IMAGE GENERATİON WİTH ARTİFİCİAL INTELLİGENCE

PRAPARER: GÜLSEHER BAŞAĞA

gulseher.basaga@ogr.sakarya.edu.tr

1. Is it possible to generate images with AI?

YES, ARTİFİCİAL İNTELLİGENCE CAN GENERATE REALİSTİC İMAGES FROM SCRATCH.

USİNG DEEP LEARNİNG AND NEURAL NETWORKS, AI SYSTEMS CAN LEARN PATTERNS FROM LARGE DATASETS AND CREATE ENTİRELY NEW VİSUALS.

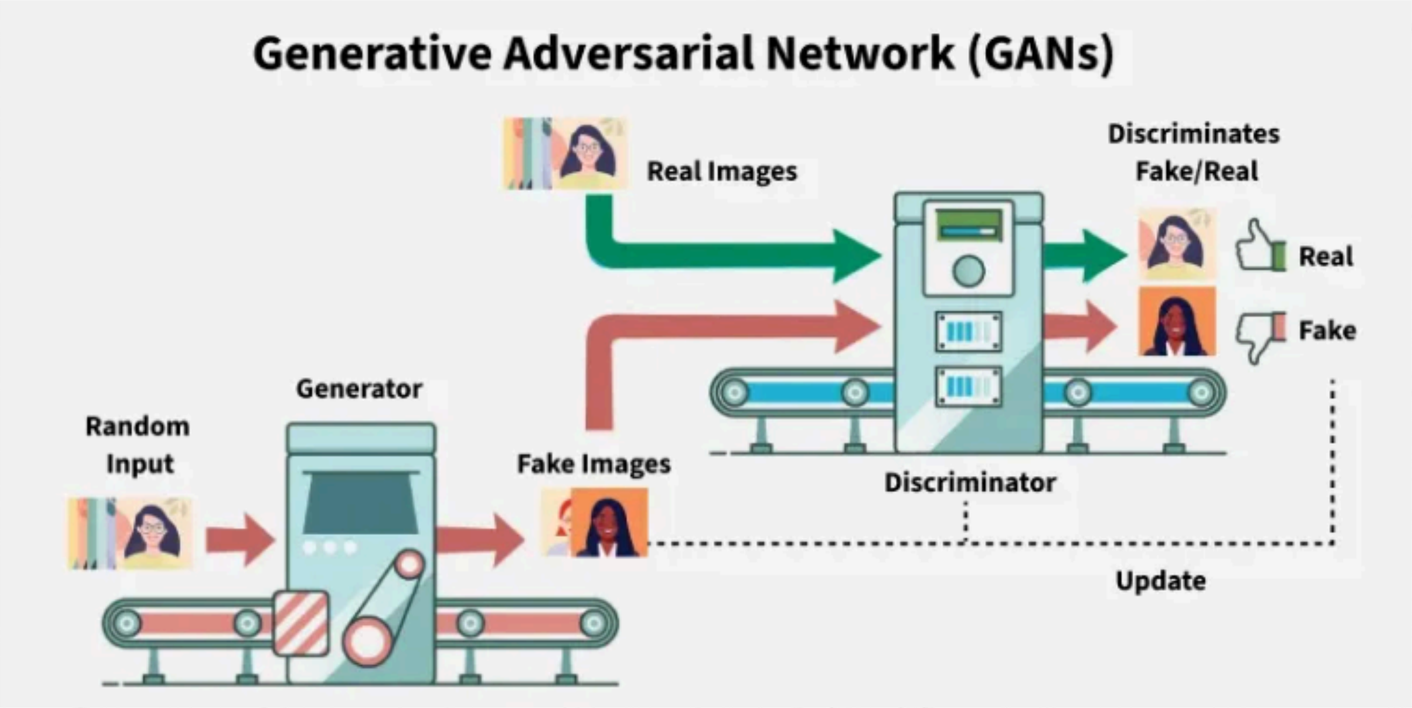
2. How do we do this? What technologies can we use?

•To generate images with AI, we use deep learning and neural networks.

•Technologies and tools:
•- GANs (Generative Adversarial Networks)
•- CNNs (Convolutional Neural Networks)
•- Autoencoders
•- Diffusion models
•- Tools: TensorFlow, PyTorch, Keras, etc.

What is a GAN?

- GAN stands for **Generative Adversarial Network.**
- It consists of two neural networks: **a Generator** and **a Discriminator.**
- The Generator creates fake images, while the Discriminator tries to detect which images are real.



Detailed Architecture of GANs

1. Generator Model

The generator is a deep neural network that takes random noise as input to generate realistic data samples (e.g., images or text). It learns the underlying data distribution by adjusting its parameters through **backpropagation**.

The generator's objective is to produce samples that the discriminator classifies as real. The loss function is:

JG=-1mΣi=1mlogD(G(zi))JG=-m1Σi=1mlogD(G(zi))

Detailed Architecture of GANs

2. Discriminator Model

The discriminator is a neural network that classifies input data as real or fake. During training, it receives both real and generated samples and learns to identify which is which.

It acts like a judge: real data gets a high score (close to 1) and fake data gets a low score (close to 0).

Over time, it becomes better at spotting fake data and helps the generator improve by challenging it to create more realistic outputs. This competition drives both models to get better.

Detailed Architecture of GANs

MinMax Loss in GANs

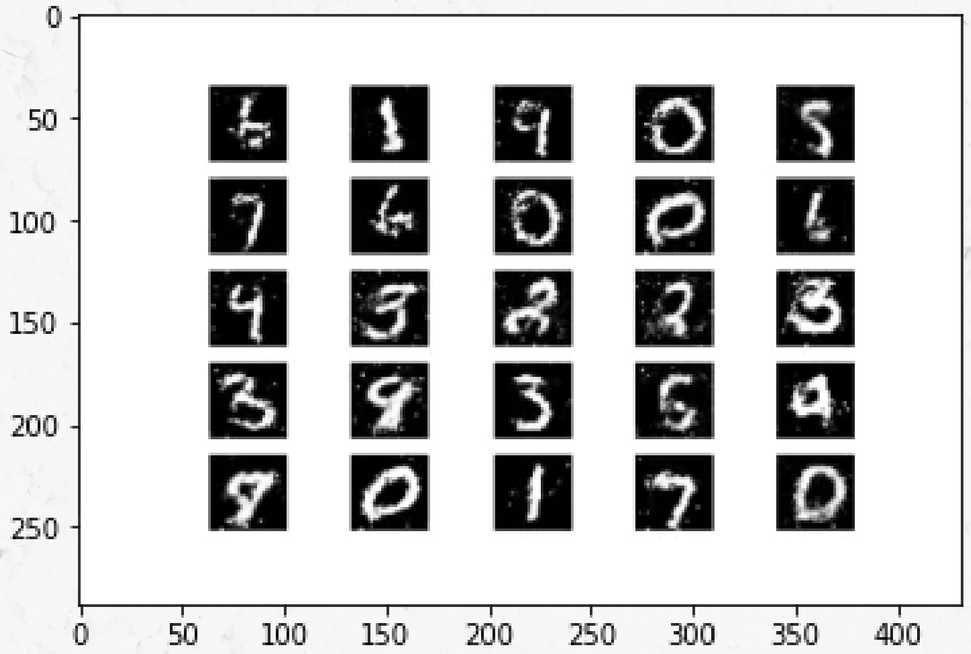
In Generative Adversarial Networks (GANs), training is framed as a two-player minimax game between the generator and the discriminator. The objective function is designed so that:

- The discriminator tries to maximize the probability of correctly classifying real and fake data.
- The generator tries to minimize the ability of the discriminator to distinguish fake data from real.

$$min_G\ max_D(G, D) = [\mathbb{E}_{x \sim p_{data}}[log\ D(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(g(z)))]$$

Types of GANs:

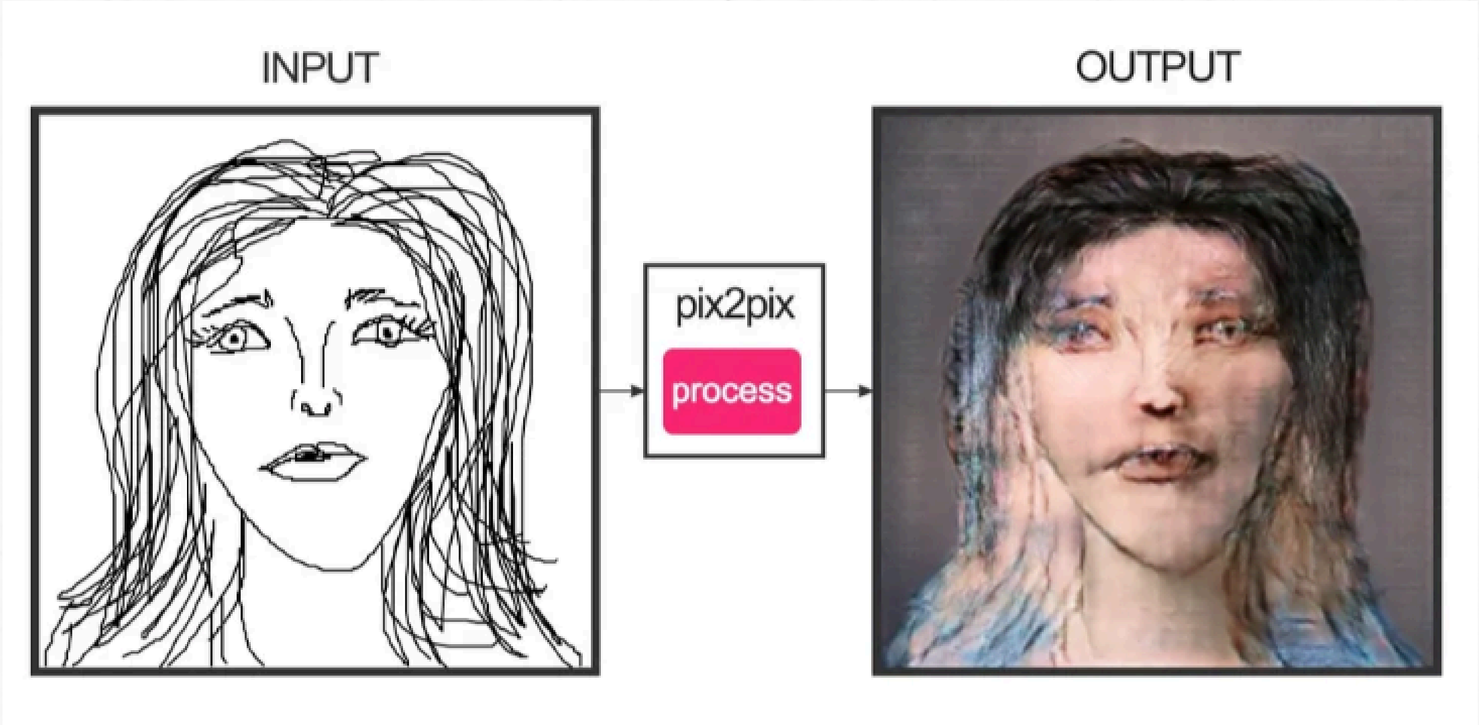
- DCGAN (Deep Convolutional GAN)
- CGAN (Conditional GAN)
- LSGAN (Least Squares GAN)
- ACGAN (Auxiliary Classifier GAN)
- DVD-GAN (Dual Video Discriminator GAN)
- SRGAN (Super-Resolution GAN)
- CycleGAN
- InfoGAN



Some Types of GANs

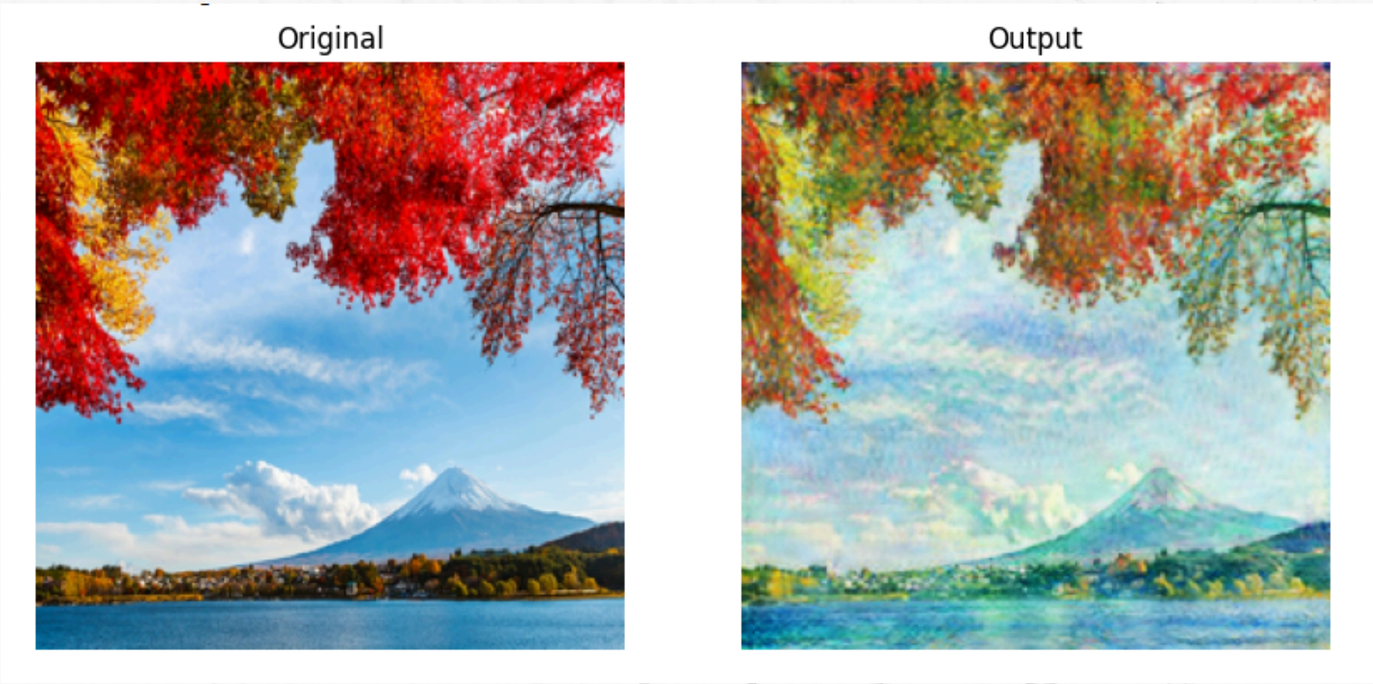
Pix2Pix

Pix2Pix is a supervised image-to-image translation model. It learns to convert images from one domain to another using paired training data. For example, it can learn to turn black-and-white images into color photos or convert a sketch into a realistic image.



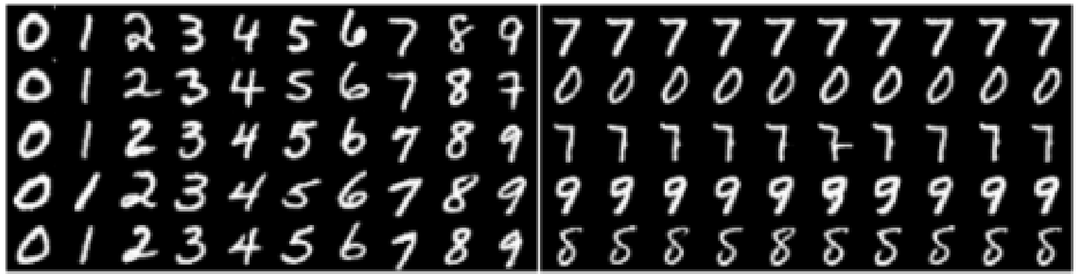
CycleGAN

CycleGAN is an unsupervised image-to-image translation model. It works without paired images and can translate styles between two domains (e.g., turning horse images into zebras or summer scenes into winter scenes) using cycle consistency loss.



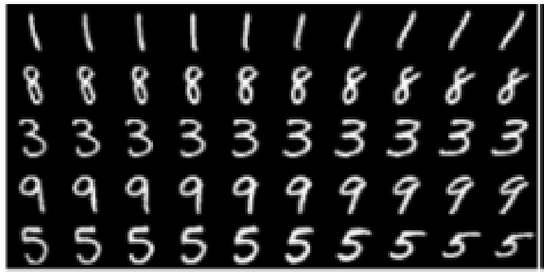
InfoGAN

InfoGAN is a GAN model that learns meaningful features from data, enabling it to generate more realistic and controlled outputs. It improves the interpretability of the generated data by focusing on important latent variables.

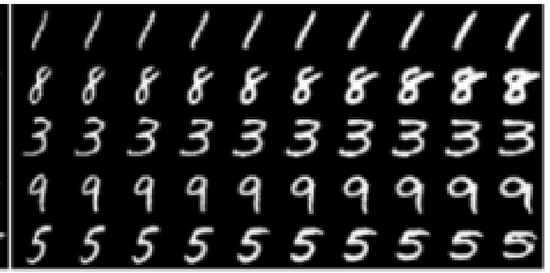


(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)



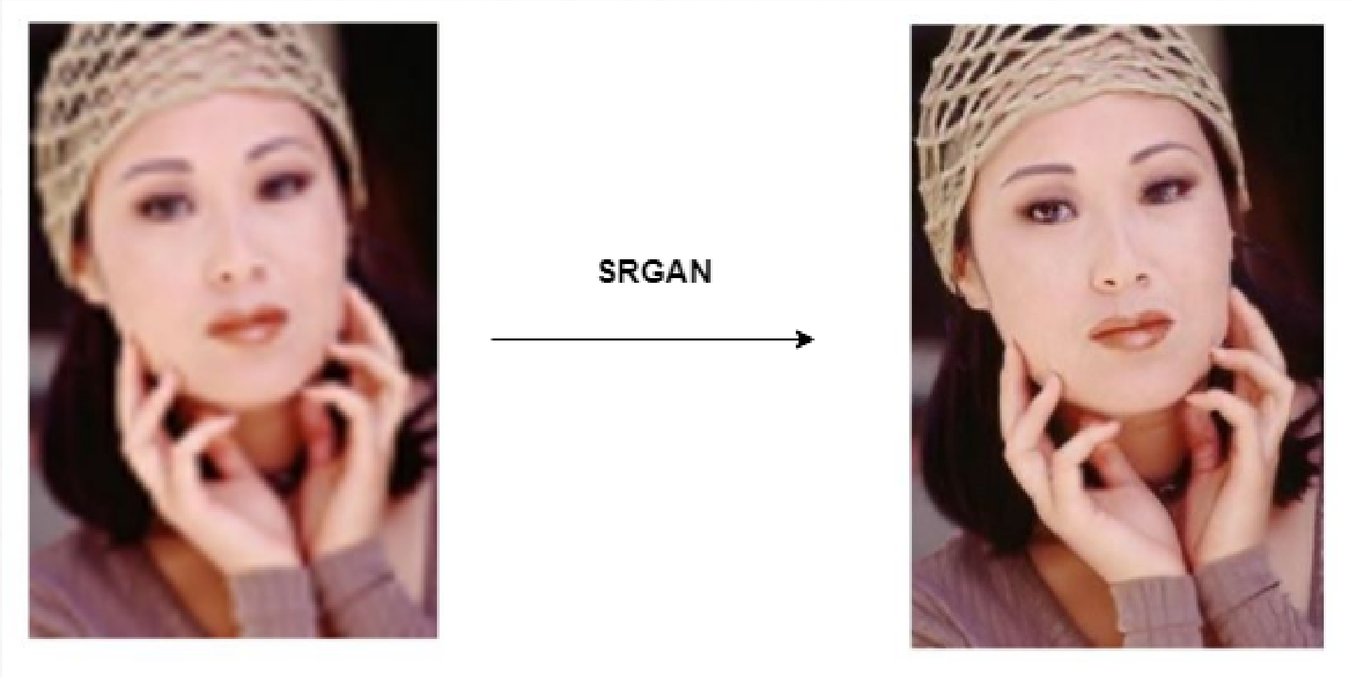
(c) Varying c_2 from −2 to 2 on InfoGAN (Rotation)



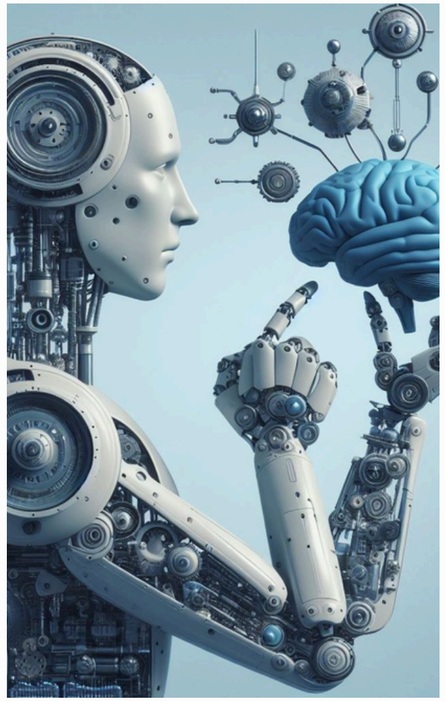
(d) Varying c_3 from −2 to 2 on InfoGAN (Width)

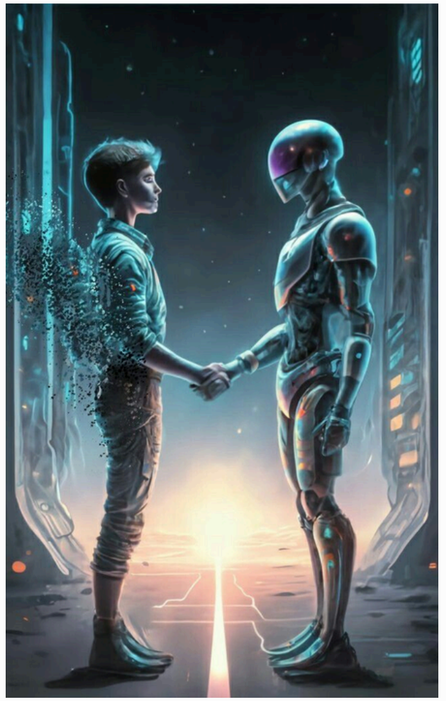
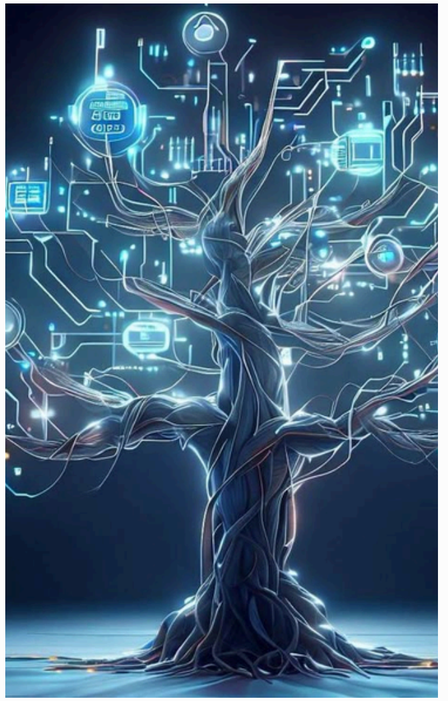
SRGAN

SRGAN (Super-Resolution GAN) is a type of GAN designed to convert low-resolution images into high-resolution ones. Its primary function is to improve image quality by increasing the resolution, making the details sharper and clearer. This process is known as super-resolution.



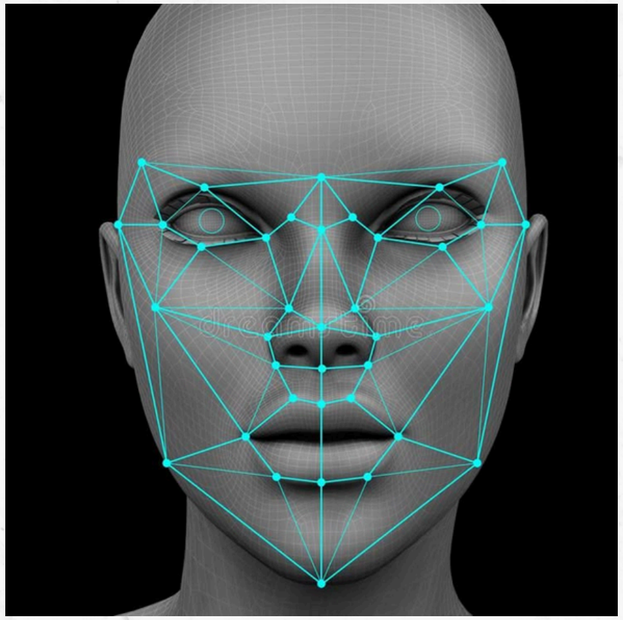
DO WE USE GANS İN EVERYDAY LİFE?

Face Filters on Social Media:

Social media apps like Instagram and Snapchat use GANs to apply real-time face filters, enhancing or altering facial features for fun or artistic effects.



Personalized Recommendations:

GANs contribute to recommendation systems on platforms like Netflix, YouTube, and Spotify by generating personalized suggestions based on user preferences.

Gaming and Virtual Reality:

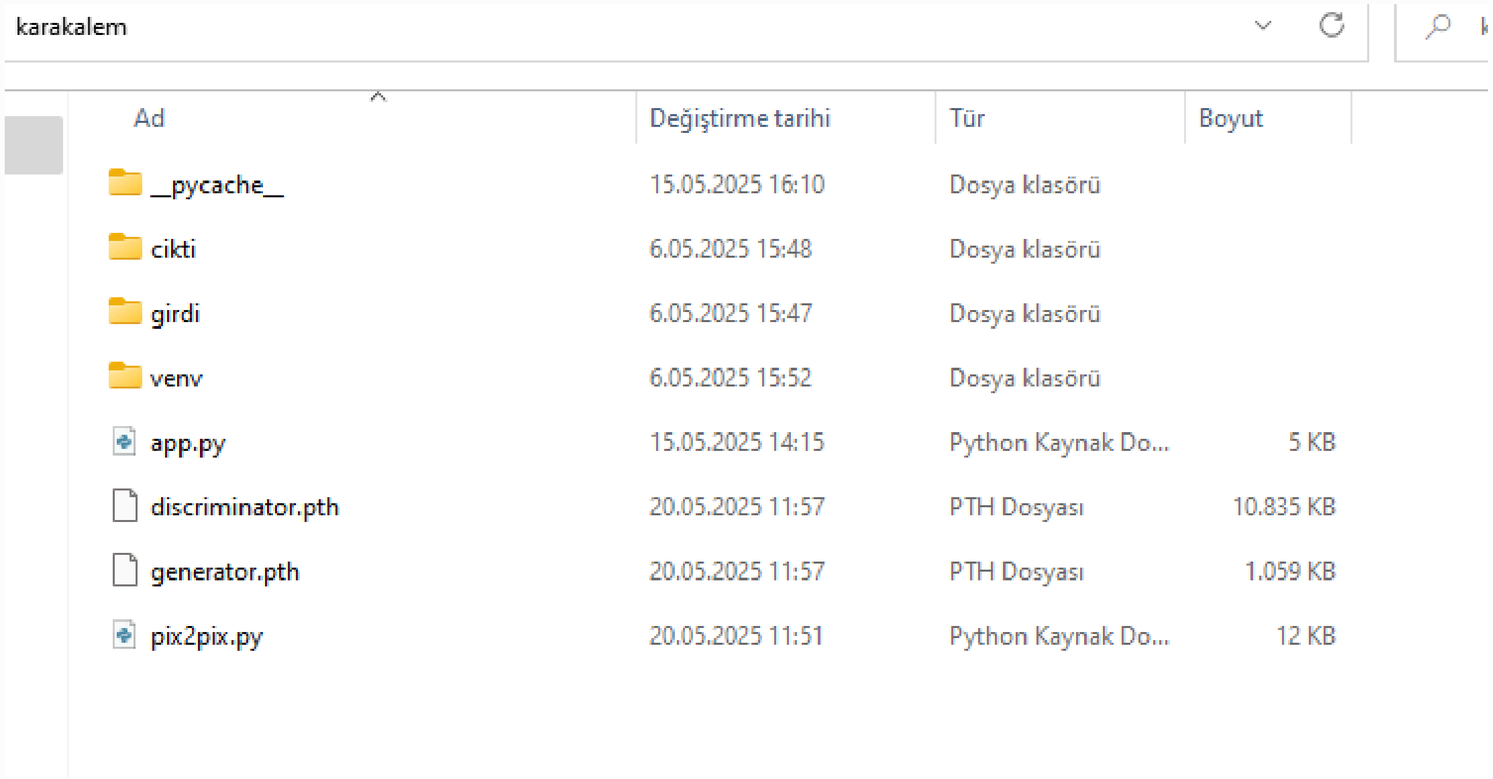
GANs help create realistic environments, characters, and assets for video games and virtual reality experiences, making them more immersive.

Virtual Try-Ons in Online Shopping:

Many online stores use GANs to offer virtual try-ons, allowing customers to see how clothes, accessories, or makeup would look on them without physically trying them on.

Example Project Using Pix2pix

File Structure



Girdi and Cikti Folders

```python
# Dosya isimlerini içindeki sayıya göre sıralamak için
def sort_by_number(file_list):
    return sorted(file_list, key=lambda x: int(re.findall(r'\d+', x)[0]))

class PairedDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.normal_dir = os.path.join(root_dir, "girdi")
        self.sketch_dir = os.path.join(root_dir, "cikti")

        self.normal_images = sort_by_number(os.listdir(self.normal_dir))
        self.sketch_images = sort_by_number(os.listdir(self.sketch_dir))

        #Eğer bir klasörde fazla görüntü varsa, eşleşmenin bozulmaması için daha az olan kadar veri kullanılır.
        self.length = min(len(self.normal_images), len(self.sketch_images))
        self.transform = transform

    def __len__(self):
        return self.length

    def __getitem__(self, idx): #TAM YOL VERİR
        normal_path = os.path.join(self.normal_dir, self.normal_images[idx])
        sketch_path = os.path.join(self.sketch_dir, self.sketch_images[idx])

        normal_image = Image.open(normal_path).convert("RGB")
        sketch_image = Image.open(sketch_path).convert("RGB")

        if self.transform:
            normal_image = self.transform(normal_image)
            sketch_image = self.transform(sketch_image)

        return normal_image, sketch_image #Sonuç olarak bir (girdi, çıktı) çifti döner.
```

The input and output folders contain the dataset that we send to the model for training. The model associates the input data with the output data."

This code prepares input (e.g., real image) and output (e.g., sketch) pairs for model training in PyTorch format. It sorts the images, applies transformations, and returns an (input, output) pair each time.

generator.pth and discriminator.pth

"With the help of Generator and Discriminator models, we produce an output similar to the input."

1.generator :

```python
class Generator(nn.Module):  #nn.Module, tüm modellerin türediği ana sınıftır. generator bu sınfı miras alır
    def __init__(self, latent_dim=3): #rgb görüntüler için 3 kanal
        super(Generator, self).__init__()
        #Tüm katmanları sıralı bir şekilde tanımlarız.- nn.Sequential, bir dizi katmanı sırayla uygular (ileri beslem
        self.model = nn.Sequential(
            #konvülasyon katmanı
            #2D görüntüler için konvolüsyon (filtreleme) işlemi yapar
            #kernel_size=4: Filtre boyutu 4x4
            #İki pikselde bir kayarak görüntüyü yarı boyuta indirir.
            nn.Conv2d(latent_dim, 64, 4, stride=2, padding=1),

            #relu aktivasyon fonksiyonudur ve negatif degerler 0 yapar, true ise belleği optimize eder
            nn.ReLU(True),

            nn.Conv2d(64, 128, 4, stride=2, padding=1),

            #Batch Normalization, öğrenmeyi hızlandırır ve denge sağlar.128 kanalı normalize eder
            nn.BatchNorm2d(128),

            nn.ReLU(True),

            #Konvolüsyonun Tersi (TransposeConv)
            #bu katman görüntüyü yeniden büyütür
            nn.ConvTranspose2d(128, 64, 4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 3, 4, stride=2, padding=1),

            #egitim sırasında veriler normalize edilmistir. bu yüzden çıktıyı Çıktıyı [-1, 1] aralığına sıkıştırır.
            nn.Tanh()
        )

    #Bu fonksiyon modelin çalışmasını tanımlar.
    def forward(self, x):
        return self.model(x)
```

The generator takes a random input (latent vector) and transforms it into an image.

The layers perform the following:

Conv2d: Filters and reduces image size.

ReLU: Activation function that zeroes out negatives.

BatchNorm: Stabilizes and accelerates training.

ConvTranspose2d: Upsamples to increase image size.

Tanh: Normalizes output to the -1,1-1, 1-1,1 range.

The model applies these layers sequentially to generate an image from noise.

2. discriminator:

```python
class Discriminator(nn.Module):
    def __init__(self, in_channels=3):
        super(Discriminator, self).__init__()

        #Katman Oluşturucu Fonksiyon (block):Bu fonksiyon, sık tekrar edilen katmanları olu
        #LeakyReLU: Aktivasyon fonksiyonu. Negatif değerleri sıfırlamak yerine küçük bir or
        def block(in_filters, out_filters, normalize=True):
            layers = [nn.Conv2d(in_filters, out_filters, 4, stride=2, padding=1)]
            if normalize:
                layers.append(nn.BatchNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers #Bu fonksiyon bir liste döndürür: Bu katmanlar tek başına kullanı

        self.model = nn.Sequential(
            #Girdi olarak hem orijinal resim hem de çıktı resim (gerçek ya da sahte) veriyo
             #Yani: 3 kanal (girdi) + 3 kanal (çıktı) = 6 kanal
             #İlk katman: Normalize edilmez (genelde ilk katmanda yapılmaz).
             #Girdi: 6 kanal → Çıktı: 64 kanal
             #Görsel boyutu küçülür.
            *block(in_channels * 2, 64, normalize=False),

            #64 → 128 kanala çıkar. Normalize edilir.
            *block(64, 128),

            #128 → 256 kanal
            *block(128, 256),
            *block(256, 512),

             #512 kanal → 1 kanallı sonuç haritası üretir.
            #Bu çıktı, her bölgenin "gerçek mi sahte mi" olduğuna dair tahminidir.
            #Bu yaklaşıma PatchGAN denir: Görselin tamamı yerine küçük parçalar değerlendir
            nn.Conv2d(512, 1, 4, padding=1)
        )
```

This code defines the Discriminator model used in GANs, specifically implementing the PatchGAN approach. The model takes both the real and generated images as input and tries to determine whether they are real or fake.

The block function builds repeated Conv-BatchNorm-LeakyReLU structures.
Input has 6 channels (3 from the real image + 3 from the generated image).
As it goes deeper, the spatial size decreases while the number of filters increases.

The final output is a single-channel realness map, where each patch is judged independently — a key idea in PatchGAN.

pix2pix.py

In this project, a GAN (Generative Adversarial Network) architecture is implemented using PyTorch. The Generator creates new images based on input images, while the Discriminator tries to distinguish between real and generated ones. A paired dataset of input-output images is used. The Generator is trained using both adversarial feedback from the Discriminator and pixel-wise comparison with real images. Two loss functions (MSELoss and L1Loss) are applied during training. Finally, after training, the models are saved as .pth files.

Required Libraries

```python
import os # Dosya ve klasör işlemleri için (veri yükleme vs.)
import re  # 👈 Dosya isimlerinden sayı çekmek için eklendi
import torch  #PyTorch ana kütüphane - tensörler ve model eğitimi için
import torch.nn as nn  # Derin öğrenme katmanlarını oluşturmak için (CNN, ReLU vb.)
from torchvision import transforms  # Görsel dönüşüm işlemleri için (resize, normalize vb.)
from torch.utils.data import Dataset, DataLoader # Verileri batch'leyip modele vermek için
from PIL import Image # Verileri batch'leyip modele vermek için
```

In this project, os and re are used for file handling, torch and torch.nn for tensors and model structure, torchvision.transforms for image transformations, Dataset and DataLoader for data loading, and PIL.Image for opening image files.

```python
# ------------------------
# Ayarlar
# ------------------------
epochs = 10
batch_size = 4
img_size = 256
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#
```

epochs = 10

The model will train by going through the entire dataset 10 times (10 epochs). It means the model sees all training data 10 times during learning.

batch_size = 4

The model processes data in small groups of 4 images at a time (a batch). This allows the model to work with data chunks that fit into memory.

img_size = 256

The size of the images processed is set to 256 pixels by 256 pixels (width and height). Usually, images are resized to square shape for consistent input size.

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

*This determines whether the model and data will be run on **GPU or CPU.** If an NVIDIA GPU with CUDA support is available, "cuda" (GPU) is used for faster computations; otherwise, CPU is used.*

Training loop

```
or epoch in range(epochs):
    for i, batch in enumerate(dataloader):
        if batch is None:
            continue
        real_A, real_B = batch
        real_A = real_A.to(device)
        real_B = real_B.to(device)

        valid = torch.ones((real_A.size(0), 1, 15, 15), requires_grad=False).to(device)
        fake = torch.zeros((real_A.size(0), 1, 15, 15), requires_grad=False).to(device)

        optimizer_G.zero_grad()

        fake_B = generator(real_A)
        g_loss = adversarial_loss(discriminator(real_A, fake_B), valid) + 100 * pixelwise_loss(fake_B, real_B)
        g_loss.backward()
        optimizer_G.step()

        optimizer_D.zero_grad()

        real_loss = adversarial_loss(discriminator(real_A, real_B), valid)
        fake_loss = adversarial_loss(discriminator(real_A, fake_B.detach()), fake)
        d_loss = 0.5 * (real_loss + fake_loss)
        d_loss.backward()
        optimizer_D.step()

        print(f"[Epoch {epoch}/{epochs}] [Batch {i}/{len(dataloader)}] [D loss: {d_loss.item():.4f}] [G loss: {g_loss.item():.4f}]")
```

This code trains a GAN. In each epoch, data is processed in batches. The Generator creates fake images from real ones, computes loss to fool the Discriminator, and updates its weights. The Discriminator calculates separate losses for real and fake images and updates its weights accordingly. Losses are printed after each batch.

app.py

This app takes a text prompt from the user and uses OpenAI's DALL·E model to generate an image based on that input. Then, it converts the generated image into a sketch-style drawing using a pre-trained Pix2Pix Generator model.

Everything runs on a Streamlit web interface.

The app performs two main tasks:

1) Generate an image using DALL·E

2) Convert the image into a sketch drawing using Pix2Pix

```python
# --- Generator Modelini Yükle ---
generator = Generator().to(device)  # Generator modeli oluşturulup ilgili cihaza gönderilir
generator.load_state_dict(torch.load("generator.pth", map_location=device))
# Eğitilmiş model ağırlıkları 'generator.pth' dosyasından yüklenir
generator.eval()  # Model 'evaluation' moduna alınır (dropout, batchnorm vs. devre dışı)
```

Model is loaded

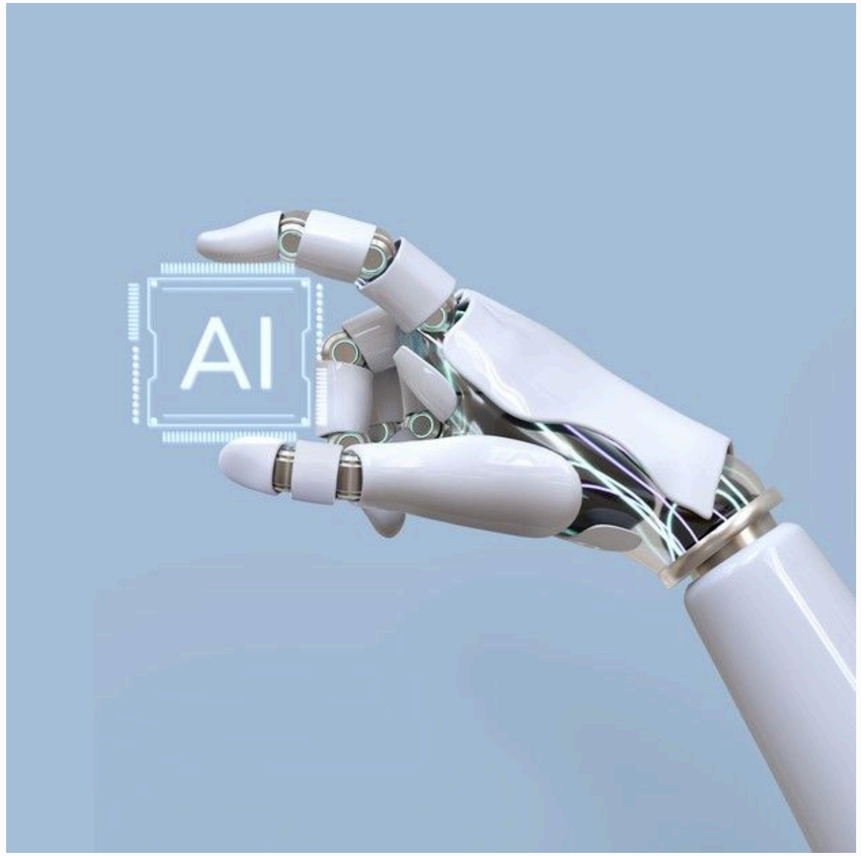
```python
# --- Görseli Karakaleme Çeviren Fonksiyon ---
def generate_sketch(image):  # Bu fonksiyon girilen görseli karakalem çizime dönüştürür
    transform = transforms.Compose([
        transforms.Resize((256, 256)),  # Görseli 256x256 boyutuna getirir
        transforms.ToTensor()  # Görseli [0,1] aralığında tensöre çevirir
    ])
    input_image = transform(image).unsqueeze(0).to(device)
    # Görselin boyutları [1, 3, 256, 256] olacak şekilde düzenlenip cihaza aktarılır

    with torch.no_grad():  # Bu blokta grad hesaplaması yapılmaz, bu da hız ve bellek kazancı sağlar
        output = generator(input_image)  # Görsel Generator modeline verilir ve çıktı alınır

    output_image = output.squeeze().cpu().clamp(0, 1)
    # Çıktı tensörü CPU'ya alınır, boyutu sıkıştırılır, değerler 0-1 aralığına kısıtlanır
    return transforms.ToPILImage()(output_image)  # Tensor tekrar PIL görseline çevrilip döndürülür
```

the image is converted into pencil drawing

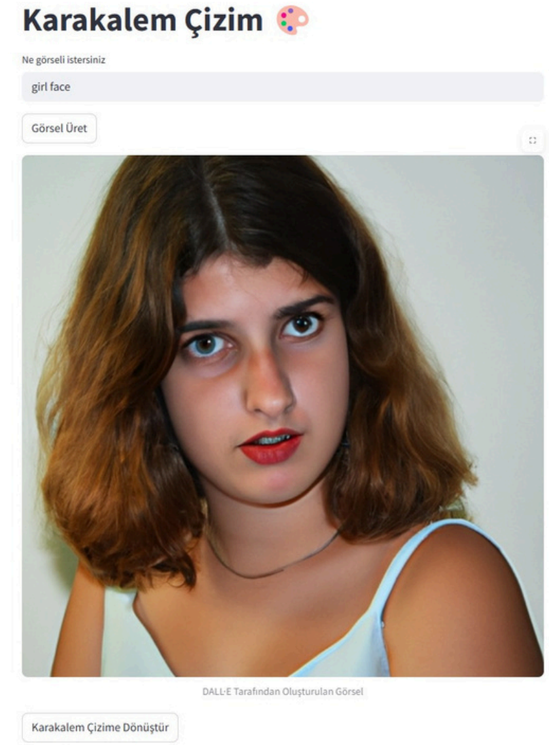
```python
# --- Görsel Üret Butonu ---
if st.button("Görsel Üret"):  # Bu butona tıklandığında aşağıdaki işlemler gerçekleşir
    if prompt:  # Eğer kullanıcı boş prompt vermediyse
        with st.spinner("DALL·E görsel oluşturuyor..."):  # İşlem süresince spinner gösterilir
            response = openai.Image.create(
                prompt=prompt,  # Kullanıcının yazdığı metin (prompt) modele verilir
                n=1,  # Kaç görsel oluşturulacağı (1 tane)
                size="512x512"  # Görsel boyutu (512x512 önerilen boyuttur)
            )
            image_url = response["data"][0]["url"]  # Oluşturulan görselin URL'si çekilir
            image_response = requests.get(image_url)  # URL üzerinden görsel indirilir
            dalle_image = Image.open(BytesIO(image_response.content)).convert("RGB")
            # Görsel RGB formatında PIL Image olarak açılır
            st.image(dalle_image, caption="DALL·E Tarafından Oluşturulan Görsel", use_container_width=True)
            # Görsel arayüzde gösterilir (yeni parametre: use_container_width)

            # Görseli oturumda sakla
            st.session_state["generated_image"] = dalle_image  # Görseli session'da saklıyoruz
    else:
        st.error("Lütfen bir prompt girin.")  # Eğer prompt boşsa kullanıcı uyarılır

# --- Karakalem Butonu ---
if "generated_image" in st.session_state:  # Eğer önceki aşamada bir görsel oluşturulduysa
    if st.button("Karakalem Çizime Dönüştür"):  # Bu butona basıldığında aşağıdaki işlemler yapılır
        with st.spinner("Karakalem çizim hazırlanıyor..."):  # İşlem süresince spinner gösterilir
            sketch = generate_sketch(st.session_state["generated_image"])  # Görseli karakalem haline çevir
            st.image(sketch, caption="Karakalem Çizim", use_container_width=True)  # Sonuç arayüzde gösterilir
            sketch.save("generated_sketch_from_prompt.jpg")  # Görsel diske kaydedilir
            st.success("Karakalem çizim tamamlandı!")  # Kullanıcıya başarı mesajı gösterilir
```

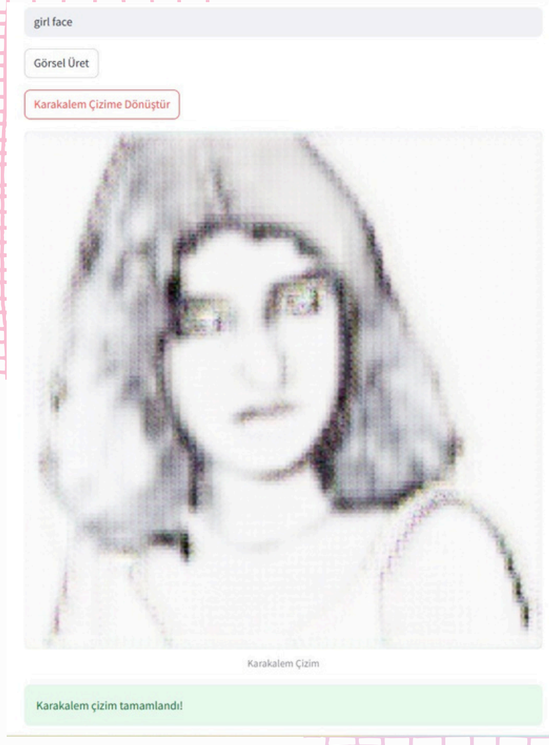
This code generates an image using OpenAI's DALL·E model based on a user prompt, then converts that image into a pencil sketch using a Pix2Pix model. When the "Generate Image" button is clicked, a DALL·E image is created and displayed. When "Convert to Sketch" is clicked, the image is transformed into a sketch and both shown and saved.

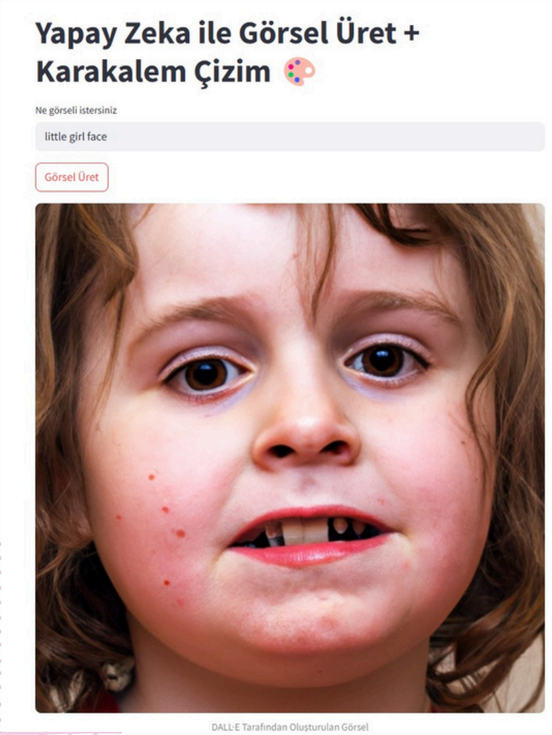
"Code Execution"

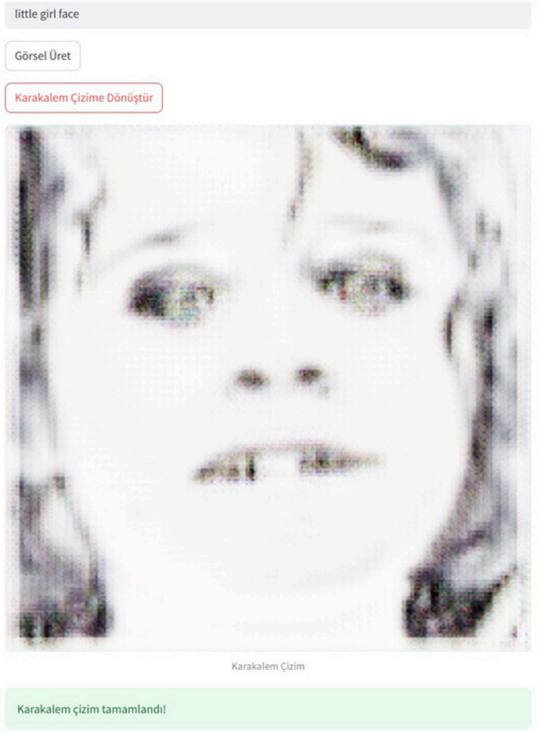


When you run the application, enter a few words describing the image you want to generate in the prompt field.









Here are some sample inputs and outputs.

You can visit my GitHub repository to access the code:
https://github.com/GulseherB