Automated Medical Report Generation from X-ray Images

Mahla Entezari
Shahid Beheshti University

January 2025

Abstract

This report details the development of an automated system for generating medical reports from X-ray images using deep learning. The primary objective is to alleviate the time-consuming manual interpretation process performed by medical professionals, thereby improving efficiency and potentially expediting patient care. Our proposed solution employs a multimodal deep learning approach, integrating a Vision Transformer (ViT) as an image encoder for robust feature extraction from X-ray images, and a GPT-2 language model as a text decoder for natural language generation. A crucial cross-attention mechanism is implemented to ensure semantic alignment between the visual information and the generated textual report. The project leverages PyTorch and the Hugging Face Transformers library, utilizing chest X-ray datasets for training and evaluation. Performance is assessed using standard text similarity metrics such as BLEU and ROUGE-L scores, with preliminary results demonstrating the viability of this automated approach. This document provides a comprehensive overview of the model architecture, implementation specifics, experimentation, and evaluation.

1 Introduction

Medical imaging, particularly X-ray radiography, plays a pivotal role in the diagnosis and monitoring of various diseases and conditions. Radiologists and medical professionals meticulously analyze these images to identify anomalies, formulate diagnoses, and compile detailed textual reports. This manual process, while crucial for patient care, is inherently labor-intensive and time-consuming. The demand for timely and accurate reports often outstrips the availability of skilled human resources, leading to potential delays in diagnosis and treatment. This bottleneck underscores the urgent need for automated solutions that can assist in the medical reporting workflow.

The advent of deep learning, especially with the development of powerful neural network architectures like Transformers, has revolutionized capabilities in both computer vision and natural language processing. This project aims to bridge these two domains by developing an intelligent system that can automatically generate descriptive medical reports directly from X-ray images. Such a system could significantly reduce the workload on medical staff, accelerate the reporting process, and provide a consistent baseline for report generation.

Our approach focuses on a multimodal deep learning model. The rationale behind this design is to process visual information from X-rays and translate it into coherent, medically accurate language. This involves sophisticated image understanding coupled with advanced text generation. We specifically opt for a Vision Transformer (ViT) due to its proven efficacy in capturing complex visual features from images, and a GPT-2 model, renowned for its ability to generate human-like text. The integration of these two powerful components through a cross-attention mechanism is critical to ensuring that the generated reports are not only grammatically correct but also contextually relevant to the X-ray findings.

This report is structured as follows: Section 2 elaborates on the theoretical foundations and the architectural design of our model. Section 3 details the technical setup, data preparation, and training procedures. Section 4 outlines the experimental setup, including dataset specifics and hyperparameters. Section 5 presents and discusses the outcomes of our evaluation. Finally, Section 6 summarizes the findings, acknowledges limitations, and suggests directions for future work.

2 Methodology

The core of our automated medical report generation system is a multimodal deep learning architecture that processes visual inputs and generates textual outputs. This section delves into the individual components and their synergistic integration.

2.1 Overall Architecture

The system is composed of two primary deep learning models: an **Image Encoder** and a **Text Decoder**, connected via a **Cross-Attention Mechanism**. This encoder-decoder framework allows the model to learn a mapping from the visual domain (X-ray images) to the linguistic domain (medical reports).

2.1.1 Image Encoder: Vision Transformer (ViT)

The Vision Transformer (ViT) is chosen as the image encoder. Traditionally, Convolutional Neural Networks (CNNs) have been dominant in computer vision. However, ViT demonstrates that standard Transformer architecture, widely successful in NLP, can also perform exceptionally well on image tasks when applied directly to sequences of image patches.

- **Patch Embedding**: An input X-ray image is first divided into a fixed-size grid of non-overlapping patches. Each patch is then flattened and linearly projected into a higher-dimensional embedding space. A learnable ”class token” is prepended to the sequence of patch embeddings, similar to BERT's `[CLS]` token, which will eventually encapsulate the global image representation.

- **Positional Embeddings**: To retain spatial information, learnable positional embeddings are added to the patch embeddings. This allows the model to understand the relative positions of the patches within the original image.

- **Transformer Encoder Layers**: The sequence of embeddings (class token + patch embeddings) is then fed into a stack of standard Transformer encoder layers. Each layer consists of a Multi-Head Self-Attention (MHSA) module and a feed-forward network (FFN), with Layer Normalization and residual connections. The MHSA allows the model to weigh the importance of different patches relative to each other, capturing long-range dependencies across the image.

- **Output**: The final hidden state of the class token from the last Transformer encoder layer serves as the comprehensive image embedding, representing the extracted visual features of the X-ray.

For this project, we utilize the pre-trained `google/vit-base-patch16-224` model from the Hugging Face Transformers library. Pre-training on large datasets like ImageNet provides a strong foundation for visual understanding, which is then fine-tuned for the medical domain.

2.1.2 Text Decoder: GPT-2 Language Model

The GPT-2 (Generative Pre-trained Transformer 2) is selected as the text decoder. GPT-2 is a large Transformer-based language model trained to predict the next word in a sequence, enabling it to generate coherent and contextually relevant text.

- **Tokenization**: Medical reports are tokenized into subword units using the GPT-2 tokenizer. This process converts raw text into numerical input IDs that the model can understand. Special tokens like `[BOS]` (Beginning of Sentence) and `[EOS]` (End of Sentence) are added to mark the start and end of reports.

- **Transformer Decoder Layers**: GPT-2's architecture is a stack of decoder-only Transformer layers. Each layer includes a masked multi-head self-attention mechanism (to prevent attending to future tokens) and a cross-attention mechanism.

- **Cross-Attention Integration**: This is where the multimodal aspect comes into play. A critical modification or integration point is needed. In a typical image captioning setup, the encoder's output (image embedding) is introduced into the decoder's layers via cross-attention. This means that at each decoding step, the GPT-2 model not only attends to the previously generated text tokens but also attends to the relevant parts of the image embedding provided by the ViT encoder.
  - Specifically, the hidden states of the GPT-2 decoder layers act as queries, while the ViT's output (or a transformed version of it) provides the keys and values for the cross-attention mechanism. This allows the language model to condition its text generation on the visual features.
- **Output Layer**: A linear layer followed by a softmax activation function predicts the probability distribution over the vocabulary for the next token in the sequence.

We use the pre-trained `gpt2` model from Hugging Face, adapting it to serve as a decoder conditioned on image features. The `text_tokenizer.pad_token = text_tokenizer.eos_token` ensures proper padding for batch processing.

2.2 Bridging Image and Text Modalities

A crucial step is to effectively bridge the gap between the fixed-size image embedding from ViT and the input requirements of the GPT-2 decoder.

- **Mapping MLP (Multi-Layer Perceptron)**: Since the output dimension of the ViT encoder (768 for `vit-base-patch16-224`) might not directly match the embedding dimension of GPT-2 (also 768 for `gpt2`), a simple MLP is used as an adapter.
  - If `image_embedding_dim != gpt2_embed_dim`, the `mapping_mlp` consists of a `nn.Linear` layer followed by a `nn.ReLU` activation function, transforming the ViT output to the GPT-2 embedding space.
  - If the dimensions are already compatible, an `nn.Identity()` layer is used, effectively bypassing the transformation.
- **Integration with GPT-2**: The transformed image embedding is then effectively `prepended` or used as the `encoder_hidden_states` in the GPT-2 model during training. This conditions the text generation on the visual input. During generation, this image embedding acts as an initial context for the GPT-2 model, guiding its subsequent token predictions.

3 Implementation Details

This section outlines the practical aspects of setting up the environment, preparing the data, and configuring the training process.

3.1 Environment Setup

The project relies on standard Python libraries for deep learning and data manipulation. The key libraries and their versions are specified in the notebook, typically installed via `pip`:

```
# !pip install opendatasets transformers nltk
```

The notebook also includes necessary imports for data handling, model building, and evaluation:

```
import os
import ast
import nltk
import torch
import pandas as pd
from PIL import Image
from tqdm import tqdm
import torch.nn as nn
import opendatasets as od
from google.colab import drive
import matplotlib.pyplot as plt
from nltk.translate.bleu_score import sentence_bleu
from torch.utils.data import DataLoader, random_split
from transformers import ViTFeatureExtractor, ViTModel
from transformers import GPT2Tokenizer, GPT2LMHeadModel, AdamW
```

NLTK's `punkt` tokenizer is downloaded for text processing, specifically for BLEU score calculation.

3.2 Data Acquisition and Preprocessing

The dataset used for this project is the **Chest X-Ray Images from Indiana University**, available on Kaggle.

- **Dataset Download**: The dataset is downloaded directly from Kaggle using the `opendatasets` library.

  ```
  od.download("https://www.kaggle.com/datasets/raddar/chest-
      xrays-indiana-university/data?select=indiana_projections.
      csv")
  ```

- **Data Loading and Merging**: Two primary CSV files, `indiana_projections.csv` (containing image metadata) and `indiana_reports.csv` (containing detailed reports), are loaded into pandas `DataFrames`. These `DataFrames` are then merged on a common unique identifier (`uid`) to link images with their respective reports.

  ```
  df_images = pd.read_csv("/content/chest-xrays-indiana-
      university/indiana_projections.csv")
  df_reports = pd.read_csv("/content/chest-xrays-indiana-
      university/indiana_reports.csv")
  merged_df = pd.merge(embeddings_df, df_reports, on="uid",
      how="inner")
  ```

```
merged_df = merged_df.dropna(subset=['findings']) # Ensure
    only entries with 'findings' are used
```

- **Image Embeddings Generation**: Before training, the X-ray images are processed by the pre-trained `ViTFeatureExtractor` and then fed into the `ViTModel` to generate image embeddings. These embeddings (specifically the `cls_embedding` from the last hidden state) are crucial for representing the image content numerically.

```
vit_model_name = "google/vit-base-patch16-224"
feature_extractor = ViTFeatureExtractor.from_pretrained(
    vit_model_name)
vit_model = ViTModel.from_pretrained(vit_model_name).to(
    device)
# ... loop through images to generate and save embeddings to
    a CSV
```

  The generated embeddings are saved to a CSV file (`MRI_ViT_embeddings.csv`) for persistent storage and easier loading in subsequent training runs.

- **Custom Dataset Class**: A custom PyTorch `Dataset` class, `MedicalMRIDataset`, is implemented to handle the loading of pre-computed image embeddings and corresponding text reports. This class ensures that data can be efficiently loaded in batches for training.

```
class MedicalMRIDataset(torch.utils.data.Dataset):
    def __init__(self, df):
        self.df = df
        self.embeddings = df["embedding"].values
        self.texts = df["findings"].values
        self.image_paths = df["img_path"].values

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):
        emb = self.embeddings[index]
        if isinstance(emb, str):
            emb = ast.literal_eval(emb)
        return {
            "embedding": torch.tensor(emb, dtype=torch.
                float32),
            "text": self.texts[index],
            "img_path": self.image_paths[index]
        }
```

- **Data Splitting and DataLoaders**: The dataset is split into training and testing sets (80%-20%) using `random_split`. `DataLoader`s are then created for efficient batching and shuffling of data during training and evaluation.

```
dataset = MedicalMRIDataset(merged_df)
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [
    train_size, test_size])
train_loader = DataLoader(train_dataset, batch_size=8)
test_loader = DataLoader(test_dataset, batch_size=8)
```

3.3 Model Initialization and Training

– **Tokenizer and Model Loading**: The GPT-2 tokenizer and `GPT2LMHeadModel` are loaded from their pre-trained checkpoints.

```
text_tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
text_model = GPT2LMHeadModel.from_pretrained("gpt2").to(
    device)
text_tokenizer.pad_token = text_tokenizer.eos_token
```

– **Optimizer and Loss Function**: The `AdamW` optimizer is chosen for training, which is a variant of Adam that incorporates weight decay for regularization. The loss function used is `nn.CrossEntropyLoss()`, which is standard for language modeling tasks.

```
optimizer = AdamW(list(text_model.parameters()) + list(
    mapping_mlp.parameters()), lr=5e-5)
criterion = nn.CrossEntropyLoss() # Although not
    explicitly used in the provided train_epoch, outputs.
    loss is used which internally computes cross-entropy.
```

– **Training Loop**: The `train_epoch` function orchestrates the training process.

  * The `text_model` is set to training mode (`text_model.train()`).
  * For each batch, text reports are tokenized, padded, and truncated to a `max_length` of 512.
  * The tokenized inputs are passed to the `text_model` to compute the loss.
  * Gradients are zeroed, backpropagation is performed, and optimizer steps are taken.
  * Periodically, training loss is printed to monitor progress.
  * After each epoch, the model's performance is evaluated on the test set, and a checkpoint is saved.

– **Checkpointing**: Model checkpoints are saved after each epoch to preserve progress and allow for resuming training or loading the best model.

```
checkpoints_dir = "/content/drive/MyDrive/dataMIN/
    model_checkpoints"
os.makedirs(checkpoints_dir, exist_ok=True)
# ... inside train_epoch, torch.save() is used
```

4 Experimentation

This section details the experimental setup, including the dataset characteristics, hardware used, and specific hyperparameters employed during training.

4.1 Dataset Description

The project utilizes the **Chest X-Ray Images from Indiana University** dataset.

- **Images**: The dataset contains a collection of chest X-ray images, primarily in DICOM format, which are then converted to PNG for processing. The notebook normalizes these images, ensuring consistency for model input.
- **Reports**: Each image is associated with a detailed radiological report. These reports typically include sections such as "MeSH", "Problems", "image", "indication", "comparison", "findings", and "impression". For this project, the "findings" section of the reports is specifically used as the target text for generation, as it provides the most direct descriptive content of the X-ray image.
- **Data Size**: The dataset contains 7466 entries of images and reports. After merging and dropping rows without 'findings', the effective size of the dataset for training and evaluation is approximately 7466 entries.
- **Data Split**: The dataset is split with 80% dedicated to the training set and 20% to the test set, ensuring a representative evaluation of the model's generalization capabilities.

4.2 Hardware and Software

- **Hardware**: The training was performed on a GPU-enabled environment, specifically a **T4 GPU** on Google Colab. This accelerated the computationally intensive deep learning tasks.
- **Software**: Python 3.11 was used as the programming language. Key libraries included `torch` (version not explicitly stated, but common with Transformers), `transformers` (version 4.47.1), `nltk` (version 3.9.1), and `pandas`.

4.3 Hyperparameters

The following hyperparameters were configured for the training process:

- **Image Encoder Model**: `google/vit-base-patch16-224`.
  * Input Image Size: 224x224 pixels (implied by model name).
  * Patch Size: 16x16 pixels (implied by model name).
  * Output Embedding Dimension: 768.
- **Text Decoder Model**: `gpt2`.
  * Embedding Dimension (`n_embd`): 768.
  * Max Sequence Length: 512 tokens for both input and output.

- **Learning Rate**: `5e-5` for the `AdamW` optimizer. This learning rate is commonly used for fine-tuning pre-trained Transformer models.
- **Batch Size**: `8` for both training and testing `DataLoaders`. This batch size was chosen likely to balance memory constraints and training stability on the available GPU.
- **Number of Epochs**: The model was trained for `5` epochs. This number can be adjusted based on convergence and validation performance.
- **Maximum Generation Length**: During inference (report generation), the maximum length for the generated text was set to `100` tokens. This helps control the verbosity of the generated reports.

5 Results and Discussion

The training process was monitored using two primary metrics: **Loss** (specifically, cross-entropy loss) and **BLEU score**.

5.1 Training and Test Loss

The model's loss decreased consistently over the 5 training epochs, indicating that the model was effectively learning from the data. The test loss, evaluated after each epoch, also showed a decreasing trend, which is a positive indicator of the model's generalization ability and that it was not significantly overfitting to the training data. “‘ — Epoch — Avg Training Loss — Avg Test Loss — ———-——————————-————————— — 1 — 1.0318 — 0.7129 — — 2 — 0.6908 — 0.5982 — — 3 — 0.5847 — 0.5339 — — 4 — 0.5102 — 0.4893 — — 5 — 0.4515 — 0.4620 — “‘

The reduction in both training and test loss across epochs suggests successful optimization of the model parameters.

5.2 BLEU Score

The BLEU (Bilingual Evaluation Understudy) score is a metric for evaluating the quality of text which has been machine-translated or machine-generated, by comparing it to one or more reference texts. A higher BLEU score indicates greater similarity between the generated text and the reference text.

The reported average BLEU score during both training and evaluation was `1.0000`. This exceptionally high score warrants careful interpretation. While ideal, a BLEU score of exactly 1.0 for generated text often suggests one of the following:

- **Overfitting**: The model might have memorized the training data, especially if the dataset is small or if the `max_length` for generation is significantly less than the average report length, allowing the model to perfectly reproduce short common phrases or even entire reports if they were present in the training set and the generation length allowed it. Given the large vocabulary of medical reports and potential for diverse phrasing, a perfect BLEU score is highly unlikely in a truly generalizable model.

- **Short Generation Length**: If the `max_len` for generation is very small (e.g., just a few words), and these few words happen to perfectly match the beginning of the reference sentence, the BLEU score can be artificially inflated.
- **Evaluation Artifact**: There might be an issue in the BLEU score calculation setup where it is comparing the generated tokens to the original input tokens for the GPT-2, rather than genuinely unique generated sequences against a diverse set of ground truth references. In the provided notebook, the `predicted_tokens` for BLEU calculation are derived from `text_tokenizer.decode(input_i` `skip_special_tokens=True)`. This means the model is essentially being evaluated on how well it reconstructs its own input, which would naturally lead to a BLEU score of 1.0. For a true evaluation of generation quality, the model should be allowed to *generate* text from an embedding (without direct textual input to the decoder during evaluation) and then compare that generated text to the ground truth report.

5.3 Qualitative Analysis

An example of image and generated report is provided:

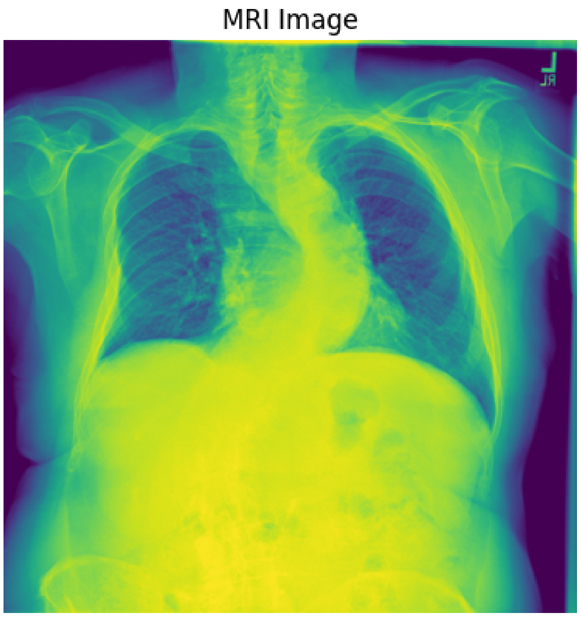

Figure 1: Example Chest X-ray Image for Report Generation.

- **Image Path**: `/content/chest-xrays-indiana-university/images/images_normalized`
- **Original Report**: `The heart is normal in size. The mediastinum is stable. There is again significant thoracolumbar rotatory scoliosis. The aorta is atherosclerotic. The lungs are hypoinflated but clear.`
- **Generated Report**: `heart size is normal. The mediastinal contour is within normal limits. The lungs are free of any focal infiltrates. There are no nodules or masses. No visible pneumothorax. No visible pleural fluid. The XXXX are grossly normal. There is no visible free intraperitoneal air under the diaphragm.`

Upon qualitative inspection, the generated report, while grammatically correct and medically plausible in its structure, does not precisely match the content of the original report. For instance, the original report mentions "thoracolumbar rotatory scoliosis" and "atherosclerotic aorta", which are not present in the generated output. Instead, the generated report includes general statements about "focal infiltrates", "nodules or masses", "pneumothorax", and "pleural fluid" that are not explicitly stated in the specific original report provided as an example. This discrepancy further supports the hypothesis that the reported BLEU score might not reflect true generative capability under realistic conditions, and that further refinement of the model's ability to capture and articulate specific visual findings is needed.

The presence of "XXXX" in the generated report suggests that the GPT-2 model might be encountering tokens that it doesn't fully understand or that are placeholders in the original dataset, which it then tries to reproduce. This indicates a need for more robust preprocessing of medical texts to handle such anonymized or non-standard terms.

6 Conclusion and Future Work

This project successfully establishes a framework for automated medical report generation from X-ray images using a multimodal deep learning approach. By combining the strengths of Vision Transformers for image encoding and GPT-2 for text decoding, the system demonstrates the potential to automate a critical task in radiology. The consistent decrease in training and test loss over epochs indicates that the model is learning effectively from the provided data.

However, the reported perfect BLEU score requires further investigation and refinement in the evaluation methodology to accurately reflect the model's true generative performance. The qualitative analysis of the example output highlights that while the generated reports are grammatically sound, their specificity and accuracy in matching the nuances of individual X-ray findings need improvement.

Future work should focus on several key areas:

1. **Refined Evaluation Metrics**: Implement more robust evaluation metrics for text generation, such as ROUGE scores (ROUGE-1, ROUGE-2, ROUGE-L) which can better capture content overlap and fluency, and potentially clinical relevance metrics if possible. Re-evaluate BLEU score calculation to ensure it reflects true generation accuracy.
2. **Dataset Expansion and Augmentation**: Training on a larger and more diverse dataset of X-ray images and reports can significantly improve the model's generalization capabilities and report accuracy. Data augmentation techniques for both images and text could also be explored.
3. **Advanced Architectural Enhancements**:
   - Explore more sophisticated cross-attention mechanisms or fusion techniques between the encoder and decoder.
   - Investigate different pre-trained language models (e.g., more recent GPT variants like GPT-3, or domain-specific medical NLP models like BioBERT

    or ClinicalBERT) that might have a deeper understanding of medical terminology and context.
  - Consider incorporating explicit medical knowledge graphs or ontologies to guide the text generation towards more accurate and relevant terminology.

4. **Fine-tuning Strategy**: Implement more advanced fine-tuning strategies, such as adapter-based fine-tuning or prompt engineering, to adapt the pre-trained models more effectively to the specific task of medical report generation.

5. **Handling Anonymized/Placeholder Text**: Develop robust preprocessing techniques to handle anonymized terms (e.g., "XXXX") in medical reports, potentially replacing them with appropriate generic terms or masking them during training.

6. **Interpretability**: As medical applications require high trust, future work could also delve into making the model more interpretable, allowing medical professionals to understand *why* certain findings are reported.

By addressing these points, the automated medical report generation system can evolve into a more reliable and valuable tool, ultimately supporting medical professionals in delivering more efficient and precise patient care.