Code explanations!

A. VIT from scratch

1. **Imports and Classes Definitions**:
    - The code begins by importing necessary libraries and defining classes for various components of the Vision Transformer, such as NewGELUActivation, PatchEmbeddings, Embeddings, Attention Head, MultiHeadAttention, FasterMultiHeadAttention, MLP, Block, Encoder, and ViTForClassfication.
    - Each class is well-commented, explaining their roles and the transformations they apply to the inputs.

    ViT Implementation:

    - Implementation of various components needed for Vision Transformers, such as:
    - NewGELUActivation: An activation function.
    - PatchEmbeddings: Converts the image into patches and projects them into a vector space.
    - Embeddings: Combines patch embeddings with class token and position embeddings.
    - AttentionHead and MultiHeadAttention: Implements the attention mechanism.
    - FasterMultiHeadAttention: Optimized multi-head attention.
    - MLP: A multi-layer perceptron module.
    - Block: A single transformer block.
    - Encoder: The transformer encoder module.
    - ViTForClassfication: The Vision Transformer model for classification.

2. **Data Preparation**:
    - prepare_data function prepares the CIFAR-10 dataset.
    - Key parameters:

        - batch_size: Number of samples processed before the model is updated.
        - num_workers: Number of subprocesses to use for data loading.
        - train_sample_size and test_sample_size: Number of samples to use from the training and testing datasets.
3. **Utility Functions**:
    - Functions for saving and loading experiments, visualizing images, and visualizing attention maps.
4. **Training the Vision Transformer**:
    - Training configuration:
        - exp_name: Experiment name.
        - batch_size: 32.
        - epochs: 10.
        - lr: Learning rate set to 0.01.
        - save_model_every: How often (in terms of epochs) to save the model.
    - The Trainer class handles the training process, including training for one epoch, evaluating the model, and saving checkpoints.
5. **Main Training Loop**:
    - The main function sets up the training parameters, loads the CIFAR-10 dataset, creates the model, optimizer, and loss function, and starts training.

Summary

- **Batch Size**: 32
- **Epochs**: 10
- **Learning Rate (lr)**: 0.01
- **Patch Size**: 4 (Image size: 32x32, divided into 8x8 patches)
- **Hidden Size**: 48
- **Number of Hidden Layers**: 4
- **Number of Attention Heads**: 4

- **Intermediate Size**: 192 (4 times the hidden size)
- **Dropout Probabilities**: 0.0 for both hidden and attention probabilities
- **Number of Classes**: 10 (CIFAR-10 dataset)
- **Number of Channels**: 3 (RGB images)

The code implements a Vision Transformer for image classification, utilizing the CIFAR-10 dataset. It includes components for patch embedding, multi-head attention, and transformer blocks, with a training loop that evaluates and saves the model periodically. The utility functions aid in saving and visualizing the experiment's progress.

B.VITB16

Imports and Device Setup

1. **Imports**: You're importing the necessary libraries for machine learning, specifically PyTorch, Torchvision (for pre-trained models and transforms), and matplotlib (for visualization).
2. **Device Setup**: The code checks for GPU availability to use CUDA if possible, which will significantly speed up model training and evaluation. If not available, it defaults to CPU.

Defining Class Labels

The class_names list defines four labels: 'glioma', 'meningioma', 'notumor', and 'pituitary'. These represent the types of brain tumor (or absence thereof) that the model will classify.

Model Setup Functions

1. **setup_model()**:
    - This function initializes a pre-trained Vision Transformer (ViT) model with the vit_b_16 architecture, which is a smaller version of ViT but still highly effective for image classification.
    - **Freezing Layers**: By calling freeze_parameters(model), it prevents the main layers of the pre-trained model from updating, meaning only the classifier layer will learn from scratch on this specific dataset.

- **Changing the Classifier Head**: Since the pre-trained ViT might have been trained on a different dataset (e.g., ImageNet with 1000 classes), you replace its final classifier layer to output 4 classes.

2. **Freezing Model Parameters**:
    - freeze_parameters(model) iterates over each layer of the model and sets requires_grad = False, ensuring that only the final classifier will be trainable. This is commonly done to transfer learning because it allows the model to leverage existing learned patterns while focusing only on distinguishing the specific classes you care about.
3. **Changing the Classifier Head**:
    - change_classifier_head(model, class_names, device) replaces the classifier (output) layer with one that matches your number of classes, i.e., 4. It uses nn.Linear to create a new fully connected layer with 768 input features (matching the ViT output) and 4 output features (one for each class).

Model Summary

- The print_model_summary() function, along with torchinfo.summary, provides a structured view of the model, listing each layer's input size, output size, and whether it's trainable. The input size (32, 3, 224, 224) signifies:
    - **32**: Batch size, i.e., the number of images processed in parallel.
    - **3**: Number of color channels (RGB).
    - **224 x 224**: Height and width of each image.

This summary helps verify that your model's setup and output layers align with the expected input dimensions.

Data Preparation

1. **Transforms**:
    - Using get_and_print_transforms(weights), you set up transforms aligned with how the model was originally trained. Typical transforms include resizing, cropping, and normalizing the images

to match the model's expected input format, improving the effectiveness of transfer learning.

2. **Data Loaders**:
    - **setup_data_loaders()**: This function loads the images in batches of 32 for training and testing. It shuffles training data to ensure that the model generalizes better and uses multiple CPU cores to speed up data loading.
    - **Batch Size**: Set at 32, meaning the model processes 32 images at a time.

Model Training

1. **Setting Training Hyperparameters**:
    - **Learning Rate (1e-3)**: This rate controls how much the model's weights are adjusted with respect to the loss gradient during training.
    - **Epochs (30)**: The number of complete passes through the training dataset.
2. **Creating Optimizer and Loss Function**:
    - **Adam Optimizer**: Adam is widely used due to its adaptive learning rate features, which make it effective for complex models.
    - **Cross-Entropy Loss**: Common for multi-class classification, it calculates the error based on the difference between the model's predictions and the actual class labels.
3. **Training Function**:
    - **train_model()**: This function orchestrates the training process over 30 epochs, reporting the training and validation losses and accuracies after each epoch. Monitoring these metrics helps determine whether the model is improving, overfitting, or underfitting.

Saving Model and Results

- You save the trained model, test data, and results using pickle. Saving the model allows you to load it later without retraining, while saving the results lets you analyze model performance in greater detail.

Plotting Loss and Accuracy Curves

- **plot_loss_curves()**: This function visualizes the model's training and testing performance over time. A well-trained model should ideally show decreasing training and testing losses and increasing accuracies. If the testing loss decreases and then rises again, it could indicate overfitting.

Model Evaluation and Metrics

1. **Evaluating the Model**:
    - evaluate_model() assesses the trained model's performance on the test set. By setting the model to evaluation mode and turning off gradient tracking, it speeds up the inference and reduces memory usage.
2. **Classification Report**:
    - Outputs precision, recall, and F1-score for each class, along with overall accuracy, to measure the model's effectiveness for each tumor type.
3. **Confusion Matrix**:
    - Visualizes the model's predictions across each class, showing where the model performed well or made mistakes. For example, if glioma is often misclassified as meningioma, this will appear on the confusion matrix, which helps in diagnosing model issues.

Key Parameter Summary

- **Batch Size**: 32
- **Epochs**: 30
- **Learning Rate**: 1e-3
- **Image Size**: (224, 224) expected by the model
- **Model Architecture**: Pre-trained ViT (Vision Transformer)