Optimizing Bio-Inspired Deep Convolutional Neural Networks

Code Documentation

Author: Luis Morales Layja
Candidate Number: 282549

Date: 24/08/2024

Table of Contents

1 Introduction

The code used for this research project was based on the work by Evans et al., 2022. New functionalities were implemented to investigate the impact of incorporating a Retinal Information Bottleneck (RIB) and Recurrent Connections (RCs) into the VGG-16 architecture with Gabor filters in the first convolutional layer, as detailed in the official dissertation project document.

2 Directory Structure

The expected directory structure before training the models is as follows:

```
bionet
    config.py
    explain.py
    __init__.py
    plots.py
    assess.py
    bases.py
    plots.py
    utils.py
    preparation.py
data
    CIFAR-10G
model.py
README.md
```

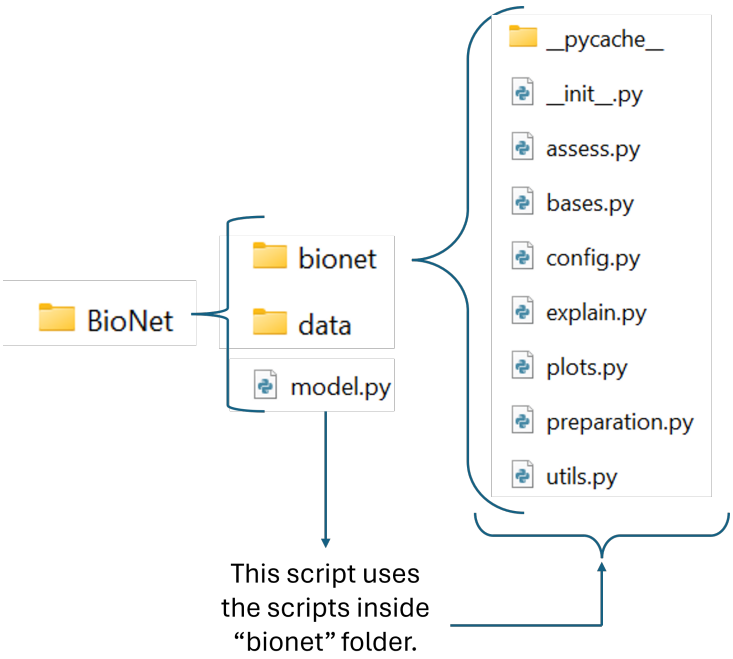

Figure 1: Ideal directory structure.

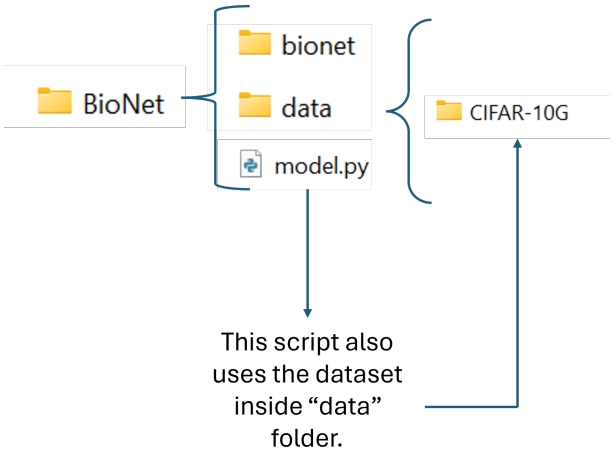

Figure 2: Ideal directory structure for CIFAR-10G.

As shown in Figures 1 and 2, the script "model.py" utilizes the functions and resources from the other scripts stored in the "bionet" folder. The "data" folder contains the "CIFAR-10G" dataset.

3 Structure

Given the extensive nature of the code, which encompasses many functionalities, several of which were irrelevant and not used for the dissertation project, a schematic overview of the relevant parts of the workflow involved in the simulations conducted to train the

models and reproduce the results in the research project is introduced here. Below is a brief description of each script used:

- __init__.py: This file initializes the Python package and allows the import of the modules within the "bionet" directory.

- assess.py: Generates the necessary data from the results. Specifically, it produces CSV files containing accuracy data of the models on each training dataset and for each type of added noise. These data are later used for plotting graphs.

- bases.py: Contains implementations of Convolutional Neural Network (CNN) architectures, specifically ResNet, ALL-CNN, and VGG-16. It also includes the additions made for the dissertation project, particularly the functions related to the Retinal Information Bottleneck (RIB) and Recurrent Connections (RCs).

- preparation.py: Provides functions for preprocessing, adding various types of noise to images, and generating datasets using the "ImageDataGenerator" function from TensorFlow and Keras. It prepares the input data and applies different types of noise during model evaluation.

- config.py: Contains constants and key names used throughout the project. This includes parameters such as models, image statistics, interpolation methods, and training configurations.

- utils.py: Offers auxiliary functions for image preprocessing, model and data loading, and also provides necessary classes to apply custom filters. These filters are used in the first convolutional layer of the models, such as Gabor filters, although it also includes additional filters used in the paper by Evans et al., 2022.

- model.py: This is the main script that calls functions from the other scripts. It accepts parameters to select the base model, whether to incorporate RCs into the base model, and whether to use a bottleneck. Additionally, it allows choosing whether to skip model training and go directly to evaluation, introduces a different training dataset, among other functionalities.

4 Execution

The following information is based on the README.md presented in https://github.com/bdevans/BioN
Additional arguments, which are contributions of the research dissertation project, have been included.

```
usage: model.py [-h] [--convolution CONVOLUTION] [--base BASE]
    [--pretrain]
                [--architecture ARCHITECTURE] [--recurrent]  [--
                    bottleneck BOTTLENECK] [--continue_train] [--
                    interpolation INTERPOLATION]
                [--optimizer {SGD,RMSprop,Adagrad,Adadelta,Adam,
                    Adamax,Nadam}]
                [--lr LR] [--decay DECAY] [--use_initializer]
                [--internal_noise INTERNAL_NOISE] [--trial TRIAL]
                [--label LABEL] [--seed SEED] [-t] [--
                    recalculate_statistics]
                [--epochs EPOCHS] [--batch BATCH] [--image_path
                    IMAGE_PATH]
                [--train_image_path TRAIN_IMAGE_PATH] [--
                    test_generalisation]
                [--invert_test_images INVERT_TEST_IMAGES]
                [--test_perturbations] [--data_augmentation]
                [--extra_augmentation] [-c] [--skip_test] [-l]
                    [--save_images]
                [-p] [--gpu GPU] [--project_dir PROJECT_DIR] [-v
                    VERBOSE]

optional arguments:
  -h, --help            show this help message and exit
  --convolution CONVOLUTION
                        Name of convolutional filter to use
  --base BASE           Name of model to use
  --pretrain            Flag to use pretrained ImageNet weights
     in the model
  --architecture ARCHITECTURE
                        Parameter file (JSON) to load
```

```
22  --interpolation INTERPOLATION
23                        Method to interpolate the images when
                             upscaling.
24                        Default: 0 ("nearest" i.e. no
                             interpolation)
25  --optimizer {SGD,RMSprop,Adagrad,Adadelta,Adam,Adamax,Nadam}
26                        Name of optimizer to use: https://keras.
                             io/optimizers/
27  --lr LR, --learning_rate LR
28                        Learning rate for training
29  --decay DECAY         Optimizer decay for training
30  --use_initializer     Flag to use the weight initializer (then
       freeze
31                        weights) for the Gabor filters
32  --internal_noise INTERNAL_NOISE
33                        Standard deviation for adding a Gaussian
                             noise layer
34                        after the first convolutional layer
35  --trial TRIAL         Trial number for labeling different runs
       of the same
36                        model
37  --label LABEL         For labeling different runs of the same
       model
38  --seed SEED           Random seed to use
39  -t, --train           Flag to train the model
40  --recalculate_statistics
41                        Flag to recalculate normalisation
                             statistics over the
42                        training set
43  --epochs EPOCHS       Number of epochs to train model
44  --batch BATCH         Size of mini-batches passed to the
       network
45  --image_path IMAGE_PATH
46                        Path to image files to load
47  --train_image_path TRAIN_IMAGE_PATH
48                        Path to training image files to load
```

```
  --test_generalisation
                        Flag to test the model on sets of
                           untrained images
  --invert_test_images INVERT_TEST_IMAGES
                        Flag to invert the luminance of the test
                           images
  --test_perturbations  Flag to test the model on perturbed
     images
  --data_augmentation   Flag to train the model with data
     augmentation
  --extra_augmentation  Flag to train the model with additional
     data
                        augmentation
  -c, --clean           Flag to retrain model
  --skip_test           Flag to skip testing the model
  -l, --log             Flag to log training data
  --save_images         Flag to save preprocessed (perturbed)
     test images
  -p, --save_predictions
                        Flag to save category predictions
  --gpu GPU             GPU ID to run on
  --project_dir PROJECT_DIR
                        Path to the root project directory
  -v VERBOSE, --verbose VERBOSE
                        Verbosity level
```

Listing 1: Usage of model.py.

To run the training sessions as well as their evaluations, first, navigate to the directory containing "model.py" (as specified in the directory structure in Section 2). Once there, enter the following command in the terminal, depending on the model you wish to train and evaluate:

Baseline Model Training and Evaluation:

```
python model.py --convolution Gabor --base VGG16 --epochs 100 --
   train --use_initializer --clean  --use_initializer --
   data_augmentation --test_generalisation --
```

```
   recalculate_statistics --test_perturbations --save_predictions
    --log --verbose 1 --gpu 0 --label Bottleneck --batch 32 --
   seed 42
```

Retinal Information Bottleneck Model Incorporation, Training, and Evaluation:

```
python model.py --convolution Gabor --base VGG16 --epochs 100 --
   train --use_initializer --clean --bottleneck 2 --
   use_initializer --data_augmentation --test_generalisation --
   test_perturbations --save_predictions --recalculate_statistics
    --log --verbose 1 --gpu 0 --label Bottleneck --batch 32 --
   seed 42
```

Recurrent Connections Model Incorporation, Training, and Evaluation:

```
python model.py --convolution Gabor --base VGG16 --epochs 100 --
   train --clean --recurrent --use_initializer --
   data_augmentation --test_generalisation --test_perturbations
   --save_predictions --recalculate_statistics --log --verbose 1
   --gpu 0 --label Recurrent --batch 32 --seed 42
```

Retinal Information Bottleneck and Recurrent Connections Model Incorporation, Training, and Evaluation:

```
python model.py --convolution Gabor --base VGG16 --epochs 100 --
    train --clean --recurrent --bottleneck 2 --use_initializer --
    data_augmentation --test_generalisation --test_perturbations
    --save_predictions --recalculate_statistics --log --verbose 1
    --gpu 0 --label Recurrent --batch 32 --seed 42
```

5 Contributions

5.1 model.py

```
107  parser.add_argument('--recurrent', action='store_true',
108                      help='Flag to add recurrent connections')
109  parser.add_argument('--bottleneck', type=int, default=None,
110                      help='Number of channels for the retinal bottleneck. If not set, the bottleneck is not used.')
111  parser.add_argument('--continue_train', action='store_true',
```

Figure 3: These additions enable `model.py` to receive the arguments "recurrent," "bottleneck," and "continue_train." "recurrent" is a boolean variable that is set to True when provided as an argument, otherwise it defaults to False. "bottleneck" is an integer variable that defines the size of the bottleneck. If not set, no bottleneck is incorporated. "continue_train" provides an easier way to resume training.

```
199  recurrent = args['recurrent']
200  bottleneck_width = args['bottleneck']
201  continue_training = args["continue_train"]
```

Figure 4: Additions in lines 199, 200, and 201 of the `model.py` script. The arguments are assigned to the corresponding variables.

```
738  # Retinal Information Bottleneck incorporation
739  if bottleneck_width != None:
740      model = retinal_bottleneck(model, bottleneck_width)
741
742  # Adding recurrent connections
743  if recurrent:
744      model = recurrent_connections_LSTM(model)
```

Figure 5: Additions in lines 738-744. When the "bottleneck" argument is set to an integer, the "model" is modified to incorporate the bottleneck with the specified size. Similarly, if "recurrent" is True, the "model" is modified to incorporate RCs. These functions are introduced in the `bases.py` script.

```
973         if continue_training:
974             checkpoint_path = os.path.join(models_dir, "model.ckpt")
975             latest_checkpoint = tf.train.latest_checkpoint(models_dir)
976             if latest_checkpoint:
977                 model.load_weights(latest_checkpoint)
978                 print(f"Loaded weights from {latest_checkpoint}")
979             else:
980                 print("No checkpoint found. Training from scratch.")
981         else:
982             print("Training from scratch.")
```

Figure 6: Additions from lines 973 to 982. If "continue_train" is set to True, the script checks for available weights to resume the previous training. If no weights are found, training starts from scratch.

5.2 bases.py

```
652     def retinal_bottleneck(model, N):
653         """
654             Defines a bottleneck with three convolutional
655             layers placed at the beginning of the current model.
656 
657             Parameters
658             ----------
659             model : the current model in which the bottleneck will be implemented
660             N : Number of neurons in the second and third layer of the bottleneck
661 
662             Returns
663             -------
664             model_with_bottleneck : Model object of tensorflow.keras
665                 The model introduced as argument with the bottleneck integrated
666                 at the beggining of it.
667          """
```

Figure 7: Function added to the `bases.py` script from lines 652 to 818. This function incorporates the RIB into the current model (see `BioNet_Code_Layja/bionet/bases.py` for the full function).

```
822 def recurrent_connections_LSTM(model):
823     """
824         Places LSTM in the first convolutional layer of "model"
825         It replaces the first convolutional layer for a ConvLSTM (convolutional recurrent layer using LSTM)
826         with the same kernels and parameters.
827 
828         Arguments
829         ----------
830         model :the current model in which the recurrent connections will be implemented
831 
832         Returns
833         -------
834         model_with_recurrent : Model object of tensorflow.keras
835             The model introduced as argument with recurrent connections (LSTM) in its first convolutional layer
836     """
837     print("--------------\nRecurrent Connections \n-----------------")
```

Figure 8: Function added to the `bases.py` script from lines 822 to 912. This function incorporates RCs into the current model in the first convolutional layer. If the current model has a bottleneck, the RCs are incorporated in the first layer after the bottleneck (see `BioNet_Code_Layja/bionet/bases.py` for the full function).

6 Results

The data obtained from the aforementioned code represents the results. An additional Python script was used to graph this data. This script utilizes the data collected by the `model.py` script after evaluating the models. The script of model.py generates a "results" folder after evaluating the models. This folder contains the accuracy of all the models in the different datasets with the different noise level in csv format. This data needs to be unified as it is in the existing csv's inside the following directories: results /metrics / cifar10

- results /metrics / generalisation
- results /metrics / perturbed_cifar
- results /metrics / perturbed_generalisation

The resulting graphs are those presented in the "Results" section of the original research project document. The script and the resulting data can be found in the "results" folder provided. It contains "results_visualizations.py" and "functions.py" scripts, where functions have been done and used to visualize the data.

References

Evans, B. D., Malhotra, G., & Bowers, J. S. (2022). Biological convolutions improve dnn robustness to noise and generalisation. *Neural Networks*, *148*, 96–110. https://doi.org/10.1016/j.neunet.2021.12.005