Generative Adversarial Networks

When AI Learns to Paint

Prompts

- I have a set of cat pictures. Could you write a simple Python code with Keras to make the GAN draw a cat for me?

Strategy

- Prepare the data
- Build a neural network, compile the model, train it, and evaluate

Code & Results

1. Prepare the data

This code loads the dataset.

```python
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# ----------------------------
# Config
# ----------------------------
DATA_DIR = "./data/cats"
IMAGE_SIZE = 32
BATCH_SIZE = 8
NOISE_DIM = 32
STEPS = 100000  # total training iterations
SAVE_EVERY = 100
OUT_DIR = "./gan_tiny_out"
os.makedirs(OUT_DIR, exist_ok=True)

# ----------------------------
# Load data (small & safe)
# ----------------------------
def load_image(path):
    img = tf.io.read_file(path)
    img = tf.image.decode_image(img, channels=3)
    img.set_shape([None, None, 3])
    img = tf.image.resize(img, [IMAGE_SIZE, IMAGE_SIZE])
    img = tf.cast(img, tf.float32) / 127.5 - 1.0
    return img

def make_dataset(data_dir):
    files = [f for f in glob.glob(os.path.join(data_dir, "*"))
             if f.lower().endswith((".jpg",".jpeg",".png"))]
    if not files:
        raise RuntimeError(f"No cat images in {data_dir}")
    print(f"🐱 Found {len(files)} images.")
    ds = tf.data.Dataset.from_tensor_slices(files)
    ds = ds.shuffle(len(files)).map(load_image).batch(BATCH_SIZE)
    return ds
```

≡ kaggle

kaggle.com/datasets/

Cats-And-Dogs-Mini-Dataset

2. Build and evaluate the Model

This code builds the Generator and Discriminator.

```python
# ----------------------------
# Very small models
# ----------------------------
def make_generator():
    return tf.keras.Sequential([
        layers.Input(shape=(NOISE_DIM,)),
        layers.Dense(128, activation="relu"),
        layers.Dense(IMAGE_SIZE * IMAGE_SIZE * 3, activation="tanh"),
        layers.Reshape((IMAGE_SIZE, IMAGE_SIZE, 3))
    ])

def make_discriminator():
    return tf.keras.Sequential([
        layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3)),
        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dense(1, activation="sigmoid")
    ])
```

This code saves sample images.

```python
# ----------------------------
# Save sample images
# ----------------------------
def save_images(generator, step):
    noise = tf.random.normal([9, NOISE_DIM])
    imgs = generator(noise, training=False)
    imgs = (imgs + 1) / 2.0
    fig, axes = plt.subplots(3, 3, figsize=(4,4))
    for i, ax in enumerate(axes.flatten()):
        ax.imshow(imgs[i])
        ax.axis("off")
    plt.tight_layout()
    out = os.path.join(OUT_DIR, f"cats_{step:04d}.png")
    plt.savefig(out)
    plt.close()
    print(f"💾 Saved {out}")
```

This code trains the model.

```python
# ----------------------------
# Training Loop
# ----------------------------
def train():
    ds = make_dataset(DATA_DIR)
    gen = make_generator()
    disc = make_discriminator()
    bce = tf.keras.losses.BinaryCrossentropy()
    g_opt = tf.keras.optimizers.Adam(1e-4)
    d_opt = tf.keras.optimizers.Adam(1e-4)

    for step, real in enumerate(ds.repeat(), start=1):
        noise = tf.random.normal([BATCH_SIZE, NOISE_DIM])

        # --- Train discriminator ---
        with tf.GradientTape() as tape:
            fake = gen(noise, training=True)
            real_pred = disc(real, training=True)
            fake_pred = disc(fake, training=True)
            d_loss = bce(tf.ones_like(real_pred), real_pred) + \
                     bce(tf.zeros_like(fake_pred), fake_pred)
        grads = tape.gradient(d_loss, disc.trainable_variables)
        d_opt.apply_gradients(zip(grads, disc.trainable_variables))

        # --- Train generator ---
        noise = tf.random.normal([BATCH_SIZE, NOISE_DIM])
        with tf.GradientTape() as tape:
            fake = gen(noise, training=True)
            preds = disc(fake, training=False)
            g_loss = bce(tf.ones_like(preds), preds)
        grads = tape.gradient(g_loss, gen.trainable_variables)
        g_opt.apply_gradients(zip(grads, gen.trainable_variables))

        if step % 50 == 0:
            print(f"Step {step}: D={d_loss:.3f}  G={g_loss:.3f}")
        if step % SAVE_EVERY == 0:
            save_images(gen, step)
        if step >= STEPS:
            break

    gen.save(os.path.join(OUT_DIR, "generator_final.h5"))
    print("✅ Done.")

if __name__ == "__main__":
    train()
```

Made by: [@katerynakononova](https://katerynakononova)
Website: https://katerynakononova.github.io/meowlearning/
Book: https://www.amazon.com/dp/B0CW9SFYXF