

Code Documentation

Daniel Sobral Blanco*
*Department of Theoretical Physics,
University of Geneva*

June 7, 2023

Introduction

This document provides the documentation for each `.py` script used in the code developed for the project. It includes descriptions of classes, methods, and their usage. The instructions to run the code from terminal are contained in the `README.md` file.

The code has been generated with the help of ChatGPT, using both GPT-3.5 and GPT-4 versions. It has proven very useful at the time of writing the scripts and debugging the methods. The docstrings, code documentation and `README.md` file are entirely generated by the model.

`run.py`

This is the script that runs the code. It can be used from terminal or in another Python interface such as Jupyter notebooks.

```
1  def run(config_file, save_model=True, save_history=True, seed=48):
2      """
3      Run the Galaxy Zoo training and evaluation process.
4
5      Parameters
6      -----
7      config_file : str
8          Path to the configuration file. Must be a .yaml file.
9      save_model : bool, optional
10         Whether to save the trained model. Defaults to True.
11      save_history : bool, optional
12         Whether to save the training history. Defaults to True.
13      seed : int, optional
14         Seed for reproducibility. Defaults to 48.
15      """
```

If `save_model` and `save_history` flags are set to `True` (default value), the code will save the model `.h5` file and its training history in the correspondents paths. One must set this paths in the `config_file` (see the `README.md`). The code also produces plots of the loss and the first metric passed in the `metrics` list. It also generates the confusion matrices.

`models.py`

This is the main script of the code. It contains a `class` to build, compile and train CNN models with different architectures. It allows to build standard, single output models and multi-output models

*Email: daniel.sobralblanco@unige.ch

with **base** architecture of the form:

$$\text{IN} + n \cdot (\text{Conv2D} + \text{BN} + \text{MaxPool2D}) + \text{Flattening} + n \cdot (\text{Dense} + \text{Maxout} + \text{Dropout}) + \text{OUT}$$

Here BN stands for Batch Normalization. The Flattening part can be chosen to be the standard Flatten layer, or a GlobalMaxPooling2D layer. Maxout operation is fixed to $n = 2$, which averages out every two hidden neurons. The OUT layer will be different depending on the type of model one wants to build (see the report for more details about the model types).

GalaxyZooClassifier

Custom class to build classifiers for the Galaxy Zoo challenge. This class builds a custom model architecture, allowing for base, hierarchical, and branch models. Different parameters allow for flexible customization of the architecture, such as number and type of convolutional layers, dense layers, batch normalization, pooling method, and dropout.

```
1  def __init__(self, model_type, input_shape, n_classes, conv_layers, dense_units,
2      batch_normalization = False, activation='relu',
3      pool_size=(2,2), flattening = 'Flatten', class_weights = False,
4      out_activation = 'softmax', dropout_rate=None, max_out=False,
5      early_stop_patience=None, monitor='val_loss',
6      data_augmentation=None):
7      """
8      Initializes a GalaxyZooClassifier object.
9
10     Parameters
11     -----
12     model_type: str
13         The type of model to be built. Should be either 'base', 'hierarchy', or '
14         ↳ branch'.
15     input_shape: tuple
16         The shape of the input data.
17     n_classes: int
18         The number of classes for classification.
19     conv_layers: list of tuples
20         The configuration of the convolutional layers, with each tuple containing
21         ↳ (number of filters, kernel size).
22     dense_units: list of int
23         The number of units for each dense layer.
24     batch_normalization: bool, optional (default=False)
25         Whether to use batch normalization.
26     activation: str, optional (default='relu')
27         The activation function to use in the layers.
28     pool_size: tuple, optional (default=(2, 2))
29         The pool size for the max pooling layers.
30     flattening: str, optional (default='Flatten')
31         The type of flattening to be used before the dense layers.
32     class_weights: bool, optional (default=False)
33         Whether to use class weights during training.
34     out_activation: str, optional (default='softmax')
35         The activation function to use in the output layer.
36     dropout_rate: float, optional
37         The dropout rate to use after each layer.
38     max_out: bool, optional (default=False)
39         Whether to use Maxout in the dense layers.
```

```

38     early_stop_patience: int, optional
39         The number of epochs with no improvement after which training will be
           ↪ stopped. If None, early stopping is not used.
40     monitor: str, optional (default='val_loss')
41         Quantity to be monitored for early stopping.
42     data_augmentation: dict, optional
43         The parameters for data augmentation. If None, data augmentation is not
           ↪ used.
44
45     Raises
46     -----
47     ValueError:
48         If 'model_type' is not 'base', 'hierarchy', or 'branch'.
49     """

1  def build_convolutional_blocks(self, input_layer):
2      """
3      Builds a sequence of convolutional layers based on the 'conv_layers'
           ↪ attribute.
4
5      Parameters
6      -----
7      input_layer: tensorflow.python.keras.engine.input_layer.InputLayer
8          The input layer of the model.
9
10
11
12     Returns
13     -----
14     x: tensorflow.python.keras.engine.base_layer
15         The output of the last layer in the block.
16
17     Notes
18     -----
19     This method builds a sequence of convolutional layers, each followed by an
           ↪ activation function, a max pooling layer,
20     and possibly a batch normalization layer, based on the 'conv_layers'
           ↪ attribute. The number of convolutional layers,
21     their filter sizes, and their kernel sizes are determined by 'conv_layers'.
           ↪ After the last convolutional layer,
22     the output is flattened if 'flattening' is 'Flatten', or global max pooling
           ↪ is applied if 'flattening' is not 'Flatten'.
23     """

1  def build_dense_blocks(self, input_layer):
2      """
3      Builds a sequence of fully connected (dense) layers based on the '
           ↪ dense_units' attribute.
4
5      Parameters
6      -----
7      input_layer: tensorflow.python.keras.engine.base_layer
8          The input layer of the model.
9

```

```

10     Returns
11     -----
12     x: tensorflow.python.keras.engine.base_layer
13         The output of the last layer in the block.
14
15     Notes
16     -----
17     This method builds a sequence of dense layers, each followed by an
18         ↪ activation function, and optionally a dropout layer,
19     based on the 'dense_units' attribute. The number of dense layers and their
20         ↪ units are determined by 'dense_units'.
21     If 'max_out' is True, a maxout layer is applied after each dense layer. If '
22         ↪ dropout' is not None, a dropout layer is
23     applied after each dense layer with 'dropout' rate.
24     """
25
26 def build_architecture(self, input_layer):
27     """
28     Builds the architecture of the model by combining convolutional and dense
29         ↪ blocks.
30
31     Parameters
32     -----
33     input_layer: tensorflow.python.keras.engine.base_layer
34         The input layer of the model.
35
36     Returns
37     -----
38     DenseBlocks: tensorflow.python.keras.engine.base_layer
39         The output of the last layer in the dense block.
40
41     Notes
42     -----
43     This method calls 'build_convolutional_blocks' and 'build_dense_blocks'
44         ↪ methods to build the architecture
45     of the model. The output of the convolutional blocks is passed as input to
46         ↪ the dense blocks.
47     """
48
49 def build_model(self):
50     """
51     Builds the base model with the defined architecture.
52
53     Returns
54     -----
55     model: keras.Model
56         The base model with the defined architecture.
57
58     Notes
59     -----
60     This method first creates an input layer with the shape defined in self.
61         ↪ in_shape.
62     It then calls the 'build_architecture' method to build the architecture of
63         ↪ the model.

```

```

14         The output layer is a dense layer with a number of units equal to the number
           ↳ of classes,
15         and activation function defined in self.out_activation.
16         """

1  def build_hierarchy_model(self, n_classes, embeddings = False):
2         """
3         Builds a model with a hierarchical output architecture.
4
5         Parameters
6         -----
7         n_classes: tuple
8             A tuple specifying the number of classes for each output layer.
9
10        Returns
11        -----
12        model: keras.Model
13            The model with the defined hierarchical architecture.
14
15        Notes
16        -----
17        This method first creates an input layer with the shape defined in self.
           ↳ in_shape.
18        It then calls the 'build_architecture' method to build the architecture of
           ↳ the model.
19        The output layers are dense layers with number of units equal to the number
           ↳ of classes
20        for each layer, and activation function defined in self.out_activation.
           ↳ These layers
21        are then processed to enforce the class hierarchy. The final model has
           ↳ multiple outputs,
22        one for each class hierarchy.
23        """

1  def build_branch_model(self, n_classes):
2         """
3         Builds a model with a branching output architecture.
4
5         Parameters
6         -----
7         n_classes: tuple
8             A tuple specifying the number of classes for each output layer.
9
10        Returns
11        -----
12        model: keras.Model
13            The model with the defined branching architecture.
14
15        Notes
16        -----
17        This method first creates an input layer with the shape defined in self.
           ↳ in_shape.
18        It then builds the first level of the model using the 'build_architecture'
           ↳ method

```

```

19     and adds an output layer. For the second level, it builds two branches each
    ↪ with
20     its own architecture and output layer. The outputs of the first level and
    ↪ each
21     branch of the second level are concatenated and then processed with dense
    ↪ blocks.
22     The outputs of the dense blocks are then processed to enforce the class
    ↪ hierarchy.
23     The final model has multiple outputs, one for each branch.
24     """

1  def compile_model(self, learning_rate, loss_function, metrics=['accuracy']):
2      """
3      Compiles the model with the specified parameters.
4
5      Parameters
6      -----
7      learning_rate: float
8          The learning rate to use for the Adam optimizer.
9
10     loss_function: str or dict
11         The loss function to use. If it's a string, it is used for all outputs.
12         If it's a dictionary, it should map output names to loss functions.
13
14     metrics: list, optional
15         The list of metrics to compute during training and testing.
16         Default is ['accuracy'].
17
18     Raises
19     -----
20     Exception
21         If the model has not been built before this method is called.
22
23     Notes
24     -----
25     The method first checks if the model has been built. If not, it raises an
    ↪ exception.
26     It then creates an Adam optimizer with the specified learning rate and
    ↪ compiles the model
27     with this optimizer, the specified loss function, and the metrics. If there
    ↪ are multiple
28     outputs and the loss function is a string, it uses this loss function for
    ↪ all outputs.
29     If the loss function is a dictionary, it should map output names to loss
    ↪ functions.
30     """

1  def fit_model(self, X_train, y_train, X_val, y_val, learning_rate,
2                loss_function = 'categorical_crossentropy', metrics=['accuracy'],
3                batch_size=100, epochs=25, threshold = 0.1, take_weights_log=True):
4      """
5      Fits the model to the training data.
6
7

```

```

8      Parameters
9      -----
10     X_train: array
11         The training inputs.
12
13     y_train: array
14         The training targets.
15
16     X_val: array
17         The validation inputs.
18
19     y_val: array
20         The validation targets.
21
22     learning_rate: float
23         The learning rate to use for the Adam optimizer.
24
25     loss_function: str or dict
26         The loss function to use. If it's a string, it is used for all outputs.
27         If it's a dictionary, it should map output names to loss functions.
28
29     metrics: list, optional
30         The list of metrics to compute during training and testing.
31         Default is ['accuracy'].
32
33     batch_size: int, optional
34         The number of samples per batch. Default is 100.
35
36     epochs: int, optional
37         The number of epochs to train the model. Default is 25.
38
39     threshold: float, optional
40         The threshold for the class weights. Default is 0.1.
41
42     take_weights_log: bool, optional
43         Whether to take the log of the class weights. Default is True.
44
45     Raises
46     -----
47     Exception
48         If the model has not been built and compiled before this method is called
49         ↪ .
50
51     Notes
52     -----
53     The method first checks if the model has been built and compiled. If not, it
54     ↪ raises an exception.
55     It then compiles the model with the specified parameters and fits it to the
56     ↪ training data. It also computes
57     class weights if specified, and uses a separate loss function for each
58     ↪ output if there are multiple outputs.
59     The training process can also include early stopping and data augmentation
60     ↪ if specified.
61     """

```

```

1  def save_model(self, models_path, name, diagram = False):
2      """
3      Saves the model and optionally its architecture diagram.
4
5      Parameters
6      -----
7      models_path: str
8          The path to the directory where the model should be saved.
9
10     name: str
11         The name of the model. The model will be saved as a .h5 file with this
12         ↪ name.
13
14     diagram: bool, optional
15         Whether to save a diagram of the model's architecture. Default is False.
16         If True, the diagram is saved as a .pdf file with the same name as the
17         ↪ model.
18
19     Notes
20     -----
21     The method first saves the model as a .h5 file in the specified directory.
22     If the diagram flag is set to True, it also saves a diagram of the model's
23     ↪ architecture as a .pdf file in the same directory.
24     """

```

The scripts also contain these additional methods:

```

1  def F1_score(y_true, y_pred):
2      """
3      Compute the F1 Score between the true and predicted labels.
4
5      The F1 score is the harmonic mean of precision and recall. Compared to the
6      ↪ regular mean,
7      the harmonic mean gives much more weight to low values. The best value is 1 and
8      ↪ the worst is 0.
9
10     Parameters
11     -----
12     y_true : array-like
13         Ground truth (correct) target values.
14     y_pred : array-like
15         Estimated targets as returned by a classifier.
16
17     Returns
18     -----
19     f1_sc : float
20         The F1 Score between 0 and 1.
21
22     Notes
23     -----
24     The F1 score is especially useful for balanced datasets, as it takes both false
25     ↪ positives
26     and false negatives into account.
27     """

```



```

1  def calculate_class_weights(y_train, threshold, take_weights_log=True):
2      """
3      Calculate class weights for imbalanced datasets. The function only considers
4          ↪ classes that have a representation
5      above a certain threshold.
6
7      Parameters
8      -----
9      y_train : numpy.ndarray
10         The one-hot encoded target variables for the training set. Shape should be (
11             ↪ n_samples, n_classes).
12     threshold : float
13         The minimum representation threshold for a class to be considered valid.
14             ↪ Should be between 0 and 1.
15
16     Returns
17     -----
18     class_weight_dict : dict
19         A dictionary mapping from class index to weight. Only includes classes above
20             ↪ the representation threshold.
21     valid_classes : numpy.ndarray
22         An array of the classes that are above the representation threshold.
23
24     Notes
25     -----
26     If a class is below the representation threshold, it is not included in the
27         ↪ output dictionary.
28     """

```

eval.py

Script containing utilities for the evaluation of the model performance: calculating the scores, confusion matrices, plotting history, error analysis.

```

1  def evaluate_my_model(model, test_data, set_name = 'testing'):
2      """
3      Evaluates a trained model using the provided testing data.
4
5      Parameters
6      -----
7      model : keras.Model
8         The trained model to be evaluated.
9      test_data : list
10         A list containing the testing data. The first element of the list
11         is expected to be the input data (features), and the second element
12         is expected to be the labels (target).
13     set_name : str, optional
14         A string specifying the name of the testing dataset. It is only used for
15         printing purposes to inform the user about the dataset being used for
16             ↪ evaluation.
17         The default value is 'testing'.
18
19

```

```

20 Notes
21 -----
22 This function assumes that the 'test_data' list contains exactly two elements.
23 The first element is expected to be the input data (features),
24 and the second element is expected to be the labels (target).
25
26 This function uses the 'evaluate' method of the keras.Model class to evaluate
    ↪ the model.
27 The 'evaluate' method returns a list of metrics values. The metrics are
    ↪ specified
28 during the compilation of the model.
29
30 After the model is evaluated, this function prints the name of each metric
31 and its corresponding value.
32 """

1 def plot_history(history, metric, labels, size=(18, 6), folder_path=None):
2     """
3     Plots the total training and validation losses and specified metrics from a
        ↪ model's history.
4
5     Parameters
6     -----
7     history : keras.callbacks.History object
8         The history object generated by the training method 'model.fit()'.
9     metric : str
10        The base name of the metric(s) to be plotted. The function will plot all
        ↪ metrics
11        that contain this string in their name.
12    labels : tuple
13        A tuple containing labels for the generated plot. The tuple is expected
14        to have two elements - 'model_name' and 'learning_rate'.
15    size : tuple, optional
16        A tuple specifying the size of the generated plot. The default size is (12,
        ↪ 5).
17    folder_path : str, optional
18        A string specifying the path to the folder where the plot should be saved.
19        If this is set to None, the plot is not saved. The default value is None.
20
21    Returns
22    -----
23    None
24
25    Notes
26    -----
27    This function generates two plots - one for the total loss and one for the
        ↪ specified metric(s).
28    Each plot shows the values of the respective quantity for both the training and
29    validation data across the epochs of training. If the model has multiple outputs
        ↪ ,
30    the function plots the specified metric(s) for each output on the right subplot.
31
32    If 'folder_path' is not None, this function saves the generated plot in that
        ↪ folder

```

```

33     as a pdf file. The file is named as '<model_name>_<learning_rate>.pdf'. It is
        ↳ assumed
34     that 'labels' is a tuple containing exactly two elements - 'model_name' and '
        ↳ learning_rate'.
35
36     This function uses matplotlib.pyplot for generating the plots. This must be
        ↳ installed
37     in your environment in order to use this function.
38     """

1  def ConfusionMatrix(model, data_test, name, folder_path=None):
2      """
3      Generates and plots a confusion matrix for the given model and test data.
4
5      Parameters
6      -----
7      model : keras Model object
8          The model for which the confusion matrix is to be generated.
9      data_test : tuple
10         A tuple containing the test data. The tuple is expected to contain
11         two numpy arrays - 'X_test' and 'y_test', where 'X_test' is the
12         input data and 'y_test' are the true labels.
13         In the case of multi-output models, 'y_test' should be a list
14         containing the 'y_test[i]' arrays for each output.
15     name : str
16         A string to be used in the title of the generated plot.
17     folder_path : str, optional
18         A string specifying the path to the folder where the plot should
19         be saved. If this is set to None, the plot is not saved. The default
20         value is None.
21
22     Returns
23     -----
24     None
25
26     Notes
27     -----
28     This function generates a confusion matrix for the given model and test
29     data and plots it using seaborn. It first uses the model to predict labels
30     for the test data. It then compares these predicted labels with the true
31     labels to generate the confusion matrix.
32
33     The function generates a plot showing the confusion matrix and saves it
34     in the specified folder if 'folder_path' is not None. The plot is saved
35     as a pdf file named ''ConfusionMatrix_<name>.pdf''.
36
37     If 'y_test' is a list for multi-output models, the function creates a subplot
38     grid based on the number of outputs. Each subplot represents the confusion
39     matrix for a specific output. The overall title for the plot indicates the
40     name of the model and the type of output. The size of each subplot can be
41     adjusted by modifying the 'subplot_size' parameter in the function.
42
43     If 'y_test' is a single numpy array, the function generates a single plot
44     for the confusion matrix.

```

```

45
46     The seaborn and matplotlib libraries must be installed in your environment
47     to use this function.
48     """

1  def error_analysis(model, data_test):
2      """
3      Performs error analysis on the model's predictions and returns indices of the
4          ↪ misclassified data.
5
6      Parameters
7      -----
8      model : keras Model object
9          The trained model which predictions are to be analyzed.
10     data_test : tuple
11         A tuple containing the test data. The tuple is expected to contain
12         two numpy arrays - 'X_test' and 'y_test', where 'X_test' is the
13         input data and 'y_test' are the true labels.
14
15     Returns
16     -----
17     wrong_indices : numpy.ndarray
18         An array containing the indices of the test data that were misclassified.
19     y_test_labels : numpy.ndarray
20         An array of the true labels.
21     y_pred_labels : numpy.ndarray
22         An array of the predicted labels.
23
24     Notes
25     -----
26     This function makes predictions on the test data using the provided model,
27     then compares the predicted labels with the true labels to identify the indices
28     where the model's predictions were incorrect. These indices are then returned
29     along with the true and predicted labels for further analysis.
30     """

31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642

```

```

17     num_img : int, optional
18         Number of misclassified images to be displayed. Default is 6.
19
20     Notes
21     -----
22     This function selects a number of images that were misclassified by the model,
23     then generates a plot displaying these images along with their true and
24     ↪ predicted labels.
25     The plot is saved to a specified location if 'folder_path' is not None.
26     """

```

custom_layers.py

Script containing classes to define the **Maxout** layers and the **MultiOutputDataGenerator**. The latter is a data augmentation generator adapted to work in multi-output set ups.

Maxout

Implements the maxout operation, where the output is the maximum of a group of inputs ($n = 2$ by default). This layer is useful in neural networks to add non-linearity.

```

1  def __init__(self, group_size, **kwargs):
2      """
3      Initialize the Maxout layer.
4
5      Parameters
6      -----
7      group_size : int
8          The number of inputs in each group. The output will be the maximum
9          of each group of inputs.
10
11     Notes
12     -----
13     The __init__ function is special in Python classes. It gets called
14     when you create a new instance of the class. In this case, it is
15     ↪ initializing
16     the group_size attribute and then calling the parent's __init__ function.
17     """
18
19     def build(self, input_shape):
20         """
21         Build the Maxout layer.
22
23         Parameters
24         -----
25         input_shape : tuple
26             The shape of the inputs to this layer.
27
28         Notes
29         -----
30         The purpose of this method is to validate if the last dimension of the input
31         ↪ shape

```

```

31         is divisible by the group_size. If not, it raises a ValueError. This
32         ↪ function is specific
33         to keras.Layer subclasses, which helps in creating the weight variables.
34         """
35     def call(self, inputs):
36         """
37         Call the Maxout layer.
38
39         Parameters
40         -----
41         inputs : Tensor
42             The inputs to the layer.
43
44         Returns
45         -----
46         outputs : Tensor
47             The output of the maxout operation.
48
49         Notes
50         -----
51         The call method defines the layer's computation. For Maxout, this involves
52         ↪ reshaping
53         the input tensor and applying the max operation.
54         """
55     def compute_output_shape(self, input_shape):
56         """
57         Compute the output shape of the Maxout layer.
58
59         Parameters
60         -----
61         input_shape : tuple
62             The shape of the inputs to this layer.
63
64         Returns
65         -----
66         shape : tuple
67             The shape of the output of this layer.
68
69         Notes
70         -----
71         This function calculates the shape of the output of the layer.
72         This is required in Keras, to inform the model about the output size of the
73         ↪ layer.
74         """
75     def get_config(self):
76         """
77         Get the configuration of the Maxout layer.
78
79
80

```

```

81     Returns
82     -----
83     config : dict
84         A dictionary containing the configuration of the Maxout layer.
85
86     Notes
87     -----
88     The get_config method is used when saving the model. It returns a
89     dictionary containing the configuration of the layer.
90     """
91
92     @classmethod
93     def from_config(cls, config):
94         """
95         Create a Maxout layer from its configuration.
96
97         Parameters
98         -----
99         config : dict
100             A dictionary containing the configuration of the Maxout layer.
101
102         Returns
103         -----
104         maxout : Maxout
105             A new Maxout layer instance.
106
107         Notes
108         -----
109         This is a class method that is used to create a new instance of the
110         class using the configuration dictionary returned by get_config.
111         This is essential for Keras' model saving and loading functionality.
112         """

```

MultiOutputDataGenerator

MultiOutputDataGenerator class extends the Sequence class in Keras to generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely. This class allows the handling of multiple output labels and uses the Keras ImageDataGenerator for the actual data augmentation.

```

1
2
3     def __init__(self, x_set, y_set, batch_size, parameters, shuffle = True):
4         """
5         Parameters
6         -----
7         x_set : np.ndarray
8             Input data.
9         y_set : list of np.ndarray or np.ndarray
10             List of output data in case of multiple outputs or single output data in
11             ↪ case of single output.
12         batch_size : int
13             Number of samples per gradient update.
14         parameters : dict

```

```

14         Parameters to be used for data augmentation by the internal
           ↳ ImageDataGenerator instance.
15         For more details on the parameters refer to the documentation of
           ↳ ImageDataGenerator.
16
17     Notes
18     -----
19     In this class, we assume that 'y_set' is a list of numpy arrays for multiple
           ↳ outputs or a single numpy
20     array for a single output. If you are providing a different format for '
           ↳ y_set', you might need to adjust
21     the '__getitem__' method accordingly.
22     """
23
24     def __len__(self):
25         """
26         Get the number of batches per epoch.
27
28         Returns
29         -----
30         int
31             The number of batches per epoch.
32         """
33
34     def __getitem__(self, idx):
35         """
36         Generates a batch of augmented data.
37
38         This method is called for every mini-batch and it should return a complete
           ↳ batch each time.
39         It allows the generator to give the model a type of 'view' on the dataset,
           ↳ returning a batch of
40         inputs (and targets) at each call. In this implementation, the method also
           ↳ performs data augmentation
41         using the ImageDataGenerator instance.
42
43         Parameters
44         -----
45         idx : int
46             Index of the mini-batch. It ranges between 0 and the total number of
           ↳ batches.
47
48         Returns
49         -----
50         tuple
51             A tuple of two elements. The first element is a numpy array of input data
           ↳ after augmentation.
52             If there are multiple outputs, the second element is a list of numpy
           ↳ arrays corresponding to each
53             output. If there's a single output, the second element is a numpy array
           ↳ for that output.
54
55

```



```

56     Notes
57     -----
58     For multi-output models, the second element of the tuple is a list of numpy
59         ↪ arrays. The order of the
60     numpy arrays in the list is the same as the order of the outputs in the
61         ↪ model.
62
63     In the case of single-output models, the second element of the tuple is a
64         ↪ single numpy array
65     corresponding to the output of the model.
66     """
67
68     def on_epoch_end(self):
69         """
70         Shuffle the data at the end of every epoch.
71
72         Notes
73         -----
74         The method is called at the end of every epoch and is used to shuffle the
75             ↪ input and output data to
76         ensure the model doesn't learn any unintended sequential patterns from the
77             ↪ data.
78         """
79
80     def save(self, filename):
81         """
82         Save the current state of the data generator to a file.
83
84         Parameters
85         -----
86         filename : str
87             The name of the file where the state of the data generator will be saved.
88         """
89
90     @classmethod
91     def load(cls, filename):
92         """
93         Load the state of a data generator from a file.
94
95         Parameters
96         -----
97         filename : str
98             The name of the file from which the state of the data generator will be
99             ↪ loaded.
100
101         Returns
102         -----
103         MultiOutputDataGenerator
104             A new instance of MultiOutputDataGenerator with its state loaded from the
105             ↪ file.
106         """

```

custom_losses.py

This script contains three classes that define custom loss functions: the focal loss and weighted versions of the mean squared error and categorical cross_entropy. The standard Keras losses have been adapted to work in the case of multi-output set ups. Weights are applied ‘manually’ instead than with the standard Keras interface (via the `model.fit()` method).

FocalLoss

Implements the Focal Loss function, which is particularly useful for a classification task when there is an extreme imbalance between the classes. To use this loss, you have to pass the string ‘focal_loss’ to the `loss_function` parameter in the `config.yaml` file.

```
1     def __init__(self, alpha=0.2, gamma=1.5, alpha_threshold=0.05, class_weight_dict
    ↪ =None, **kwargs):
2         """
3         Parameters
4         -----
5         alpha : float, optional
6             The scaling factor for the loss. Default is 0.25.
7
8         gamma : float, optional
9             The focusing parameter. Default is 2.0.
10
11        alpha_threshold : float, optional
12            The threshold for the alpha value. Default is 0.05.
13
14        class_weight_dict : dict, optional
15            A dictionary that maps each class to a custom weight. Default is None.
16
17        """
18
19    def focal_loss(self, y_true, y_pred):
20        """
21        Implements the logic of the Focal Loss function.
22
23        Parameters
24        -----
25        y_true : Tensor
26            The true labels.
27
28        y_pred : Tensor
29            The predicted labels.
30
31        Returns
32        -----
33        focal_loss : Tensor
34            The computed Focal Loss.
35
36        Note
37        ----
38        This function is not intended to be used directly. Instead, it's used in the
    ↪ 'call' method.
39        """
40
```

```

41 def call(self, y_true, y_pred):
42     """
43     Calls the Focal Loss function.
44
45     Parameters
46     -----
47     y_true : Tensor
48         The true labels.
49
50     y_pred : Tensor
51         The predicted labels.
52
53     Returns
54     -----
55     focal_loss : Tensor
56         The computed Focal Loss.
57     """
58
59 def get_config(self):
60     """
61     Gets the configuration of the FocalLoss class.
62
63     Returns
64     -----
65     config : dict
66         A dictionary containing the configuration of the FocalLoss class.
67     """
68
69 @classmethod
70 def from_config(cls, config):
71     """
72     Creates a FocalLoss class from its configuration.
73
74     Parameters
75     -----
76     config : dict
77         A dictionary containing the configuration of the FocalLoss class.
78
79     Returns
80     -----
81     FocalLoss
82         A new FocalLoss class.
83
84     Note
85     ----
86     This is a class method that returns a new FocalLoss class given its
87         ↪ configuration.
88     """

```

WeightedMeanSquaredError

Implements the Weighted Mean Squared Error (WMSE) loss function. This class allows to set custom weights for each sample which is particularly useful when the samples are imbalanced. To use this loss, you need to pass the string 'weighted_mse' to the `loss_function` in the `config.yaml` file.

```
1  def __init__(self, class_weights, **kwargs):
2      """
3      Initializes the WeightedMeanSquaredErrorLoss class.
4
5      Parameters
6      -----
7      class_weights : dict
8          A dictionary mapping each output to its weight.
9      kwargs : dict
10         Additional keyword arguments.
11     """
12
13     def call(self, y_true_list, y_pred_list):
14         """
15         Computes the weighted mean squared error loss for each output.
16
17         Parameters
18         -----
19         y_true_list : list of Tensors
20             The true labels for each output.
21         y_pred_list : list of Tensors
22             The predicted labels for each output.
23
24         Returns
25         -----
26         loss_list : list of Tensors
27             The computed weighted mean squared error loss for each output.
28         """
29
30     def get_config(self):
31         """
32         Gets the configuration of the WeightedMeanSquaredErrorLoss class.
33
34         Returns
35         -----
36         config : dict
37             A dictionary containing the configuration of the
38             ↪ WeightedMeanSquaredErrorLoss class.
39         """
40
41     @classmethod
42     def from_config(cls, config):
43         """
44         Creates a WeightedMeanSquaredErrorLoss class from its configuration.
45
46         Parameters
47         -----
48         config : dict
```

```

48         A dictionary containing the configuration of the
           ↳ WeightedMeanSquaredErrorLoss class.
49
50     Returns
51     -----
52     WeightedMeanSquaredErrorLoss
53         A new WeightedMeanSquaredErrorLoss class instance.
54
55     Note
56     -----
57     This is a class method that returns a new WeightedMeanSquaredErrorLoss class
           ↳ instance
58     given its configuration.
59     """

```

WeightedCategoricalCrossEntropy

Implements the Weighted Categorical Cross Entropy (WCCE) loss function. This class allows to set custom weights for each class, which is particularly useful when dealing with imbalanced classes. To use this loss, you need to pass the string 'weighted_cce' to the `loss_function` parameter in the `config.yaml` file.

```

1     def __init__(self, class_weights, **kwargs):
2         """
3         Parameters
4         -----
5         class_weights : dict
6             A dictionary that maps each class to a custom weight.
7         """
8     def weighted_categorical_crossentropy(self, y_true, y_pred):
9         """
10        Computes the weighted categorical cross entropy loss.
11
12        Parameters
13        -----
14        y_true : Tensor
15            The true labels.
16
17        y_pred : Tensor
18            The predicted labels.
19
20        Returns
21        -----
22        weighted_loss : Tensor
23            The computed weighted categorical cross entropy loss.
24
25        Note
26        -----
27        This function is not intended to be used directly. Instead, it's used in the
           ↳ 'call' method.
28        """
29
30
31

```

```

32 def call(self, y_true, y_pred):
33     """
34     Calls the weighted categorical cross entropy loss function.
35
36     Parameters
37     -----
38     y_true : Tensor
39         The true labels.
40
41     y_pred : Tensor
42         The predicted labels.
43
44     Returns
45     -----
46     weighted_loss : Tensor
47         The computed weighted categorical cross entropy loss.
48     """
49
50     return self.weighted_categorical_crossentropy(y_true, y_pred)
51
52 def get_config(self):
53     """
54     Gets the configuration of the WeightedCategoricalCrossentropy class.
55
56     Returns
57     -----
58     config : dict
59         A dictionary containing the configuration of the
60         ↪ WeightedCategoricalCrossentropy class.
61
62     """
63
64 @classmethod
65 def from_config(cls, config):
66     """
67     Creates a WeightedCategoricalCrossentropy class from its configuration.
68
69     Parameters
70     -----
71     config : dict
72         A dictionary containing the configuration of the
73         ↪ WeightedCategoricalCrossentropy class.
74
75     Returns
76     -----
77     WeightedCategoricalCrossentropy
78         A new WeightedCategoricalCrossentropy class instance.
79
80     Note
81     ----
82     This is a class method that returns a new WeightedCategoricalCrossentropy
83         ↪ class instance
84     given its configuration.
85     """

```

utils.py

This script contains miscellaneous methods to load and preprocess the data.

```
1  def image_loader(folder, indices=None, crop=False, crop_size=None, img_size=(64,
    ↪ 64), normalize = True, grayscale = False):
2      """
3      Loads all the images contained in the specified folder.
4
5      Parameters
6      -----
7      folder : string
8          Path to the folder where the images are stored.
9      indices : array-like, optional
10         Indices of specific images to load. Defaults to None, which loads all images
    ↪ .
11     crop : bool, optional
12         Whether to crop images before resizing. Defaults to False.
13     crop_size : tuple, optional
14         Area to crop from the images (left, upper, right, lower). Defaults to None.
15     img_size : tuple, optional
16         The size to which the original image should be resized. Defaults to (64, 64)
    ↪ .
17     normalize : bool, optional
18         Whether to normalize pixel values to be between 0 and 1. Defaults to True.
19     grayscale : bool, optional
20         Whether to convert images to grayscale. Defaults to False.
21
22     Returns
23     -----
24     images : numpy.ndarray
25         An array with the images arrays. Shape is (n_images, img_size[0], img_size
    ↪ [1], n_channels),
26         where n_channels is 1 for grayscale and 3 for RGB.
27
28     Raises
29     -----
30     ValueError
31         If crop is True but no crop_size is specified.
32     """
33
34  def labels_loader(folder, task = [1, 2], min=0.8, one_hot_enc = True):
35      """
36      Loads labels from the specified file and performs optional transformations.
37
38      Parameters
39      -----
40      folder : string
41          Path to the file where the labels are stored.
42      task : list, optional
43          Selects the relevant labels according to the Task. Default value [1, 2]
    ↪ gives the labels dataset.
44      min : float, optional
```

```

45         The minimum value to keep a label. Rows with all label values below this
         ↪ threshold are dropped. Defaults to 0.8.
46 one_hot_enc : bool, optional
47     Whether to perform one-hot encoding on the labels. Defaults to True.
48
49 Returns
50 -----
51 final_df : pandas.DataFrame
52     A DataFrame with the labels arrays.
53
54 Notes
55 -----
56 The function assumes that the labels are stored in a CSV file with a specific
57 ↪ structure. Make sure that the
58 task parameter matches the structure of your file.
59 """
60 def data_loader(images_folder, labels_path, task, min=0.8, one_hot_labels = False,
61 ↪ crop=False, crop_size=None, img_size=(64, 64), normalize=True, grayscale = 0,
62 ↪ training_size = 0.8, test_size = None, random_seed=48):
63     """
64     Load and split the data into training and validation (and optionally test) sets.
65
66 Parameters
67 -----
68 images_folder : str
69     Path to the folder where the images are stored.
70 labels_path : str
71     Path to the file where the labels are stored.
72 task : int
73     Task to work on.
74 min : float, optional
75     The minimum value to keep a label. Rows with all label values below this
76     ↪ threshold are dropped. Defaults to 0.8.
77 one_hot_labels : bool, optional
78     Whether to perform one-hot encoding on the labels. Defaults to False.
79 crop : bool, optional
80     Whether to crop images before resizing. Defaults to False.
81 crop_size : tuple, optional
82     Area to crop from the images (left, upper, right, lower). Defaults to None.
83 img_size : tuple, optional
84     The size to which the original image should be resized. Defaults to (64, 64)
85     ↪ .
86 normalize : bool, optional
87     Whether to normalize pixel values to be between 0 and 1. Defaults to True.
88 grayscale : int, optional
89     Whether to convert images to grayscale. 1 means grayscale, 0 means RGB.
90     ↪ Defaults to 0 (RGB).
91 training_size : float, optional
92     Proportion of the dataset to include in the training split. Default is 0.8.
93 test_size : float, optional
94     If not None, proportion of the training set to include in the test split.
95     ↪ Default is None.

```



```

90     random_seed : int, optional
91         Seed used by the random number generator for reproducibility. Default is 48.
92
93     Returns
94     -----
95     tuple
96         Contains the training data (X_train, y_train), validation data (X_val, y_val
97         ↪ ), and if test_size is not None,
98         test data (X_test, y_test) too.
99
100     Notes
101     -----
102     The function assumes that the labels are stored in a CSV file with a specific
103     ↪ structure.
104     Make sure that the task parameter matches the structure of your file.
105     """
106
107 def image_plotter(images, names, ncols, plot_size=(12,4), spacing=(0.025, 0.025),
108     ↪ title=None, folder_path=None):
109     """
110     Plot and optionally save an array of images in a grid layout.
111
112     Parameters
113     -----
114     images : array
115         The array that contains the images to be plotted.
116     names : list
117         List of image names/titles to be displayed above each image.
118     ncols : int
119         The number of columns in the plot grid.
120     plot_size : tuple, optional
121         The overall size of the entire plot. Defaults to (32,20).
122     spacing : tuple, optional
123         The spacing between images in the plot, defined as (vertical space,
124         ↪ horizontal space). Defaults to (0.025, 0.025).
125     folder_path : str, optional
126         If provided, the plot will be saved as a PDF file in the specified folder.
127         ↪ The folder is created if it does not exist. Default is None.
128
129     Returns
130     -----
131     None
132         This function does not return a value. It shows the plot in the output.
133
134     Notes
135     -----
136     The images are plotted in a grid layout with a specified number of columns (
137     ↪ ncols). The number of rows is determined
138     based on the total number of images and ncols. Each image is displayed with its
139     ↪ respective title from 'names'.
140     """

```

```

136 def load_file(file):
137     """
138     Loads a YAML configuration file.
139
140     Parameters
141     -----
142     file : str
143         The path to the YAML configuration file to load.
144
145     Returns
146     -----
147     dict
148         A dictionary containing the configuration options loaded from the file.
149
150     Raises
151     -----
152     FileNotFoundError
153         If the specified file does not exist.
154
155     Notes
156     -----
157     The function raises an exception if the file does not exist. The function
158     ↪ assumes the file content is in the YAML format.
159     """
160
161 def set_paths(paths):
162     """
163     Sets the paths for loading data and saving outputs (models and plots).
164
165     Parameters
166     -----
167     paths : dict
168         A dictionary containing the paths. It should include the following keys:
169         'images_path' - Path to the directory where the image data is stored.
170         'labels_path' - Path to the file where the labels are stored.
171         'models_path' - Path to the directory where models should be saved.
172         'plots_path' - Path to the directory where plots should be saved.
173
174     Returns
175     -----
176     tuple
177         A tuple containing four strings: the path to the images, the path to the
178         ↪ labels, the path to the models directory,
179         and the path to the plots directory.
180
181     Notes
182     -----
183     The function creates the directories for saving models and plots if they do not
184     ↪ already exist.
185     """

```

```

186 def save_config(config_file):
187     """
188     Prints the configuration parameters and saves them into a log file.
189
190     Parameters
191     -----
192     config_file : str
193         Path to the YAML configuration file.
194
195     Notes
196     -----
197     This function captures the current state of the standard output (stdout),
198         ↳ redirects it to a string IO object,
199     and restores it after the operation. The captured output (configuration
200         ↳ parameters) is written into a text file.
201
202     The 'sort_keys' parameter in the yaml.dump() method is set to False. This means
203         ↳ the order of keys in the original
204     YAML file is preserved in the output, rather than being sorted in default Python
205         ↳ 's lexicographical order.
206     The function creates a directory named 'logs' if it does not already exist to
207         ↳ store the log files.
208
209     """
210
211 def save_hist(history, models_path, name):
212     """
213     Saves the history of a model's training into a pickle file.
214
215     Parameters
216     -----
217     history : History object
218         The history object generated by the training process of a model.
219     models_path : str
220         The path to the directory where the history file will be saved.
221     name : str
222         The name to be given to the saved history file.
223
224     Notes
225     -----
226     The history object typically includes useful data like loss and accuracy metrics
227         ↳ recorded at each epoch
228     during the training process. It is saved using the pickle module, which allows
229         ↳ for serialization and de-serialization
230     of Python object structures.
231
232     """
233
234 def one_hot_encoder(df):
235     """
236     Converts a DataFrame to one-hot encoding.
237
238     This function takes a DataFrame and converts it to one-hot encoded form. This
239         ↳ means that each row is a vector

```

```

231     where only the element corresponding to the category of that sample is set to 1
        ↳ and all other elements are 0.
232
233     Parameters
234     -----
235     df : DataFrame
236         The DataFrame to be converted to one-hot encoding.
237         It is expected that the DataFrame is numeric and that the column with the
        ↳ highest value in each row
238         corresponds to the category of that sample.
239
240     Returns
241     -----
242     one_hot_df : DataFrame
243         A DataFrame that is the one-hot encoded version of the input df.
244         It has the same shape as the input df, with values replaced by 0s and 1s.
245     """
246
247     def hierarchy_one_hot(df, min = 0.5):
248         """
249         Converts a DataFrame to one-hot encoding based on the maximum value in each
        ↳ hierarchical group.
250
251         Parameters
252         -----
253         df : pandas.DataFrame
254             The DataFrame to be converted to one-hot encoding.
255             It should contain columns for each class and sub-class.
256         min : float, optional
257             The threshold value used for the one-hot encoding of Class1. Defaults to
        ↳ 0.5.
258
259         Returns
260         -----
261         df_one_hot : pandas.DataFrame
262             A DataFrame that is the one-hot encoded version of the input df based on the
        ↳ max value in each group.
263         """
264
265     def one_hot_encoder_array(array):
266         """
267         Converts an array to one-hot encoding.
268
269         This function takes a 2D numpy array and converts it to one-hot encoded form.
        ↳ This means that each row is a vector
270         where only the element corresponding to the category of that sample is set to 1
        ↳ and all other elements are 0.
271
272         Parameters
273         -----
274         array : ndarray
275             The 2D numpy array to be converted to one-hot encoding.

```

```

276         It is expected that the array is numeric and that the column with the
           ↪ highest value in each row
277         corresponds to the category of that sample.
278
279     Returns
280     -----
281     new_array : ndarray
282         A 2D numpy array that is the one-hot encoded version of the input array.
283         It has the same shape as the input array, with values replaced by 0s and 1s.
284     """
285
286 def data_inspector(labels):
287     """
288     Inspects the assigned classes based on the predicted label probabilities.
289
290     Parameters
291     -----
292     labels : pandas.DataFrame
293         The predicted label probabilities for each sample. The DataFrame is expected
           ↪ to have the following columns:
294         'Class1.1', 'Class1.2', 'Class1.3', 'Class2.1', 'Class2.2', 'Class7.1', '
           ↪ Class7.2', 'Class7.3'.
295
296     Returns
297     -----
298     None
299
300     Prints
301     -----
302     Counts of Class1 Subclasses:
303     Prints the counts of samples assigned to each Class1 subclass and their
           ↪ corresponding labels.
304
305     Counts of Class2 Subclasses among those assigned to Class1.2:
306     Prints the counts of samples assigned to each Class2 subclass among those
           ↪ assigned to Class1.2 and their corresponding labels.
307
308     Counts of Class7 Subclasses among those assigned to Class1.1:
309     Prints the counts of samples assigned to each Class7 subclass among those
           ↪ assigned to Class1.1 and their corresponding labels.
310     """

```

Conclusion

In summary, this document provides detailed documentation for the code developed in the project. It serves as a reference for understanding the classes, methods, and their usage.