Image-to-image enhancement with NAFNet: A complete technical deep dive

Real estate photo enhancement represents a compelling application of modern image restoration techniques, transforming amateur property photographs into professionally-graded images with **HDR-like dynamic range**, corrected exposure, and cinematic color grading. This white paper presents a comprehensive technical framework for building such a system using **NAFNet (Nonlinear Activation Free Network)** and the **BasicSR** framework—a combination that achieves state-of-the-art quality while maintaining computational efficiency suitable for production deployment.

The approach outlined here leverages **577 paired before/after images at 3300×2200 resolution**, trained on **NVIDIA GB10 hardware** with 128GB unified memory. NAFNet's radical simplification of neural network design—eliminating nonlinear activation functions entirely—enables stable training, fast inference, and deployment flexibility that GAN-based or diffusion alternatives cannot match. With a target training time of **4-5 hours**, this pipeline demonstrates that professional-quality photo enhancement is achievable without enterprise-scale infrastructure.

The business case for automated real estate photo enhancement

Professional real estate photography commands premium pricing, yet most property listings feature amateur smartphone captures with poor exposure, color casts, and limited dynamic range. Manual editing in Lightroom or Photoshop requires **15-30 minutes per image** from skilled retouchers. At scale—thousands of listings daily—this creates an unsustainable bottleneck.

Automated enhancement addresses this gap by learning the transformation from amateur inputs to professional outputs directly from paired examples. Unlike preset-based approaches (which apply fixed adjustments), neural networks learn **context-aware** transformations: recognizing that kitchen lighting differs from bathroom lighting, that windows require different treatment than walls, and that wood textures demand different color grading than carpets.

The challenge lies in achieving this without the artifacts, inconsistencies, and computational overhead that plague existing approaches. GAN-based methods like Pix2Pix suffer from training instability and mode collapse. Diffusion models like InstructPix2Pix deliver impressive quality but require **3-10+ seconds per image**—impractical for batch processing thousands of listings. NAFNet offers an alternative: deterministic, fast, and remarkably simple while matching or exceeding transformer-based models on standard benchmarks.

NAFNet architecture: Simplicity as a design principle

NAFNet emerged from a systematic investigation at MEGVII Technology into what makes image restoration networks actually work. [Ecva] [ecva] The surprising conclusion, published at **ECCV 2022** by Chen et al. in "Simple Baselines for Image Restoration," was that most architectural complexity—multi-stage processing, sophisticated attention mechanisms, nonlinear activation functions—provides minimal benefit while complicating training and deployment. [ecva +2]

The U-shaped encoder-decoder foundation

NAFNet employs a single-stage U-shaped architecture with skip connections, deliberately avoiding the multi-stage complexity of networks like MPRNet:

```
Input → Intro Conv (3×3) → Encoder Stages → Middle Blocks → Decoder Stages → Ending Conv → Output
                         ↓                          ↑
                  Skip Connections ─────────────────────┘
```

The standard configuration uses **64 base channels** with encoder block counts of [2, 2, 4, 8], 12 middle blocks, and decoder block counts of [2, 2, 2, 2]—totaling 36 NAFBlocks. Each encoder level applies 2× downsampling across 4 levels, enabling the network to capture both local textures and global scene statistics essential for exposure and color correction.

SimpleGate: Multiplication replaces activation functions

The core innovation lies in **SimpleGate**, which replaces GELU, ReLU, and other activation functions with element-wise multiplication of split feature channels:

$$\text{SimpleGate}(\mathbf{X}) = \mathbf{X}_1 \odot \mathbf{X}_2$$

where $\mathbf{X}_1$ and $\mathbf{X}_2$ are obtained by splitting the input along the channel dimension. GitHub This seemingly trivial operation provides nonlinearity through the interaction of two learned linear projections—the product of two linear functions is nonlinear. Unlike sigmoid (which saturates) or ReLU (which creates dead neurons), multiplication maintains stable gradients throughout training.

The mathematical insight derives from Gated Linear Units (GLUs), where standard formulation is:

$$\text{GLU}(\mathbf{X}) = f(\mathbf{X}) \odot \sigma(g(\mathbf{X}))$$

NAFNet's contribution is demonstrating that the nonlinear activation σ can be removed entirely without performance degradation. ACM Digital Library ecva On the challenging GoPro deblurring benchmark, SimpleGate with identity function (no activation) achieves **32.85 dB PSNR**, outperforming versions with ReLU (32.59 dB), GELU (32.72 dB), and Sigmoid (32.50 dB). ecva

Simplified Channel Attention without softmax

Traditional channel attention (Squeeze-and-Excitation blocks) applies sigmoid gating:

$$\text{CA}(\mathbf{X}) = \mathbf{X} * \sigma(W_2 \cdot \text{ReLU}(W_1 \cdot \text{pool}(\mathbf{X})))$$

NAFNet's Simplified Channel Attention (SCA) removes both sigmoid and ReLU:

\text{SCA}(\mathbf{X}) = \mathbf{X} \ast (W \cdot \text{pool}(\mathbf{X}))$

ecva

This single linear projection after global average pooling captures cross-channel dependencies for scene-wide statistics—essential for white balance and exposure correction—while eliminating activation function overhead. The result: **+0.11 dB on SIDD** and **+0.50 dB on GoPro** compared to traditional channel attention. ecva

The complete NAFBlock forward pass

Each NAFBlock processes input $\mathbf{X}_{inp} \in \mathbb{R}^{C \times H \times W}$ through two sequential modules:

Spatial Processing Module:

$$\mathbf{X} = \text{LayerNorm}(\mathbf{X}_{inp})$$

$$\mathbf{X} = \text{DWConv}_{3\times3}(\text{Conv}_{1\times1}^{C\to2C}(\mathbf{X}))$$

$$\mathbf{X}_1, \mathbf{X}_2 = \text{Split}(\mathbf{X}); \quad \mathbf{X} = \mathbf{X}_1 \odot \mathbf{X}_2$$

$$\mathbf{X} = \text{Conv}_{1\times1}(\mathbf{X} * \text{SCA}(\mathbf{X}))$$

$$\mathbf{Y} = \mathbf{X}_{inp} + \beta \cdot \mathbf{X}$$

Channel Mixing Module:

$$\mathbf{X} = \text{LayerNorm}(\mathbf{Y})$$

$$\mathbf{X} = \text{Conv}_{1\times1}^{C\to2C}(\mathbf{X})$$

$$\mathbf{X}_1, \mathbf{X}_2 = \text{Split}(\mathbf{X}); \quad \mathbf{X} = \mathbf{X}_1 \odot \mathbf{X}_2$$

$$\mathbf{X}_{out} = \mathbf{Y} + \gamma \cdot \text{Conv}_{1\times1}(\mathbf{X})$$

The learnable scalars β and γ (initialized to 0) enable residual learning, while LayerNorm stabilization allows **10× higher learning rates** than networks without normalization. ecva

Why NAFNet excels for photo enhancement

For real estate enhancement specifically, NAFNet offers several advantages:

- **Linear processing preserves color relationships**: Without nonlinear activations saturating values, accurate color relationships persist throughout the network
- **Global information via SCA**: Channel attention captures scene-wide color statistics essential for white balance correction
- **Wide dynamic range handling**: Multiplication-based gating handles both bright windows and dark shadows without clipping
- **Training efficiency**: 65 GMACs (vs. 778 GMACs for MPRNet) enables training on consumer hardware ecva

---

BasicSR: The framework enabling reproducible image restoration research

BasicSR (Basic Super Restoration), developed by XPixelGroup, has become the de facto standard for image restoration research. deepwiki Projects including Real-ESRGAN, GFPGAN, NAFNet, and SwinIR build upon its infrastructure. github The framework's value lies not in algorithmic innovation but in providing battle-tested implementations of data loading, augmentation, loss computation, and distributed training.

Registry-based component architecture

BasicSR employs a registry pattern for dynamic component instantiation:

```python
from basicsr.utils.registry import ARCH_REGISTRY, MODEL_REGISTRY, LOSS_REGISTRY

@ARCH_REGISTRY.register()
class NAFNet(nn.Module):
    def __init__(self, img_channel=3, width=64, ...):
        # Architecture implementation
```

This design enables swapping architectures, losses, and datasets through configuration without code changes. The system automatically discovers registered components by scanning files with `_arch.py`, `_model.py`, and `_dataset.py` suffixes. GitHub GitHub

YAML-driven experiment configuration

All training parameters live in YAML configuration files, enabling reproducible experiments:

```yaml
name: train_NAFNet_RealEstate
model_type: ImageCleanModel
scale: 1
num_gpu: 1

datasets:
  train:
    type: PairedImageDataset
    dataroot_gt: ./datasets/real_estate/enhanced
    dataroot_lq: ./datasets/real_estate/original
    gt_size: 512
    use_hflip: true
    use_rot: true
    batch_size_per_gpu: 8

network_g:
  type: NAFNet
  width: 64
  enc_blk_nums: [2, 2, 4, 8]
  middle_blk_num: 12
  dec_blk_nums: [2, 2, 2, 2]

train:
  optim_g:
    type: AdamW
    lr: !!float 1e-3
    betas: [0.9, 0.9]
  scheduler:
    type: CosineAnnealingRestartLR
    periods: [92000]
    restart_weights: [1]
    eta_min: !!float 1e-7
```

Paired data loading with consistent augmentation

The `PairedImageDataset` class handles the critical requirement of applying identical augmentations to input-output pairs:

```python
def paired_random_crop(img_gts, img_lqs, gt_patch_size, scale):
    """Random crop maintaining LQ/GT correspondence."""
    h_lq, w_lq, _ = img_lqs[0].shape
    top = random.randint(0, h_lq - patch_size)
    left = random.randint(0, w_lq - patch_size)

    # Identical crop coordinates for both images
    img_lqs = [v[top:top+size, left:left+size, ...] for v in img_lqs]
    img_gts = [v[top:top+size, left:left+size, ...] for v in img_gts]
    return img_gts, img_lqs
```

Augmentation options include horizontal/vertical flips (50% probability each) and 90°/180°/270° rotations—applied identically to both images to maintain correspondence. GitHub For 3300×2200 images with 512×512 patches, this provides approximately **182 unique crops per image**, effectively expanding the 577-image dataset substantially.

LMDB for efficient data loading

For large datasets, BasicSR supports LMDB (Lightning Memory-Mapped Database) storage:

```yaml
io_backend:
  type: lmdb
dataroot_gt: datasets/real_estate/enhanced.lmdb
```

LMDB stores compressed PNG images in a memory-mapped format, enabling faster I/O than individual file access while maintaining CPU decompression efficiency. GitHub For 577 images at 3300×2200, this reduces data loading bottlenecks that would otherwise limit GPU utilization.

Loss functions that capture perceptual quality

The choice of loss function fundamentally shapes what the network learns. Pixel-wise losses ensure color accuracy; perceptual losses preserve structural integrity and texture. For photo enhancement, combining both proves essential.

L1 Loss: Preserving colors without blur

Mean Absolute Error provides the training foundation:

$$\mathcal{L}_{L1} = \frac{1}{HWC} \sum_{h,w,c} |y_{h,w,c} - \hat{y}_{h,w,c}|$$

L1 outperforms L2 (MSE) for image restoration because L2 minimization corresponds to Gaussian likelihood maximization—producing the "average" of all possible solutions, which manifests as blur. L1 corresponds to Laplacian distributions, producing sharper results. Medium Zhao et al. (2017) demonstrated in "Loss Functions for Image Restoration with Neural Networks" that L1 yields **0.5-1.0 dB PSNR improvement** over L2 across super-resolution, denoising, and artifact removal tasks.

Perceptual Loss: Learning from VGG feature spaces

Johnson et al. (2016) introduced perceptual loss in "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," measuring similarity in pretrained VGG feature space rather than pixel space:

$$\mathcal{L}_{feat}^{\phi,j} = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

where $\phi_j(x)$ represents VGG activations at layer j. Springer Different layers capture different information:

Layer	Captures	Typical Use
relu1_2	Fine edges, textures	Style transfer
relu2_2	Low-level structure	Super-resolution
relu3_4, relu4_4	Mid-level features	Content preservation
relu5_4	Semantic content	High-level similarity

For photo enhancement, layers relu3_4 through relu5_4 preserve scene structure while allowing pixel-level flexibility for color grading adjustments.

LPIPS: Learned similarity calibrated to human perception

Zhang et al. (2018) introduced LPIPS in "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," training perceptual weights on **484,000+ human similarity judgments**:

$$\text{LPIPS}(x, x') = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \|w_l \odot (\hat{y}_{hw}^l - \hat{y}_{hw}'^l)\|_2^2$$

The learned weights w_l calibrate each channel's contribution to perceptual similarity. LPIPS achieves **77-78% agreement** with human judgments (vs. 63% for SSIM, 63% for PSNR)—approaching the **~83% human ceiling** on the BAPPS benchmark.

Optimal loss combination for enhancement

For real estate photo enhancement, the recommended combination balances color fidelity with perceptual quality:

$$\mathcal{L}_{total} = \mathcal{L}_{L1} + \lambda_{perc} \cdot \mathcal{L}_{perceptual}$$

Starting weights of $\lambda_{perc} = 0.1$ provide good results, with tuning in the **0.01-0.5 range** depending on desired sharpness-fidelity tradeoff. Pure perceptual loss training can be unstable (VGG pooling makes it non-bijective); L1 provides the stable baseline while perceptual loss adds texture quality.

Training pipeline for paired image enhancement

Patch-based training strategy

Training on full 3300×2200 images would require prohibitive memory. Instead, random **512×512 patches** are extracted during training, with the network learning local transformations that generalize to full images during inference.

The patch size choice involves tradeoffs:

- **Smaller patches (256×256)**: More crops per image, lower memory, but limited receptive field
- **Larger patches (512×512)**: Better context for global adjustments, higher memory requirement
- **Very large patches (1024×1024)**: Best for scene-wide color grading, but limits batch size

For 577 paired images with 512×512 patches and augmentation, the effective training set contains approximately **100,000+ unique patch pairs**—sufficient for learning robust enhancement transformations.

Optimizer and learning rate configuration

AdamW (Adam with decoupled weight decay) provides stable optimization:

```yaml
optim_g:
  type: AdamW
  lr: !!float 1e-3
  betas: [0.9, 0.9]
  weight_decay: !!float 1e-4
```

NAFNet's LayerNorm enables unusually high learning rates (1e-3 vs. typical 1e-4), accelerating convergence. ecva The reduced $\beta_2 = 0.9$ (vs. default 0.999) improves stability for image restoration tasks.

Cosine annealing with warm restarts provides the learning rate schedule:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{T_{cur}}{T_{max}}\pi))$$

This smoothly decays the learning rate while periodically "restarting" to escape local minima:

```yaml
scheduler:
  type: CosineAnnealingRestartLR
  periods: [46000, 46000]
  restart_weights: [1, 0.5]
  eta_min: !!float 1e-7
```

Training iteration calculation

For a 4-5 hour training target on GB10 with 577 images:

- **Batch size**: 8-16 patches (with BF16 mixed precision)
- **Iterations per epoch**: 577 images × ~6 patches/image / batch_size ≈ 430 iterations
- **Total iterations**: ~92,000 (200+ epochs)
- **Estimated time**: ~4.5 hours at ~5.7 iterations/second

Transfer learning from pretrained weights

BasicSR supports initializing from pretrained NAFNet weights (e.g., SIDD denoising checkpoint):

```yaml
path:
  pretrain_network_g: experiments/pretrained_models/NAFNet-SIDD-width64.pth
  strict_load_g: false  # Allow architecture differences
```

Transfer learning typically reduces required training iterations by **30-50%** and improves final quality, particularly when source and target tasks share characteristics (both involve noise/exposure issues).

Inference at full resolution: Tiled processing strategies

Why tiling is essential

A 3300×2200 image with NAFNet's activations requires **2-8 GB GPU memory** depending on precision. More critically, models trained on 512×512 patches may produce artifacts when processing very different spatial dimensions. Tiled inference with overlap blending solves both problems.

Overlap blending to eliminate seams

Simple tile stitching produces visible seams at boundaries. Overlap blending with cosine-weighted falloff creates seamless results:

```python
def tiled_inference(model, image, tile_size=512, overlap=64):
    B, C, H, W = image.shape
    stride = tile_size - overlap

    output = torch.zeros_like(image)
    weight_map = torch.zeros((B, 1, H, W), device=image.device)

    # Cosine blend weights
    blend = create_cosine_blend_weight(tile_size, overlap)

    for y in range(0, H - overlap, stride):
        for x in range(0, W - overlap, stride):
            tile = extract_tile(image, y, x, tile_size)
            tile_output = model(tile)

            output[:, :, y:y+tile_size, x:x+tile_size] += tile_output * blend
            weight_map[:, :, y:y+tile_size, x:x+tile_size] += blend

    return output / (weight_map + 1e-8)
```

The **64-pixel overlap** (minimum half the model's receptive field) ensures sufficient blending region. For 3300×2200 images with 512×512 tiles, this produces approximately **42 tiles** per image.

Memory management for batch processing

For production throughput, process tiles in batches:

python

```python
# Collect all tiles first
tiles = []
positions = []
for y, x in tile_positions:
    tiles.append(extract_tile(image, y, x, tile_size))
    positions.append((y, x))

# Batch process (8 tiles at once)
for i in range(0, len(tiles), batch_size):
    batch = torch.stack(tiles[i:i+batch_size])
    with torch.no_grad(), torch.cuda.amp.autocast():
        outputs = model(batch)
    # Accumulate with blending
```

This maximizes GPU utilization while maintaining reasonable memory consumption.

NVIDIA GB10: Unified memory for high-resolution workflows

Grace Blackwell architecture specifications

The **NVIDIA GB10 Superchip** combines a Grace CPU (20 ARM cores) with a Blackwell GPU (6,144 CUDA cores) connected via **NVLink-C2C at 600 GB/s**—7× faster than PCIe Gen 5. The defining feature is **128GB unified LPDDR5X memory** shared between CPU and GPU.

Specification	GB10	A100 80GB	H100 80GB
Memory	128GB unified	80GB HBM2e	80GB HBM3
Memory Bandwidth	273-301 GB/s	2.0 TB/s	3.35 TB/s
FP32 TFLOPS	31	19.5	51
TDP	140W	400W	700W
Price	~$3,000-4,000	Cloud rental	Cloud rental

Unified memory advantages for image enhancement

Traditional discrete GPUs require explicit memory transfers between CPU and GPU. NVIDIA Developer A 3300×2200 image at FP32 occupies **~87MB**; with batch size 8 and intermediate activations, total memory easily exceeds 10GB. On systems with 24GB GPUs, this necessitates gradient checkpointing or reduced batch sizes.

GB10's unified memory eliminates these constraints:

- **No cudaMemcpy overhead**: Data resides in single address space (NVIDIA Developer)
- **Larger effective batch sizes**: 128GB accommodates full training state
- **Simplified code**: No explicit memory management between host/device (NVIDIA Developer)
- **High-resolution direct processing**: Load full 3300×2200 images without streaming

For the target 4-5 hour training with 577 images, GB10 enables batch size **8-16 with BF16 mixed precision**—impossible on consumer GPUs without extensive memory optimization.

Optimal training configuration for GB10

python

```python
# BF16 mixed precision (recommended for Blackwell)
from torch.cuda.amp import autocast, GradScaler

scaler = GradScaler()
for images, targets in dataloader:
    optimizer.zero_grad()
    with autocast(dtype=torch.bfloat16):  # BF16 more stable than FP16
        outputs = model(images)
        loss = criterion(outputs, targets)
    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()
```

BF16 (bfloat16) maintains FP32's dynamic range via 8-bit exponent while halving memory—more stable than FP16, which requires careful loss scaling. GB10's 5th-generation Tensor Cores accelerate both BF16 and FP16 operations. (NVIDIA Newsroom)

Evaluation metrics: Measuring enhancement quality

PSNR: Necessary but insufficient

Peak Signal-to-Noise Ratio provides a baseline quality metric:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{\text{MSE}}\right) = 20 \cdot \log_{10}\left(\frac{255}{\sqrt{\text{MSE}}}\right)$$

For 8-bit images, **35-40 dB** indicates good quality, **>40 dB** excellent. However, PSNR poorly correlates with human perception—two images with identical PSNR can have vastly different visual quality. Enhancement tasks that correctly improve exposure may actually *decrease* PSNR against original targets while appearing visually superior.

SSIM: Structural similarity

Wang et al. (2004) introduced SSIM to address PSNR's limitations:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

The three components—luminance, contrast, and structure comparison—model human visual system characteristics. SSIM ranges from 0 to 1 (higher is better), with **>0.95** indicating excellent structural preservation.

For enhancement tasks, MS-SSIM (multi-scale SSIM) better handles varying viewing conditions by computing similarity across multiple downsampled scales.

LPIPS: The perceptual gold standard

For photo enhancement evaluation, **LPIPS provides the most reliable quality assessment**. Lower scores indicate better perceptual similarity:

LPIPS Value	Interpretation
0.00-0.10	Nearly imperceptible difference
0.10-0.20	Minor perceptual difference
0.20-0.40	Noticeable difference
>0.40	Significant perceptual difference

A well-trained enhancement model should achieve LPIPS **<0.15** on held-out test images while maintaining PSNR **>30 dB** and SSIM **>0.90**.

Production deployment: From PyTorch to optimized inference

ONNX export for portability

Convert trained NAFNet to ONNX with dynamic shape support:

python

```
torch.onnx.export(
    model,
    dummy_input,
    "nafnet_enhance.onnx",
    opset_version=17,
    input_names=['input'],
    output_names=['output'],
    dynamic_axes={
        'input': {0: 'batch', 2: 'height', 3: 'width'},
        'output': {0: 'batch', 2: 'height', 3: 'width'}
    }
)
```

Dynamic axes enable processing arbitrary resolutions without recompilation—essential for real estate images with varying aspect ratios.

TensorRT optimization for 2-3× speedup

TensorRT performs layer fusion (Conv+BN+ReLU → single kernel), kernel auto-tuning, and precision optimization: Ultralytics

python

```
config = builder.create_builder_config()
config.set_flag(trt.BuilderFlag.FP16)  # Enable FP16 for 2× speedup

# Optimization profiles for common resolutions
profile = builder.create_optimization_profile()
profile.set_shape('input',
    min=(1, 3, 512, 512),
    opt=(1, 3, 1080, 1920),
    max=(4, 3, 2200, 3300)
)
```

Expected speedups for NAFNet: **2-3× with FP16**, with negligible quality loss. INT8 quantization provides **3-4× speedup** but requires calibration data and careful validation for enhancement tasks where color accuracy matters.

API deployment patterns

For production serving, FastAPI with batched inference maximizes throughput:

```python
@app.post("/enhance")
async def enhance_image(file: UploadFile):
    contents = await file.read()
    result = await batch_processor.process(preprocess(contents))
    return StreamingResponse(encode_png(result), media_type="image/png")
```

The `batch_processor` collects requests over a **50ms window**, processes them as a single GPU batch, then returns individual results. This achieves **80-90% GPU utilization** versus **30-40%** for individual request processing.

Why NAFNet over alternative architectures

Versus transformer-based methods (SwinIR, Restormer)

SwinIR (Liang et al., ICCVW 2021) achieves excellent super-resolution quality via Swin Transformer blocks, but requires **~50 GMACs** with **500ms-1s inference** per image. **Restormer** (Zamir et al., CVPR 2022) uses transposed attention for linear complexity, achieving state-of-the-art on deblurring with **~140 GMACs**.

NAFNet matches or exceeds both with **~16-65 GMACs** depending on width configuration—achieving **33.69 dB** on GoPro deblurring (vs. Restormer's 32.92 dB) at **8.4% of the computational cost**. ecva For high-volume real estate processing, this efficiency difference is decisive.

Versus GAN-based methods (Pix2Pix, Pix2PixHD)

Pix2Pix (Isola et al., CVPR 2017) and Pix2PixHD (Wang et al., CVPR 2018) learn arbitrary image-to-image mappings via adversarial training. TheCVF While capable of dramatic style transformations, they suffer from:

- **Training instability**: Generator-discriminator balance is fragile
- **Mode collapse**: Limited output diversity
- **Artifacts**: Checkerboard patterns, hallucinated textures
- **Non-deterministic**: Same input produces different outputs

For professional real estate enhancement requiring **consistent, artifact-free results**, GANs introduce unacceptable unpredictability.

Versus diffusion models (InstructPix2Pix)

InstructPix2Pix (Brooks et al., CVPR 2023) enables natural language-guided editing with impressive quality, but requires **3-10+ seconds per image** on high-end GPUs. For batch processing thousands of listings, this is impractical. Additionally, diffusion models may hallucinate details—problematic for real estate where accurate representation matters legally.

Architecture	Inference Time	Training Stability	Deterministic	Memory
NAFNet	50-100ms	★★★★★	Yes	~1-2GB
SwinIR	500ms-1s	★★★★	Yes	~3-4GB
Restormer	200-400ms	★★★★	Yes	~4-6GB
Pix2PixHD	500ms+	★★	No	~4-8GB
InstructPix2Pix	3-10s+	★★★	No	~6-10GB

Conclusion: A practical path to production enhancement

Building a production real estate photo enhancement system requires balancing quality, speed, and deployment complexity. The NAFNet + BasicSR combination achieves this balance through radical simplification: removing nonlinear activations (SimpleGate), simplifying attention (SCA), and leveraging a mature training framework.

Key implementation takeaways:

- **Architecture**: NAFNet-width64 with 36 blocks provides optimal quality/speed tradeoff
- **Training**: L1 + perceptual loss (λ=0.1), AdamW at 1e-3 learning rate, cosine annealing schedule
- **Data**: 512×512 patches with flip/rotation augmentation, LMDB storage for efficiency
- **Hardware**: GB10's 128GB unified memory enables larger batch sizes and simpler memory management
- **Deployment**: ONNX export → TensorRT FP16 optimization → tiled inference with overlap blending

The 577 paired images at 3300×2200 resolution, combined with patch-based training and augmentation, provide sufficient data for learning robust enhancement transformations. The **4-5 hour training target** is achievable on GB10 with proper configuration.

For production scaling, TensorRT-optimized inference with request batching achieves **2-5 images/second** per GPU—sufficient for processing thousands of listings daily. The deterministic, artifact-free output meets professional standards while eliminating manual editing bottlenecks.

The path from amateur real estate photos to professional-quality enhanced images is now computationally tractable, economically viable, and technically robust. NAFNet's "less is more" philosophy—that simpler networks can outperform complex ones—provides not just a working solution but a maintainable one, deployable across diverse hardware from edge devices to cloud infrastructure.

Key references

NAFNet: Chen, L., Chu, X., Zhang, X., & Sun, J. (2022). "Simple Baselines for Image Restoration." *European Conference on Computer Vision (ECCV)*. Springer.

BasicSR: Wang, X., Xie, L., Yu, K., Chan, K.C.K., Loy, C.C., & Dong, C. (2022). "BasicSR: Open Source Image and Video Restoration Toolbox." https://github.com/XPixelGroup/BasicSR

Perceptual Loss: Johnson, J., Alahi, A., & Fei-Fei, L. (2016). "Perceptual Losses for Real-Time Style Transfer and Super-Resolution." *European Conference on Computer Vision (ECCV)*.

LPIPS: Zhang, R., Isola, P., Efros, A.A., Shechtman, E., & Wang, O. (2018). "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

SSIM: Wang, Z., Bovik, A.C., Sheikh, H.R., & Simoncelli, E.P. (2004). "Image Quality Assessment: From Error Visibility to Structural Similarity." *IEEE Transactions on Image Processing*, 13(4), 600-612.

Loss Functions: Zhao, H., Gallo, O., Frosio, I., & Kautz, J. (2017). "Loss Functions for Image Restoration with Neural Networks." *IEEE Transactions on Computational Imaging*, 3(1), 47-57.

SwinIR: Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., & Timofte, R. (2021). "SwinIR: Image Restoration Using Swin Transformer." *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*.

Restormer: Zamir, S.W., Arora, A., Khan, S., Hayat, M., Khan, F.S., & Yang, M.-H. (2022). "Restormer: Efficient Transformer for High-Resolution Image Restoration." *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Pix2Pix: Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A.A. (2017). "Image-to-Image Translation with Conditional Adversarial Networks." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

InstructPix2Pix: Brooks, T., Holynski, A., & Efros, A.A. (2023). "InstructPix2Pix: Learning to Follow Image Editing Instructions." *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.