**Computer Vision and Generative AI**

# Image Generation and GANs

Generative Adversarial Networks (GANs) have significantly advanced the field of computer vision and were the first to bring Generative AI to the forefront of research and public opinion. These models excel in creating high-quality visuals and operate at impressive speeds.

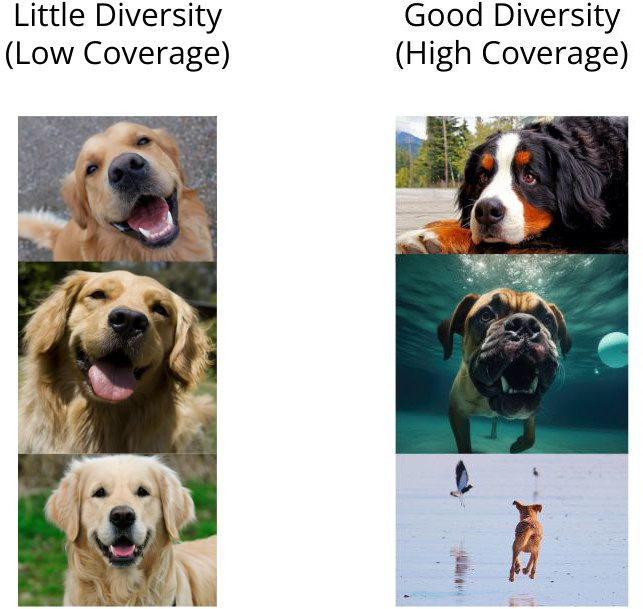
* There are different image generation categories:
* Unconditional Generation: Creating images without specific input or guidance.
* Text-to-Image: Generating images from textual descriptions.
* Image-to-Image Conversion: Transforming basic inputs like sketches into detailed images.
* Inpainting and Part Substitution: Editing images by filling in or replacing parts.

GANs are primarily unconditional image creators, although they can be adapted for conditional inputs. However, another class of Generative AI models called Diffusion Models is more commonly used for conditional generation. For this reason, in this lesson we will focus on unconditional generation.

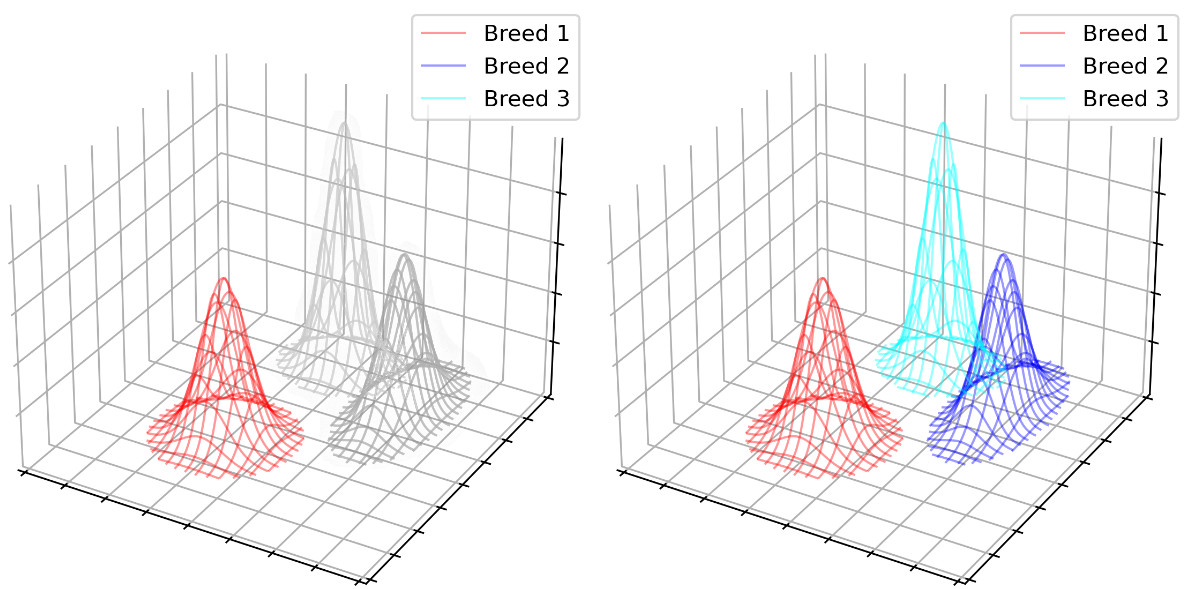
## Evaluation Gen AI in CV: Quality, Coverage, and Speed

In the realm of image generation algorithms, the concept of **'coverage'** plays a crucial role in characterizing the diversity of the output. Coverage essentially refers to how well an algorithm can produce a variety of images.

Consider two algorithms designed for generating dog images. The first produces images of the same dog breed in a similar pose, indicating low diversity or low coverage. In contrast, the second algorithm generates a wide variety of dog breeds, poses, and environments, showcasing good diversity or high coverage.



Images can be conceptualized as coming from a sparse multi-modal distribution, with density peaks representing different subjects, image types or other differentiating aspects of the images. An algorithm with low coverage is limited to sampling from a specific peak, resulting in similar images. For instance, it might only generate images of one dog breed. On the other hand, an algorithm with good coverage can sample from multiple peaks, creating diverse images in different scenarios, like various dog breeds in different situations.



**Quality** in image generation algorithms refers to how closely the generated images resemble real ones. This concept is often quantified by comparing the distribution of generated images against a real dataset. One popular metric for this is the Fréchet Inception Distance (FID), which uses a pre-trained neural network like Inception V3 to create embeddings from images and measures the distance between these distributions. A lower FID indicates higher quality, meaning the generated images are more realistic.

Finally, the **speed** at which an algorithm generates images is a practical consideration. It's about finding a balance between the time taken to produce an image and the desired quality and diversity.

## Fundamentals od GANs

Generative Adversarial Networks (GANs) represent a significant leap in the field of artificial intelligence, particularly in image generation. At its core, a GAN consists of two main components: the Generator and the Discriminator.



1. The Generator: Crafting Synthetic Images

The Generator's role is to create images. It starts with a random noise vector, often sampled from a high-dimensional distribution. This vector, known as latent z, is then passed through the Generator network, which uses strided convolutions to convert this latent representation into a synthetic image. For example, a latent vector of 100 elements might be transformed into a 64x64 pixel image, which translates to 4096 numbers.

1. The Discriminator: The Arbiter of Realism

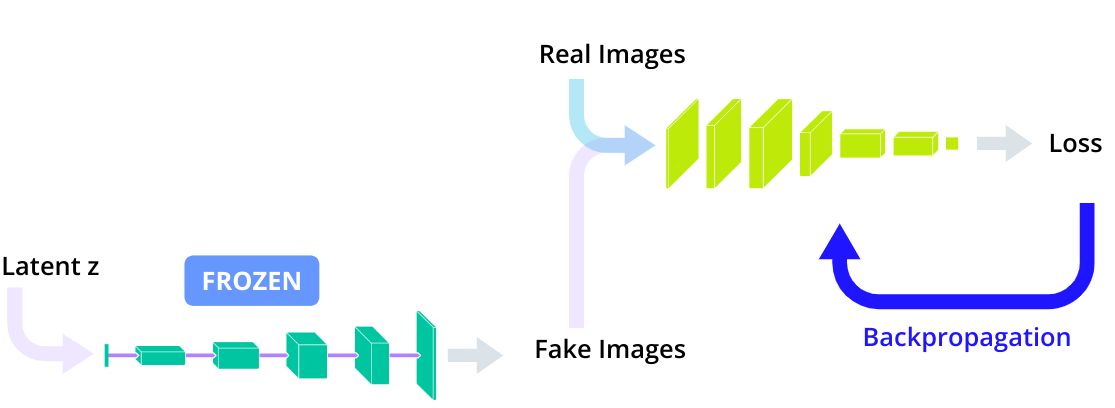
The Discriminator's job is to distinguish between real and generated (fake) images. It performs a binary classification to determine the authenticity of each image. In classical GANs, the Discriminator is typically a standard Convolutional Neural Network (CNN) used for image classification.

1. Training the GAN: A Dance Between Generator and Discriminator

Training a GAN involves an iterative process where the Generator and Discriminator continuously improve through competition. Initially, the Generator creates images, and the Discriminator learns to distinguish them from real ones. As the Generator improves, it becomes better at fooling the Discriminator. The training process is a cycle of alternating between training the Discriminator and the Generator, each time making them more adept at their tasks.

In summary, GANs harness the power of two neural networks in a unique setup, where one creates and the other critiques, leading to the generation of increasingly realistic images. This dynamic interplay between the Generator and Discriminator underpins the success of GANs in creating convincing and high-quality synthetic images.

### Discrimitator training



The Discriminator's task in a GAN is to distinguish between real and generated images. During training, this component learns to identify nuances that differentiate authentic images from those created by the Generator.

A popular method for training the Discriminator is the 'split-batch' technique. This involves two main steps:

1. Step 1: Handling Real Images

* The Discriminator is fed real images and learns to identify them as authentic.
* This process involves a forward pass of real data through the Discriminator, generating a probability score for each image being real.
* The Binary Cross Entropy (BCE) loss is then calculated by comparing the Discriminator's predictions against the true labels (real images).

1. Step 2: Dealing with Fake Images

* Next, the Discriminator is presented with fake images produced by the Generator.
* These images undergo a similar process, with the Discriminator learning to label them as fake.
* The BCE loss is again used to compare the Discriminator's predictions against the true labels (fake images).

1. Updating the Discriminator

After processing both real and fake images, the Discriminator's weights are updated. This is done using the gradients accumulated from both sets of data, ensuring that the Discriminator improves its ability to differentiate real from fake images.

Code:

optimizerD = optim.Adam(D.parameters(), …)

criterion = BCELoss()

**for** data, \_ **in** dataloader:

*# ... omitted, moving data to GPU etc.*

*# Probability for data to be real,*

*# according to the Discriminator*

D\_pred = D(data).view(-1)

*# Ground truth labels: here the images are real*

*# so the labels are all 1*

labels = torch.full((size,), 1.0, device=device)

*# Compare Discriminator prediction vs ground truth*

loss\_on\_real\_data = criterion(D\_pred, labels)

*# Compute gradients*

loss\_on\_real\_data.backward()

*# Get latent vectors and generate*

*# fake images using the Generator*

latent\_vectors = torch.randn(

b\_size,

latent\_dimension,

1,

1

)

fake\_data = G(latent\_vectors)

*# Get predictions from the Discriminator*

*# on the fake data*

*# NOTE: remember to use .detach()!*

D\_pred = D(

fake\_data.detach()

).view(-1)

*# Since these are all fake images, the ground*

*# truth should be 0 for all labels. We refill*

*# the tensor of labels we already have instead*

*# of creating a new one*

labels.fill\_(0)

*# Compare Discriminator prediction vs ground truth*

loss\_on\_fake\_data = criterion(D\_pred, labels)

*# Add gradients computed on the loss on fake data*

loss\_on\_fake\_data.backward()

*# Finally update the Discriminator*

optimizerD.step()

### Generator Training and Inference

he Generator in a GAN starts with a random noise vector, known as latent z, and transforms it into a synthetic image. The goal of the Generator is to create images so convincing that they can fool the Discriminator into believing they are real. This is accomplished by trying to maximize the loss of the Discriminator on the fake data.

The training process of the Generator involves several key steps:

* Generating Fake Images: The Generator creates fake images from the latent z vector. We are actually going to reuse the fake images we have generated previously during the training of the Discriminator, but this is just an optimization.
* Discriminator's Evaluation: These fake images are then passed through the Discriminator, which is kept frozen during this phase. The Discriminator evaluates these images and assigns a probability score to each, indicating how likely they are to be real.
* Loss Calculation and Backpropagation: The **Generator then adjusts its parameters to maximize the loss derived from the Discriminator’s evaluation. This loss reflects how well the Generator is fooling the Discriminator.**

optimizerG = optim.Adam(G.parameters(), …)

**for** data, \_ **in** dataloader:

… *# Discriminator training*

G.zero\_grad()

*# Get a prediction from the Discriminator on the*

*# fake data we already generated during Discriminator*

*# training*

D\_pred = D(fake\_data).view(-1)

*# BCE trick: instead of maximizing BCE when*

*# y = 0, we minimize the BCE when y = 1. These*

*# are equivalent, but minimizing can be done with*

*# the normal Gradient Descent algorithm*

labels.fill\_(1)

loss\_on\_fake\_G = criterion(D\_pred, labels)

*# Compute gradients*

loss\_on\_fake\_G.backward()

*# Update Generator to maximally-fool the*

*# Discriminator*

optimizerG.step()

Once training is complete, the Discriminator is discarded, and the Generator is used for image generation. New images are created by feeding random latent vectors into the Generator, which then produces diverse and realistic images.

## GANs Are Tricky To Train

* Unstable Balance: Training GANs is delicate; if either the Generator (G) or Discriminator (D) becomes too proficient too quickly, the other lags, disrupting the training process.
* No Clear Convergence Indicator: Unlike traditional neural networks, GANs lack a clear metric like validation loss to signify convergence, making it hard to determine the optimal stopping point.
* Mode Collapse: A critical issue where the Generator discovers a specific image type that always fools the Discriminator, leading to a lack of diversity in generated images.

### Advanced Variants of GANs

To address these challenges, several GAN variants have been developed:

* [Wasserstein GAN (W-GAN)(opens in a new tab)](https://arxiv.org/abs/1701.07875): Introduces a Critic instead of a Discriminator, which assigns continuous scores to images, enhancing training dynamics and reducing mode collapse.
* [Progressive GANs(opens in a new tab)](https://arxiv.org/abs/1710.10196): These GANs begin by generating low-resolution images, progressively adding details. This approach aids in faster convergence and enables the creation of high-resolution images.
* Style GANs ([v1(opens in a new tab)](https://arxiv.org/abs/1812.04948), [v2(opens in a new tab)](https://arxiv.org/abs/1912.04958) and [v3(opens in a new tab)](https://arxiv.org/abs/2106.12423)): Incorporate a mapping network to convert the latent vector into a style vector, which is fed along with the latent into the Generator. This, combined with added random noise and a few other innovations, significantly enhances sample quality and robustness.
* Conditional GANs. Anotable extension of GANs is the development of conditional GANs (see for example [here(opens in a new tab)](https://arxiv.org/abs/1907.10786)). They allow for manipulation of specific attributes in the output images, such as changing the view angle, gender, or adding a smile.

# Transformer-Based Computer Vision Models

* Vision Transformers (ViTs) analyze images by dividing them into 16x16 patches and using an encoder for analysis, advancing ImageGPT's concepts.
* ViTs surpass conventional computer vision models in accuracy and efficiency, needing five times less training resources.
* ViTs are scalable and versatile for complex tasks, but require significant computational resources and extensive data.
* Employed in diverse fields, ViTs excel in medical imaging for tumor detection and in retail for product classification and inventory management.
* Vision Transformers process images by dissecting them into 16x16 patches, flattening them into embeddings, and using these as tokens in a Transformer encoder to capture complex patterns and bridge language and vision.

## Conditional Generation

* ViT Conditioned Generation, a multi-modal transformer, is utilized in diverse fields like self-driving cars and crime detection, integrating various data types for complete understanding.
* It converts images and text into vector embeddings, merging them for a unified representation of visual and textual data.
* The model's image encoder processes these merged embeddings, analyzing visual elements and textual context in a manner akin to film direction.
* ViT Conditioned Generation uses contrastive training to align text and image encoders with relevant pairs, ensuring precise text-image matching for effective visual-textual task performance.
* ViT models use cross-attention to process text and visual data efficiently, adjusting focus contextually to preserve each modality's uniqueness.
* They transform images into patch vectors and merge these with text embeddings for a unified multimodal representation, processed by an Image Encoder and Text Decoder.
* These transformers are versatile, aiding in image description generation, visual question answering, and delivering real-time data for autonomous vehicles, capitalizing on their adept handling of visual and textual data interplay.
* Multimodal Vision Transformers integrate visual and textual data by encoding image patches and text embeddings into a shared representation, which is then refined through contrastive learning and cross-attention, enabling the model to generate context-rich interpretations for complex tasks that combine visual and textual analysis.

## DALL-E

* DALL-E is adept at creating vibrant images from text prompts, used extensively by educators, marketers, artists, and designers for various visual applications.
* Its architecture includes a CLIP-enhanced encoder for text comprehension, a Prior model for converting text to image embeddings, and a decoder for image generation.
* DALL-E's CLIP component is vital in connecting text and image understanding, analyzing images in the context of text to effectively associate visuals with related concepts and words.
* DALL-E's architecture comprises an encoder that semantically processes textual prompts using CLIP, a prior that prepares CLIP image embeddings from these text embeddings, and a decoder that translates these embeddings into the final image.

## DINOV2

* DINOv2 employs self-supervised learning for visual feature encoding, excelling at in-depth estimation, image classification, and segmentation without task-specific tuning.
* It utilizes a student-teacher network model, incorporating patch concealment and exponential moving average updates, to improve object identification in images.
* DINOv2's training uses a 142 million image dataset and processes 1.2 billion images in its pipeline, enhancing feature extraction capabilities for uses like autonomous vehicle navigation.
* DALL-E's architecture comprises an encoder that semantically processes textual prompts using CLIP, a prior that prepares CLIP image embeddings from these text embeddings, and a decoder that translates these embeddings into the final image.

## SAM

* SAM excels in accurately segmenting and identifying elements within images, using encodings to manipulate specific segments, which is ideal for tasks like precise background removal in photo editing.
* It features a Promptable Segmentation Task, encoders for image and prompt embeddings, and a mask decoder for segmentation, accommodating diverse inputs such as positions or automatic segmentations.
* SAM undergoes three training stages: Assisted-Manual, Semi-Automatic, and Fully Automatic, gradually minimizing human intervention, backed by a data engine that generates training masks for autonomous complex image segmentation.
* The model comprises an image encoder that generates image embeddings, a prompt encoder that utilizes a CLIP text encoder and convolutions for sparse and dense prompts, respectively, and a mask decoder that predicts and scores segmentation masks, returning the most accurate mask for detailed image manipulation.

# Diffusion Models

## Unconditional vs. Conditional Generation

Initially, diffusion models operated in an 'unconditional' manner, generating images without specific guidance. These models could produce any image based on their training dataset but lacked the ability to generate images based on specific instructions.

Today we have several ways of conditioning a Diffusion Model. Let's review a few:

* Sketch to Image Conditioning: In this approach, the model is conditioned with a sketch. The user provides a simple sketch or outline of an object or scene. The diffusion model then generates a detailed and realistic image representing the depicted object based on this sketch.



* Image to Image Conditioning: This form of conditioning involves starting with an existing image and asking the model to generate variations of it.



* Style Conditioning: Style conditioning involves asking the model to generate an image of something based on a given style represented by an input image.



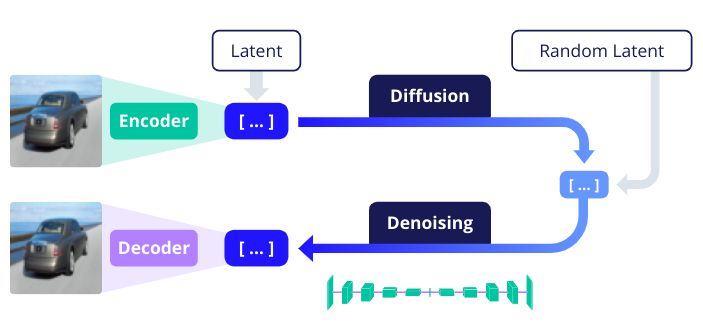
* Enter Text Conditioning: the most common conditioning by far is text conditioning. It introduces a transformative capability: guiding image generation with text descriptions.



## Latent Diffusion Models

Before diving into Latent Diffusion Models, it's essential to acknowledge the challenges faced by traditional pixel-level diffusion models. While powerful, these models are compute-intensive, especially at high resolutions. They require substantial computational resources, making them less accessible for general users.

Latent Diffusion Models represent a solution to these challenges. These models operate on 'latent space' rather than directly at the pixel level. This approach significantly reduces computational demands, allowing these models to generate high-quality images more efficiently.



* Autoencoders and Latent Space: Latent Diffusion Models use autoencoders. An autoencoder has two parts: an encoder and a decoder. The encoder compresses an image into a 'latent vector' or embedding, a much smaller representation of the image's content. The decoder does the reverse, expanding the latent vector back into an image.
* Efficient Training and Generation: The diffusion process in these models adds noise to the latent vector in several steps until it's fully randomized. The model is then trained to denoise these vectors back to their original form. Because the latent space is much smaller than the full image size, the model requires significantly less computational power.
* Practical Implications: Latent Diffusion Models can run faster and even on consumer hardware, making them more accessible. For instance, a 512x512 pixel image translates to over 260,000 numbers. In contrast, the latent representation in these models, such as the one used in Stable Diffusion, might be only about 16,000 numbers, vastly reducing the computational load.

## Pros and Cons of Diffusion Models

Advantages:

* Sample diversity
* High quality samples
* Ease d Conditioning

Disadvantages

* Speed.