Exploring Artistic Intelligence: Transforming Images into Hand-Drawn Sketches using Hybrid Deep Learning Methods: {Sketchify}

Author: Shivang

Summary: This report surveys prior work, presents mathematical foundations and equations, outlines a hybrid architecture combining image-to-sketch deep learning modules with differentiable stroke rendering, defines loss functions and optimization strategy, proposes datasets, evaluation metrics, experiments, and practical implementation notes (training, hyperparameters, ablations). The goal: produce high-quality, controllable, hand-drawn pencil/ink sketches from photographs using a mix of classical image processing, convolutional/diffusion models, stroke-based neural rendering, and differentiable vector rasterizers.

---

1. Introduction and motivation

Translating photographs into convincing hand-drawn sketches is a challenging cross-domain image-to-image translation task. Key challenges include:

- Extracting structural contours (important for line work) while preserving salient fine detail for hatching/shading.
- Producing stroke semantics (length, pressure, direction, texture) that resemble hand drawing.
- Allowing control (line thickness, stylization strength, level of abstraction).
- Being differentiable end-to-end if we want learned stroke parameters.

A **hybrid approach** leverages: (i) classical edge/structure extractors (e.g., XDoG, Canny, HED) to provide structural priors, (ii) deep neural image-to-image modules (U-Net / Pix2Pix / GANs / diffusion models) for layout and tone translation, (iii) differentiable stroke rasterizers (DiffVG, neural renderer) to output vectorized stroke sequences, and (iv) perceptual and adversarial losses to guide realism.

2. Survey of prior/related work (high level)

- **Image-to-Image translation & GANs:** Pix2Pix, CycleGAN; conditional adversarial frameworks for translating edges/sketches ↔ photographs. Useful for learning mappings between modalities.
- **Stroke/Vector based drawing:** SketchRNN (stroke sequence generation), Differentiable vector rasterizers (DiffVG) and Neural Strokes — enable outputting vector strokes optimized by gradient descent.

- **Stroke synthesis + differentiable rendering surveys:** Stroke-based rendering surveys and differentiable drawing methods provide foundations for parameterizing strokes and rasterizing them in a trainable loop.
- **Pencil/pen style specialized works:** Im2Pencil (CVPR 2019) separates outline and shading branches to produce pencil illustration. XDoG and tonal stylization methods provide algorithmic baselines.
- **Diffusion models and image stylization:** Recent diffusion-based stylization and control networks (e.g., ControlNet, Stable Diffusion variants) show strong stylization but often produce raster strokes rather than vectorizable strokes.

(References and links: included in the companion reference list below.)

---

3. Mathematical preliminaries and core equations

Below are the standard building blocks and equations used throughout the proposed pipeline.

3.1 Image formation and representation

Images are treated as continuous functions sampled on a grid :

$$I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^c$$

where $c$is the number of channels (1 for grayscale, 3 for RGB).

Discrete pixel values: $I[i,j]$for $i = 1 \dots H, j = 1 \dots W$.

3.2 Convolutions

A convolutional layer with kernel Kcomputes:

$$(I * K)[i,j] = \sum_{u=-r}^{r} \sum_{v=-r}^{r} K[u,v] \cdot I[i-u, j-v]$$

where kernel radius $r$depends on kernel size.

Gradients for learning are computed via backprop (chain rule). Standard notations apply.

3.3 Classical edge and detail detectors

Sobel operator (approximate gradient):

$$G_x = I * S_x, G_y = I * S_y$$

where $S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_y = S_x^T$.

Gradient magnitude: $G = \sqrt{G_x^2 + G_y^2}$and orientation: $\theta = \arctan 2(G_y, G_x)$.

Canny: multi-stage: smoothing (Gaussian), gradient, non-maximum suppression, double threshold, hysteresis.

XDoG (eXtended Difference of Gaussians): produces stylized contours via thresholded DoG: for grayscale image $I$and Gaussian kernels G_σ:

$$\text{DoG}(x) = G_\sigma(I) - k\, G_{k\sigma}(I)$$

then apply a soft threshold function $S_{\tau,\phi}(x)$to emphasize lines:

$$XDoG(x) = 1 - \tanh(\phi \cdot (\text{DoG}(x) - \tau))$$

Parameters k, τ, ϕcontrol line thickness and contrast.

HED (Holistically-Nested Edge Detection): a deep CNN that predicts edges with side outputs — treated as a learnable structural prior.

3.4 Perceptual / feature losses

Let $\phi_l(\cdot)$be features extracted by layer $l$of a pretrained network (e.g., VGG19). The perceptual loss (content) between generated image $\hat{S}$and target $S$is:

$$\mathcal{L}_{perc} = \sum\nolimits_{l \in L} \lambda_l \parallel \phi_l(\hat{S}) - \phi_l(S) \parallel_2^2.$$

Style loss (Gram matrices) may be used to match stroke/textural statistics.

3.5 Adversarial losses (GAN)

For generator $G$and discriminator D, classic cGAN objective (LSGAN or BCE) — binary cross entropy minimax:

$$\min_G \max_D \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_x[\log(1 - D(x, G(x)))].$$

LSGAN variant uses least squares loss to stabilize training.

PatchGAN discriminator (operates on patches) is common in image-to-image tasks.

3.6 Total variation and smoothness

Total variation regularizer for image S:

$$TV(S) = \sum_{i,j} \sqrt{(S_{i+1,j} - S_{i,j})^2 + (S_{i,j+1} - S_{i,j})^2}$$

Used to penalize flicker and noisy outputs.

3.7 Diffusion model basics (DDPM)

Forward (noising) process for Ttimesteps:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Closed form: $q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$where $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

DDPM parameterizes a neural denoiser $\epsilon_\theta(x_t, t)$. The simplified training loss is:

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{x_0,\epsilon,t} \parallel \epsilon - \epsilon_\theta(x_t, t) \parallel_2^2.$$

Conditional diffusion (conditioning on edge maps or structural features) is achieved by concatenating conditions to network inputs or using cross-attention.

3.8 Stroke parameterization and differentiable rasterization

Parameterize a stroke (e.g., quadratic Bezier) as:

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, t \in [0,1]$$

Stroke attributes: control points P_0, P_1, P_2, width w, pressure profile $p(t)$, texture brush parameters b). A stroke rasterization operator $R(\theta)$maps stroke parameters $\theta$to pixel image: $S = R(\theta)$.

Differentiable rasterizers (DiffVG, soft rasterizers) approximate pixel occupancy with smooth functions so gradients w.r.t. stroke params exist:

$$S_{ij} = 1 - \prod_s (1 - \sigma(\alpha(d_{s,ij} - r_s)))$$

where $d_{s,ij}$is signed distance from pixel center to stroke centerline, $r_s$stroke radius, and $\sigma$is a smooth sigmoid with scale α.

Stroke regularizers may include length penalties, curvature penalties, and stroke count penalties:

$$\mathcal{R}_{stroke} = \lambda_{len} \sum_s L_s + \lambda_{curv} \sum_s \int \mid \kappa_s(t) \mid^2 dt + \lambda_{count} N_{strokes}$$

where curvature $\kappa_s$for param curve $B(t)$is standard differential geometry.

4. Proposed hybrid architecture (detailed)

We propose a three-stage hybrid pipeline that is end-to-end trainable if desired, but can also be trained stage-wise.

4.1 Stage A — Structural extractor (prior)

Input photo I→ produce structure maps:

- Edge map $E$from HED and/or XDoG: $E = \text{HED}(I)$or $E = \text{XDoG}(I)$.
- Tone map / luminance $T = L(I)$(grayscale plus bilateral smoothing for large shapes).

These maps are used as conditioning signals for the generator.

4.2 Stage B — Neural translator (layout + tone)

Conditional image-to-image network (a U-Net or conditional diffusion U-Net) that generates two outputs:

1. Outline map prediction $O = G_{outline}(I, E)$
2. Shading / tonal map $H = G_{tone}(I, E, T)$

Losses for this stage:

- $\mathcal{L}_{L1} = \| O - O_{gt} \|_1 + \| H - H_{gt} \|_1$(if paired data available)
- Perceptual loss on shading: $\mathcal{L}_{perc}(H, H_{gt})$
- Adversarial loss on combined rendered raster: discriminator sees combined image $R(\theta)$(see Stage C) to push realism.

When paired dataset is scarce, use unpaired methods (cycle consistency) or weak supervision: stylized synthetic targets via procedural pencil renderer.

4.3 Stage C — Stroke generator + differentiable rasterizer

From predicted outline $O$and tone map H, generate stroke parameter set $\Theta = S_{gen}(O, H)$. Stroke generator can be implemented as:

- A transformer/RNN that outputs sequential stroke parameters (like SketchRNN) conditioned on image features; or
- A dense predictor that outputs per-pixel stroke anchors and grouping which are then vectorized via clustering and Bezier fitting.

Rasterize $\Theta$with differentiable rasterizer $R$to produce final sketch $\hat{S} = R(\Theta)$.

Stage-level losses:

- Pixel reconstruction loss between rasterized $\hat{S}$and ground truth sketch S_{gt}: $\mathcal{L}_{pix} = \| \hat{S} - S_{gt} \|_1$.
- Perceptual loss: $\mathcal{L}_{perc}(\hat{S}, S_{gt})$.
- Stroke regularization $\mathcal{R}_{stroke}$to limit stroke count and smooth paths.
- Adversarial loss where discriminator operates on raster images to encourage artistic realism.

Optional: include a differentiable match between predicted edges and rendered strokes (e.g., Chamfer distance between binary edge sets) to align high-frequency structure.

4.4 End-to-end objective

Aggregate loss:

$$\mathcal{L} = \lambda_{pix}\mathcal{L}_{pix} + \lambda_{perc}\mathcal{L}_{perc} + \lambda_{adv}\mathcal{L}_{adv} + \lambda_{stroke}\mathcal{R}_{stroke} + \lambda_{L1}\mathcal{L}_{L1}.$$

Tune weights $\lambda_*$by cross-validation.

5. Loss functions (explicit forms)

- **Pixel L1**: $\mathcal{L}_1 = \mathbb{E}[\| \hat{S} - S \|_1]$.
- **Perceptual (VGG)**: given earlier.
- **PatchGAN adversarial**: use LSGAN objective:

$$\mathcal{L}_D = \frac{1}{2}\mathbb{E}_y[(D(y) - 1)^2] + \frac{1}{2}\mathbb{E}_x[(D(G(x)))^2] \mathcal{L}_G^{adv} = \frac{1}{2}\mathbb{E}_x[(D(G(x)) - 1)^2]$$

- **Stroke curvature penalty:** discrete approx per stroke polyline with control points P_k: