

Individual Final Project Report

Dog Breed Classification and Image Generation Application

Adam Stuhltrager

December 2025

1. Introduction

This report documents my individual contribution to the group 6 Deep Learning final project, which focused on developing an end-to-end dog breed classification and image generation system. The project combines computer vision techniques with generative AI to create a comprehensive application for dog breed identification and artistic image synthesis.

1.1 Project Overview

The complete project consists of three major components developed collaboratively by the team:

1. **Dog Breed Classifier:** A fine-tuned ResNet-50 convolutional neural network trained on the Stanford Dogs dataset to classify 120 dog breeds with high accuracy.
2. **Realistic Image Generator:** A Stable Diffusion XL pipeline with base and refiner models, incorporating breed-specific anatomical priors and post-processing filters for photorealistic dog image generation.
3. **Streamlit Web Application:** An interactive web interface that integrates both the classifier and generator, providing users with an intuitive platform to classify uploaded images and generate new dog images in multiple artistic styles.

1.2 Shared Work Distribution

The project work was divided among team members as follows. My specific contribution was the development of the complete Streamlit web application, which serves as the user-facing interface integrating all project components. Other team members contributed the classifier training pipeline, the realistic generator development, and the dataset preparation and preprocessing.

2. Description of Individual Work

My primary contribution to this project was the design and implementation of the Streamlit web application that serves as the unified interface for the dog breed AI system. This section provides background on the technologies and architectural decisions involved.

2.1 Streamlit Framework

Streamlit is an open-source Python framework designed for building data science and machine learning web applications. It enables rapid prototyping by converting Python scripts into interactive web apps without requiring front-end development expertise. The framework provides built-in widgets for file uploads, sliders, buttons, and real-time visualization, making it ideal for deploying deep learning models.

2.2 Application Architecture

The application follows a modular architecture with clear separation of concerns:

Module	Description
app.py	Main Streamlit application with UI logic, tab navigation, and generation orchestration
config.py	Centralized configuration for model paths, style definitions, prompts, and default parameters
classifier.py	ResNet-50 classifier wrapper with HuggingFace model loading
generator.py	SDXL-based stylized generator with LoRA hot-swapping capability
realistic_generator.py	SDXL base+refiner pipeline with anatomical priors and realism post-processing

Table 1: Application module structure and responsibilities

2.3 Key Technical Challenges

Several technical challenges were addressed during the application development:

- **SDXL Pipeline Compatibility:** The Stable Diffusion XL encode_prompt() method returns different values across diffusers library versions (2 values in older versions, 4 values in newer versions including pooled embeddings). The solution was to bypass encode_prompt entirely and pass prompt strings directly to the pipeline.
- **Base+Refiner Step Calculation:** The SDXL ensemble requires both pipelines to use the same total_steps with denoising_end/denoising_start controlling the split. The split ratio is calculated as: $\text{split} = \text{steps_base} / (\text{steps_base} + \text{steps_refiner})$
- **Token Limit Constraints:** CLIP text encoders have a 77-token limit. The original anatomical prompts exceeded this limit, requiring condensation to approximately 40 tokens while preserving essential generation guidance.
- **Session State Management:** Streamlit reruns the entire script on each interaction. Classification results and breed selections needed to persist across tab switches using st.session_state.

3. Detailed Work Description

3.1 User Interface Design

The application features a two-tab interface designed for intuitive workflow:

Classify Tab: Users upload dog images through a file uploader widget. Upon clicking the classify button, the image is processed through the ResNet-50 model,

and the top-5 predictions are displayed with confidence percentages. The classified breed is stored in session state for optional use in generation.

Generate Tab: Users select a dog breed from a dropdown of 120 breeds or enter a custom breed name. They choose from five artistic styles (Realistic, Manga, Anime, Pastel Anime, Pixel Art). Advanced settings allow customization of generation parameters including inference steps, image dimensions, guidance scale, and random seed.

3.2 Model Integration

The application integrates multiple deep learning models with efficient caching:

Classifier Loading: The ResNet-50 classifier is loaded from HuggingFace Hub using the `huggingface_hub` library. The model weights and label mappings (`id2breed.json`) are downloaded on first use and cached locally. The `@st.cache_resource` decorator ensures the model is loaded only once per session.

Generator Loading: Two separate generators are maintained: a stylized generator (SDXL base with LoRA adapters) and a realistic generator (SDXL base + refiner). Both use lazy loading - models are only instantiated when first needed, with approximately 15GB of weights downloaded on initial use.

3.3 Realistic Generation Pipeline

The realistic generation pipeline required custom implementation in `app.py` to handle the SDXL base+refiner ensemble correctly:

Prompt Construction: A condensed prompt template ensures token efficiency: "RAW photo of a {breed} dog, full body side view, DSLR, 85mm lens, natural lighting, correct canine anatomy, four legs visible, proper joint angles, photorealistic, high detail, no stylization"

Two-Pass Generation: The base model performs initial denoising up to the split point (default 58.3%), outputting latent representations. The refiner model then continues from the split point to completion, adding fine details and improving coherence.

Post-Processing: Three realism filters are applied: `tone()` for color correction, `microfur()` for subtle texture enhancement, and `real_sensor_noise()` to add realistic camera sensor characteristics.

3.4 Style System Implementation

The application supports five distinct artistic styles through a unified configuration system:

Style	LoRA Source	Pipeline
Realistic	N/A (Base + Refiner)	SDXL Base + Refiner
Manga	LineAniRedmond-V2	SDXL + LoRA
Anime	SDXL-LoRA-slider.anime	SDXL + LoRA
Pastel Anime	pastel-anime-xl-lora	SDXL + LoRA
Pixel Art	pixel-art-xl	SDXL + LoRA

Table 2: Available generation styles and their implementations

4. Results

4.1 Application Functionality

The completed Streamlit application successfully integrates all project components into a functional web interface. Key results include:

1. **Successful Model Integration:** Both the ResNet-50 classifier and SDXL generators load and execute correctly within the Streamlit environment, with appropriate GPU utilization when CUDA is available.
2. **Cross-Tab State Persistence:** Classification results persist when switching between tabs, enabling seamless workflow from breed identification to image generation.
3. **Configurable Generation Parameters:** Users can adjust base steps (15-50), refiner steps (10-40), guidance scale (1.0-10.0), and image dimensions (512-1280 pixels) for realistic generation.
4. **Download Functionality:** Generated images can be downloaded as PNG files with descriptive filenames including breed and style information.

4.2 Performance Metrics

Metric	Value
Classification Inference Time	~0.5 seconds (GPU)
Realistic Generation Time (48 steps)	~45-60 seconds (NVIDIA T4)
Stylized Generation Time (50 steps)	~30-40 seconds (NVIDIA T4)
Initial Model Load Time	~2-3 minutes (first run)
Total Model Size	~15 GB (all models)

Table 3: Application performance metrics on AWS g4dn.xlarge instance

4.3 Discussion

The application successfully demonstrates the integration of classification and generative models in a unified interface. The modular architecture allows for easy extension with additional styles or models. The caching system effectively prevents redundant model loading, significantly improving user experience after initial startup.

The realistic generation pipeline shows improved anatomical accuracy compared to naive prompt approaches, though some breed-specific features may still require fine-tuning. The condensed prompt strategy successfully addresses the 77-token limitation while maintaining generation quality.

5. Summary and Conclusions

5.1 Summary

This project successfully developed a comprehensive Streamlit web application integrating dog breed classification and multi-style image generation. The application provides an intuitive interface for users to classify uploaded dog images and

generate new images in various artistic styles including photorealistic, manga, anime, and pixel art.

5.2 Key Learnings

1. **Diffusers Library Evolution:** The rapid development of the HuggingFace diffusers library means API compatibility requires careful attention, particularly for encode_prompt return values.
2. **SDXL Ensemble Mechanics:** The base+refiner pattern requires matching total_steps and careful split ratio calculation for optimal results.
3. **Prompt Engineering Constraints:** CLIP's 77-token limit necessitates careful prompt optimization to maximize generation guidance within constraints.
4. **Streamlit State Management:** Complex multi-page applications require explicit session state management to maintain data across reruns.

5.3 Future Improvements

- **Batch Generation:** Add support for generating multiple images simultaneously with seed variations.
- **Image-to-Image Mode:** Enable users to upload reference images for style transfer or pose guidance.
- **Additional Styles:** Expand the style library with more LoRA adapters for diverse artistic outputs.
- **Model Optimization:** Implement model quantization or distillation for faster inference and reduced memory requirements.
- **User Authentication:** Add user accounts for saving generation history and personal style preferences.

6. Code Originality Calculation

The following calculation documents the originality of my code contribution to the Streamlit application:

6.1 File Summary

File	Lines	Description
app.py	407	Main Streamlit application
config.py	255	Configuration (130 lines code + 125 lines breed data)
Total	662	537 lines of code logic

Table 4: Code contribution summary by file

6.2 External Source Analysis

Code adapted from Streamlit documentation (~20 lines): Basic patterns including st.set_page_config(), @st.cache_resource decorator usage, and widget initialization patterns.

Code adapted from HuggingFace Diffusers documentation (~20 lines): SDXL pipeline call patterns for base and refiner models, including denoising_end/denoising_start parameter usage.

Lines modified from adapted code: Approximately 30 lines were significantly modified to implement custom logic including the split ratio calculation, prompt construction, and error handling.

6.3 Calculation

Using the formula: (copied – modified) / (copied + original) × 100

- Total code lines (excluding breed data list): 537
- Lines copied from external sources: 40
- Lines modified from copied code: 30
- Original lines written: 497

$$(40 - 30) / (40 + 497) \times 100 = 10 / 537 \times 100 = 1.86\%$$

Result: Approximately 1.86% of the code was derived from external sources, with 98.14% being original implementation for my contribution. The DOG_BREEDS list (125 lines) was excluded from this calculation as it represents data imported from the Stanford Dogs dataset rather than programmatic logic.

7. References

- [1] Streamlit Documentation. "Streamlit API Reference." Available: <https://docs.streamlit.io/>
- [2] HuggingFace. "Diffusers Library Documentation." Available: <https://huggingface.co/docs/diffusers/>
- [3] Stability AI. "Stable Diffusion XL: Improving Latent Diffusion Models for High-Resolution Image Synthesis." arXiv:2307.01952, 2023.
- [4] He, K., Zhang, X., Ren, S., & Sun, J. "Deep Residual Learning for Image Recognition." CVPR 2016.
- [5] Khosla, A., Jayadevaprakash, N., Yao, B., & Li, F. F. "Novel Dataset for Fine-Grained Image Categorization: Stanford Dogs." CVPR Workshop 2011.
- [6] Hu, E. J., et al. "LoRA: Low-Rank Adaptation of Large Language Models." ICLR 2022.