

# Siamese\_auxiliary

July 12, 2023

## 1 run load\_data.ipynb BEFORE running this!

```
[25]: import numpy as np
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
s = {
    'problem'          : "regression",
    'approach'         : "few-shot learning",
    'method'           : "non-parametric",
    'algorithm'        : "siamese network",
    'goal'              : "learn a distribution using few samples from it",
    'input'             : "samples from a distribution",
    'input type'        : "vectors",
    'input meaning'     : "spectrum",
    'output'            : "samples from a distribution",
    'output type'       : "one number",
    'output meaning'    : "temperature or pressure, depending on distribution",
    'number of ways'    : 2,
    'number of shots'   : 1,
    'number of folds'   : 8,
    'support-query ratio': 0.8,
    'task size'         : 5,
    'learning rate'     : 1e-4,
    'input dimension'   : 10000,
    'output dimension'  : 1,
    'feature dimension' : 300,
    'epoch'             : 1000,
    'epoch development' : 4,
    'data'              : 'pressure_230516_discrete',
    'cross validation round': 16,
    'cross validation round-development' : 3,
    'batch size'        : 32,
    'best model folder' : 'siamese_best_model/'
}
```

```
[26]: import data_accessor as acc
data_names_list = [
```

```

        'temperature_230509_discrete',
        'pressure_230516_discrete'
    ]
    data_dictionary = acc.setup(data_names_list)

```

```

loading temperature_230509_discrete_____
    input shape (number, dimension): (6000, 10000)
    label shape (number, dimension): (6000, 1)
    there are 16 folds
    4200 for training, 600 for validating, 1200 for testing
loading pressure_230516_discrete_____
    input shape (number, dimension): (5000, 10000)
    label shape (number, dimension): (5000, 1)
    there are 16 folds
    3500 for training, 500 for validating, 1000 for testing

```

```

[27]: # task layout July 5, 2023

# siamese network extract feature space difference
# auxiliary network convert that difference into label difference

# Bing's feedback July 7th 2023
# need to fix:
# 1 convergence graph good for some CV round, bad for others
# 2 wrong use of cross-validation
# 3 try bigger dim, to fix 1
# 4 test on pressure data
# 5 try different architecture
# if layer k has dimension d_k, d_{k+1} should be > d_k / 10

```

```

[28]: import torch.nn as nn
class SiameseNetwork(torch.nn.Module):
    def __init__(self, device, input_dimension, feature_dimension,
        ↪output_dimension):
        """ Input: input, anchor, anchor label
        Output: prediction for input"""
        super().__init__()
        self.input_dimension = input_dimension
        self.hidden_dimension = 400
        self.feature_hidden_dimension = 100
        self.feature_dimension = feature_dimension
        self.output_dimension = output_dimension
        self.device = device
        self.feature_sequential = torch.nn.Sequential(
            torch.nn.Linear(self.input_dimension, self.hidden_dimension),
            nn.ReLU(),
            torch.nn.Linear(self.hidden_dimension, self.hidden_dimension),

```

```

        nn.ReLU(),
        torch.nn.Linear(self.hidden_dimension, self.feature_dimension)
    )
    self.auxiliary_sequential = torch.nn.Sequential(
        torch.nn.Linear(self.feature_dimension, self.
↪feature_hidden_dimension),
        nn.ReLU(),
        torch.nn.Linear(self.feature_hidden_dimension, self.
↪feature_hidden_dimension),
        nn.ReLU(),
        torch.nn.Linear(self.feature_hidden_dimension, self.
↪output_dimension)
    )
    self.to(device)
    self.float()
    def forward(self, input, anchor, anchor_label):
        feature_input = self.feature_sequential(input)
        feature_anchor = self.feature_sequential(anchor)
        feature_space_difference_input_from_anchor = feature_input -
↪feature_anchor
        label_difference_input_from_anchor = self.
↪auxiliary_sequential(feature_space_difference_input_from_anchor)
        prediction = anchor_label + label_difference_input_from_anchor
        return prediction

```

```

[29]: from tools import SaveBestModel, PatienceEarlyStopping, Scheduler, plot_loss
class Manager:
    """ DOES: train & evaluate a Siamese network
        """
    def __init__(self, epoch, cross_validation_round):
        self._network = SiameseNetwork(device, s['input dimension'], s['feature_
↪dimension'], s['output dimension'])
        self._network.apply(self.initializer)
        self._learning_rate = s['learning rate']
        self._optimizer = torch.optim.Adam(
            params=self._network.parameters(), lr=self._learning_rate,
            weight_decay=3e-3)
        self._energy = nn.MSELoss()
        self._train_loss = []
        self._valid_loss = []
        self._test_loss = []
        self._epoch = epoch
        self._stopper = PatienceEarlyStopping(patience=5, min_delta=1e-7)
        self._cross_validation_round = cross_validation_round
        self._saver = SaveBestModel(s['best model folder'])
        self._scheduler = Scheduler(optimizer=self._optimizer,

```

```

        minimum_learning_rate=1e-6, patience=5, factor=0.5)
def initializer(self, layer):
    if type(layer) == nn.Linear:
        nn.init.kaiming_normal_(layer.weight) # normal version
def _step(self, job):
    input, input_label, anchor, anchor_label = job
    # print(f"input dtype is {input_1.dtype}")
    prediction = self._network(input, anchor, anchor_label)
    loss = self._energy(input_label, prediction)
    return loss
def train(self, train_dataloader, valid_dataloader):
    """ DOES: calculate loss from tasks
    NOTE: we have a BATCH of tasks here """
    for e in range(self._epoch):
        # print(f"train() epoch {e}")
        batch_train_loss = []
        for _, batch in enumerate(train_dataloader):
            self._optimizer.zero_grad()
            loss = self._step(batch)
            loss.backward()
            self._optimizer.step()
            batch_train_loss.append(loss.item())
        self._train_loss.append(np.mean(batch_train_loss))
        batch_valid_loss = []
        with torch.no_grad():
            for _, batch in enumerate(valid_dataloader):
                loss = self._step(batch)
                batch_valid_loss.append(loss.item())
            self._valid_loss.append(np.mean(batch_valid_loss))
        # saving, early stopping, scheduler for EACH epoch!
        self._saver(current_loss=np.mean(batch_valid_loss),
                    model=self._network,
                    round=self._cross_validation_round
                    )
        self._scheduler(np.mean(batch_valid_loss))
        self._stopper(np.mean(batch_valid_loss))
        if self._stopper.early_stop == True:
            print(f"EARLY STOPPING @ epoch {e}")
            break
        # summary printout, after we're done with epochs
        print(f"min train loss: {np.min(self._train_loss)}")
        print(f"min valid loss: {np.min(self._valid_loss)}")
        plot_loss(self._train_loss, self._valid_loss)
        return np.min(self._valid_loss)
def test(self, test_dataloader):
    with torch.no_grad():
        batch_test_loss = []

```

```

        for _, batch in enumerate(test_dataloader):
            loss = self._step(batch)
            batch_test_loss.append(loss.item())
        self._test_loss.append(np.mean(batch_test_loss))
    return np.min(self._test_loss)

```

```

[30]: from torch.utils.data import DataLoader
      from tools import SiameseDataset, SaveBestCrossValidationModel

      CV_saver = SaveBestCrossValidationModel(s['best model folder'])
      test_indices = data_dictionary[s['data']]['test indices']
      epoch = s['epoch']
      print(f"data: {s['data']}")
      cross_validation_loss = []
      for cross_validation_round, (train, valid) in enumerate(zip(
          data_dictionary[s['data']]['train indices'],
          data_dictionary[s['data']]['valid indices'])):
          if cross_validation_round < s['cross validation round']:
              print(f"CV round {cross_validation_round}")
              network_object = Manager(epoch, cross_validation_round)
              valid_loss = network_object.train(
                  DataLoader(SiameseDataset(
                      data_dictionary[s['data']]['data'],
                      data_dictionary[s['data']]['label'],
                      data_dictionary[s['data']]['train indices'][cross_validation_round],
                      device=device,), shuffle=False, batch_size=s['batch size']),
                  DataLoader(SiameseDataset(
                      data_dictionary[s['data']]['data'],
                      data_dictionary[s['data']]['label'],
                      data_dictionary[s['data']]['valid indices'][cross_validation_round],
                      device=device,), shuffle=False, batch_size=s['batch size']))
              CV_saver(current_loss=valid_loss, round=cross_validation_round)
              cross_validation_loss.append(valid_loss)

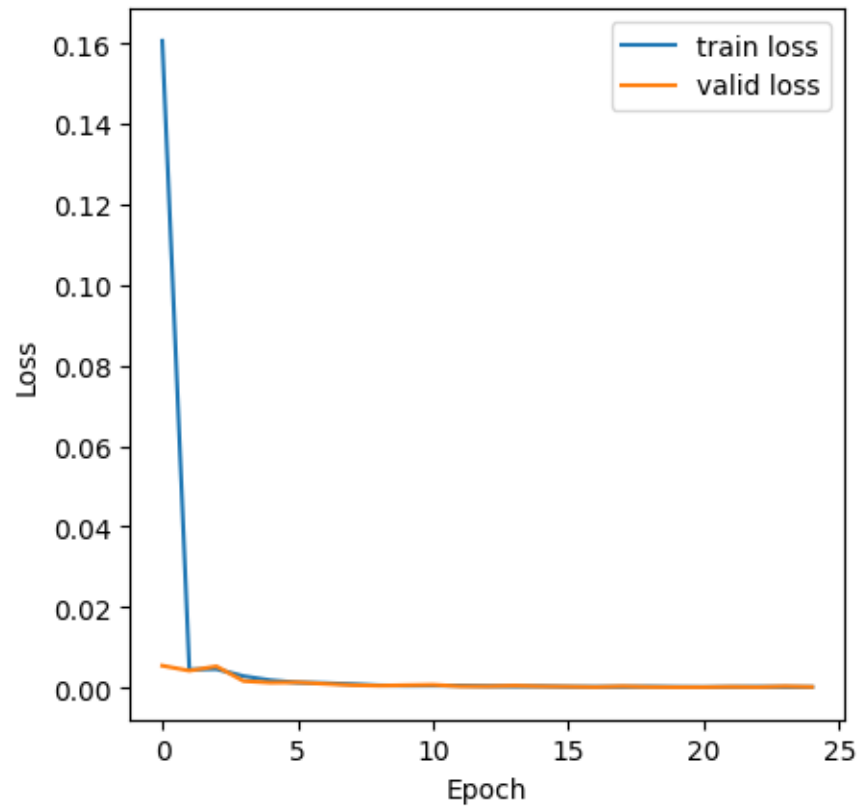
      print()
      print(f"\nbest model is: {CV_saver.best_model_name} with {CV_saver.
          ↪current_best_loss}")
      print(f"The aggregate performance is: mean {np.mean(cross_validation_loss)},
          ↪std {np.std(cross_validation_loss)}")

```

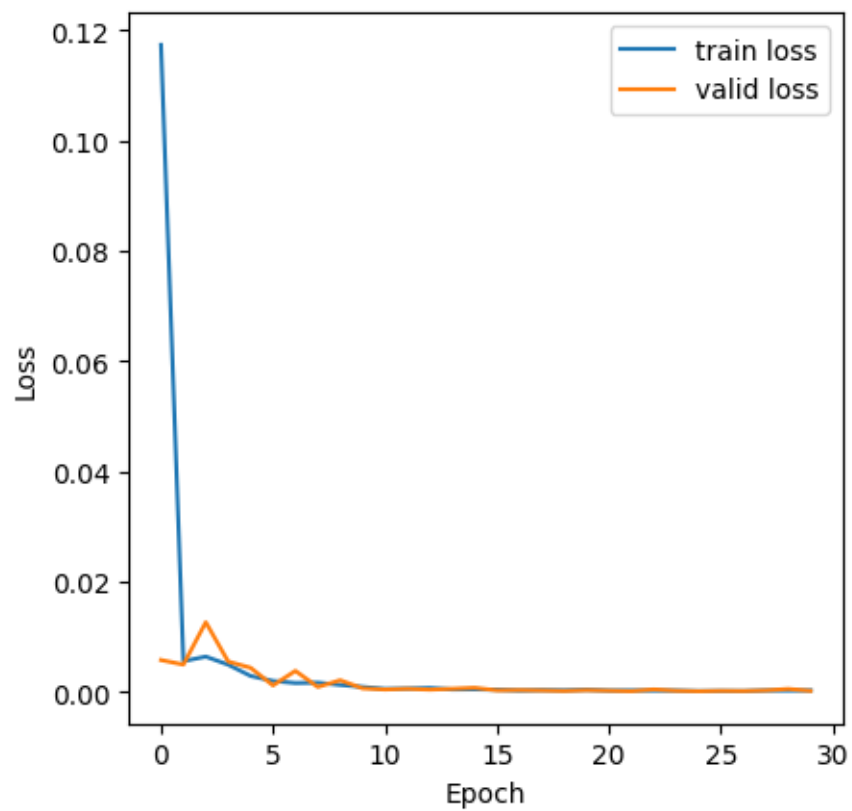
```

data: pressure_230516_discrete
CV round 0
EARLY STOPPING @ epoch 24
min train loss: 0.00017306936358429747
min valid loss: 0.00013779852042716811

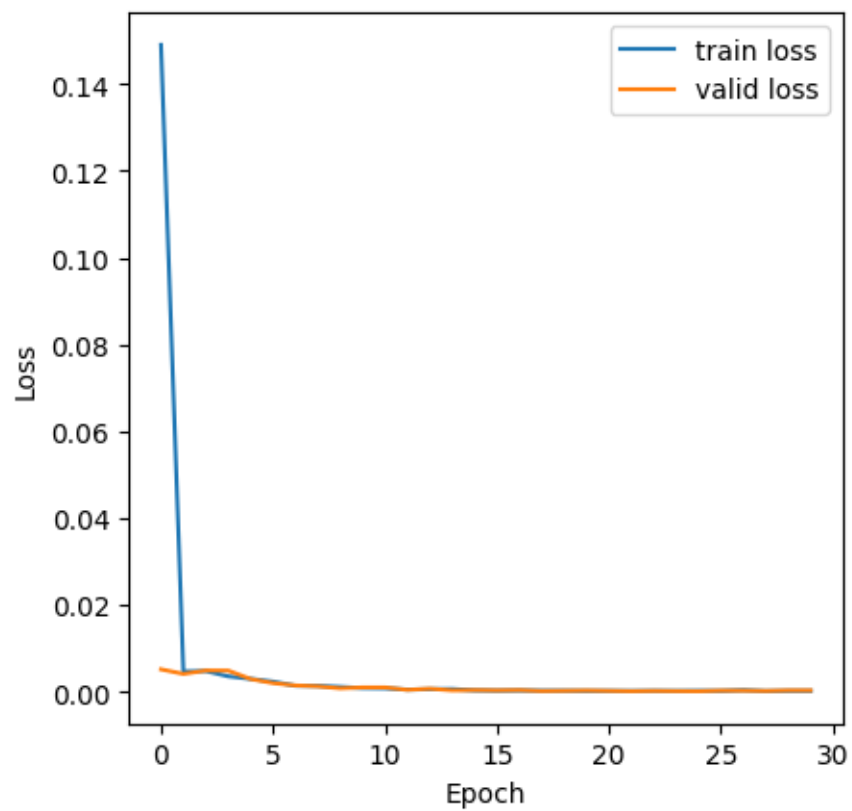
```



CV round 1  
EARLY STOPPING @ epoch 29  
min train loss: 0.0001266538158895194  
min valid loss: 0.00011381913645891473

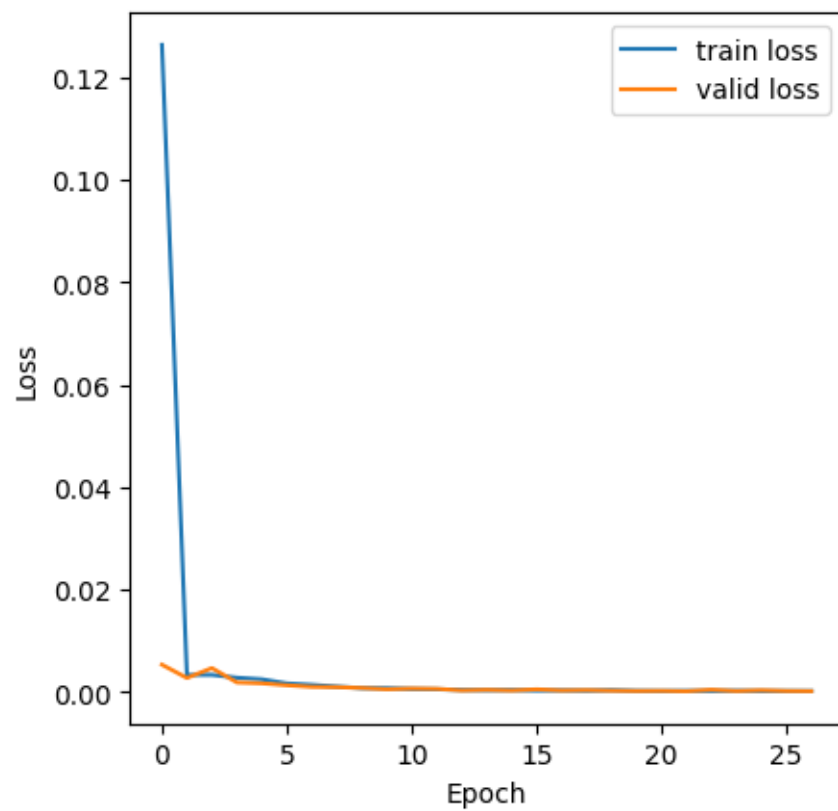


CV round 2  
EARLY STOPPING @ epoch 29  
min train loss: 0.0001323393067815976  
min valid loss: 0.00010679203569452511

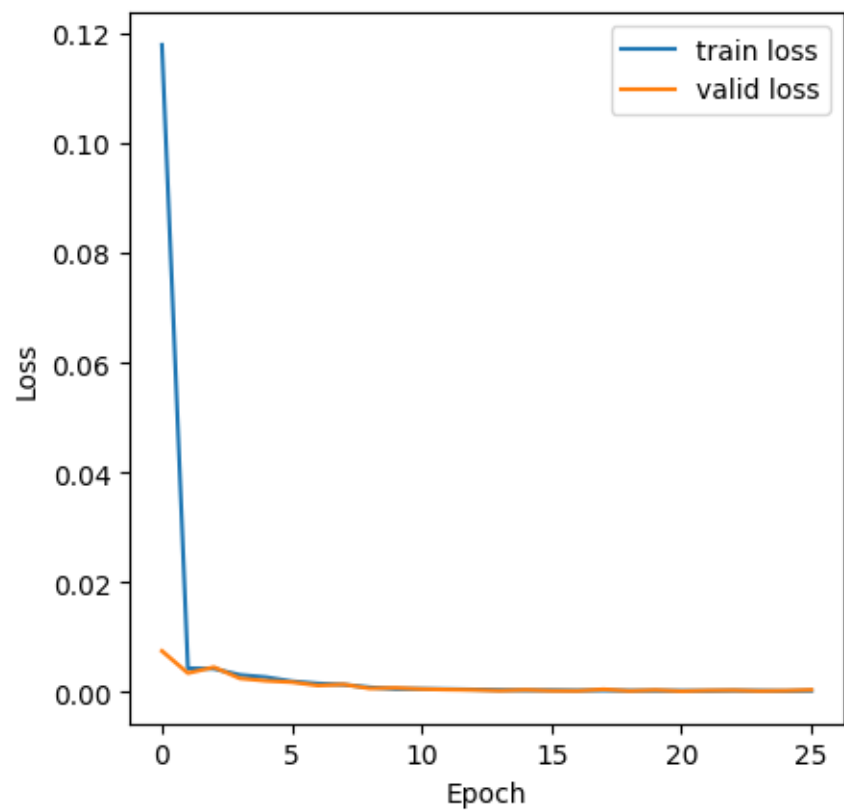


CV round 3  
EARLY STOPPING @ epoch 26  
min train loss: 0.0001268067367081213  
min valid loss: 0.0001007187133836851

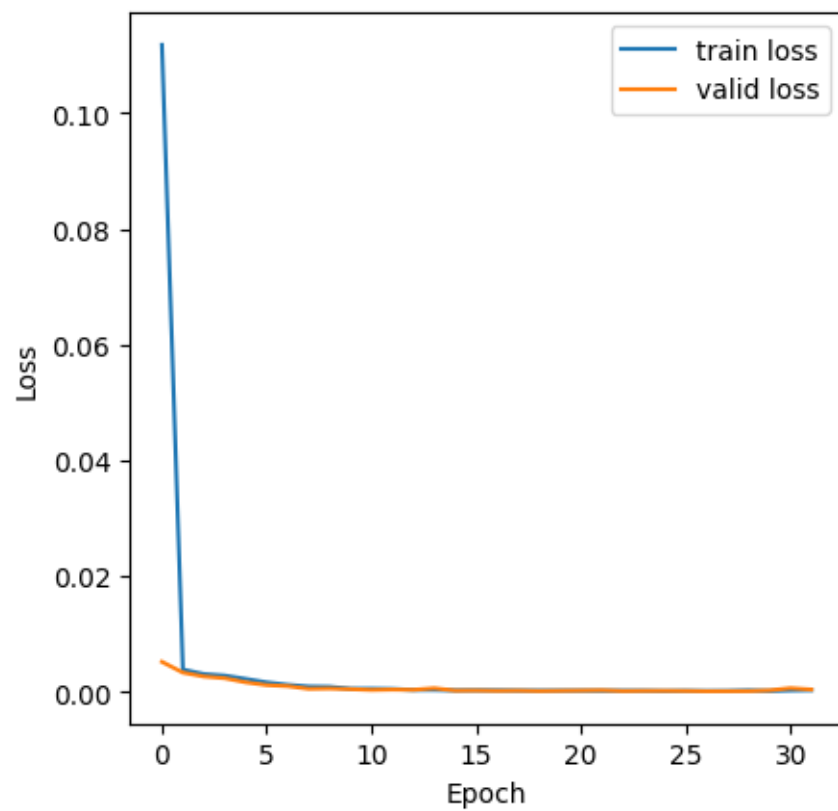




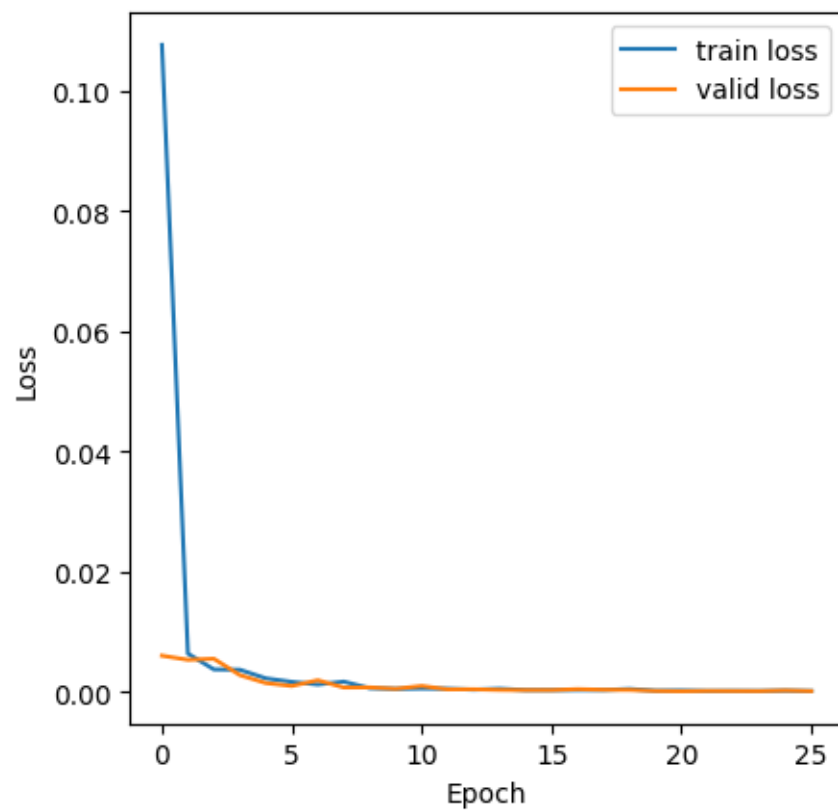
CV round 4  
EARLY STOPPING @ epoch 25  
min train loss: 0.0001400993069348095  
min valid loss: 0.00010316407838217856



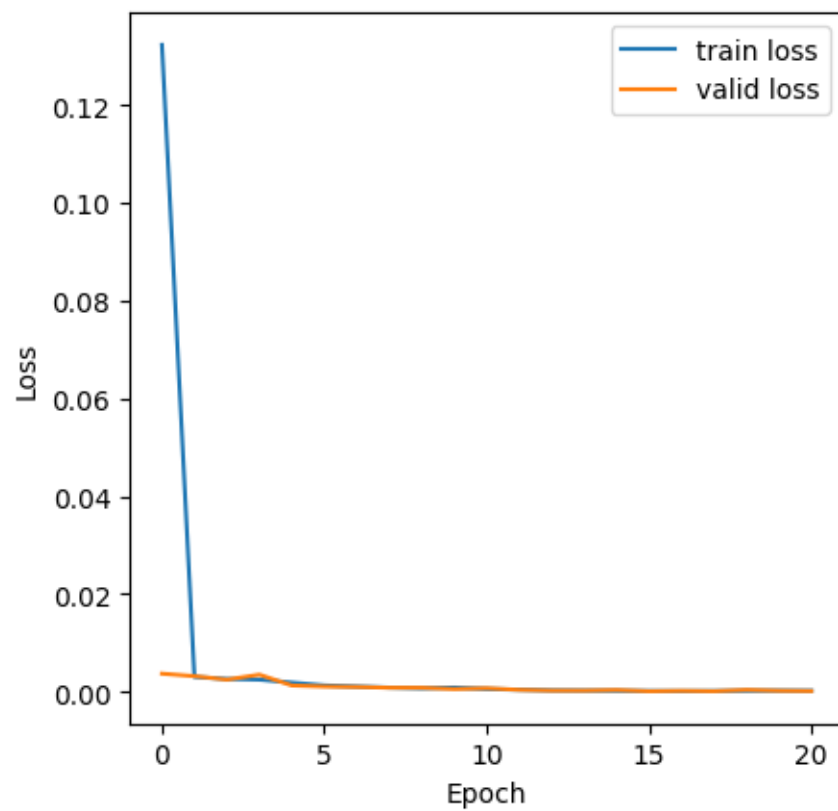
CV round 5  
EARLY STOPPING @ epoch 31  
min train loss: 0.00010800854619116184  
min valid loss: 7.263691554726392e-05



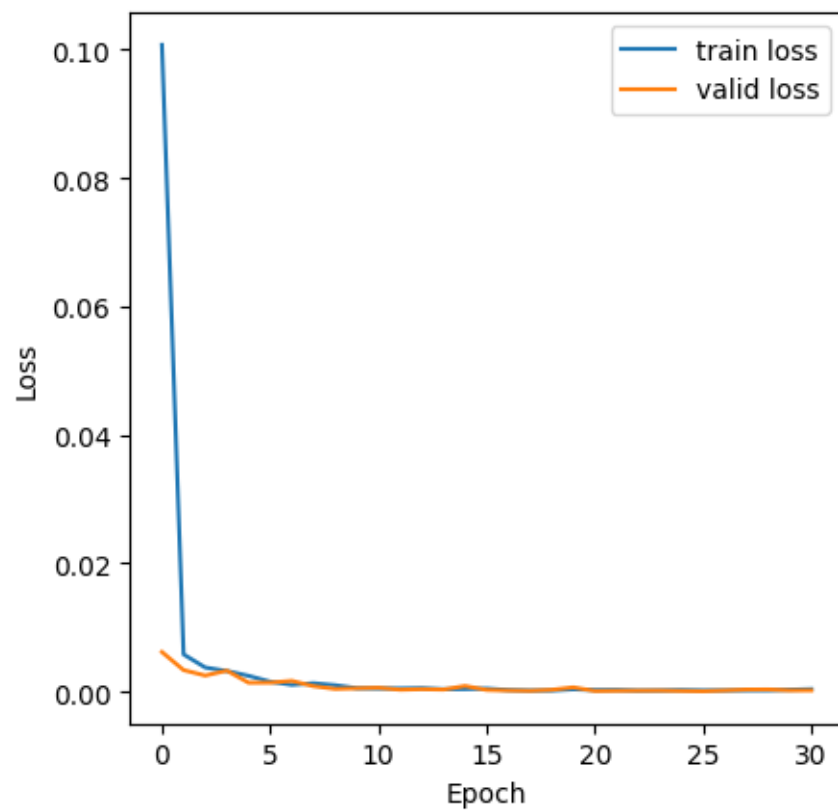
CV round 6  
EARLY STOPPING @ epoch 25  
min train loss: 0.00013005256256623065  
min valid loss: 9.900300847220933e-05



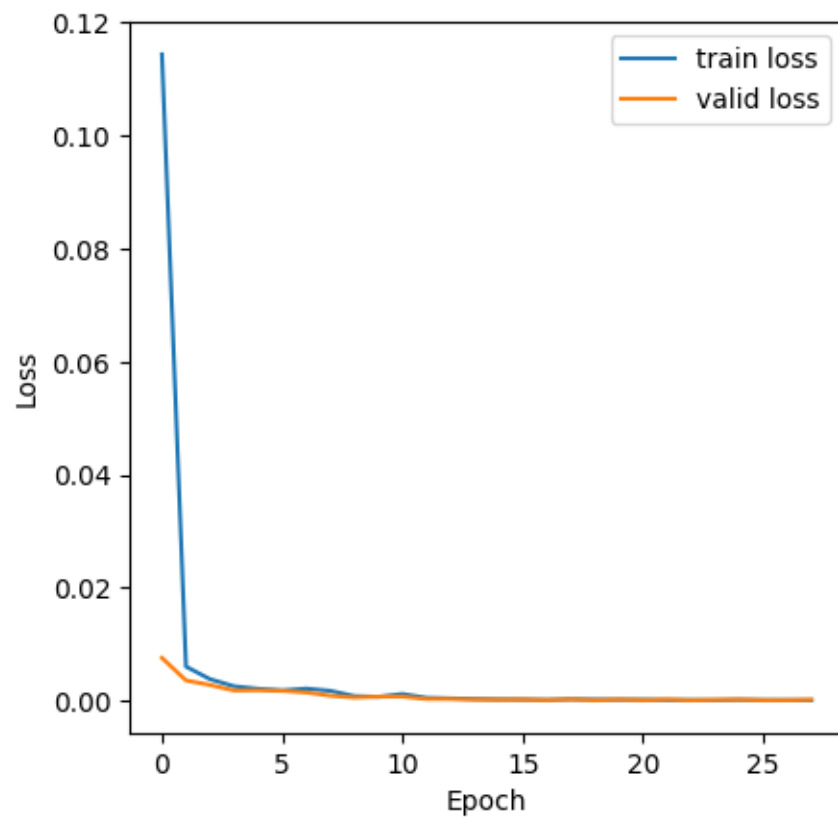
CV round 7  
EARLY STOPPING @ epoch 20  
min train loss: 0.00018565682099010289  
min valid loss: 0.0001699877911960357



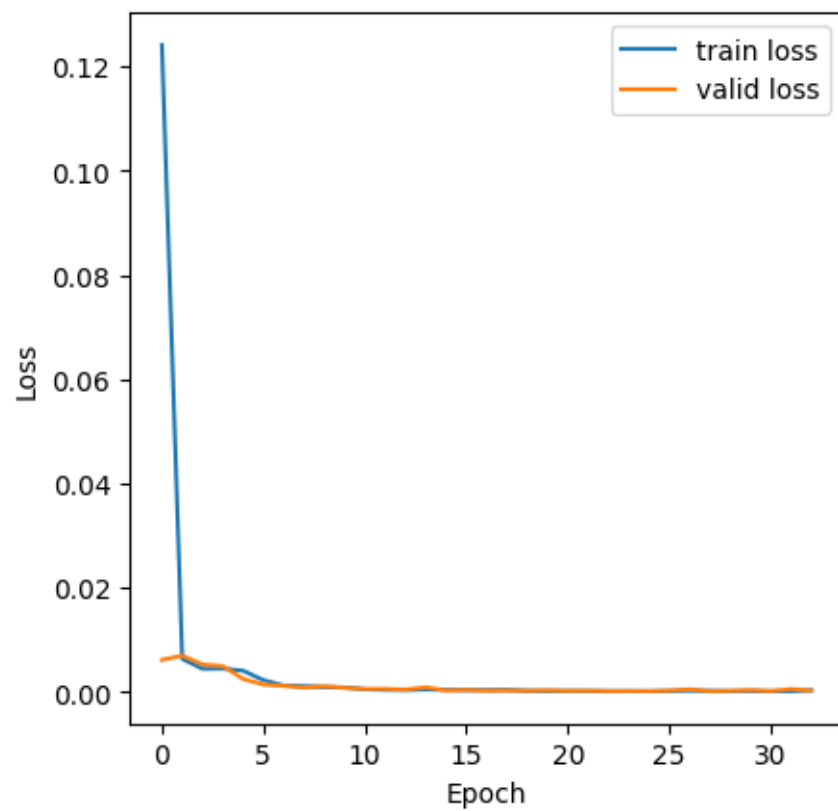
CV round 8  
EARLY STOPPING @ epoch 30  
min train loss: 0.00017517547884463884  
min valid loss: 7.70257886415493e-05



CV round 9  
EARLY STOPPING @ epoch 27  
min train loss: 0.00014701743103532036  
min valid loss: 0.0001264191914742696

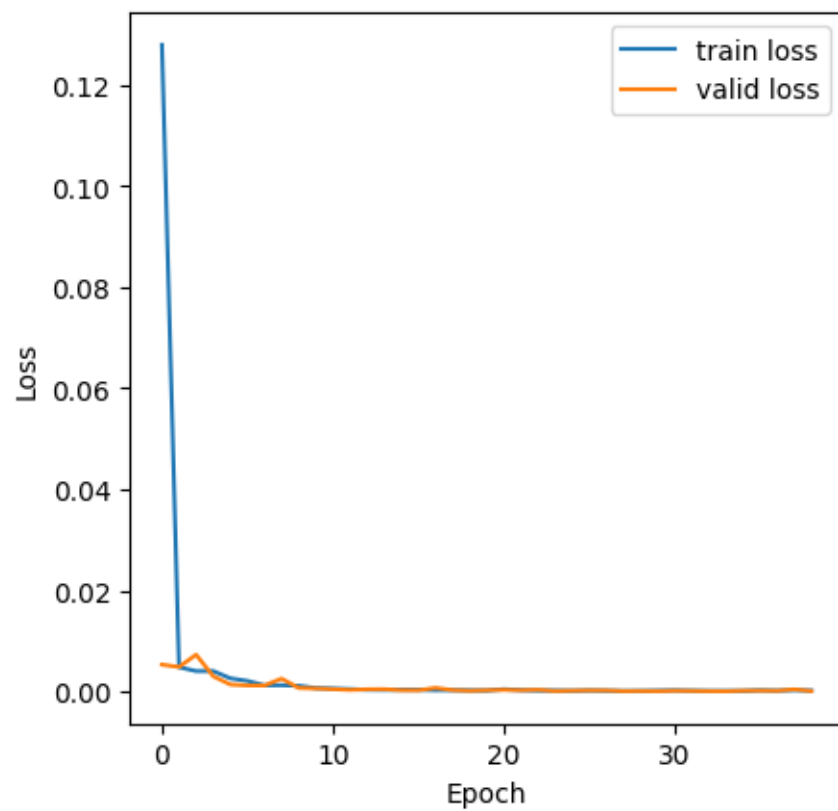


CV round 10  
EARLY STOPPING @ epoch 32  
min train loss: 0.00014796297936382787  
min valid loss: 0.00015321240925914026

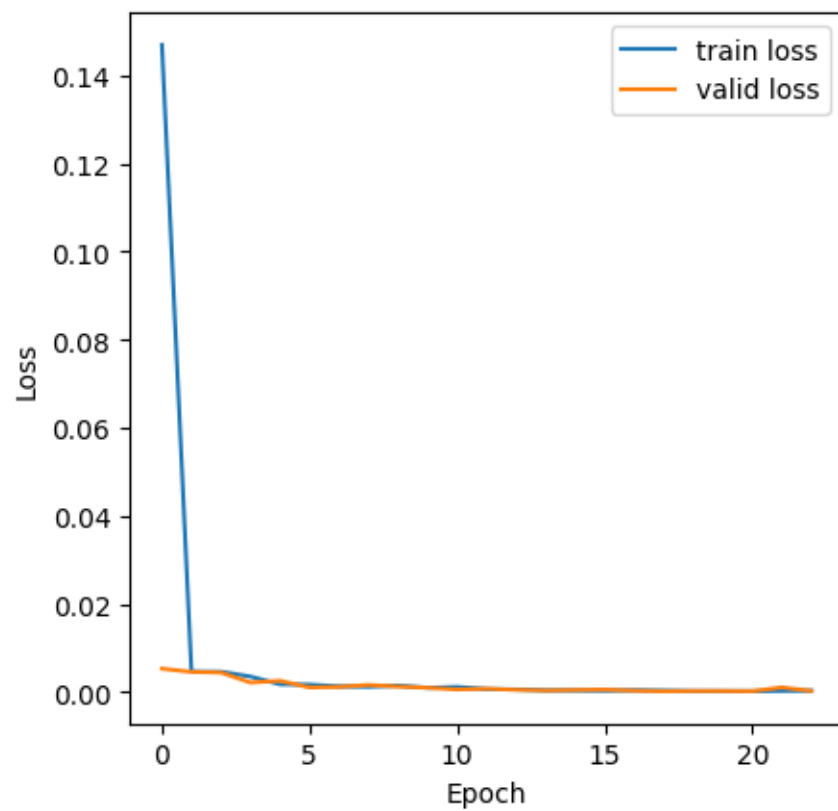


CV round 11  
EARLY STOPPING @ epoch 38  
min train loss: 0.00010934330655965658  
min valid loss: 6.864582269372477e-05

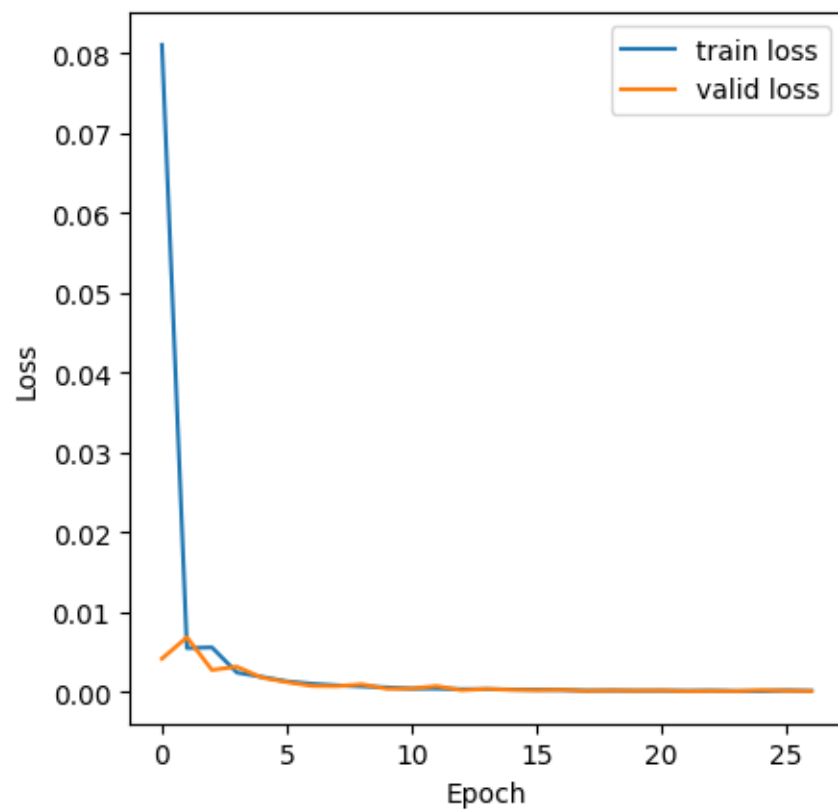




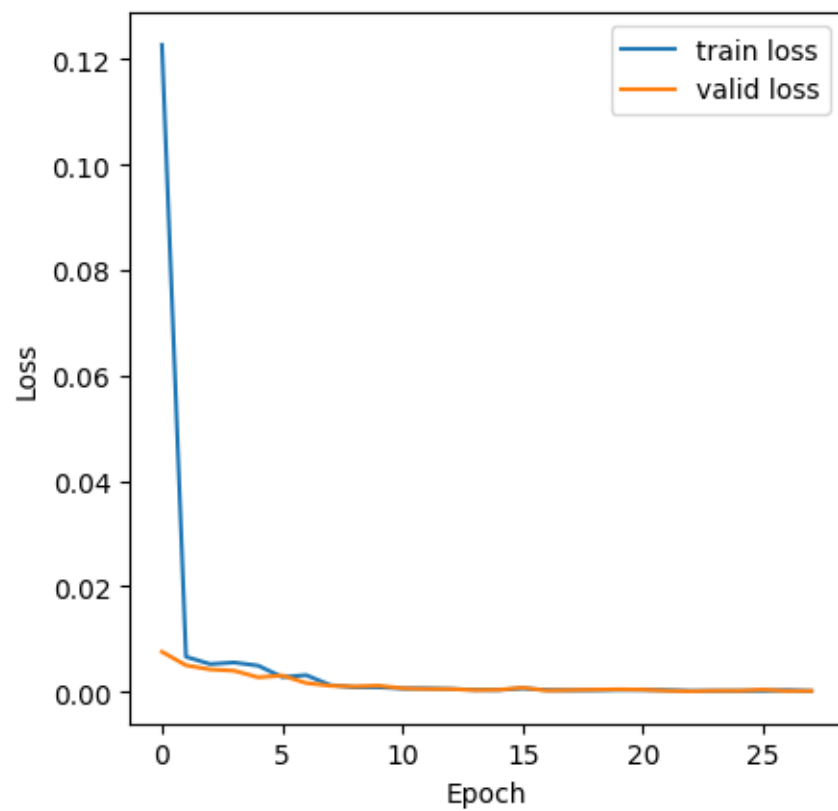
CV round 12  
EARLY STOPPING @ epoch 22  
min train loss: 0.00018911994742335414  
min valid loss: 0.0001840337013163662



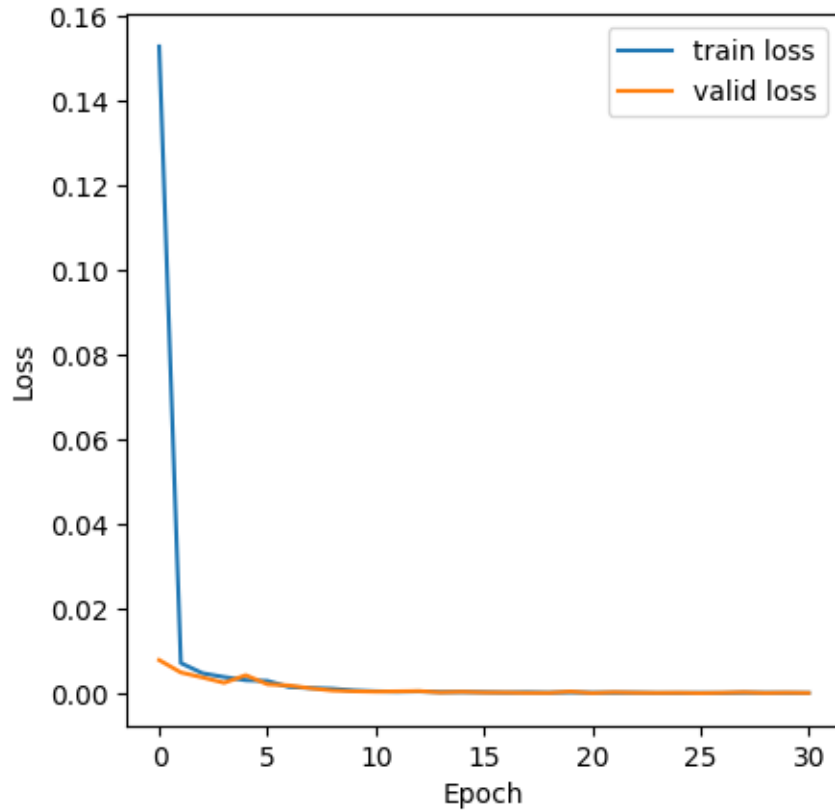
CV round 13  
EARLY STOPPING @ epoch 26  
min train loss: 9.736020021426322e-05  
min valid loss: 9.21398273021623e-05



CV round 14  
EARLY STOPPING @ epoch 27  
min train loss: 0.0001700021616257304  
min valid loss: 0.00011745436950150179



CV round 15  
EARLY STOPPING @ epoch 30  
min train loss: 0.0001336912813051392  
min valid loss: 0.00010704575038289477



best model is: CV=11.pth with 6.864582269372477e-05  
The aggregate performance is: mean 0.00011436856625834935, std  
3.215126787094058e-05

```
[31]: network_object._network.load_state_dict(torch.load(s['best model folder'] +
    ↪CV_saver.best_model_name))
test_loss = network_object.test(
    DataLoader(SiameseDataset(
        data_dictionary[s['data']]['data'],
        data_dictionary[s['data']]['label'],
        data_dictionary[s['data']]['test indices'],
        device=device,), shuffle=False, batch_size=s['batch size']))
print(f"testing loss: {test_loss}")
```

testing loss: 0.00012311287446209462