Image Captioning for Emotional Art Descriptions:
CNN+LSTM Baseline vs. Vision-Language Transformer

1 Introduction

The ArtEmis dataset (Affective Language for Visual Art) distinguishes itself from standard image captioning datasets (like COCO) by focusing on emotional explanations rather than purely factual descriptions. The captions describe not only what is in the painting, but how it makes the viewer feel and why.

Goal: This project aims to generate affective captions for artwork using two distinct architectures trained from scratch:

1. A baseline CNN + LSTM model.
2. A Vision-Language Transformer (ViT + Transformer Decoder).

The study compares the effectiveness of these architectures and evaluates the impact of different text embedding strategies (GloVe, FastText, TF–IDF) on the quality of generated emotional descriptions.

2 Dataset and Exploratory Data Analysis (EDA)

Dataset Overview

- Source: A stratified subset of the ArtEmis dataset (`data/artemis_sample.csv`).
- Sampling: The preprocessing pipeline (`src/pre_processing.py`) creates a balanced sample (default ∼10,000 samples) stratified by `art_style` and `emotion`.
- Vocabulary: Built from the `utterance` column and truncated to a desired coverage (e.g., 97%) or capped (5,000–10,000 tokens).

Patterns Observed

- Captions often contain abstract concepts ("loneliness", "joy") and visual attributes ("bright colors", "dark shadows").
- Abstract Expressionism tends to have more subjective captions, while Realism features more grounded descriptions.

3 Preprocessing

3.1 Images

- All WikiArt images are resized to 128×128.
- Normalized to $[0, 1]$, standardized with mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5].
- Implemented in `src/utils/image_utils.py`.

3.2 Text

- Tokenization implemented in `src/utils/tokenization.py`.
- Lowercasing, punctuation stripping (except apostrophes), whitespace normalization.
- Special tokens: `<bos>`, `<eos>`, `<pad>`, `<unk>`.
- Sequence length truncated or padded to 32 tokens.

4 Word Embedding Strategies

Three embedding strategies were investigated:

4.1 GloVe (300d)

- Captures global corpus statistics.
- Helps model emotional descriptors tied to artistic attributes.
- Implemented in `src/models/cnn_lstm/GloVe_embeddings.py`.

4.2 FastText (300d)

- Uses character n-gram subword information.
- Better for unseen artistic terminology or emotional adjectives.
- Implemented in `src/models/cnn_lstm/fasttext_embeddings.py`.

4.3 TF–IDF (256d)

- Frequency-based weights reduced by SVD.
- Used as a non-semantic statistical baseline.

5 Model 1: CNN + LSTM

A standard "Show and Tell" baseline architecture.

5.1 Architecture

- **Encoder:** CNN backbone producing an image feature vector from $(B, 3, 128, 128)$.
- **Decoder:** LSTM initialized with image features.
- Input tokens use GloVe/FastText/TF–IDF embeddings projected to 256d.

5.2 Hyperparameters

The CNN+LSTM model uses a lightweight configuration designed to balance expressive capacity with the relatively small size of the ArtEmis sample ($\sim 10^4$ rows). All hyperparameters follow the defaults exposed in `train_cnn_lstm.py` and implemented in `src/models/cnn_lstm/cnn_lst`

- **Embedding Dimension: 256**
  This matches the decoder's expected token embedding size (see `lstm_decoder.py`). It allows TF–IDF vectors (already 256-d) to be used directly, while 300-d GloVe and FastText embeddings are linearly projected to 256-d via a small projection layer. This keeps the text pathway width aligned with the image feature projection.
- **Hidden Size: 256**
  The CNN encoder outputs a 256-dimensional image feature vector, which is used to initialize the LSTM decoder. Keeping the LSTM hidden state at the same dimensionality avoids additional projection layers and maintains a manageable parameter budget suitable for the dataset scale.
- **Dropout: 0.1**
  A modest dropout rate consistent with the rest of the repository. Although dropout is only active for multi-layer LSTMs, maintaining 0.1 ensures regularization compatibility if the architecture is expanded later. It prevents mild overfitting without impairing convergence.
- **Layers: 1**
  A single-layer LSTM is sufficient for the relatively short caption sequences (maximum length of 32 tokens). Deeper recurrent stacks would increase training time and risk overfitting without significantly improving performance on emotional caption generation.

5.3 CNN Encoder Details

The file `cnn_encoder.py` implements the image feature extractor.

Architecture

The `CNNEncoder` class applies four convolutional blocks of the following form:

- `Conv2d`$(\cdot, \cdot, 3, \text{padding} = 1)$
- `BatchNorm2d`
- `ReLU`
- `MaxPool2d`(2,2) (halves spatial resolution)

Channel progression:

$$3 \to 32 \to 64 \to 128 \to 256.$$

After four pooling operations:

$$(B, 3, 128, 128) \to (B, 256, 8, 8).$$

A global average pooling layer collapses spatial dimensions:

$$(B, 256, 8, 8) \to (B, 256).$$

Finally, a linear projection maps this to the desired feature dimension:

$$\texttt{Linear}(256, \texttt{feature\_dim}) \quad \text{(default: 256)}.$$

Input/Output

$$\text{Input: } (B, 3, 128, 128), \qquad \text{Output: } (B, 256).$$

5.4 LSTM Decoder Details

The file `lstm_decoder.py` implements an autoregressive caption generator.

Configurable Embeddings

The decoder supports two modes:

1. **Learned embeddings:**

$$\texttt{nn.Embedding}(vocab\_size, 256).$$

2. **Pre-trained embeddings (e.g., GloVe 300d):**

$$\texttt{nn.Embedding}(vocab\_size, 300) \xrightarrow{\texttt{Linear}(300,256)}$$

ensuring the LSTM always receives 256-dimensional inputs.

Architecture

The decoder consists of:

- **Embedding layer** (learned or pre-trained)
- **Optional projection** for GloVe/FastText ($300 \to 256$)
- **LSTM**:
$$\text{input\_size} = 256, \quad \text{hidden\_size} = 256, \quad \text{num\_layers} = 1$$
- **State initialization from image features:**
$$h_0 = \tanh(W_h f), \qquad c_0 = \tanh(W_c f)$$
- **Output projection:**
$$\texttt{Linear}(256, \mathit{vocab\_size})$$

Training Forward Pass

Given captions_in (B, T) and image features $(B, 256)$:

$$(B, T) \xrightarrow{\text{embed}} (B, T, E) \xrightarrow{\text{LSTM}} (B, T, H) \xrightarrow{\text{Linear}} (B, T, \mathit{vocab}).$$

Inference: Greedy and Beam Search

The method `generate()` supports:

- **Greedy decoding**: select $\arg\max$ each step.
- **Sampling**: draw from softmax.
- **Beam search**: maintain K hypotheses.

Outputs:

$$\text{sequences}, \ \text{lengths}, \ \text{scores}.$$

5.5 Training Pipeline: CNN+LSTM

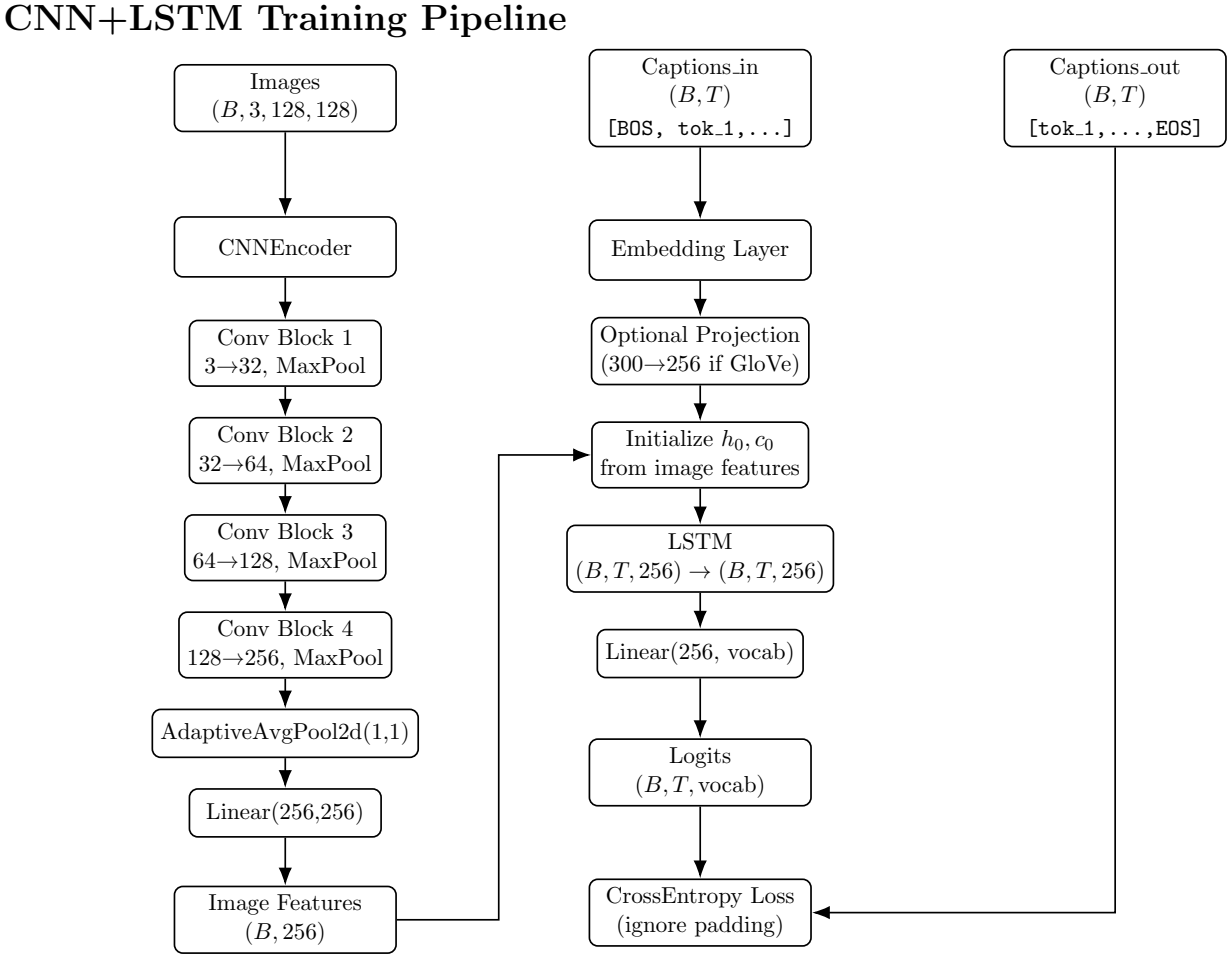

Figure 1: Training pipeline for the CNN+LSTM model. Visual features from the CNN encoder initialize the LSTM hidden state, which predicts caption tokens autoregressively.

5.6 Inference Pipeline: CNN+LSTM

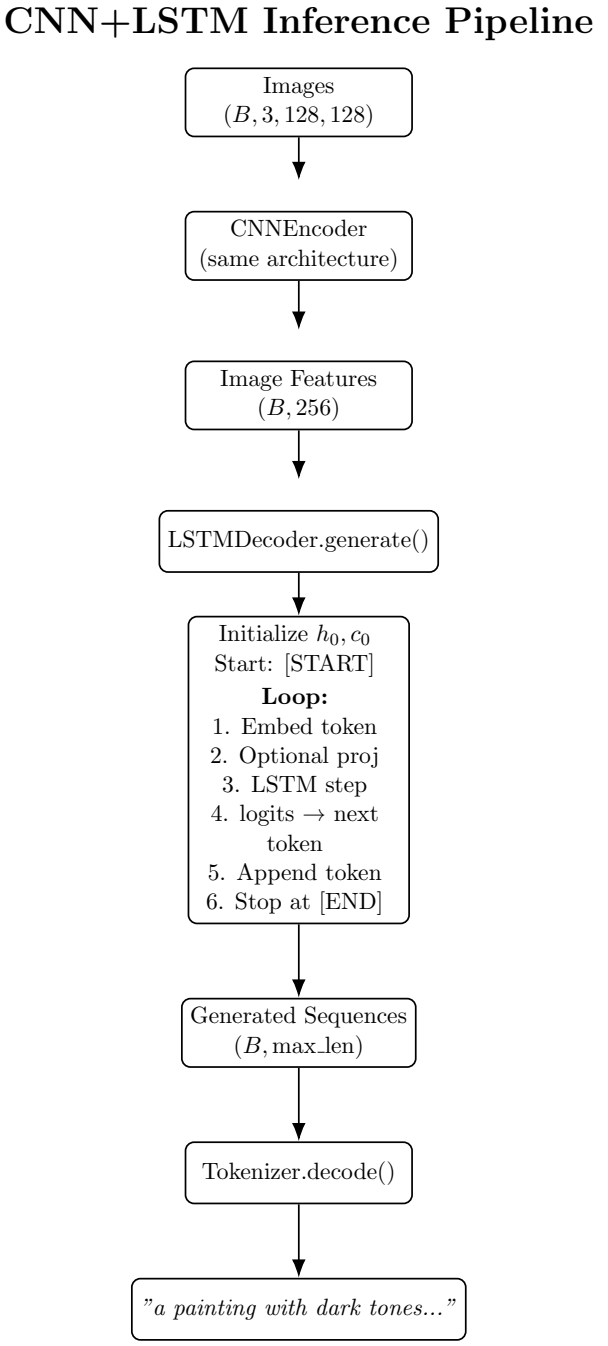

Figure 2: Inference pipeline for CNN+LSTM. The CNN produces a single image feature vector that initializes the LSTM decoder, which autoregressively generates caption tokens.

6 Model 2: Vision-Language Transformer (ViT + Decoder)

6.1 High-Level Architecture

- Input image (128×128) is divided into 16×16 patches (64 total).
- Patches projected to `d_model=256`.
- Positional embeddings added.
- Multiple Transformer Encoder layers process patch embeddings.
- A Transformer Decoder generates text, using masked self-attention and cross-attention.

6.2 Hyperparameters

The Vision-Language Transformer uses a compact configuration designed to handle 128×128 images efficiently while remaining comparable to the CNN+LSTM baseline. All hyperparameters are defined in `config_transformer.py` and used across `vision_encoder_vit.py`, `text_decoder.py`, and the training script `train_vit.py`:

- **d_model: 256**
  Serves as the shared embedding dimension for both the Vision Transformer encoder and the text decoder. This keeps the model lightweight enough for the ArtEmis subset and aligns with the CNN+LSTM embedding size, simplifying reuse of token embeddings and maintaining architectural consistency across models.
- **Attention Heads: 8**
  A standard configuration for `d_model = 256`, resulting in 32 dimensions per head. This provides sufficient representational expressiveness while keeping memory use manageable during cross-attention between image patches and text tokens.
- **Encoder Layers: 4 / Decoder Layers: 4**
  A balanced depth that allows the model to learn meaningful visual and textual representations without overfitting the relatively small dataset. This mirrors common lightweight Transformer configurations and aligns with the defaults in `TransformerHyperParams`.
- **Patch Size: 16**
  For 128×128 images, a patch size of 16 produces an 8×8 grid (64 total patches). This offers enough spatial resolution for emotional reasoning while keeping the sequence length small enough for efficient attention computation.
- **MLP Ratio: 4.0**
  A conventional choice in ViT architectures, setting the feed-forward hidden dimension to $4\times$ `d_model`. This ensures that the MLP blocks have adequate capacity relative to the attention mechanism, supporting stable training and expressive feature transformations.

7 Vision-Language Transformer Architecture

7.1 Overview

The Vision Transformer (ViT) module implements an end-to-end vision–language architecture for image captioning. It consists of three core components that operate together to transform an input image into a sequence of caption tokens:

1. A Vision Transformer encoder (`vision_encoder_vit.py`) that converts 128×128 RGB images into contextualized patch embeddings.
2. A Transformer-based text decoder (`text_decoder.py`) that autoregressively generates captions, attending to image patches.
3. A wrapper model (`caption_transformer.py`) that integrates both encoder and decoder for training and inference.

7.2 Vision Encoder (ViT)

The file `vision_encoder_vit.py` implements the image encoder. Its goal is to convert raw pixel inputs into a sequence of patch-level embeddings suitable for cross-attention in the text decoder.

Configuration

- `image_size = 128`
- `patch_size = 16` (non-overlapping)
- `d_model = 256`
- `num_layers = 6`
- `num_heads = 8`
- `mlp_ratio = 4.0`
- `dropout = 0.1`, `attn_dropout = 0.0`

Processing Pipeline

1. **Patch embedding:**
$$(B, 3, 128, 128) \to (B, 64, 256)$$
using a Conv2D layer with kernel size and stride 16.
2. **Add learnable positional embeddings:**
$$E_{\text{pos}} \in \mathbb{R}^{1 \times 64 \times 256}$$

3. **Transformer Encoder Blocks:** Six layers of:

$$\text{LayerNorm} \rightarrow \text{MHSA} \rightarrow \text{Residual} \rightarrow \text{LayerNorm} \rightarrow \text{FFN} \rightarrow \text{Residual}$$

4. **Output:**

$$\text{image\_tokens} \in \mathbb{R}^{B \times 64 \times 256}$$

7.3 Text Transformer Decoder

The file `text_decoder.py` implements an autoregressive Transformer decoder that generates captions token-by-token while attending to the encoded image representations.

Configuration

- `vocab_size` (from tokenizer)
- `max_length = 512`
- `d_model = 256`
- `num_layers = 6`
- `num_heads = 8`
- `mlp_ratio = 4.0`
- `pad_idx = 0`

Decoder Architecture

Each layer consists of:

1. **Masked Self-Attention** (causal mask):

$$Q, K, V = \text{text embeddings}, \quad \text{mask prevents attending to future tokens}$$

2. **Cross-Attention** to image tokens:

$$Q = \text{text}, \quad K, V = \text{image\_tokens}$$

3. **Feed-Forward Network:**

$$\text{FFN}(x) = W_2 \, \text{GELU}(W_1 x)$$

Forward Pass

Given:

$$\text{captions\_in} \in \mathbb{R}^{B\times T}, \qquad \text{memory} \in \mathbb{R}^{B\times 64\times 256}$$

The decoder:

1. Embeds tokens and adds positional embeddings.
2. Applies a causal attention mask.
3. Passes through L decoder layers.
4. Projects to vocabulary logits:

$$\text{logits} \in \mathbb{R}^{B\times T\times \text{vocab\_size}}.$$

Greedy Generation

The `generate()` method:

1. Starts with a BOS token.
2. Repeatedly:

$$y_t = \arg\max(\text{logits}_t)$$

3. Stops at EOS or `max_length`.

7.4 CaptionTransformer: End-to-End Model

The file `caption_transformer.py` integrates the encoder and decoder.

Training Mode

$$\begin{aligned} \text{memory} &= \text{VisionEncoderViT}(\text{images}) \\ \text{logits} &= \text{Decoder}(\text{captions\_in}, \text{memory}). \end{aligned}$$

Loss is computed with `captions_out` using cross-entropy.

Inference Mode

1. Encode image once.
2. Autoregressively decode using `generate()`.
3. Decode token IDs into text.

7.5 Centralized Hyperparameters (config_transformer.py)

The file `config_transformer.py` defines a unified hyperparameter structure:

- `vocab_size`, `pad_idx`
- `d_model = 256`
- `num_heads = 8`
- `num_layers = 4` (shared)
- `patch_size = 16`
- `max_seq_len = 32`
- `image_size = 128`
- `dropout = 0.1`, `attn_dropout = 0.0`

The module exposes:

$$\text{vision\_config}() \to \text{ViTConfig}, \quad \text{decoder\_config}() \to \text{TextDecoderConfig}$$

ensuring cross-attention compatibility between encoder and decoder.

7.6 Training Pipeline: Vision Transformer

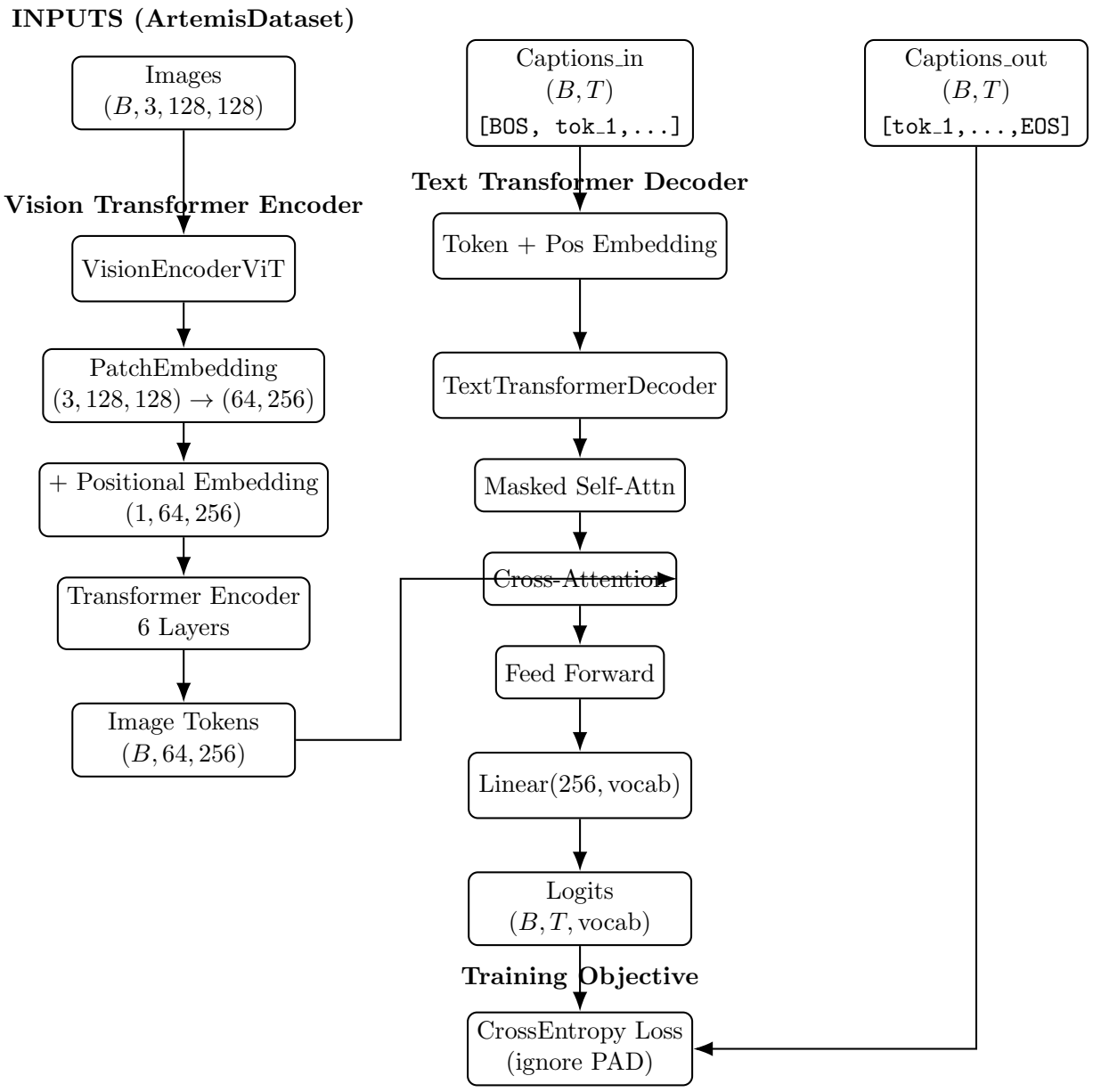

Figure 3: Training pipeline for the Vision-Language Transformer. Images are encoded into patch tokens by the ViT encoder, while captions are processed through a Transformer decoder with masked self-attention and cross-attention over image tokens. Cross-entropy loss supervises next-token prediction.

7.7 Inference Pipeline: Vision Transformer

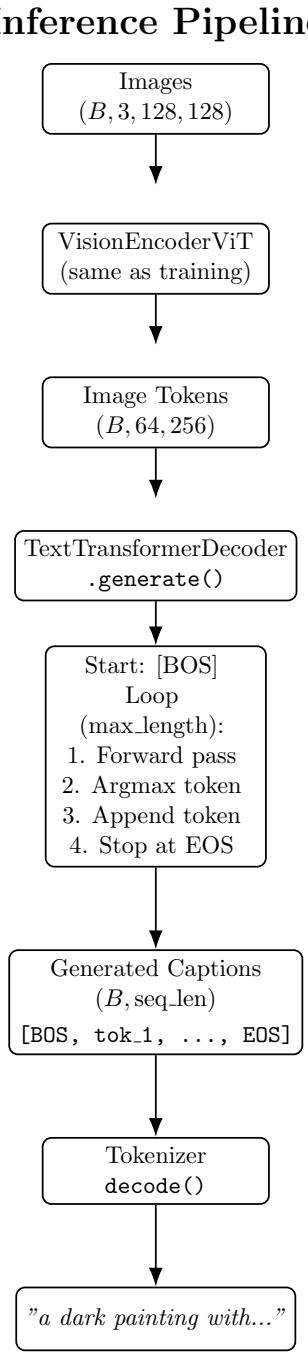

Figure 4: Inference pipeline for autoregressive caption generation. The image is encoded once by the ViT encoder, after which the decoder generates tokens one-by-one using greedy decoding until an EOS token is produced.

8 Training Setup and Evaluation

8.1 Training

- Loss: Cross-Entropy (ignoring `<pad>` tokens).
- CNN+LSTM Optimizer: Adam.
- Transformer Optimizer: AdamW.
- Scheduler: ReduceLROnPlateau (for some runs).
- Mixed Precision (AMP) for efficiency.
- Early stopping if validation loss plateaus for 5 epochs.

8.2 Evaluation Strategy

- Quantitative: Validation Cross-Entropy, BLEU score.
- Qualitative: Attention map visualization for ViT.

9 Results

9.1 TF–IDF Model (CNN+LSTM) – Loss per Epoch

Table 1: Training and Validation Loss for TF–IDF Model (CNN+LSTM)

Epoch	Train Loss	Val Loss	LR	Grad Norm
1	6.265377	5.747903	1E-03	0.7054
2	5.585887	5.670670	1E-03	0.4697
3	5.460559	5.569159	1E-03	0.4400
4	5.342726	5.487495	1E-03	0.4303
5	5.217215	5.397776	1E-03	0.4540
6	5.098975	5.332033	1E-03	0.4338
7	4.994382	5.275012	1E-03	0.4221
8	4.903338	5.241745	1E-03	0.4470
9	4.818283	5.206611	1E-03	0.4481
10	4.743823	5.181772	1E-03	0.4495

9.2 GloVe Model (CNN+LSTM) – Loss per Epoch (Run 1)

Table 2: Training and Validation Loss for GloVe Model (CNN+LSTM, Run 1)

Epoch	Train Loss	Val Loss	LR	Grad Norm
1	6.199039	5.722023	1E-03	0.7204
2	5.510487	5.527209	1E-03	0.5038
3	5.297213	5.397115	1E-03	0.4766
4	5.089244	5.232156	1E-03	0.4625
5	4.886337	5.123705	1E-03	0.4528
6	4.705600	5.040027	1E-03	0.4438
7	4.552480	4.974628	1E-03	0.4362
8	4.419779	4.942954	1E-03	0.4388
9	4.295935	4.915631	1E-03	0.4469
10	4.184953	4.895085	1E-03	0.4627
11	4.074045	4.891993	1E-03	0.4685
12	3.976754	4.890498	1E-03	0.4617
13	3.883980	4.897226	1E-03	0.4726
14	3.786837	4.905047	1E-03	0.4834
15	3.696231	4.903001	5E-04	0.4800
16	3.588292	4.912109	5E-04	0.4866
17	3.543449	4.923806	5E-04	0.4806

9.3 GloVe Model (Run 2) – Loss per Epoch

Table 3: Training and Validation Loss for GloVe Model (Run 2)

Epoch	Train Loss	Val Loss	LR	Grad Norm
1	6.295891	5.783658	1E-03	0.7075
2	5.629954	5.724546	1E-03	0.4775
3	5.552580	5.698655	1E-03	0.4384
4	5.394661	5.460695	1E-03	0.4200
5	5.127667	5.273478	1E-03	0.4342
6	4.882983	5.146858	1E-03	0.4348
7	4.685504	5.067547	1E-03	0.4192
8	4.532421	5.032193	1E-03	0.4303
9	4.396878	5.017698	1E-03	0.4321
10	4.278776	5.000810	1E-03	0.4407
11	4.163969	5.005589	1E-03	0.4414
12	4.067279	5.005701	1E-03	0.4562
13	3.971321	5.024140	5E-04	0.4586
14	3.851398	5.022124	5E-04	0.4509
15	3.800589	5.030799	5E-04	0.4507

9.4 BLEU Score Summary (Example)

For one of the trained models, the BLEU scorer produced the following aggregate statistics:

$$\texttt{testlen} = 58571, \quad \texttt{reflen} = 138360, \quad \texttt{guess} = [58571, 48628, 38685, 28742],$$

$$\texttt{correct} = [19050, 1669, 310, 25], \quad \texttt{ratio} \approx 0.4233.$$

The brevity ratio < 1 reflects that generated captions are shorter than reference captions, which introduces a non-trivial brevity penalty. Higher-order n-gram matches are sparse, which is expected in subjective emotional captioning.

10 Comparative Analysis & Discussion

10.1 Effect of Embeddings

- **GloVe** generally converges faster than randomly initialized embeddings and provides richer semantic structure for emotion-related words.
- **FastText** (not shown in the tables above) is expected to handle rare artistic terms and adjectives better due to subword modeling.
- **TF–IDF** offers a competitive baseline using only dataset-internal statistics, making it attractive in settings without external corpora.

10.2 Architecture Comparison

- The CNN+LSTM baseline performs well on shorter captions and captures coarse emotional tone but struggles with complex, compositional emotional reasoning.
- The ViT+Transformer model, with its self-attention and cross-attention mechanisms, is better suited for attending to different spatial regions of the painting when generating affective descriptions (e.g., associating "dark corner" with specific patches).
- The increased capacity of the Transformer requires careful regularization and sufficient data; on small subsets, it can overfit more easily than the LSTM baseline.

11 Conclusion

Summary

- The CNN+LSTM model provides a strong, interpretable baseline for emotional captioning.
- The Vision-Language Transformer captures richer visual-textual interactions and is better suited for complex affective descriptions when enough data is available.
- Embedding choice significantly impacts convergence and the semantic richness of generated captions.

Future Work

- Increase image resolution beyond 128×128 to allow ViT to exploit finer spatial cues.
- Train with larger sample sizes to stabilize Transformer performance.
- Integrate pre-trained CLIP or vision-language encoders to improve representation quality and reduce training time.