Code explanations!

A. CNN from scratch

1. Setting up and Loading Data

- **Imports:** The code imports libraries like torch and torchvision for deep learning and matplotlib for visualization.
- **Dataset Paths:** data_dir and test_data_dir specify the locations for training and testing images.
- **Transforms:** The images are resized to 150x150 and converted to tensors (numerical format for models).
- **Loading Data:** ImageFolder loads images from folders, assigning labels based on folder names.

2. Displaying Sample Data

- **Image Display Function:** The display_img function shows an image from the dataset and prints its label.
- **DataLoader:** DataLoader wraps datasets into batches, shuffles them, and enables efficient loading.

3. Saving DataLoaders

- **Using Pickle:** The code saves the test_dl (test data loader) as a file, which can be reloaded later.

4. Batch Display Function

- **show_batch:** This function visualizes a grid of images from a single batch to get a quick overview of the dataset.

5. Model Training and Evaluation

- **Accuracy Function:** Calculates accuracy by comparing predicted labels to actual labels.
- **Evaluation Function (evaluate):** Tests the model on the test dataset and returns the results.
- **Training Function (fit):** Trains the model over a specified number of epochs, adjusting weights to reduce loss and improve accuracy.

6. Model Definition

- **Base Class (ImageClassificationBase):** This class provides functions for training, testing, and summarizing performance per epoch.

- **Model (NaturalSceneClassification):** Defines the model structure, with several layers:
    - **Convolutional Layers:** These layers detect features (edges, textures) in the images.
    - **ReLU (Activation) Layers:** Adds non-linearity, helping the model learn complex patterns.
    - **Pooling Layers:** Reduce image size while retaining important information.
    - **Fully Connected Layers:** Combines features to make final predictions.

7. Training the Model

- **Model Setup:** Initializes the model, sets learning rate, and optimizer (Adam for automatic tuning).
- **Train Loop:** Runs the model over 30 epochs, printing training and testing losses and accuracy.

8. Saving the Model and History

- **Saving with Pickle:** The trained model and training history are saved for later use.

9. Plotting Performance Metrics

- **plot_accuracies and plot_losses:** These functions plot accuracy and loss over epochs to show how the model improved during training.

10. Confusion Matrix

- **get_predictions:** A function to get model predictions and true labels from the test set.
- **Confusion Matrix:** Displays a confusion matrix, showing where the model predicts correctly or incorrectly for each class.

The key points from the code, covering important settings, hyperparameters, and choices:

1. **Data Preprocessing:**
    - **Image Resizing:** Each image is resized to **150x150 pixels**.

- **Transformation to Tensor:** Images are converted to tensors, which are required for PyTorch processing.

2. **Data Loading:**
   - **Batch Size:** The training DataLoader uses a **batch size of 128**, while the test DataLoader uses **256**.
   - **Shuffling:** Training data is shuffled to randomize batches each epoch, improving generalization.
3. **Data Split:**
   - **Training Set Size: 5,712** images.
   - **Test Set Size: 1,311** images.
4. **Model Architecture:**
   - **Convolutional Layers:** The model includes multiple convolutional layers with increasing filter sizes (32, 64, 128, and 256) to capture features at different levels.
   - **Activation Function:** ReLU is used after each convolution to introduce non-linearity.
   - **Pooling Layers:** MaxPooling layers reduce the spatial dimensions, helping to decrease computational requirements and add translation invariance.
   - **Fully Connected Layers:** After the convolutional layers, the model has three fully connected layers with output sizes of **1024, 512,** and **4** (for the four classes).
   - **Flattening:** The feature maps are flattened before passing them into the fully connected layers.
5. **Optimizer and Learning Rate:**
   - **Optimizer:** Adam optimizer is used, as it adapts learning rates for each parameter, helping with faster convergence.
   - **Learning Rate:** Set to **0.001**, a common choice for training CNNs with Adam.
6. **Loss Function:**
   - **Cross-Entropy Loss:** Used as it's well-suited for multi-class classification problems, comparing predicted probabilities with true class labels.

7. **Training and Evaluation:**
   - **Epochs:** The model is trained for **30 epochs**.
   - **Accuracy Calculation:** A custom function calculates accuracy by comparing predicted labels with true labels.
   - **Evaluation:** The model is evaluated after each epoch, with metrics such as training loss, test loss, and test accuracy printed.
8. **Model Saving and Loading:**
   - The trained model and training history are saved as pickle files (model1.pkl and history1.pkl), allowing the model to be loaded for future use without retraining.
9. **Performance Visualization:**
   - **Confusion Matrix:** Used to visualize model performance on the test set by showing true versus predicted labels.
   - **Accuracy Plot:** A line plot shows accuracy across epochs, helping to track model improvements.
   - **Loss Plot:** A plot displays both training and test losses per epoch, which helps diagnose overfitting or underfitting.

B. CNN by VGG16

1. **Import Libraries**: Essential libraries like torch, torchvision, and matplotlib are imported for model training, data loading, transformations, and visualization.
2. **Data Loading and Transformation**:
   - Paths to training and testing data directories are specified.
   - Data is loaded using ImageFolder, which helps manage images within folders representing classes.
   - Images are resized to (150, 150) and converted to tensors for input to the model.
3. **Visualizing Data**:
   - The display_img function shows the first image in the dataset with its label.

- show_batch displays a grid of images from a single batch to give an overview of the training data.

4. **VGG16 Model Setup**:
   - A pre-trained VGG16 model is loaded. VGG16 has several convolutional layers followed by dense layers (or fully connected layers).
   - The last layer in model.classifier is adjusted to match the number of classes in the dataset (4 classes: glioma, meningioma, no tumor, pituitary).
   - The model is then moved to the GPU if available for faster computation.
5. **Training Setup**:
   - **Loss Function**: Cross-entropy loss is used as it's suitable for multi-class classification.
   - **Optimizer**: Adam optimizer with a learning rate of 0.001 is chosen to update model weights.
   - train_dl and test_dl are DataLoaders that help load data in batches of size 128 for training and 256 for testing.
6. **Training and Testing Functions**:
   - train_model goes through a specified number of epochs (30) and updates model weights.
   - **Testing**: At the end of each epoch, the model's performance on the test data is evaluated using the test_model function.
7. **Training and Testing Output**:
   - For each epoch, training and testing loss and accuracy are printed.
   - Training and test losses/accuracies are plotted over epochs to observe improvement trends.
8. **Model Saving**:
   - The trained model and its performance metrics are saved as pickle files for later use or evaluation.
9. **Confusion Matrix**:
   - Predictions are generated, and a confusion matrix displays the number of correct/incorrect classifications for each class.

Key Details in the Code

- **Batch Size**: 128 for training, 256 for testing.
- **Epochs**: 30 epochs for training.
- **Model Architecture**: VGG16, with the final fully connected layer adjusted for 4 output classes.
- **Learning Rate**: 0.001 (Adam optimizer).
- **Loss Function**: Cross-entropy loss.
- **Data Augmentation**: Only resizing, with no further augmentation applied.
- **Device**: GPU, if available, otherwise CPU.
- **Nodes in Dense Layer**: The final dense layer has 4 nodes, corresponding to the 4 classes in the dataset.

Freezing Layers and Dense Layer Details

The code does **not freeze any layers** of the VGG16 model. This means all layers (both convolutional and fully connected) are being trained on this dataset. As for the dense layer, the last layer in model.classifier is modified to have 4 nodes—one node per class—replacing the original 1000-node layer used in the original VGG16 for ImageNet classification.

C. CNN by RESNET50

1. **Import Libraries**: TensorFlow's ImageDataGenerator, ResNet50, and other model-building tools are imported along with necessary libraries like pickle for saving data and matplotlib for visualization.
2. **Data Loading and Transformation**:
   - Training and testing image directories are specified.
   - Data generators (ImageDataGenerator) are used to load images. Images are normalized (rescaled) by dividing pixel values by 255.
   - Images are resized to (224, 224), and a batch size of 32 is set for both training and testing data.

3. **Model Architecture (ResNet50)**:
    - **Pre-trained ResNet50 model**: The model is loaded with weights trained on ImageNet, excluding the top classification layer (include_top=False). This means we use the convolutional part of ResNet50 as a feature extractor.
    - **Custom Layers**: A global average pooling layer is added to reduce the number of parameters, followed by a dense layer with 256 nodes and ReLU activation. Finally, an output dense layer with softmax activation is added to classify the images into four classes (glioma, meningioma, no tumor, and pituitary).
4. **Freezing Layers**:
    - **Freezing the ResNet50 Layers**: All layers in the base ResNet50 model are frozen, meaning they won't be updated during training. This allows the model to leverage the learned features of ResNet50 without retraining its weights.
5. **Model Compilation**:
    - The model is compiled with the Adam optimizer (for adaptive learning rate), categorical_crossentropy loss (suitable for multi-class classification), and accuracy as a performance metric.
6. **Model Training**:
    - The model is trained for 30 epochs using fit, with both training and validation (testing) accuracy and loss monitored.
    - After each epoch, the accuracy and loss values for both training and validation sets are displayed.
7. **Saving Results**:
    - Training and validation losses and accuracies are saved in a dictionary and then saved to a .pkl file.
    - The final model is saved as a .pkl file as well.
8. **Model Evaluation**:
    - Training and test losses/accuracies are plotted across epochs to visually track progress.
    - The confusion matrix is generated, showing the classification results for each class, with correct predictions along the diagonal and misclassifications elsewhere.

Key Details in the Code

- **Batch Size**: 32 for both training and testing.
- **Image Dimensions**: (224, 224, 3), which matches the ResNet50 input requirements.
- **Epochs**: 30 epochs for training.
- **Model Architecture**: Pre-trained ResNet50 with two added layers:
  - GlobalAveragePooling2D
  - Dense layer with 256 nodes and ReLU activation
  - Final Dense layer with softmax activation and 4 nodes, one for each class.
- **Optimizer**: Adam optimizer, a common choice for deep learning tasks.
- **Loss Function**: Categorical cross-entropy for multi-class classification.
- **Metrics**: Accuracy for monitoring model performance.
- **Device**: CPU (from TensorFlow logs, GPU is not detected).

Explanation of Layer Freezing and Dense Layer Nodes

The code **freezes all layers in the ResNet50 base model**, so these layers will not be updated during training. This approach is useful when you want to utilize a well-trained feature extractor, like ResNet50, and avoid the computational cost of retraining it. Only the newly added layers (GlobalAveragePooling2D and Dense layers) are trainable and will be optimized during training.

The final dense layer has **4 nodes**, corresponding to the four classes in the dataset, with softmax activation to provide class probabilities for multi-class classification.

D. CNN with K fold

1. Data Loading and Transformation

- **Train and Test Datasets**: ImageFolder is used to load the images from specified directories. Each image is resized to 150 ×150 and converted to tensors.

- **Classes and Display**: The dataset consists of four classes (glioma, meningioma, notumor, and pituitary). The first image is displayed to verify data loading.

2. DataLoader Setup

- The script splits data into train and test loaders with batch size 128, utilizing multiple workers for efficiency.

3. CNN Architecture

- **Model**: The NaturalSceneClassification class inherits from ImageClassificationBase. It contains several convolutional and pooling layers followed by fully connected layers.
- **Parameter Summary**: torchsummary is used to print model details. The model has ~86.6 million parameters, emphasizing the computational load.

4. Training and Evaluation

- **Training**: fit function performs the training for a specified number of epochs, using an optimizer like Adam with a learning rate of 0.001.
- **Evaluation**: evaluate calculates accuracy and loss for the test data.

5. k-Fold Cross-Validation

- The model is trained across 5 folds, with each model saved separately. Results are stored for each fold, and average validation loss and accuracy are calculated.

6. Ensemble Predictions

- **Majority Voting**: This technique aggregates predictions by selecting the most common prediction across all models.
- **Average Voting**: Instead of direct class predictions, it calculates the average of the softmax outputs and selects the class with the highest probability.

7. Performance Metrics

- **Confusion Matrix**: After final predictions, a confusion matrix is plotted to visualize class-specific performance.
- **Accuracy**: The script outputs test accuracy, showcasing model performance on the unseen test dataset.

Output Highlights

- Final accuracy of approximately 97.5% demonstrates the model's high classification capability.
- Confusion matrices provide a clear, class-specific error analysis for improved interpretability.

---

1. Data and Transformation:

- **Image Size**: Each image is resized to 150×150.
- **Classes**: The dataset contains 4 classes — glioma, meningioma, notumor, and pituitary.
- **Transformations**: Images are resized and converted to tensors.

2. DataLoader Configuration:

- **Batch Size**:
    - **Training**: 128
    - **Testing**: 256 (double the training batch size for efficient evaluation)
- **Number of Workers**: 4 (to parallelize data loading)
- **Pin Memory**: Enabled for faster data transfer to GPU.
- **Data Split**:
    - **Training Data**: 5712 images
    - **Testing Data**: 1311 images

3. Model Architecture:

- **Convolutional Layers**:
    - First Conv Layer: 3 input channels, 32 output channels, kernel size 3, padding 1.
    - Second Conv Layer: 32 input channels, 64 output channels, kernel size 3, padding 1.
    - Further layers with 128 and 256 output channels, using ReLU activation and MaxPooling after every few layers.
- **Fully Connected Layers**:

        - Flatten layer to transform CNN output into a 1D vector (size 82944).
        - Three Dense Layers:
            - First FC Layer: 82944 to 1024 nodes
            - Second FC Layer: 1024 to 512 nodes
            - Output Layer: 512 to 6 nodes (original code indicates 6 classes, but should be adjusted to 4 as per dataset)
    - **Total Parameters**: 86,589,638

4. **Training and Optimization:**
    - **Epochs**: 30 (for each fold in cross-validation)
    - **Learning Rate**: 0.001
    - **Optimizer**: Adam optimizer (specified in opt_func)
    - **Loss Function**: Cross-entropy loss for multi-class classification.
    - **Accuracy Calculation**: Custom accuracy function based on top-1 accuracy.

5. **k-Fold Cross-Validation:**
    - **Number of Folds**: 5
    - **Data Split per Fold**: Each fold uses a subset for training and validation.
    - **Seed**: Random seed of 42 for reproducibility in split.

6. **Ensemble Predictions:**
    - **Voting Methods**:
        - **Majority Voting**: Aggregates the most common class prediction across folds.
        - **Average Voting**: Takes the mean of class probabilities across folds, selecting the class with the highest average probability.
    - **Ensemble Output**: Produces a final prediction array after evaluating all models on the test set.

7. **Evaluation and Visualization:**
    - **Metrics**:
        - Average validation loss and accuracy across folds.

    - Test set accuracy calculated on the aggregated predictions.
- **Confusion Matrix**: Provides detailed class-wise performance.
- **Plots**:
    - Validation loss and accuracy plotted for each fold.
    - Final confusion matrix plotted for test predictions.

8. Model Saving:

- **Per-Fold Model**: Each model trained on a fold is saved using pickle (model_fold_<fold_num>.pkl).
- **Cross-Validation Results**: Saved as cv_results.pkl.
- **Ensemble Models**: Entire list of trained models saved in models.pkl.

Summary Outputs:

- **Final Test Accuracy**: Approximately 97.5% (based on majority and average voting).
- **Estimated Total Model Size**: ~401 MB due to high parameter count.