CS599 (Deep Learning)

Homework – 13

1. Python Code:

```
import torch
import pandas as pd
import matplotlib
#matplotlib.use("agg")
import numpy as np
import plotnine as p9
import math
import pdb
```

```
from sklearn.model_selection import KFold, GridSearchCV, ParameterGrid
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegressionCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from collections import Counter
```

```
data_set_dict = {"zip": ("zip.test.gz", 0)}
data_dict = {}
```

```
for data_name, (file_name, label_col_num) in data_set_dict.items():
    data_df = pd.read_csv(file_name, sep = " ", header = None)
    data_label_vec = data_df.iloc[:, label_col_num]
    is_01 = data_label_vec.isin([0, 1])
    data_01_df = data_df.loc[is_01, :]
    is_label_col = data_df.columns == label_col_num
    data_features = data_01_df.iloc[:, ~is_label_col]
    data_labels = data_01_df.iloc[:, is_label_col]
    data_dict[data_name] = (data_features, data_labels)
```

```
zip_df = pd.read_csv("zip.test.gz", sep = " ", header = None)
zip_label_col_num = 0
zip_label_vec = zip_df.iloc[:, zip_label_col_num]
is_71 = zip_label_vec.isin([7,1])
zip_71_df = zip_df.loc[is_71, :]
is_label_col = zip_71_df.columns == zip_label_col_num
zip_features = zip_71_df.iloc[:, ~is_label_col]
zip_labels = zip_71_df.iloc[:, is_label_col]
zip_labels = zip_labels.replace(7, 0)
data_dict["zip_71"] = (zip_features, zip_labels)
```

```
{data_name:X.shape for data_name, (X,y) in data_dict.items()}
```

```
{'zip': (623, 256), 'zip_71': (411, 256)}
```

```
class TorchModel(torch.nn.Module):
    def __init__(self, units_per_layer):
        super(TorchModel, self).__init__()
        self.conv = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=(3, 3)),
            torch.nn.Conv2d(32, 64, kernel_size=(3, 3)),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size = (3, 3)),
            torch.nn.Flatten(start_dim = 1))
        self.lin_seq = torch.nn.Sequential(
            torch.nn.Linear(1, 128),
            torch.nn.ReLU(),
            torch.nn.Linear(128, 2))
        self.fc = torch.nn.Sequential(
            self.conv,
            self.lin_seq)
```

```python
    def forward(self, features):
        _, flattened_size = self.conv(features[:1]).shape
        self.lin_seq[0] = torch.nn.Linear(flattened_size, 128)
        x = self.fc(features)
        return x
```

```python
class TorchModel(torch.nn.Module):
    def __init__(self, units_per_layer):
        super(TorchModel, self).__init__()
        self.conv = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=(3, 3)),
            torch.nn.Conv2d(32, 64, kernel_size=(3, 3)),
            torch.nn.ReLU(),
            torch.nn.Flatten(start_dim = 1))
        self.lin_seq = torch.nn.Sequential(
            torch.nn.Linear(1, 128),
            torch.nn.ReLU(),
            torch.nn.Linear(128, 2))
        self.fc = torch.nn.Sequential(
            self.conv,
            self.lin_seq)

    def forward(self, features):
        _, flattened_size = self.conv(features[:1]).shape
        self.lin_seq[0] = torch.nn.Linear(flattened_size, 128)
        x = self.fc(features)
        return x
```

```python
class CSV(torch.utils.data.Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels
    def __getitem__(self, item):
        return self.features[item,:], self.labels[item]
    def __len__(self):
        return len(self.labels)
```

```python
class TorchLearner:
    def __init__(
            self, units_per_layer, opt_name, opt_params,
            batch_size=20, max_epochs=100):
        self.max_epochs = max_epochs
        self.batch_size=batch_size
        self.model = TorchModel(units_per_layer)
        self.loss_fun = torch.nn.CrossEntropyLoss()
        self.initial_step_size = 0.1
        self.end_step_size = 0.001
        self.last_step_number = 50
        self.opt_name = opt_name
        self.opt_params = opt_params
    def get_step_size(self, iteration):
        if iteration > self.last_step_number:
            return self.end_step_size
        prop_to_last_step = iteration/self.last_step_number
        return (1 - prop_to_last_step) * self.initial_step_size +\
            prop_to_last_step * self.end_step_size
    def fit(self, split_data_dict):
        ds = CSV(
            split_data_dict["subtrain"]["X"],
            split_data_dict["subtrain"]["y"])
        dl = torch.utils.data.DataLoader(
            ds, batch_size=self.batch_size, shuffle=True)
        train_df_list = []

        for epoch_number in range(self.max_epochs):
            step_size = self.get_step_size(epoch_number)
            if self.opt_name == "SGD":
                self.optimizer = torch.optim.SGD(self.model.parameters(), **self.opt_params, lr = step_size)
            elif self.opt_name == "Adam":
                self.optimizer = torch.optim.Adam(self.model.parameters(), **self.opt_params, lr = step_size)
            for batch_features, batch_labels in dl:
                nrow, ncol = batch_features.shape
                self.optimizer.zero_grad()
                loss_value = self.loss_fun(
                    self.model(batch_features.view(nrow, 1, 16, 16)), batch_labels)
                loss_value.backward()
                self.optimizer.step()

            for set_name, set_data in split_data_dict.items():
                n_row, n_col = set_data["X"].shape
                pred_vec = self.model(set_data["X"].reshape(n_row, 1, 16, 16))
                set_loss_value = self.loss_fun(pred_vec, set_data["y"])
                train_df_list.append(pd.DataFrame({
                    "set_name":[set_name],
                    "loss":float(set_loss_value),
                    "epoch":[epoch_number]
                }))
        self.train_df = pd.concat(train_df_list)
    def decision_function(self, test_features):
        with torch.no_grad():
            row, col = test_features.shape
            pred_vec = self.model(test_features.reshape(row, 1, 16, 16))
        return pred_vec

    def predict(self, test_features):
        pred_scores = self.decision_function(test_features)
        _, predicted = torch.max(pred_scores, 1)
        return predicted
```

```python
class TorchLearnerCV:
    def __init__(self, n_folds, units_per_layer, opt_name = 'SGD', opt_params = {'momentum': 0.5}):
        self.units_per_layer = units_per_layer
        self.opt_name = opt_name
        self.opt_params = opt_params
        self.n_folds = n_folds
    def fit(self, train_features, train_labels):
        train_nrow, train_ncol = train_features.shape
        times_to_repeat=int(math.ceil(train_nrow/self.n_folds))
        fold_id_vec = np.tile(torch.arange(self.n_folds), times_to_repeat)[:train_nrow]
        np.random.shuffle(fold_id_vec)
        cv_data_list = []
        for validation_fold in range(self.n_folds):
            is_split = {
                "subtrain":fold_id_vec != validation_fold,
                "validation":fold_id_vec == validation_fold
                }
            split_data_dict = {}
            for set_name, is_set in is_split.items():
                set_y = train_labels[is_set]
                split_data_dict[set_name] = {
                    "X":train_features[is_set,:],
                    "y":set_y}
            learner = TorchLearner(self.units_per_layer, self.opt_name, self.opt_params)
            learner.fit(split_data_dict)
            cv_data_list.append(learner.train_df)
        self.cv_data = pd.concat(cv_data_list)
        self.train_df = self.cv_data.groupby(["set_name","epoch"]).mean().reset_index()
        #print(self.train_df)
        valid_df = self.train_df.query("set_name=='validation'")
        #print(valid_df)
        best_epochs = valid_df["loss"].argmin()
        self.min_df = valid_df.query("epoch==%s"%(best_epochs))
        print("Best Epoch: ", best_epochs)
        #pdb.set_trace()
        self.final_learner = TorchLearner(self.units_per_layer, self.opt_name, self.opt_params, max_epochs=(best_epochs + 1))
        self.final_learner.fit({"subtrain":{"X":train_features,"y":train_labels}})
        return self.cv_data
    def predict(self, test_features):
        return self.final_learner.predict(test_features)
```

```python
class MyCV:
    def __init__(self, estimator, param_grid, cv):
        """estimator: learner instance
        pram_grid: list of dictionaries
        cv: number of folds"""
        self.cv = cv
        self.param_grid = param_grid
        self.estimator = estimator
    def fit_one(self, param_dict, X, y):
        """Run self.estimator.fit on one parameter combination"""
        for param_name, param_value in param_dict.items():
            #print(f"param_name = {param_name}, param_value = {param_value}")
            setattr(self.estimator, param_name, param_value)
        self.estimator.fit(X, y)
    def fit(self, X, y):
        """cross-validation for selecting the best dictionary is param_grid"""
        validation_df_list = []
        train_nrow, train_ncol = X.shape
        times_to_repeat = int(math.ceil(train_nrow/self.cv))
        fold_id_vec = np.tile(np.arange(self.cv), times_to_repeat)[:train_nrow]
        np.random.shuffle(fold_id_vec)
        for validation_fold in range(self.cv):
            is_split = {
                "subtrain": fold_id_vec != validation_fold,
                "validation": fold_id_vec == validation_fold
            }
            split_data_dict = {}
            for set_name, is_set in is_split.items():
                split_data_dict[set_name] = (
                X[is_set],
                y[is_set])
            for param_number, param_dict in enumerate(self.param_grid):
                self.fit_one(param_dict, *split_data_dict["subtrain"])
                X_valid, y_valid = split_data_dict["validation"]
                pred_valid = self.estimator.predict(X_valid)
                #pdb.set_trace()
                is_correct = pred_valid == y_valid
                #self.estimator.fit(*split_data_dict["validation"])
                valid_loss = self.estimator.train_df.query("set_name=='validation'")["loss"].mean()
                subtrain_loss =  self.estimator.train_df.query("set_name=='subtrain'")["loss"].mean()
                validation_row1 = pd.DataFrame({
                "set_name": "subtrain",
                "validation_fold": validation_fold,
                "accuracy_percent": float(is_correct.float().mean()),
                "param_number": [param_number],
                "loss": float(subtrain_loss)
                }, index = [0])
                validation_row2 = pd.DataFrame({
                "set_name": "validation",
                "validation_fold": validation_fold,
                "accuracy_percent": float(is_correct.float().mean()),
                "param_number": [param_number],
                "loss": float(valid_loss)
                }, index = [0])
                validation_df_list.append(validation_row1)
                validation_df_list.append(validation_row2)
        self.validation_df = pd.concat(validation_df_list)
        self.mean_valid_loss = self.validation_df.groupby("param_number")["loss"].mean().reset_index()
        self.train_df = self.validation_df.groupby(["set_name", "loss"]).mean().reset_index()
        best_index = self.mean_valid_loss["loss"].argmin()
        #pdb.set_trace()
        valid_df = self.train_df.query("set_name == 'validation'")
        self.min_df = valid_df.query("param_number==%s"%(best_index))
        self.best_param_dict = self.param_grid[best_index]
        self.fit_one(self.best_param_dict, X, y)

    def predict(self, X):
        return self.estimator.predict(X)
```

```python
accuracy_data_frames = []
loss_data_dict = {}
min_df_dict = {}
for data_name, (data_features, data_labels) in data_dict.items():
    kf = KFold(n_splits=3, shuffle=True, random_state=3)
    enum_obj = enumerate(kf.split(data_features))
    for fold_num, index_tup in enum_obj:
        zip_obj = zip(["train", "test"], index_tup)
        split_data = {}
        for set_name, set_indices in zip_obj:
            split_data[set_name] = (torch.from_numpy(data_features.iloc[set_indices, :].to_numpy()).float(),
                                    torch.from_numpy(np.ravel(data_labels.iloc[set_indices])).flatten())
        x = {data_name:X.shape for data_name, (X,y) in split_data.items()}
        print(f"{data_name}: ", x)
        train_features, train_labels = split_data["train"]
        nrow, ncol = train_features.shape
        print(f"{data_name}: ", nrow, ncol)
        test_features, test_labels = split_data["test"]

        #kneighbors
        knn = KNeighborsClassifier()
        hp_parameters = {"n_neighbors": list(range(1, 21))}
        grid = GridSearchCV(knn, hp_parameters, cv=3)
        grid.fit(train_features, train_labels)
        best_n_neighbors = grid.best_params_['n_neighbors']
        print("Best N-Neighbors = ", best_n_neighbors)
        knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
        knn.fit(train_features, train_labels)
        knn_pred = knn.predict(test_features)

        #linear model
        pipe = make_pipeline(StandardScaler(), LogisticRegressionCV(cv=3, max_iter=2000))
        pipe.fit(train_features, train_labels)
        lr_pred = pipe.predict(test_features)

        #Featureless
        y_train_series = pd.Series(train_labels)
        # create a featureless baseline
        most_frequent_label = y_train_series.value_counts().idxmax()
        print("Most Frequent Label = ", most_frequent_label)
        featureless_pred = np.repeat(most_frequent_label, len(test_features))

        param_grid = []
        for momentum in 0.1, 0.5:
            param_grid.append({
                "opt_name":"SGD",
                "opt_params":{"momentum":momentum}
```

```
            })
        for beta1 in 0.85, 0.9, 0.95:
            for beta2 in 0.99, 0.999, 0.9999:
                param_grid.append({
                    "opt_name":"Adam",
                    "opt_params":{"betas":(beta1, beta2)}
                })

        #TorchLearnerCV + Deep
        conv_learner = MyCV(
            estimator = TorchLearnerCV(3, [ncol, 64, 32, 128, 1]),
            param_grid = param_grid,
            cv = 2)
        conv_learner.fit(train_features, train_labels)
        print(f"Best param_dict: {conv_learner.best_param_dict}")
        cl_pred = conv_learner.predict(test_features)

        min_df_dict[data_name] = {'min_df': conv_learner.estimator.min_df}

        loss_data_dict[data_name] = {'conv_learner': conv_learner.estimator.train_df}

        # store predict data in dict
        pred_dict = {'KNeighborsClassifier + GridSearchCV': knn_pred,
                     'LogisticRegressionCV': lr_pred,
                     'ConvolutionalMLP': cl_pred,
                     'featureless': featureless_pred}
        test_accuracy = {}
        for algorithm, predictions in pred_dict.items():
            accuracy = accuracy_score(test_labels, predictions)
            test_accuracy[algorithm] = accuracy

        for algorithm, accuracy in test_accuracy.items():
            print(f"{algorithm} Test Accuracy: {accuracy * 100}")
            accuracy_df = pd.DataFrame({
                "data_set": [data_name],
                "fold_id": [fold_num],
                "algorithm": [algorithm],
                "accuracy": [test_accuracy[algorithm]]})
            accuracy_data_frames.append(accuracy_df)
        print(f"****************************End of {data_name}({fold_num})*****************************")
```

```
total_accuracy_df = pd.concat(accuracy_data_frames, ignore_index = True)
print(total_accuracy_df)
```

```
gg = p9.ggplot(total_accuracy_df, p9.aes(x ='accuracy', y = 'algorithm'))+\
 p9.facet_grid('.~data_set') + p9.geom_point()
```

```
print(gg)
```

```
gg1 = p9.ggplot() +\
     p9.geom_line(
     p9.aes(
     x = "epoch",
     y= "loss",
     color = "set_name"
     ),
     data = loss_data_dict["zip"]['conv_learner']) +\
     p9.geom_point(
     p9.aes(
     x = "epoch",
     y = "loss",
     color = "set_name"
     ),
     data = min_df_dict["zip"]['min_df']) +\
     p9.ggtitle("Subtrain/Validation Loss vs Epochs(Zip - Data) with Max Pooling")
```

```
print(gg1)
```

```
gg2 = p9.ggplot() +\
     p9.geom_line(
     p9.aes(
     x = "epoch",
     y= "loss",
     color = "set_name"
     ),
     data = loss_data_dict["zip_71"]["conv_learner"]) +\
     p9.geom_point(
     p9.aes(
     x = "epoch",
     y = "loss",
     color = "set_name"
     ),
     data = min_df_dict["zip_71"]["min_df"]) +\
     p9.ggtitle("Subtrain/Validation Loss vs Epochs(zip_71 - Data) with Max Pooling")
```

```
print(gg2)
```

2. Output:

Accuracy with Max Pooling:

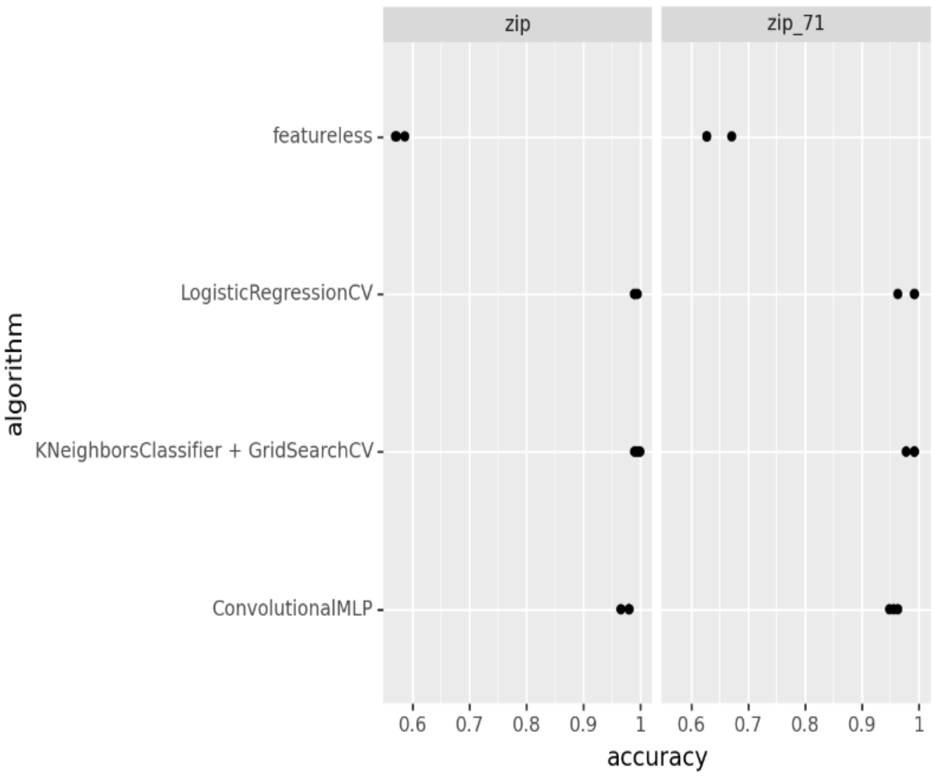
```
zip:  {'train': torch.Size([415, 256]), 'test': torch.Size([208, 256])}
zip:  415 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  88
Best Epoch:  75
Best Epoch:  87
Best Epoch:  43
Best Epoch:  92
Best Epoch:  79
Best Epoch:  89
Best Epoch:  96
Best Epoch:  31
Best Epoch:  31
Best Epoch:  27
Best Epoch:  82
Best Epoch:  43
Best Epoch:  75
Best Epoch:  37
Best Epoch:  75
Best Epoch:  63
Best Epoch:  97
Best Epoch:  47
Best Epoch:  90
Best Epoch:  86
Best Epoch:  22
Best Epoch:  43
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 100.0
LogisticRegressionCV Test Accuracy: 99.51923076923077
ConvolutionalMLP Test Accuracy: 96.63461538461539
featureless Test Accuracy: 58.65384615384615
****************************End of zip(0)****************************
```

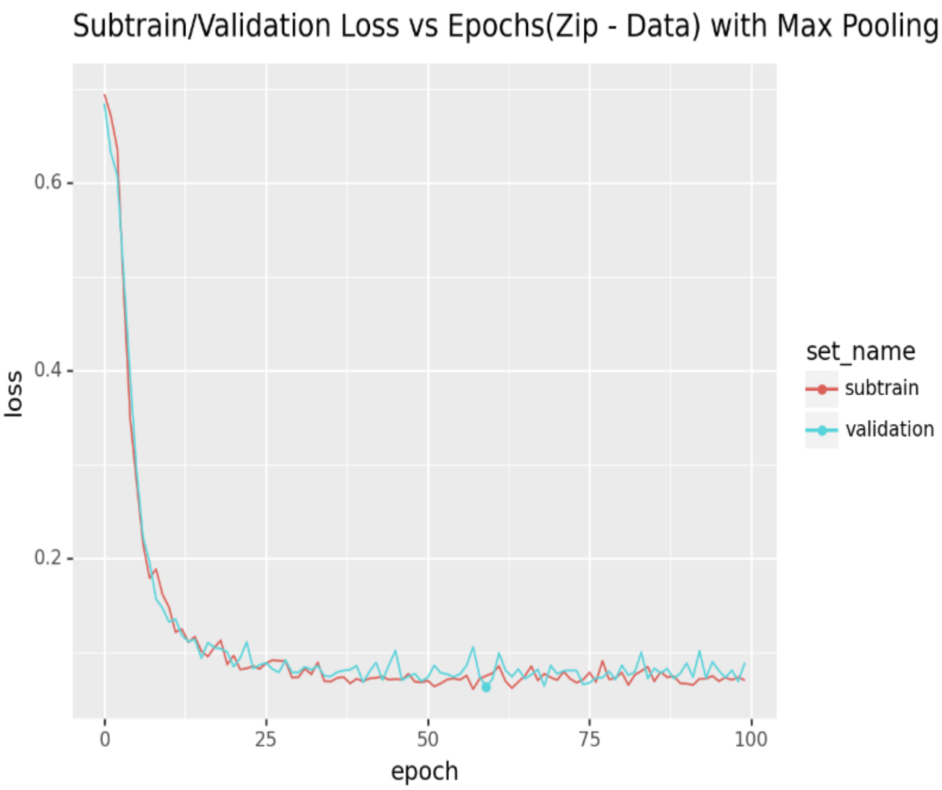
```
zip:  {'train': torch.Size([415, 256]), 'test': torch.Size([208, 256])}
zip:  415 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  80
Best Epoch:  72
Best Epoch:  21
Best Epoch:  73
Best Epoch:  15
Best Epoch:  66
Best Epoch:  87
Best Epoch:  24
Best Epoch:  35
Best Epoch:  94
Best Epoch:  32
Best Epoch:  49
Best Epoch:  77
Best Epoch:  50
Best Epoch:  54
Best Epoch:  29
Best Epoch:  44
Best Epoch:  60
Best Epoch:  55
Best Epoch:  29
Best Epoch:  47
Best Epoch:  31
Best Epoch:  26
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 99.51923076923077
LogisticRegressionCV Test Accuracy: 99.03846153846155
ConvolutionalMLP Test Accuracy: 98.07692307692307
featureless Test Accuracy: 57.21153846153846
****************************End of zip(1)****************************
```

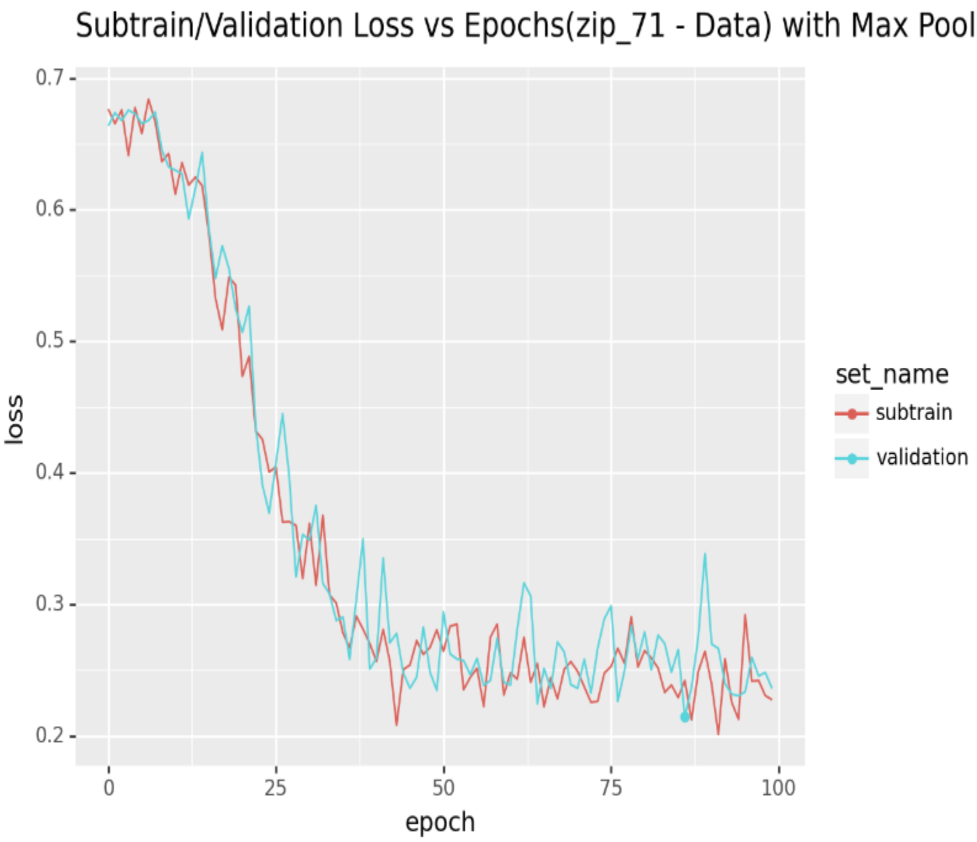
```
zip:  {'train': torch.Size([416, 256]), 'test': torch.Size([207, 256])}
zip:  416 256
Best N-Neighbors =  4
Most Frequent Label =  0
Best Epoch:  95
Best Epoch:  93
Best Epoch:  27
Best Epoch:  20
Best Epoch:  37
Best Epoch:  50
Best Epoch:  20
Best Epoch:  48
Best Epoch:  19
Best Epoch:  21
Best Epoch:  28
Best Epoch:  75
Best Epoch:  91
Best Epoch:  61
Best Epoch:  80
Best Epoch:  40
Best Epoch:  94
Best Epoch:  73
Best Epoch:  26
Best Epoch:  36
Best Epoch:  39
Best Epoch:  39
Best Epoch:  59
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 99.03381642512076
LogisticRegressionCV Test Accuracy: 99.03381642512076
ConvolutionalMLP Test Accuracy: 98.06763285024155
featureless Test Accuracy: 57.00483091787439
****************************End of zip(2)****************************
zip_71:  {'train': torch.Size([274, 256]), 'test': torch.Size([137, 256])}
zip_71:  274 256
Best N-Neighbors =  1
Most Frequent Label =  1
Best Epoch:  61
Best Epoch:  89
Best Epoch:  99
Best Epoch:  89
Best Epoch:  41
Best Epoch:  34
Best Epoch:  52
Best Epoch:  38
Best Epoch:  46
Best Epoch:  98
Best Epoch:  91
Best Epoch:  63
Best Epoch:  68
Best Epoch:  63
Best Epoch:  84
Best Epoch:  54
Best Epoch:  65
Best Epoch:  67
Best Epoch:  63
Best Epoch:  48
Best Epoch:  39
Best Epoch:  83
Best Epoch:  71
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 99.27007299270073
LogisticRegressionCV Test Accuracy: 99.27007299270073
ConvolutionalMLP Test Accuracy: 96.35036496350365
featureless Test Accuracy: 62.77372262773723
*****************************End of zip_71(0)******************************
```

```
zip_71:  {'train': torch.Size([274, 256]), 'test': torch.Size([137, 256])}
zip_71:  274 256
Best N-Neighbors =  1
Most Frequent Label =  1
Best Epoch:  87
Best Epoch:  97
Best Epoch:  82
Best Epoch:  46
Best Epoch:  45
Best Epoch:  81
Best Epoch:  98
Best Epoch:  42
Best Epoch:  50
Best Epoch:  61
Best Epoch:  30
Best Epoch:  49
Best Epoch:  89
Best Epoch:  33
Best Epoch:  75
Best Epoch:  68
Best Epoch:  49
Best Epoch:  57
Best Epoch:  69
Best Epoch:  71
Best Epoch:  96
Best Epoch:  96
Best Epoch:  86
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 97.8102189781022
LogisticRegressionCV Test Accuracy: 96.35036496350365
ConvolutionalMLP Test Accuracy: 95.62043795620438
featureless Test Accuracy: 62.77372262773723
****************************End of zip_71(1)****************************
zip_71:  {'train': torch.Size([274, 256]), 'test': torch.Size([137, 256])}
zip_71:  274 256
Best N-Neighbors =  2
Most Frequent Label =  1
Best Epoch:  95
Best Epoch:  97
Best Epoch:  91
Best Epoch:  32
Best Epoch:  84
Best Epoch:  56
Best Epoch:  44
Best Epoch:  33
Best Epoch:  89
Best Epoch:  62
Best Epoch:  80
Best Epoch:  43
Best Epoch:  94
Best Epoch:  83
Best Epoch:  48
Best Epoch:  44
Best Epoch:  75
Best Epoch:  90
Best Epoch:  95
Best Epoch:  79
Best Epoch:  45
Best Epoch:  91
Best Epoch:  86
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 99.27007299270073
LogisticRegressionCV Test Accuracy: 99.27007299270073
ConvolutionalMLP Test Accuracy: 94.8905109489051
featureless Test Accuracy: 67.15328467153284
****************************End of zip_71(2)****************************
```

	data_set	fold_id	algorithm	accuracy
0	zip	0	KNeighborsClassifier + GridSearchCV	1.000000
1	zip	0	LogisticRegressionCV	0.995192
2	zip	0	ConvolutionalMLP	0.966346
3	zip	0	featureless	0.586538
4	zip	1	KNeighborsClassifier + GridSearchCV	0.995192
5	zip	1	LogisticRegressionCV	0.990385
6	zip	1	ConvolutionalMLP	0.980769
7	zip	1	featureless	0.572115
8	zip	2	KNeighborsClassifier + GridSearchCV	0.990338
9	zip	2	LogisticRegressionCV	0.990338
10	zip	2	ConvolutionalMLP	0.980676
11	zip	2	featureless	0.570048
12	zip_71	0	KNeighborsClassifier + GridSearchCV	0.992701
13	zip_71	0	LogisticRegressionCV	0.992701
14	zip_71	0	ConvolutionalMLP	0.963504
15	zip_71	0	featureless	0.627737
16	zip_71	1	KNeighborsClassifier + GridSearchCV	0.978102
17	zip_71	1	LogisticRegressionCV	0.963504
18	zip_71	1	ConvolutionalMLP	0.956204
19	zip_71	1	featureless	0.627737
20	zip_71	2	KNeighborsClassifier + GridSearchCV	0.992701
21	zip_71	2	LogisticRegressionCV	0.992701
22	zip_71	2	ConvolutionalMLP	0.948905
23	zip_71	2	featureless	0.671533




Subtrain/Validation Loss vs Epochs(Zip - Data) with Max Pooling
loss
0.6
0.4
0.2
0
25
50
75
100
epoch
set_name
subtrain
validation


Subtrain/Validation Loss vs Epochs(zip_71 - Data) with Max Pool
loss
0.7
0.6
0.5
0.4
0.3
0.2
0
25
50
75
100
epoch
set_name
subtrain
validation

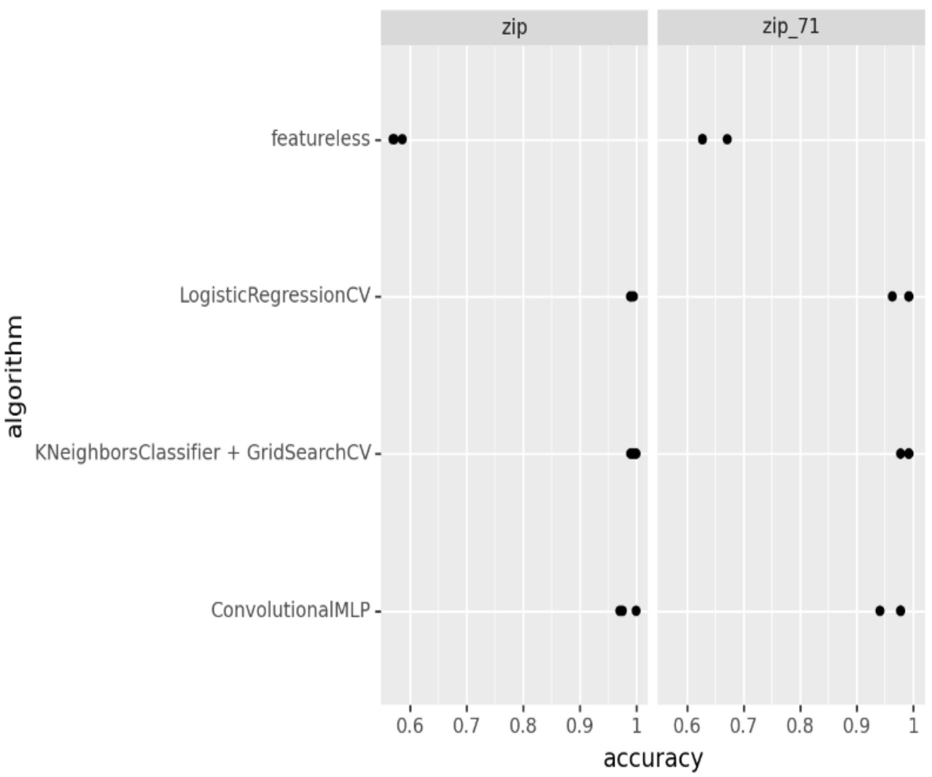
Accuracy without Max Pooling:

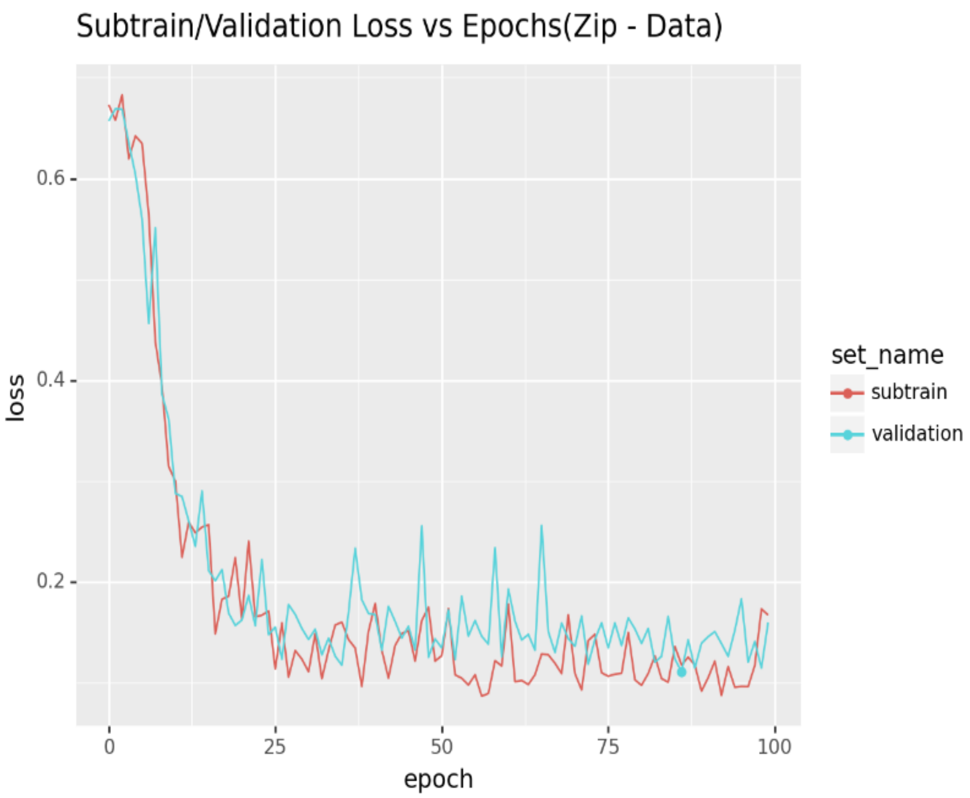
```
zip:  {'train': torch.Size([415, 256]), 'test': torch.Size([208, 256])}
zip:  415 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  63
Best Epoch:  50
Best Epoch:  31
Best Epoch:  53
Best Epoch:  98
Best Epoch:  48
Best Epoch:  27
Best Epoch:  60
Best Epoch:  37
Best Epoch:  96
Best Epoch:  35
Best Epoch:  52
Best Epoch:  61
Best Epoch:  51
Best Epoch:  82
Best Epoch:  25
Best Epoch:  31
Best Epoch:  95
Best Epoch:  68
Best Epoch:  94
Best Epoch:  27
Best Epoch:  66
Best Epoch:  91
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 100.0
LogisticRegressionCV Test Accuracy: 99.51923076923077
ConvolutionalMLP Test Accuracy: 100.0
featureless Test Accuracy: 58.65384615384615
****************************End of zip(0)*****************************
zip:  {'train': torch.Size([415, 256]), 'test': torch.Size([208, 256])}
zip:  415 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  94
Best Epoch:  67
Best Epoch:  38
Best Epoch:  75
Best Epoch:  81
Best Epoch:  38
Best Epoch:  99
Best Epoch:  97
Best Epoch:  38
Best Epoch:  21
Best Epoch:  31
Best Epoch:  81
Best Epoch:  73
Best Epoch:  42
Best Epoch:  44
Best Epoch:  37
Best Epoch:  52
Best Epoch:  54
Best Epoch:  89
Best Epoch:  96
Best Epoch:  65
Best Epoch:  60
Best Epoch:  48
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 99.51923076923077
LogisticRegressionCV Test Accuracy: 99.03846153846155
ConvolutionalMLP Test Accuracy: 97.11538461538461
featureless Test Accuracy: 57.21153846153846
****************************End of zip(1)*****************************
```

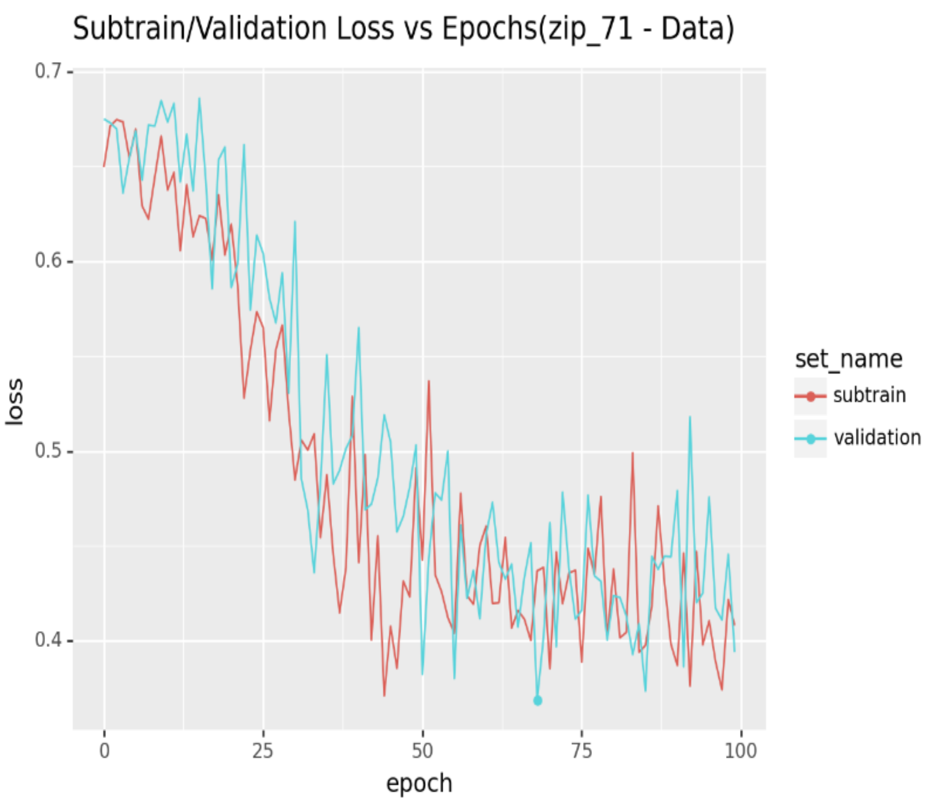
```
zip:  {'train': torch.Size([416, 256]), 'test': torch.Size([207, 256])}
zip:  416 256
Best N-Neighbors =  4
Most Frequent Label =  0
Best Epoch:  75
Best Epoch:  69
Best Epoch:  58
Best Epoch:  34
Best Epoch:  27
Best Epoch:  42
Best Epoch:  21
Best Epoch:  79
Best Epoch:  59
Best Epoch:  73
Best Epoch:  31
Best Epoch:  63
Best Epoch:  58
Best Epoch:  79
Best Epoch:  68
Best Epoch:  28
Best Epoch:  39
Best Epoch:  31
Best Epoch:  29
Best Epoch:  54
Best Epoch:  92
Best Epoch:  18
Best Epoch:  86
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 99.03381642512076
LogisticRegressionCV Test Accuracy: 99.03381642512076
ConvolutionalMLP Test Accuracy: 97.58454106280193
featureless Test Accuracy: 57.00483091787439
****************************End of zip(2)****************************
zip_71:  {'train': torch.Size([274, 256]), 'test': torch.Size([137, 256])}
zip_71:  274 256
Best N-Neighbors =  1
Most Frequent Label =  1
Best Epoch:  91
Best Epoch:  79
Best Epoch:  64
Best Epoch:  40
Best Epoch:  95
Best Epoch:  60
Best Epoch:  37
Best Epoch:  85
Best Epoch:  42
Best Epoch:  91
Best Epoch:  71
Best Epoch:  90
Best Epoch:  75
Best Epoch:  73
Best Epoch:  57
Best Epoch:  31
Best Epoch:  65
Best Epoch:  49
Best Epoch:  84
Best Epoch:  34
Best Epoch:  87
Best Epoch:  50
Best Epoch:  91
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 99.27007299270073
LogisticRegressionCV Test Accuracy: 99.27007299270073
ConvolutionalMLP Test Accuracy: 97.8102189781022
featureless Test Accuracy: 62.77372262773723
****************************End of zip_71(0)****************************
```

```
zip_71:  {'train': torch.Size([274, 256]), 'test': torch.Size([137, 256])}
zip_71:  274 256
Best N-Neighbors =  1
Most Frequent Label =  1
Best Epoch:  90
Best Epoch:  94
Best Epoch:  43
Best Epoch:  96
Best Epoch:  60
Best Epoch:  23
Best Epoch:  41
Best Epoch:  85
Best Epoch:  75
Best Epoch:  86
Best Epoch:  56
Best Epoch:  87
Best Epoch:  82
Best Epoch:  93
Best Epoch:  93
Best Epoch:  66
Best Epoch:  95
Best Epoch:  94
Best Epoch:  51
Best Epoch:  88
Best Epoch:  98
Best Epoch:  82
Best Epoch:  98
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 97.8102189781022
LogisticRegressionCV Test Accuracy: 96.35036496350365
ConvolutionalMLP Test Accuracy: 94.16058394160584
featureless Test Accuracy: 62.77372262773723
****************************End of zip_71(1)*****************************
 zip_71:  {'train': torch.Size([274, 256]), 'test': torch.Size([137, 256])}
 zip_71:  274 256
 Best N-Neighbors =  2
 Most Frequent Label =  1
 Best Epoch:  90
 Best Epoch:  43
 Best Epoch:  98
 Best Epoch:  52
 Best Epoch:  75
 Best Epoch:  33
 Best Epoch:  53
 Best Epoch:  92
 Best Epoch:  71
 Best Epoch:  65
 Best Epoch:  59
 Best Epoch:  95
 Best Epoch:  29
 Best Epoch:  99
 Best Epoch:  52
 Best Epoch:  53
 Best Epoch:  41
 Best Epoch:  46
 Best Epoch:  89
 Best Epoch:  92
 Best Epoch:  78
 Best Epoch:  94
 Best Epoch:  68
 Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
 KNeighborsClassifier + GridSearchCV Test Accuracy: 99.27007299270073
 LogisticRegressionCV Test Accuracy: 99.27007299270073
 ConvolutionalMLP Test Accuracy: 97.8102189781022
 featureless Test Accuracy: 67.15328467153284
 ****************************End of zip_71(2)*****************************
```

	data_set	fold_id	algorithm	accuracy
0	zip	0	KNeighborsClassifier + GridSearchCV	1.000000
1	zip	0	LogisticRegressionCV	0.995192
2	zip	0	ConvolutionalMLP	1.000000
3	zip	0	featureless	0.586538
4	zip	1	KNeighborsClassifier + GridSearchCV	0.995192
5	zip	1	LogisticRegressionCV	0.990385
6	zip	1	ConvolutionalMLP	0.971154
7	zip	1	featureless	0.572115
8	zip	2	KNeighborsClassifier + GridSearchCV	0.990338
9	zip	2	LogisticRegressionCV	0.990338
10	zip	2	ConvolutionalMLP	0.975845
11	zip	2	featureless	0.570048
12	zip_71	0	KNeighborsClassifier + GridSearchCV	0.992701
13	zip_71	0	LogisticRegressionCV	0.992701
14	zip_71	0	ConvolutionalMLP	0.978102
15	zip_71	0	featureless	0.627737
16	zip_71	1	KNeighborsClassifier + GridSearchCV	0.978102
17	zip_71	1	LogisticRegressionCV	0.963504
18	zip_71	1	ConvolutionalMLP	0.941606
19	zip_71	1	featureless	0.627737
20	zip_71	2	KNeighborsClassifier + GridSearchCV	0.992701
21	zip_71	2	LogisticRegressionCV	0.992701
22	zip_71	2	ConvolutionalMLP	0.978102
23	zip_71	2	featureless	0.671533




Subtrain/Validation Loss vs Epochs(Zip - Data)
loss
0.6
0.4
0.2
0
25
50
75
100
epoch
set_name
subtrain
validation


Subtrain/Validation Loss vs Epochs(zip_71 - Data)
loss
0.7
0.6
0.5
0.4
0
25
50
75
100
epoch
set_name
subtrain
validation