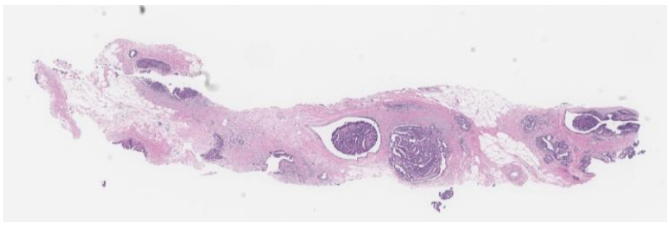
A TECHNICAL REPORT ON

Pathology Report Generation for Gigapixel-Scale WSIs Using Gemma-3n and Vision Encoders

---

Google - The Gemma 3n Impact Challenge



Submitted by

Aneesh Mukkamala

PROBLEM STATEMENT

Medical image understanding in the domain of pathology is a field of growing significance, driven by the increasing availability of high-resolution giga-pixel scale whole slide images (WSIs) and the need for scalable and accurate diagnostic solutions. In clinical practice, expert pathologists typically examine WSIs and then compose concise diagnostic reports that guide treatment planning. Automating this workflow, especially the generation of text-based summaries or labels from WSIs presents a challenging problem that requires leveraging solutions that integrate both computer vision and natural language processing. The **REG2025 Grand Challenge**, titled *Pan-Asia Whole Slide Image Pathology Report Generation*, is a competitive initiative that focuses on this exact problem. The task is to build models capable of producing short diagnostic reports from raw histopathology slides, covering a diverse range of anatomical sites. However, a major constraint imposed by this challenge is that the submitted solutions must be entirely **CPU**-compatible, without any hardware acceleration. This is motivated by the practical need for deployment in extremely low-resource clinical settings. While this constraint is both realistic and forward-thinking, it severely restricts the use of modern large language models **(LLMs)** or vision-language models **(VLMs)** which generally require GPU acceleration even for inference, let alone training. This constraint limits the ability to leverage the power of VLMs and LLMs, despite their proven adaptability in producing impactful solutions across diverse domains. As a result, there is a significant risk of overlooking models with the potential to contribute strong and effective solutions in clinical settings which have chances of unlocking an impactful and reliable solution. As a result, I chose not to participate nor to curate a solution that fits the requirements to **REG2025**, since the solution I was developing would not comply with this "CPU only" requirement. Instead, I chose to retain the same **problem statement** (generating medical reports from WSIs) and curate an even more advanced solution under the **Google Gemma-3n Hackathon** utilising Gemma-3n, custom Vision transformers, CNN and a lot more models all working together to generate a well performing system with promising benchmarks for future research and developments. The main reason why I choose to adapt the problem statement is to test and explore the power and capabilities of state-of-the-art language models like Gemma-3n for tasks having such complexity. Given the multi-modal nature of Gemma-3n, the solution presented in this report focuses exclusively on leveraging its NLP components (language-specific layers and weights) and to test its strength while developing and integrating custom vision models for training. This approach was taken because the default image backbone of Gemma-3n is optimized for general purpose imagery and is not well-suited for the complexity and domain specific characteristics of pathology images and slides. I have explained the complete workflow, including data preparation, model architecture, training procedure, the math behind the training and the forward pass of data through all the models used in detail. Further insights can be found in this report.

With that, proceed to subsequent sections for a more detailed walkthrough. Happy reading!!

Table of Contents

1.Introduction

Whole-slide images (WSI) are revolutionizing digital pathology by enabling the capture, analysis, and sharing of entire histopathological slides at microscopic resolution. Unlike traditional medical images (e.g., chest X-rays or MRIs), which typically represent a macroscopic region in a few megapixels, WSIs are gigapixel-scale images often exceeding dimensions of 100,000 × 100,000 pixels because they aim to preserve cellular-level and even subcellular-level detail across entire tissue sections of body organs.

These WSIs are commonly stored in the .tiff or .svs file formats, often using pyramidal tiling structures internally. Pyramidal tiling is a storage technique where the image is saved at multiple resolutions in a pyramid-like hierarchy, with the highest resolution at the base (level 0) and progressively lower resolutions at higher levels. However, many publicly available datasets (including the one used in this project) provide WSIs with only a single resolution level (level 0) meaning the image is available at its highest native resolution without any down sampled levels. This makes them extremely large, both in terms of storage (hundreds of megabytes to several gigabytes per image) and memory footprint during processing. The data volume and resolution make WSIs uniquely challenging for machine learning tasks which include tasks like reading, tiling, augmenting. Feeding such images to deep learning models is computationally intensive, requiring thoughtful engineering and resource management. Due to the large sizes, images are loaded and processed in form of patches which involves reading only a certain region of the entire WSI. This area of region that is loaded for reading from a WSI depends on the RAM available as larger patches require more memory. For this project, a minimum of 30 Gb is needed to read the entire image at once in the form a NumPy array. While 30 Gb was a decent amount of RAM for easy EDA and other tasks, higher memory significantly speeds up the process and this does not require any hardware acceleration like GPUs or TPUs, better CPUs handle data processing thoroughly.

Despite these challenges, WSIs are invaluable for computational pathology. They contain rich spatial information and textural features and some popular uses of WSIs include diagnosis for gland formation in prostate cancer, detecting mitotic figures in breast cancer, detecting infectious pathogens in tissues, assessing transplant rejection of various body parts, image segmentation of cancer tumours and detecting early onset of cancerous cells in body parts and many more.

The goal of this project is to explore how we can use these high-resolution WSIs with large language models (LLMs) and other vision models to generate short, pathology-style diagnostic reports. Unlike conventional classification tasks of WSIs (using LLMs and vision models) involves use cases where it deals with predicting a single label (generally finding the corresponding organ linked to the WSI), this project involves automating and generating 1–2 line text reports that correspond to the descriptions of the WSI which are usually generated by certified medical professionals or pathologists.

The project covers many new ideas to perform multi modal training of an LLM without using its image components and purely relying on the overall power of the LLM by solely using its language layers. Along with this, a new approach for training LLMs through this pathway is curated which significantly outperforms state of the art methods like CLIP and SigLIP to solve the problem statement of this project which is highly domain specific and is not generalized with respect to the LLMs training data.

2. Data Preparation Phase

Working with whole-slide images (WSIs) presents a unique set of technical challenges due to their massive scale, irregular dimensions, and class imbalance across anatomical sites. This section outlines the steps taken to acquire, preprocess, and organize the pathology image dataset used for the project and other technicalities that were handled

2.1 Data downloading and exploratory data analysis (EDA)

The dataset consisted of 8,600 whole-slide TIFF images hosted on an FTP server, totalling over 1.6 TB (Terra-Bytes) of raw data. Each image was stored at level 0 resolution only, meaning no pyramidal down sampling was applied; every slide was preserved at full resolution.

Each whole-slide image (WSI) was accompanied by a short textual report, typically one to two lines in length, formatted as:

"Prostate, biopsy; Acinar adenocarcinoma, Gleason's score 7 (4+3)".

To perform EDA, organ label/names were extracted by string patterns and regex functions. A dictionary was then constructed from these labels to understand organ occurrence across the full set of 8,600 slides.

The resulting distribution of organs from the dictionary revealed visible class imbalance. Organs such as "**Prostate**" and "**Breast**" were found to be heavily overrepresented, while others had limited representation. To counter this issue and to prevent unstable training of the models, a target class distribution dictionary was defined, specifying upper bounds on the number of samples per organ. Based on this simple rule, a filtered subset of 5,650 slides was selected to ensure more balanced representation across organ types. This selection was also guided by hardware and storage limitations. To reduce download time and manage storage requirements, metadata including filenames and file sizes was first retrieved and sorted in ascending order by size of the files. The filtered 5,650 files storage totalled to 737 GB.

The downloading was conducted using Kaggle notebooks, each offering a maximum of 19 GB of local disk storage. The main reason behind choosing Kaggle is due to the high network bandwidth in the kernels which significantly accelerated the download of data amounting to 737 GB. A simple monitoring logic was used to track disk usage during downloads. Once the 19 GB limit was reached, downloaded files were immediately uploaded to Hugging Face, local storage was cleared, and the loop resumed with the next batch of files that totalled 19 GB. Failed downloads were tracked and retried to ensure completeness. To further accelerate the process, multiple Kaggle notebooks were executed in parallel, with each handling approximately 1,000 files. This distributed strategy resulted in a 5-x speedup compared to sequential local downloads and the dataset which was finally hosted on Hugging Face provided convenient access to the data during training. The same data is published on Kaggle also.

Furthermore, during downloading and data processing of the filtered 5600 files, many files had either repeated errors while downloading from the FTP server or the available memory did not support loading some large WSI files. Finally, 5015 files were successfully downloaded and processed for training.

Visualizations of the organ distributions of the original (all WSI files), filtered and of the final files excluding the files which did not go through the data preparation phase are provided in Appendix **A1**.

2.2 Data processing

The Open Slide Python library was used to read metadata from each slide and extract level 0 dimensions (width and height). The height and width refer only to the spatial dimensions of the image and not the colour channel information (RGB or grey scale). Detailed statistics of the width and height across all files can be found in Appendix **A2** and **A3**.

The actual pixel-wise information is retrieved using Open Slide which returns a PIL Image object in RGBA mode by default where the fourth channel "a" represents "alpha" which corresponds to the amount of transparency of the WSI organ's tissue content relative to the background. For training purposes, the transparency channel is not used in general and thus the images are converted to RGB format.

The resulting RGB images' dimensions are still in giga-pixel ranges. The slides were processed with the below two different approaches.

2.2.1) Resizing to Median Dimensions with Patch Extraction:

A fixed resolution of 20,500 along width and 20,500 pixels height dimension was selected based on the average spatial dimensions observed across all slides during EDA which excludes outliers (WSI with much variance with respect to the mean dimensions). The width and height of all raw WSIs varied approximately between 10,000 and 100,000 pixels. Slides with dimensions smaller than the median were upscaled, and those larger were downscaled using bicubic interpolation to standardize input sizes. Following resizing, a fixed-size sliding window of 256×256 pixels was applied across the image to extract non-overlapping patches. These patches were indexed and stored. The overall mean of the width and height dimensions are approximately 38,000 x 22,000 but to ensure that the inputs to the image models have a uniform shape across all dimensions, the number 20,500 to was chosen across both the dimensions.

2.2.2) Direct Resizing to 256×256:

In this approach, entire WSI was directly resized to a 256×256×3 RGB image using **Lanczos interpolation algorithm**, a high-quality resampling method known for preserving sharpness during downscaling. Although this level of down sampling is extreme and may initially appear unsuitable for medical imaging tasks, visual inspection confirmed that key structural features and textures were largely preserved. Unlike in the earlier method, no patches were generated using sliding window.

A sample image of a WSI (before and after processing/resizing) is shown in Appendix **A4**. Both methods were used to experiment with a wide range of model architectures for training Further comparisons and justifications are discussed in the next section.

3. Training phase

This section covers details of all models used, the training setup, the optimization, loss function, improved training approach compared to CLIP and SigLIP and other related components. Selecting the image models were done on the training accuracies on a classification task which required the models to classify the downscaled WSI images to their corresponding organ. This was done as a fine-tuning strategy to align the weights of the image model to the WSI data. Later, after this stage of training during integrating the model with the LLM, the classification head was removed and the image model's feature vectors/embeddings of the downscaled WSI are used. The classification ensured that these embeddings had reasonable separation so that training of further components in the architecture mentioned later would become easier. Details of the same can be found in this section as well.

3.1 Vision Transformers (ViT)

Vision Transformers (ViTs) have been proven to be a powerful alternative to convolutional networks by modelling image patches as token sequences processed through self-attention mechanisms. This makes them well-suited for tasks that benefit from sequence modelling and were chosen as the first choice for the image components in this project. Several variants and methods of ViTs which were explored can be found below.

3.1.1 ViT on Full-Slide Inputs (~25k×25k)

An initial approach involved training a Vision Transformer (ViT) model from scratch using full-resolution resized whole-slide images (approximately 25,000 × 25,000 pixels). To handle the extremely large spatial dimensions, a patch-based attention mechanism was made with large patch sizes using the stored 256-pixel ×256-pixel shaped patches which were extracted using sliding window as in Section **2.2.1**. This architecture enabled the model to attend to spatially distant yet semantically related tissue regions by leveraging the sliding-window patches stored during preprocessing.

Despite its theoretical appeal, this approach encountered severe computational limitations., Memory consumption during both training and inference exceeded 40 GB per image, resulting in high latency for processing a single slide. Initial experiments involved passing slides through the untrained ViT model to evaluate memory and latency requirements, confirming these constraints. Consequently, this strategy was not pursued further for model development or deployment.

3.1.2 Vision Transformer (ViT) on Standardized 256×256 Inputs

To achieve a balance between resolution and computational efficiency, a second category of ViT models were trained on fixed size 256×256 RGB images as obtained in Section **2.2.2**. The following two variants were explored.

1) **From-Scratch ViT:** Custom ViT architectures were designed and trained entirely from scratch using 256×256 size WSI obtained from Section **2.2.2**. This baseline aimed to evaluate the performance of transformer-based models trained without external supervision. Theoretically, training a base vision transformer model completely from scratch requires extensive amounts of data due to the way transformer models are engineered as the weights of the model's layers centred around attention mechanism rely on larger batch sizes during training. This approach was attempted on the limited 5015 samples and the training did not show significant improvement.

2) **Pretrained ViT (TIMM):** A pretrained ViT model (**vit_base_patch16_224**) from the TIMM library was fine-tuned by using a lightweight classification head configured with num_classes = 8, corresponding to the number of organ categories. During fine-tuning, gradients were updated through both the classification head and the ViT backbone. The training is done with the same data used for training the previous mentioned ViT made from the ground up using data from Section **2.2.2**

3.2 Convolutional Neural Networks (CNNs)

In parallel with Vision Transformers, convolutional neural networks (CNNs) were also explored due to their proven effectiveness in modelling localized spatial patterns. This is particularly relevant in medical imaging, where textures, edges, and morphological features are critical for accurate classification.

Unlike the some of the ViT-based models listed earlier, no CNNs were not applied to full-resolution (20.5k × 20.5k) inputs or to large-scale sliding window patch extraction. Instead, all CNN experiments were conducted using standard 256×256 RGB patches directly extracted from WSIs, ensuring computational efficiency and alignment with common CNN input dimensions.

3.2.1 CNNs Trained from Scratch

Like the earlier section, custom CNN architectures were implemented and trained from scratch using the 256×256 patches. The model consists of multiple convolutional blocks with batch normalization and ReLU activations, followed by spatial down sampling via max pooling layers. After feature extraction, the resulting feature maps were flattened and passed through fully connected layers, finally into in a SoftMax classifier for 8 class organ prediction.

3.2.2 Pretrained CNNs

Pretrained convolutional neural network (CNN) models from the TIMM library were fine-tuned on 256×256 RGB patches extracted from the WSIs. Several architectures were initially evaluated based on training loss convergence, classification accuracy, and computational latency. Among the candidates, **mobilenetv4_conv_medium** and **efficientnet_b0** demonstrated the most promising trade-offs between efficiency and performance. Ultimately, **EfficientNet-B0** was selected as the final backbone due to its optimal balance of accuracy and resource usage. A lightweight classification head with num_classes = 8 was stacked to the **EfficientNet-B0** backbone to accommodate the organ classification task. The entire network, both the pretrained backbone and the classification head was fine-tuned end-to-end with full gradient updates, for ensuring adaptation of the model to pathology data. This stage of training is applied on the earlier ViT models as well with the same dataset form Section **2.2.2**

Among the various image models evaluated including both custom-built and pre-trained ViTs and CNNs, training was done for the 8-organ classification task. While ViT-based models, particularly the TIMM ViT, achieved the highest test accuracy, the **TIMM CNN backbone** was selected for further stages of the project. This decision was made based on several trade-offs although the ViT models showed superior performance in accuracy, they typically require **higher memory**, have **slower inference times**, and are less efficient on smaller datasets due to their data-hungry nature. Given the limited number of examples in the dataset and small test set, model generalization was a key concern. Instead of selecting the highest accuracy model, the **second-best** model in terms of test loss **TIMM CNN** was chosen. Following the model selection and the classification fine-tuning, the classification head was removed, and the **forward-features** method of the **TIMM CNN (EfficientNet-B0)** was used to extract dense, high-dimensional visual embeddings from each 256×256 image from Section **2.2.2**. These embeddings were subsequently passed through a trainable projection module, which transformed them into tensors of shape (B, seq_len, embed_dim) the format required by downstream language model (LLM) decoder blocks for multimodal integration. This trained **EfficientNet-B0** was incorporated into the next and **final training stage**, where the

visual and textual modalities were combined. Given the significant computational cost of large-scale LLM training, these preliminary evaluations were critical in minimizing redundant experimentation at the multimodal stage. The design of the projection modules and the details of its role in integration of image embeddings and text embeddings with the LLM without contractual learning are described in the following sections.

The training curves of **timm based CNN model** can be found in Appendix **A5**
The training curves of **timm based ViT model** can be found in Appendix **A6**
The training curves of **custom CNN models** and their variants can be found in Appendix **A8**
The training curves of **custom ViT models** and their variants can be found in Appendix **A9**

A table depicting the training, validation and test metrics of all image models/backbones can be found in Appendix **A7**.

3.3 Embedding Projector and Multimodal Integration

In this section, details regarding the complete model architecture involving LLM integration, projector modules to support multi-modality for the WSI images is covered. Details regarding training configurations, hardware specs, optimization and the overall training workflow for the final phase of training is described

3.3.1 Model Architecture Overview & Input Processing

The model uses four primary components working together. **EfficientNet-B0** serves as the frozen feature extractor for processing input images into visual embeddings without any parameter updates during training (final phase of training that involves the LLM). The **main projector** (**Projector4to3d**) transforms these visual features into language model compatible embeddings through spatial attention mechanisms. **Gemma-3B language model** with **LoRA** adapters handles the generative aspects. Additionally, **contrastive projection heads** (image and text heads) create a pathway for semantic alignment between visual and textual representations using CLIP-style training.

The architecture operates through two simultaneous learning mechanisms during this phase. The primary generative pathway concatenates visual embeddings with text prompt and ground-truth tokens, training the language model to predict medical descriptions autoregressively. This handles the core task of medical report generation. In parallel, the contrastive alignment heads extracts global representations from both visual patches and text embeddings, projecting them into a vector like space where matching image-text pairs are pulled together while non-matching pairs are pushed apart. ARC face-loss in face detection systems uses this type of functionalities and my current idea is slightly like this.

The downscaled WSI images enter as 256×256 patches and undergo feature extraction through the frozen **EfficientNet-B0** backbone, producing spatial feature maps or for simplicity they can be interpreted as image embeddings. Simultaneously, text inputs including both the instruction prompt ("Describe the medical image:") and ground-truth reports are tokenized and embedded through the LLM. The main projector (**Projector4to3d**) processes visual features into 32 compressed patch tokens that align dimensionally with text embeddings, enabling concatenation with text embeddings for the generative task. Meanwhile, global representations are derived from these same features using contrastive projection heads to support the contrastive learning objective.

3.3.2 Visual Feature Extraction & Main Projection Pipeline

EfficientNet-B0 Feature Extraction

The frozen **EfficientNet-B0** backbone processes input medical images through its convolutional layers, transforming 256×256×3 RGB patches into dense feature representations. The backbone's forward-features() method outputs spatial feature maps of shape (B, 1280, 8, 8), where each of the 64 spatial locations (8×8 grid) contains a 1280-dimensional feature vector encoding the images.

Main Projector Architecture (Projector_4to3d)

The main projector transforms CNN features into language model embeddings through a multi-stage process combining spatial reasoning, cross-modal attention, and token compression. The architecture begins with spatial rearrangement using **einops rearrangement**:

(B, 1280, 8, 8) → (B, 64, 1280), converting the 2D feature grid into a sequence of 64 spatial tokens. Learnable positional embeddings are added to each spatial location, helping the model understand geometric relationships between different image regions.

Projection and Attention Mechanisms

The mentioned input projection layer maps visual features to the language model's embedding dimension:

Linear (1280) → LayerNorm → ReLU → Dropout → (B, 64, 2048)

This projection aligns visual and textual representations in the same semantic space. Multi-head spatial attention (8 -16 heads) enables reasoning across different image regions, allowing the model to identify relationships between anatomical structures. When available during training, cross-attention with text embeddings facilitates direct alignment between visual patches and corresponding textual descriptions, enhancing the model's understanding of image-text correspondences. Furthermore the 64 image tokens are compressed to 32. The idea behind the compression from 64 → 32 is intentional as this is roughly equal to or less than the number of the prompt + label text embeddings. This would help align the training to map image tokens with the text tokens.

Feed-Forward Processing and Token Compression

A transformer-style FFN with expansion ratio 4 processes the attended features:

Linear (2048 → 8192) → GELU → Dropout → Linear (8192 → 2048)

surrounded by residual connections and layer normalization.

Complete Shape Transformation Flow

Here are the complete transformations of the visual information through the following tensor shapes for easier interpretability:

- Input: (B, 256, 256, 3) medical image patches
- CNN Features: (B, 1280, 8, 8) spatial feature maps from **EfficientNet-B0**
- Spatial Tokens: (B, 64, 1280) after **einops** rearrangement and positional encoding
- Projected Features: (B, 64, 2048) after input projection to LLM dimension
- Attended Features: (B, 64, 2048) after spatial and cross-attention processing
- Final Output: (B, 32, 2048) compressed visual embeddings ready for concatenation

3.3.3 Multimodal components & Embedding concatenation

Prompt Embedding Creation and Buffering

The instruction prompt "Describe the medical image:" is tokenized and converted into embeddings through the language model's vocabulary. This produces approximately P = 6 tokens with shape (1, P, 2048), which are stored as a registered buffer prompt_embeddings to avoid recomputation during training as the same/fixed prompt is used across all examples in the dataset. The same prompt is used during inference as well to maintain consistency. The embeddings are expanded to match the batch size, yielding (B, P, 2048) for concatenation across all examples.

Text Tokenization and Embedding Strategy

All labels/reports of the WSI are tokenized where each text sequence is appended with an EOS token before processing. The tokenizer applies padding and truncation to a fixed length. ensuring consistent sequence lengths within each batch. These tokenized sequences are converted to embeddings through the language model's input embedding layer, producing text representations of shape (B, T, 2048) where T corresponds to the fixed sequence length of the labels. EDA revealed that medical reports typically contain around 30 tokens on average with a maximum of 59 tokens. The value of T is set as 50 and not the max length of 59 as this was done to remove a few outliers as only 5 examples had label tokens length between 50 and 59

Embedding concatenation

The final input sequence combines three embedding types along the sequence dimension to create a comprehensive multimodal representation. Prompt embeddings (B, P, 2048) provide consistent instruction conditioning, compressed visual embeddings (B, 32, 2048) from the main projector encode spatial medical images, and text embeddings (B, T, 2048) contain the target medical descriptions/labels. The concatenation produces sequences of shape

(B, P + 32 + T, 2048), typically around (B, 88, 2048).

3.3.4 Multimodal components & Embedding concatenation

Primary Language Modelling Loss

The concatenated multimodal sequence flows through the Gemma language model to produce logits of shape (B, P + 32 + T_max, vocab-size). The model performs standard autoregressive language modelling, where each position predicts the next token based on all preceding tokens in the sequence. Cross-entropy loss is computed exclusively on text token positions using the label masking strategy, where prompt and visual tokens are ignored (label = -100) and only actual medical report tokens contribute to the gradient. This ensures the model learns to generate medical descriptions conditioned on both the instruction prompt and visual context, while avoiding reconstruction of input conditioning information as mentioned earlier.

Contrastive Learning Branch: Beyond CLIP and SigLIP

The contrastive learning component draws inspiration from CLIP but introduces several key improvements tailored for medical multimodal understanding. Global representations are extracted by averaging spatial tokens using contrastive projection heads (image and text heads)

image-global = patch-embeddings.mean(dim=1) produces (B, 2048) vectors from the 32 compressed visual tokens, while

text-global = text-embeddings.mean(dim=1) creates (B, 2048) representations from variable-length text sequences.

Enhanced Contrastive Projection Architecture

Unlike CLIP's simpler projection layers, this implementation uses contrastive projection heads with intermediate activation functions. Both image_projection_head and text_projection_head follow the architecture:

Linear (2048 → 512) → ReLU → Linear (512 → 256), creating normalized 256-dimensional embeddings in a shared semantic space.

The learnable temperature parameter

self.temperature = nn.Parameter(torch.ones([]) * np.log(1 / 0.07))

enables adaptive scaling of similarity scores, initialized to CLIP's standard value but allowing optimization during training.

CLIP-Style Contrastive Loss with Improvements

The contrastive loss follows CLIP's bidirectional formulation but operates in a multi-task learning context. Similarity scores are computed as

logits = torch.matmul(image-proj, text-proj.t()) * temperature.exp()

creating a (B, B) matrix where diagonal elements represent positive pairs.

The symmetric contrastive loss:

(F.cross-entropy(logits, labels) + F.cross-entropy(logits.t(), labels)) / 2

ensures both image→text and text→image retrieval capabilities. However, unlike standalone CLIP models, this contrastive objective serves as an auxiliary loss that enhances alignment without compromising generative performance.

Multi-Task Learning Integration

The final loss combines both objectives strategically:

total_loss = language loss + 0.1 * symmetric_contrastive_loss.

This weighting factor (0.1) prioritizes the primary generative task while using contrastive learning to improve semantic alignment between visual and textual representations. This approach differs significantly from CLIP, which uses contrastive learning as the sole objective, and from SigLIP, which employs sigmoid based pairwise classification.

Training vs Inference Mode Differences

During training ("forward mode" in the complete model wrapped as **PyTorch lightning** module) both losses are computed using ground-truth text, enabling the model to learn both generation and alignment simultaneously. The contrastive branch receives actual image-text pairs from the training batch, creating positive and negative examples. During inference (generate mode), only the generate function (another "mode" like "forward" as per pytorch lightning) operates, using the concatenated prompt and visual embeddings to produce text autoregressively without contrastive loss computation.

Advantages Over Traditional Contrastive Approaches

This implementation surpasses traditional CLIP/SigLIP approaches for medical multimodal tasks through a few tricks and additions that are listed above. For this project, which is very new and domain specific, CLIP and SigLIP did not perform well at all. The responses of models trained using CLIP and SigLIP were broken and do not have semantic at all. A similar workflow explanation of layer-wise forward pass of models trained using traditional CLIP and SigLIP can be found in the code files that are tagged in Appendix A10. The model's responses can be found in the jupyter notebook code files itself and the explanations are written out as comments in the code.

3.4 Training configuration and Hyperparameters

3.4.1 Attention Masking Strategy

An attention mask of shape (B, P + 32 + T_) is made to handle prompt and image sequences and padding tokens. The mask contains 1's for tokens including all prompt tokens (P positions), all image tokens (32 positions), and actual text tokens. Padding positions receive 0's, preventing the model from attending to meaningless padding tokens during training. This masking strategy ensures that attention mechanisms focus only on meaningful content while maintaining computational efficiency across batched training.

3.4.2 Label Construction for Loss Computation

Label tensors are created with the same shape as the input sequence (B, P + 32 + T) to enable proper loss computation during training. The first P + 32 positions corresponding to prompt and visual tokens are set to -100 (HuggingFace's ignore index), ensuring that loss computation focuses exclusively on text generation rather than reconstruction of input conditioning. Actual text tokens retain their original token IDs as prediction targets, while padding positions are also set to -100 to exclude them from gradient computation. This labelling strategy enables the model to learn autoregressive text generation on both textual instructions and visual context, with the standard cross-entropy loss computed only over the target text tokens that the model should predict.

3.4.3 Different learning rates

The most important piece of the training involves choosing different learning rate for different modules in the architecture. This is the main reason why pytorch lightning was used as this supports seamless integration of this feature. Vision-related components having the main projector, image projection head, text projection head, and temperature parameter use a higher learning rate of "VM-LR" (vision modules learning rate) = 2e-4, enabling faster adaptation of visual feature processing. Language model parameters that require gradient updates use a more conservative LLM_LR = 2e-5, preventing loss of pre-trained weights while allowing adaptation to multimodal inputs.

The weight decay configuration applies differential regularization across components. Vision-related modules used weight_decay = 1e-4 to prevent overfitting in the visual processing, while the language model used lighter regularization with weight_decay = 1e-5 to preserve pre-trained knowledge. The temperature parameter receives no weight decay (weight_decay = 0.0), allowing unrestricted optimization of the contrastive scaling factor.

AdamW optimizer with eps = 1e-8 handles the multi-component parameter groups in pytorch lightning, providing adaptive learning rates with improved weight decay decoupling. The OneCycleLR scheduler implements learning rate scheduling with cosine annealing strategy. The configuration uses 10% warmup (pct_start = 0.1) to gradually increase learning rates during initial training phases, preventing gradient explosion in the early stages when different components are learning to work together.

The scheduler employs aggressive learning rate variation with div_factor = 25.0 and final_div_factor = 1000.0, creating a learning rate range that starts at max_lr / 25, peaks at max_lr, and decays to max_lr / 1000. This wide range enables the model to escape local minima during the peak learning phase while ensuring fine-grained optimization during the final convergence phase and a gradient clipping with global norm = 1.0 was also applied

In this final stage of training the **EfficientNet**-B0 backbone remains completely static during training, serving purely as a feature extractor without any parameter updates.

3.4.4 LoRA (Low Rank adaptation) usage

LoRA (Low-Rank Adaptation) is used to minimize computational overhead while maintaining reasonable training given the limited compute available on Kaggle. To maximise the performance and to ensure most learning occurs, a large rank of 256 was used with LoRA alpha to be 4 x the rank and LoRA was applied on the language layers. The audio and vision components of Gemma-3n remain untouched. To support this functionality, **Unsloth** was used to minimize configurational overhead from my side.

3.4.5 Computation

Unsloth and **Pytorch Lightning** are the main libraries used for the final stage of training. Kaggle offers **4 L4** GPUs (**4 x 24 = 96Gb RAM**) for notebooks that attach the ARC-AGI as a competition dataset. Since I have been also working on ARC-AGI, I allotted a few weeks of my Kaggle GPU time dedicated for this project. Kaggle also has other GPU offerings namely the 2 **T4** GPUs (2 x 15 = 30Gb RAM) and 1 **P100** GPU (**16 Gb RAM**)

Loading the LLM, the 3 projector modules and the image model required significant RAM and due to Pytorch Lightning's unstable and beta nature of multi-GPU support in interactive environments like jupyter notebooks (Kaggle/Colab), this posed as a real challenge. To counter this the LLM was loaded in **bfloat16** precision. After applying the LoRA adapters, the LoRA weights had **float32** precision and to avoid memory issues and precision errors, the complete LLM with LoRA weights was casted to **bfloat16**. Consequently, the concatenated embeddings were also casted to **bfloat16** while the 3 projector modules and the passage of activations through them and the image model remained in **float32**.

The **bfloat16** precision LLM required 15Gb of GPU RAM, and this was after casting the model to bfloat16. And during training to ensure good batch sizes, the batch size was set to 10 and this required close to 18Gb of GPU RAM. This made me to use a **single L4** GPU of the **4 L4** present on Kaggle.

Following the evaluation of the bfloat16 LLM and its promising results (as detailed in the next section), two additional variants were trained: a modified version of the bfloat16 model and a full float32 precision model. In the float32 variant, all computations were performed in float32, requiring approximately 46 GB of GPU memory. These training runs were conducted using the Modal cloud platform. The primary difference between these new Modal-based models (bfloat16 and float32) and the earlier version trained on Kaggle is an increased loss weighting factor from 0.1 to 0.4 which resulted in substantial improvements in model performance

A single NVIDIA L40S GPU was used for the **float32** model training, while the modified bfloat16 model was trained on an L4 GPU, also on Modal (as the weekly Kaggle resources had been exhausted). Both models were trained for 10 epochs and yielded improved results. These runs were conducted using Modal's monthly free credit plan available to all users. Due to the high memory requirements of the **float32** model, its testing was performed on Modal as well. Whereas the bfloat16 model trained on Modal was tested on Kaggle, as its memory footprint remained within the platform's constraints. Training logs for both the **bfloat16** and **float32** models are provided in Appendix **A10** and **A11**, respectively. These training plots track overall, language and contrastive losses at each training step.

NOTE: A minor bug in the code caused language and contrastive losses to not get tracked (for plotting curves and graphs) for the **bfloat16** model trained on Modal. However, training code of the float32 model was modified to ensure tracking all losses during training and this bug was fixed. And the training on Modal used **Transformers** library and not **Unsloth,** due to multiple errors related to **Unsloth's** package installation and CUDA compatibility issues.

4. Evaluation and Benchmarks

The trained image model, projector modules and the language model with the merged LoRA adapter was tested on 200 samples. The sampling parameters (generator-kwargs*) used a **temperature** of 0.4, **top-p** of 0.9 and **maximum number of output tokens** to be 150. **BERT Similarity score** and **Levenshtein Ratio** was used for comparing model's predictions with the ground truth WSI reports. The results of the final **bfloat16** and **float32** precision variants of LLMs trained on Modal are as follows.

MODEL	BERT SIMILARITY SCORE		LEVENSHTEIN RATIO	
	Max	**Mean**	**Max**	**Mean**
Bfloat16 variant	0.7669	0.4128	0.4839	0.2536
Float32 variant	0.8552	0.4886	0.597	0.2672

5. Future Scope and Conclusion

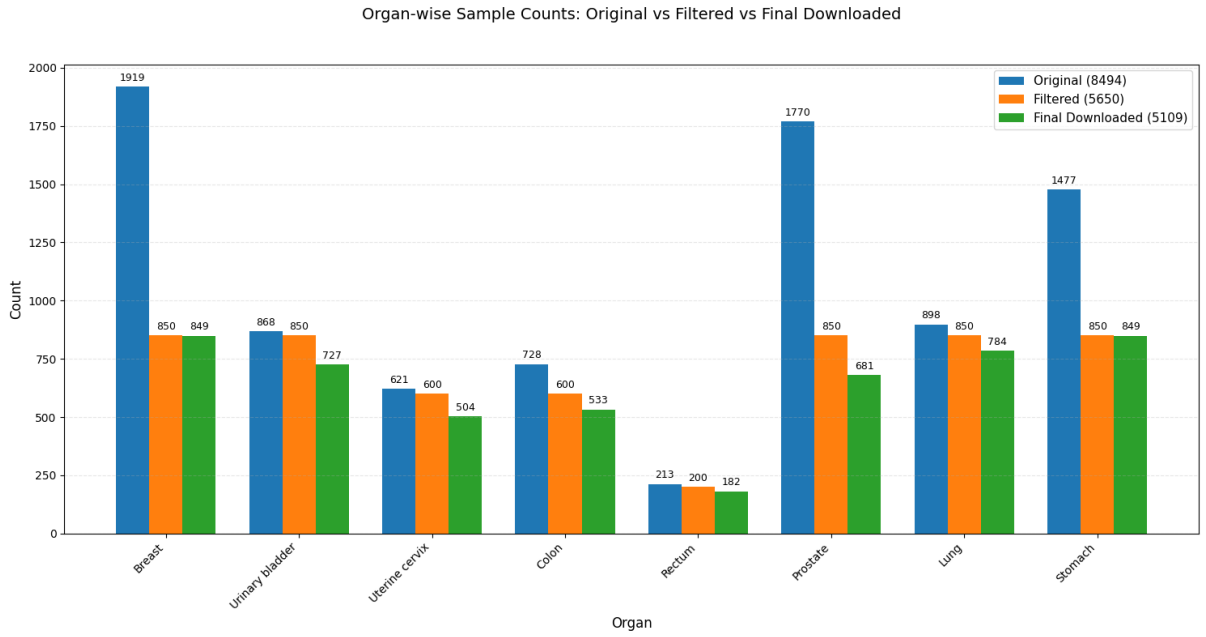
This project showcases the ability of generating diagnostic pathology reports from gigapixel WSIs using a vision + NLP only LLM architecture. Projects like this can also serve as valuable benchmarks for evaluating the overall domain adaptability strength of large language models.

Following the listed methods and techniques above, here are some future directions that can be explored for this project to develop solutions with greater performance

- Train full-resolution ViT models on patches like in **Section 3.1.1** (e.g., 256×256) from entire WSIs, enabling fine-grained attention across spatially distant yet semantically linked regions.
- Experiment with more image models from the TIMM library or coming up with newer/custom architectures as some image models achieved accuracies of 96% in this project
- Develop pathology-specialized vision backbones by pretraining or finetuning on histopathology datasets. Existing models like HE-Net, PathCNN, and BiT are promising but they are often non-open-source or require high compute
- Create pathology-specific CNNs using better domain-focused datasets
- Expand the dataset significantly beyond the current 5,000 WSIs to improve generalization and reduce overfitting. Exploring public datasets like TCGA, PanNuke, CAMELYON16, etc.
- Develop better variants of CLIP and SigLIP like I tried. Adding modality tokens like [**IMG**] or [**TEXT**] before concatenation of the embeddings might help the model process inputs better.
- Optimize the contrastive temperature parameter (currently set to ~0.07). Experiment with learned vs. fixed temperatures and study its effect on alignment performance.
- Tune sampling strategies during inference, including parameters like temperature, top-k, and top-p to maximize report quality and diversity.

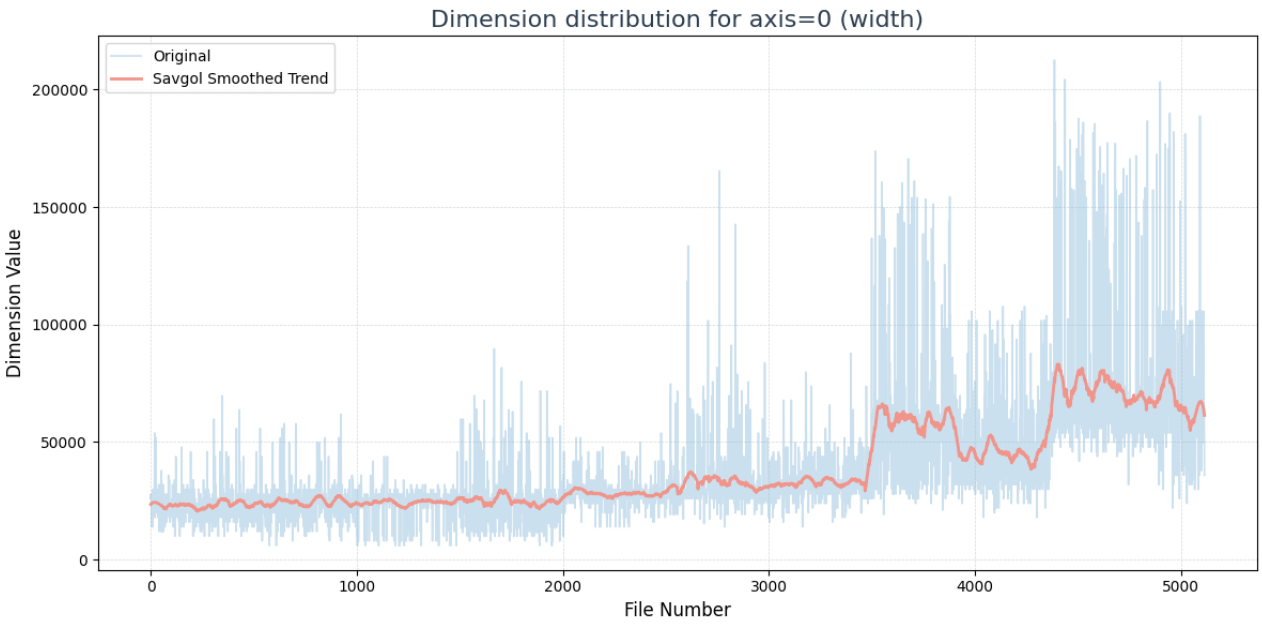
  For example, this notebook explains how to do it
- Experiment with different LLMs for better domain adaptation. Experiment with reasoning models and compare which LLMs best aligns for domain adaptation.
- Create tailored calibration datasets for quantization strategies (similar to AWQ, GPTQ) for fast, low-latency deployment of multimodal models.
- Explore optimal weighting of the contrastive loss in the final multi-task objective. The current 0.1 factor can be tuned dynamically based on training curves performance. Perform something like an ablation study to determine the best balance. Simply changing it to 0.4 yielded better responses.

6. Appendices

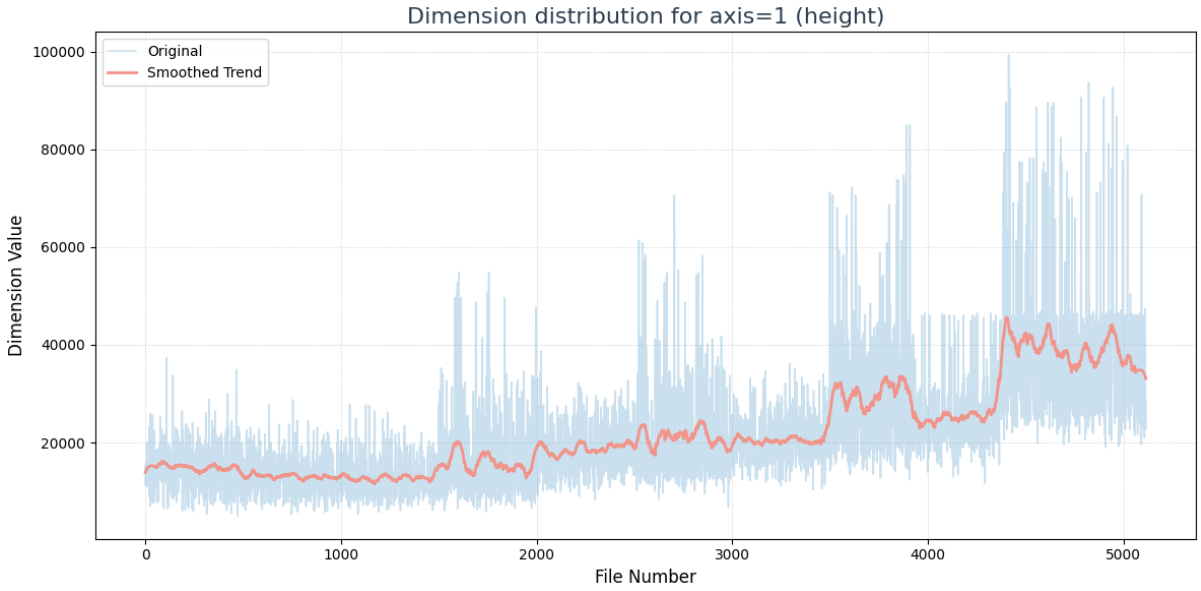


A1: Original, filtered and final counts of organs spread across all WSI files.

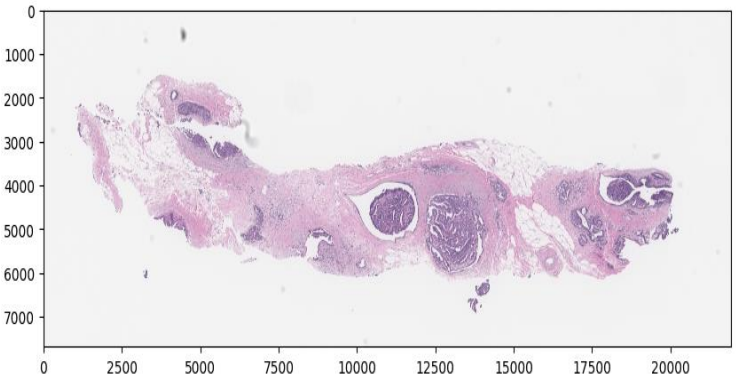
Note: Final counts/ number of files is less than actual targeted filtered count as some files had issues while downloading from the FTP server and some files were too large to load into the memory during data processing.

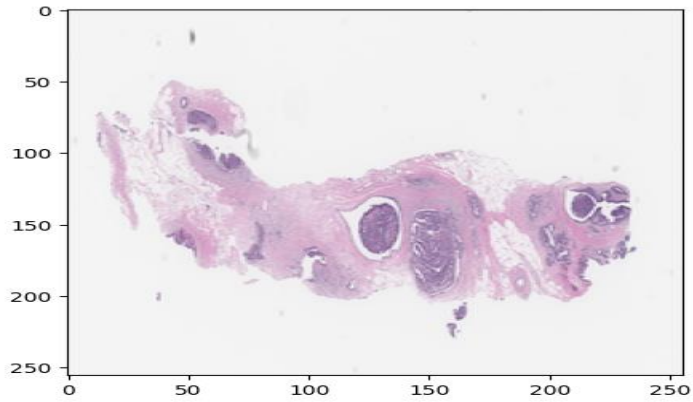


A2: Distributions of level 0 “width” dimensions (axis=0) across filtered files



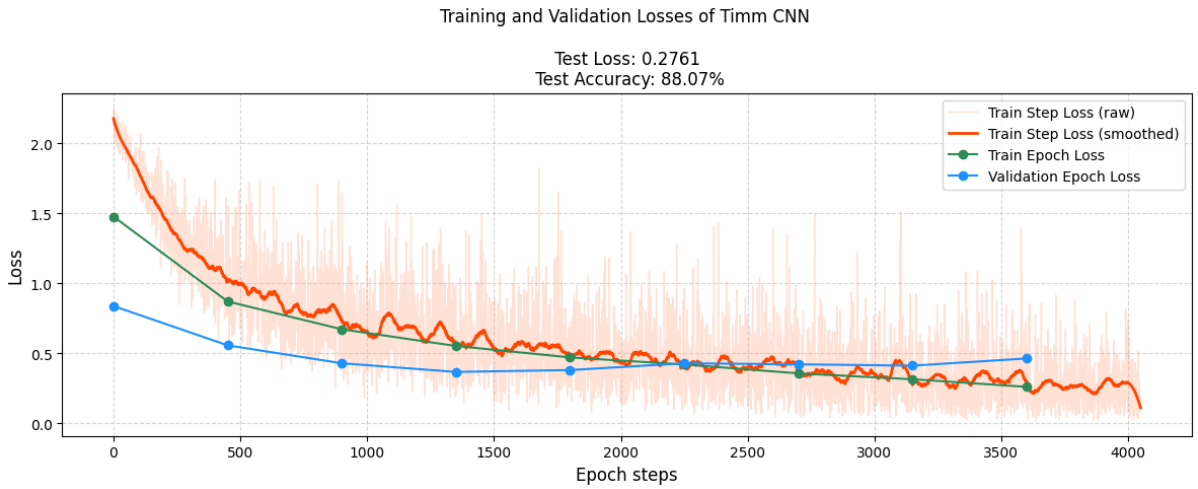
A3: Distributions of level 0 “height” dimensions (axis=1) across filtered files



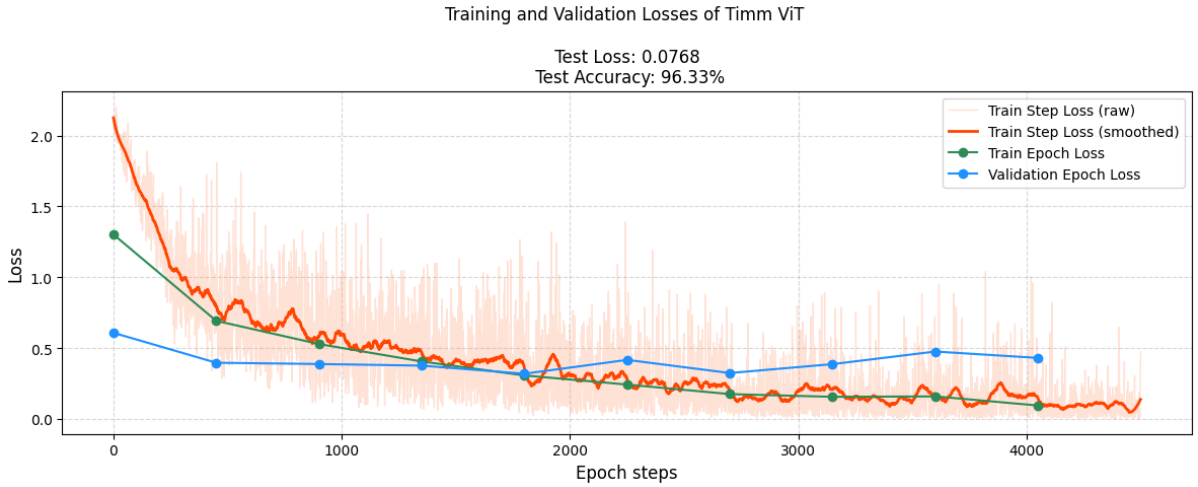


A4: The above images depict the full region of a WSI of dimensions (21932, 7686) before and after processing using **Lanczos** algorithm for resizing using PIL library. The filename and the corresponding label of the WSI is as follows and can be found in the dataset as well.

{'id': 'PIT_01_00119_01.tiff', 'report': 'Breast, core-needle biopsy; Papillary neoplasm'}



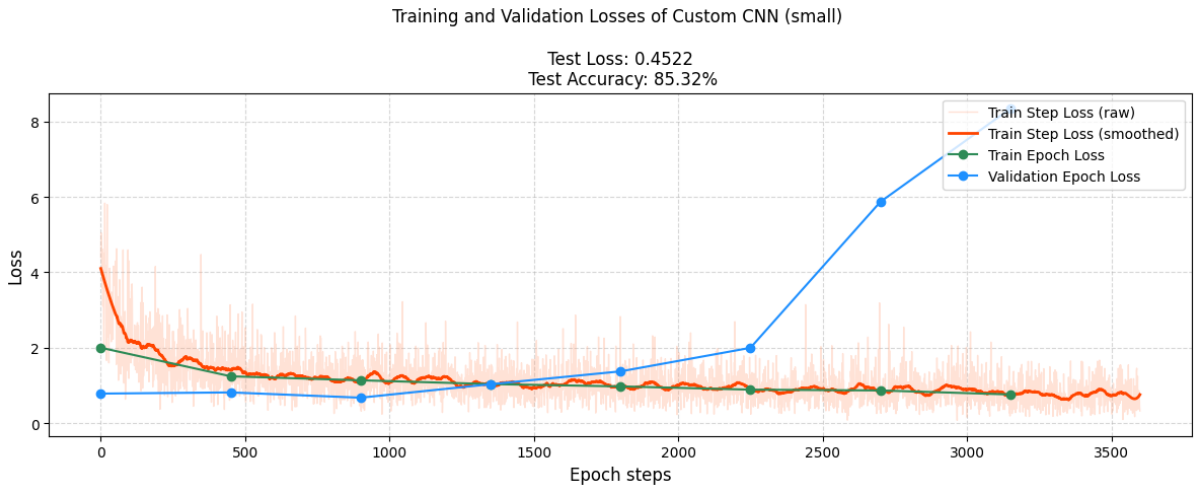
A5: Training logs of timm CNN model (efficientnet_b0) on 8 class organ classification task

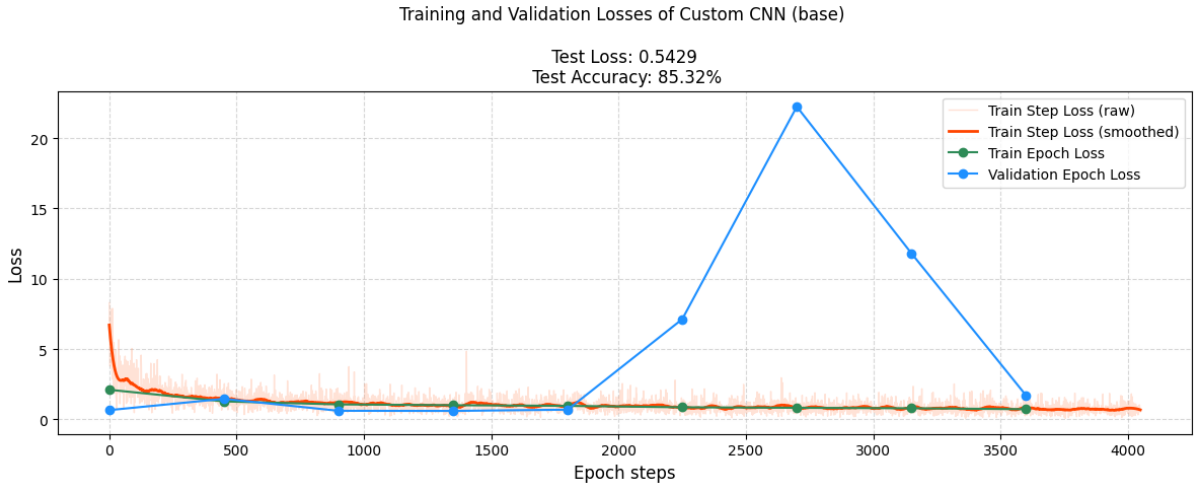


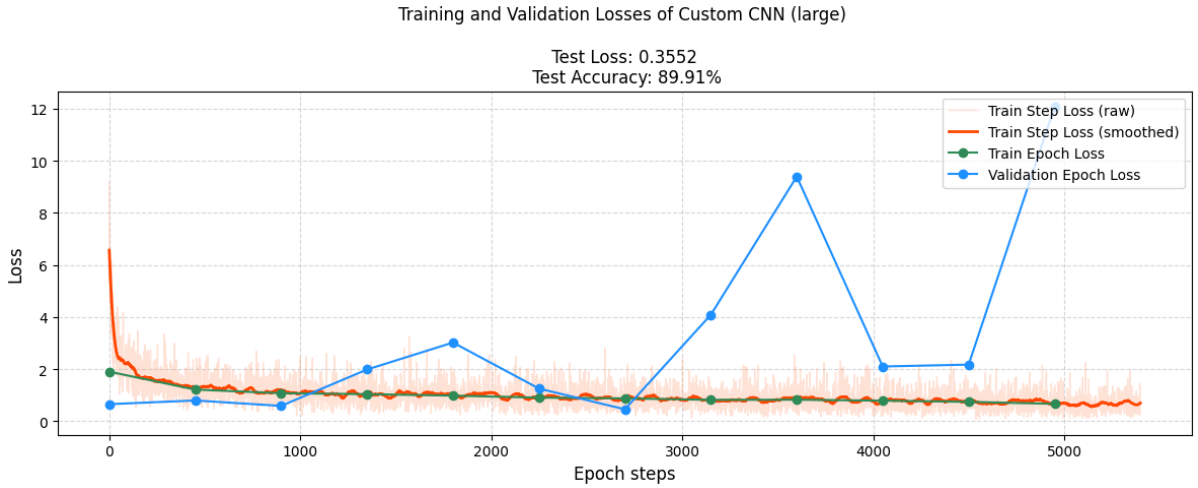
A6: Training logs of timm ViT model (vit_small_patch16_224) on 8 class organ classification task

Model	Final Train Loss	Final Val Loss	Final Val Accuracy %	Test Accuracy %	Test Loss
Timm ViT	0.0942	0.4302	88.20	96.33	0.0768
Timm CNN	0.2613	0.4639	85.40	88.07	0.2761
Custom CNN (small)	0.7583	8.3415	43.00	85.32	0.4522
Custom CNN (base)	0.7279	1.7101	58.20	85.32	0.5429
Custom CNN (large)	0.6665	12.0650	39.00	89.91	0.3552
Custom ViT (tiny)	0.6529	0.6772	81.40	92.66	0.2869
Custom ViT (small)	0.9974	1.3029	47.20	5.50	1.9359

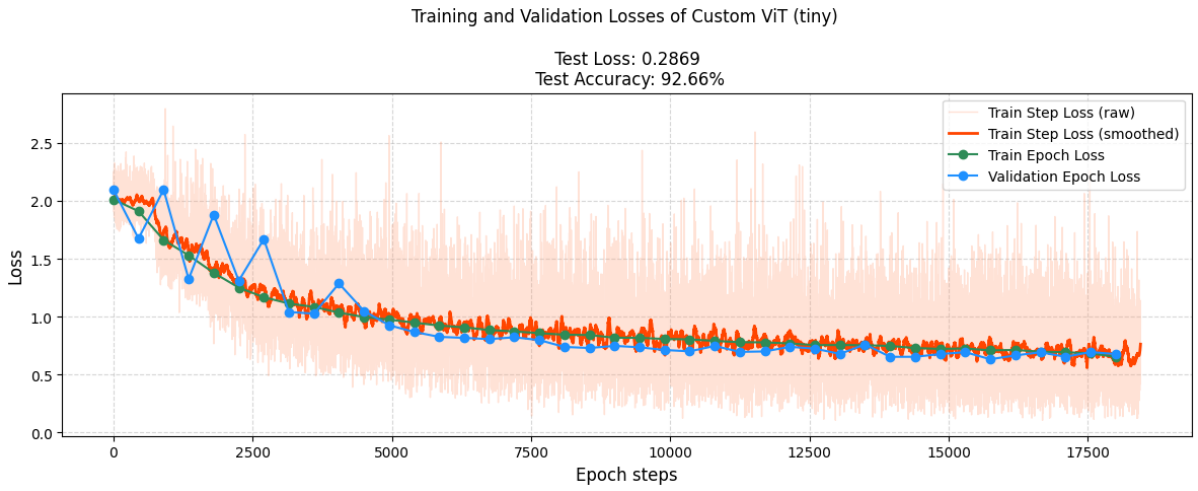
A7: Overall comparison of all trained image models on 8 class organ classification task.

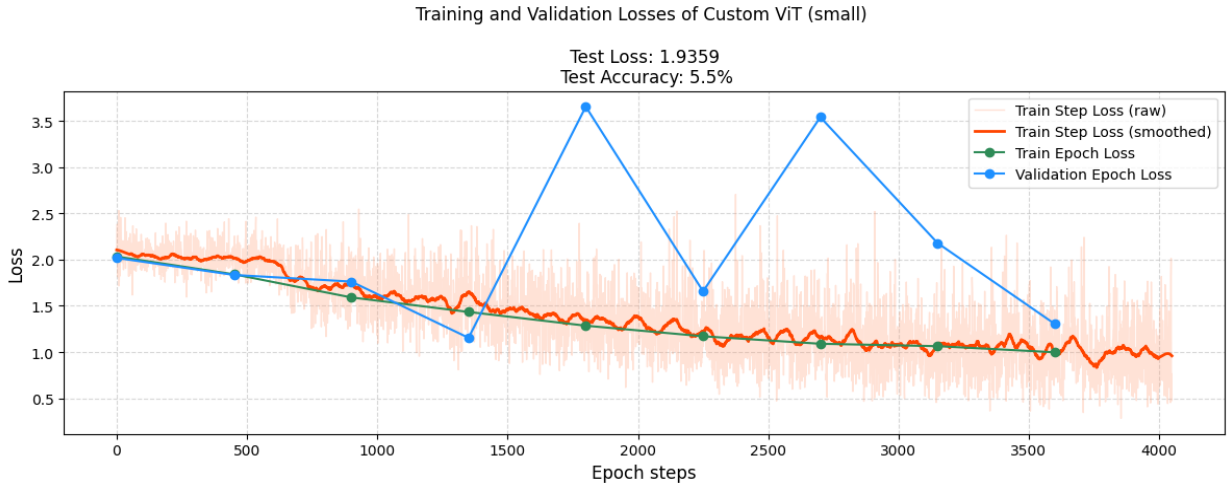




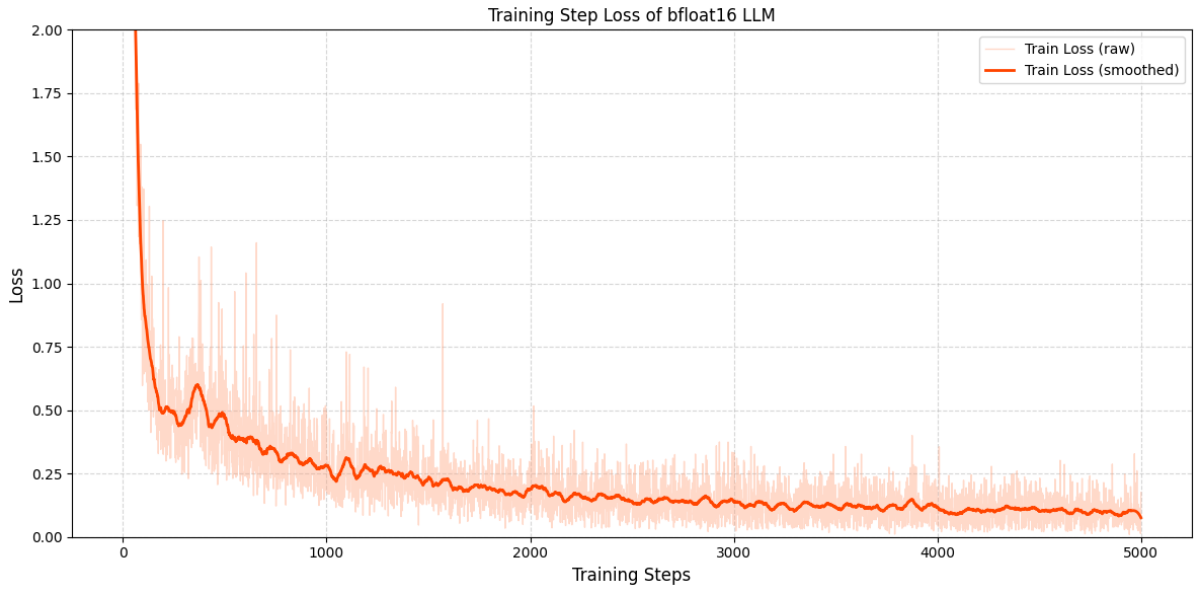


A8: Training logs and results of custom CNN models on 8 class organ classification task.

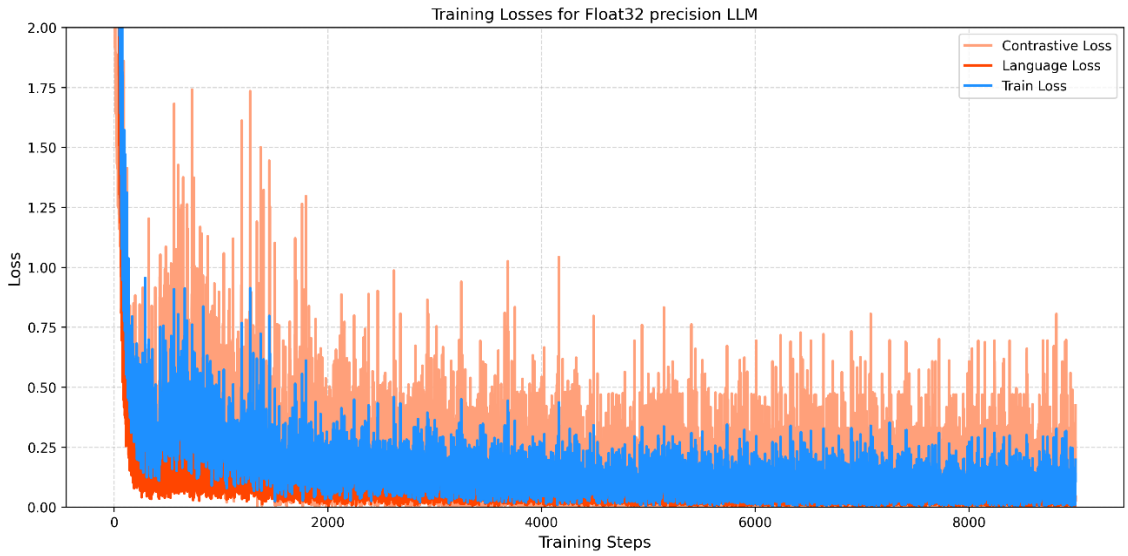




A9: Training logs and results of custom ViT models on 8 class organ classification task.



A10: Training logs and results of bfloat16 LLM on final phase training



A11: Training logs and results of float32 LLM on final phase training

A10: Links of datasets and code files can be found in this repository:
https://github.com/aneeshm11/Gemma3nChallenge

Navigate through the **README.md** file for access to all files related to this project

The demo video hosted on YouTube for a short intro for this project can be found here:
https://www.youtube.com/watch?v=2rA48SUgUwE

I am actively working on many other methods that uses LLMs along with other heavier architectures to solve this project. If my work seems interesting to you, check out the above repository for regular updates and advancements in the project and other ones *(related to medical imaging and healthcare tasks that involves using this kind of architecture)* of starting from August 23.

Hope you liked my project!!

Thank you for your time!!

x----------x----------x