Certainly! Below are **complete TensorFlow implementations** for:

1. **Simple GAN**: Generates 2D data resembling the "moons" dataset.
2. **Variational Autoencoder (VAE)**: Trained on the MNIST dataset.
3. **Neural Style Transfer**: Applies the artistic style of one image to the content of another.Paperspace by DigitalOcean Blog+3GeeksforGeeks+3TensorFlow+3

🌀 1. Simple GAN in TensorFlow (on make_moons)

This GAN learns to generate 2D data points similar to the "moons" dataset.

python

CopyEdit

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
import tensorflow as tf
from tensorflow.keras import layers

# Generate the dataset
X, _ = make_moons(n_samples=1000, noise=0.1)
X = X.astype('float32')

# Define the generator model
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(16, activation='relu', input_shape=(2,)),
        layers.Dense(2)
    ])
    return model

# Define the discriminator model
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Dense(16, activation='relu', input_shape=(2,)),
```

```
        layers.Dense(1, activation='sigmoid')
    ])
    return model

# Instantiate models
generator = build_generator()
discriminator = build_discriminator()

# Compile the discriminator
discriminator.compile(optimizer='adam', loss='binary_crossentropy')

# Combine models for the GAN
discriminator.trainable = False
gan_input = tf.keras.Input(shape=(2,))
generated_data = generator(gan_input)
gan_output = discriminator(generated_data)
gan = tf.keras.Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy')

# Training loop
epochs = 10000
batch_size = 64

for epoch in range(epochs):
    # Train discriminator
    idx = np.random.randint(0, X.shape[0], batch_size)
    real_data = X[idx]
    noise = np.random.normal(0, 1, (batch_size, 2))
    fake_data = generator.predict(noise, verbose=0)

    d_loss_real = discriminator.train_on_batch(real_data, np.ones((batch_size, 1)))
    d_loss_fake = discriminator.train_on_batch(fake_data, np.zeros((batch_size, 1)))
```

```
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)


    # Train generator

    noise = np.random.normal(0, 1, (batch_size, 2))

    g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))


    # Display progress

    if epoch % 1000 == 0:

        print(f"Epoch {epoch}, Discriminator Loss: {d_loss}, Generator Loss: {g_loss}")

        # Plot generated data

        generated_samples = generator.predict(np.random.normal(0, 1, (1000, 2)),
verbose=0)

        plt.figure(figsize=(6, 6))

        plt.scatter(X[:, 0], X[:, 1], c='blue', label='Real Data', alpha=0.5)

        plt.scatter(generated_samples[:, 0], generated_samples[:, 1], c='red',
label='Generated Data', alpha=0.5)

        plt.legend()

        plt.title(f"Epoch {epoch}")

        plt.show()
```

🔁 2. Variational Autoencoder (VAE) on MNIST

This VAE learns to encode and decode MNIST digits.[keras.io]

python

CopyEdit

```
import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers


# Load MNIST dataset

(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()

x_train = x_train.astype('float32') / 255.

x_train = x_train.reshape(-1, 28 * 28)
```

```
x_test = x_test.astype('float32') / 255.
x_test = x_test.reshape(-1, 28 * 28)

latent_dim = 2

# Define the encoder
encoder_inputs = tf.keras.Input(shape=(784,))
x = layers.Dense(512, activation='relu')(encoder_inputs)
z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)

# Sampling function
def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.random.normal(shape=(tf.shape(z_mean)[0], latent_dim))
    return z_mean + tf.exp(0.5 * z_log_var) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])

# Define the decoder
decoder_inputs = tf.keras.Input(shape=(latent_dim,))
x = layers.Dense(512, activation='relu')(decoder_inputs)
decoder_outputs = layers.Dense(784, activation='sigmoid')(x)
decoder = tf.keras.Model(decoder_inputs, decoder_outputs)

# Connect encoder and decoder
encoder = tf.keras.Model(encoder_inputs, z)
outputs = decoder(z)

# Define the VAE model
vae = tf.keras.Model(encoder_inputs, outputs)
```

```
# Define the loss
reconstruction_loss = tf.keras.losses.binary_crossentropy(encoder_inputs, outputs)
reconstruction_loss *= 784
kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
kl_loss = -0.5 * tf.reduce_sum(kl_loss, axis=1)
vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
vae.compile(optimizer='adam')

# Train the VAE
vae.fit(x_train, x_train, epochs=10, batch_size=128)

# Generate new samples
z_samples = np.random.normal(size=(16, latent_dim))
generated_images = decoder.predict(z_samples)
generated_images = generated_images.reshape(-1, 28, 28)

# Plot the generated images
plt.figure(figsize=(4, 4))
for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.imshow(generated_images[i], cmap='gray')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

🎨 3. Neural Style Transfer (TensorFlow)

This implementation applies the style of one image to the content of another using a pre-trained VGG19 model.[GeeksforGeeks]

python

CopyEdit

```
import tensorflow as tf
```

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications import vgg19
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model

# Load and preprocess images
def load_and_process_image(path):
    img = load_img(path, target_size=(224, 224))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    return vgg19.preprocess_input(img)

# Deprocess image
def deprocess_image(x):
    x = x.reshape((224, 224, 3))
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    x = x[:, :, ::-1]
    return np.clip(x, 0, 255).astype('uint8')

# Load images
content_path = 'path_to_content_image.jpg'
style_path = 'path_to_style_image.jpg'
content_image = load_and_process_image(content_path)
style_image = load_and_process_image(style_path)

# Define the model
model = vgg19.VGG19(weights='imagenet', include_top=False)
model.trainable = False
```

```
# Define layers to use
content_layers = ['block5_conv2']
style_layers = ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1']

# Get outputs of the selected layers
outputs = [model.get_layer(name).output for name in (style_layers + content_layers)]
model = Model([model.input], outputs)

# Compute content and style features
def get_features(image):
    outputs = model(image)
    style_outputs = outputs[:len(style_layers)]
    content_outputs = outputs[len(style_layers):]
    return style_outputs, content_outputs

# Compute Gram matrix
def gram_matrix(tensor):
    channels = int(tensor.shape[-1])
    a = tf.reshape(tensor, [-1, channels])
    return tf.matmul(a, a, transpose_a=True)

# Compute loss
def compute_loss(generated, style_features, content_features):
    generated_style, generated_content = get_features(generated)
    style_loss = tf.add_n([tf.reduce_mean((gram_matrix(gs) - gram_matrix(ts)) ** 2)
                           for gs, ts in zip(generated_style, style_features)])
    content_loss = tf.add_n([tf.reduce_mean((gc - tc) ** 2)
                             for gc, tc in zip(generated_content, content_features)])
    return style_loss * 1e-4 + content_loss

# Initialize generated image
generated_image = tf.Variable(content_image, dtype=tf.float32)
```

```
# Extract features
style_features, _ = get_features(style_image)
_, content_features = get_features(content_image)

# Optimizer
optimizer = tf.optimizers.Adam(learning_rate=5.0)

# Training loop
epochs = 100
for i in range(epochs):
    with tf.GradientTape() as tape:
        loss = compute_loss(generated_image, style_features, content_features)
    grad = tape.gradient(loss, generated_image)
    optimizer.apply_gradients([(grad, generated_image)])
    generated_image.assign(tf.clip_by_value(generated_image, -127, 127))
    if i % 20 == 0:
        print(f"Epoch {i}, Loss: {loss.numpy()}")

# Display the stylized image
final_img = deprocess_image(generated_image.numpy())
plt.imshow(final_img)
plt.axis('off')
plt.show()
```

These implementations provide foundational examples of GANs, VAEs, and Neural Style Transfer using TensorFlow.