```
#!/usr/bin/env python3

"""
Aggregates results in a batch of experiments
"""

import sys
import json
import pickle
import os
import glob

import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics

from datasets import Core50Dataset

def main():

    # Declare experiment batch
    batch_name = "shallow_test_1"
    fbase = "results/{}/".format(batch_name)

    # Load experiment batch
    results = load_batch_results(fbase)

    # Create list of validation accuracy curves
    argmin_losses = [( np.amin(result["history"]["val_loss"]), np.argmin(result["history"
]["val_loss"]) ) for i, result in enumerate(results)]
    acc_curves = [result["history"]["val_acc"] for i, result in enumerate(results)]

    # Plot learning curves
    plot_learning_curves(fbase, argmin_losses, acc_curves, "Validation")

    # Create list of ROC curves
    all_predictions = [result["predict_val"] for result in results]
    outs = Core50Dataset().load_data()["val"]["outs"]
    roc_curves = [generate_roc_curve(outs, predictions) for predictions in all_prediction
s]

    # Plot ROC curves
    plot_roc_curves(fbase, roc_curves, "Validation")

def load_batch_results(fbase):
    """Load the results for the whole batch"""

    file_pattern = fbase + "/experiment*/"
    filepaths = glob.glob(file_pattern)

    results = []
    for filepath in filepaths:
        results.append( load_result_from_experiment(filepath) )

    return results

def load_result_from_experiment(fbase):
    """Load result of given experiment"""
    filename = fbase + "results_dict.pkl"
    with open(filename, "rb") as fp:
        return pickle.load(fp)

def generate_roc_curve(outs, predictions):
        '''
        Produce a ROC plot given a model, a set of inputs and the true outputs
        Assume that model produces N-class output; we will only look at the class 0 score
s
        '''
        # Compute false positive rate & true positive rate + AUC
        fpr, tpr, thresholds = sklearn.metrics.roc_curve(outs[:,0], predictions[:,0])
```

```
        auc = sklearn.metrics.auc(fpr, tpr)

        curve = {
            "fpr": fpr,
            "tpr": tpr,
            "auc": auc
        }

        return curve

def plot_learning_curves(fbase, argmin_losses, curves, set_name):
    """Plot learning curves for the given curves and losses"""
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    # Accumulate the accuracy values for averaging, and plot curves
    acc_sum = 0
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    for i, curve in enumerate(curves):
        loss = argmin_losses[i][0]
        vmax = curve[argmin_losses[i][1]]
        acc_sum += vmax
        ax.plot(curve, label="Best Loss {:.2f}:  Epoch {}, Accuracy {:.2f}".format(argmin
_losses[i][0], argmin_losses[i][1], vmax))

    # Organize figure
    plt.title("{} Learning Curves -- Shallow CNN".format(set_name))
    ax.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.ylabel("Accuracy")
    plt.xlabel("Epochs")

    # Save figure
    fig.savefig(fbase + "learning_curves.png", dpi=fig.dpi, bbox_inches="tight")

    # Save mean accuracies
    with open("{}mean_validation.txt".format(fbase), "w") as f:
        f.write("Average accuracy: {}".format(acc_sum/len(curves)))

def plot_roc_curves(fbase, curves, set_name):
    """Plot roc curves given a set of curves"""

    if not os.path.exists(fbase):
        os.mkdir(fbase)

    # Generate the plot
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    for curve in curves:
        ax.plot(curve["fpr"], curve["tpr"], 'r', label='AUC = {:.3f}'.format(curve["auc"]
))

    # Organize figure
    plt.title("{} ROC Curves -- Shallow CNN".format(set_name))
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')

    # Save figure
    fig.savefig(fbase + "roc_curves.png", dpi=fig.dpi)

if __name__ == "__main__":
    main()
```