VGG19

1.

1. Importing necessary libraries:

  - `numpy` as `np`: For numerical operations on arrays.

  - `cv2`: For image processing using OpenCV.

  - `os`: For operating system related operations, like directory listing.

  - `re`: For regular expressions (though not used in this code).

  - `tensorflow`: For deep learning functionalities.

  - `ImageDataGenerator`, `preprocess_input` from `tensorflow.keras.preprocessing.image`: For image data preprocessing.

  - `drive` from `google.colab`: For mounting Google Drive.

2. Mounting Google Drive:

  - Mounts Google Drive to access files.

3. Load dataset and preprocess:

  - Define paths to directories containing training and testing images for healthy and schizophrenia-affected brains.

4. Function `load_dataset`:

  - Takes four arguments: paths to healthy and schizophrenia directories for training and testing.

  - Initializes empty lists to store images and labels for training and testing.

  - Loops through images in the healthy and schizophrenia directories for training and testing:

    - Loads each image using `load_img` from `tensorflow.keras.preprocessing.image` and resizes it to (224, 224) pixels.

    - Converts the image to an array using `img_to_array`.

    - Appends the image array to the respective `images_train` or `images_test` list.

    - Appends the label (0 for healthy, 1 for schizophrenia) to the respective `labels_train` or `labels_test` list.

  - Returns numpy arrays for images and labels for training and testing.

5. Calling `load_dataset` with paths to load the dataset.

6. Printing dataset information:

- Prints the length of `X_train_dl` (number of training images), length of the first element of `X_train_dl` (number of channels in the first image), shape of the first image in `X_train_dl`, length of `X_test_dl` (number of testing images), length of `y_train_dl` (number of labels for training), and length of `y_test_dl` (number of labels for testing).

7. Displaying images:

- Uses `cv2_imshow` from `google.colab.patches` to display the 6th image from both the training and testing datasets (`X_train_dl[5]` and `X_test_dl[5]`).

This code sets up the dataset for a deep learning model by loading images from specified directories, resizing them, and storing them as arrays along with their labels. The dataset is then used for training and testing a model to classify healthy and schizophrenia-affected brains.

2.

This code segment prepares the dataset for training a deep learning model using VGG19 architecture for image classification. Here's a breakdown of the code:

1. Importing necessary libraries:

- `os`: For operating system related operations.
- `time`: For measuring time taken for training.
- `numpy` as `np`: For numerical operations on arrays.
- `tensorflow`: For deep learning functionalities.
- `layers` from `tensorflow.keras`: For building neural network layers.
- `ImageDataGenerator`, `preprocess_input` from `tensorflow.keras.preprocessing.image`: For image data preprocessing.
- `VGG19` from `tensorflow.keras.applications.vgg19`: Pre-trained VGG19 model.
- `ModelCheckpoint`, `EarlyStopping`, `ReduceLROnPlateau` from `tensorflow.keras.callbacks`: For model callbacks.
- `Model`, `Sequential` from `tensorflow.keras.models`: For defining the model architecture.
- `train_test_split` from `sklearn.model_selection`: For splitting the dataset into training and testing sets.
- Various metrics from `sklearn.metrics`: For evaluating the model.
- `to_categorical` from `tensorflow.keras.utils`: For converting labels to categorical format.
- `matplotlib.pyplot` as `plt`: For plotting graphs.

2. Splitting the dataset:

- `X_train_dl_split` and `y_train_dl_split` are the training images and labels, respectively.
- `X_test_dl_split` and `y_test_dl_split` are the testing images and labels, respectively.
- `to_categorical` is used to convert the labels to a categorical format.

3. Printing dataset information:

- Prints the length of `X_train_dl_split` (number of training images), length of the first element of `X_train_dl_split` (number of channels in the first image), shape of the first image in `X_train_dl_split`, length of `X_test_dl_split` (number of testing images), length of `y_train_dl_split` (number of labels for training), and length of `y_test_dl_split` (number of labels for testing).

This code sets up the dataset for training a deep learning model using VGG19 architecture. The dataset is split into training and testing sets, and labels are converted to a categorical format. The model can now be defined and trained using this prepared dataset.

3.

This code segment builds and compiles a deep learning model using the VGG19 architecture for image classification. Here's a breakdown of the code:

1. `print_layer_trainable` function:

- Iterates through all layers in the base model (VGG19) and prints whether each layer is trainable or not, along with its name.

2. Build and compile the model:

- `base_model`: Loads the pre-trained VGG19 model with weights from ImageNet, excluding the top (classification) layers, and sets the input shape to (224, 224, 3).
- Freezes all layers in the base model (sets `trainable` attribute to False).
- Unfreezes the last 15 layers for fine-tuning by setting their `trainable` attribute to True.
- `model`: Defines the full model architecture using a Sequential model.
  - Rescales input images to the range [0, 1].
  - Appends the base model (VGG19) to the model.
  - Adds a Flatten layer to flatten the output from the base model.
  - Adds Dense layers with ReLU activation and Dropout for regularization.
  - Adds a Dense output layer with softmax activation for binary classification.

- Compiles the model using the Adam optimizer, binary crossentropy loss, and accuracy metric.

3. Model Checkpoint:

- `tl_checkpoint_h5` and `tl_checkpoint_tf` are ModelCheckpoint callbacks that monitor the validation accuracy and save the best weights to files in HDF5 and TensorFlow formats, respectively.

This code segment sets up a deep learning model for binary image classification using transfer learning with VGG19. The model is compiled and ready for training.

4.

This code segment sets up ImageDataGenerator instances for data augmentation and creates generators for training and validation data. Here's a breakdown of the code:

1. `train_datagen`:

- Creates an `ImageDataGenerator` object for data augmentation.
- Specifies various augmentation options:
  - `rotation_range`: Degree range for random rotations.
  - `width_shift_range` and `height_shift_range`: Fraction of total width or height for random horizontal or vertical shifts.
  - `shear_range`: Shear intensity in radians.
  - `zoom_range`: Range for random zoom.
  - `horizontal_flip`: Randomly flips images horizontally.
  - `fill_mode`: Strategy for filling in newly created pixels.

2. `train_test_split`:

- Splits the original training data (`X_train_dl_split`, `y_train_dl_split`) into training and validation sets (`X_train`, `X_val_dl_split`, `y_train`, `y_val_dl_split`) using a 80-20 split ratio.

3. `train_generator` and `val_generator`:

- Create generator objects using the `flow` method of `train_datagen`.
- `train_generator` generates augmented images and labels for training.
- `val_generator` generates augmented images and labels for validation.

These generators will be used to train the model with augmented images, which can improve the model's performance and generalization.

5.

This code segment calculates the confusion matrix, specificity, and recall, and then plots the training history (accuracy and validation accuracy) of the model. Here's a breakdown of the code:

1. Calculating confusion matrix, specificity, and recall:

   - `y_pred_classes`: Predicted classes obtained by taking the argmax of `y_pred_dl` along axis 1.
   - `y_true_classes`: True classes obtained by taking the argmax of `y_test_dl_split` along axis 1.
   - `conf_matrix`: Confusion matrix computed using `confusion_matrix` from `sklearn.metrics`.
   - `specificity`: Calculated as the true negative rate (TN / (TN + FP)).
   - `recall`: Calculated as the true positive rate or sensitivity (TP / (TP + FN)).

2. Plotting training history:

   - Creates a figure with a size of (12, 8).
   - Plots training accuracy and validation accuracy against epochs.
   - Labels the x-axis as 'Epochs' and the y-axis as 'Accuracy'.
   - Sets the title of the plot as 'Accuracy vs. Epochs'.
   - Adds a legend to differentiate between training and validation accuracy.

These visualizations and metrics help in understanding the performance and behavior of the model during training and evaluation.

6.

Roc curve

7.parameters

8.file generation