WiDS 2k24 Report

Next Gen Visual Models

Pranathi Sreeja Meesala

Mentors: Himanshu Raj, Shourya Sethia

February 2025

1 Week 0: Introduction to ML, RL, and DL

Machine Learning (ML)

Machine Learning is a subset of Artificial Intelligence (AI) where models learn from data to make predictions or decisions.

Applications:

- Spam detection in emails
- Recommendation systems (Netflix, Amazon)
- Medical diagnosis
- Fraud detection in banking

Supervised Learning

In supervised learning, the model is trained on labeled data, where each input has a corresponding output.

Example Applications:

- Image classification (cats vs. dogs)
- Sentiment analysis in text
- Speech recognition (converting speech to text)

Unsupervised Learning

In unsupervised learning, the model is trained on unlabeled data and tries to find hidden patterns.

Example Applications:

- Customer segmentation for marketing
- Anomaly detection in network security
- Clustering genes in bioinformatics

Reinforcement Learning (RL)

Reinforcement Learning (RL) involves training agents to make a sequence of decisions to maximize rewards in an environment.

Example Applications:

- Game playing (AlphaGo, OpenAI Gym)
- Robot control
- Autonomous driving

Deep Learning (DL)

Deep Learning (DL) is a subset of ML that uses deep neural networks for feature extraction and learning.

Example Applications:

- Facial recognition
- Language translation (Google Translate)
- Self-driving cars (Tesla Autopilot)

Python Revision Assignment

Before diving into ML, we completed a Python revision assignment in Week 0. This included fundamental programming concepts such as data structures, functions, and basic algorithmic problem-solving to ensure readiness for the upcoming topics.

2 Week 1

2.1 Perceptron, MLPs, and Backpropagation

In Week 1, we explored foundational deep learning concepts:

- **Perceptron:** The simplest type of artificial neural network, used for binary classification.
- **Multi-Layer Perceptrons (MLPs):** Neural networks with multiple layers to learn complex patterns.
- **Backpropagation using Gradient Descent:** A method to update weights in neural networks to minimize error.

$$w = w - \eta \frac{\partial L}{\partial w} \tag{1}$$

where w represents weights, η is the learning rate, and $\frac{\partial L}{\partial w}$ is the gradient of the loss function.

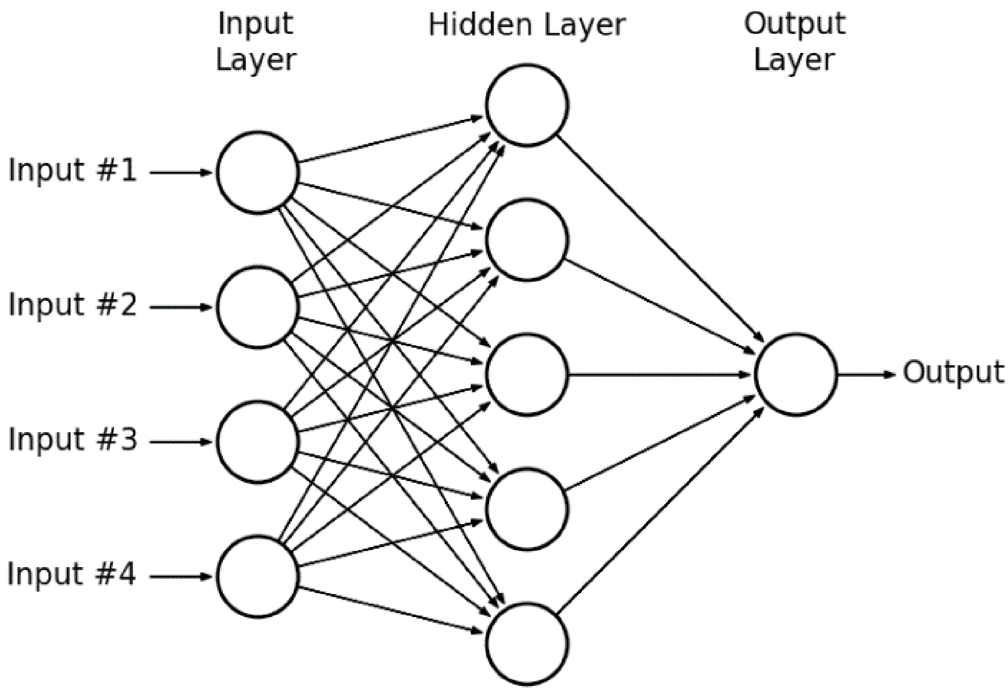

Figure 1: Basic Perceptron Model

2.2 Why MLPs are Not Ideal for Image Processing

While MLPs work well for structured data, they are not efficient for image processing due to:

- **High Number of Parameters:** Every pixel in an image is treated as an independent input, leading to a massive number of weights and making MLPs computationally expensive.
- **Loss of Spatial Information:** MLPs flatten the input, losing the spatial relationships between pixels, which are crucial for image analysis.
- **Overfitting and Poor Generalization:** Due to the large number of parameters, MLPs tend to overfit small datasets and do not generalize well to new images.

2.3 Introduction to CNNs

- **What are CNNs?** Deep learning models designed for processing grid-like data such as images.
- **Convolution Operation:** The core component that applies filters to input data to extract features.

- **Padding and Striding:** Techniques used to manage spatial dimensions and computational efficiency in CNNs.
- **Pooling Layers:** Downsampling layers that reduce dimensionality and retain essential features.
  - **Max Pooling:** Retains only the maximum value in each window, improving translational invariance and reducing computation.
  - **Average Pooling:** Computes the average value of each window, preserving more information but less commonly used than max pooling.
- **Why CNNs are Preferred Over MLPs for Images:**
  - **Preservation of Spatial Hierarchy:** CNNs maintain spatial relationships between pixels using convolution layers.
  - **Parameter Sharing:** The same filter is used across the image, reducing the number of parameters significantly.
  - **Automatic Feature Extraction:** CNNs learn hierarchical features (edges, textures, objects) without manual intervention.

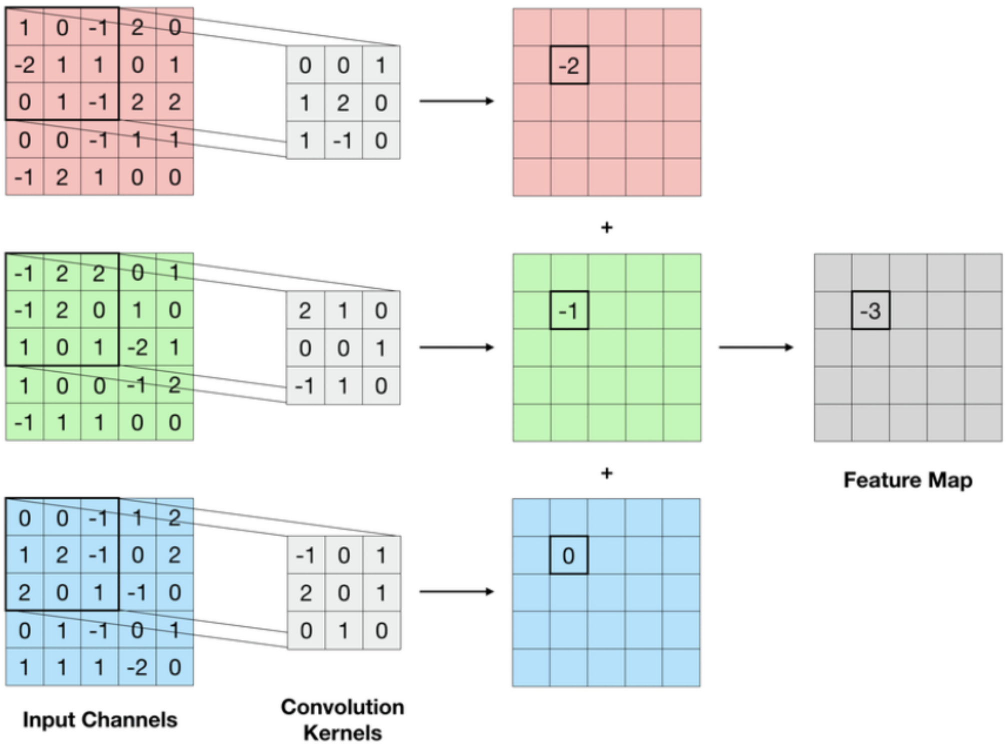

Figure 2: Convolution operation

2.4 Assignment: Implementing MLP from Scratch

As part of our learning, we implemented a Multi-Layer Perceptron (MLP) from scratch using Python and NumPy to understand how weights update through backpropagation.

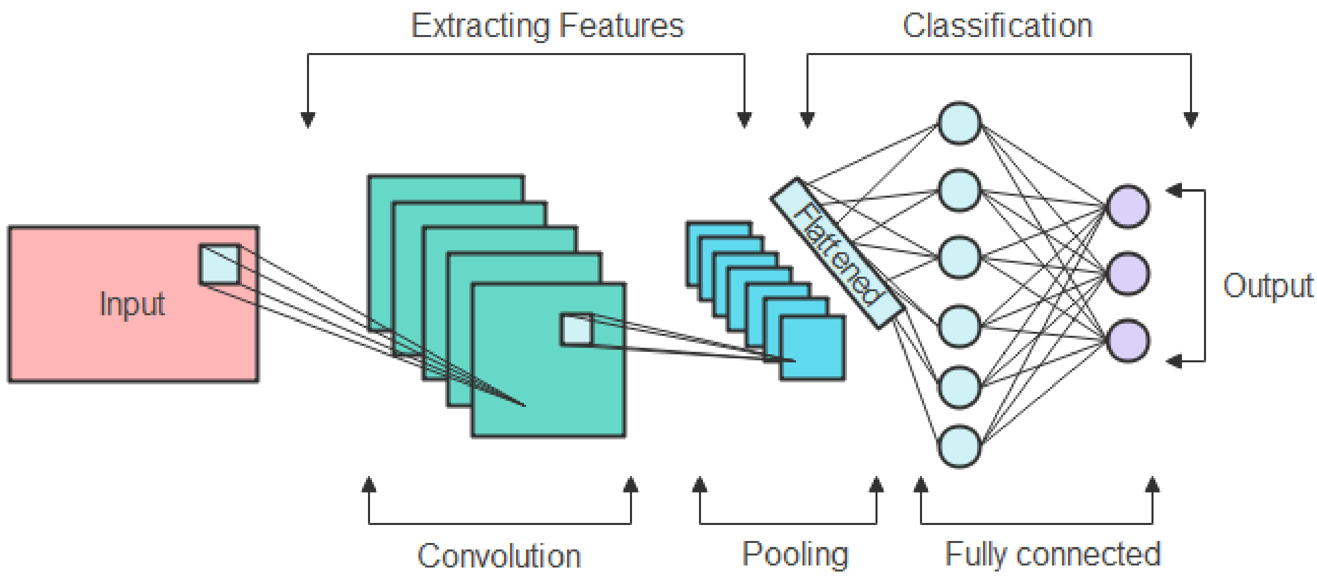

Figure 3: Convolutional Neural Network (CNN) Architecture

3 Week 2: Data Augmentation, Transfer Learning

In Week 2, we explored fundamental techniques to enhance deep learning models, including activation functions, data augmentation, transfer learning, and well-known architectures.

- **ReLU vs. Leaky ReLU:** The Rectified Linear Unit (ReLU) is a widely used activation function in neural networks, defined as $f(x) = \max(0, x)$. While effective, it suffers from the "dying ReLU" problem, where neurons can become inactive if they receive negative inputs. Leaky ReLU addresses this by allowing a small negative slope for negative inputs, defined as $f(x) = \max(\alpha x, x)$, where α is a small constant (e.g., 0.01). This prevents neurons from becoming permanently inactive.

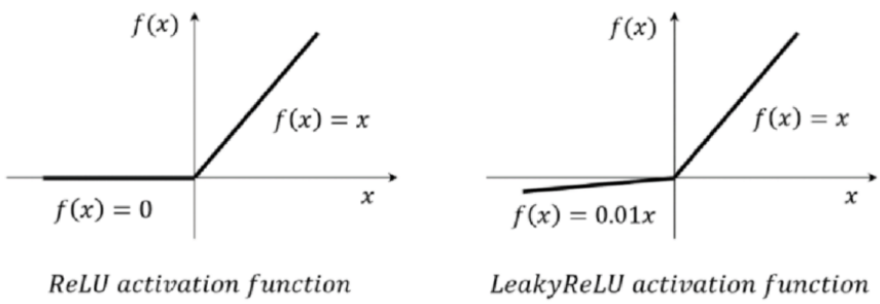

Figure 4: RELU vs Leaky RELU

- **Data Augmentation:** Data augmentation enhances model generalization by artificially expanding the training dataset. Techniques include:
  - *Geometric Transformations:* Rotation, scaling, flipping, cropping.
  - *Color Augmentation:* Brightness, contrast, saturation adjustments.
  - *Noise Injection:* Adding Gaussian noise to improve robustness.

  These transformations help models generalize better by learning from diverse variations of the input data.

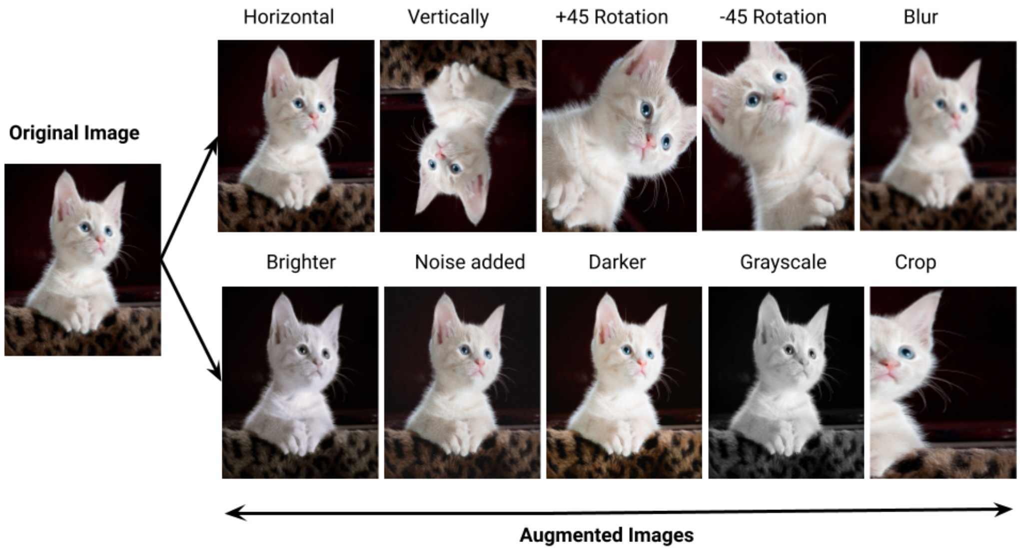

Figure 5: Data Augmentation Example

- **Transfer Learning:** Transfer learning involves using pre-trained models trained on large datasets (such as ImageNet) and fine-tuning them for specific tasks. Instead of training from scratch, a model's early layers, which capture generic features, are reused, while later layers are fine-tuned for the new dataset. This significantly reduces training time and improves performance, especially for small datasets.
- **Famous Architectures:** Several deep learning architectures have revolutionized computer vision:
  - **Inception Networks:** Use multiple convolutional filters of different sizes in parallel (Inception modules) to capture multi-scale features while reducing computational cost with 1×1 convolutions.
  - **ResNets (Residual Networks):** Introduce residual connections (skip connections) to enable training of very deep networks without vanishing gradients, allowing gradients to flow directly through layers.
  - **VGG (Visual Geometry Group) Networks:** Consist of deep architectures with small 3×3 convolutional filters, improving feature extraction but requiring higher computational resources.

  These architectures have significantly improved performance in image classification and object detection tasks.

Assignments

We completed four assignments:

- **CNN on CIFAR-100:** Implemented a CNN model for image classification on the CIFAR-100 dataset.
- **Transfer Learning with VGG-19:** Tested accuracy on CIFAR-10 using pre-trained weights from ImageNet, then fine-tuned the model for better performance.
- **Binary Classification:** Built a CNN to classify pneumonia vs. normal chest X-ray images.
- **CNN on Fashion-MNIST:** Compared CNN performance with MLP models on the Fashion-MNIST dataset.

4 Week 3: Autoencoders

4.1 Concepts

Autoencoders are neural networks designed to learn efficient representations of data by encoding inputs into a compressed latent space and then reconstructing them. They are widely used for dimensionality reduction, noise removal, and data generation. The fundamental structure of an autoencoder consists of three main components:

- **Encoder:** Compresses the input data into a lower-dimensional representation.
- **Bottleneck Layer:** The latent space where the compressed information is stored.
- **Decoder:** Reconstructs the original data from the compressed representation.

Several types of autoencoders exist, each tailored to specific tasks:

- **Vanilla Autoencoders:** The basic form of autoencoders used for compression and reconstruction.
- **Denoising Autoencoders:** Train on noisy inputs to reconstruct clean versions of data.
- **Variational Autoencoders (VAEs):** Generate new data by learning a probabilistic latent space representation.
- **Sparse Autoencoders:** Introduce sparsity constraints to enhance feature learning.

4.2 Assignment

The assignment focuses on Variational Autoencoders (VAEs), which follow an encoder-decoder architecture. The encoder progressively reduces the dimensionality of the input, while the decoder expands it back to its original form. The tasks include:

1. Implementing a standard VAE with a simple encoder-decoder structure.
2. Modifying the encoder to include a decoder after downsampling.
3. Modifying the decoder to include an encoder before upsampling.
4. Replacing both the encoder and decoder with encoder-decoder architectures.

The performance of these architectures will be compared based on accuracy and output quality to determine the most effective design.

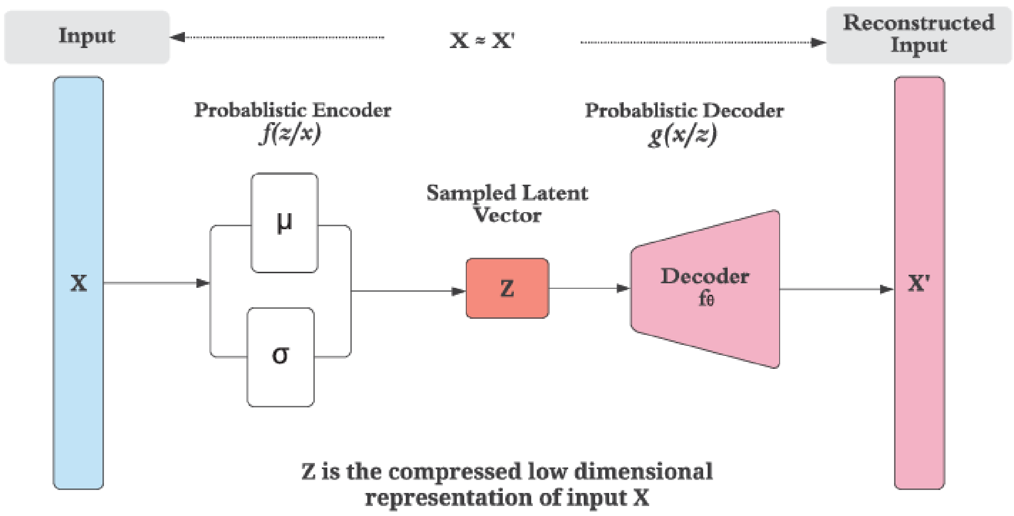

Figure 6: Variational Autoencoder Architecture

5 Week 4: Generative Models and GANs

5.1 Concepts

Week 4 introduced Generative Models and Generative Adversarial Networks (GANs), which are widely used for data synthesis, image generation, and style transfer. Topics covered include:

- **Generative Models:** Models that learn the underlying distribution of data to generate new samples.
- **GANs:** A framework consisting of two neural networks—a generator and a discriminator—engaged in a competitive process. The generator learns to create data that resemble real samples, while the discriminator tries to distinguish between real and generated data.
- **Role of Generator and Discriminator:**
  - **Generator:** Takes in random noise (latent space) and transforms it into realistic-looking data samples. It aims to generate outputs that are indistinguishable from real data.
  - **Discriminator:** Acts as a classifier that receives both real data and generated data, learning to differentiate between them. It provides feedback to the generator, helping it improve its outputs over time.
  - Over multiple training iterations, the generator improves at fooling the discriminator, while the discriminator becomes better at identifying fakes. This adversarial process results in highly realistic generated samples.
- **Loss Functions:**
  - Binary cross-entropy loss for the discriminator to classify real and generated samples correctly.
  - Adversarial loss for the generator to fool the discriminator by producing data that appear real.
- **Applications of GANs:** Image synthesis, super-resolution, data augmentation, deepfake generation, and domain adaptation.

5.2 Assignment: Implementing a GAN

In this week's assignment, we implemented a basic Generative Adversarial Network (GAN) to generate images of cats from a dataset of cats. The assignment involved:

- Building a simple generator and discriminator using fully connected layers.
- Training the GAN using alternating updates to minimize loss functions.
- Visualizing generated samples at different epochs to observe improvements.

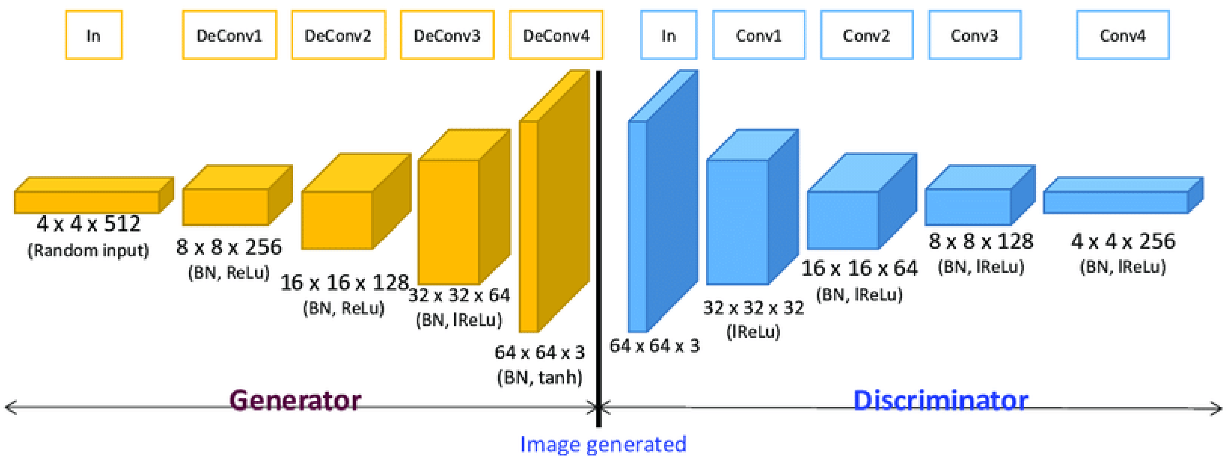

Figure 7: GAN architecture

6 Week 5: Diffusion Models

6.1 Concepts

In Week 5, we learned about diffusion models, a class of generative models that iteratively refine noisy data to generate realistic images. These models have gained popularity due to their ability to produce high-quality samples, particularly in image generation tasks. The key topics covered include:

- **Introduction to Diffusion Models:** Diffusion models generate data by gradually refining noisy inputs through a series of denoising steps. They are inspired by thermodynamic diffusion processes, where particles transition from an ordered state to randomness over time.
- **Forward and Reverse Diffusion Processes:**
  - **Forward Process:** Also known as the noising process, it progressively adds Gaussian noise to the input data over multiple steps, transforming it into pure noise.
  - **Reverse Process:** The model learns to invert the noising process by denoising step by step, reconstructing realistic data from the noisy distribution.
- **The Role of Noise Schedules and Denoising Steps:**
  - **Noise Schedule:** Defines how much noise is added at each step in the forward process. A well-designed schedule ensures effective training and stable generation.
  - **Denoising Steps:** The reverse process is performed iteratively, gradually removing noise at each step until a high-quality sample emerges. This stepwise refinement allows for fine-grained control over the generated output.

6.2 Assignment: Implementing a Diffusion Model

The assignment required implementing a diffusion model, varying hyperparameters such as learning rate, noise schedule (beta), and denoising steps. Tasks included:

- Producing results with two sets of hyperparameters and analyzing the loss curves.
- Generating one sample from each class in the CIFAR-10 dataset.
- Experimenting with denoising partially noised samples at different noise levels (50, 10, and 5 iterations) and analyzing the results.

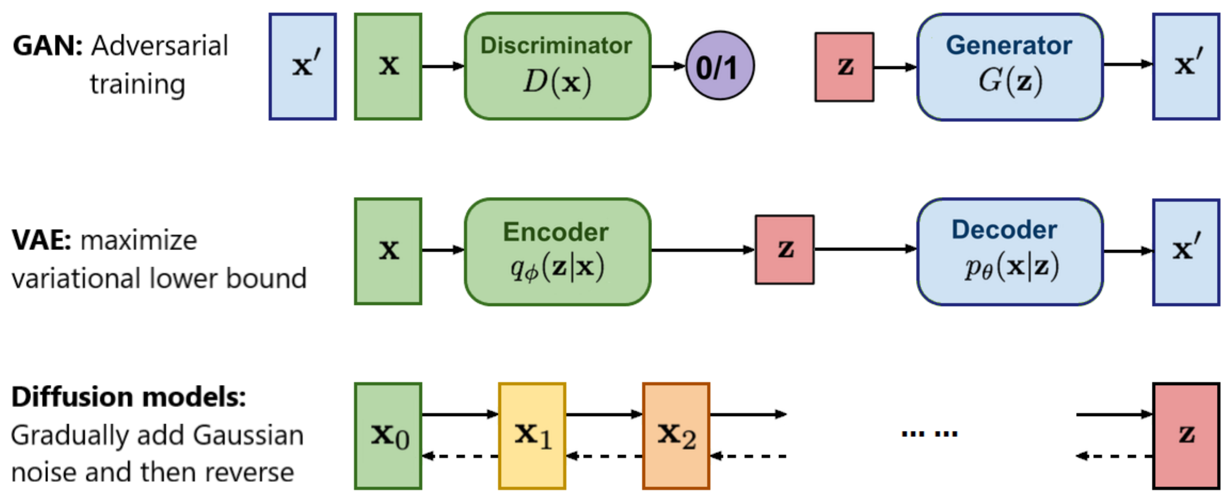

Figure 8: Diffusion model