Project Title

Bidirectional Vision–Language Networks for Cross-Modal Generation and Reconstruction

Project Description

This project investigates how neural networks can learn **bidirectional mappings** between visual data and natural-language descriptions. Using a paired image–caption dataset (3k–20k samples, each image described with 100–200 words), the goal is to train two complementary neural models:

1. Text-to-Image Decoder

A generative model that takes a long-form textual description as input and produces an image that matches the described content.

- The caption is converted into a vector representation (via learned embeddings or pretrained text encoders).
- A decoder network generates an image conditioned on this embedding.
- The objective is to reconstruct the original image as accurately as possible.

2. Image-to-Text Encoder

A vision model that takes an image as input and outputs a detailed natural-language description.

- A CNN or Vision Transformer extracts visual features.
- A language model (e.g., LSTM/Transformer decoder) generates a multi-sentence caption approximating the original 100–200 word label.

Both models are trained **independently** but on the **same dataset with the same image–caption pairs**.

3. Cross-Modal Evaluation: Image-to-Image Translation Pipeline

As a final evaluation step, the two trained networks are connected sequentially:
Image → Caption → Reconstructed Image

This allows testing whether the image–caption–image cycle preserves semantic content and visual structure.

- If the networks generalize well, the reconstructed image should resemble the original.
- This forms an interpretable cross-modal autoencoder without joint training.

Overall Goal

The project aims to explore how well independently trained vision–language models can communicate through a shared semantic space and whether text serves as an effective bottleneck representation for visual reconstruction.

Below is a **clean, academic-style task description** of your project + a **step-by-step technical plan** for how you will train both models **from scratch** (no pretrained encoders, no pretrained diffusion, no BLIP, no CLIP).
This fits directly into a project report, proposal, or thesis chapter.

📌 Project Description: Bidirectional Vision–Language Networks Trained from Scratch

This project explores whether vision and language models, trained independently and without pretraining, can communicate through a shared semantic bottleneck formed by long natural-language descriptions. Using a paired dataset of 5k–50k images with human-written or LLM-generated paragraph captions (100–200 words each), two neural networks are trained from scratch: (1) a **Text-to-Image Decoder**, which generates an image given a long-form caption, and (2) an **Image-to-Text Encoder**, which reconstructs a detailed caption from an input image. The networks are **not jointly trained** and do not share parameters; instead, they rely on long text as an explicit cross-modal latent space. After training both models independently, they are connected to form a **cross-modal reconstruction pipeline**:
Image → Caption → Reconstructed Image.
This cycle evaluates the degree to which semantic content is preserved when information is translated across modalities. If both models learn expressive and consistent representations, the reconstructed images should approximate the original in terms of object composition,

spatial layout, colors, and described attributes. The project ultimately tests whether **language can act as a meaningful information bottleneck** for visual reconstruction when all components are trained from scratch on a moderately sized dataset.

📌 Required Components and Steps

Below is a structured plan for implementing the entire system.

✅ 1. Dataset Preparation (Historic Art Dataset)

```
artwork/                ← contains .jpg images
artwork_dataset.csv     ← metadata for each artwork
info_dataset.csv        ← additional metadata
```

Input

~14,000 artwork images stored in:

complete/artwork/*.jpg

-

Metadata provided in:

complete/artwork_dataset.csv
complete/info_dataset.csv

-
- Long captions (100–200 words) will be **generated by an LLM**, since the dataset does not include descriptive paragraphs.

Tasks

1. Load and Validate Images

Read all `.jpg` files from:

complete/artwork/

-

- Extract unique identifiers (e.g., filename without extension).

2. Read Metadata

Use:

artwork_dataset.csv
info_dataset.csv

-
- Join on artwork ID or filename where needed.
- Metadata sources:
    - Title
    - Artist
    - Year
    - Medium
    - Style / Movement
    - Short descriptions (if available)

3. Generate Long Descriptions (LLM)

Since the dataset does not include 100–200 word captions:

- For each image:
    - Send the image and metadata to an LLM
    - Generate a **150–200 word visual description**
      (Focus on composition, colors, subjects, details, textures, symbolism)

Store mapping:

{ "image_id": "long caption ... " }

-

4. Preprocess Images

To make training-from-scratch stable:

- Resize all images to either:
    - **128×128** (fastest, most stable)
    - or **256×256** (more detail, heavier compute)

Normalize pixel values to:

[-1, 1] or [0, 1]

-

5. Tokenize Captions

Train your **own tokenizer** on the generated caption corpus:

- Use Byte-Pair Encoding (BPE) or SentencePiece
- Vocabulary size: **5,000–10,000 tokens**

Tokenize all captions:

tokenized_captions[image_id] = [tokens...]

-

6. Build Dataset Splits

Split dataset by image ID:

- **80% train**
- **10% validation**
- **10% test**

Store as:

```
splits/
  train_ids.txt
  val_ids.txt
  test_ids.txt
```

7. Store Final Dataset

A clean, training-ready format:

```
dataset/
  images/
    00001.jpg
    00002.jpg
    ...
  captions.json        ← long 100–200 word descriptions
  metadata.csv         ← filtered + cleaned metadata if needed
  tokenized_captions.json
  splits/
    train_ids.txt
    val_ids.txt
    test_ids.txt
```

Where `captions.json` looks like:

```
{
  "00001": "A richly detailed oil painting featuring ... 150–200 words ...",
  "00002": "This classical artwork depicts a dramatic ...",
  ...
}
```

1.

---

2. Image-to-Text Model (Image Encoder + Caption Decoder)

Goal

Generate a paragraph-length natural-language description from an image.

Model Architecture

You will implement a simple transformer-based captioning model:

Encoder:

- A CNN built from scratch:

    - Example: 6× Conv → ReLU → Pool blocks, output shape ~ (H/32 × W/32 × d)
    - Flatten → Linear → produce **image embedding (vector of size 512–1024)**

Decoder:

- Transformer or LSTM
- Inputs:
    - start token `<s>`
    - previous words
    - encoder embedding as memory/key/value

Training Objective

Next-token cross-entropy:

$$L_{capt} = - \Sigma \log P(w_t \mid w_{<t}, imageEmbedding)$$

-

Training Notes

- Train **30–60 epochs**
- Batch size: 8–32
- Learning rate: 1e-4
- Teacher forcing for first 80% of training
- Validation: perplexity, BLEU, ROUGE, BERTScore

---

3. Text-to-Image Model (Caption Encoder + Image Decoder)

Goal

Generate a visual reconstruction conditioned on text.

Model Architecture

You will create a **conditional generative model** trained from scratch:

Caption Encoder

- Train a text encoder from scratch:
    - Embedding layer (dim 256–512)
    - Transformer encoder (2–4 layers)
    - Output a **global text embedding vector** (mean-pooled or `[CLS]` token)

Image Decoder

Pick one of these architectures:

Option A — Conditional VAE (most stable from scratch)

- Latent space z ~ N(0, I)
- Conditioning: concatenate text embedding with z
- Decoder: deconvolutional network producing image

Loss:

$L = L_{pixel} + \beta KL(q(z|x,t) || p(z))$

Option B — Conditional GAN (higher quality)

- Generator: deconvolutional network
- Discriminator: CNN predicting real vs fake conditioned on text

Loss:

$L_{GAN} = E[\log D(x,t)] + E[\log(1 - D(G(t), t))]$

Option C — Diffusion Model From Scratch (requires >10k images)

- U-Net
- Conditional noise prediction εθ(x_t, t, textEmbed)

---

4. Independent Training of Both Networks

Training Order (Important)

1. **Train the Image → Text model first**
    - It is easier and helps validate caption quality.
2. **Train the Text → Image model next**
    - It depends heavily on caption richness.

Why independent?

- Matches your project's hypothesis:
  "Can text serve as a shared semantic bottleneck without joint optimization?"

Training Tips

- Use early stopping on validation loss.
- Only use small images (128px) for stable training.
- Balance dataset: avoid too many images of one type.
- Apply augmentations only to Image→Text model, not Text→Image.

---

5. Cross-Modal Cycle Evaluation (Image → Caption → Image)

Steps:

1. Take a test image.

Run:

caption = ImageToText(image)

2.

Feed caption to:

reconstructed = TextToImage(caption)

3.
4. Compute cycle metrics:
    - **CLIPScore (you can calculate this without using clip to train)**
    - LPIPS (perceptual similarity)
    - Color histogram similarity
    - Caption similarity between original caption and regenerated caption
    - Human evaluation (qualitative)

What you expect:

- Semantic consistency (objects, layout)
- Color + structure partially preserved
- Increased diversity due to text bottleneck
- Errors cluster around fine details (faces, textures)

📌 Summary of Total Pipeline

DATASET
├── 5k–50k images
└── 100–200 word captions

MODEL 1: Image → Text
 ├── CNN encoder
 └── Transformer/LSTM decoder

MODEL 2: Text → Image
 ├── Transformer text encoder
 └── VAE/GAN/Diffusion image decoder

CYCLE
 image → caption → reconstructed image

The core research question:
Do independently trained networks using language as a bottleneck preserve cross-modal semantics?

📌 Tokenizer Details

2. How the Tokenizer Is Used in Both Networks

You will **reuse the same pretrained tokenizer object** everywhere so that:

- The **Image→Text model** learns to output token IDs in *that* vocabulary.
- The **Text→Image model** receives token IDs in the same format.

A. Image → Text (captioning model)

- **Training:**
    - Take image x.

Load its caption and tokenize:

```
 enc = tokenizer(caption, max_length=256, truncation=True, padding="max_length")
input_ids = enc["input_ids"]      # target tokens
attention_mask = enc["attention_mask"]
```

    -

    - Use `input_ids[:, :-1]` as decoder input and `input_ids[:, 1:]` as targets for next-token prediction.
    - Loss: cross-entropy over the tokenizer's vocabulary size `V`.
- **Inference (to get text):**
    - The model predicts a sequence of token IDs: `pred_ids`.

Turn back to text with:

caption = tokenizer.decode(pred_ids, skip_special_tokens=True)

    -

B. Text → Image (generative model)

- **Training:**

Take caption text, tokenize with the **same tokenizer**:

enc = tokenizer(caption, max_length=256, truncation=True, padding="max_length")
input_ids = enc["input_ids"]
attention_mask = enc["attention_mask"]

    -
    - Feed token IDs (or their embeddings) into your text encoder to get a text embedding.
    - Condition your image generator (VAE / GAN / diffusion) on that embedding.
- **Inference (from generated caption):**
    - Take generated caption text from Image→Text model.
    - Tokenize with the same tokenizer.
    - Feed tokens into text encoder → image decoder → reconstructed image.

So the pattern is:

- **Tokenizer** = fixed, pretrained, widely available (e.g. GPT-2 tokenizer).

- **Both models** use that tokenizer for:
    - `text → token IDs → embeddings` (input)
    - `logits → token IDs → text` (output, for the captioning model).
- 100–200 word captions generated by an LLM for each image.
- A **standard pretrained tokenizer**, e.g. from HuggingFace:
    - `gpt2`
    - or `bert-base-uncased`
    - or `facebook/bart-base`
      (any widely used subword tokenizer is fine; just pick one and stick to it for the whole project.)

Tokenize captions using a pretrained tokenizer
Steps:

Load a well-known tokenizer, e.g.:

```
 from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

For each caption:

```
 enc = tokenizer(
    caption,
    max_length=256,
    truncation=True,
    padding="max_length"
)
input_ids = enc["input_ids"]
attention_mask = enc["attention_mask"]
```

Save tokenized captions (or regenerate on the fly) as:

```
 tokenized_captions.json
{
  "00001": {
```

```
    "input_ids": [...],
    "attention_mask": [...]
  },
  ...
}
```