation

Despite their strong capabilities in feature extraction and classification tasks, *CNNs* face limitations when it comes to generating new and high-quality images. Traditional CNNs are primarily discriminative models which means they excel at recognizing and analyzing existing data, but they are not designed to create new content and this becomes a challenge in tasks like image synthesis or style transfer where realism and creativity are crucial. To overcome these limitations, **Generative Adversarial Networks** (*GANs*) were introduced. Unlike CNNs, GANs are generative models capable of producing realistic and high-resolution images by learning the underlying data distribution, making them a powerful alternative in the field of generative image modeling.

In *CNNs*, one of the most powerful features is their ability to extract hierarchical representations from images which makes them excellent for tasks like classification and style-content separation. In

GANs, especially in the generator, feature extraction is also crucial, it helps the generator learn the structure and texture of images to produce realistic outputs but there's a key difference:

- In CNNs, feature extraction is the main goal, the model is often used to understand or classify an image.
- In GANs, feature extraction is a means to an end, it helps the generator/discriminator learn how to create or judge images, not just understand them.

Which means that feature extraction is central to both, but it plays a supporting role in *GANs* (for generation and discrimination), and a primary role in *CNNs* (for recognition and analysis).

2.4 What is GAN?:

2.4.1 Introduction:

Deep learning models can be broadly classified into:

- **Generative models** learn the distribution of the data itself. They model the joint probability *P(x,y)* and can generate new data samples that are similar to the training data. An example of a generative model is a **Generative Adversarial Network** (*GAN*), which consists of a generator that creates data and a discriminator that evaluates the authenticity of the generated data.
- **Discriminative models** learn the decision boundary between different classes, they focus on modeling the conditional probability where y is the label and x is the input data. These models are used primarily for classification tasks. A common example of a discriminative model is a *CNN*, which is widely used for image classification tasks where the model classifies images into predefined categories.

2.4.2 What is GAN?:

Generative Adversarial Networks are a class of deep learning frameworks designed by **Ian Goodfellow and his colleagues** in *2014*. They consist of two interlinked neural networks:

- **the generator:** responsible for creating data from random noise.

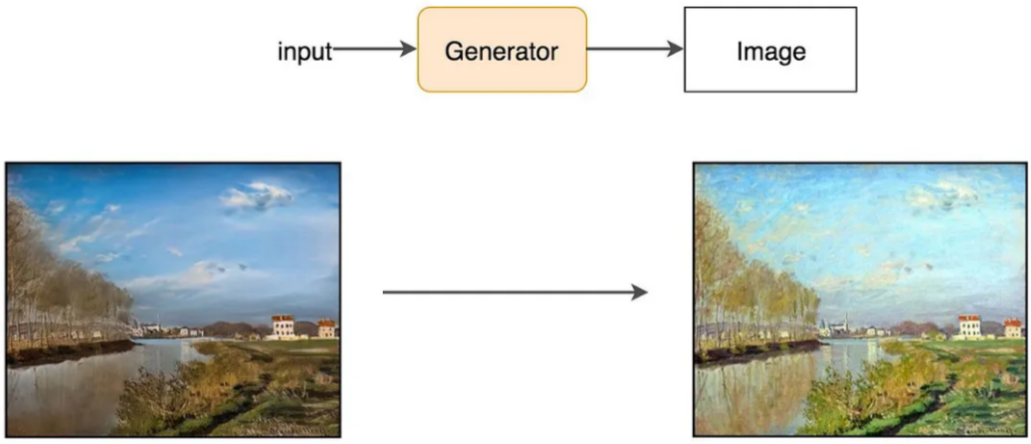

Figure 11: Generator example

- **the discriminator:** whose task is to differentiate between genuine and simulated data.

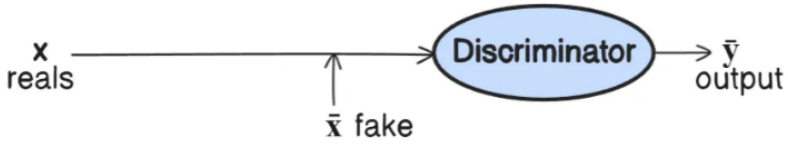

Figure 12: Discriminator example

They are trained simultaneously by competing against each other and this interplay results in a unique training process, with GANs operating in an "adversarial" manner formulated as a supervised learning challenge, the *Generator* tries to produce data that the *Discriminator* can't distinguish from real data, while it tries to get better at differentiating real data from fake data. The two networks are in essence competing in a game: the *Generator* aims to produce convincing fake data, and the *Discriminator* aims to tell real from fake which leads to the Generator creating increasingly better data over time and allows GANs to produce highly realistic synthetic data.

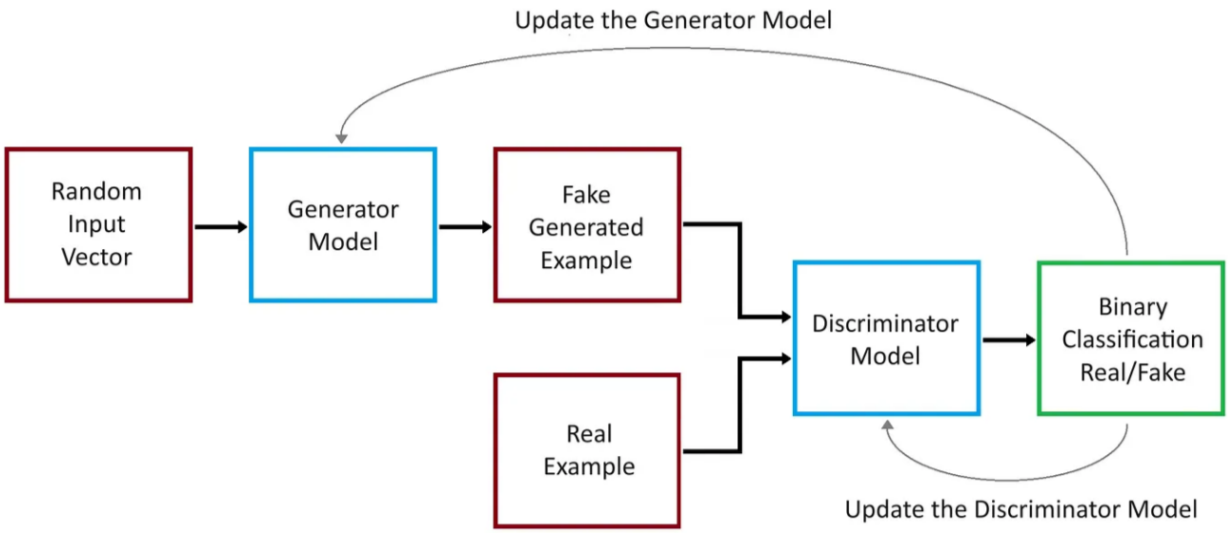

Figure 13: GAN Architecture

2.4.3 Loss Functions:

In *GANs*, two main loss functions guide the training process:

- **The discriminator loss:** evaluates how well the discriminator can distinguish between real and generated (fake) data, it's trained to maximize the probability of correctly identifying real images as real and generated images as fake.
- **The generator loss:** measures how successfully the generator can fool the discriminator, the generator aims to minimize this loss by producing images that are realistic enough to be classified as real by the discriminator.

The interplay between these two losses creates a dynamic adversarial training process where both models improve simultaneously.

2.4.4 Variants of GANs:

- **Conditional GANs (C-GANs):** GANs that generate data based on additional input information (e.g., class labels, text, or attributes) which allows control over the generated output.
- **CycleGAN:** Designed for image-to-image translation tasks where paired training data is not available (e.g., translating horses to zebras without having exact matching images).
- **DC-GAN:** Introduces convolutional and transposed convolutional layers in place of fully connected layers for better image quality and more stable training.
- **Wasserstein GAN (W-GAN):** Uses the *Wasserstein distance* instead of the traditional loss, which improves training stability and helps mitigate mode collapse (where the generator produces limited varieties of outputs).

2.5 What is CycleGAN?:

CycleGAN is a powerful variant of *Generative Adversarial Networks* (*GANs*) that enables image-to-image translation without the need for paired training data. Traditional supervised GAN approaches require aligned image pairs (e.g., a photo and its corresponding painting), which can be expensive or impossible to obtain. So, what *CycleGAN* does differently from a standard *GAN* is that it doesn't generate images from random noise, instead, it uses a given image to get a different version of that image, this is the image-to-image translation that allows *CycleGAN* to change a horse into a zebra. However, image-to-image translation is not a feature that is unique to *CycleGAN* but it's one of the first models to allow for unpaired image-to-image training. What that means is that we don't have to have a picture of a horse and a picture of what that horse would look like as a zebra in the dataset. Instead, we can have a bunch of horses and a bunch of zebras separately. This is useful in situations where we aren't able to get paired data.

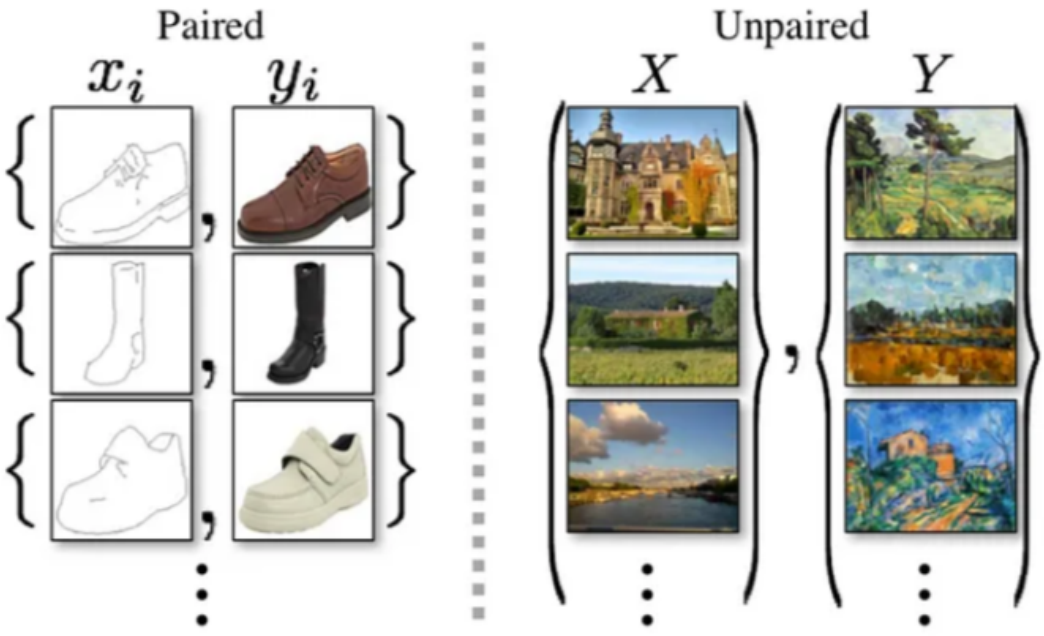

Figure 14: Paired and unpaired dataset

While traditional *GANs* consist of a single generator and discriminator and require paired datasets, *CycleGAN* removes the need for aligned image pairs, instead, it uses two generators (one for each direction of translation, e.g., A $\rightarrow$ B and B $\rightarrow$ A) and two discriminators (one for each domain). A key innovation in *CycleGAN* is the *cycle consistency loss*, which ensures that translating an image to the target domain and back results in the original image (i.e., A $\rightarrow$ B $\rightarrow$ A $\approx$ A). This architecture allows *CycleGAN* to learn meaningful mappings between domains even without direct correspondence, making it particularly useful in real-world applications like artistic style transfer, medical imaging, and domain adaptation.

2.6 Artistic Style Transfer: Transforming Photographs into Paintings:

One popular application of *CycleGAN* is in the transformation of artistic images, specifically, making real-world photographs appear as if they were painted by renowned artists. When we look at a photo, we might ask ourselves:

How would Monet paint this scene?

3 Dataset Description:

The **Monet2Photo dataset** is a well-known dataset used for artistic style transfer, specifically designed for testing the effectiveness of *CycleGAN* in transforming photos into the style of famous artists like *Claude Monet*. It consists of two primary domains: *Monet* and *Photo* images.

Monet2Photo dataset consists of *8231* images, *1193* (*14%*) *Monet Paintings* and *7038* (*86%*) *Natural Photos* which it's splitted into train and test subsets where training set size is *89%* and only *11%* for the test set. We have two main folders, trainA and testA refer to *Monet images* where trainB and testB refer to *Photo images* with *256x256* images size.

In training set, we have:

```
Total images in `/trainA`: 1072
Most common image sizes (in sample): [((256, 256), )]
Total images in `trainB`: 6287
Most common image sizes (in sample): [((256, 256), )]
```

Figure 15: Train images

3.1 Sample Images:

Figure 16: Monet images

Figure 17: Photo images