```python
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, precision_score, recall_score
from bootstrap import bootstrap

def run_ann(X, y, n_trials=5):
    """
    Runs artifical neural network for a dataset
    5 trials of ANN with 5-fold cross validation and a gridsearch over hyperparameters:
        hidden_layer_sizes, momentum
    and computes mean accuracy over metrics
        accuracy, f1, roc, precision, recall

    parameters
    ----------
    X: feature vector
    y: target vector
    n_trials: number of trials to run

    returns
    --------
    train_metrics: average of each metric on training set across 5 trials
    test_metrics: average of each metric on test set across 5 trials

    """

    # hyperparameters
    layers = [[50],[50,50],[50,50,50,50,50],[50,50,50,50,50,50,50,50],[100],[100,100],[100,100,100,100,100],
              [100,100,100,100,100,100,100,100],[200],[200,200],[200,200,200,200,200],
              [200,200,200,200,200,200,200,200]]
    momentums = [0, 0.2, 0.5, 0.9]
    params = {'hidden_layer_sizes':layers, 'momentum':momentums}

    # metric evaluation scores
    scores = ['accuracy', 'f1', 'roc_auc', 'precision', 'recall']

    # to hold calculated metric performances
    train_metrics = []
    test_metrics = []

    for trial in range(n_trials):

        # initialize model for cross validation grid search
        ANN = MLPClassifier(solver='sgd', learning_rate='adaptive')
        ANN = MLPClassifier(solver='sgd', learning_rate='adaptive')
        GS = GridSearchCV(ANN, params, scoring=scores, refit=False)

        # bootstrap training and testing sets
        X_train, X_test, y_train, y_test = bootstrap(X,y)
        GS.fit(X_train, y_train)

        # collect results
        res = GS.cv_results_

        # collect/store hyperparameters for visualization
        hyperp = pd.DataFrame(res['params'])
        hyperp['acc'] = res['mean_test_accuracy']
        hyperp['f1'] = res['mean_test_f1']
        hyperp['roc'] = res['mean_test_roc_auc']
        hidden_layers = []
        # rename hidden layer sizes column from list to str
        for i,row in hyperp.iterrows():
            if 50 in row['hidden_layer_sizes']:

 hidden_layers.append('{}x50'.format(len(row['hidden_layer_sizes'])))
            elif 100 in row['hidden_layer_sizes']:

 hidden_layers.append('{}x100'.format(len(row['hidden_layer_sizes'])))
            elif 200 in row['hidden_layer_sizes']:

 hidden_layers.append('{}x200'.format(len(row['hidden_layer_sizes'])))
        hyperp['hidden_layer_sizes'] = hidden_layers

        test_per = [] # test set performances
        train_per = [] # train set performances

        # get best hyperparameters for each metric and use on test set
        for s in scores:

            # train rf with best hyperparameters for metric
            best_p = res['params'][np.argmax(res['mean_test_{}'.format(s)])]
            ANN = MLPClassifier(solver='sgd', learning_rate='adaptive',

 hidden_layer_sizes=best_p['hidden_layer_sizes'],
momentum=best_p['momentum'])
            ANN.fit(X_train, y_train)

            # predictions for train and test sets
            y_pred = ANN.predict(X_test)
            y_pred_train = ANN.predict(X_train)

            # evaluate metric on test set
            if s == 'accuracy':
                test_per.append(accuracy_score(y_test, y_pred))
```

```python
                train_per.append(accuracy_score(y_train, y_pred_train))
            elif s == 'f1':
                test_per.append(f1_score(y_test, y_pred))
                train_per.append(f1_score(y_train, y_pred_train))
            elif s == 'roc_auc':
                test_per.append(roc_auc_score(y_test, y_pred))
                train_per.append(roc_auc_score(y_train, y_pred_train))
            elif s == 'precision':
                test_per.append(precision_score(y_test, y_pred))
                train_per.append(precision_score(y_train, y_pred_train))
            elif s == 'recall':
                test_per.append(recall_score(y_test, y_pred))
                train_per.append(recall_score(y_train, y_pred_train))

        train_metrics.append(train_per)
        test_metrics.append(test_per)

        print('Trial {} done'.format(trial+1))

    # take mean of each metric across 5 trials
    train_metrics = np.mean(np.array(train_metrics), axis=0)
    test_metrics = np.mean(np.array(test_metrics), axis=0)

    return train_metrics, test_metrics, hyperp
```