SAPIENZA
UNIVERSITÀ DI ROMA

Homework 2

Image Classification on Small Datasets

Machine Learning course 2021-2022

Recap

- Ensembles of small-scale convolutional networks provide good accuracy in small-data settings (Seminar 1)

- Tuning the hyper-parameters like learning rate and weight decay is a critical factor to ensure the highest performance (Seminar 2)

(Some) open questions

- Do custom/newer architectures work better than popular ResNets?

- Can we do even better choosing hyper-parameters?

- Do heterogeneous ensembles perform better than homogeneous ones?

SAPIENZA
UNIVERSITÀ DI ROMA

Answering these questions with crowd-sourcing

What if each student will train a “small” network to provide an ensemble member?

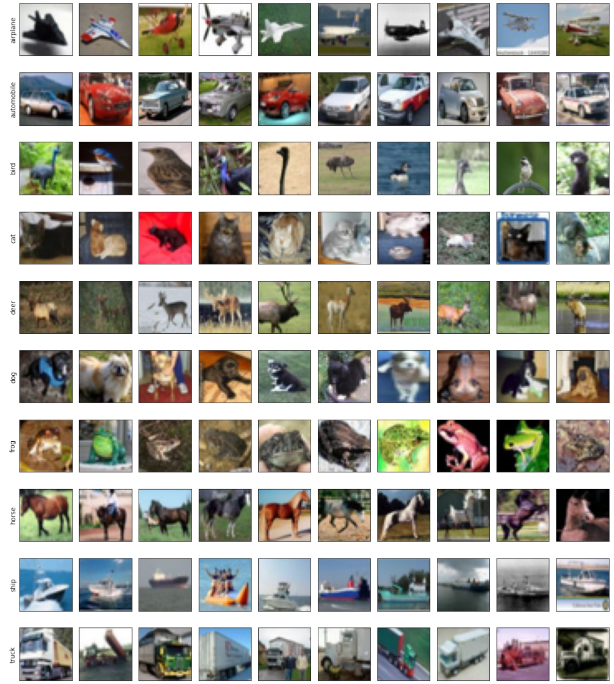
In this way:

- Obtaining a (very!) large and heterogeneous ensemble

- Exploring its own search space is easier for each single member (i.e. hyper-parameters)

Dataset [link]

- The dataset is ciFAIR-10 of the DEIC Benchmark (Seminar 2)
    - 10-class problem
    - 50 training images per class
    - 1000 testing images per class
    - image dimension 32x32x3



- Categories:
    - airplane
    - automobile
    - bird
    - cat
    - deer
    - dog
    - frog
    - horse
    - boat
    - truck

Objectives

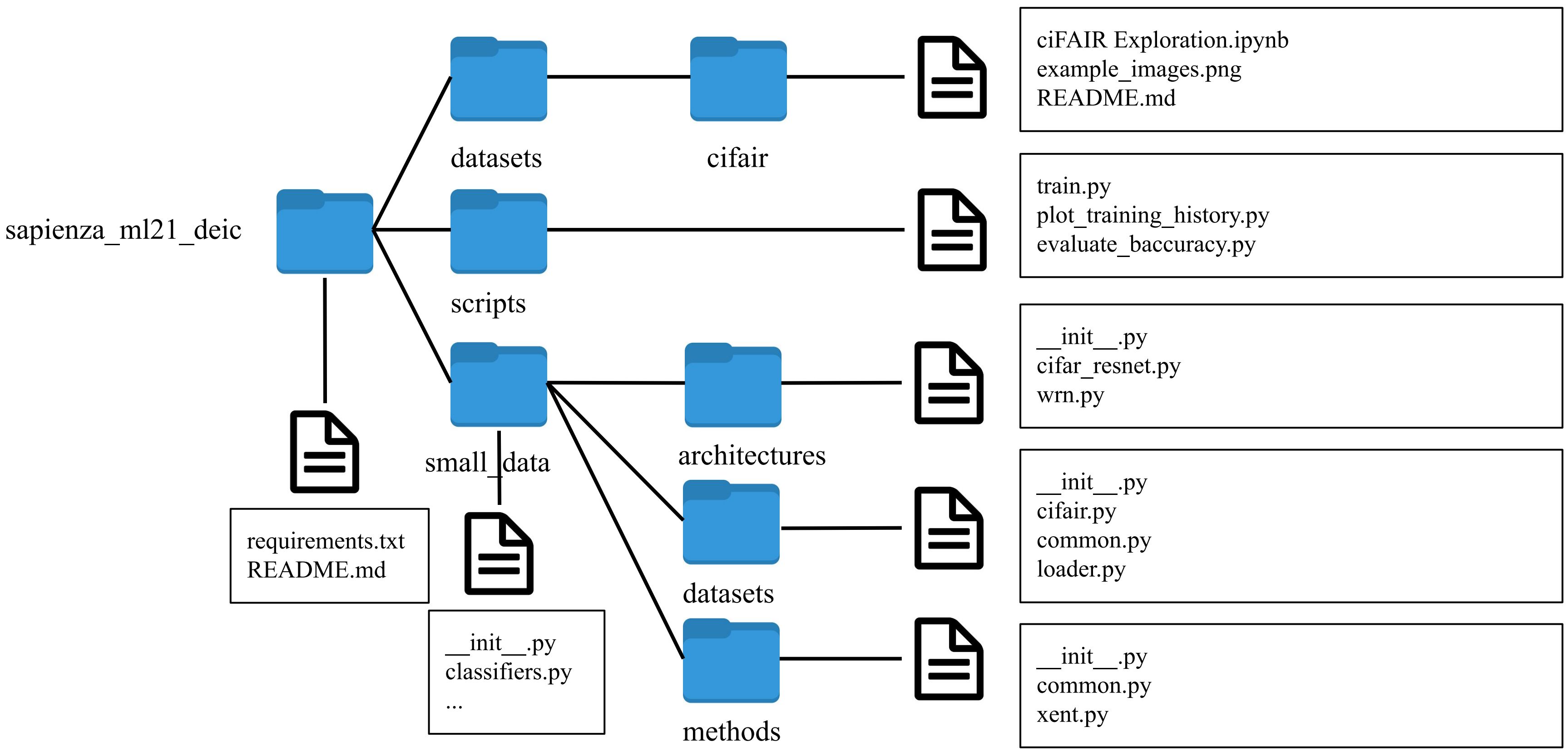
- Train your favorite network architecture:
  - published one (e.g. ResNet, Wide ResNet, EfficientNet…)
  - custom one (note that published ones are a good starting point)

- Analyze results concerning few modifications of chosen architecture:
  - size (e.g., network depth/width)
  - layer configurations (e.g., kernel size, convolution stride)

- Analyze results concerning few modifications of optimization process:
  - optimizer (e.g. SGD, Adam, ...)
  - optimizer parameters (e.g. learning rate, weight decay, momentum)
  - training (e.g. epochs, batch size)

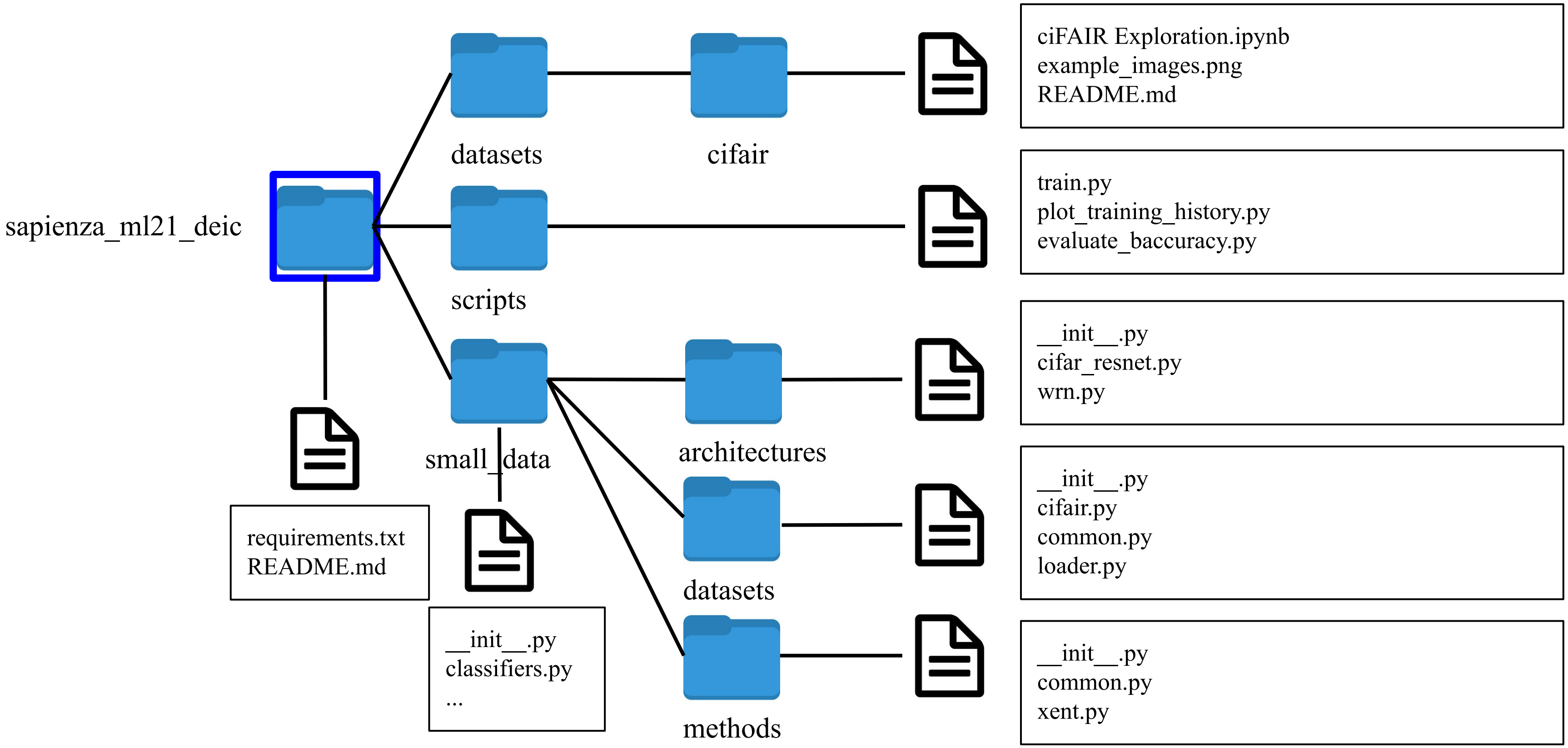
Code Repository [link]

We provide a code repository as starting point for your experiments



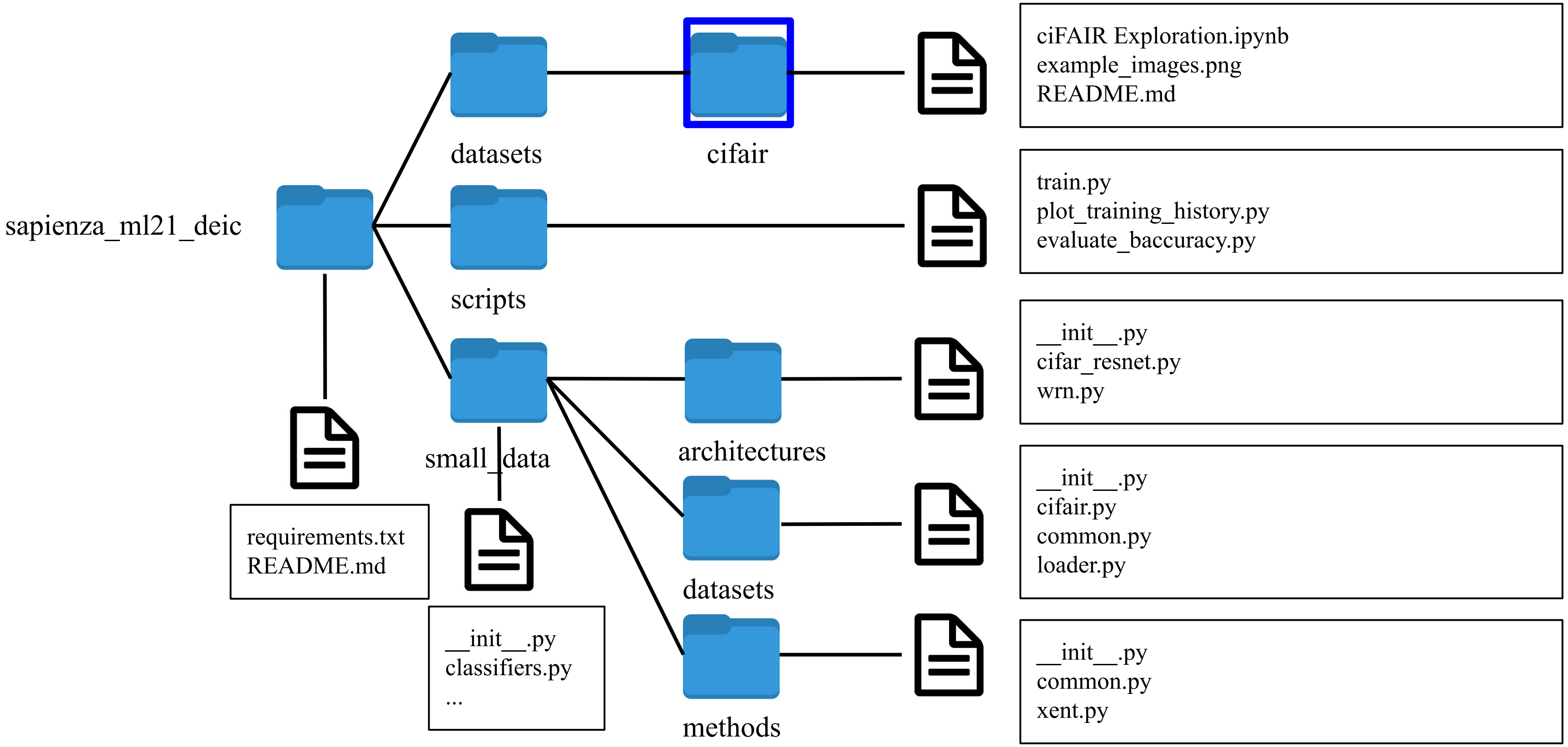
Code Repository

sapienza_ml21_deic is the main directory containing: file with package requirements, readme of the repository and subdirectories.



Code Repository

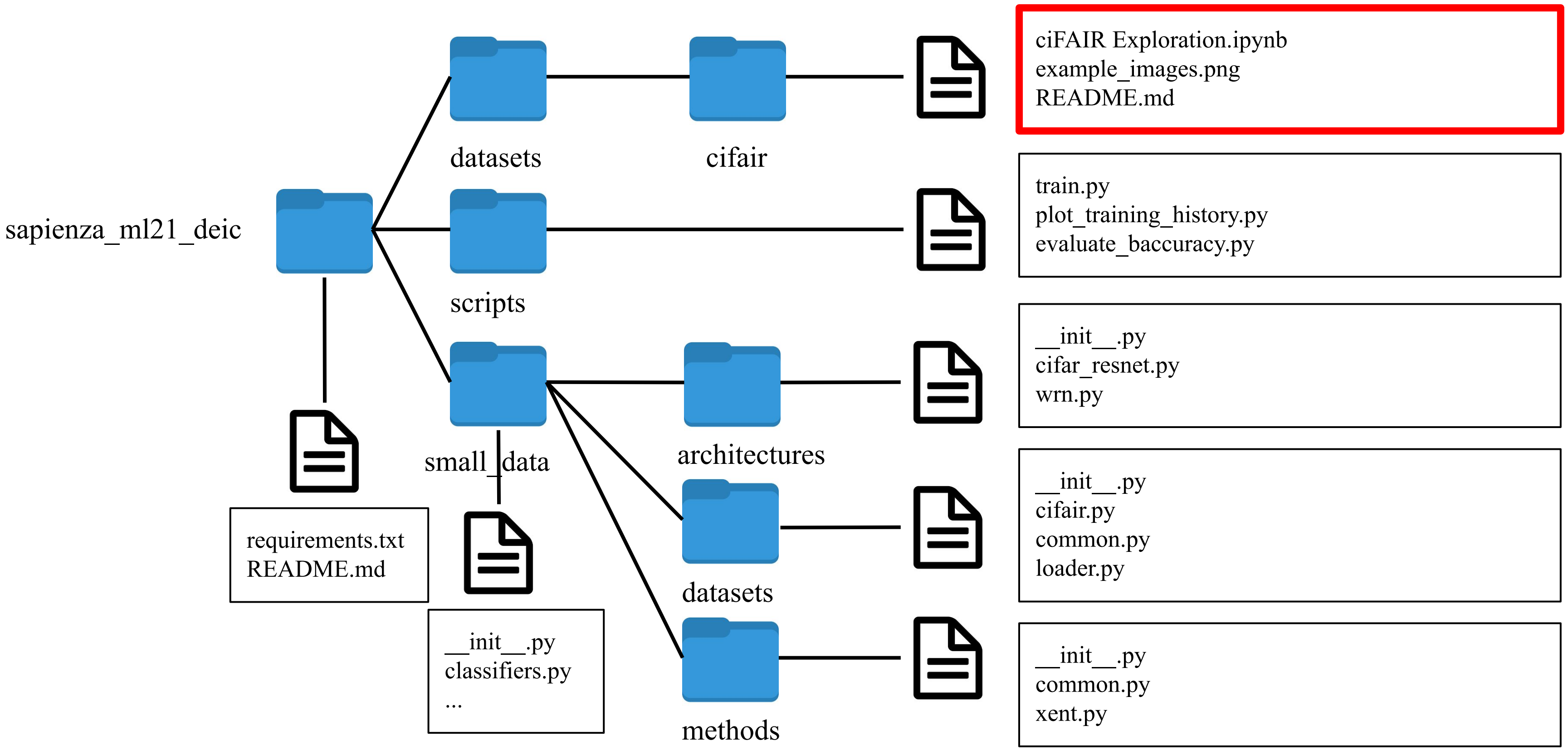
Put the provided dataset repository (ciFAIR-10 shared on Drive) inside this directory.



SAPIENZA
UNIVERSITÀ DI ROMA

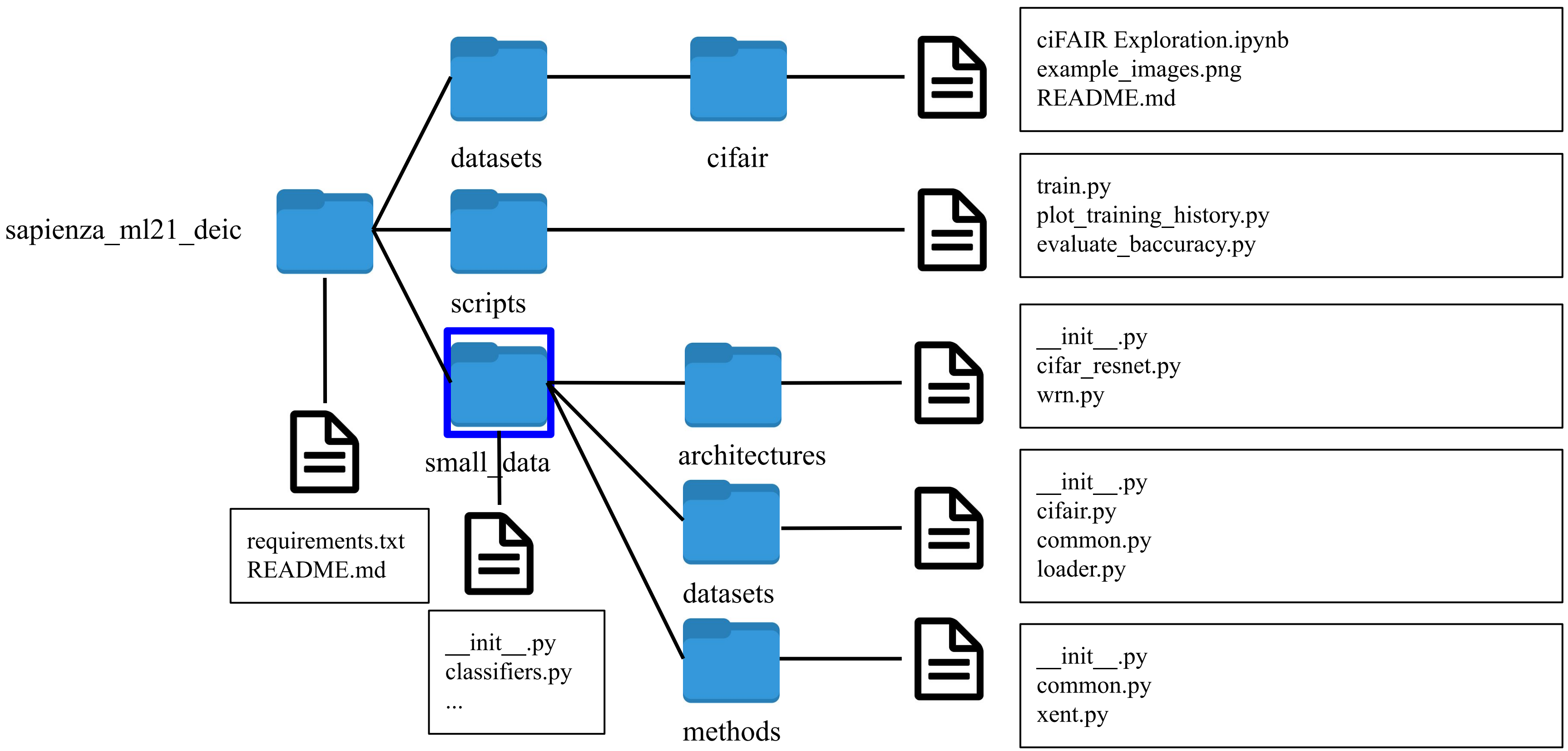
Code Repository

Notebook for dataset visualization and readme file with baseline performance along with terminal command to reproduce the baseline result.



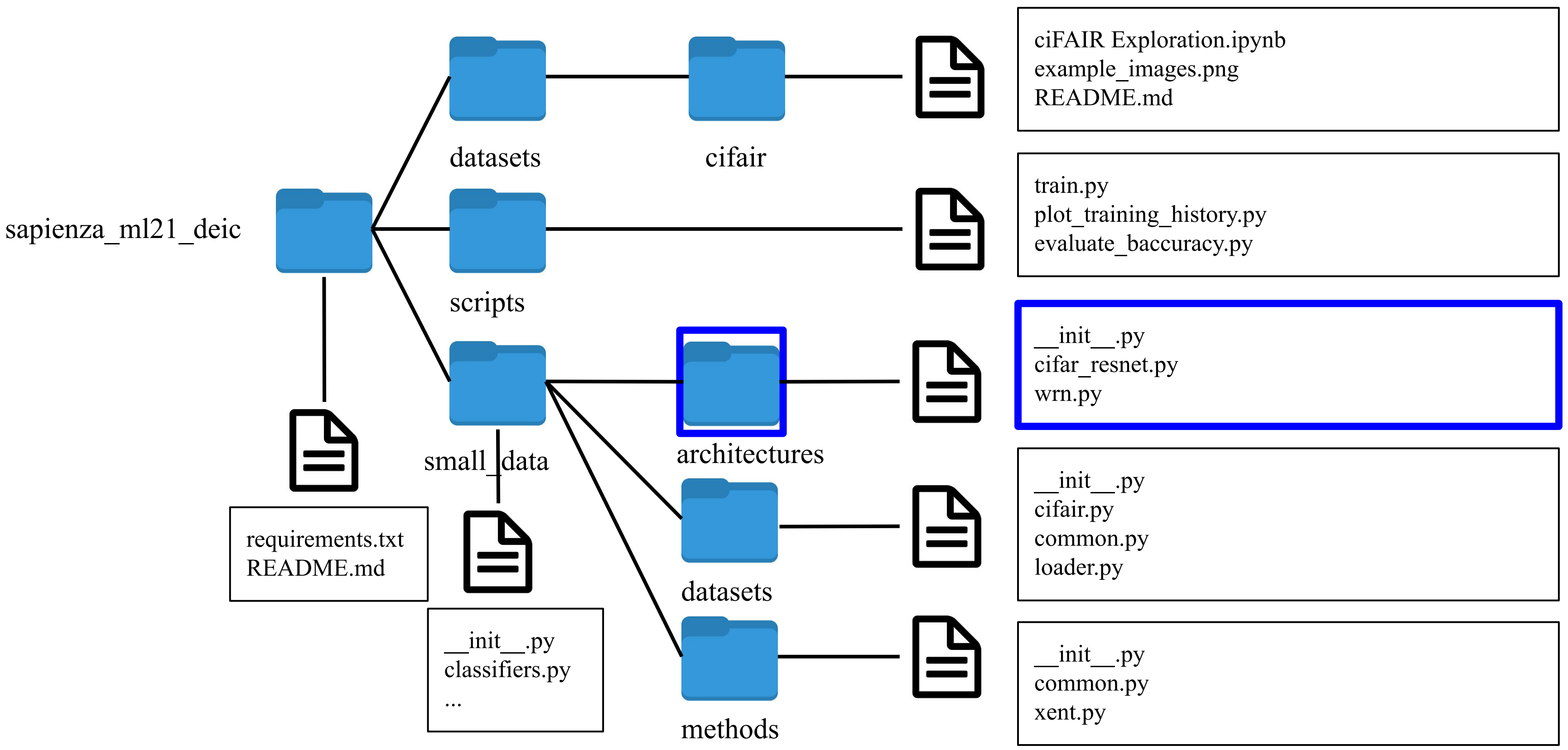
Code Repository

small_data is the directory that you will submit, with all its content.



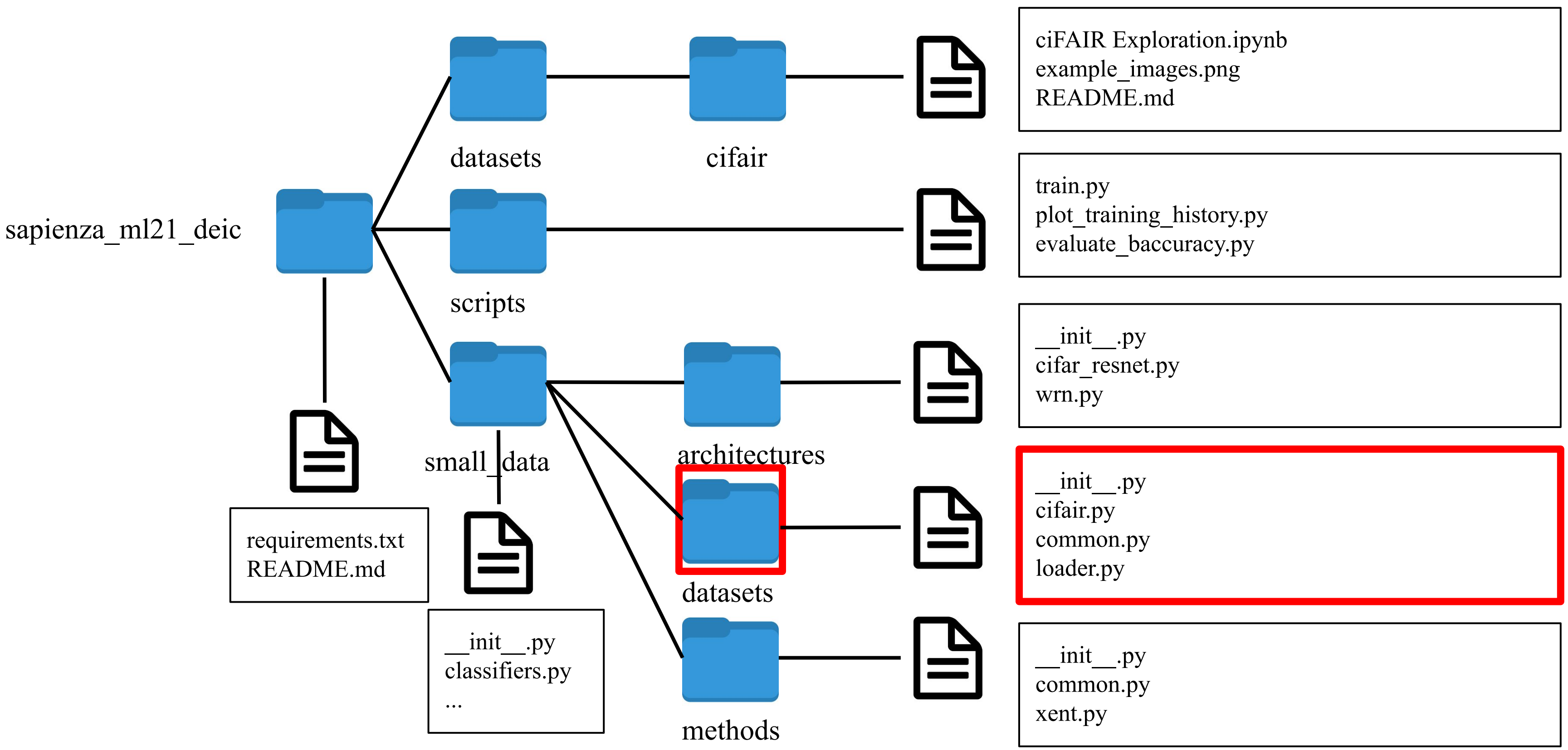
Code Repository

The folder **architectures** is the place to add other architecture files (e.g., densenet.py).



Code Repository

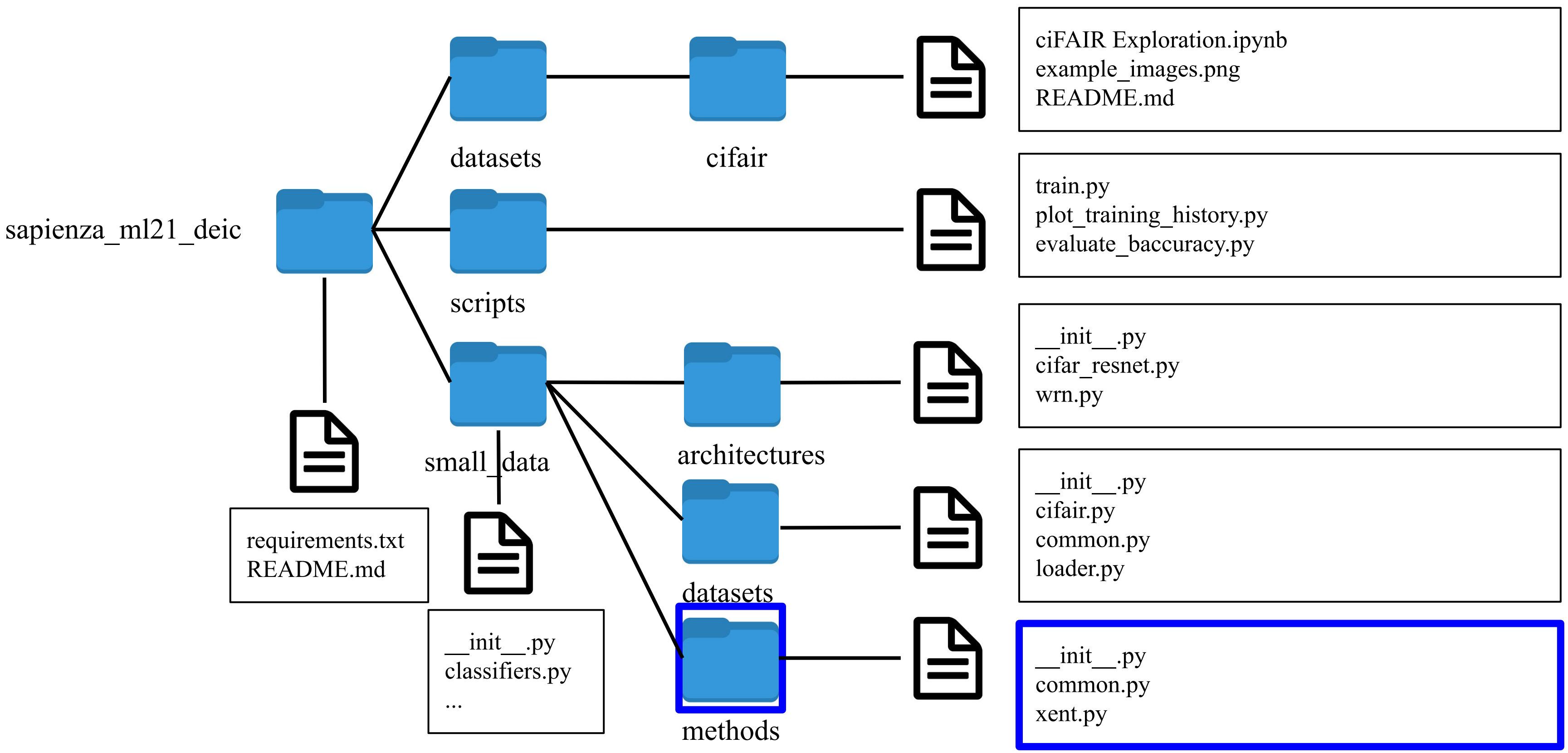
The folder **datasets** contains functions for loading data.



SAPIENZA
UNIVERSITÀ DI ROMA

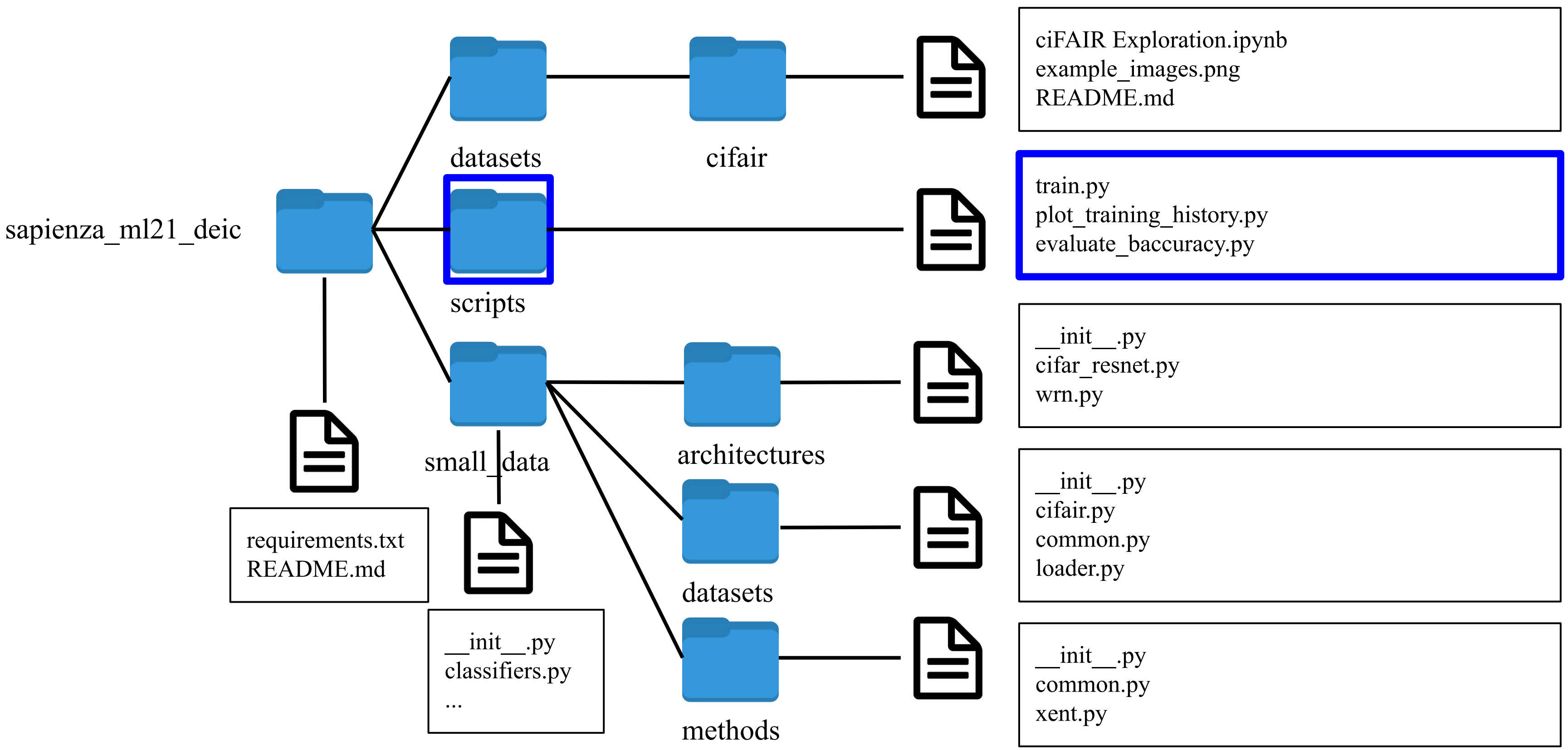
Code Repository

The folder **methods** contains functions for running the training algorithms.



Code Repository

The folder **scripts** contains higher-level scripts to actually run the training or visualize results.



Google Colab Demo

- Setup the code repository on Google Drive with SetupGitRepoOnDrive.ipynb
- Run baseline experiment with ExampleTrain.ipynb

Your Task

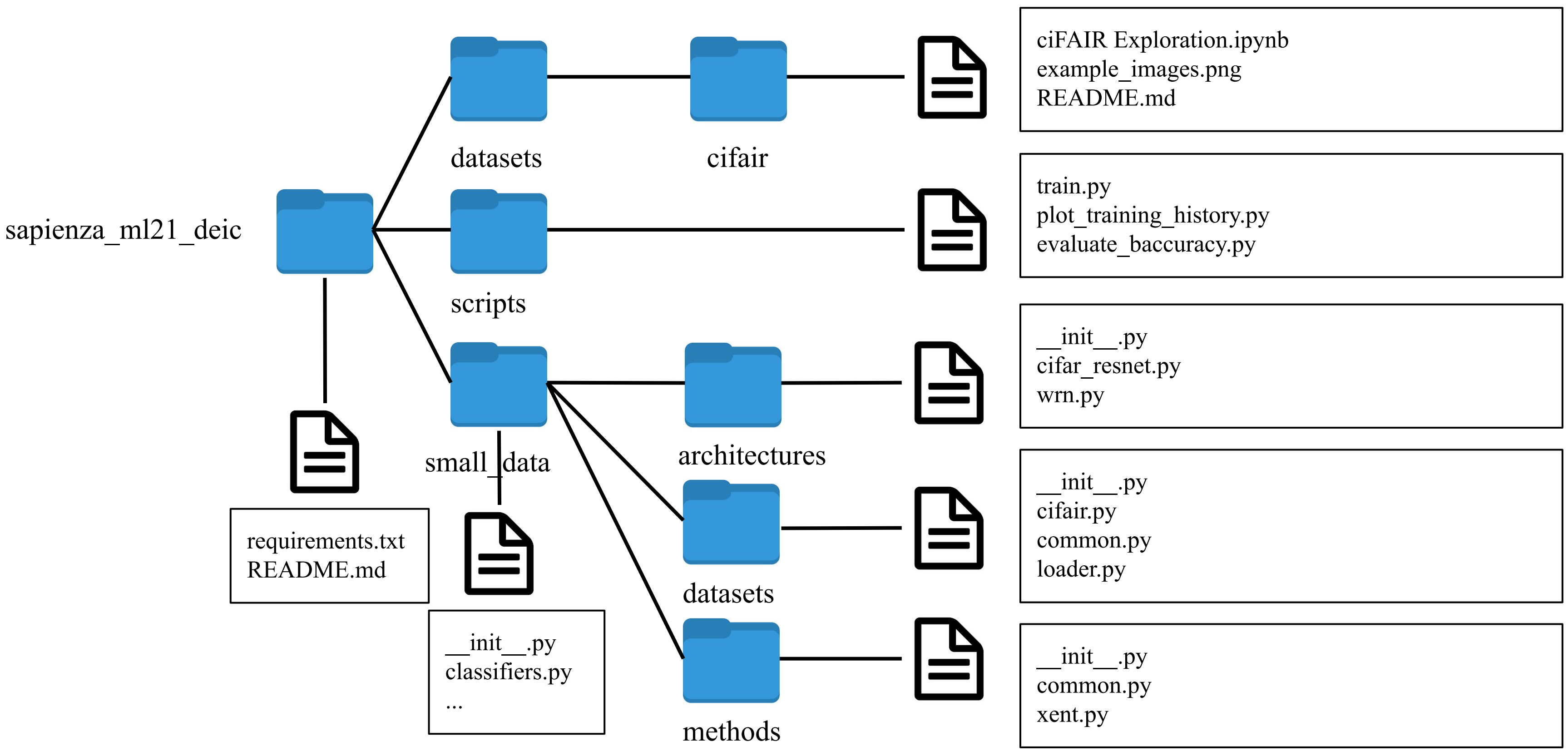
Add something in the code, save the trained model and evaluate it on the test set.

Examples:

- New architecture
- A different optimizer for training
- A choosable hyper-parameter (e.g., momentum parameter of SGD)

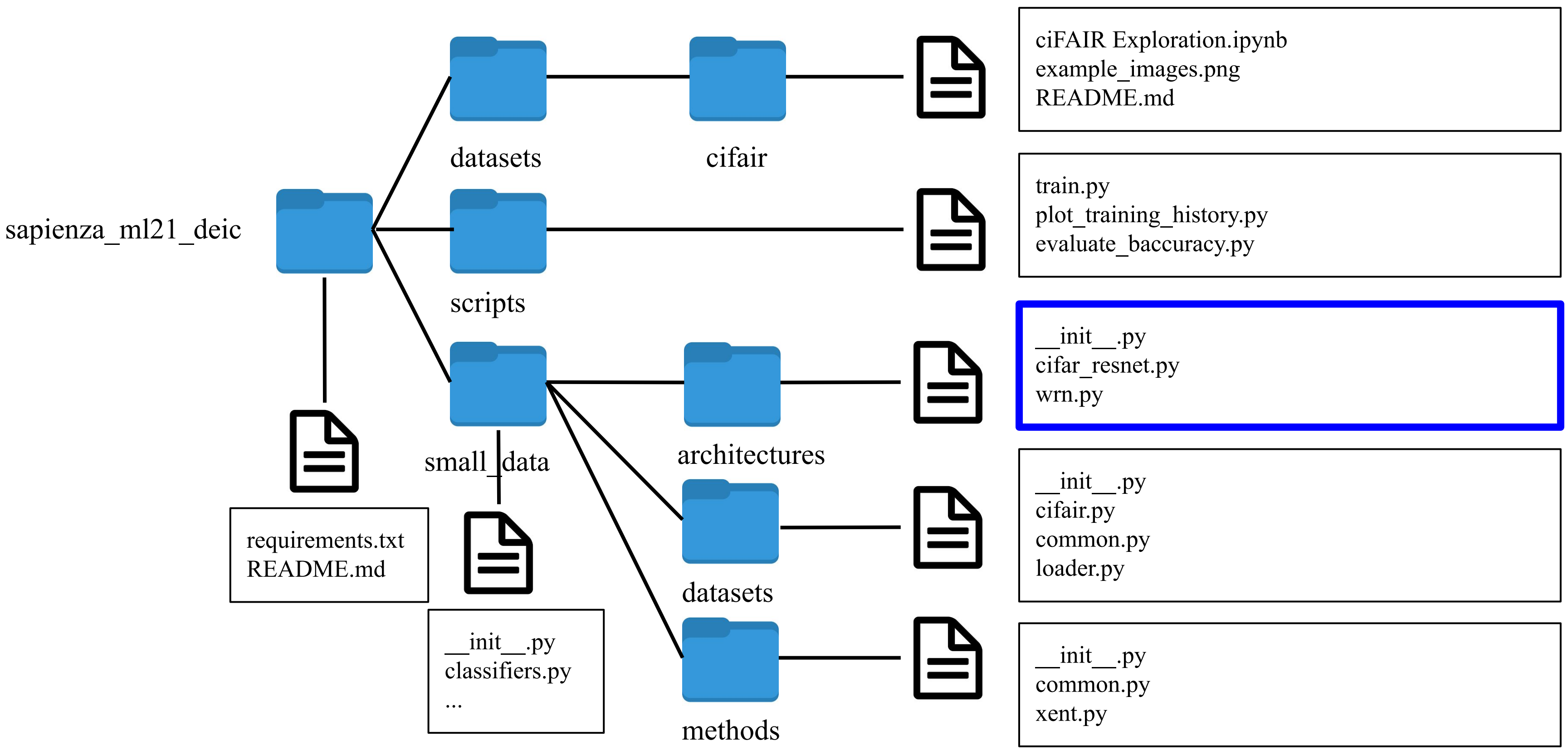
- **At least perform two of these tasks.**
- **Only running the demo would translate in 0 points for your homework.**

Adding a new architecture



Adding a new architecture

You should add a file that define the architecture inside **architectures**.



Adding a new architecture

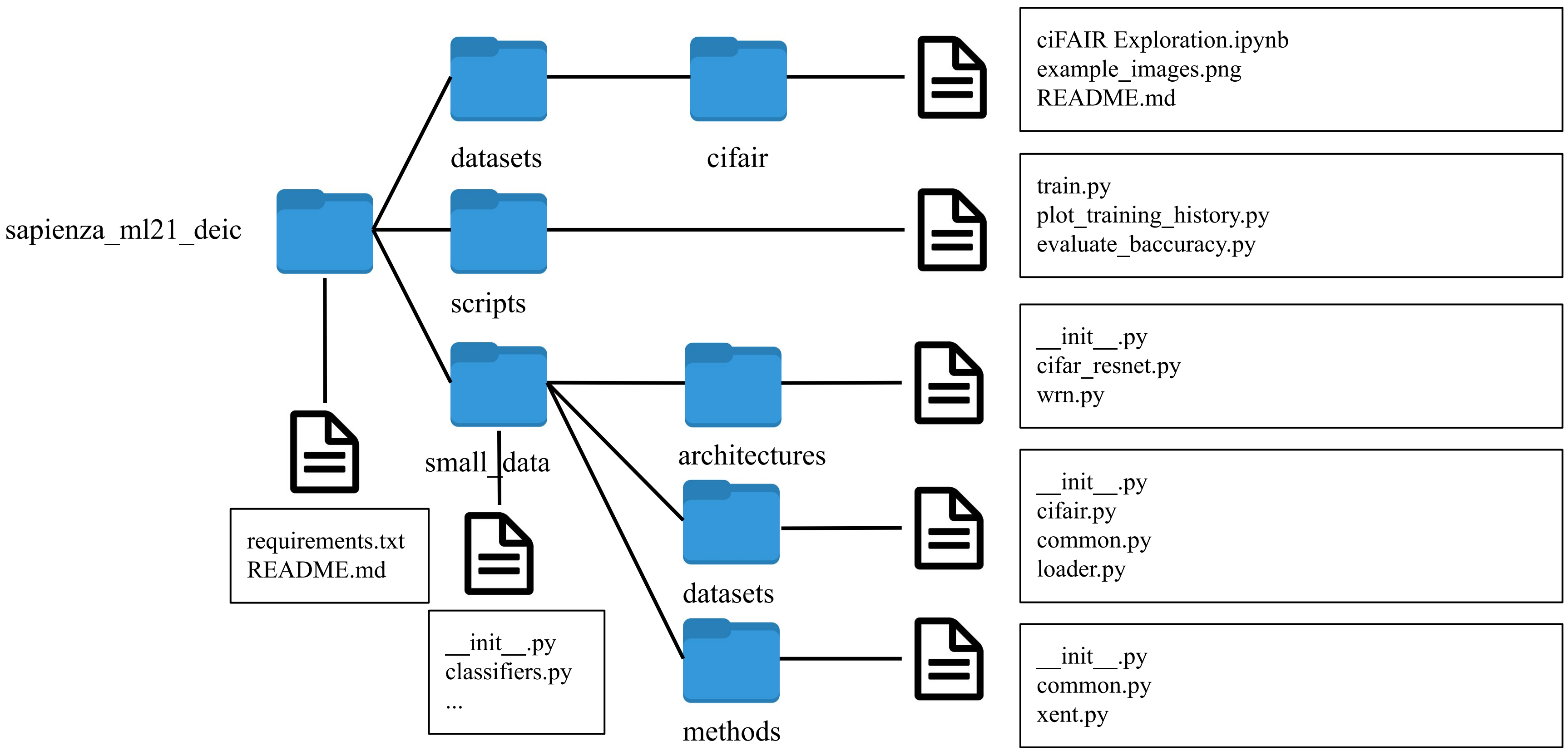
Steps:

- Create a new file inside folder **architectures** (e.g., wrn.py) to contain all the python classes/functions needed by the given architecture.

- The main class must have an `__init__`, `forward`, `get_classifiers`, and `build_classifier` method.

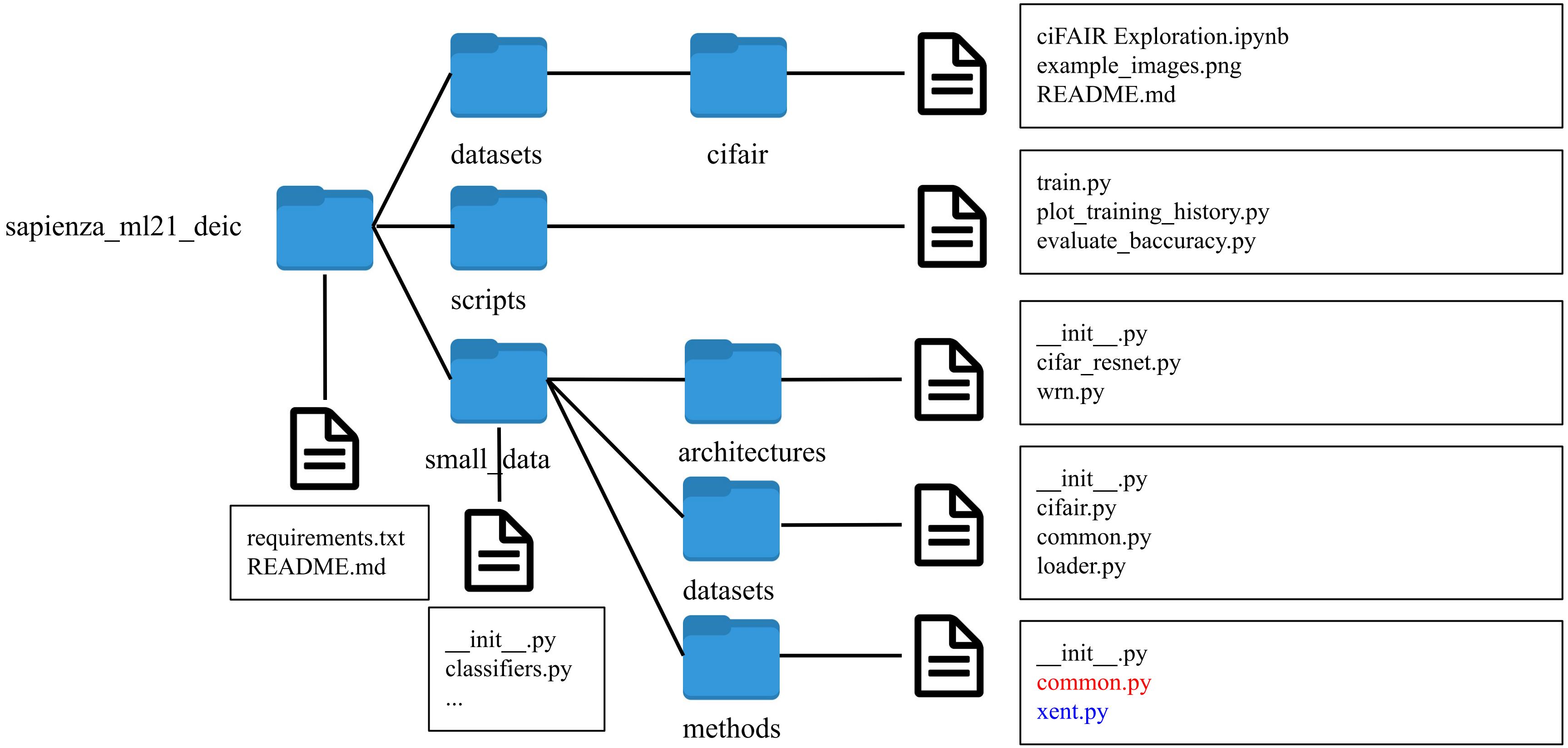
```
class WideResNet(nn.Module):

    def __init__(self, depth, num_classes, input_channels=3, widen_factor=1, ...):
        …

    def forward(self, x):
        …

    @staticmethod
    def get_classifiers():
      return ['wrn-16-8', 'wrn-16-10', ..., 'wrn-28-12']


    @classmethod
    def build_classifier(cls, arch: str, num_classes: int, input_channels: int):

      _, depth, widen_factor = arch.split('-')

      cls_instance = cls(int(depth), num_classes, input_channels=input_channels,
widen_factor=int(widen_factor))

      return cls_instance
```

Using a different optimizer for training



Using a different optimizer for training

You should sub-class the method `get_optimizer` of `LearningMethod` defined in common.py with a different instance inside xent.py.



Using a different optimizer for training

The main training class contains multiple methods that can be sub-classed:

- `LearningMethod` is the general class for a training pipeline of a method defined in common.py.
- An instance of this class is `CrossEntropyClassifier` which is defined in xent.py.
- `CrossEntropyClassifier`, when called will override all methods of `LearningMethod` defined inside its file.
- Any change can be applied by modifying the desired class method (e.g., `get_optimizer`).

Using a different optimizer for training

Original `get_optimizer` function defined inside common.py.

```
def get_optimizer(self, model: nn.Module, max_epochs: int, max_iter: int):

       optimizer = torch.optim.SGD(model.parameters(), lr=self.hparams['lr'], momentum=0.9,
weight_decay=self.hparams['weight_decay'])

       scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=max_iter)

       return optimizer, scheduler
```

Using a different optimizer for training

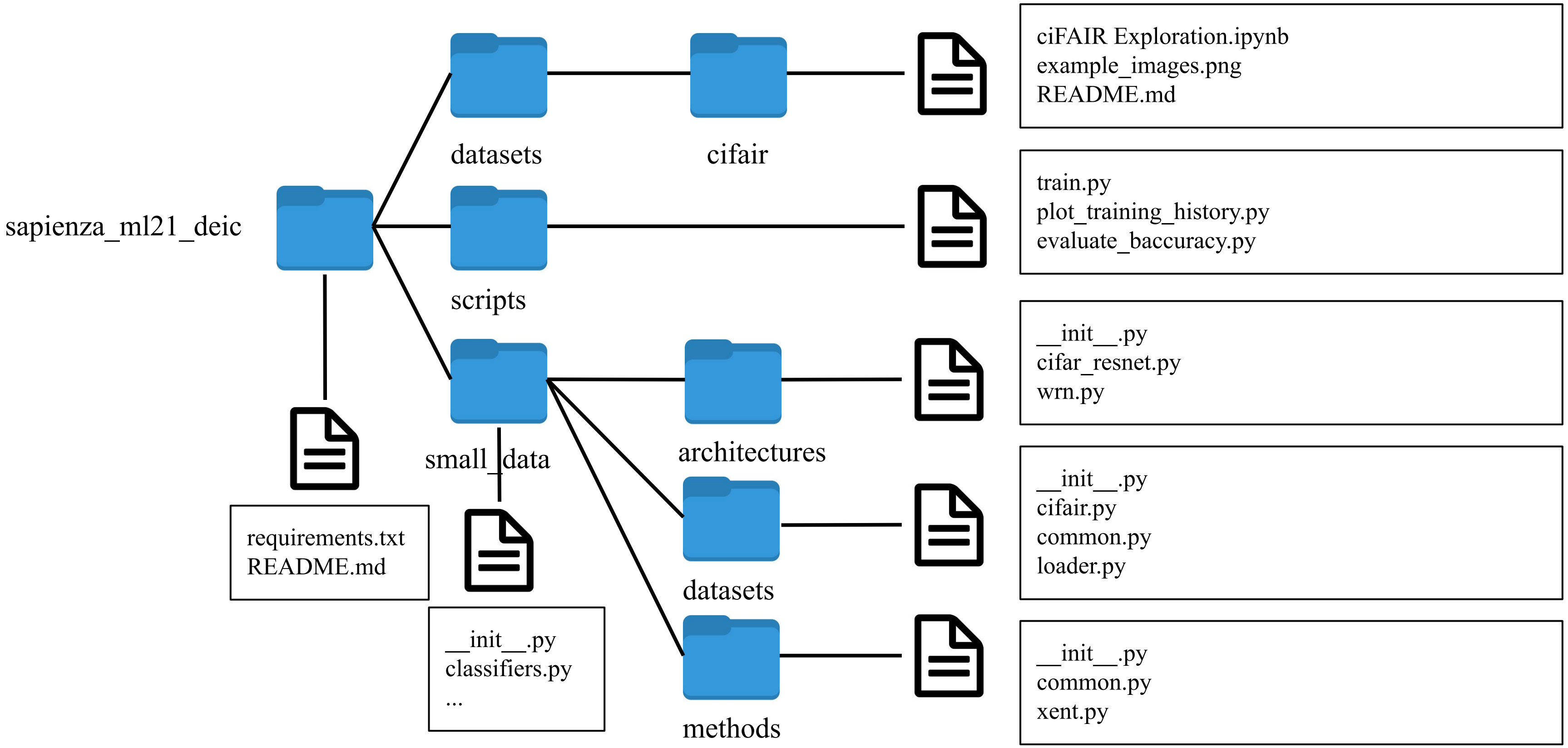
New `get_optimizer` method (with `Adam` instead of `SGD`) of `CrossEntropyClassifier` defined inside xent.py.

```
def get_optimizer(self, model: nn.Module, max_epochs: int, max_iter: int):

       optimizer = torch.optim.Adam(model.parameters(), lr=self.hparams['lr'],
weight_decay=self.hparams['weight_decay'])

       scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=max_iter)

       return optimizer, scheduler
```

Sapienza Università di Roma

Add choosable hyper-parameter

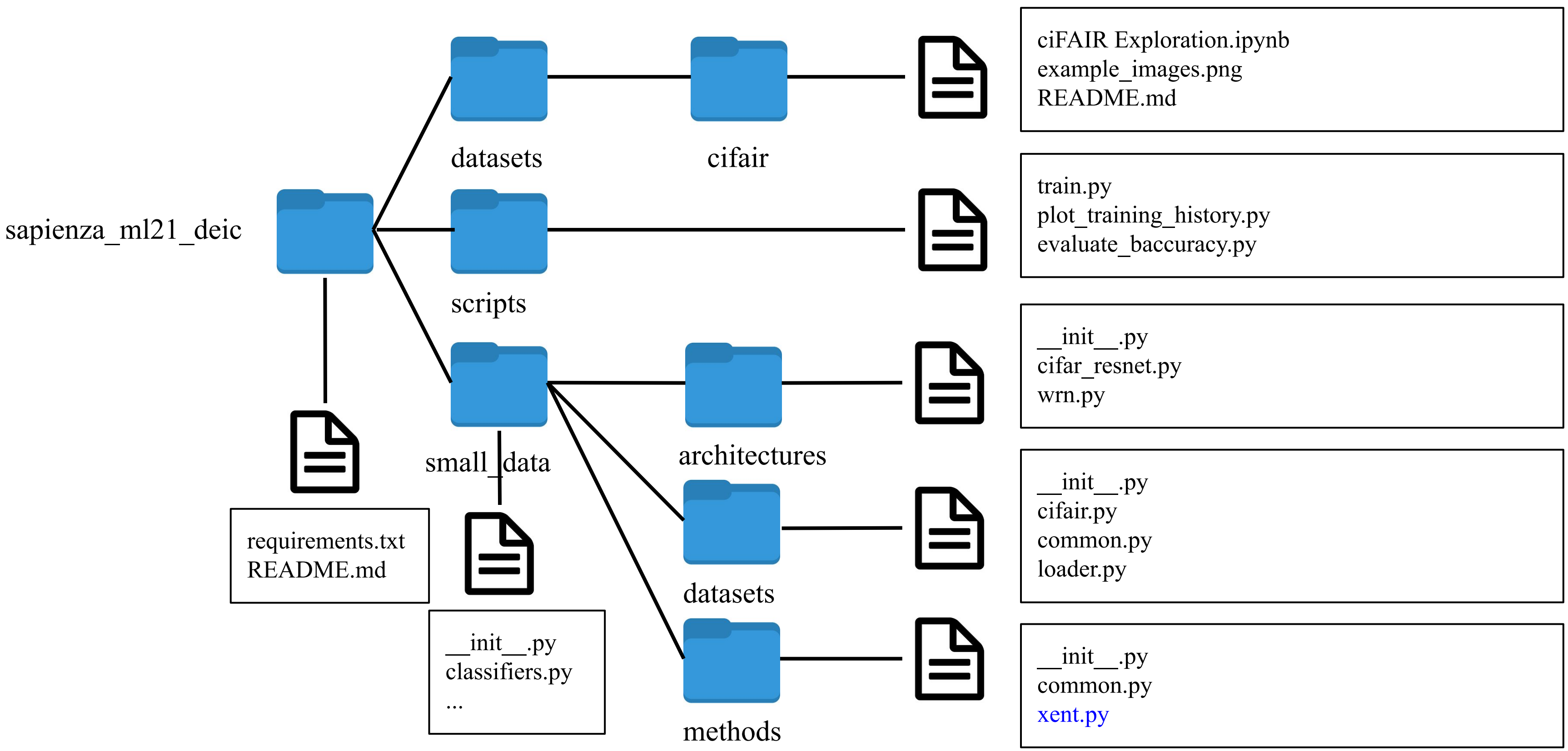
Making the momentum parameter of SGD as choosable hyper-parameter by the user.



SAPIENZA
UNIVERSITÀ DI ROMA

Add choosable hyper-parameter

You should sub-class the static methods `default hparams` and `get_optimizer` of class `CrossEntropyClassifier` in xent.py.



Add choosable hyper-parameter

Sub-class the static `default_hparams` method of `CrossEntropyClassifier` to add `momentum`:

```
@staticmethod
def default_hparams() -> dict:
    return {
        **super(CrossEntropyClassifier, CrossEntropyClassifier).default_hparams(),
        'momentum' : 0.9
    }
```

Update the `get_optimizer` method too:

```
def get_optimizer(self, model: nn.Module, max_epochs: int, max_iter: int):

       optimizer = torch.optim.SGD(model.parameters(), lr=self.hparams['lr'],
momentum=self.hparams['momentum'],     weight_decay=self.hparams['weight_decay'])

       scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=max_iter)

       return optimizer, scheduler
```

SAPIENZA
UNIVERSITÀ DI ROMA

Deep Learning Library

The code is written in PyTorch which is very similar to Tensorflow:

- PyTorch (1.7) documentation [link].

- The modules of `torch.nn` are equivalent to layers in `tf.keras.layers`. Docs [link].

- To use different network architectures I advise looking at the available models at `torchvision.models` [link] and copying, using/modifying their source code (e.g., EfficientNet).

EfficientNet

```
torchvision.models.efficientnet_b0(pretrained: bool = False, progress: bool = True,
**kwargs: Any) → torchvision.models.efficientnet.EfficientNet [SOURCE]
```

```
class EfficientNet(nn.Module):
    def __init__(
            self,
```

Save the trained model

Models in PyTorch are saved similarly to Tensorflow:

- A `torch.nn.Module` has a state dictionary containing all the information concerning the layer weights.

- It is enough to simply call:
  `torch.save(model.state_dict(), “model.pth”)`

- Note that to load back the model, you should first instantiate an instance of the model class and then override the state dictionary.

- Therefore, you should call:
  `model.load_state_dict(torch.load(“model.pth”))`

- Additional documentation is available [link]

SAPIENZA
UNIVERSITÀ DI ROMA

Suggestions

To start from a good baseline take into account these tricks:

- Small batch sizes (e.g., 10 or less) are preferable to improve generalization.
- An high number of epochs (e.g., > 500) allow the network to find better local minima with better generalization.
- Not all lr-wd couples work the same. Choosing a good couple strongly increases the results (Seminar 2) and may change with network width/depth.
- Network width ease the optimization process. With the right hyper-parameters, a wider network (i.e., more conv. filters) may find better solutions than thinner ones.
- Baseline to beat:
  - Wide ResNet-16-8 (wrn-16-8) , epochs = 500, bs = 10, lr = 4.55e-3, wd = 5.29e-3
  - Accuracy on test set: 58.22%
- **Note**: the homework evaluation won't be related to the test set performance obtained (unless for extremely bad results).

Submission

Through Google Classroom:

- **PDF report** with description of the models and results on the test set.
- **ZIP file of small_data folder**, named with your matricola code (e.g, 1771597.zip).
- **Your trained model** saved with your matricola code (e.g, 1771597.pth).

SAPIENZA
UNIVERSITÀ DI ROMA

Expected Outcome

Publish a paper with 100 names!! Or less optimistically:

- Get interesting insights and results about this crowd-sourcing experiment with ensembles on small datasets.

- Not getting the same model from all of you since this would basically ruin the experiment:
  - predictions of average of 100 identical models = predictions of 1 model

- And would give 0 points to your homework!!

- You'll learn important tools and practices to train neural networks.

Summary

To run the code you need a GPU either locally or on the Cloud:

- Use Colab or make sure to have PyTorch locally installed with GPU support
- [For Colab] Use the notebook SetupGitRepoOnDrive.ipynb to clone the repository on your Google Drive.
- [For Colab] Use/take inspiration from ExampleTrain.ipynb to run experiments.
- [For Colab] Note that to change the running code, you should modify the files of the repo that are located in your Google Drive.