Conditional Image Generation using Gaussian GAN

Team Aspirers
Partha Mete and Sudam Kumar Paul
metepartha2001@gmail.com and 2002sudam@gmail.com

August 20, 2025

Abstract

This project builds a GauGAN-based image generator that creates realistic images from semantic segmentation masks. The generator uses SPADE layers in a residual network and takes two inputs: the segmentation mask (class ID image, converted to a one-hot label map) and a latent code from an encoder, which allows diverse outputs from the same mask. Training combines multiple objectives: hinge adversarial loss for realism, feature matching loss for stable structure, VGG perceptual loss for visual quality, and KL divergence for diversity. A multi-scale PatchGAN discriminator judges real versus fake images and provides intermediate features for generator training. The dataset is prepared with random cropping, one-hot label encoding, and normalization. Performance is evaluated using Fréchet Inception Distance (FID) for realism and mean Intersection-over-Union (mIoU) with pixel accuracy from a pretrained DeepLabV3-ResNet_50 model for semantic consistency. Results show that the model generates diverse, realistic, and semantically faithful images, while also revealing challenges such as normalization choices, loss balancing, and dataset preprocessing. This work delivers a complete TensorFlow/Keras pipeline for training and evaluating GauGAN..

Through ablation studies, we confirmed that SPADE layers significantly outperformed traditional normalization techniques in preserving semantic information during generation. Overall, our project demonstrates the effectiveness of SPADE and GauGAN architecture for conditional photorealistic image synthesis across both simple and complex datasets.

1 Introduction

What?
This project develops a GauGAN-based generative model that can create realistic images from semantic segmentation masks. A segmentation mask is an image where each pixel corresponds to a class label (e.g., building, tree, sky), and by using it as input, the generator learns to produce photorealistic images that follow this structure. The model uses

SPADE (Spatially Adaptive Denormalization) layers in the generator, a multi-scale PatchGAN discriminator, and an image encoder to enable diversity in outputs. The system is trained with a combination of adversarial, perceptual, and regularization losses, and evaluated using Fréchet Inception Distance (FID) and segmentation consistency metrics (mIoU, pixel accuracy).

Why?
Standard GANs can generate high-quality images but lack control over the generated content. For applications such as urban scene generation, content creation, simulation, and augmented reality, it is important to guide the model with semantic structure. GauGAN solves this by allowing users to sketch or provide segmentation masks that determine the layout of the generated scene. This makes the model useful for both practical applications (automatic scene rendering, data augmentation for segmentation tasks) and creative tasks (turning rough sketches into realistic scenes).
However, building GauGAN in TensorFlow/Keras requires careful design of the generator, discriminator, and training losses, along with robust evaluation methods. This project addresses these challenges and provides a complete, reproducible pipeline for training and evaluating GauGAN on paired image–segmentation datasets.

How?
The model architecture and training procedure are designed as follows:

1. **Encoder** Maps real images to a latent distribution, allowing sampling of different latent codes for diverse outputs.

2. **Gaussian Sampler** Samples latent vectors from the learned distribution.

3. **Generator** Built with SPADE ResBlocks that adapt normalization using segmentation masks. It takes as input a latent vector (from the encoder) and the one-hot label map of the segmentation mask, and produces realistic RGB images.

4. **Discriminator** A multi-scale PatchGAN that evaluates real vs fake images, conditioned on segmentation maps. Its intermediate activations are also used for feature matching loss, which improves stability.

Training is done using:

1. *Hinge adversarial loss* (realism),

2. *Feature matching loss* (stability),

3. *VGG perceptual loss* (texture quality),

4. *KL divergence loss* (diversity in latent space).

We have used the Facades and PASCAL VOC 2012 datasets, with proper data preprocessing, label maps with random cropping, segmentation masks, one-hot label encoding and normalization. Evaluation metrics included FID for visual quality and a segmentation-consistency metric (mIoU and pixel accuracy from DeepLabV3) to check alignment between generated images and input masks.

Through this design, the project delivers a complete end-to-end GauGAN training framework in TensorFlow/Keras, capable of producing diverse, semantically faithful, and perceptually realistic images.

2 Literature review

Image-to-Image Translation with Conditional Adversarial Networks" by Isola et al. [2](2018) presents a general framework for image-to-image translation using Conditional Generative Adversarial Networks (cGANs), which learn to map input images to output images while also learning the loss function that guides the translation. By combining an adversarial loss with a traditional L1 loss, the model generates sharper and more realistic results than previous methods relying solely on pixel-wise differences. The authors use a U-Net-based generator and a PatchGAN discriminator to encourage local detail and global coherence. Their approach is applied successfully to a variety of tasks—including colorization, edge-to-photo, and map-to-aerial photo translation—demonstrating that a single architecture and training procedure can generalize across many domains without task-specific design.

Semantic Image Synthesis with Spatially-Adaptive Normalization" by Park et al.[4](2019) introduces SPADE (SPatially-Adaptive (DE)normalization), a novel normalization technique designed for semantic image synthesis, where the goal is to generate photorealistic images from segmentation masks. Traditional normalization layers in deep networks often erase semantic information, but SPADE retains and leverages this by injecting the segmentation map directly into the normalization layers through spatially-varying affine transformations. The authors demonstrate that using SPADE in a lightweight generator architecture significantly improves both the visual fidelity and semantic alignment of generated images across several datasets, including COCO-Stuff, ADE20K, and Cityscapes. The approach also supports multi-modal and style-guided synthesis by combining SPADE with random or style-encoded inputs, enabling controllable and diverse outputs. Extensive experiments, ablation studies, and human evaluations show that SPADE outperforms existing methods like pix2pixHD and CRN in both objective metrics and subjective preference.

Image Segmentation by MAP-ML Estimations" by Shifeng Chen, Liangliang Cao et al. [1](2010) proposes a novel probabilistic image segmentation algorithm based on an itera-

tive optimization framework that alternates between Maximum A Posteriori (MAP) and Maximum Likelihood (ML) estimations. The MAP estimation is modeled with Markov Random Fields (MRFs) and solved using graph cuts to enforce spatial smoothness, while the ML estimation updates region parameters assuming a Gaussian distribution of pixel features, including color and texture. A key advantage of this method is its ability to automatically determine the number of segments without prior specification. Extensive experiments on the Berkeley segmentation benchmark show that this approach outperforms six state-of-the-art methods both qualitatively and quantitatively, achieving better alignment with human perception. The algorithm is also shown to be robust, efficient, and adaptable to various statistical models beyond Gaussian.

For our implementation of Gaussian GAN we have trained our model on Pascal VOC 2012 dataset, which required writing a custom one-hot preprocessing pipeline with void-class handling.We have implemented quantitative evaluation such as FID.Also our version includes model checkpointing, loss tracking, and custom training callbacks, making it easier to scale and reuse.Then we also made the code modular using tf.data pipeline and AUTOTUNE so it can be used with other datasets.

3 Proposed methodology

To address the problem of semantic image synthesis from segmentation maps, we propose an end-to-end deep learning framework based on the GauGAN architecture, which combines a variational autoencoder with a SPADE-based conditional GAN. Our method allows for controllable and diverse image generation aligned with the semantic structure provided in the input segmentation maps. The key components of our methodology are outlined below:

1. **Data Preparation**
   We have used two datasets for this project:

   - **Facades Dataset:** Contains building façade layouts with corresponding real images.
   - **PASCAL VOC 2012 Dataset:** A more complex dataset with 21 semantic classes such as people, animals, and vehicles. In Original version of this dataset there are 11000 real images but for GauGAN training, segmentation map and label are needed corresponding to each image. In the dataset those are not avaliable corresponding to each image so after cleaning the data paired segmentation map and lable are found for only 2913 images. The rest real images and left over segmentation maps and lables were removed from our processed dataset.

Each image is preprocessed by resizing, normalization to the range $[-1, 1]$, and conversion of segmentation masks to one-hot encoded label maps. A random cropping strategy is employed to introduce spatial diversity.

2. **Model Architecture**

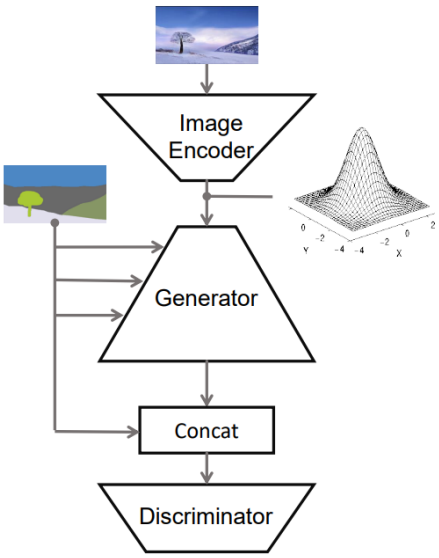

Figure 1: Overview of the model architecture.

- **Encoder:** A convolutional encoder processes the real image and outputs a mean and variance vector. These define a Gaussian distribution from which we sample a latent vector using the reparameterization trick, enabling multimodal generation.
- **Gaussian Sampler:** A Gaussian sampler layer has implemented that takes the encoder's outputs and generates a latent code using:

$$z = \mu + \sigma \cdot \epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, I)$$

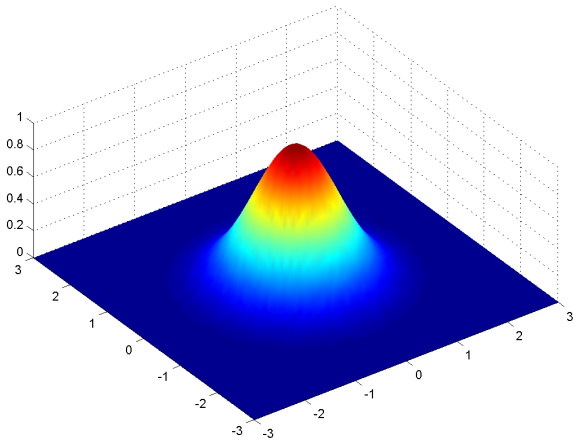

Figure 2: Gaussian Distribution.

- **SPADE Generator:** The generator receives the sampled latent vector and the semantic label map as input. It uses SPADE (Spatially-Adaptive Denormalization) ResNet blocks, which preserve spatial semantic details by modulating normalization parameters (γ and β) at every pixel, based on the input segmentation map. The generator progressively upsamples the features to produce a high-resolution image.

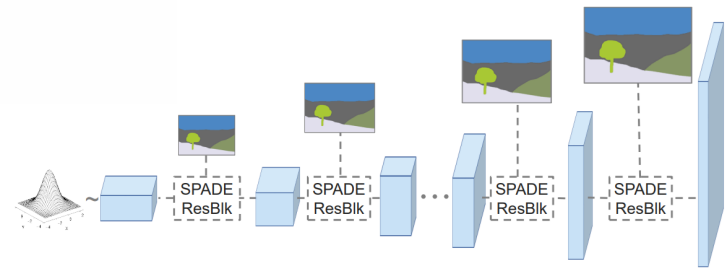

Figure 3: Architecture of the SPADE Generator.

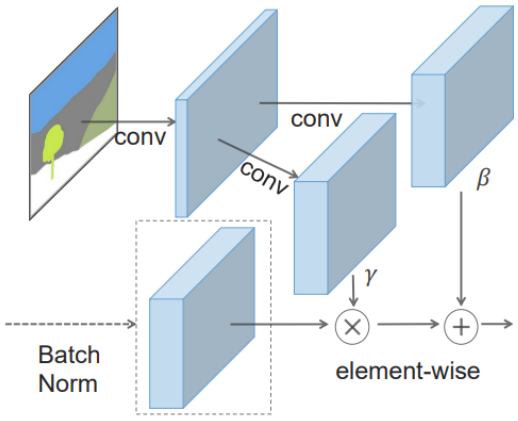

Figure 4: Feature Modulation Inside SPADE Residual Blocks.

Let $m \in \mathcal{L}^{H \times W}$ be a segmentation mask (input), where:

- $\mathcal{L}$ is the set of semantic labels (e.g., *sky*, *road*, *tree*), usually represented as integers.
- H and W are the height and width of the image (or mask).

Each element $m_{y,x}$ is an integer label denoting the class of the pixel at position (y, x). Each entry in m denotes the semantic label of a pixel.

Our target is to learn a function or mapping that takes this segmentation mask m and generates a photorealistic image that visually corresponds to the semantics of the layout (e.g., "trees should look like trees").

SPADE (Spatially-Adaptive Denormalization):
SPADE is the main innovation of the GauGAN model. It's a normalization method that does not forget spatial information from the segmentation map. Let us consider the i-th layer of the generator network.
Let h^i be the activation map (i.e., the output of that layer before normalization) for a batch of N samples. The shape of this activation map is:

$$N \times C_i \times H_i \times W_i$$

where:

- C_i: number of channels,
- H_i, W_i: height and width of the feature map at layer i.

SPADE modifies the standard normalization operation by introducing spatially-varying scale and bias. The output at a specific location (n, c, y, x) is given by:

$$\text{SPADE Output} = \gamma^i_{c,y,x}(m) \cdot \frac{h^i_{n,c,y,x} - \mu^i_c}{\sigma^i_c} + \beta^i_{c,y,x}(m)$$

where:

- $h^i_{n,c,y,x}$: The input activation at location (n, c, y, x).
- μ^i_c and σ^i_c: The mean and standard deviation of the activations across (N, H_i, W_i) for each channel c.
- $\gamma^i_{c,y,x}(m)$ and $\beta^i_{c,y,x}(m)$: Learnable scale and bias parameters computed from the segmentation map m, varying with spatial location.

Where for channel c, the mean is computed as:

$$\mu^i_c = \frac{1}{N \cdot H_i \cdot W_i} \sum_{n=1}^{N} \sum_{y=1}^{H_i} \sum_{x=1}^{W_i} h^i_{n,c,y,x}$$

The standard deviation is:

$$\sigma^i_c = \sqrt{\frac{1}{N \cdot H_i \cdot W_i} \sum_{n=1}^{N} \sum_{y=1}^{H_i} \sum_{x=1}^{W_i} \left(h^i_{n,c,y,x}\right)^2 - \left(\mu^i_c\right)^2}$$

These represent the mean and standard deviation across the batch and spatial dimensions, computed separately for each channel. So, unlike BatchNorm or InstanceNorm, SPADE does not use global scale and shift parameters. Instead, it uses per-pixel (y, x) scale and shift values, which are functions of the segmentation map m. Thus, every position (c, y, x) in the feature map has its own learned γ and β values, depending on the semantic label at that location. This allows the model to preserve and inject semantic information deep into the generator, thereby improving spatial consistency.

Calculation for γ and β:

For each layer i, the functions $\gamma^i_{c,y,x}(m)$ and $\beta^i_{c,y,x}(m)$ are implemented using a small 2-layer convolutional network:

- **Input:** A one-hot encoded or embedded version of the segmentation mask m.
- **Output:** The spatial maps γ and β over the entire (C_i, H_i, W_i).

- **PatchGAN Discriminator:** A discriminator takes the generated image along with its segmentation map and classifies it as real or fake on a patch-level basis, encouraging both global and local realism.

4 Implementation

This section describes the design of the GauGAN model, the training pipeline, and the inference procedure used to generate and evaluate images.

1. **Training Pipeline:** Training follows a structured pipeline of dataset preparation, forward passes, loss computation, and parameter updates.

   - **Step 1: Dataset Preparation**
     - **Input:** Paired RGB images, semantic segmentation masks, and label maps.
     - **Preprocessing:** It includes resizing to 288×288, random cropping to 256×256, normalization to [-1,1], and one-hot encoding of labels into different classes (21 Classes of PASCAL VOC 2012 Dataset).
   - **Step 2: Encoder → Latent Code**
     - The image encoder processes real images into a distribution defined by mean and log-variance.
     - Using the reparameterization trick, a latent vector is sampled as

       $$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1).$$

     - This latent code provides stochasticity, enabling multiple diverse outputs for the same segmentation mask.
   - **Step 3: Generator Forward Pass**
     - The generator takes two inputs: The one-hot segmentation label map, and The latent vector z from the encoder.
     - Using SPADE residual blocks and progressive upsampling, the generator outputs a synthetic RGB image (256×256).
   - **Step 4: Discriminator Forward Pass**
     - The discriminator takes as input a concatenation of the segmentation mask and an image (either real or generated).
     - It produces: Final logits for adversarial classification (real vs fake), and Intermediate feature maps at multiple scales (used in feature matching loss).
   - **Step 5: Loss Computation**
     - **Adversarial Loss (Hinge):**
       (a) **Discriminator**: distinguishes real vs fake.
       (b) **Generator**: encourages fake images to appear real.
     - **Feature Matching Loss:** Generator minimizes L1 distance between discriminator feature maps of real vs fake images.

- **VGG Perceptual Loss:** Generator minimizes difference between real and fake images in VGG19 feature space.
- **KL Divergence Loss:** Encoder regularization to keep latent space close to standard Gaussian.
- **Total Losses:**

(a) **Generator:**

$$L_G = L_{\text{adv}} + \lambda_{FM} L_{FM} + \lambda_{VGG} L_{VGG} + \lambda_{KL} L_{KL}.$$

(b) **Discriminator:**

$$L_D = \frac{1}{2}\left(L_{\text{real}} + L_{\text{fake}}\right).$$

- **Step 6: Parameter Updates**
  - **Discriminator step:** Update discriminator weights with real/fake adversarial losses.
  - **Generator + Encoder step:** Update generator and encoder with adversarial, feature matching, perceptual, and KL losses.
- **Step 7: Evaluation**
  - After each epoch: Save model checkpoints (for resuming training), and Generate sample images (mask $\rightarrow$ real vs generated comparison).
  - After training: **FID (Fréchet Inception Distance)** which evaluates realism, and **Segmentation Consistency**: **mIoU** and **pixel accuracy** from **DeepLabV3** on generated images vs input masks.

2. **Inference:** Once training is complete, the GauGAN model is used for inference, i.e., generating realistic images from unseen segmentation masks. The inference pipeline follows these steps:

- **Step 1: Inputs**

(a) **Segmentation Mask:** A semantic mask (class ID per pixel). This is converted into a one-hot label map of shape $(H, W, 21)$.

(b) **Latent Code z:** There are two options:

i. **Sample from encoder:** Pass a real image through the encoder to obtain $\mu, \log \sigma^2$. Then sample

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1).$$

ii. **Sample from Gaussian prior:** Directly sample

$$z \sim \mathcal{N}(0, I),$$

allowing new image generation without a real input image.

- **Step 2: Generator Forward Pass**
  The generator takes as input:
  - the one-hot segmentation label map, and
  - the latent vector z.

  Using SPADE residual blocks, the generator outputs a synthetic RGB image of resolution 256×256. The image is in the range $[-1, 1]$ and is rescaled back to $[0, 255]$ for visualization and saving.
- **Step 3: Output Collection**
  Generated images, real images, and label maps are saved to disk in separate folders for evaluation.
- **Step 4: Evaluation**
  - **FID**: Measures similarity between distributions of real and generated images.
  - **Segmentation Consistency (mIoU, Pixel Accuracy)**: A pretrained DeepLabV3 network predicts segmentation maps of generated images, which are compared to ground-truth label maps.

This pipeline allows both qualitative evaluation (visual inspection of generated samples) and quantitative evaluation (FID, mIoU, pixel accuracy).

Mean Intersection-over-Union (mIoU)

Definition:
Mean Intersection-over-Union (mIoU) is a standard metric for evaluating semantic segmentation. It measures the overlap between the predicted segmentation and the ground-truth segmentation for each class, and then averages across all classes.

For a given class c:

$$IoU_c = \frac{TP_c}{TP_c + FP_c + FN_c}$$

where:

- TP_c: True Positives (pixels correctly predicted as class c)
- FP_c: False Positives (pixels incorrectly predicted as class c)
- FN_c: False Negatives (pixels of class c missed by the prediction)

The mean IoU across all N classes is:

$$mIoU = \frac{1}{N} \sum_{c=1}^{N} IoU_c$$

How it is Used in This Project:

1. The generator produces a synthetic image from the segmentation mask.
2. A pretrained DeepLabV3-ResNet50 semantic segmentation model is applied to the generated image, producing a predicted segmentation map.
3. This predicted mask is compared with the ground-truth segmentation mask (the one originally given as input to the generator).
4. For each of the 21 classes in the dataset, the IoU is calculated.
5. The mean of these IoUs across all classes is reported as mIoU.

Interpretation:

- A high mIoU means the generated image respects the semantic structure of the input segmentation mask (e.g., sky looks like sky, building looks like building).
- A low mIoU indicates the generated image does not align well with the intended semantic layout.

Example: If the input mask says "sky + building + road," then:

- High mIoU $\rightarrow$ generated image clearly contains sky, building, and road in correct positions.
- Low mIoU $\rightarrow$ generated image is blurry, missing classes, or has incorrect placement.

Pixel Accuracy

Definition:

Pixel Accuracy is a widely used metric in semantic segmentation tasks. It measures the proportion of correctly classified pixels over the total number of pixels. Unlike mIoU, which evaluates overlap on a per-class basis, Pixel Accuracy provides a simpler pixel-wise correctness score.

$$\text{Pixel Accuracy} = \frac{\text{Number of Correctly Classified Pixels}}{\text{Total Number of Pixels}}$$

How it is Used in This Project:

1. The generator produces a synthetic image from the segmentation mask.
2. A pretrained DeepLabV3-ResNet50 model predicts a segmentation map for the generated image.
3. This predicted map is compared pixel-by-pixel with the ground-truth segmentation mask.
4. The ratio of correctly matched pixels to total pixels is reported as Pixel Accuracy.

Interpretation:

- A high Pixel Accuracy means that most pixels in the generated image align correctly with the input segmentation mask.
- A lower Pixel Accuracy suggests many pixels are misclassified, even if large structures appear correct.

Note: Pixel Accuracy is less strict than mIoU since it can be biased towards large classes (e.g., "sky" or "road"), while mIoU gives equal weight to all classes.

Fréchet Inception Distance (FID)

Definition:

The Fréchet Inception Distance (FID) is a widely used metric to evaluate the quality of images generated by generative models. It compares the statistical distribution of generated images with that of real images using deep features extracted from a pretrained Inception-V3 network. A lower FID indicates that the generated images are more realistic and closer to the real images.

$$FID = ||\mu_r - \mu_g||^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

where:

- μ_r, Σ_r: mean and covariance of real image features
- μ_g, Σ_g: mean and covariance of generated image features

How it is Used in This Project:

1. After training, real images and generated images are saved in separate folders.
2. The `cleanfid` library is used to extract feature representations from an Inception-V3 network for both sets of images.
3. The mean and covariance of these feature vectors are computed for real and generated images.
4. The Fréchet distance between these two distributions is calculated as the FID score.

Interpretation:

- A low FID score indicates that the distribution of generated images is very close to that of real images, meaning they look realistic.
- A high FID score means the generated images are far from real images in terms of quality and diversity.

Note: Unlike mIoU and Pixel Accuracy, which measure semantic consistency with the segmentation mask, FID measures the visual realism of generated images. Thus, the three metrics complement each other.

5 Experimental result

- **Datasets Used:** We have used two well-known datasets for semantic image synthesis:

  1. **Facades Dataset:** It contains input segmentation maps, segmentation levels corresponding to building facades.
     - Dataset Source: From Kaggle
     - Total Samples: 506 images of Building Facades of 12 classes and corresponding Segmentations of various dimensions that means (506+506).
     - Training Samples: 404 images (404+404)
     - Validation Samples: 51 images (51+51)
     - Test Samples: 51 images (51+51)
  2. **Pascal_VOC_2012 Dataset:** A more challenging dataset with 21 object classes (including background) such as person, car, dog, etc.
     - Dataset Source: From Oxford University Website
     - Total Samples: 2913 real images of various object classes such as persons, different animals, different vehicles, different indoor interiors and corresponding Segmentations, where each image has same width 500 but different heights that means (2913 + 2913).
     - Training Samples: 2330 images (2330 + 2330)
     - Validation Samples: 292 images (292 + 292)
     - Test Samples: 291 images (291 + 291)

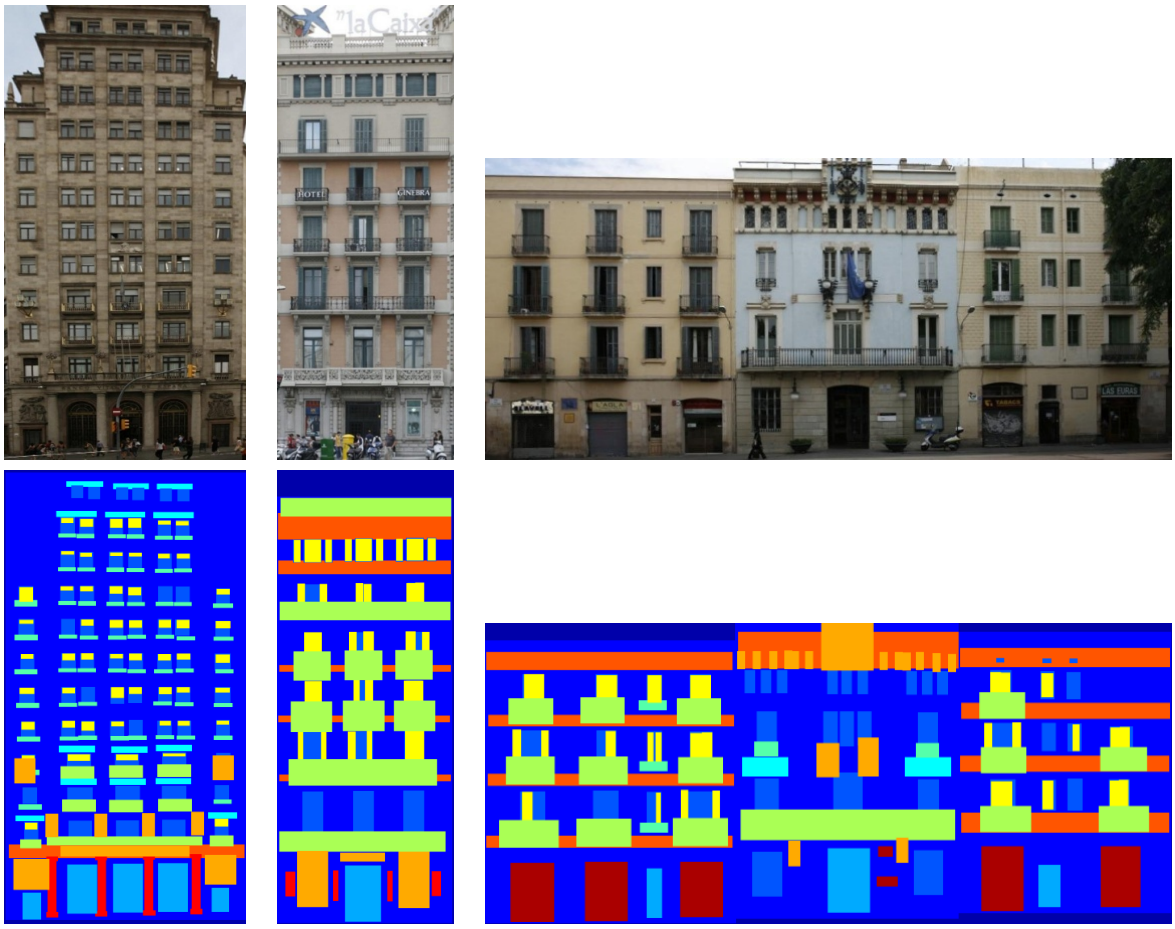

Figure 5: Sample images from the Facades dataset showing Segmentation masks corresponding Real images.

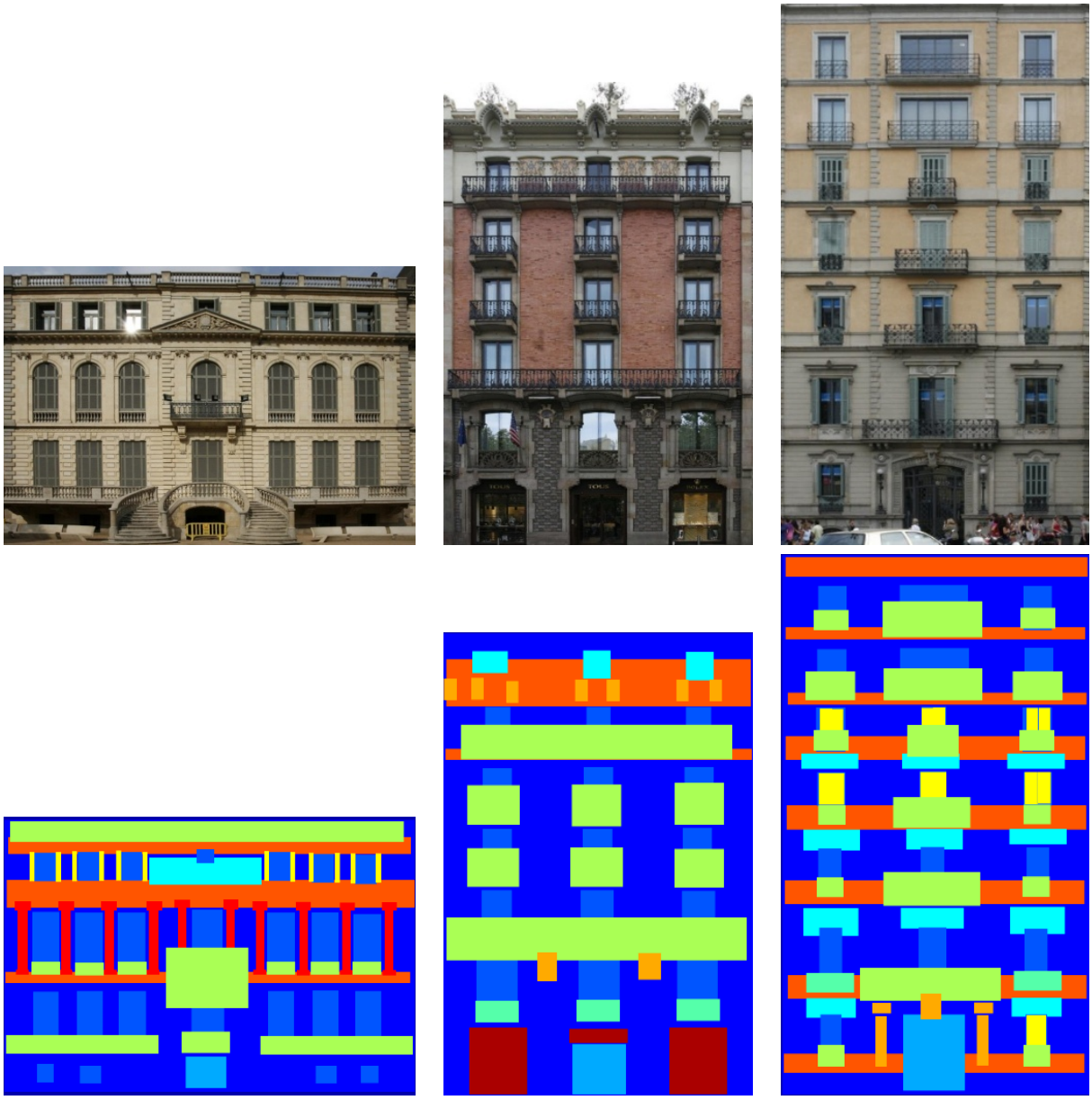

Figure 6: Sample images from the Facades dataset showing Segmentation masks corresponding Real images.

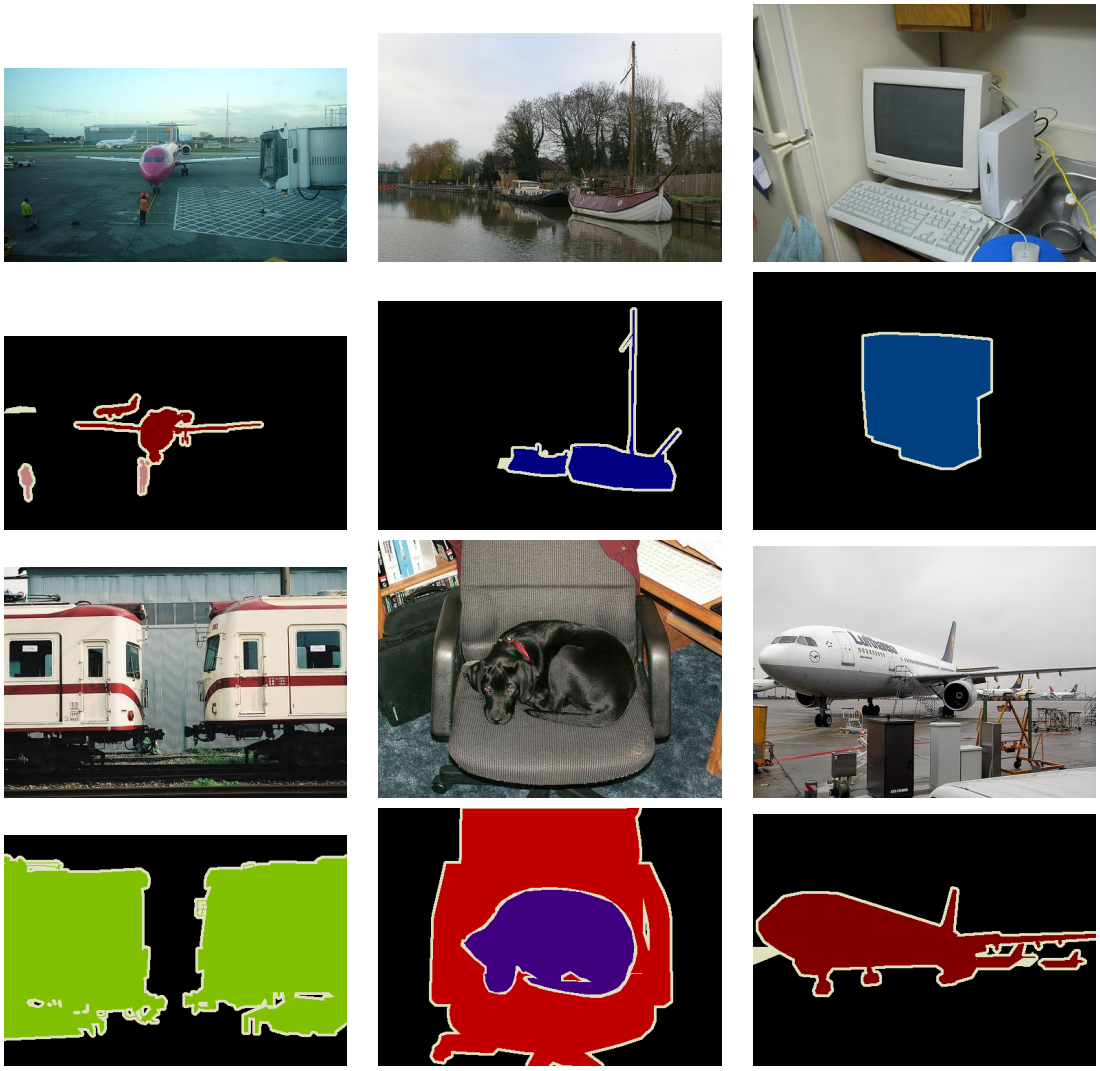

Figure 7: Sample images from the Pascal_VOC_2012 dataset showing Segmentation mask corresponding Real images.

- **Experimental settings:** For the above two datasets
  1. **Preprocessing and Augmentation:**
     - **Resizing:** All images, segmentation maps, and label maps are resized to 288×288 pixels to allow room for cropping.
     - **Cropping:** A random crop of 256×256 is extracted from each image triplet (input image, segmentation map, label map).

- **Normalization:** Pixel values for both the input image and segmentation map are normalized to the range [-1, 1].
- **One-hot Encoding:** Label maps are converted to one-hot encoded masks across 21 semantic classes. Void class (255) is masked and mapped to background (0) before encoding.

2. **Model Components:**

a. Encoder

Input: Real image (256, 256, 3)

Architecture:

- 5 downsampling blocks using strided convolutions
- Output flattened and passed to two Dense layers for:
  - Mean vector $\mu \in \mathbb{R}^{256}$
  - Log-variance vector $\sigma^2 \in \mathbb{R}^{256}$

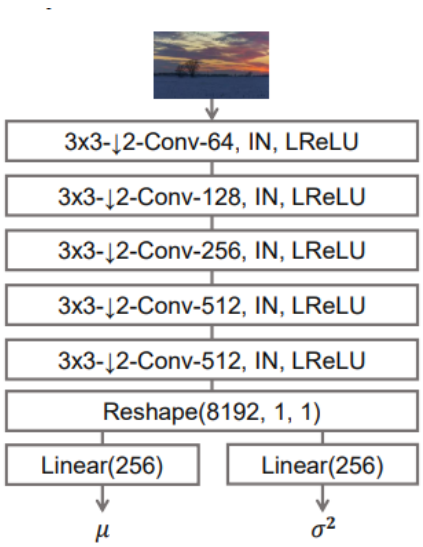

Figure 8: Architecture of Encoder.

b. Generator

Input: latent vector $z \in \mathbb{R}^{256}$ and one-hot label mask of shape (256, 256, 21)

Architecture:

- Fully connected layer reshaped to $4 \times 4 \times 1024$
- 6 upsampling layers interleaved with SPADE-ResBlocks
- Progressive feature reduction: $1024 \rightarrow 512 \rightarrow 256 \rightarrow 128$
- Final layer: Conv2D(3) + tanh activation

– Output: Generated image of shape (256, 256, 3)

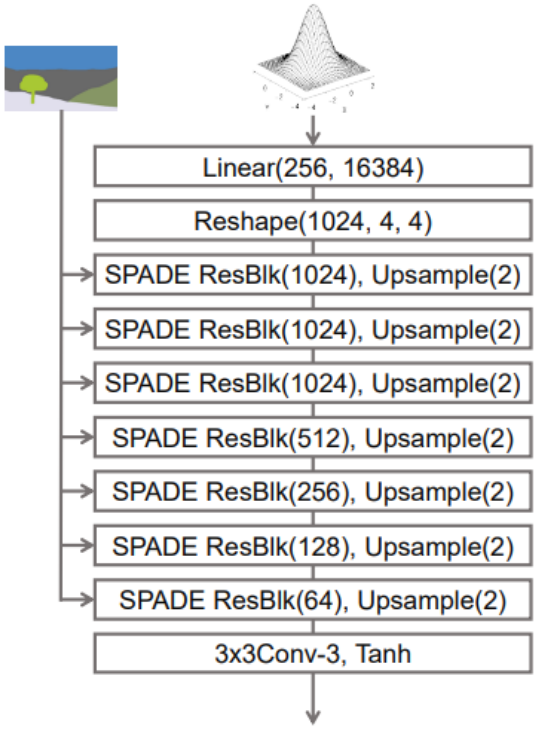

Figure 9: Architecture of Generator.

c. Sampler

Uses the reparameterization trick to sample $z = \mu + \sigma \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$

Latent dimension: 256

d. Discriminator

Input: Concatenated pair (segmentation_map, generated_image / real_image)

Architecture:

– 4 downsampling blocks using strided convolutions.

– Outputs 5 multi-scale feature maps (x1–x5).

– Final output used for adversarial training (patch-level discriminator).

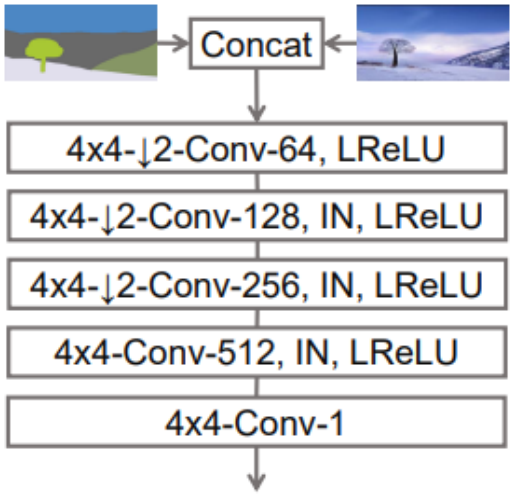

Figure 10: Architecture of Discriminator.

3. **SPADE Residual Blocks**
   - **Each residual block consists of:**
     * SPADE normalization
     * LeakyReLU activation
     * 3×3 convolution
     * Another SPADE → LeakyReLU → 3×3 convolution
     * A residual skip connection (with optional learned projection if input/output channels differ)
   - **Usage:**
     * SPADE-ResBlocks are used throughout the generator at all spatial resolutions.
     * They progressively upsample the latent feature vector while maintaining spatial semantic alignment.
   - **Activation:** LeakyReLU with $\alpha = 0.2$
   - **Normalization:** SPADE applied to each convolution layer within the block.
   - **Skip Connections:** If the number of input/output channels differs, an additional convolutional projection is applied to the skip path with its own SPADE normalization.

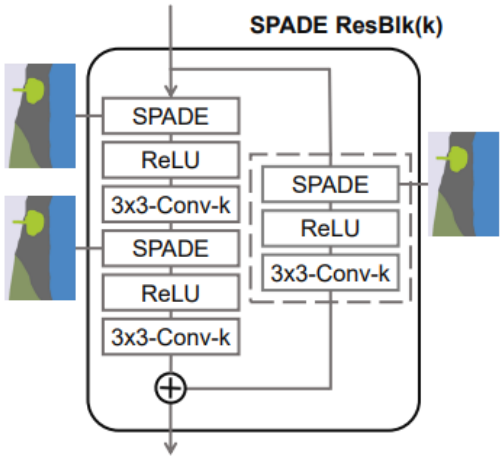

Figure 11: Architecture of SPADE Residual Block.

4. **SPADE Normalization**
   - **Input**
     - `input_tensor` of shape (H, W, C) and `raw_mask` of shape $(256, 256, \text{num\_classes})$
   - **Resize Semantic Mask**
     - Resize `raw_mask` to $(H \times W)$ using nearest-neighbor interpolation
   - **Shared Convolution for Feature Extraction**
     - `Conv2D(128, 3, padding="same", activation="relu")` $\rightarrow$ produces shared semantic features
   - **Modulation Parameter Branches**
     - **Gamma branch:** `Conv2D(filters, 3, padding="same")` $\rightarrow$ generates scale map γ
     - **Beta branch:** `Conv2D(filters, 3, padding="same")` $\rightarrow$ generates bias map β
   - **Normalization of Input Tensor**
     - Channel-wise normalization using per-channel mean and variance
   - **Apply Spatial Modulation**
     - `output` $= \gamma \cdot$ `normalized` $+ \beta$
     - (Element-wise modulation across height, width, and channels)
   - **Output**
     - Final output shape: same as `input_tensor`, now spatially modulated by semantic layout

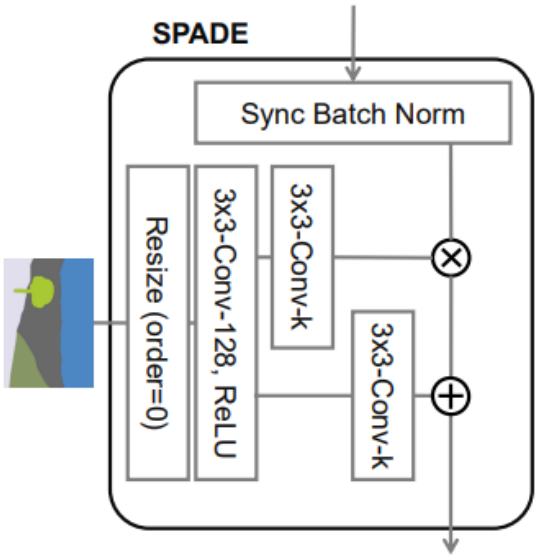

Figure 12: Architecture of SPADE Normalization.

Segmentation Labels:

- A segmentation label is a 2D array of shape (H, W), where each pixel holds a class index (e.g., 0, 1, 2, ...).
- For training, this label is converted into a 3D one-hot encoded tensor of shape $(H, W, \texttt{num\_classes})$. Example: If a pixel is labeled as class 5 in a 21-class setting, its one-hot vector becomes: `[0, 0, 0, 0, 0, 1, 0, ..., 0]`
- This one-hot tensor is used as input to the SPADE layers in the generator.
- The Generator gets the segmentation labels (the simple black-and-white map) as its input. It uses this map to know what to draw and where. Its special SPADE layers use the label for each pixel to create the correct texture and color (e.g., green for a "tree" pixel, blue for a "sky" pixel).

Role in SPADE:

- SPADE uses the one-hot encoded segmentation label to generate spatially-varying normalization parameters.
- Each pixel's class identity determines the local modulation of feature maps — enabling fine-grained semantic control.

Segmentation Map:

- A segmentation map is a visual representation of the segmentation label, where each class index is assigned a unique RGB color (e.g., red = car, green = tree).
- The Discriminator gets pairs of images. Its job is to tell the difference between a real photo and a fake one, but it's given the segmentation map (the colorful version) as a clue. This map acts as the ground truth. It helps the discriminator see if the generator not only made a realistic image, but if that image correctly matches the original layout.

So, the labels guide the Generator's creation, and the maps help the Discriminator evaluate it.

5. **Training Configuration**
   - Batch size: 4
   - Image resolution: $256 \times 256 \times 3$

- Latent dimension: 256
- Train/Validation split: 80% train / 20% validation and testing.

6. **Loss Functions and Weights**

Loss Name	Description	Weight
Adversarial Loss	Hinge loss applied to discriminator's output for real vs fake images	1.0
Feature Matching	L1 distance between intermediate discriminator activations (real vs fake)	10.0
VGG Feature Loss	L1 distance between VGG19 activations (real vs fake)	0.1
KL Divergence	KL loss between encoded latent distribution and standard normal	0.1

7. **Optimization**
   - **Optimizers:**
     * Generator: Adam (lr=1e-4, $\beta_1 = 0.0$, $\beta_2 = 0.999$)
     * Discriminator: Adam (lr=4e-4, $\beta_1 = 0.0$, $\beta_2 = 0.999$)
   - **Training Duration:** Several phases (e.g., epochs 0–99, 99–123, 123–135)
8. **Monitoring & Callbacks**
   - **GanMonitor:** Generates images at fixed epoch intervals and visualizes (mask, real, fake) triplets.
   - **Checkpointing:** Saves weights periodically and deletes older checkpoints.
   - **Logging:** Training and validation losses plotted for each component (e.g., disc/gen/VGG/KL).
9. **Evaluation Metrics**
   - **Pixel Accuracy:** Fraction of correctly predicted pixels across the entire image
   - **Mean Intersection-over-Union (mIoU):** Calculated using a confusion matrix across 21 classes for Pascal_VOC_2012 dataset and 12 classes for Facades dataset.
   - **Fréchet Inception Distance (FID):** Measures distributional similarity between real and generated images. Feature vectors extracted via InceptionV3 or tensorflow_gan.

- **Experimental results and comparison with the state-of-the-art methods**

  We evaluated our SPADE-based GauGAN model on two benchmark datasets: **PASCAL VOC** and the **Facade dataset**. The evaluation was conducted using multiple performance metrics, including:

– **Discriminator Loss** (`disc_loss`): Measures how well the discriminator distinguishes real images from generated ones using hinge loss.

$$\mathcal{L}_D = \mathbb{E}_{x \sim p_{\text{real}}} [\max(0, 1 - D(x))] + \mathbb{E}_{\hat{x} \sim p_G} [\max(0, 1 + D(\hat{x}))]$$

Where:

* $D(x)$: Discriminator output for a real image x.
* $D(\hat{x})$: Discriminator output for a fake (generated) image $\hat{x}$.

This loss encourages the discriminator to satisfy:

$$D(x) > 1 \quad \text{for real images, and} \quad D(\hat{x}) < -1 \quad \text{for fake images.}$$

– **Generator Loss** (`gen_loss`): Encourages the generator to produce images that the discriminator classifies as real.

$$\mathcal{L}_{\text{adv}} = -\mathbb{E}_{\hat{x} \sim p_G} [D(\hat{x})]$$

This loss encourages the generator to fool the discriminator by maximizing $D(\hat{x})$.

– **Feature Matching Loss** (`feat_loss`): Minimizes the difference between intermediate discriminator features of real and fake images to stabilize training.

$$\mathcal{L}_{\text{FM}} = \sum_{l=1}^{L} \mathbb{E} \left[\left\| f^{(l)}(x) - f^{(l)}(\hat{x}) \right\|_1 \right]$$

where $f^{(l)}(x)$ is the feature map at layer l of the discriminator for the real image, and $f^{(l)}(\hat{x})$ is the feature map at the same layer for the generated image.

This loss encourages the generator to match the internal representations of real images, not just to fool the discriminator. It Stabilizes training by forcing the generator to match internal discriminator features, preventing mode collapse.

– **VGG Perceptual Loss** (`vgg_loss`): Measures the L1 distance between VGG19 feature activations of real and generated images to enhance perceptual quality.

$$\mathcal{L}_{\text{VGG}} = \sum_{l=1}^{L} \mathbb{E} \left[\left\| \phi^{(l)}(x) - \phi^{(l)}(\hat{x}) \right\|_1 \right]$$

where $\phi^{(l)}(\cdot)$ denotes the output of the l-th layer of a pretrained VGG19 network. where $\phi^{(l)}$ are VGG19 feature activations (layers: block1–block5 conv1) and w_j are layer weights.

This loss compares the perceptual similarity between the generated image and the real image in the feature space. It captures style, structure, and especially textures.

– **KL Divergence Loss** (`kl_loss`): Regularizes the encoder by aligning the latent distribution with a standard normal distribution. It Improves visual quality by ensuring generated images match real ones in high-level perceptual features (textures, colors, structure).

$$\mathcal{L}_{\text{KL}} = \frac{1}{2} \sum_{i=1}^{d} \left(\mu_i^2 + \sigma_i^2 - \log(\sigma_i^2) - 1 \right)$$

where μ_i is the mean of the latent variable and $\sigma_i^2 = \exp(\log \sigma^2)$ is the variance output from the encoder.

This loss regularizes the latent space toward a unit Gaussian $\mathcal{N}(0, I)$, encouraging smoothness and continuous sampling.

After training for 580 epochs on the PASCAL VOC dataset and 15 epochs on the Facade dataset, the above loss values are shown in the below table:

1. **Facade Dataset**:

Loss Type	disc_loss	feat_loss	gen_loss	kl_loss	vgg_loss
Train Loss	0.5605	10.4694	116.8223	88.9915	16.2509
Val Loss	0.4924	10.7599	119.0715	90.7308	16.5760

Table 1: Training and Validation Losses after 15 Epochs on the Facade Dataset

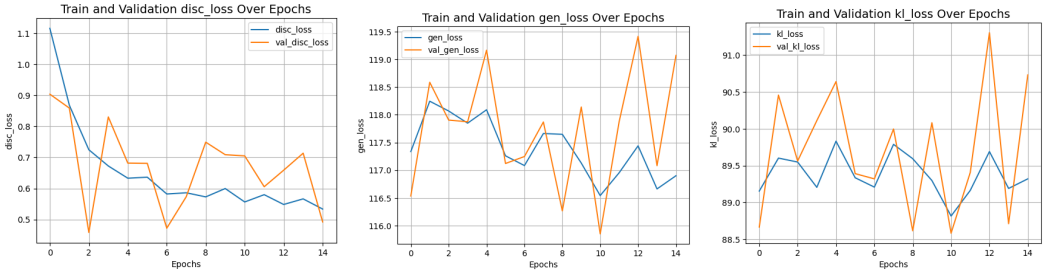

Figure 13: All Losses after 15 epochs on the Facade dataset

2. **Pascal_VOC_2012 Dataset**:

Loss Type	disc_loss	feat_loss	gen_loss	kl_loss	vgg_loss
Train Loss	0.3796	8.7047	111.596	86.5707	14.6655
Val Loss	0.4742	9.0074	113.5179	86.4467	15.4282

Table 2: Training and Validation Losses after 580 Epochs on the PASCAL VOC Dataset

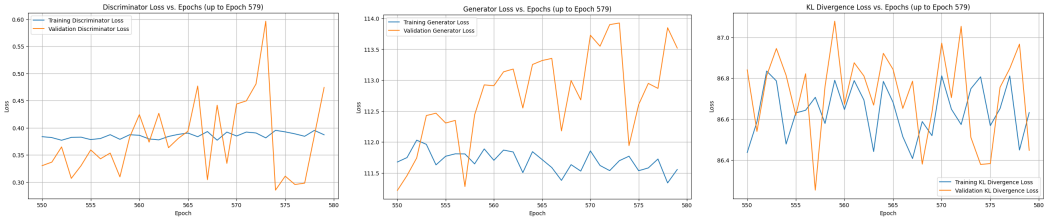

Figure 14: All Losses after 580 epochs on the Pascal_VOC dataset

We additionally compute standard image synthesis metrics for both the datasets:

- **Pixel Accuracy:** The percentage of pixels where the predicted class matches the ground truth.
- **Mean Intersection-over-Union (mIoU):** The average overlap between predicted and true segmentation masks across all semantic classes.
- **Fréchet Inception Distance (FID):** Measures the distributional distance between real and generated images using features extracted by a pretrained Inception network; lower values indicate better quality.

For the test dataset these are reported in as follows:

Dataset	FID ↓	Pixel Accuracy (%) ↑	mIoU (%) ↑
PASCAL VOC	81.14	70.44	34.8
Facade	10.54	90.8	84.8

Table 3: Quantitative evaluation results of our model on PASCAL VOC and Facade datasets. Lower FID and higher accuracy/mIoU indicate better performance.

6 Generated images during evaluation

1. **Facade Dataset:**

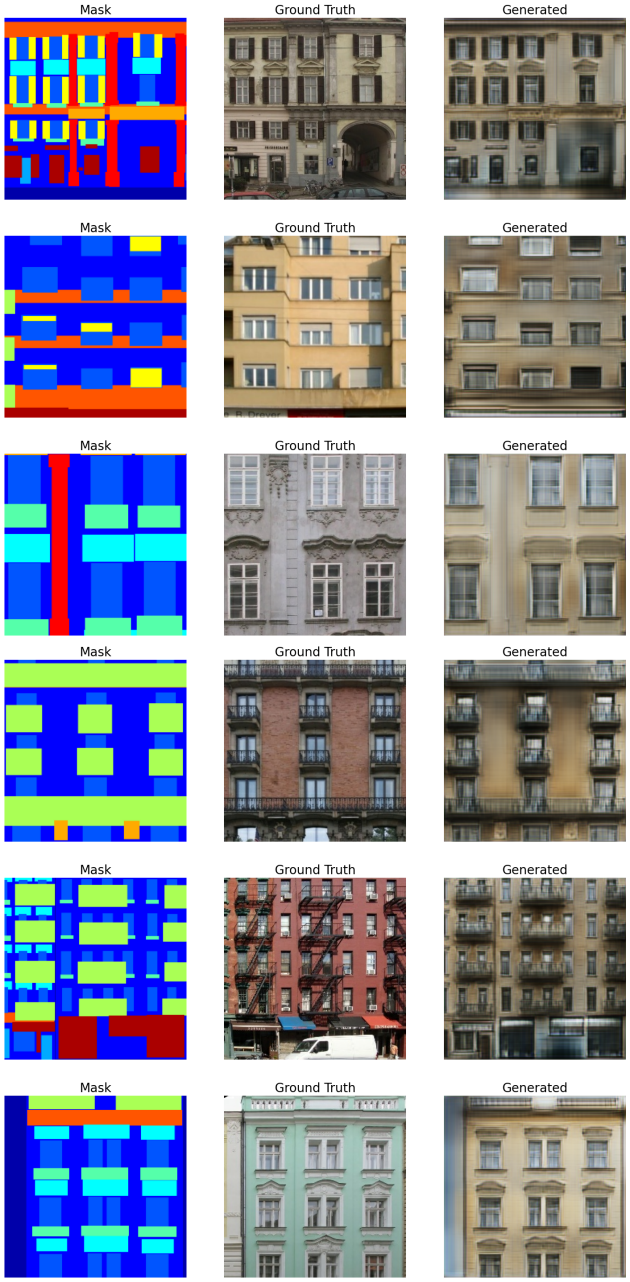

Figure 15: Generated Images for Facades Dataset.

2. **Pascal_VOC_2012 Dataset:**

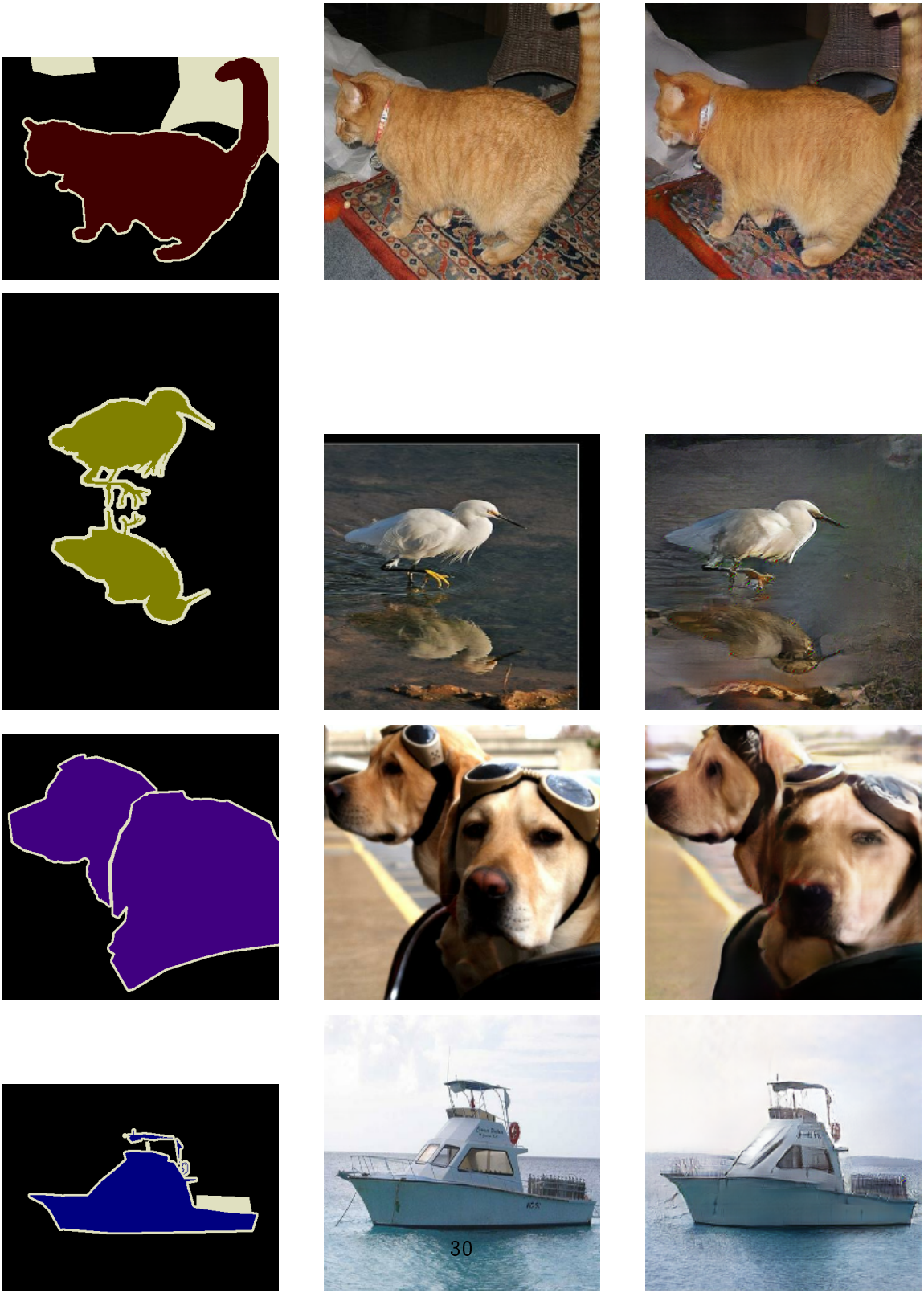

Figure 16: Generated Images for Pascal VOC Dataset.

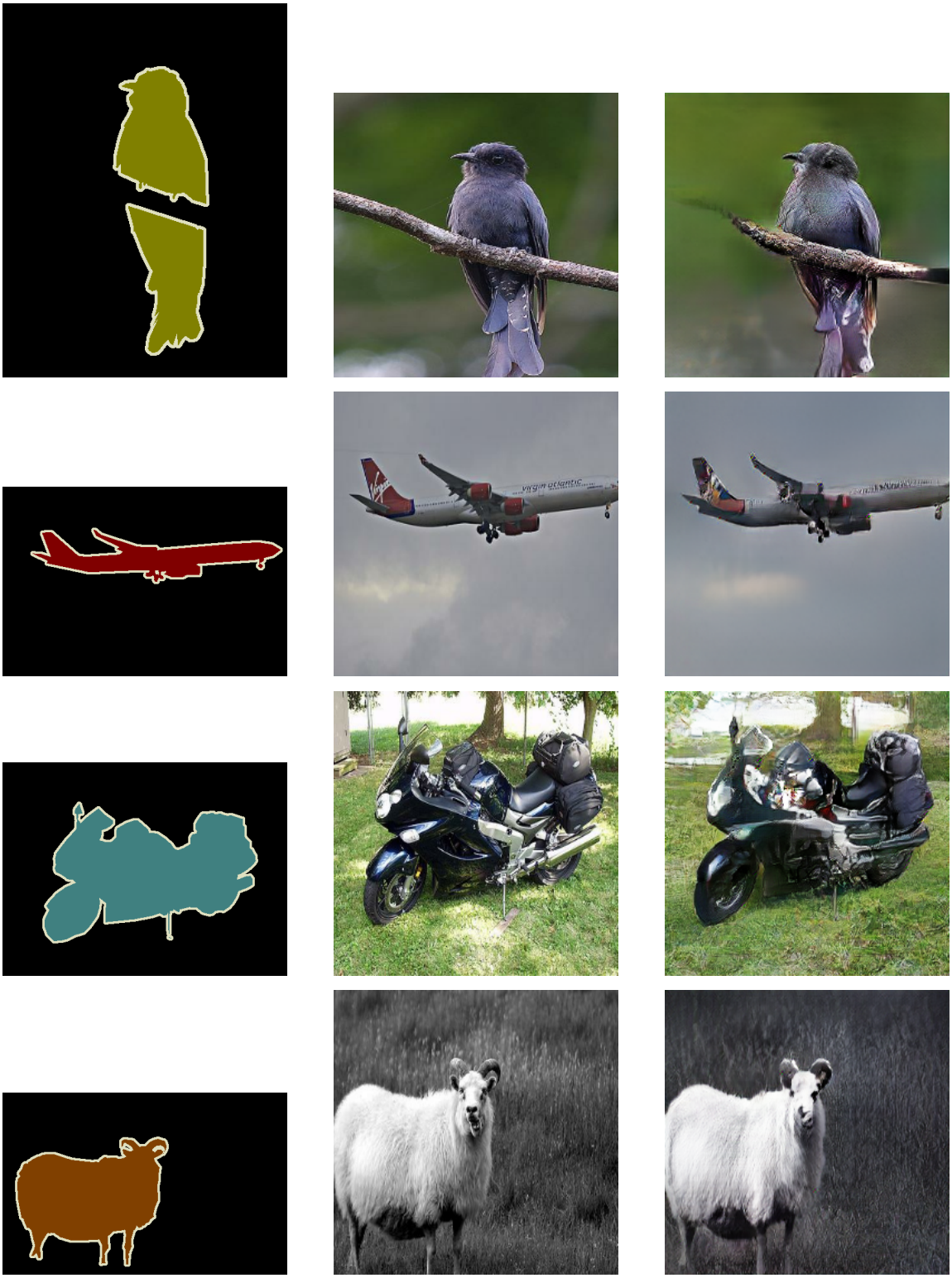

Figure 17: Generated Images for Pascal VOC Dataset.

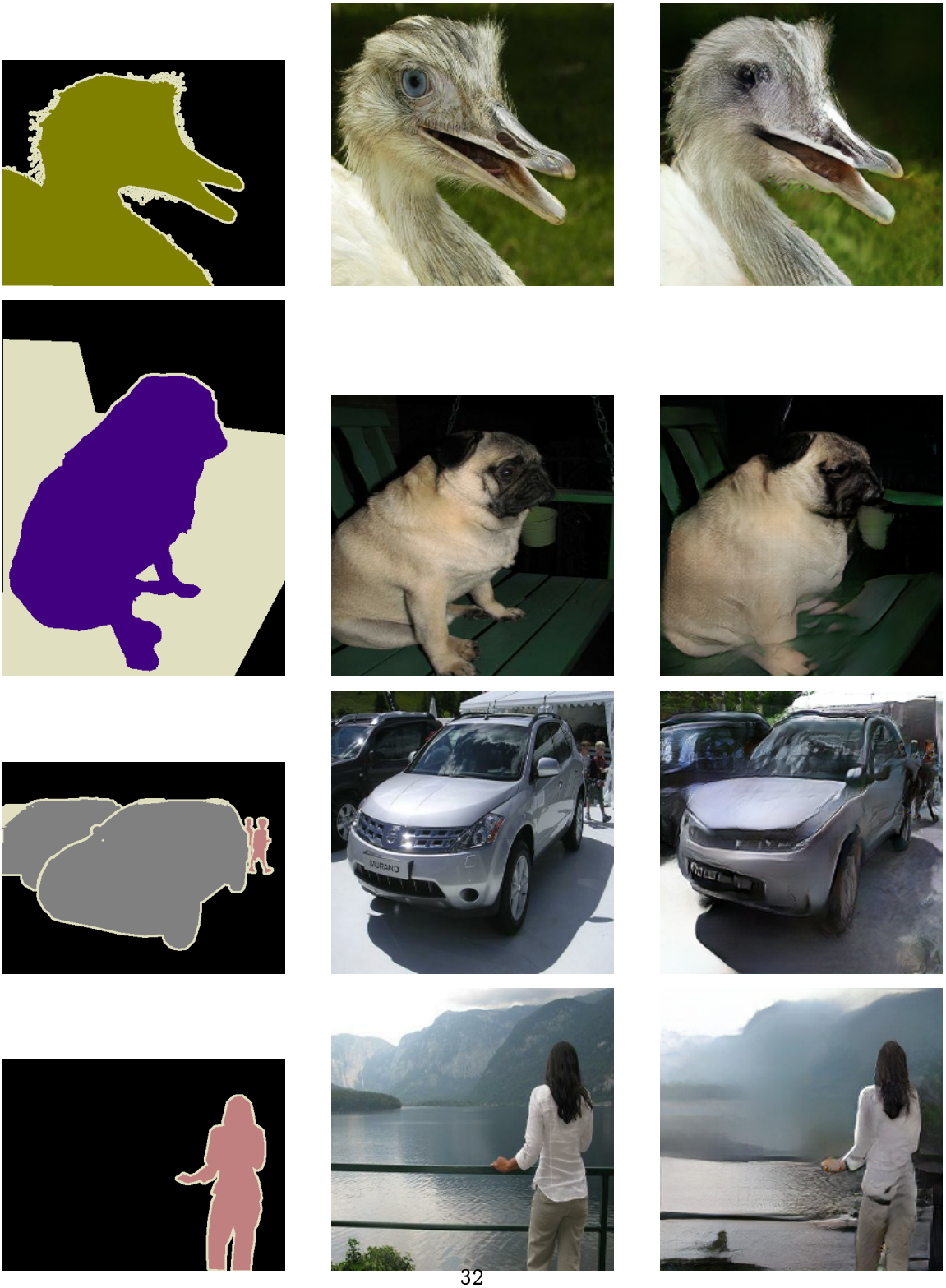

Figure 18: Generated Images for Pascal VOC Dataset.

7 Limitations

This project and the dataset used (PASCAL VOC 2012) come with several limitations that affect performance, generalization, and applicability. These can be grouped into three categories: dataset limitations, segmentation map limitations, and project/model limitations.

1. **PASCAL VOC 2012 Dataset:**
    - **Coarse Annotations**: Object boundaries are not finely labeled, limiting precision.
    - **Low Contextual Diversity**: Most images have only one or a few objects, with few complex, real-world scenes.
    - **Class Imbalance**: Frequent categories like *person* and *car* dominate, while rare ones *(sheep, sofa, cow)* are underrepresented.
2. **Segmentation Maps:**
    - **Imprecise Boundaries**: Many masks are roughly annotated, with jagged edges around objects.
    - **Incomplete Annotations**: Some small or background objects are missing from the masks.
    - **Restricted to 21 Classes**: Objects outside the defined categories are ignored or labeled as background.
    - **Imbalance in Maps**: Frequent categories appear in many masks, rare ones appear in very few.
    - **No Fine-Grained Detail**: No instance-level (per-object) or part-level (object parts) segmentation is provided.
    - **Label Noise**: Some masks include errors, ambiguities, or misclassified regions.
    - **Low Semantic Layout Diversity**: Most maps contain simple layouts with only 1–3 objects.
3. **Project / Model:**
    - **Limited Evaluation Metrics**: Uses only mIoU, Pixel Accuracy, and FID; lacks perceptual and diversity measures like LPIPS or human evaluation.
    - **Dependence on Pretrained Models**: The segmentation-based evaluation (mIoU & Pixel Accuracy) relies on a pretrained DeepLabV3-ResNet50 model. If DeepLab misclassifies generated images, the metrics will be inaccurate. Thus, the evaluation quality is only as good as the pretrained model.

- **High Computational Demand**: Multi-loss training (GAN, feature matching, VGG, KL) requires large GPU memory and compute power.
- **Generalization Limits**: Trained mainly on VOC-style masks; may not transfer well to very different domains (e.g., medical or satellite imagery).
- **Mode Collapse Risk**: As with most GANs, the generator may produce limited diversity of images.
- **KL Loss Limitation**: The KL-divergence loss used for latent regularization may sometimes lead to posterior collapse, reducing diversity in outputs. The model might ignore latent variation and generate nearly identical images for different random codes.

8 Summary

In this project, we implemented a GauGAN-based image synthesis framework enhanced with SPADE normalization, tailored for semantic image-to-image translation. We applied this model to two datasets — PASCAL VOC and Facade both of which are relatively underexplored for generative tasks. The architecture incorporated a VAE-style encoder, a ResNet-based SPADE-conditioned generator, and a multi-scale PatchGAN discriminator. Training was stabilized and guided using a combination of adversarial, feature matching, VGG perceptual, and KL divergence losses.

To ensure robust training, we designed a complete preprocessing pipeline that handled images with variable dimensions, standardized them through resizing and cropping, and transformed semantic labels into one-hot encoded maps. The training loop was custom-built using Keras subclassing, allowing full control over generator and discriminator updates.

We evaluated the model using standard generative and segmentation metrics. The model achieved a Fréchet Inception Distance (FID) of 81.14 with a pixel accuracy of 70.44% and 34.8% mIoU on the PASCAL VOC dataset, and a notably lower FID of 10.5 with 90.8% accuracy and 84.8% mIoU on the Facade dataset. These results demonstrate the model's ability to generate structurally consistent and visually realistic images that align well with the input segmentation maps.

To the best of our knowledge, this work is among the first to apply SPADE-based generative models to the PASCAL VOC dataset. The results serve as a strong baseline for future studies in semantic image synthesis on structured datasets like VOC and Facade. The experimental setup, architecture, and evaluation strategy laid out in this project can be readily extended to other domain-specific datasets as well.

References

[1] Shifeng Chen, Liangliang Cao, Yueming Wang, Jianzhuang Liu, and Xiaoou Tang. Image segmentation by map-ml estimations. *IEEE Transactions on Image Processing*, 19(9):2254–2265, 2010.

[2] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976. IEEE, 2017.

[3] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[4] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2337–2346. IEEE, 2019.