```python
from pathlib import Path
from scipy.io import loadmat
from sklearn.model_selection import KFold
import torch
from torch.nn import CrossEntropyLoss
from torch.optim import Adam
from torch.utils.data import DataLoader
from dataset import CustomDataset
from model import Model
from sklearn.model_selection import train_test_split
from utilities import train, test

ROOT = Path(__file__).parent


def problem_2(
,    data_path: str = ROOT / "digits.mat", labels_path: str = ROOT / "labels.mat"
,):
    """Perform K-Fold validation and use those data to train a neural network. The training process utilizes early stopping to prevent overfitting.

    Args:
        data_path (str, optional): path to the data file. Defaults to ROOT/"digits.mat".
        labels_path (str, optional): path to the labels file. Defaults to ROOT/"labels.mat".
    """
    # Hyperparameters
    n_splits = 15
    lr = 0.001
    batch_size = 32
    num_epochs = 1000
    patient = 10

    # Load data from .mat files
    data = loadmat(data_path)["data"].T
    labels = loadmat(labels_path)["labels"].T.flatten()

    # K-Fold cross validation
    kf = KFold(n_splits, shuffle=True)

    # Keep track of currrent fold
    split = 0

    # Initialize average metrices
    avg_train_loss = 0
    avg_train_acc = 0
    avg_val_loss = 0
    avg_val_acc = 0
    avg_test_loss = 0
    avg_test_acc = 0

    # Perform k-fold cross validation
    for train_indices, test_indices in kf.split(data):
        # Split dataset into train, valid, and test set
        X_train, X_val, y_train, y_val = train_test_split(
            data[train_indices], labels[train_indices]
        )
        X_test, y_test = data[test_indices], labels[test_indices]
```

```
57
58          # Convert numpy array to dataset
59          train_dataset = CustomDataset(X_train, y_train)
60          val_dataset = CustomDataset(X_val, y_val)
61          test_dataset = CustomDataset(X_test, y_test)
62
63          # Load train dataset into dataloader for batch training
64          train_dataloader = DataLoader(train_dataset, batch_size)
65
66          # Define features and classes
67          in_shape = torch.from_numpy(data).size()
68          num_classes = torch.unique(torch.from_numpy(labels)).size()[0]
69
70          # Define model, loss function, and optimizer
71          model = Model(in_shape, num_classes)
72          loss_fn = CrossEntropyLoss()
73          optimizer = Adam(model.parameters(), lr)
74
75          # Initialize per fold metrices
76          train_loss = 0
77          train_acc = 0
78          val_loss = 0
79          val_acc = 0
80          test_loss = 0
81          test_acc = 0
82
83          # Early Stop: store multiple losses
84          es_loss_list = []
85
86          # Train the model
87          for epoch in range(num_epochs):
88
89              # Batch training
90              train_loss, train_acc, model = train(
91                  model, loss_fn, optimizer, train_dataloader
92              )
93
94              # Validate the model
95              val_loss, val_acc, model = test(model, loss_fn, val_dataset)
96
97              # Print per epoch metrices
98              print(
99                  f"Epoch: {epoch+1}/{num_epochs},",
100                 f" train_loss:{train_loss:.5f},",
101                 f" train_acc: {train_acc:.3f},",
102                 f" val_loss:{val_loss:.5f},",
103                 f" val_acc:{val_acc:.3f}",
104                 end="\r",
105             )
106
107             # Early stop algorithm
108             # Add latest loss to the end of the list
109             es_loss_list.append(val_loss)
110
111             # Remove first loss if the list is larger than patient
112             if len(es_loss_list) > patient:
113                 es_loss_list = es_loss_list[1:]
```

```python

            # Check if the list is not decreasing
            not_decreasing = all(a <= b for a, b in zip(es_loss_list, es_loss_list[1:]))

            # Stop training once val loss list not decreasing
            if not_decreasing and len(es_loss_list) == patient:
                break

        # Test the model
        test_loss, test_acc, model = test(model, loss_fn, test_dataset)

        # Print per fold metrices
        print(
            f"Fold: {split+1}/{n_splits},",
            f" test_loss:{test_loss:.5f},",
            f" test_acc:{test_acc:.3f},",
            f" train_loss:{train_loss:.5f},",
            f" train_acc: {train_acc:.3f},",
            f" val_loss:{val_loss:.5f},",
            f" val_acc:{val_acc:.3f}",
        )

        # Update average metrices
        avg_val_loss += val_loss / n_splits
        avg_val_acc += val_acc / n_splits
        avg_train_loss += train_loss / n_splits
        avg_train_acc += train_acc / n_splits
        avg_test_loss += test_loss / n_splits
        avg_test_acc += test_acc / n_splits

        # Update split
        split += 1

    # Print average metrices
    print(
        "----Average Metrices----\n",
        f"num_folds: {n_splits}\n",
        f"avg_test_acc: {avg_test_acc:.3f}\n",
        f"avg_train_acc: {avg_train_acc:.3f}\n",
        f"avg_val_acc: {avg_val_acc:.3f}\n",
        f"avg_test_loss: {avg_test_loss:.5f}\n",
        f"avg_train_loss: {avg_train_loss:.5f}\n",
        f"avg_val_loss: {avg_val_loss:.5f}",
    )


if __name__ == "__main__":
    problem_2()
```