CV_siamese_v5

July 7, 2023

1 run load_data.ipynb BEFORE running this!

[11]:

```
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'     : 5e-5,
    'input dimension'   : 10000,
    'output dimension'  : 1,
    'feature dimension' : 300,
    'epoch'             : 1000,
    'epoch development' : 36,
    'data'              : 'temperature_230509_discrete',
    'cross validation round': 16,
    'cross validation round-development' : 3,
    'best model folder' : 'siamese_best_model/'
}
```

[12]:

```
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
```

[13]:

```
# 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
```

[14]:

```
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 = 32
        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_dimension),
            nn.ReLU(),
            torch.nn.Linear(self.feature_dimension, self.feature_dimension),
            nn.ReLU(),
            torch.nn.Linear(self.feature_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
```

[15]:

```
import matplotlib.pyplot as plt
def plot_loss(train_loss, valid_loss):
    plt.subplot()
    plt.plot(train_loss)
    plt.plot(valid_loss)
    plt.legend(["train loss", "valid loss"], loc ="upper right")
    plt.show()
```

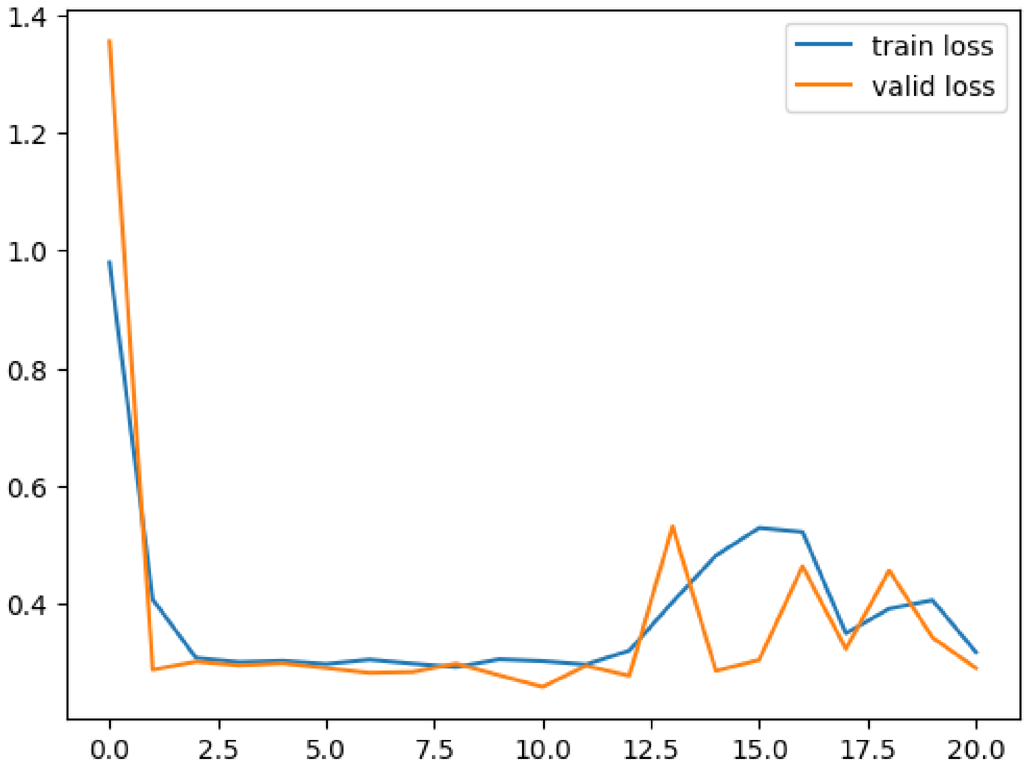
[16]:

```
from tools import SaveBestModel, PatienceEarlyStopping, Scheduler
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 = SiameseNetwork(device, 10000, 16, 1)
        self._network.apply(self.initializer)
        self._learning_rate = 1e-4
        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=10, 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):
                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 = []
            with torch.no_grad():
                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)
```

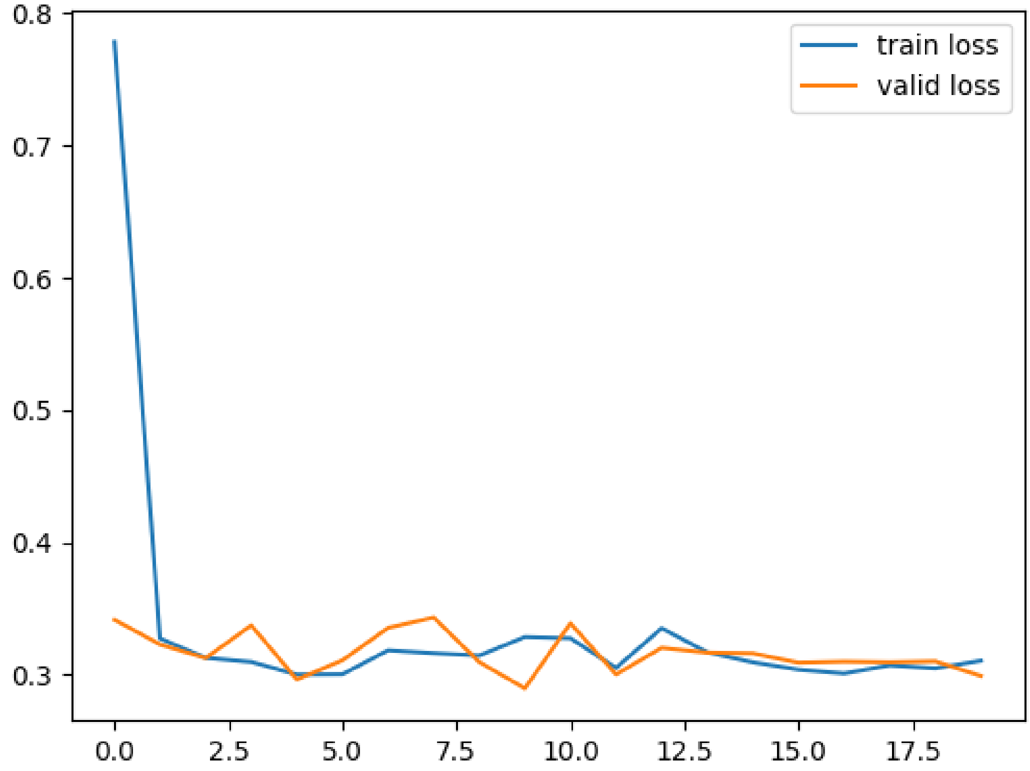
[17]:
```
from torch.utils.data import DataLoader
from tools import SiameseDataset, SaveBestCrossValidationModel

CV_saver = SaveBestCrossValidationModel(s['best model folder'])
test_indices = data_dictionary[s['data']]['test indices']
epoch = s['epoch']
print(f"data: {s['data']}")
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=32),
            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=32))
        CV_saver(current_loss=valid_loss, round=cross_validation_round)
print(f"\nbest model is: {CV_saver.best_model_name} with {CV_saver.
 ↪current_best_loss}")
```

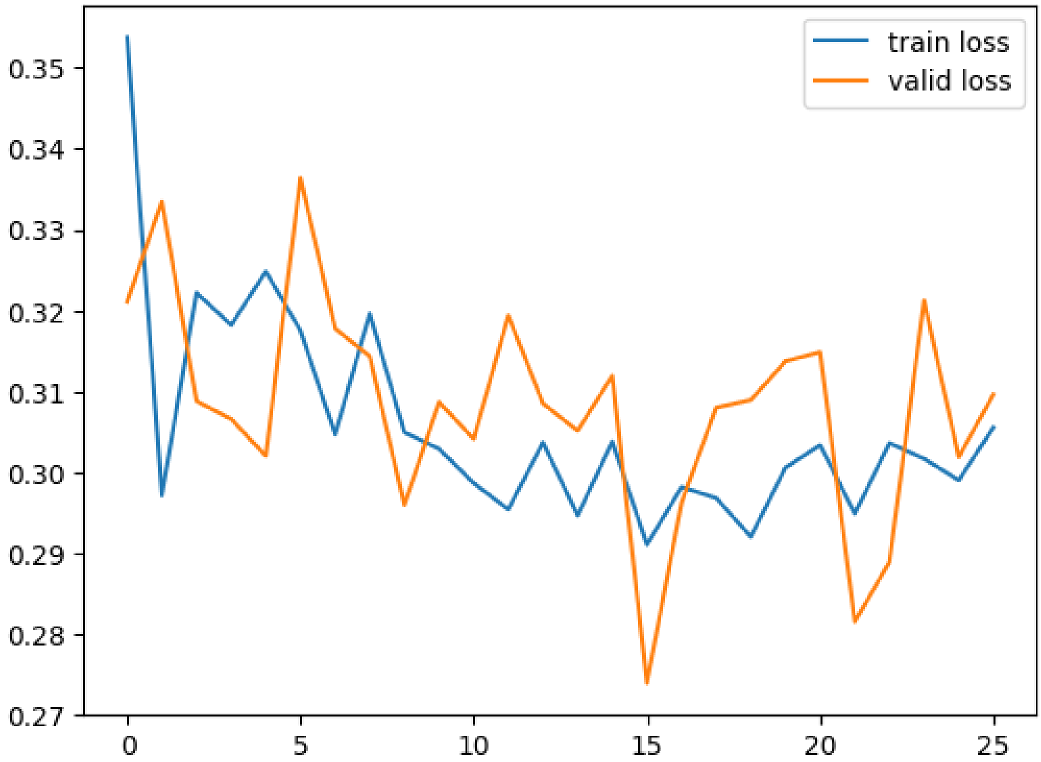
```
data: temperature_230509_discrete
CV round 0
EARLY STOPPING @ epoch 20
min train loss: 0.2921668171431079
min valid loss: 0.2586006294739874
```



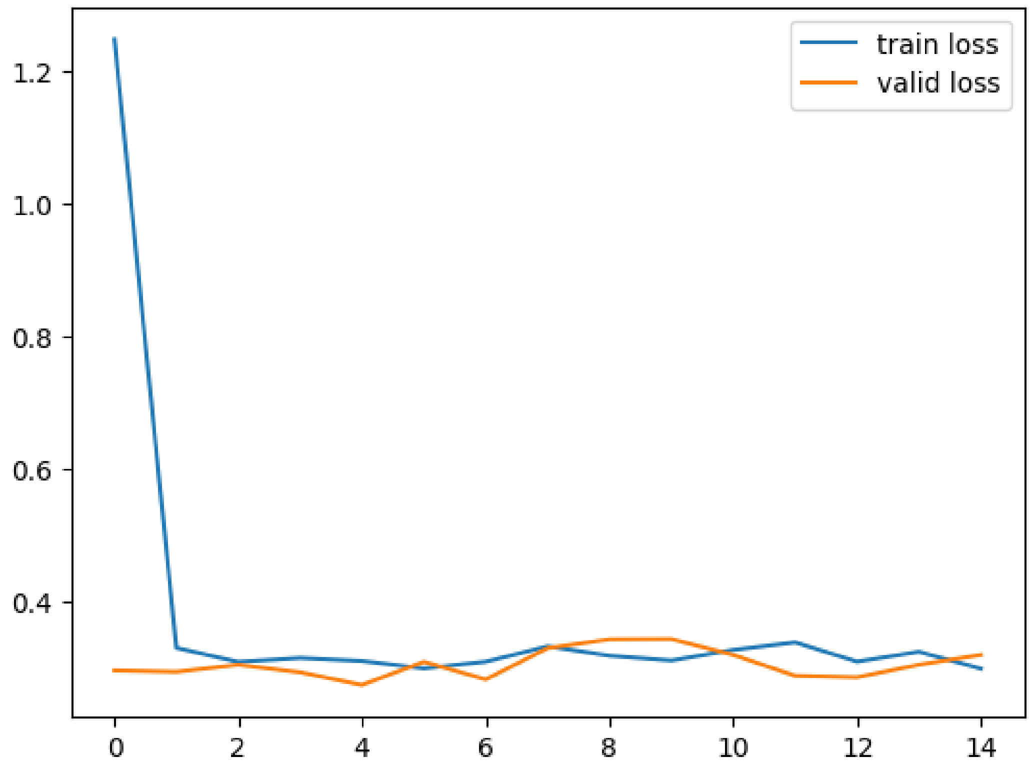
```
CV round 1
EARLY STOPPING @ epoch 19
min train loss: 0.30011898617852817
min valid loss: 0.2896560665808226
```



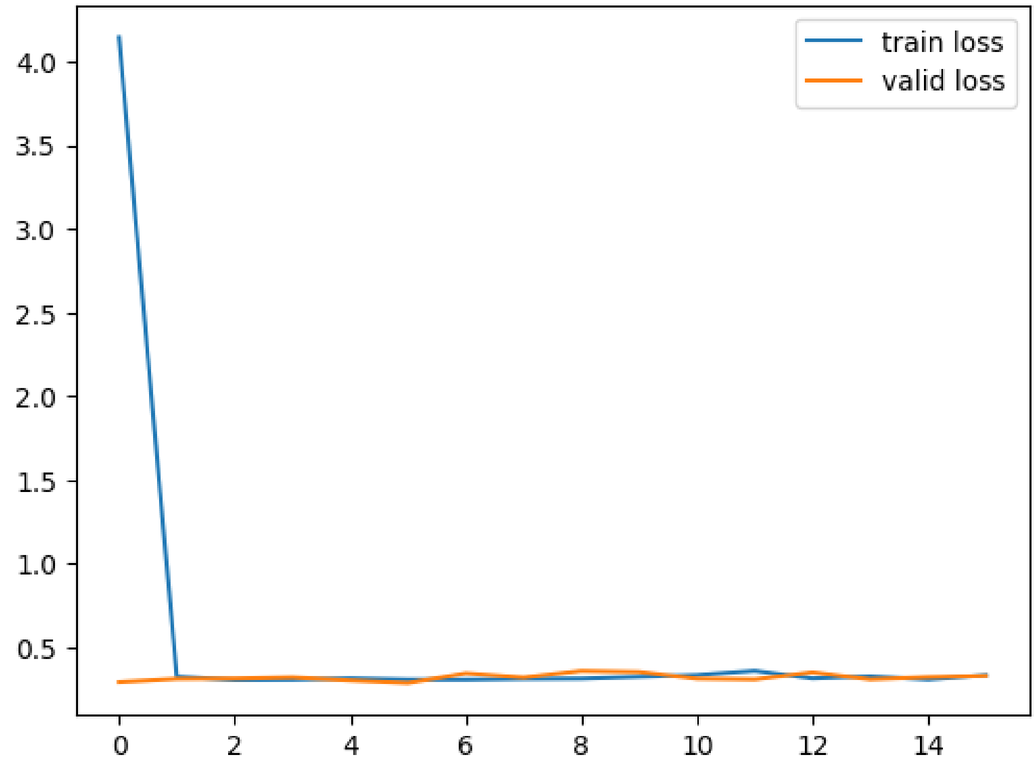
```
CV round 2
EARLY STOPPING @ epoch 25
min train loss: 0.29107263447208837
min valid loss: 0.2739839632260172
```



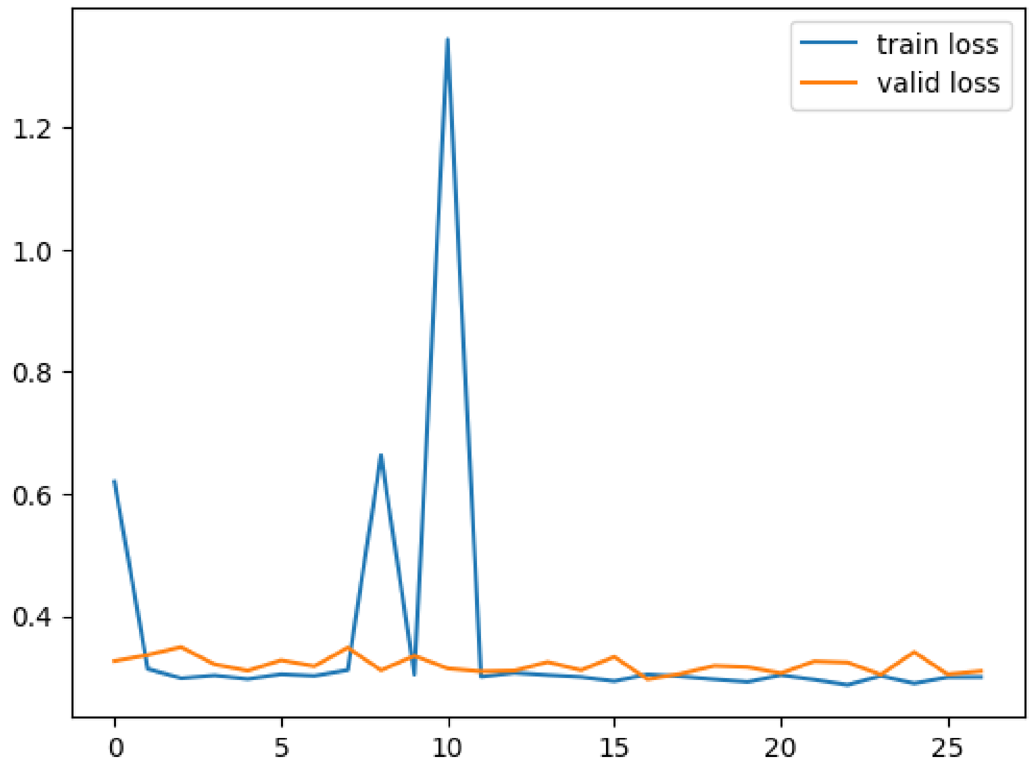
```
CV round 3
EARLY STOPPING @ epoch 14
min train loss: 0.29763769911545696
min valid loss: 0.2730579838940972
```



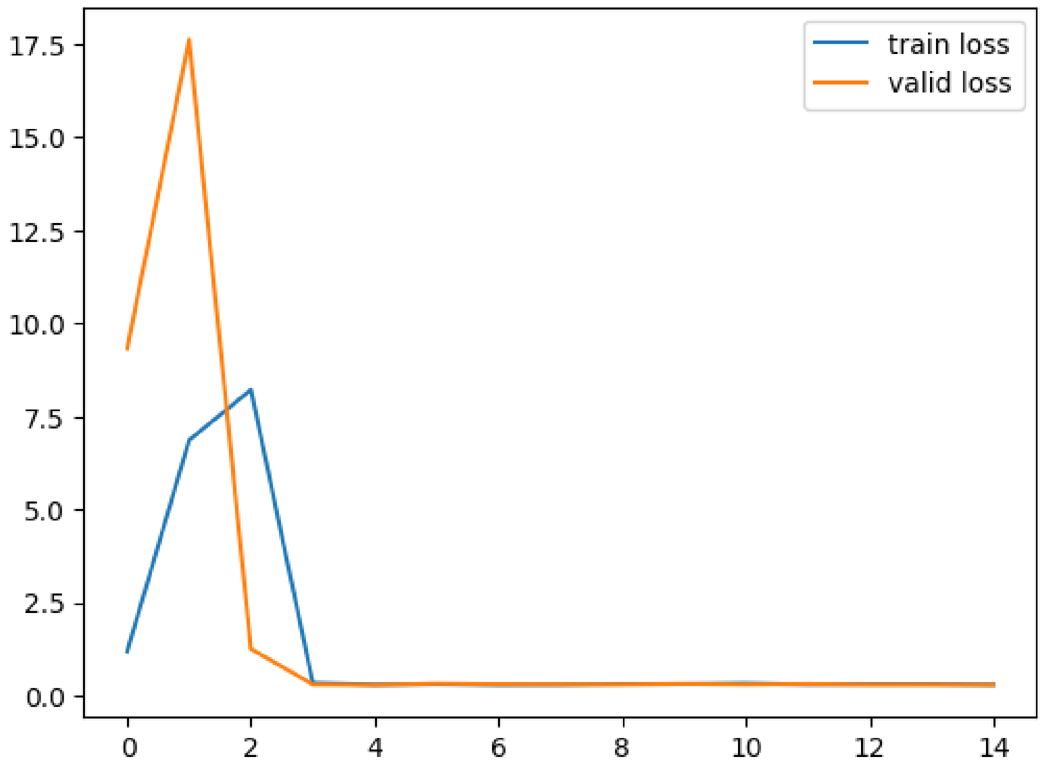
```
CV round 4
EARLY STOPPING @ epoch 15
min train loss: 0.305350693902283
min valid loss: 0.28792562139661687
```



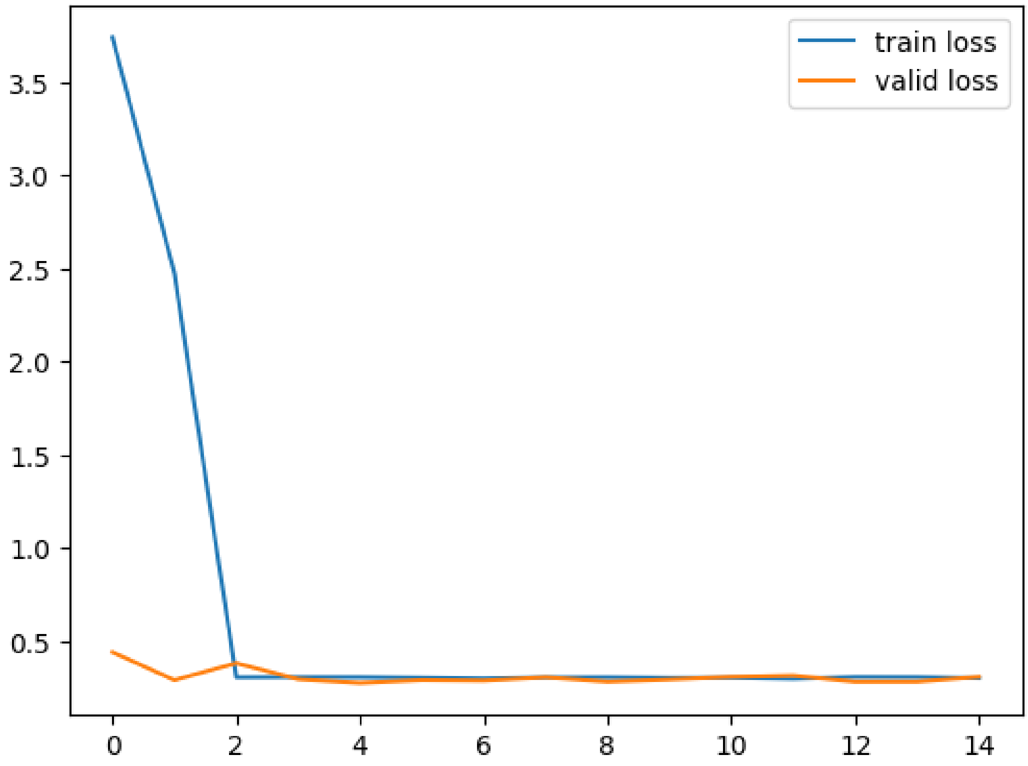
```
CV round 5
EARLY STOPPING @ epoch 26
min train loss: 0.28732144076264265
min valid loss: 0.2964565314744648
```



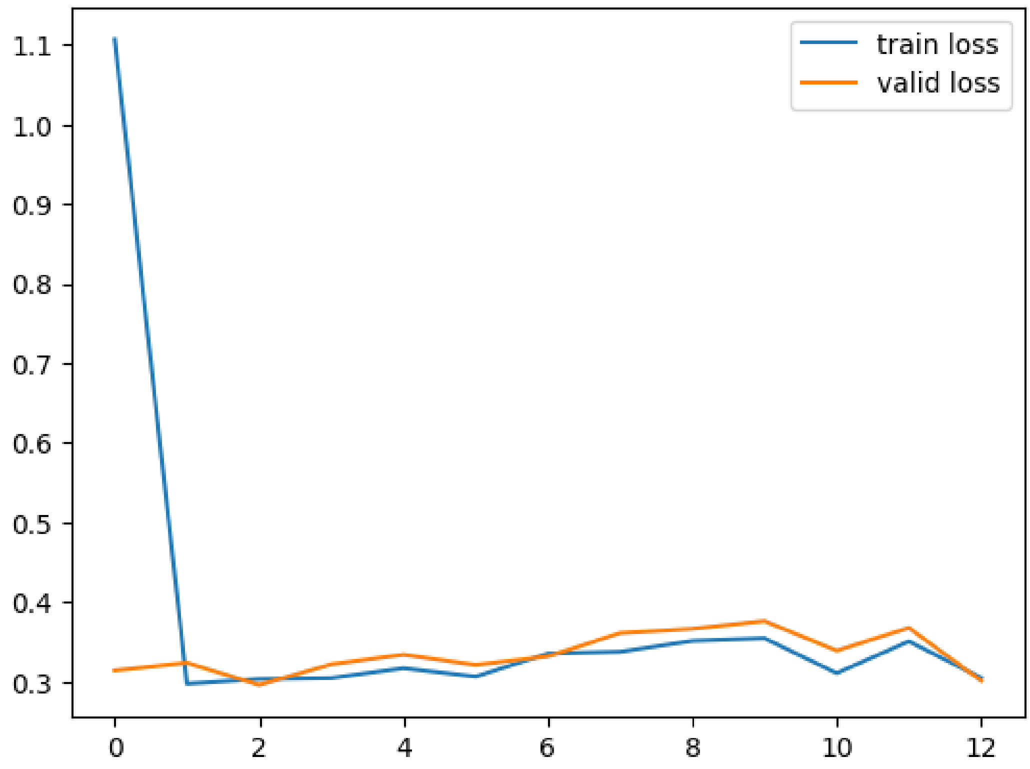
```
CV round 6
EARLY STOPPING @ epoch 14
min train loss: 0.28683323142203415
min valid loss: 0.2821706564802873
```



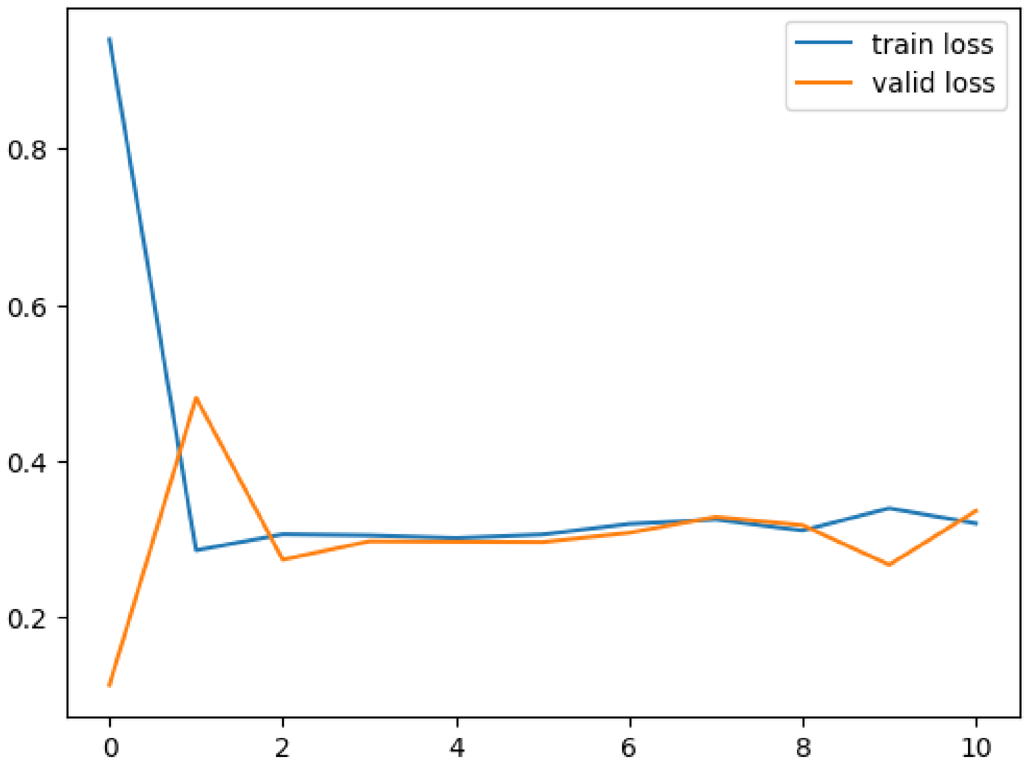
```
CV round 7
EARLY STOPPING @ epoch 14
min train loss: 0.29972398777802783
min valid loss: 0.2774925357417056
```



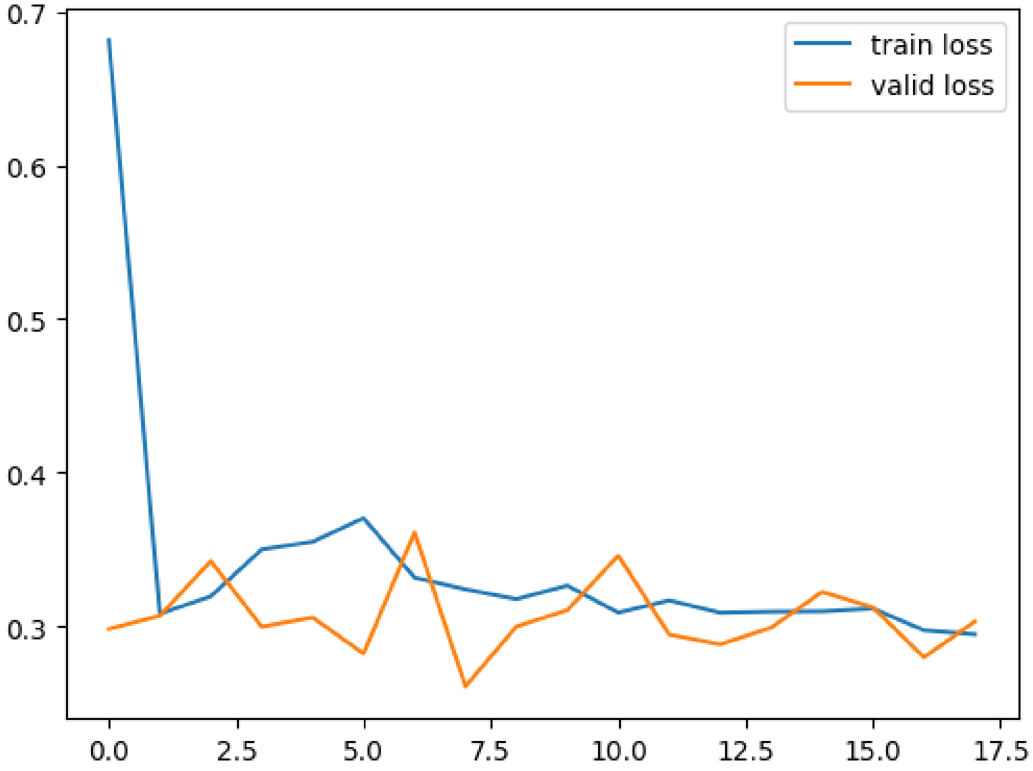
```
CV round 8
EARLY STOPPING @ epoch 12
min train loss: 0.2971803545951843
min valid loss: 0.2956592746471104
```



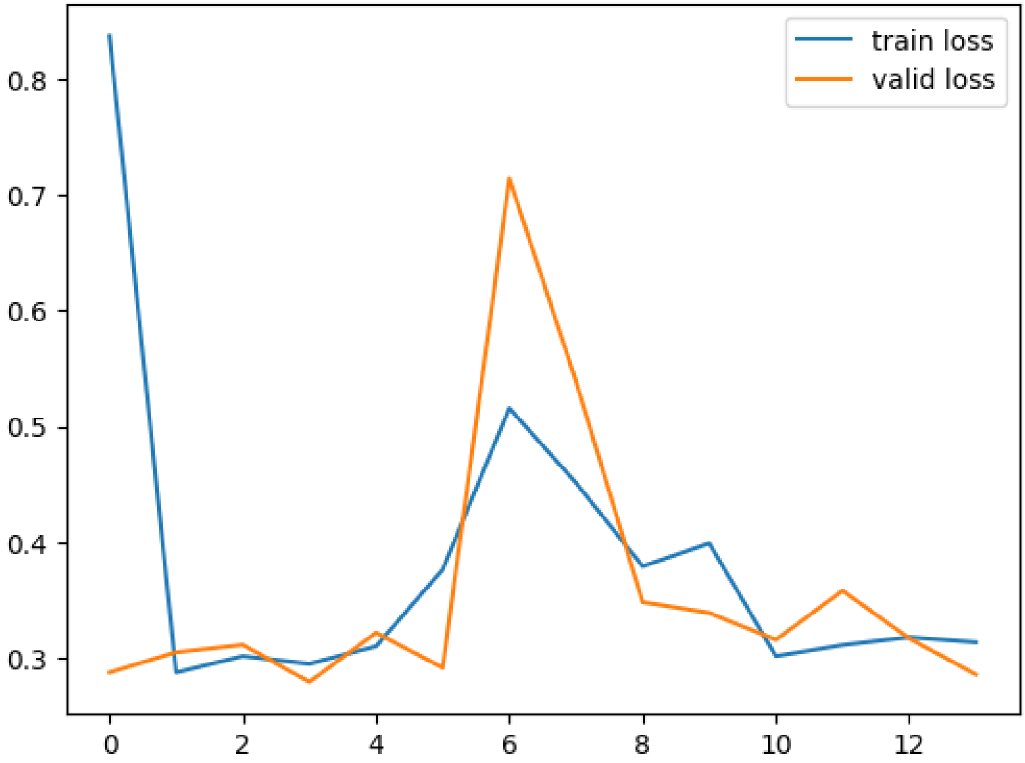
```
CV round 9
EARLY STOPPING @ epoch 10
min train loss: 0.28574483799324796
min valid loss: 0.11320586855474271
```



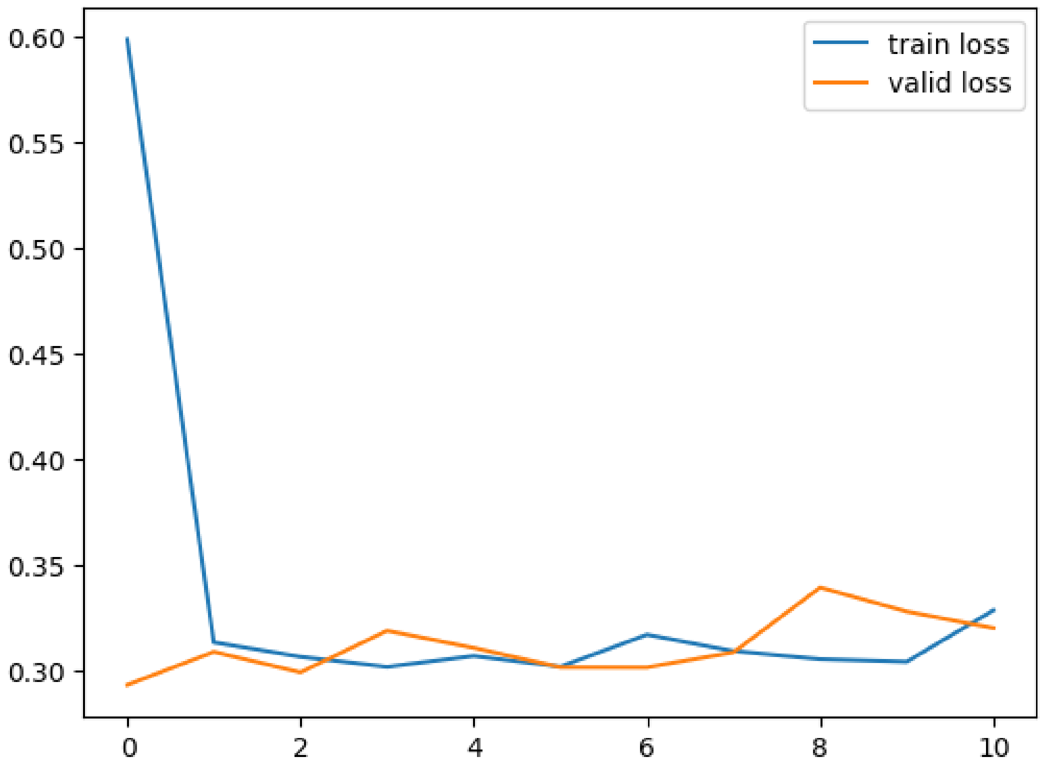
```
CV round 10
EARLY STOPPING @ epoch 17
min train loss: 0.2947077210428137
min valid loss: 0.2608380090249212
```



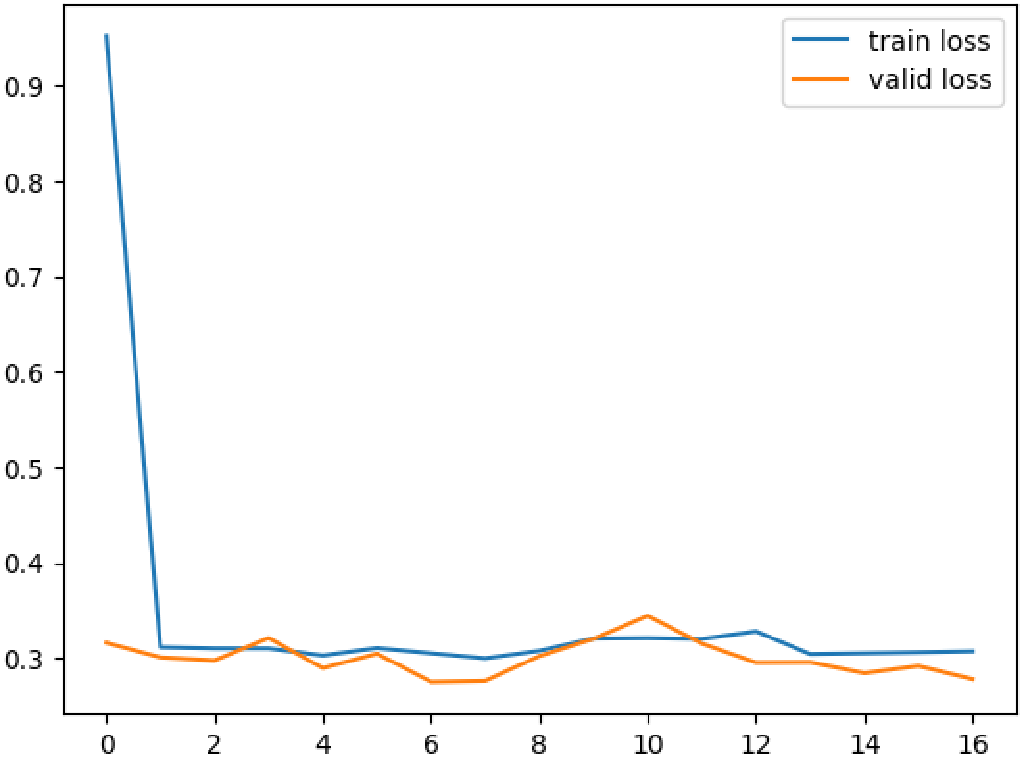
```
CV round 11
EARLY STOPPING @ epoch 13
min train loss: 0.2877337613679243
min valid loss: 0.27950759467325714
```



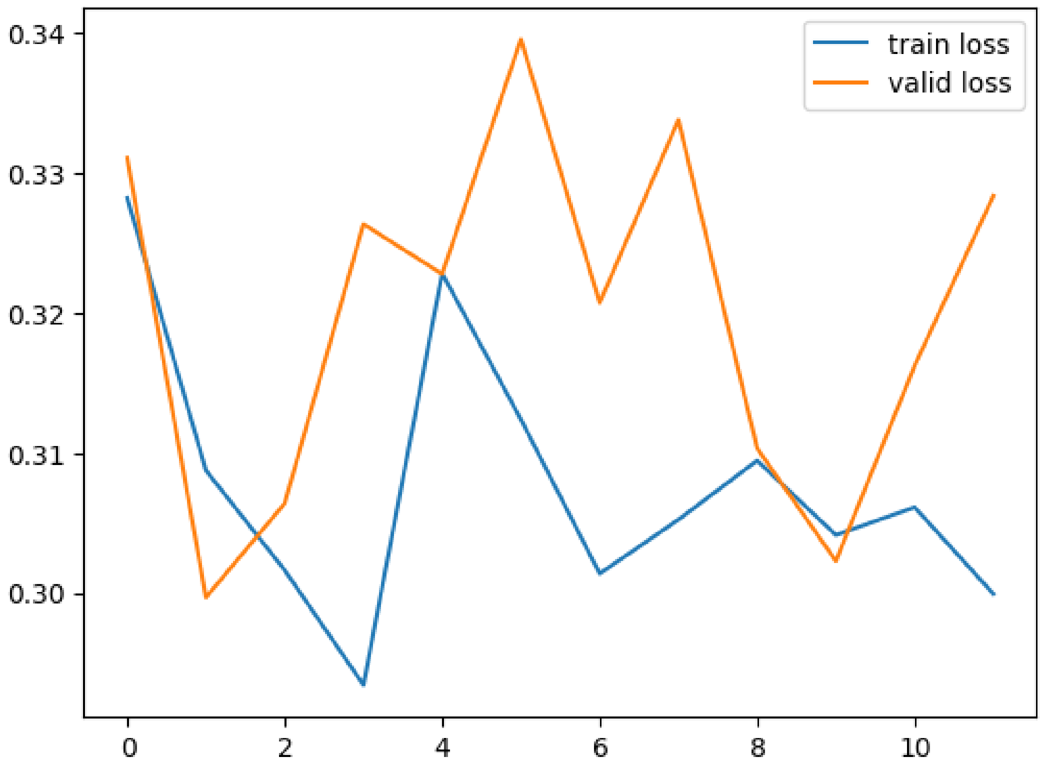
```
CV round 12
EARLY STOPPING @ epoch 10
min train loss: 0.30183328326904413
min valid loss: 0.29331014501421077
```



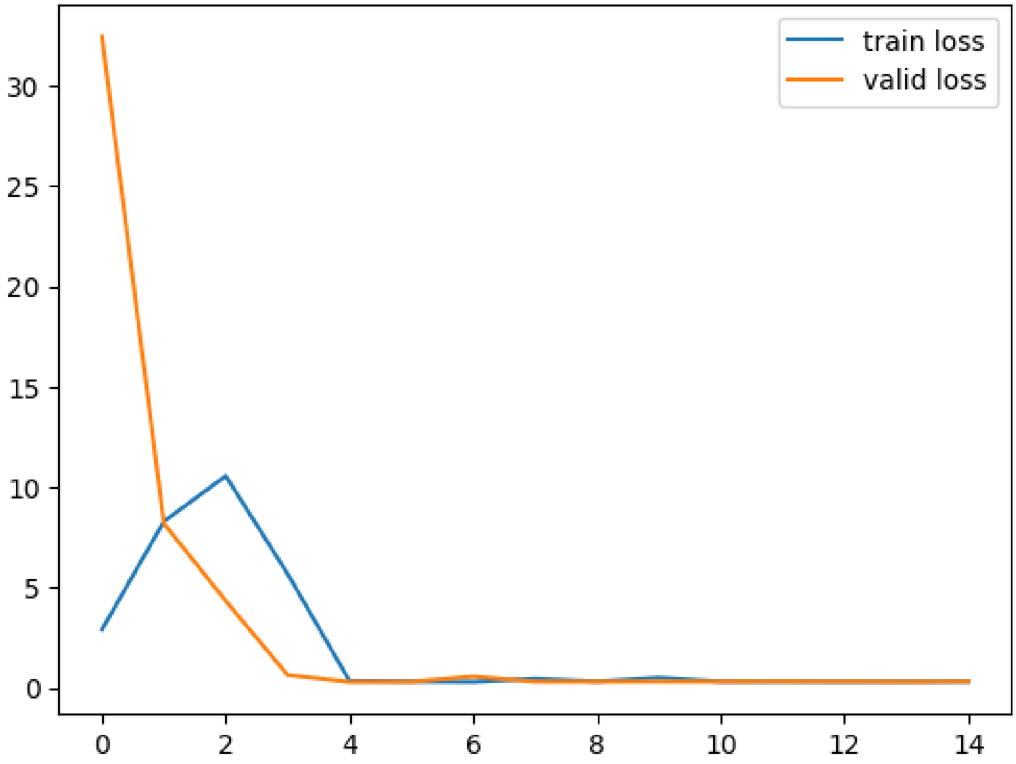
```
CV round 13
EARLY STOPPING @ epoch 16
min train loss: 0.29976136293826683
min valid loss: 0.2752775973395297
```



```
CV round 14
EARLY STOPPING @ epoch 11
min train loss: 0.2934210763849092
min valid loss: 0.29968001811127915
```



```
CV round 15
EARLY STOPPING @ epoch 14
min train loss: 0.2943132966192383
min valid loss: 0.2893774815295872
```



```
best model is: CV=9.pth with 0.11320586855474271
```

[18]:

```
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=32))
print(f"testing loss: {test_loss}")
```

```
testing loss: 0.35837633398018387
```