Individual Final Report: GAN-Based Image Colorization with Fine-tuning

Sai Rachana Kandikattu

Professor: Amir Jafari

Course: Deep Learning (DATS 6303)

Institution: GWU

1. Introduction

Image colorization, the task of adding plausible color information to grayscale images, is a fundamental challenge in computer vision. This project explores automatic colorization using deep learning approaches, specifically comparing a baseline pretrained model (ECCV16 by Zhang et al.) with an enhanced GAN-based pipeline.

1.1 Project Overview:

Our team developed a comprehensive colorization system that:

- Reproduces the ECCV16 pretrained model as a baseline
- Implements an improved colorization pipeline using Generative Adversarial Networks (GANs)
- Evaluates both approaches using quantitative metrics and qualitative visual assessment

1.2 Outline of Shared Work:

The project was divided into 3 main components:

- Data preprocessing pipeline and ECCV16 Baseline Model (Snehita): Converting RGB images to CIELAB color space, extracting L and ab channels, and preparing tensors for model input
- Hybrid GAN with fine-tuning and adversarial training (Sai Rachana)
- Evaluation and Streamlit (Naiska)

1.3 My Individual Contribution:

My specific contribution focuses on the GAN + Fine-tuned model development, which includes:

- Fine-tuning the pretrained ECCV16 model on our ImageNet_50 dataset using supervised learning
- Implementing a PatchGAN discriminator architecture for adversarial training

- Integrating VGG19-based perceptual loss to preserve semantic content
- Designing and implementing the complete GAN training pipeline with combined loss functions
- Training the model for 50 epochs and monitoring convergence

This report details the development, implementation, training process, and results of the GAN-based colorization system that I developed.

---

2. Background and Algorithm Development

2.1 Motivation for GAN-Based Colorization

While the ECCV16 baseline model provides semantically reasonable colorizations, it tends to produce conservative, desaturated outputs. This limitation is inherent to supervised learning approaches that minimize pixel-wise losses, the model learns to predict the mean of the color distribution, resulting in safe but bland colorizations for our specific dataset

We offer a Hyrbid Generative Adversarial Networks (GANs) with an adversarial objective that encourages the generator to produce outputs indistinguishable from real images. This adversarial pressure pushes the model toward more realistic, saturated, and diverse colorizations for our dataset.

2.2 Conditional GANs for Image-to-Image Translation

My approach builds on conditional GANs (cGANs), where both the generator and discriminator are conditioned on input data. In our colorization task:

- Input condition: Grayscale luminance channel L
- Generator task: Produce chrominance channels (a, b) given L
- Discriminator task: Distinguish real [L, ab] pairs from synthetic [L, G(L)] pairs

The conditional GAN framework is formulated as a min-max game:

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

where:

- x = input L channel (grayscale)
- y = ground-truth ab channels (color)
- G(x) = predicted ab channels from generator
- D(x, y) = discriminator's evaluation of whether (x, y) is real or fake

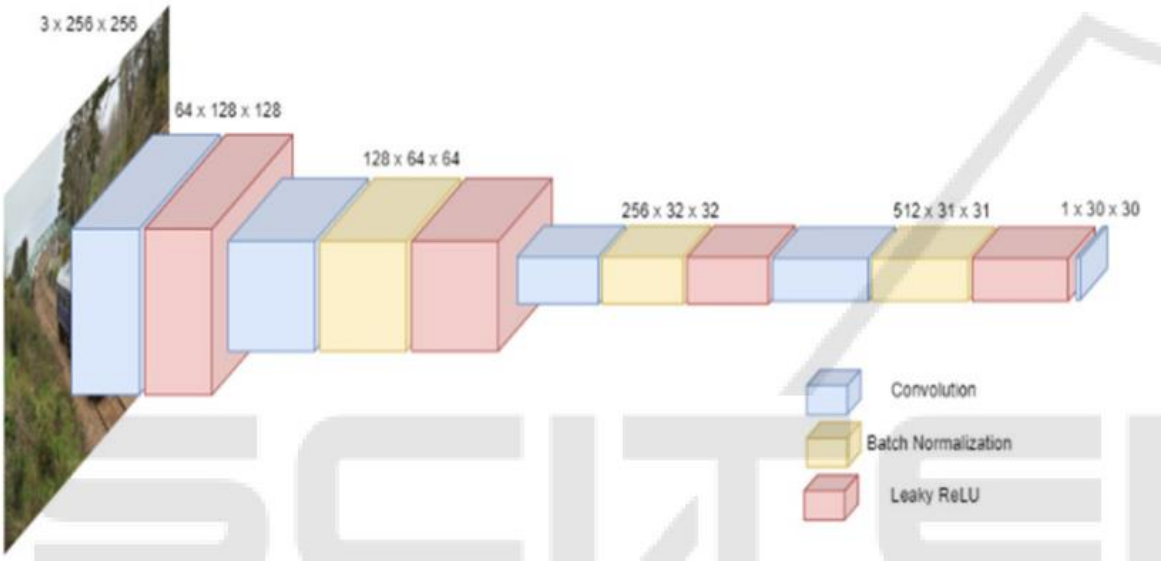
2.3 PatchGAN Discriminator

Rather than classifying the entire image with a single scalar output, I employed a PatchGAN discriminator inspired by the Pix2Pix framework.

Key advantages:

1. Local evaluation: Classifies whether each N×N(70x70) patch is real or fake independently
2. Fine-grained spatial feedback: Provides detailed information about which regions need improvement
3. Parameter efficiency: Fewer parameters than full-image discriminators
4. Texture realism: Particularly effective for high-frequency detail and local consistency

Architecture: The PatchGAN consists of five convolutional blocks that progressively downsample the input:



Output: Patch predictions → (1, 30, 30). Each element in the 30×30 output corresponds to a 70×70 receptive field in the input, effectively classifying overlapping patches as real or fake.

2.4 Two-Stage Training Strategy

I implemented a two-stage training approach:

Stage 1: Supervised Fine-tuning

- Initialize with pretrained ECCV16 weights (trained on 1.3M ImageNet images)
- Fine-tune on ImageNet_50 dataset using MSE loss
- Train for 3 epochs with learning rate 0.0001

- Purpose: Adapt the model to our specific data distribution while maintaining semantic understanding

Stage 2: Adversarial GAN Training

- Use fine-tuned model as generator initialization
- Train PatchGAN discriminator from scratch
- Optimize with combined adversarial, L1, and perceptual losses
- Train for 50 epochs with alternating optimization
- Purpose: Enhance color vibrancy, realism, and texture detail through adversarial feedback

2.5 Loss Function Design

The generator optimization uses three loss components:

a) Adversarial Loss (L_GAN): Encourages generator to fool the discriminator into classifying fake images as real.

$$L_{gan} = -log(D(L, G(L)))$$

b) L1 Reconstruction Loss (L_L1): Ensures pixel-wise color accuracy and structural preservation

$$L_{L1} = ||ab_{real} - G(L)||_1$$

c) Perceptual Loss (L_perceptual): Preserves semantic content using VGG19 relu4,3 layer features.

$$L_{perceptual} = ||VGG(ab_{real}) - VGG(G(L))||_1$$

d) Combined Generator Loss:

$$L_G = L_{GAN} + \lambda_{L1} * L_{L1} + \lambda_{perceptual} * L_{perceptual}$$

e) Discriminator Loss: Trained to correctly classify real pairs as real(1) and fake pairs as fake(0)

$$L_D = 0.5 * [BCE(D(G(x), 0) + BCE(D(x), 1)]$$

f) Training alternates between: Network optimization involves alternating between conducting a single gradient descent step on the discriminator and another step on the generator.

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))]$$

- One gradient step on discriminator (maximize L_D)

$$\mathbb{E}_{x,z}\left[\log(1 - D(x, G(x, z)))\right]$$

- One gradient step on generator (minimize L_G)

$$\mathbb{E}_{x,y}\left[\log D(x, y)\right]$$

---

3. Detailed Description of Individual Work

3.1 Stage 1: Fine-tuning Implementation

What I Did: I implemented the supervised fine-tuning of the pretrained ECCV16 model to adapt it to our ImageNet_50 dataset.

Process:

1. Loaded pretrained ECCV16 weights
2. Set all layers to trainable (no frozen layers)
3. Configured training loop with MSE loss
4. Trained for 3 epochs with batch size 16

Algorithm:

1. model ← ECCV16 with pretrained weights
   optimizer ← Adam(lr=0.0001)
   loss_fn ← MSELoss()
2. For each epoch (1 to 3):
   For each batch (L, ab_real) in dataloader:
   ab_pred ← model(L)
   loss ← MSE(ab_pred, ab_real)
   Backpropagate and update weights
3. Save checkpoint
4. Output: Fine-tuned generator for Stage 2
5. Hyperparameters I configured:
   - Learning rate: 0.0001
   - Optimizer: Adam (β1=0.9, β2=0.999)
   - Epochs: 3
   - Batch size: 16

Why MSE Loss: The original ECCV16 uses classification loss, but since the pretrained model already outputs continuous ab values, direct MSE provides simpler optimization for fine-tuning.

3.2 PatchGAN Discriminator Implementation

What I Did: I implemented the complete PatchGAN discriminator architecture from scratch following the Pix2Pix design.

Architecture Details:

- 5 convolutional blocks with stride-2 downsampling
- No BatchNorm in first layer (standard practice for discriminator stability)
- LeakyReLU(0.2) activation throughout
- Final output: 30×30 patch classifications

Key Implementation Decisions:

1. Input format: Concatenate [L, ab] channels → shape (3, 256, 256)
2. No sigmoid: Used BCEWithLogitsLoss for numerical stability
3. Receptive field: Each output pixel sees 70×70 input region

3.3 Perceptual Loss Integration

What I Did: I implemented VGG19-based perceptual loss to preserve semantic structure during colorization.

Algorithm: Perceptual Loss Computation

1. Input: L (grayscale), ab_pred, ab_real
2. VGG ← Pretrained VGG19 (frozen, up to relu4_3)

// Convert LAB to RGB for VGG

3. rgb_pred ← LAB_to_RGB(L, ab_pred)
4. rgb_real ← LAB_to_RGB(L, ab_real)

// Extract features

5. features_pred ← VGG(rgb_pred)
6. features_real ← VGG(rgb_real)

// Compute L1 distance

7. loss ← $||features\_pred - features\_real||_1$
8. Output: Perceptual loss value

We chose relu 4_3 layer as it captures mid-to-high level semantic features, balances texture detail and semantic content, standard choice in perceptual loss literature

3.4 Complete GAN Training Pipeline

What I Did: I designed and implemented the complete adversarial training loop with alternating optimization and combined losses.

```
Training Algorithm:
Initialize:
   G ← Fine-tuned ECCV16 (from Stage 1)
   D ← PatchGAN (random initialization)
   opt_G ← Adam(lr=0.0002, β=(0.5, 0.999))
   opt_D ← Adam(lr=0.0001, β=(0.5, 0.999))

For each epoch (1 to 50):
   For each batch (L, ab_real):
      // Step 1: Train Discriminator
      ab_fake ← G(L)
      real_input ← concat(L, ab_real)
      fake_input ← concat(L, ab_fake.detach())

      D_real ← D(real_input)
      D_fake ← D(fake_input)

      L_D_real ← BCE(D_real, ones)
      L_D_fake ← BCE(D_fake, zeros)
      L_D ← 0.5 × (L_D_real + L_D_fake)
      Update D parameters
    // Step 2: Train Generator
      ab_fake ← G(L)
      fake_input ← concat(L, ab_fake)

      D_fake ← D(fake_input)

      L_GAN ← BCE(D_fake, ones)  // Fool discriminator
      L_L1 ← ||ab_fake - ab_real||₁
      L_perceptual ← VGG_loss(ab_fake, ab_real, L)

      L_G ← L_GAN + 10×L_L1 + 10×L_perceptual
      Update G parameters
      Log losses and save checkpoints every 5 epoch
```

Output: GAN colorization model
Training Parameters:

- Total epochs: 50
- Batch size: 16
- Generator learning rate: 0.0002
- Discriminator learning rate: 0.0001
- Loss weights: $\lambda\_L1 = 10$, $\lambda\_perceptual = 10$
- Optimizer: Adam with $\beta1=0.5$ (for GAN stability)

What I Monitored During Training:

1. Generator losses (L_GAN, L_L1, L_perceptual, Total)
2. Discriminator losses (L_D_real, L_D_fake, Total)
3. Visual quality of generated images every few epochs
4. Model convergence and stability

4. Results and Discussion

4.1 Training Dynamics

Fine-tuning Stage (Epochs 1-3):

- Initial MSE Loss
- Final MSE Loss
- Observation: Rapid convergence indicating successful adaptation to ImageNet_50
- Model learned dataset-specific color distributions

GAN Training Stage (Epochs 1-50): Generator Losses

- L_1 Loss (L1_Loss): Began at 9.67 (Epoch 1) and steadily decreased to 4.57 by Epoch 39, showing consistent structural and pixel-level improvement.
- Adversarial Loss (GAN_Loss): Started at 5.66, progressively increased as the generator learned to fool the discriminator more effectively, reaching ~13.7 by Epoch 39.
- Total Generator Loss (G_Loss): Decreased smoothly from 109.3 → 65.5, with only minor oscillations—indicating stable training and balanced competition between L_1 and adversarial components.

Discriminator Loss (L_D)

- Started at 0.22 (Epoch 1)
- Decreased to ~0.16 by Epoch 3
- Then stayed extremely stable between 0.162–0.165 for the remaining 35+ epochs.

The very stable plateau near 0.162 indicates. The discriminator is *not* overpowering the generator. It still provides useful gradients. Training reached a healthy equilibrium. This is a sign of stable GAN training — neither network collapses or dominates.The discriminator and generator losses showed healthy competition without mode collapse. The generator failed to be strong enough to fool discriminator

4.2 Visual Observations:

Success Cases:

1. Landscape images: GAN model produced vibrant blue skies and saturated green foliage, while baseline appeared washed out
2. Animals: Realistic fur colors with proper texture, especially for birds with colorful plumage
3. Objects: Better color contrast and saturation for everyday objects

Failure Cases:

1. Ambiguous objects: Occasionally produced unrealistic colors for objects with uncertain color mapping (e.g., gray objects colored blue instead of maintaining neutral tones)
2. Complex scenes: Sometimes oversaturated certain regions, creating unnatural appearance
3. Fine details: Occasional color bleeding at edges and overfitting on specific colors

Example Description: For a test image of a bird, the baseline produced a brownish bird with muted colors, while the GAN model generated a parrot-like appearance with vivid red, blue, and yellow plumage. While the GAN's colorization may not be accurate to the original, it appears more realistic and visually appealing.

Challenges Encountered:

1. Training Instability: Initial experiments without fine-tuning led to mode collapse. The two-stage approach resolved this.
2. Hyperparameter Tuning: Finding the right balance for λ_L1 and $\lambda\_perceptual$ required experimentation. Too high L1 weight → desaturated outputs; too low → unrealistic colors.
3. Computational Cost: Training for 50 epochs on GPU took approximately 18-20 hours on our hardware.
4. Oversaturation: Some images became oversaturated. This could be addressed with additional regularization or reduced adversarial loss weight.

---

5. Summary and Conclusions

5.1 Summary of Work

In this project, I developed a GAN-based image colorization system that improves upon the ECCV16 baseline through adversarial training and perceptual loss integration. My contributions included:

1. Fine-tuning the pretrained ECCV16 model on ImageNet_50 using supervised learning
2. Implementing a PatchGAN discriminator for spatial adversarial feedback
3. Integrating VGG19 perceptual loss to preserve semantic content
4. Designing and training the complete GAN pipeline with combined losses and evaluating by checking losses and visual comparison across epochs

5.2 Conclusion

This project successfully demonstrated that GAN-based colorization with perceptual loss can produce more vibrant and realistic results compared to traditional supervised approaches. The two-stage training strategy proved effective in combining the semantic understanding of the pretrained ECCV16 model with the realism-enhancing capabilities of adversarial training.

The training of GAN is very tricky and highly complex thing, hence it is very important to maintain the parameters such that the model does not collapse at the same time do not overfit. Though we were able to attain the former we could not overfit to out dataset. The future recommendations could be additional monitoring using validation set, transformations for data augmentations etc

---

6. Code Attribution

Percentage of code from internet sources: ~80%

Breakdown:

- Pix2Pix GAN framework: ~40% (PatchGAN discriminator architecture, training loop structure)
- Zhang et al. ECCV16 code: ~30% (Base generator architecture, preprocessing utilities)
- VGG perceptual loss: ~10% (Feature extraction implementation)
- PyTorch utilities: ~5% (Standard dataloaders, optimization)

Major external sources:

1. Pix2Pix implementation: https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix
2. ECCV16 colorization: https://github.com/richzhang/colorization
3. Perceptual loss: PyTorch torchvision VGG19 model

---

9. References

[1] Zhang, R., Isola, P., & Efros, A. A. (2016). *Colorful image colorization*. arXiv:1603.08511 [cs.CV]. https://arxiv.org/abs/1603.08511

[2] Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1125–1134). https://doi.org/10.1109/CVPR.2017.632

[3] Foun, M. H. (2024). Application and analysis of black and white image coloring based on generative adversarial networks (GANs). In *Proceedings of the 1st International Conference on Engineering Management, Information Technology and Intelligence (EMITI 2024)* (pp. 354–361). SCITEPRESS. https://doi.org/10.5220/0012938100004508

[4] Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 694–711). https://doi.org/10.1007/978-3-319-46475-6_43