Deep Learning Image Classification with TensorFlow and Keras

1. Import Libraries

The code starts by importing the necessary libraries, including TensorFlow, Keras, and other related packages for data preprocessing and visualization.

2. Data Preparation

Data Augmentation

- Two data generators, `train_gen` and `valid_gen`, are created using the `ImageDataGenerator` from Keras.
- `train_gen` applies various data augmentation techniques such as rescaling pixel values, random rotations, shifts, horizontal flips, and zoom to the training images.
- `valid_gen` only rescales pixel values for the validation images.

Data Loading

- Using `flow_from_directory`, `train_flow` and `valid_flow` are created to load batches of images from the specified directory for training and validation. These are used as input for the model.

3. Model Architecture

- The code defines a Convolutional Neural Network (CNN) model for image classification.
- It uses the MobileNetV2 architecture as the base model, which is pre-trained on the ImageNet dataset.
- The code adds a global average pooling layer and a fully connected dense layer with 6 units and a softmax activation function for multi-class classification.
- The resulting model is assigned to the variable `my_model`.

4. Transfer Learning

- The code freezes the first 50 layers of the MobileNetV2 base model for transfer learning. Only the added dense layers will be trained on the current dataset.

5. Model Compilation

- The model is compiled using the Adam optimizer with a learning rate of 0.001, categorical cross-entropy loss, and accuracy as the evaluation metric.

6. Callbacks

- The code defines a learning rate scheduler function `learningRate_scheduler` to reduce the learning rate after the first 10 epochs.
- It creates a `LearningRateScheduler` object, `learningRate_schedule`, which adjusts the learning rate during training based on the scheduler function.

7. Model Training

- The code trains the model using the `fit` method, providing the training and validation data generators, specifying 20 epochs, and utilizing the learning rate scheduler as a callback.
- An optional `EarlyStopping` callback is provided but commented out, which can be used to stop training early if the validation loss doesn't improve for a specified number of epochs.

8. Model Evaluation

- The code evaluates the model on the validation data using the `evaluate` method and prints the validation accuracy.

9. Confusion Matrix

- The code computes the confusion matrix and visualizes it using `confusion_matrix` and `ConfusionMatrixDisplay` from scikit-learn. This helps assess the model's performance on different classes.

10. Training and Validation Loss/Accuracy Plot

- Finally, the code plots the training and validation loss and accuracy over epochs using Matplotlib. These plots help visualize how well the model is learning and whether it is overfitting or underfitting.

The code is structured for training a deep learning model for image classification with transfer learning, using data augmentation and various callbacks to monitor and adjust the training process. It also provides a visualization of model performance through the confusion matrix and training/validation loss and accuracy plots.