Brain Tumor Detection with CLIP

Problem Statement:

Our goal is to automate the process of brain tumor detection using whole slide images (WSIs) of brain tumors, specifically focusing on identifying two types: Glioblastoma Multiforme (GBM) and Low-Grade Glioma (LGG). We aim to achieve this by training the CLIP (Contrastive Language-Image Pre-training) model with our training dataset, extracting feature vectors from the images using CLIP, and training a classification model based on these features. Subsequently, we'll utilize the trained model to predict the tumor type of unseen test images.

Workflow:

1. Data Preparation:
    - Our training dataset consists of SVS files containing whole slide images of brain tumors, categorized into GBM and LGG.
2. Training CLIP Model:
    - We'll train the CLIP model with our training data, enabling it to understand the visual features associated with GBM and LGG tumor types.
3. Feature Extraction with CLIP:
    - Utilizing the trained CLIP model, we'll extract feature vectors from the whole slide images in our training dataset. These features encapsulate the visual characteristics of the tumors.
4. Model Training:
    - We'll train a classification model using the extracted feature vectors and their corresponding labels (GBM or LGG). A logistic regression classifier is a suitable choice for this task.
5. Testing and Prediction:
    - Once the model is trained, we'll evaluate its performance using a separate test dataset. The model will predict the tumor type (GBM or LGG) of each whole slide image in the test dataset.
6. Automation and Reporting:
    - The automated pipeline will generate predictions for each test image. These predictions can be integrated into a larger workflow, potentially involving GPT (Generative Pre-trained Transformer) models for generating reports or further analysis.

RESEARCH PAPERS:
We started by reading the research papers “The Rise of AI Language Pathologists: Exploring Two-level Prompt Learning for Few-shot Weakly-supervised Whole Slide Image Classification” and “Attention is all we need”.

The paper discusses how Vision-Language (V-L) models, especially CLIP (Contrastive Language-Image Pre-training), can be used for classifying pathology Whole Slide Images (WSIs) with few examples and weak supervision. Here’s a simplified summary of the main points:

1. Vision-Language Models (V-L Models): V-L models like CLIP, ALIGN, and FLIP are important for understanding and transferring visual information. These models have two parts: one that processes images and another that processes text, allowing them to link images and text together.
2. CLIP Framework: CLIP uses an Image Encoder (like ResNet-50 or ViT) and a Text Encoder (a type of Transformer) to get features from images and text. It is trained on a huge dataset of image-text pairs, using a method that teaches it to match images with the right text in a shared space.
3. Prompt Learning: Prompt learning is crucial for good classification results. CLIP uses a specific prompt template to classify images by matching their features. This involves training an additional prompt vector using a small amount of labeled data from the specific task.
4. Zero-Shot Classification: CLIP can classify images without needing additional training for each new task. It can predict if an image matches a text description because it was pre-trained on a wide variety of image-text pairs, including some medical data.
5. Application in Pathology WSI Classification: The paper explores using CLIP for classifying pathology WSIs with few examples and weak supervision. By using CLIP’s prompt learning, the goal is to improve the accuracy and efficiency of classifying these medical images.

VISION TRANSFORMER->CLIP:

The shift from Vision Transformers to models like CLIP (Contrastive Language-Image Pre-training) is due to several key advantages:

1. Better Cross-Modal Understanding: CLIP excels in understanding both images and text by aligning their features in a shared space. This alignment helps the

model learn better representations that work well across different types of data, improving performance on tasks that need both visual and textual understanding.

2. Wider Applicability: While Vision Transformers are mainly used for visual tasks, CLIP's architecture handles both images and text, making it useful for a broader range of tasks that involve both types of data. This versatility makes CLIP more attractive for complex tasks requiring multimodal understanding.
3. Zero-Shot Learning: CLIP is trained on a large dataset of image-text pairs, which allows it to handle new tasks or classes without needing additional training. This ability to generalize to new scenarios without extra data makes CLIP a powerful tool, especially for tasks with limited labeled data.
4. Better Transfer Learning: CLIP's training on diverse image-text pairs helps it learn representations that can be easily adapted to new tasks. This transfer learning capability means CLIP can quickly adjust to new tasks with minimal fine-tuning, which is crucial for applications needing fast adaptation.
5. Efficient Representation Learning: CLIP's training method, which uses contrastive learning, helps the model learn meaningful representations that capture the relationships between images and text. This efficient learning leads to better performance on tasks that need to understand the semantic connections between visual and textual data.

SELF-ATTENTION AND CLIP ADVANTAGES:

In Vision Transformers, the self-attention mechanism is essential for capturing long-range dependencies and modeling relationships within an input sequence, such as the pixels in an image. However, there are limitations to the self-attention mechanism's effectiveness, which have led to the use of models like CLIP that extract feature vectors more efficiently. Here's why extracting feature vectors from CLIP is considered more effective:

1. Complexity and Scalability: Self-attention mechanisms in Vision Transformers can be very computationally expensive and use a lot of memory, especially with large, high-resolution images. This complexity makes it difficult for Vision Transformers to handle very large images efficiently. On the other hand, CLIP uses pre-trained encoders like ResNet-50 or ViT to extract feature vectors, offering a more efficient way to process images.
2. Cross-Modal Alignment: CLIP is designed to align image and text features in a shared space, which is crucial for understanding both types of data together. By using pre-trained encoders for both images and text, CLIP can capture

meaningful relationships between visual and textual data, improving performance on tasks that need both types of comprehension, like image classification based on text descriptions.

3. Transfer Learning: CLIP is pre-trained on a large and diverse dataset of image-text pairs, allowing it to learn rich representations that can be transferred to new tasks. The feature vectors extracted by CLIP encode important semantic information about both images and text, making them useful for understanding the concepts in the data. This capability enhances CLIP's adaptability and performance on various tasks.
4. Semantic Understanding: The feature vectors from CLIP contain high-level semantic information about the input data, allowing the model to understand and differentiate between different visual categories based on text descriptions. This deep semantic understanding is vital for tasks like image classification.

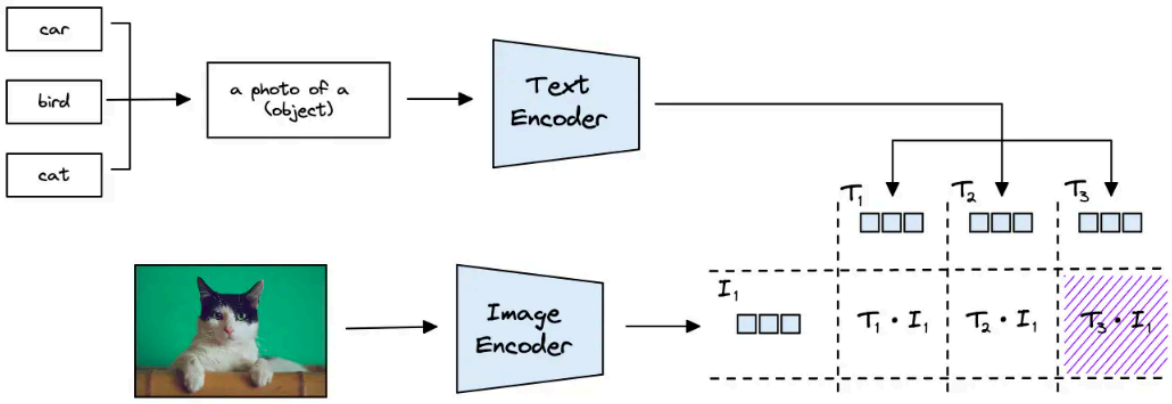
CLIP ARCHITECTURE:

The architecture of CLIP (Contrastive Language-Image Pre-training) is made up of two main components: the Image Encoder and the Text Encoder, which collaborate to align features from images and text in a common space:

1. Image Encoder:
    - CLIP uses architectures like ResNet-50 or Vision Transformer (ViT) to process input images and extract features.
    - The Image Encoder takes the input image and produces a feature vector that represents the visual information extracted from the image.
2. Text Encoder:
    - Transformer-based architectures are employed as the Text Encoder to handle textual descriptions or prompts.
    - The Text Encoder generates embeddings that capture the meaning and context of the input text, like descriptions of image categories or concepts.
3. Contrastive Learning Objective:
    - CLIP is trained using a contrastive learning method. It aims to make matching pairs of image-text inputs similar while making non-matching pairs dissimilar.
    - This objective encourages the model to align image and text features in a shared space, promoting cross-modal understanding.
4. Shared Embedding Space:

- CLIP maps both images and text into a common embedding space where similar concepts are placed close together.
- By aligning features in this shared space, CLIP can perform tasks like zero-shot learning and few-shot learning, even with limited labeled data.

5. Prompt Learning:
    - CLIP utilizes prompts, which are carefully crafted textual descriptions, to guide the model in understanding and classifying images.
    - These prompts are processed by the Text Encoder to generate embeddings that assist in image classification based on their semantic content.
6. Cross-Modal Alignment:
    - The Image Encoder and Text Encoder work together to achieve cross-modal alignment. This allows CLIP to effectively understand and relate information from both images and text.
    - This alignment enables CLIP to handle tasks that require understanding across multiple modalities, such as classifying images based on textual descriptions.



We reformat the one-word classes into sentences and encode them into label-based text vectors. When we encode and compare an image of a cat, we expect to share the highest similarity with *"a photo of a cat"*.

We tried to implement the above process incrementally. Initially, we read the whole slide image to gather important visual information using CLIP, extracting feature vectors to train our model and predict the type of tumor on our test dataset.

Due to a lack of computational resources, we tested this approach on a very small part of the dataset.

First Attempt

We read only 1/10th of the whole slide image, extracted the feature vectors, and trained the model with this limited information.

Improved Approach

To enhance precision, we utilized the idea of patches instead of reading the whole slide image at once. We divided the image into 25 patches, where the dimensions of each patch are (width/5, height/5), with (width, height) being the dimensions of the original image. We then predicted the tumor type for each of these patches and performed majority voting across all patches to determine the final prediction for the test dataset.

1st Code:

We used Python libraries such as `openslide` and `PIL` to handle and process the whole slide images stored in ".svs" format. The process involved:

1. Function Definition: Created a function `load_images_from_folder(folder, label)` to load images from a folder and assign labels.
2. Image Loading: Iterated through files in the folder, opening ".svs" files using `openslide.OpenSlide`.
3. Image Processing: Extracted image dimensions and loaded a region of the slide, converting it to RGB format. We took only 1/10th part of the image as training with the whole image takes a lot of memory.
4. Labeling: Assigned labels to images and stored them in lists.
5. Output: Returned lists of images and labels.
6. Data Loading: Applied the function to load training images labeled "lgg" and "gbm" from respective folders, and test images from a separate folder.

Clip:

1. Model and Processor Loading: We loaded the CLIP model and processor from the OpenAI model repository.
2. We defined a function to extract features from images, utilizing the CLIP model.Inputs to the model included both the image and its corresponding

label.Features were extracted using the `clip_model.get_image_features()` method, generating a feature vector for each image..

3. Features were extracted from both training and testing datasets.For training images, features were extracted based on their assigned labels ("lgg" or "gbm").Testing images were processed without assigned labels, typical for evaluation datasets.
4. Utilization of Extracted Features: Extracted features serve as rich representations of images, encoding semantic information learned by the CLIP model.

Training:

1. Model Training:
    - Utilized a logistic regression classifier (`LogisticRegression()`) from the scikit-learn library for training.
    - Trained the classifier using the combined features and labels (`X_train_flat` and `y_train`) via the `fit()` method.
2. Prediction Function:
    - Defined a function (`predict()`) to make predictions using the trained classifier. It will predict on each test image.

Accuracy=75 percent

2nd code:

1. Our code retrieves the slide dimensions and sets the initial cropping parameters after all the import statements and handling the svs files.
2. We then divide the slide into a grid (5 rows by 5 columns), creating 25 cropped images. After this each cropped region is resized and converted to RGB format.The processed images are appended to the `images` list, and their labels are appended to the `labels` list.
3. Dependencies used:
    a. `torch`: PyTorch, a deep learning framework.
    b. `transformers`: Hugging Face's library for state-of-the-art machine learning models, including CLIP.
    c. `numpy`: A library for numerical computations.
    d. `multiprocessing`: A Python module for parallel processing.

4. The CLIP model (`clip-vit-base-patch32`) and its processor are loaded from the Hugging Face model hub.
5. Then we have a process_image function which takes image and label as parameters and with the help of clip processors we prepare the inputs by combining the image and the corresponding text label and converting them to PyTorch tensors.
6. Using `torch.no_grad()` to ensure no gradients are calculated, the image features are extracted using the CLIP model.The extracted features are moved back to the CPU and converted to a NumPy array.Then the function returns the concatenated features.
7. In Extract_features function first we use a multiprocessing pool which is created to parallelize the processing of images.Then The `process_image` function is applied to each image-label pair using `starmap`, which allows multiple arguments to be passed to the function.The pool is closed and joined to ensure all processes complete before proceeding.The results are converted to a NumPy array and returned.
8. We combine labels and features, Concatenate LGG and GBM features and flatten them. Concatenate LGG and GBM labels.Then we use a logistic regression classifier which is trained using the combined features and labels.
9. Since we have 25 patches for each image we do majority voting, we take the majority of predictions of all 25 patches and predict the image with a label which has occurred majorly.
10. Majority voting is applied to determine the final prediction for the original image:

    If more than half of the cropped images are predicted as "gbm", the original image is labeled as "gbm"; otherwise, it is labeled as "lgg".

11. Now we evaluate performance of the model on both training and testing data.

    Accuracy = 80 percent

Future works :

1. Finding how many patches we must divide the image so that we get the best test accuracy (or F1 score) using ROC plots.
2. Instead of just using the clip model we can try to use any of the vision transformers or the combination of clip and vision transformers to get better predictions.

3. We can introduce a language model like chat gpt-4 to get more precise predictions on the test dataset .
4. We can add more input data in the form of patient medical history and combine it with the slide images to get clear evidence of why and what type of tumor the patient has .

Contributions:

Throughout the project, we have both contributed equally to all aspects of the research and implementation. We collectively reviewed and analyzed three key papers: one detailing the CLIP model and its architecture, another discussing Vision Transformers, and a third outlining the specific requirements and objectives of our project.We collaboratively developed and iteratively improved three versions of the code, each building upon the previous one. One of us experimented with a subset of classifiers, including logistic regression and decision trees, while the other worked with classifiers such as SVM and random forest, to determine the best-performing models.Both of us collaborated on implementing methods to extract key features and feature vectors. We explored different image processing techniques, including reading the whole image at once and dividing the image into patches for more detailed analysis.We then merged the important and effective parts from each other's contributions to create a final, optimized code.