SUMMER PROJECT 2024
Science and Technology Council
IIT Kanpur

CYCLEGANS-TRANSLATING IMAGES

A project by IITK CONSULTING GROUP

Documentation

Date of Submission:
July 17, 2024

CONTENTS

INTRODUCTION

- Problem Statement
- Objective

1. Basics Of Machine Learning
    - 1.1 Key Python Libraries
    - 1.2 Regression and Classification
    - 1.3 K-Means Clustering
    - 1.4 Principal Component Analysis(PCA)
    - 1.5 Closed Form Solutions
2. Artificial Neural Network(ANN)
    - 2.1 Structure
    - 2.2 Limitations
3. Convolutional Neural Network(CNN)
    - 3.1 Structure
    - 3.2 Limitations
4. Generative Adversarial Network(GAN)
    - 4.1 Introduction to GANs
    - 4.2 Deep Convolutional GANs
    - 4.3 Conditional GANs
    - 4.4 Pix2Pix
    - 4.5 PatchGAN
    - 4.6 CycleGANs

INTRODUCTION

Problem Statement

CycleGANs (Cycle-Consistent Generative Adversarial Networks) represent a remarkable breakthrough in the field of image-to-image translation. The primary objective of the CycleGAN project is to perform high quality image-to-image translation without relying on paired datasets. Traditional GANs (Generative Adversarial Networks) require a large dataset of corresponding images between the source and target domains, which is often impractical or impossible to obtain. CycleGANs address this limitation by using unpaired image data, making them extremely versatile and applicable to a wide range of tasks such as style transfer, photo enhancement, and domain adaptation.

Objective

- Understanding basics of Machine Learning- Regression and Classification, K-means Clustering, PCA,etc.
- Dive into Artificial Neural Network(ANN).
- In-depth implementation of Convolutional Neural Network(CNN).
- Understanding Generative Adversarial Network(GANs)- Deep Convolutional GAN, Pix2Pix Model, CycleGAN.

1. BASICS OF MACHINE LEARNING

Introduction to Machine Learning

Machine Learning (ML) is a field of artificial intelligence where computers learn from data to make decisions or predictions without being explicitly programmed. Think of it like teaching a computer to recognize patterns and make decisions based on those patterns. Instead of writing specific rules for the computer to follow, we feed it data, and it learns from that data.

1.1 Key Python Libraries for Machine Learning

NumPy: This library is essential for numerical computations, providing powerful tools for handling large arrays and matrices, allowing for efficient mathematical operations.

Pandas: Used for data manipulation and analysis, Pandas offers DataFrames that resemble tables in spreadsheets, making it easy to clean, sort, and filter data.

Matplotlib and Seaborn: Matplotlib is a fundamental library for creating a wide range of visualisations, while Seaborn builds on Matplotlib, providing a more aesthetic approach to statistical graphics. These libraries help in uncovering patterns and trends in datasets through visual exploration.

Scikit-Learn: This powerful library simplifies the implementation of various machine learning algorithms, such as classification and regression. It also provides tools for model evaluation and selection, making it a popular choice for many ML practitioners.

TensorFlow and Keras: TensorFlow is an open-source framework for deep learning, enabling the creation of complex neural networks. Keras, which runs on top of TensorFlow, offers a user-friendly API to simplify the process of building and training these models, making deep learning more accessible.

1.2 Regression and Classification

1.2.1 Regression

Regression analysis is a statistical method used to examine the relationships between a dependent variable and one or more independent variables. It helps in understanding how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed.

1.2.2 Types of Regression

1. **Linear Regression**

   Simple Linear Regression: Involves a single independent variable.

   Multiple Linear Regression: Involves multiple independent variables.

2. **Logistic Regression**

   Used for binary classification problems (where the dependent variable is categorical).

1.2.3 Key Concepts

Dependent and Independent Variables

- **Dependent Variable (Y)**: The variable being predicted or explained.
- **Independent Variables (X)**: The variables used to predict or explain the dependent variable.

Model Parameters

- **Coefficients (β\betaβ)**: Quantify the relationship between each independent variable and the dependent variable.
- **Intercept (β0\beta_0β0)**: The expected value of YYY when all XXX variables are zero.
- **Error Term (ϵ\epsilonϵ)**: Represents the part of YYY that the model cannot explain.

1.2.4 Assumptions of Regression Models

1. **Linearity**: The relationship between the dependent and independent variables is linear.
2. **Independence**: Observations are independent.
3. **Homoscedasticity**: Constant variance of the errors.
4. **Normality**: Errors are normally distributed (for linear regression).
5. **No Multicollinearity**: Independent variables are not highly correlated.

1.2.5 Linear Regression Model

Simple Linear Regression Model

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Multiple Linear Regression Model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$

- Y: Dependent variable
- $X_1, X_2, \ldots, X_n$: Independent variables
- β_0: Intercept
- $\beta_1, \beta_2, \ldots, \beta_n$: Coefficients (slopes for each independent variable)
- ϵ: Error term

1.2.6 Steps to Perform Regression Analysis

1. Data Collection

Gather data for the dependent and independent variables.

2. Data Preprocessing

- Handle missing values.
- Encode categorical variables.
- Normalize the data if necessary.

3. Model Building

Use statistical software or programming languages (like Python or R) to build the regression model.

In Python (using `scikit-learn`):

```
from sklearn.linear_model import LinearRegression

# Create a regression object

model = LinearRegression()

# Fit the model

model.fit(X_train, y_train)
```

4. Model Evaluation

Evaluate the model using various metrics:

- **R-squared (R^2)**: Proportion of variance explained by the model.
- **Mean Squared Error (MSE)**: Average of squared differences between actual and predicted values.
- **Root Mean Squared Error (RMSE)**: Square root of the average of squared differences.
- **Mean Absolute Error (MAE)**: Average of absolute differences between actual and predicted values.

5. Model Interpretation

Interpret the coefficients to understand the relationship between the dependent and independent variables.

6. Prediction

Use the model to predict the dependent and independent variables.

1.2.7 Metrics

Common Evaluation Metrics

1. R-squared (R^2)

$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

- SS_{res}: Sum of squares of residuals
- SS_{tot}: Total sum of squares

2. Mean Squared Error (MSE)

$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$

- Y_i: Actual values
- $\hat{Y}_i$: Predicted values

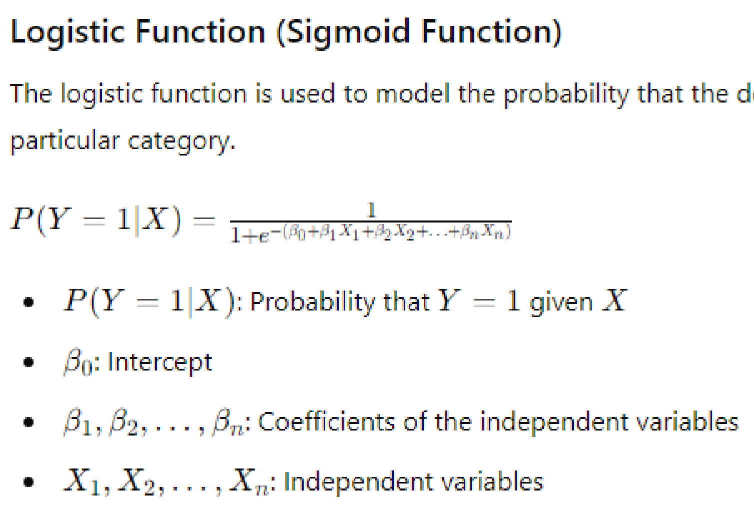
3. Root Mean Squared Error (RMSE)

$RMSE = \sqrt{MSE}$

4. Mean Absolute Error (MAE)

$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$

1.2.8 Logistic Regression



Logistic Function (Sigmoid Function)

The logistic function is used to model the probability that the dependent variable belongs to a particular category.

$$P(Y=1|X)=\frac{1}{1+e^{-(\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta_nX_n)}}$$

- $P(Y=1|X)$: Probability that $Y=1$ given X
- β_0: Intercept
- $\beta_1,\beta_2,\ldots,\beta_n$: Coefficients of the independent variables
- $X_1,X_2,\ldots,X_n$: Independent variables

1.2.9 Assumptions of Logistic Regression

1. **Binary Outcome**: The dependent variable should be binary.
2. **Independence of Observations**: Observations should be independent of each other.
3. **Linearity of Independent Variables and Log-Odds**: There should be a linear relationship between the independent variables and the log-odds.
4. **No Multicollinearity**: Independent variables should not be highly correlated.

1.2.10 Steps to Perform Logistic Regression

1. Data Collection

Collect data for the dependent and independent variables.

2. Data Preprocessing

- Handle missing values.
- Encode categorical variables.
- Normalize/standardize the data if necessary.

3. Model Building

Use statistical software or programming languages (like Python or R) to build the logistic regression model.

In Python (using `scikit-learn`):

```
from sklearn.linear_model import LogisticRegression

# Create a logistic regression object
```

```
model = LogisticRegression()

# Fit the model

model.fit(X_train, y_train)
```

4. Model Evaluation

Evaluate the model using metrics such as:

- **Accuracy**: Proportion of correctly predicted observations.
- **Precision**: Proportion of true positive predictions among all positive predictions.
- **Recall (Sensitivity)**: Proportion of true positive predictions among all actual positives.
- **F1 Score**: Harmonic mean of precision and recall.
- **ROC-AUC Score**: Area under the Receiver Operating Characteristic curve.

5. Model Interpretation

Interpret the coefficients to understand the relationship between the independent variables and the log-odds of the dependent variable.

6. Prediction

Use the model to predict the probability of the dependent variable for new data.

1.3 K-Means Clustering

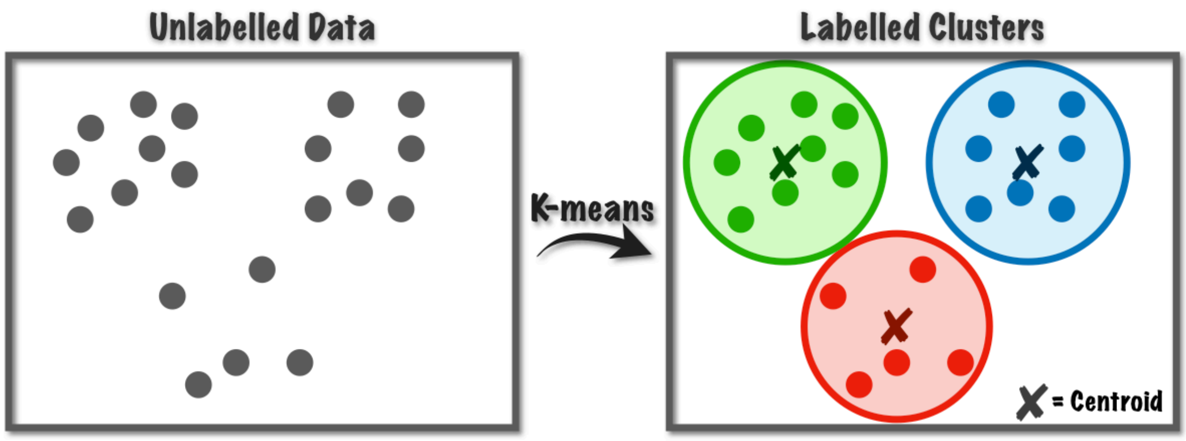
1.3.1 Introduction to K-Means Clustering

What is Clustering?

Clustering is a type of unsupervised learning where we aim to group a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). It's a common technique for statistical data analysis used in many fields, including machine learning, pattern recognition, and image analysis.

What is K-Means Clustering?

K-Means is one of the simplest and most popular clustering algorithms. The main idea behind K-Means is to divide n data points into k clusters in which each data point belongs to the cluster with the nearest mean.



1.3.2 How K-Means Clustering Works

The K-Means algorithm follows these steps:

1. **Initialize**: Select k initial cluster centers (centroids) randomly.
2. **Assign**: Assign each data point to the closest cluster center.
3. **Update**: Recompute the cluster centers as the mean of all data points assigned to each cluster.
4. **Repeat**: Repeat steps 2 and 3 until the cluster centers do not change significantly or a maximum number of iterations is reached.

Intuition Behind K-Means

Imagine you have a set of points on a plane. You want to group these points into k clusters. You start by placing k markers at random positions (these are your initial centroids). Each point then decides which marker it is closest to and "joins" that cluster. You then move each marker to the average position of all the points that joined its cluster. This process is repeated until the markers stop moving significantly.

Mathematical Details

Let's break down the steps mathematically:

1. Initialize Cluster Centers

Choose k initial cluster centres $\mu1,\mu2,...,\mu k$.

2. Assign Data Points to Closest Cluster

For each data point xi, assign it to the cluster with the nearest centroid:

$C_i = \arg\min_j \|x_i - \mu_j\|^2$

where C_i is the cluster assigned to data point x_i, and $\|x_i - \mu_j\|^2$ is the squared Euclidean distance between x_i and μ_j.

3. Update Cluster Centers

Recompute the cluster centers as the mean of all data points assigned to each cluster:

$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$

where $|C_j|$ is the number of data points in cluster C_j.

4. Repeat

Repeat steps 2 and 3 until convergence.

1.3.3 Example of K-Means Clustering

Let's consider a simple example with 2D data points:

{(2,3),(3,3),(6,6),(8,8),(1,0),(9,11)}.

Step-by-Step Execution

1. **Initialize**: Suppose we choose k=2 and initialize the centroids randomly, say μ1=(2,3) and μ2=(6,6).
2. **Assign**:
   - Data point (2, 3) is closer to μ1.
   - Data point (3, 3) is closer to μ1.
   - Data point (6, 6) is closer to μ2.
   - Data point (8, 8) is closer to μ2.
   - Data point (1, 0) is closer to μ1.
   - Data point (9, 11) is closer to μ2.
3. So, clusters are: C1={(2,3),(3,3),(1,0)} and C2={(6,6),(8,8),(9,11)}.
4. **Update**:

   New centroid for C_1: $\mu_1 = \left(\frac{2+3+1}{3}, \frac{3+3+0}{3}\right) = (2, 2)$.

   New centroid for C_2: $\mu_2 = \left(\frac{6+8+9}{3}, \frac{6+8+11}{3}\right) = (7.67, 8.33)$.

5. **Repeat**:
   - Reassign points based on new centroids and update until centroids stop changing significantly.

1.3.4 Advantages and Limitations

Advantages

1. **Simplicity**: K-Means is easy to understand and implement.
2. **Efficiency**: It works well with large datasets.

Limitations

1. **Choosing k**: The number of clusters k needs to be specified in advance.
2. **Sensitivity to Initialization**: Different initializations can lead to different results.
3. **Not suitable for all shapes**: K-Means assumes clusters are spherical and equally sized, which is not always the case.

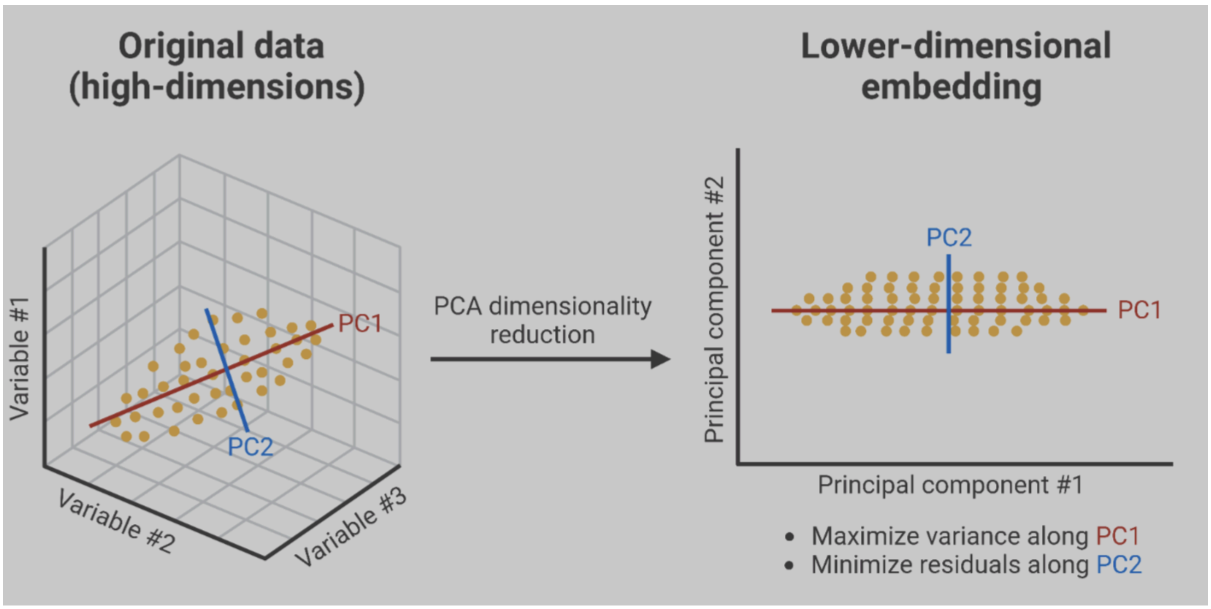
K-Means clustering is a powerful and intuitive algorithm for partitioning datasets into clusters. Its simplicity and efficiency make it a popular choice for many applications. However, it's important to be aware of its limitations, such as the need to specify the number of clusters in advance and its sensitivity to initial centroid positions.

By understanding the mathematical foundations and practical considerations of K-Means, you can effectively apply this technique to a wide range of clustering problems.

1.4 Principal Component Analysis(PCA)

1.4.1 Introduction to PCA

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction and data visualisation. It transforms high-dimensional data into a lower-dimensional form while preserving as much variability as possible.


Original data (high-dimensions)
Variable #1
Variable #2
Variable #3
PC1
PC2
PCA dimensionality reduction
Lower-dimensional embedding
Principal component #2
Principal component #1
PC2
PC1
• Maximize variance along PC1
• Minimize residuals along PC2

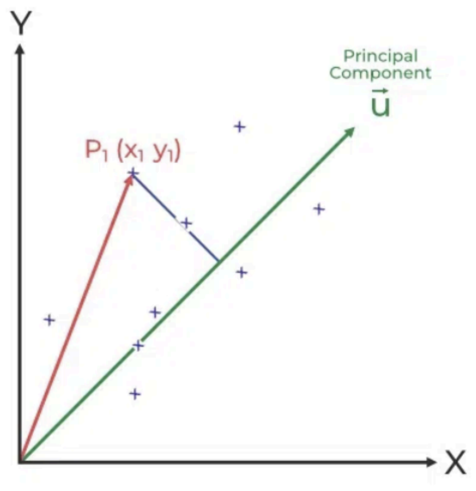
1.4.2 How PCA Works

1. **Standardise the Data**:
   - Centre the data by subtracting the mean of each feature.
   - Scale the data to unit variance by dividing by the standard deviation.

$$Z = \frac{X-\mu}{\sigma}$$

2. **Compute the Covariance Matrix**:
   - Calculate the covariance matrix to capture the relationships between features.
   - $$cov(x1, x2) = \frac{\sum_{i=1}^{n}(x1_i-\bar{x1})(x2_i-\bar{x2})}{n-1}$$

3. **Calculate Eigenvalues and Eigenvectors**:
   - Determine the eigenvalues and eigenvectors of the covariance matrix.
   - Eigenvectors indicate the direction of the principal components.
   - Eigenvalues reflect the variance explained by each principal component.
4. **Sort Eigenvalues and Select Principal Components**:
   - Arrange the eigenvalues in descending order.
   - Choose the top k eigenvectors corresponding to the largest eigenvalues to form the new feature space.
5. **Transform the Data**:
   - Project the original data onto the new feature space using the selected eigenvectors to obtain the principal components.



$$\text{Proj}_{P_1}\vec{u} = \frac{P_1 \cdot \vec{u}}{|u|}$$
$$= P_1 \cdot \vec{u} \quad .....\vec{u} \rightarrow \text{Unit Vector}$$

1.4.3 Advantages and Limitations

Advantages of Principal Component Analysis (PCA)

1. Dimensionality Reduction: Simplifies data analysis, improves performance, and enhances visualisation by reducing the number of variables.
2. Feature Selection: Identifies and selects the most important variables in a dataset, beneficial for machine learning.
3. Data Visualization: Reduces high-dimensional data to 2D or 3D for easier interpretation.
4. Multicollinearity Handling: Creates uncorrelated variables, helping to manage multicollinearity in regression analysis.
5. Noise Reduction: Enhances the signal-to-noise ratio by removing low-variance components assumed to be noise.
6. Data Compression: Reduces storage requirements and speeds up processing by using fewer principal components.
7. Outlier Detection: Identifies outliers by detecting data points far from others in the principal component space.

Disadvantages of Principal Component Analysis (PCA)

1. Interpretation of Principal Components: Difficult to interpret linear combinations of original variables.
2. Data Scaling Sensitivity: Requires proper scaling of data for effective results.
3. Information Loss: Potential loss of information depending on the number of principal components retained.
4. Non-linear Relationships: Assumes linear relationships, which may not capture non-linear relationships well.
5. Computational Complexity: Can be computationally expensive for large datasets with many variables.
6. Overfitting: Risk of overfitting if too many principal components are used or if the dataset is small.

1.5 Closed Form Solutions in Machine Learning

1.5.1 Introduction

In machine learning, the process of training a model often involves finding the optimal parameters that minimize a certain cost or loss function. This optimization can be performed using various methods, including iterative techniques like gradient descent or analytical approaches that yield closed-form solutions. Closed-form solutions offer several advantages, including computational efficiency and clarity in understanding the underlying mathematical relationships.

A closed-form solution is an explicit formula that can be directly computed using a finite sequence of arithmetic operations, such as addition, subtraction, multiplication, division, and root extractions. This contrasts with iterative methods, which approach the solution through successive approximations.

1.5.2 Examples of Closed-Form Solutions

1. **Linear Regression**: One of the most well-known closed-form solutions is found in linear regression. Given a set of training data (X,y) where X is the matrix of input features and y is the vector of target values, the goal is to find the weight vector www that minimizes the mean squared error (MSE).
   The closed-form solution for www is given by:
   $$w = (X^T X)^{-1} X^T y$$
2. **Ordinary Least Squares (OLS)**: OLS is a method for estimating the unknown parameters in a linear regression model. The closed-form solution for the parameter vector β is derived by minimizing the sum of squared residuals.
   The solution is:
   $$\beta = (X^T X)^{-1} X^T y$$
3. **Ridge Regression**: Ridge regression addresses the problem of multicollinearity by adding a regularization term to the OLS objective function. The closed-form solution for ridge regression, where λ is the regularization parameter, is:
   $$w = (X^T X + \lambda I)^{-1} X^T y$$

1.5.3 Advantages and limitations of Closed-Form Solutions

Advantages

- **Efficiency**: Closed-form solutions can be computed quickly and are typically more efficient than iterative methods, especially for large datasets.
- **Simplicity**: They provide clear and concise formulas, making the models easier to understand and analyze.
- **Deterministic**: Unlike iterative methods, closed-form solutions do not depend on initial values or convergence criteria, providing consistent and repeatable results.

Limitations

- **Existence**: Closed-form solutions do not exist for all types of models or loss functions, especially for complex or non-linear models.
- **Scalability**: For very large datasets, computing the inverse of matrices (as required in some closed-form solutions) can be computationally expensive and numerically unstable.

2. ARTIFICIAL NEURAL NETWORK(ANN)

Artificial Neural Networks (ANNs) are computational models inspired by the human brain's neural structure, designed to process complex data and identify patterns.

These networks consist of interconnected nodes (neurons) organized into layers—input, hidden, and output—where each neuron processes information by receiving inputs, applying weights to them, and producing outputs through activation functions.

2.1 Structure

The structure of an Artificial Neural Network (ANN) is inspired by the biological neural networks of the human brain but simplified and organized in a mathematical and computational framework.

2.1.1 Neurons (Nodes)

- Neurons are the basic computational units of an ANN. They receive inputs, process information, and produce outputs. Each neuron is analogous to a biological neuron and performs a specific function based on the inputs it receives.

2.1.2 Layers

- ANNs are typically organized into layers:
    - **Input Layer:** This layer receives input data, which could be raw data or features extracted from the original data.
    - **Hidden Layers:** These layers perform computations on the input data. Each neuron in a hidden layer is connected to every neuron in the previous layer and applies weights to the inputs.
    - **Output Layer:** The final layer of neurons that produce the output of the network based on the computations performed in the hidden layers. The number of neurons in the output layer depends on the type of problem being solved (e.g., classification, regression).

2.1.3 Connections(Weights and Biases)

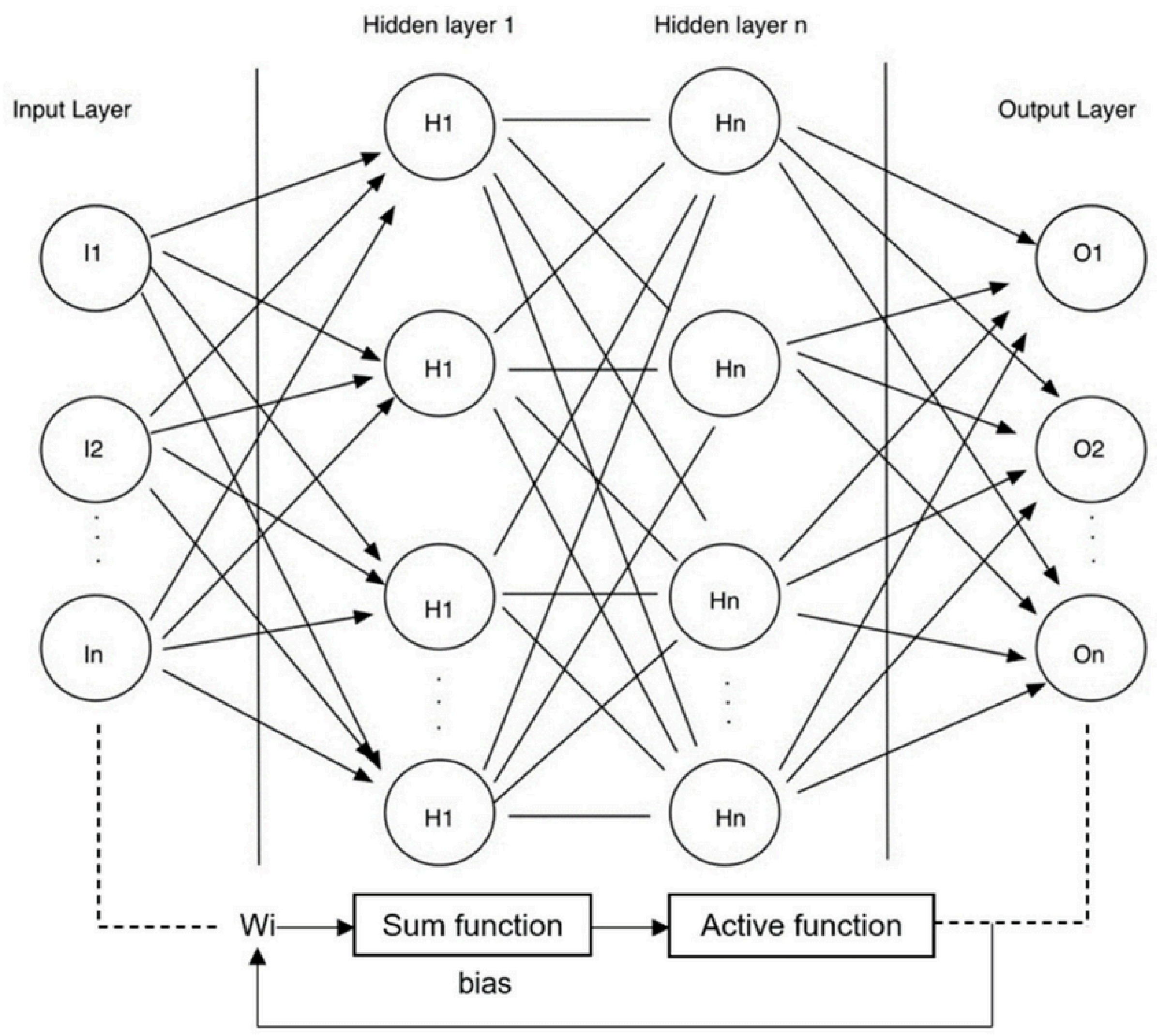
- Neurons in adjacent layers are connected by connections, each associated with a weight that determines the strength of influence between connected neurons. These weights are adjusted during the training process to minimize error and improve the network’s performance.
- Additionally, each neuron typically has an associated bias term that allows the network to model relationships between inputs and outputs more flexibly.

2.1.4 Activation Functions

- Activation functions are applied to the output of each neuron in a layer to introduce non-linearity into the network. This non-linearity is crucial for ANNs to learn complex patterns and relationships in data that are not linearly separable.

2.1.5 Training and Learning

- ANNs learn through a training process where they are presented with labeled examples (supervised learning) or unlabelled data (unsupervised learning). During training, the network adjusts its weights and biases using algorithms like backpropagation to minimize the difference between predicted and actual outputs.



2.2 Limitations of ANNs

- **Computational Resources:** ANNs can be computationally intensive, especially for large-scale problems with millions of parameters. Training deep networks requires significant computational resources, including high-performance GPUs or TPUs.
- **Lack of Time Series Handling:**Traditional feedforward ANNs are not inherently suited for processing sequential data, such as time series or sequences with temporal dependencies. Recurrent Neural Networks (RNNs) and other specialized architectures are often preferred for such tasks.
- **Interpretability:** ANNs lack transparency in their decision-making process. Understanding how and why they arrive at specific predictions or classifications can be challenging.

3. CONVOLUTIONAL NEURAL NETWORK(CNN)

Convolutional Neural Networks (CNNs) are a specialized type of artificial neural network designed specifically for processing grid-like data, such as images and videos.

3.1 Structure

3.1.1 Convolutional Layers

- **Convolution Operation:** CNNs derive their name from the convolution operation, where a filter (also known as a kernel) slides over the input data (e.g., an image), performing element-wise multiplication and summation to produce a feature map. This operation captures spatial hierarchies of features such as edges, textures, and patterns.
- **Filters:** Convolutional layers consist of multiple filters that extract different features from the input data. Each filter detects specific patterns by learning spatial relationships between pixels.

3.1.2 Pooling (Subsampling) Layers

- **Pooling Operation:** After convolution, pooling layers downsample the feature maps by reducing their spatial dimensions (e.g., width and height). This process helps in reducing computational complexity and controlling overfitting.
- **Types of Pooling:** Common pooling methods include max pooling (retaining the maximum value within a window), average pooling (taking the average value within a window), and global pooling (aggregating the entire feature map into a single value).

3.1.3 Activation Functions

- Similar to other neural networks, CNNs use activation functions (e.g., ReLU - Rectified Linear Unit) to introduce non-linearity into the network. This allows CNNs to model complex relationships and learn intricate patterns from data.

3.1.4 Fully Connected Layers

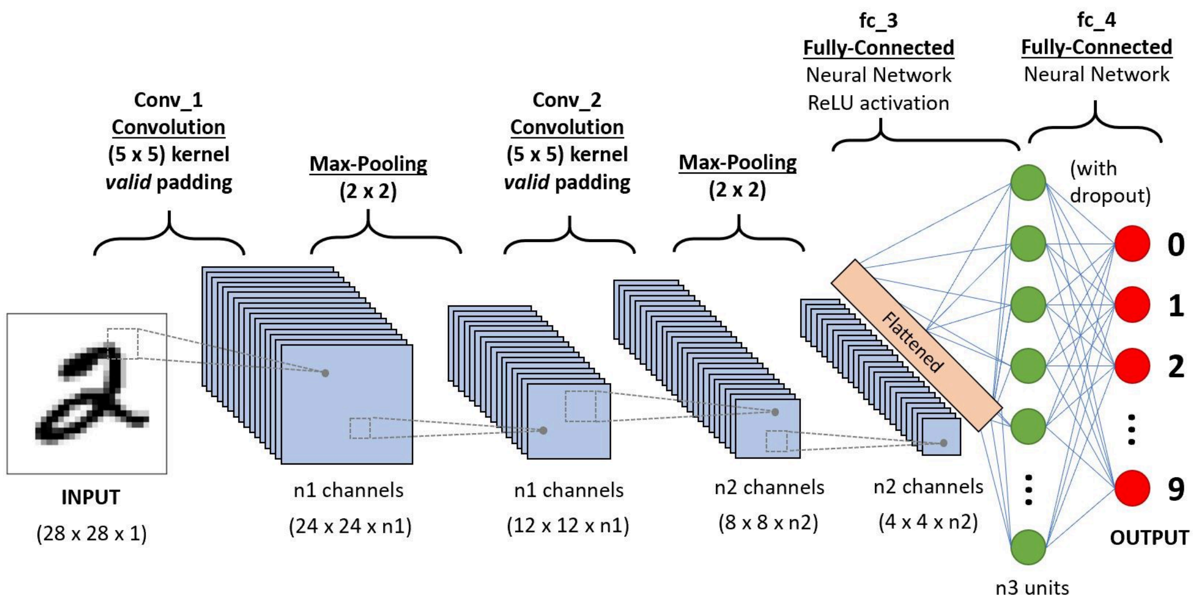
- After several convolutional and pooling layers, the extracted features are flattened into a vector and fed into one or more fully connected layers. These layers resemble traditional neural networks, where each neuron is connected to every neuron in the previous layer.
- Fully connected layers perform high-level reasoning and decision-making based on the extracted features, ultimately producing the final output (e.g., class scores in image classification).

3.1.5 Training and Learning

- CNNs are trained using labeled datasets through supervised learning techniques. The network learns by adjusting the weights of filters and fully connected layers using optimization algorithms like stochastic gradient descent (SGD) or its variants. Backpropagation is employed to propagate errors and update weights to minimize the difference between predicted and actual outputs.

3.1.6 Applications

- CNNs excel in tasks such as image classification, object detection, facial recognition, medical image analysis, and more. Their ability to automatically learn hierarchical representations of features makes them highly effective in computer vision applications.



3.2 Limitations

- **CNNs do not encode position and orientation:** CNNs detect patterns based on local features but lack explicit information about exact object positions or orientations within images, limiting precise spatial localization without additional processing.
- **Limited spatial invariance:** While CNNs are somewhat translation-invariant, they struggle with broader spatial transformations like scaling or rotation, impacting performance in scenarios requiring robustness to such variations.
- **High data dependency:** CNNs require extensive labeled data for effective generalization, crucial for learning diverse object representations and avoiding biases that can affect performance in real-world applications

4. GENERATIVE ADVERSARIAL NETWORK (GAN)

4.1 Introduction to GANs

4.1.1 What is a Generative Adversarial Network?

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. GANs are made up of two neural networks, **a discriminator and a generator.** They use adversarial training to produce artificial data that is identical to actual data.

- The Generator attempts to fool the Discriminator, which is tasked with accurately distinguishing between produced and genuine data, by producing random noise samples.
- Realistic, high-quality samples are produced as a result of this competitive interaction, which drives both networks toward advancement.
- GANs are proving to be highly versatile artificial intelligence tools, as evidenced by their extensive use in image synthesis, style transfer, and text-to-image synthesis.

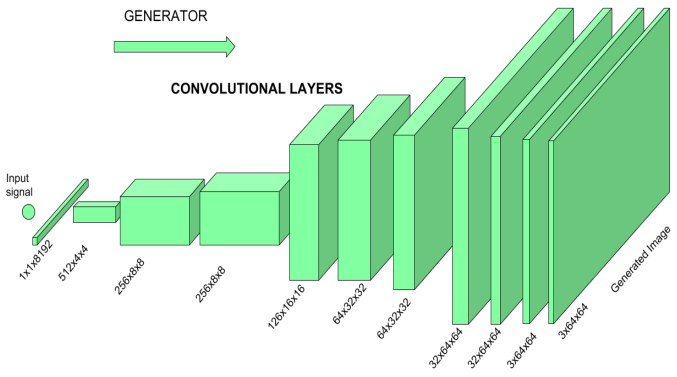
Generative Adversarial Networks (GANs) can be broken down into three parts:

- Generative: To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- Adversarial: The word adversarial refers to setting one thing up against another. This means that, in the context of GANs, the generative result is compared with the actual images in the data set. A mechanism known as a discriminator is used to apply a model that attempts to distinguish between real and fake images.
- Networks: Use deep neural networks as artificial intelligence (AI) algorithms for training purposes.

4.1.2 Architecture of GANs

Generator Model

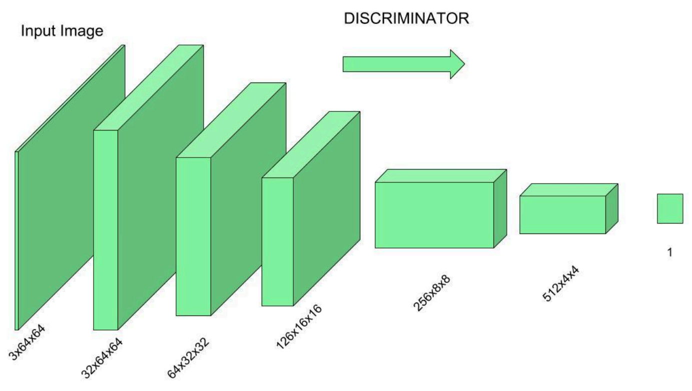
A key element responsible for creating fresh, accurate data in a Generative Adversarial Network (GAN) is the generator model. The generator takes random noise as input and converts it into complex data samples, such text or images. It is commonly depicted as a deep neural network.



The training data's underlying distribution is captured by layers of learnable parameters in its design through training. The generator adjusts its output to produce samples that closely mimic real data as it is being trained by using backpropagation to fine-tune its parameters.

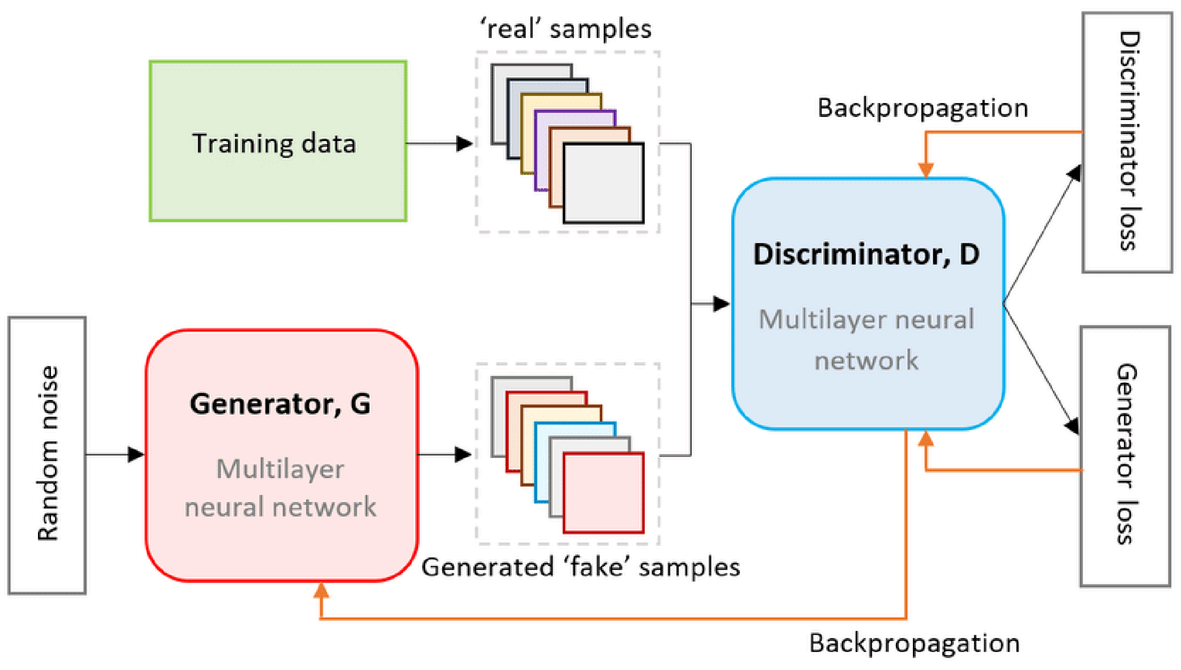
Discriminator Model

An artificial neural network called a discriminator model is used in Generative Adversarial Networks (GANs) to differentiate between generated and actual input. By evaluating input samples and allocating probability of authenticity, the discriminator functions as a binary classifier.



Over time, the discriminator learns to differentiate between genuine data from the dataset and artificial samples created by the generator.

Convolutional layers or pertinent structures for other modalities are usually used in its architecture when dealing with picture data. The discriminator grows increasingly discriminating as a result of the generator and discriminator's interaction, which helps the GAN produce extremely realistic-looking synthetic data overall



MinMax Loss

In a Generative Adversarial Network (GAN), the minimax loss formula is provided by:

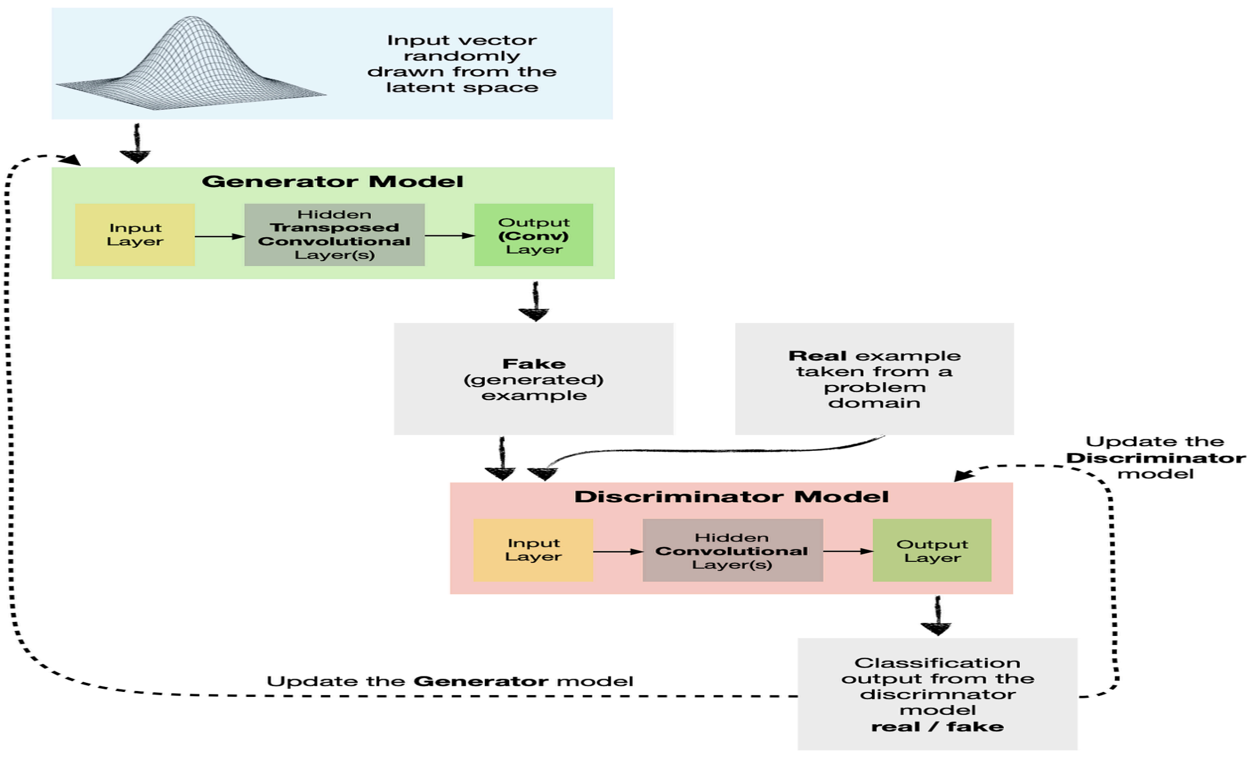
$minGmaxD(G,D)=[Ex\sim pdata[logD(x)]+Ez\sim pz(z)[log(1-D(g(z)))]$ Where,

- G is generator network and is D is the discriminator network
- Actual data samples obtained from the true data distribution $pdata(x)$ are represented by x.
- Random noise sampled from a previous distribution $pz(z)$(usually a normal or uniform distribution) is represented by z.
- D(x) represents the discriminator's likelihood of correctly identifying actual data as real.
- D(G(z)) is the likelihood that the discriminator will identify generated data coming from the generator as authentic.

4.2 Deep Convolutional GANs (DCGANs)

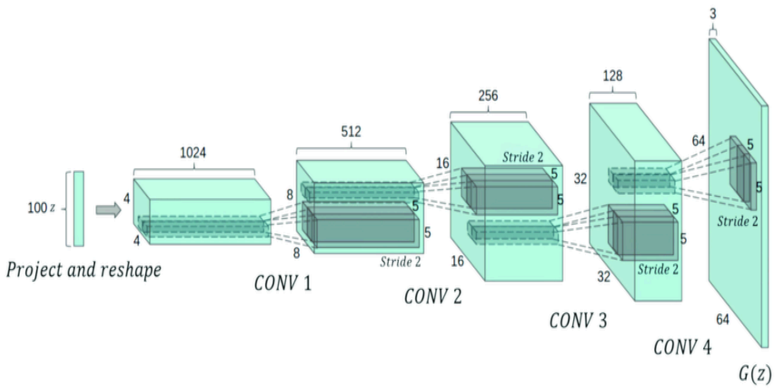
DCGANs, or Deep Convolutional Generative Adversarial Networks, are a type of Generative Adversarial Network (GAN) that leverage deep convolutional networks to generate realistic images.

4.2.1 Architecture of DCGANs


Input vector randomly drawn from the latent space
Generator Model
Input Layer
Hidden Transposed Convolutional Layer(s)
Output (Conv) Layer
Fake (generated) example
Real example taken from a problem domain
Update the Discriminator model
Discriminator Model
Input Layer
Hidden Convolutional Layer(s)
Output Layer
Classification output from the discrimnator model real / fake
Update the Generator model

Generator Model

The Generator in a DCGAN is designed to transform a random noise vector into a realistic image through a series of fractional-strided convolutions (also known as transposed convolutions or deconvolutions).The detailed description of the layers in the Generator architecture:



1. Input Layer

- A random noise vector z.Typically, z is sampled from a uniform or normal distribution.
- The length of the noise vector is usually smaller than the size of the generated images

2. Dense Layer

- **Fully Connected Layer**: The noise vector z is passed through a dense (fully connected) layer.
- The dense layer transforms the input noise vector into a higher-dimensional space.
- The output of the dense layer is reshaped into a 4D tensor with dimensions [batch_size,depth,height,width].

3. Reshape Layer

- **Reshape**: The output from the dense layer is reshaped into a 4D tensor to form the initial low-resolution activation map.

4. Fractional-Strided Convolutional Layers (Deconvolutions)

These layers progressively increase the spatial dimensions (height and width) of the activation maps while reducing the depth, generating the final image. Each fractional-strided convolutional layer is usually followed by batch normalization and an activation function (ReLU).

Generator Loss

The Generator's loss function (L(G)) is derived from the Discriminator's output when processing generated images:

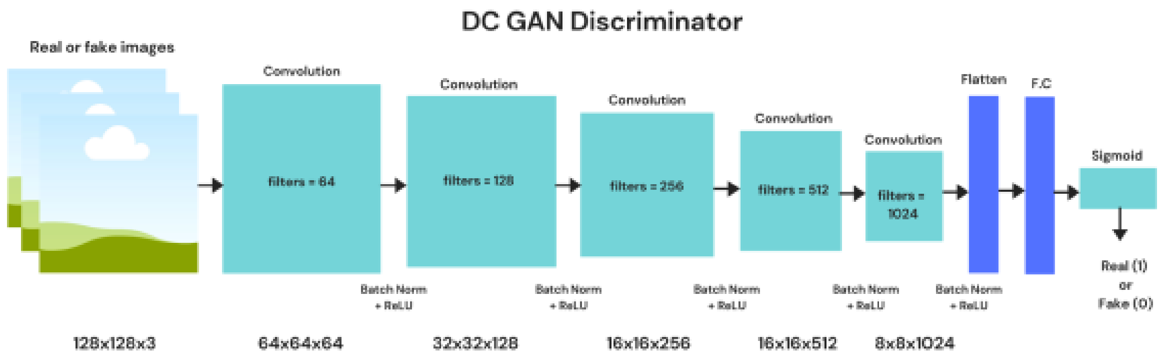
L(G)=−Ez~pz(z)[logD(G(z))]

where:

- z is a noise vector sampled from a prior distribution.
- G(z) is the Generator's output image generated from noise z.
- D(G(z)) is the Discriminator's output probability that the generated image G(z) is real.

Discriminator Model

The Discriminator in a DCGAN is a convolutional neural network (CNN) designed to distinguish between real images from the training dataset and fake images generated by the Generator. The Discriminator outputs a single scalar value representing the probability that the input image is real.



1. Input Layer: An image, either real or generated.

2. Convolutional Layers

Each convolutional layer applies a convolution operation followed by a non-linear activation function. Batch normalization is used after the first layer to improve training stability.

3. Fully Connected Layer

- **Flatten Layer**: The output from the last convolutional layer is flattened into a vector.
- **Dense Layer**: A fully connected layer that outputs a single scalar value.
- **Activation Function**: Sigmoid
  - The output of the dense layer is passed through a sigmoid activation function to produce a probability between 0 and 1.
  - The probability indicates whether the input image is real (close to 1) or fake (close to 0).

Discriminator Loss

The Discriminator's task is to distinguish between real images from the training dataset and fake images generated by the Generator. The Discriminator loss function is designed to maximize the probability of correctly classifying real and fake images.

The Discriminator's loss function (L(D)) is the sum of two terms:

- The loss when the Discriminator correctly classifies real images as real.
- The loss when the Discriminator correctly classifies generated images as fake.

Mathematically, the Discriminator loss can be defined as:

$$L(D)=-E_{x\sim p_{data}(x)}[\log D(x)]-E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

where:

- x is a real image from the training dataset.
- z is a noise vector sampled from a prior distribution (e.g., normal distribution).
- $D(x)$ is the Discriminator's output probability that x is real.
- $G(z)$ is the Generator's output image generated from noise z.

4.2.2 Applications of DCGANs

1. Image Generation and Synthesis

- **Art and Design**: DCGANs are used to create original artworks, generate new designs, and enhance creativity by providing unique visual content.
- **Fashion and Textiles**: Designers use DCGANs to generate new fashion designs, patterns, and textiles, enabling rapid prototyping and exploration of new styles.

2. Data Augmentation

- **Medical Imaging**: DCGANs generate synthetic medical images (e.g., MRI, CT scans) to augment limited datasets, aiding in training more robust and accurate models for disease diagnosis and treatment planning.
- **Object Detection and Classification**: By generating additional training data, DCGANs help improve the performance of models in tasks such as object detection and image classification.

3. Super-Resolution and Image Enhancement

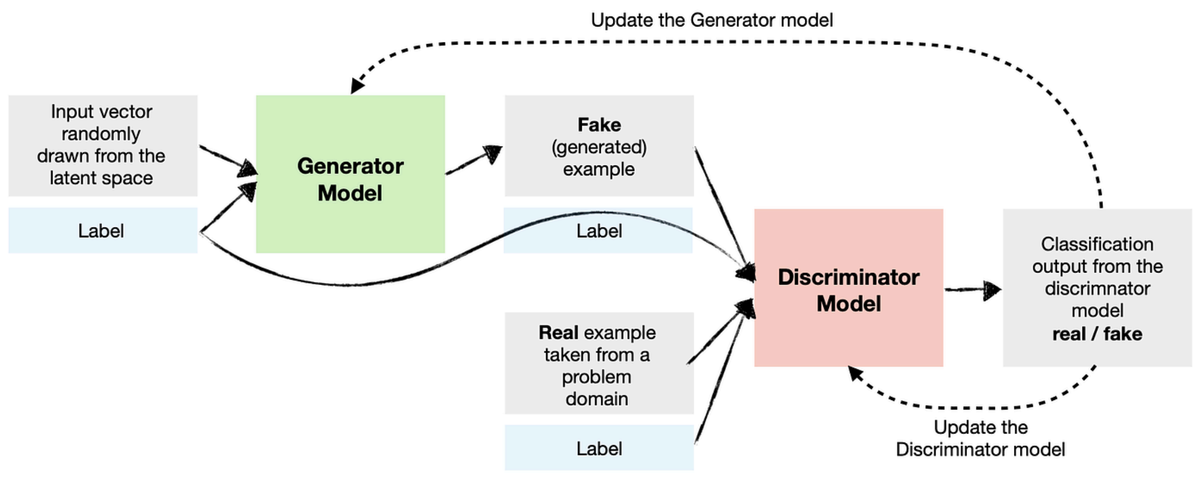
- **Image Super-Resolution**: DCGANs enhance the resolution of low-quality images, making them sharper and more detailed. This is useful in applications like satellite imagery, security footage, and medical imaging.
- **Image Denoising**: DCGANs can remove noise from images, improving their clarity and quality, which is beneficial for enhancing old or degraded photographs.

4.3 Conditional GANs(cGANs)

Conditional GANs (cGANs) are a variant of Generative Adversarial Networks (GANs) where both the generator and discriminator are conditioned on some additional information, such as class labels or data from other modalities. This conditioning allows for more control over the generated output, enabling the generation of specific types of data.

4.3.1 Architecture of cGANs

Generator Model: The Generator in a cGAN is designed to produce data that matches the conditioning information. The architecture is similar to a standard GAN but incorporates additional input that influences the generation process.


Update the Generator model
Input vector randomly drawn from the latent space
Label
Generator Model
Fake (generated) example
Label
Discriminator Model
Classification output from the discrimnator model real / fake
Real example taken from a problem domain
Label
Update the Discriminator model

1. **Input Layer**
   - **Noise Vector**: A random noise vector z typically sampled from a uniform or normal distribution.
   - **Conditioning Information**: A label or image y that conditions the generation process.

2. **Dense Layer**
   - **Fully Connected Layer**: Combines the noise vector z and the conditioning information y through a fully connected layer to produce a higher-dimensional representation.
   - **Reshape Layer**: Reshapes the output of the dense layer into a 4D tensor.

3. **Convolutional Layers**
   - **Fractional-Strided Convolutional Layers**: These layers progressively increase the spatial dimensions of the activation maps, similar to DCGANs.

Generator Loss The loss function for the Generator (L(G)) is conditioned on both the noise vector z and the conditioning information y:

$$L(G) = -\mathbb{E}_{z\sim p_z(z),y\sim p(y)}[\log D(G(z|y), y)]$$

Discriminator Model: The Discriminator in a cGAN distinguishes between real and fake data while considering the conditioning information.

1. **Input Layer**
   - **Image and Conditioning Information**: Takes an image and the corresponding conditioning information y.

2. **Convolutional Layers**
    - **Standard Convolutional Layers**: Applies convolutions followed by activation functions and batch normalization.

3. **Fully Connected Layer**
    - **Flatten Layer**: Flattens the output from the last convolutional layer.
    - **Dense Layer**: Outputs a single scalar value representing the probability that the input image is real.

Discriminator Loss The Discriminator's loss function (L(D)) includes terms for both real and generated images, conditioned on y:

$$L(D) = -\mathbb{E}_{x\sim p_{data}(x),y\sim p(y)}[\log D(x|y)] - \mathbb{E}_{z\sim p_z(z),y\sim p(y)}[\log(1 - D(G(z|y), y))]$$

4.3.2 Applications of cGANs

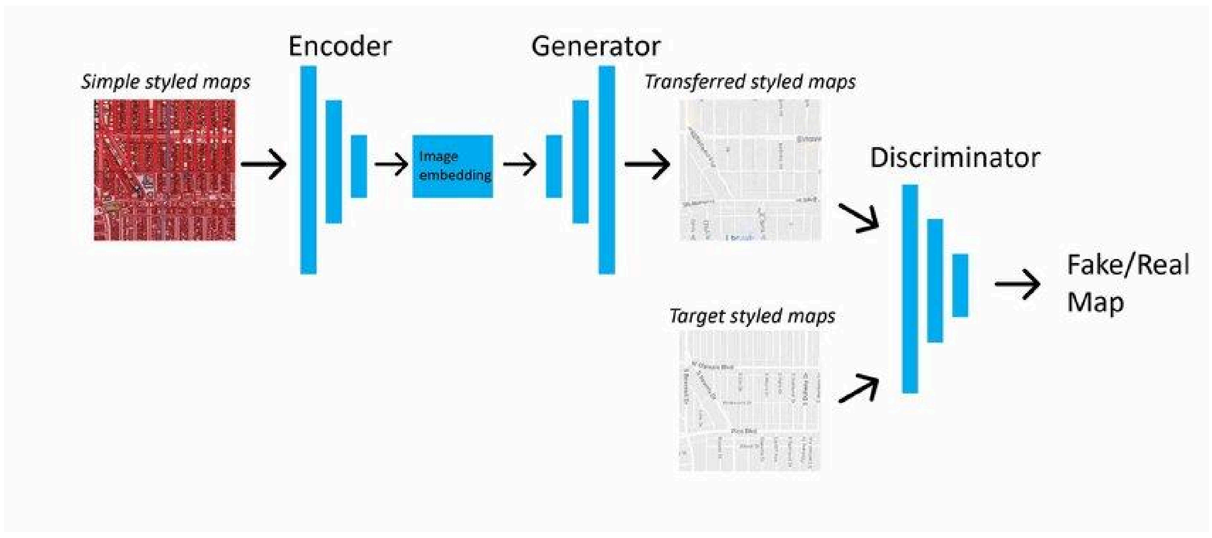
1. **Image-to-Image Translation**
- **Semantic Segmentation**: Converting labels into images.
- **Style Transfer**: Applying artistic styles to images while retaining content.

2. **Supervised Learning**
- **Data Augmentation**: Generating additional labelled data for training robust models.
- **Domain Adaptation**: Adapting models to new domains by generating domain-specific data.

3. **Image Generation with Constraints**
- **Fashion Design**: Creating specific types of clothing based on desired attributes.
- **Medical Imaging**: Generating images conditioned on patient data for improved diagnosis.

4.4 Pix2Pix

Pix2Pix is a type of conditional GAN designed specifically for image-to-image translation tasks, where an input image is translated to an output image in a supervised learning manner.



4.4.1 Architecture of Pix2Pix

Generator Model: The Generator in Pix2Pix uses an encoder-decoder architecture with U-Net connections.

1. **Input Layer**
   - **Input Image**: The image to be translated.
2. **Encoder**
   - **Convolutional Layers**: Series of convolutional layers that downsample the input image to a low-dimensional representation.
3. **Decoder**
   - **Deconvolutional Layers**: Series of fractional-strided convolutional layers that upsample the representation back to the original image size.
   - **U-Net Connections**: Skip connections between corresponding layers of the encoder and decoder to preserve spatial information.

Generator Loss The loss function combines a traditional GAN loss with an L1 loss to enforce similarity between the generated and target images:

$$L(G) = \mathbb{E}_{z\sim p_z(z),y\sim p(y)}[\log(1 - D(G(x|y),y))] + \lambda\mathbb{E}_{x,y}[|y - G(x)|_1]$$

Discriminator Model: The Discriminator in Pix2Pix is a PatchGAN, which classifies each patch in an image as real or fake.

1. **Input Layer**
   - **Concatenated Image**: The input and target images concatenated together.

2. **Convolutional Layers**
   - **Standard Convolutional Layers**: Applies convolutions followed by activation functions and batch normalization.

Discriminator Loss: The Discriminator's loss function is the same as in a standard cGAN but applied to image patches

$$L(D) = -\mathbb{E}_{x\sim p_{data}(x),y\sim p(y)}[\log D(x,y)] - \mathbb{E}_{z\sim p_z(z),y\sim p(y)}[\log(1 - D(G(x|y),y))]$$

4.4.2 Applications of Pix2Pix

1. **Image Synthesis**
   - **Photo Enhancement**: Enhancing images by translating low-quality photos to high-quality ones.
   - **Sketch to Image**: Generating realistic images from sketches or outlines.

2. **Semantic Segmentation**
   - **Label to Image**: Translating segmentation labels into realistic images for various applications such as autonomous driving.

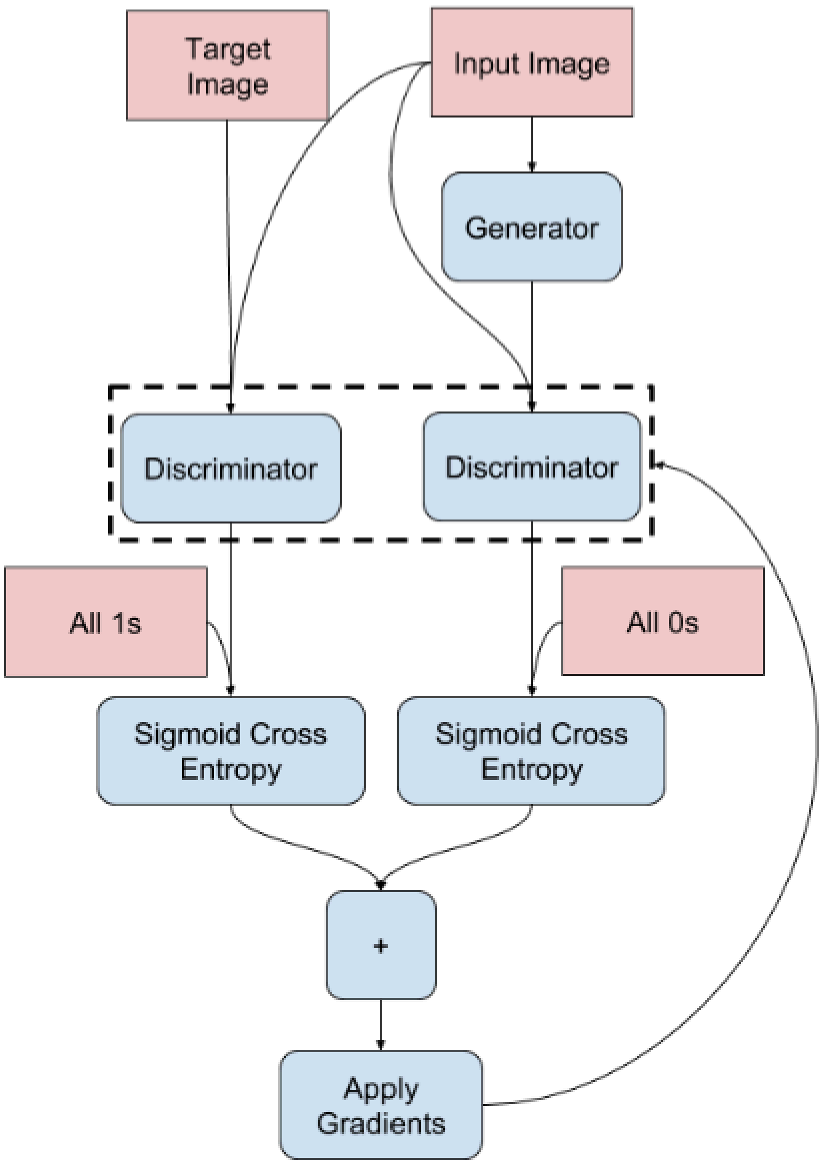
3. **Style Transfer**
   - **Image Editing**: Applying artistic styles to images while maintaining the content.

4. **Medical Imaging**
   - **Cross-Modality Image Translation**: Translating images between different modalities, such as MRI to CT scans, to provide comprehensive views for diagnosis.

4.5 PatchGAN

PatchGAN is a type of GAN architecture that classifies whether each N×N patch in an image is real or fake, focusing on local patches rather than the entire image. This approach excels in tasks where local texture and structure are critical.

4.5.1 Architecture of PatchGAN



Generator Model

The generator in a PatchGAN is responsible for creating realistic images from random noise or conditioned input data:

1. **Input Layer**
    - A noise vector zzz or conditioned input, often sampled from a uniform or normal distribution.

2. **Dense Layer**

    - A fully connected layer transforms the noise vector into a higher-dimensional space, reshaping it into a 4D tensor with dimensions [batch_size, depth, height, width].

3. **Fractional-Strided Convolutional Layers (Deconvolutions)**
    - These layers progressively increase the spatial dimensions (height and width) of the activation maps while reducing the depth, generating the final image. Each layer is usually followed by batch normalization and an activation function (ReLU for intermediate layers, Tanh for the output layer).

4. **Output Layer**
    - Produces an image with the same dimensions as the real images in the dataset.

Generator Loss

The generator's loss function (L(G)) aims to fool the discriminator by generating images that are classified as real:

$$L(G) = -E_{z \sim p_z(z)}[\log D(G(z))]$$

Discriminator Model

The PatchGAN discriminator evaluates local patches within an image to determine their authenticity:

1. **Input Layer**
    - An image (real or generated), divided into N×N patches.

2. **Convolutional Layers**
    - Each layer applies convolutions followed by non-linear activations and batch normalization (except the first layer) to improve training stability.
    - The layers progressively reduce the spatial dimensions of the patches while increasing their depth, capturing fine details.

3. **Output Layer**
    - Produces a matrix of probabilities, each representing the likelihood that a corresponding patch in the input image is real.

- This matrix matches the number of patches in the input image, providing a real/fake probability for each patch.

Discriminator Loss

The discriminator's loss function (L(D)) maximizes the probability of correctly classifying real and fake patches:

$$L(D) = -E_{x\sim p_{\text{data}}(x)}[\log D(x)] - E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

4.5.2 Applications of PatchGAN

1. **Image-to-Image Translation**
   - PatchGANs are highly effective for converting images from one domain to another (e.g., sketches to realistic photos, day to night scenes). Evaluating local patches helps preserve texture and fine details.

2. **Texture Synthesis**
   - PatchGANs excel in generating high-quality textures by ensuring each local patch is realistic, making them suitable for tasks requiring detailed and consistent textures across an image.

3. **Super-Resolution and Image Enhancement**
   - PatchGANs enhance the resolution and quality of images by focusing on local details. This is useful for applications like satellite imagery, security footage, and medical imaging where high detail fidelity is crucial.

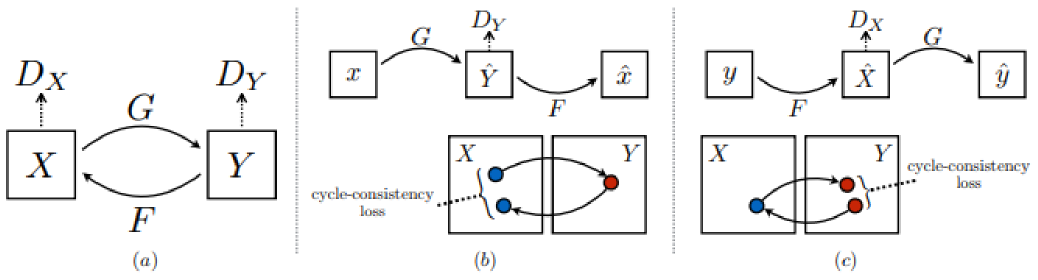
4. **Semantic Segmentation**
   - In semantic segmentation, PatchGANs help ensure that each segment of the image is realistic and consistent with the surrounding context, leading to more accurate and visually appealing segmentation results.

4.6 CycleGANs

CycleGAN is a model that aims to solve the image-to-image translation problem. The goal of the image-to-image translation problem is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, obtaining paired examples isn't always feasible. CycleGAN tries to learn this mapping without requiring paired input-output images, using cycle-consistent adversarial networks.

4.6.1 Architecture of CycleGANs

Like all the adversarial network CycleGAN also has two parts Generator and Discriminator, the job of generator is to produce the samples from the desired distribution and the job of discriminator is to figure out the sample is from actual distribution (real) or from the one that are generated by generator (fake).



The CycleGAN architecture is different from other GANs in a way that it contains 2 mapping functions (G, F) that act as generators and their corresponding Discriminators (Dx, Dy).

The generator mapping functions are as follows:

- **G** : X → Y
- **F** : Y → X

The discriminator corresponding to these are:

- **Dx** : distinguish G(X) (Generated Output) from Y (real Output)
- **Dy** : distinguish F(Y) (Generated Inverse Output) from X (Input distribution)

where X is the input image distribution and Y is the desired output distribution.

Generator Architecture

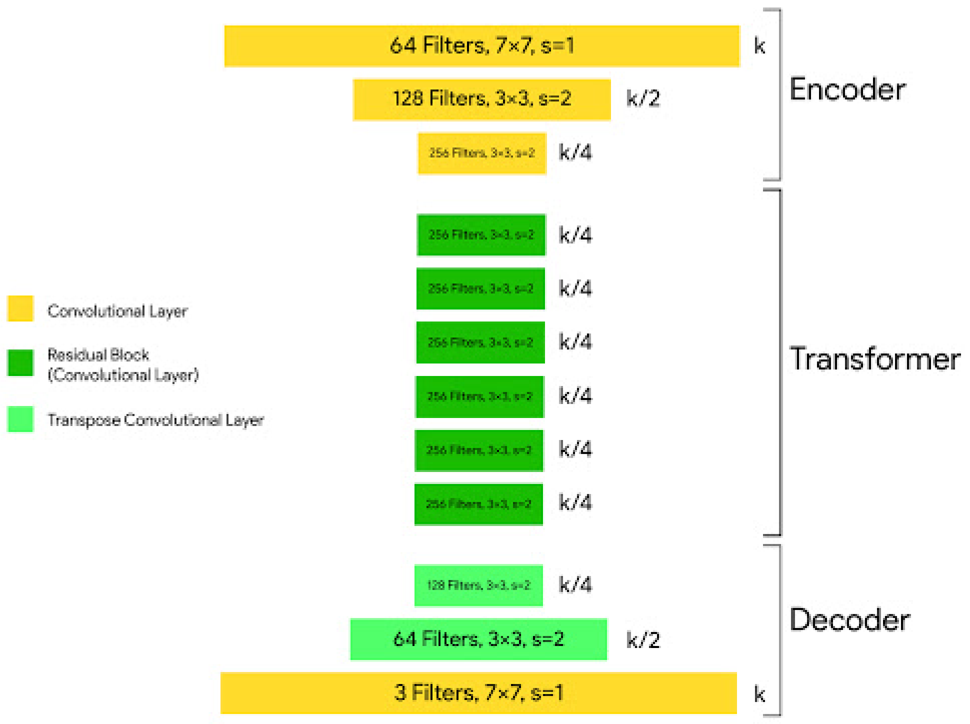
Each ResNet CycleGAN generator has three sections:

1. **Encoder:**
   The input image is passed into the encoder. The encoder extracts features from the input image by using Convolutions and compresses the representation of image but

increases the number of channels. The encoder consists of 3 convolutions that reduce the representation by 1/4 th of actual image size.

2. **Transformer:**
The output of the encoder after activation function is applied is then passed into the transformer. The transformer contains 6 or 9 residual blocks based on the size of input.

3. **Decoder:**
The output of the transformer is then passed into the decoder which uses 2 -deconvolution block of fraction strides to increase the size of representation to original size.



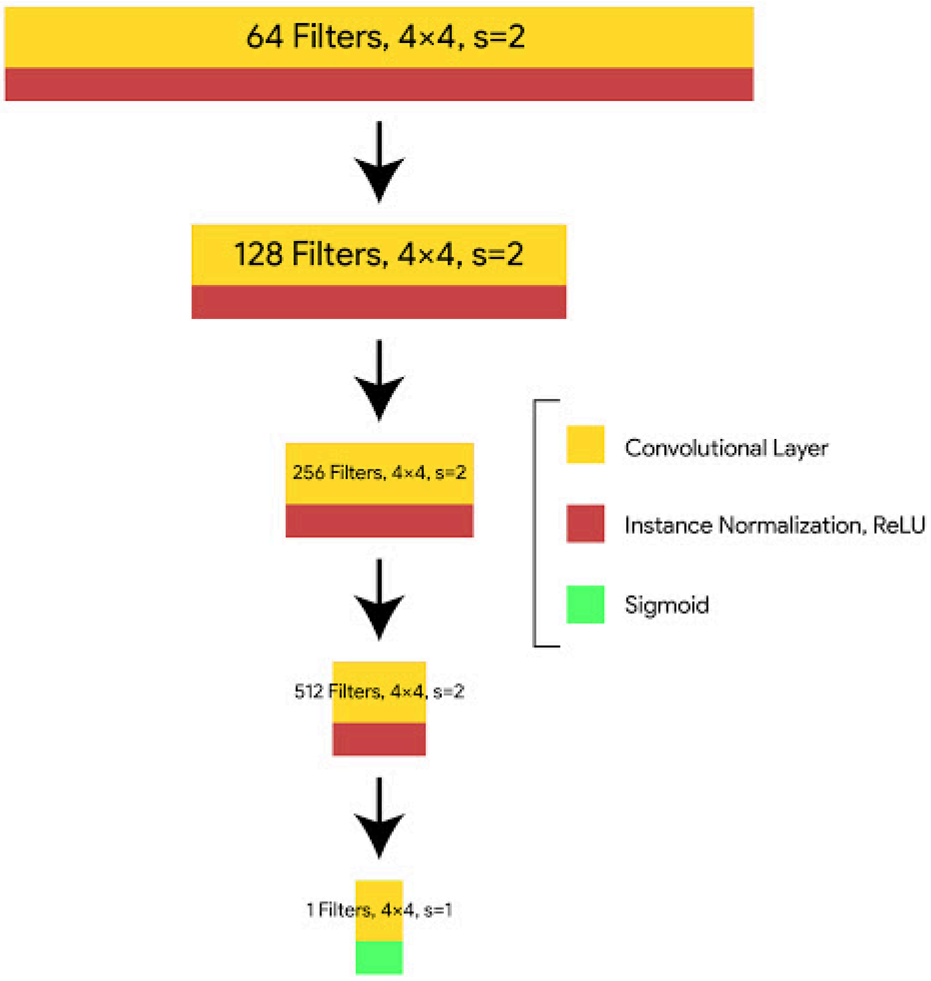
The architecture of generator is:
c7s1-64, d128, d256, R256, R256, R256, R256, R256, R256, u128, u64, c7s1-3

where,

- **c7s1-k** denotes a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1.
- **dk** denotes a 3 × 3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2.
- **Rk** denotes a residual block that contains two 3 × 3 convolution layers with the same number of filters on both layers.
- **uk** denotes a 3 × 3 fractional-strides-Convolution-InstanceNorm-ReLU layer with k filters and stride 1/2 (i.e deconvolution operation).

Discriminator Architecture

In discriminator the authors use PatchGAN discriminator. The difference between a PatchGAN and regular GAN discriminator is that rather the regular GAN maps from a 256×256 image to a single scalar output, which signifies “real” or “fake”, whereas the PatchGAN maps from 256×256 to an NxN (here 70×70) array of outputs X, where each Xij signifies whether the patch ij in the image is real or fake.



The architecture of discriminator is : C64-C128-C256-C512

where,

- Ck is a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2.
- We don’t apply InstanceNorm on the first layer (C64).
- After the last layer, we apply a convolution operation to produce a 1×1 output.

Cost Functions

- **Adversarial Loss:** We apply adversarial loss to both our mappings of generators and discriminators. This adversary loss is written as :

$$Loss_{advers}\left(G, D_y, X, Y\right) = \frac{1}{m}\sum\left(1 - D_y\left(G\left(x\right)\right)\right)^2$$
$$Loss_{advers}\left(F, D_x, Y, X\right) = \frac{1}{m}\sum\left(1 - D_x\left(F\left(y\right)\right)\right)^2$$

- **Cycle Consistency Loss:** Given a random set of images, an adversarial network can map the set of input images to random permutation of images in the output domain which may induce the output distribution similar to target distribution. Thus adversarial mapping cannot guarantee the input xi to yi . For this to happen the author proposed that the process should be cycle-consistent. This loss function is used in Cycle GAN to measure the error rate of inverse mapping G(x) → F(G(x)). The behaviour induced by this loss function is closely matching the real input (x) and F(G(x)).

$$Loss_{cyc}\left(G, F, X, Y\right) = \frac{1}{m}\left[\left(F\left(G\left(x_i\right)\right) - x_i\right) + \left(G\left(F\left(y_i\right)\right) - y_i\right)\right]$$

- **Identity Loss:** Identity loss says that, if you feed image Y to generator G, it should yield the real image Y or something close to image Y.

$$\begin{aligned}\mathcal{L}_{identity}(G, F) = {} & \mathbb{E}_{y \sim p_{data}(y)}[\|G(y) - y\|_1] \\ & + \mathbb{E}_{x \sim p_{data}(x)}[\|F(x) - x\|_1]\end{aligned}$$

The Cost function we used is the sum of adversarial loss and cyclic consistent loss and identity loss:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = {} & \mathcal{L}_{GAN}(G, D_Y, X, Y) \\ & + \mathcal{L}_{GAN}(F, D_X, Y, X) \\ & + \lambda\mathcal{L}_{cyc}(G, F) \\ & + 0.5\lambda\mathcal{L}_{identity}(G, F)\end{aligned}$$

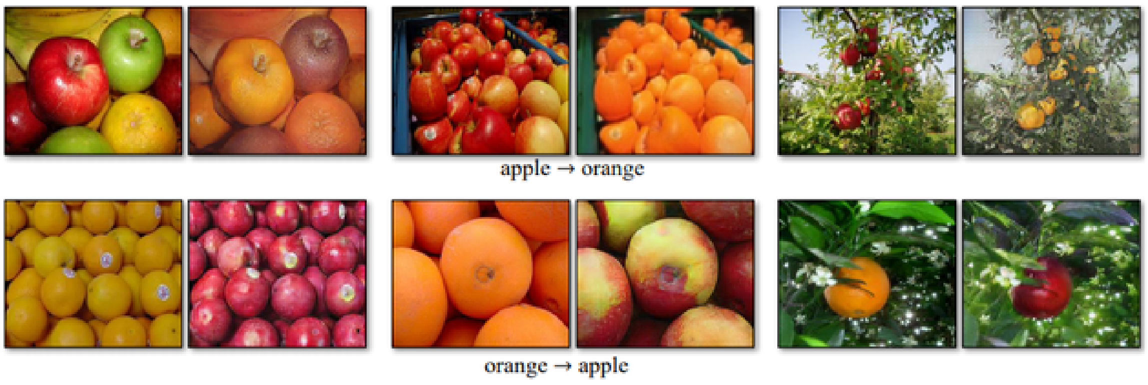
with our aim being:

$$\underset{G,F}{argmin}\,\underset{D_x, D_y}{max}\, L\left(G, F, D_x, D_y\right)$$

4.6.2 Applications of CycleGANs

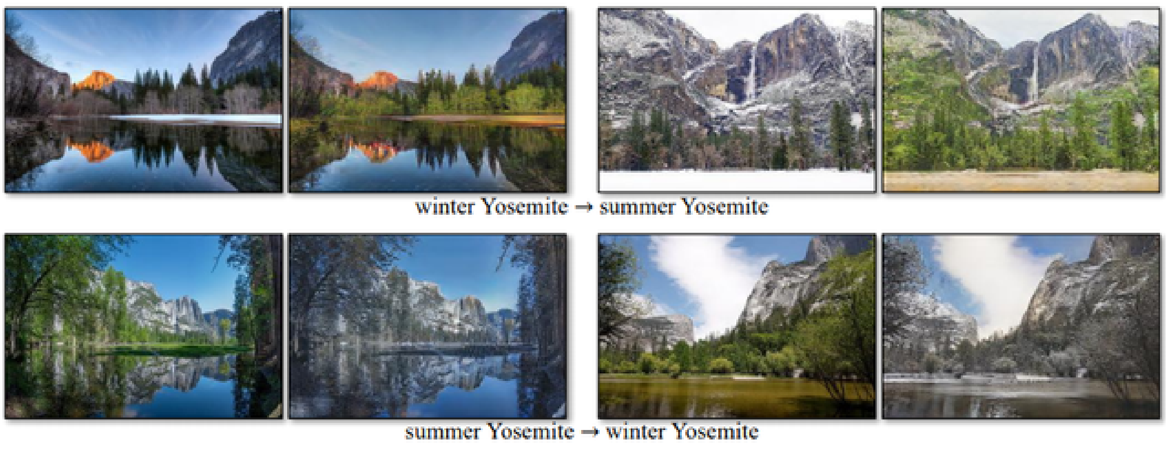
1. **Object Transformation:**

   CycleGAN can transform objects from one ImageNet class to another such as: Zebras to Horses and vice-versa, Apples to Oranges and vice versa etc.



2. **Season Transfer:**

   CycleGAN can also transfer images from Winter Season to Summer season and vice-versa.


winter Yosemite → summer Yosemite
summer Yosemite → winter Yosemite

3. **Photo enhancement :**

   CycleGAN can also be used for photo enhancement. For this the model takes images from two categories which are captured from smartphone camera (usually have deep Depth of Field due to low aperture) to DSLR (which have lower depth of Field).