ResNet Digit Classifier

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/itü/projects/pattern
```

```
/content/drive/MyDrive/itü/projects/pattern
```

```
!ls cifar-10-batches-py/
```

```
batches.meta  data_batch_2  data_batch_4  readme.html
data_batch_1  data_batch_3  data_batch_5  test_batch
```

```
import pickle

def unpickle(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict

data_batch_1 = unpickle('cifar-10-batches-py/data_batch_1')
data_batch_2 = unpickle('cifar-10-batches-py/data_batch_2')
data_batch_3 = unpickle('cifar-10-batches-py/data_batch_3')
data_batch_4 = unpickle('cifar-10-batches-py/data_batch_4')
data_batch_5 = unpickle('cifar-10-batches-py/data_batch_5')
test_batch = unpickle('cifar-10-batches-py/test_batch')
meta = unpickle('cifar-10-batches-py/batches.meta')

print("Data Batch 1 keys:", data_batch_1.keys())
print("Meta keys:", meta.keys())
```

```
Data Batch 1 keys: dict_keys([b'batch_label', b'labels', b'data', b'filenames'])
Meta keys: dict_keys([b'num_cases_per_batch', b'label_names', b'num_vis'])
```

```
!pwd
```

```
/content
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import numpy as np
import pickle
from torch.utils.data import Dataset, Subset, DataLoader
from tqdm import tqdm
from sklearn.cluster import KMeans

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {DEVICE}")

def unpickle(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict

def load_cifar10_data(data_dir):
    train_data = []
    train_labels = []
    for i in range(1, 6):
        batch = unpickle(f'{data_dir}/data_batch_{i}')
        train_data.append(batch[b'data'])
        train_labels.extend(batch[b'labels'])
    X_train = np.vstack(train_data).reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
    y_train = np.array(train_labels)

    test_batch_data = unpickle(f'{data_dir}/test_batch')
    X_test = test_batch_data[b'data'].reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
    y_test = np.array(test_batch_data[b'labels'])

    return (X_train, y_train), (X_test, y_test)

class Cifar10Raw(Dataset):
    def __init__(self, images, labels, transform=None):
        self.images = images
        self.labels = torch.tensor(labels, dtype=torch.long)
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        image = self.images[idx]
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        return image, label


class CIFAR10Distiller:
    """
    Sizin 'hard-example mining' mantığınızı kullanarak CIFAR-10 veri kümesini damıtır.
    """
    def __init__(self, model, full_dataset, device):
        self.model = model.to(device)
```

```python
        self.model = model.to(device)
        self.full_dataset = full_dataset
        self.device = device
        print("CIFAR10Distiller initialized.")

    @torch.no_grad()
    def create_distilled_dataset(self, distillation_ratio=0.1, batch_size=256):
        print(f"Creating distilled dataset with top {distillation_ratio*100:.1f}% hardest examples...")
        self.model.eval()
        losses_with_indices = []
        criterion = nn.CrossEntropyLoss(reduction='none')
        temp_loader = DataLoader(
            self.full_dataset, batch_size=batch_size,
            shuffle=False, num_workers=2, pin_memory=True
        )
        prog = tqdm(temp_loader, total=len(temp_loader), desc="Calculating losses for distillation")

        for batch_idx, (images, labels) in enumerate(prog):
            images, labels = images.to(self.device), labels.to(self.device)
            outputs = self.model(images)
            loss_per_sample = criterion(outputs, labels)
            start_index = batch_idx * batch_size
            for i, loss in enumerate(loss_per_sample):
                losses_with_indices.append((loss.item(), start_index + i))

        losses_with_indices.sort(key=lambda x: x[0], reverse=True)
        num_to_keep = int(len(losses_with_indices) * distillation_ratio)
        distilled_indices = [index for loss, index in losses_with_indices[:num_to_keep]]
        print(f"\nOriginal dataset size: {len(self.full_dataset)}")
        print(f"Distilled dataset size: {len(distilled_indices)}")
        distilled_dataset = Subset(self.full_dataset, distilled_indices)
        return distilled_dataset



class CIFAR10Distiller_Coreset:
    """
    Özellik uzayında kümeleme yaparak CIFAR-10 için bir çekirdek set (coreset) oluşturur.
    Bu yöntem, 'hard-example mining'e göre daha temsili bir alt küme seçer.
    """
    def __init__(self, model, full_dataset, device):
        self.model = model.to(device)
        self.full_dataset = full_dataset
        self.device = device
        self.feature_extractor = nn.Sequential(*list(model.children())[:-1])
        self.feature_extractor.eval()


    @torch.no_grad()
    def get_features(self, batch_size=256):
        temp_loader = DataLoader(
            self.full_dataset, batch_size=batch_size,
            shuffle=False, num_workers=2, pin_memory=True
        )

        all_features = []
        all_labels = []

        prog = tqdm(temp_loader, desc="Extracting features from all images")
        for images, labels in prog:
            images = images.to(self.device)
            features = self.feature_extractor(images)
            features = features.view(features.size(0), -1)

            all_features.append(features.cpu().numpy())
            all_labels.append(labels.cpu().numpy())

        return np.concatenate(all_features), np.concatenate(all_labels)

    def create_distilled_dataset(self, images_per_class=10):
        features, labels = self.get_features()

        num_classes = len(np.unique(labels))
        distilled_indices = []

        for class_id in range(num_classes):
            indices_in_class = np.where(labels == class_id)[0]
            features_in_class = features[indices_in_class]

            kmeans = KMeans(n_clusters=images_per_class, random_state=42, n_init='auto').fit(features_in_class)

            for cluster_center in kmeans.cluster_centers_:
                distances = np.linalg.norm(features_in_class - cluster_center, axis=1)
                closest_feature_idx = np.argmin(distances)
                original_data_idx = indices_in_class[closest_feature_idx]
                distilled_indices.append(original_data_idx)

            print(f"Selected {images_per_class} prototypes for class {class_id}.")

        print(f"\nOriginal dataset size: {len(self.full_dataset)}")
        print(f"Distilled coreset size: {len(distilled_indices)}")

        distilled_dataset = Subset(self.full_dataset, distilled_indices)
        return distilled_dataset

def train(model, train_loader, epochs, lr=0.01):
    model.to(DEVICE)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=5e-4)
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        prog = tqdm(train_loader, total=len(train_loader), desc=f"Epoch {epoch+1}/{epochs}")
        for images, labels in prog:
            images, labels = images.to(DEVICE), labels.to(DEVICE)
            optimizer.zero_grad()
            outputs = model(images)
```

```python
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            prog.set_postfix(loss=running_loss/len(prog))

def evaluate(model, test_loader):
    model.to(DEVICE)
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(DEVICE), labels.to(DEVICE)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    return accuracy

if __name__ == '__main__':
    transform_train = transforms.Compose([
        transforms.ToTensor(),
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    cifar_path = 'cifar-10-batches-py'
    (X_train, y_train), (X_test, y_test) = load_cifar10_data(cifar_path)
    full_train_dataset_no_aug = Cifar10Raw(X_train, y_train, transform=transform_test)
    full_train_dataset_with_aug = Cifar10Raw(X_train, y_train, transform=transform_train)
    test_dataset = Cifar10Raw(X_test, y_test, transform=transform_test)

    test_loader = DataLoader(test_dataset, batch_size=512, shuffle=False, num_workers=4)
    teacher_model = torchvision.models.resnet18(weights=None, num_classes=10)
    full_train_loader = DataLoader(full_train_dataset_with_aug, batch_size=256, shuffle=True, num_workers=4)

    train(teacher_model, full_train_loader, epochs=50, lr=0.01)

    teacher_accuracy = evaluate(teacher_model, test_loader)
    print(f"\nAccuracy of the pre-trained 'teacher' model: {teacher_accuracy:.2f}%\n")
    print("--- Step 2: Running the distillation process ---")
```

```
Using device: cuda
--- Step 1: Pre-training a model on the full dataset to identify hard examples ---
Epoch 1/50: 100%|██████████| 196/196 [00:07<00:00, 25.46it/s, loss=1.72]
Epoch 2/50: 100%|██████████| 196/196 [00:07<00:00, 25.88it/s, loss=1.38]
Epoch 3/50: 100%|██████████| 196/196 [00:07<00:00, 25.28it/s, loss=1.22]
Epoch 4/50: 100%|██████████| 196/196 [00:07<00:00, 25.07it/s, loss=1.11]
Epoch 5/50: 100%|██████████| 196/196 [00:07<00:00, 25.54it/s, loss=1.02]
Epoch 6/50: 100%|██████████| 196/196 [00:07<00:00, 25.49it/s, loss=0.963]
Epoch 7/50: 100%|██████████| 196/196 [00:07<00:00, 25.65it/s, loss=0.905]
Epoch 8/50: 100%|██████████| 196/196 [00:07<00:00, 25.98it/s, loss=0.866]
Epoch 9/50: 100%|██████████| 196/196 [00:07<00:00, 25.77it/s, loss=0.826]
Epoch 10/50: 100%|██████████| 196/196 [00:07<00:00, 25.93it/s, loss=0.791]
Epoch 11/50: 100%|██████████| 196/196 [00:07<00:00, 25.54it/s, loss=0.763]
Epoch 12/50: 100%|██████████| 196/196 [00:07<00:00, 25.72it/s, loss=0.734]
Epoch 13/50: 100%|██████████| 196/196 [00:07<00:00, 25.88it/s, loss=0.708]
Epoch 14/50: 100%|██████████| 196/196 [00:07<00:00, 25.43it/s, loss=0.687]
Epoch 15/50: 100%|██████████| 196/196 [00:07<00:00, 25.68it/s, loss=0.659]
Epoch 16/50: 100%|██████████| 196/196 [00:07<00:00, 25.49it/s, loss=0.644]
Epoch 17/50: 100%|██████████| 196/196 [00:07<00:00, 25.69it/s, loss=0.629]
Epoch 18/50: 100%|██████████| 196/196 [00:07<00:00, 25.51it/s, loss=0.611]
Epoch 19/50: 100%|██████████| 196/196 [00:07<00:00, 25.53it/s, loss=0.597]
Epoch 20/50: 100%|██████████| 196/196 [00:07<00:00, 25.73it/s, loss=0.576]
Epoch 21/50: 100%|██████████| 196/196 [00:07<00:00, 25.96it/s, loss=0.559]
Epoch 22/50: 100%|██████████| 196/196 [00:07<00:00, 25.69it/s, loss=0.545]
Epoch 23/50: 100%|██████████| 196/196 [00:07<00:00, 25.82it/s, loss=0.534]
Epoch 24/50: 100%|██████████| 196/196 [00:07<00:00, 25.76it/s, loss=0.521]
Epoch 25/50: 100%|██████████| 196/196 [00:07<00:00, 25.83it/s, loss=0.511]
Epoch 26/50: 100%|██████████| 196/196 [00:07<00:00, 25.65it/s, loss=0.502]
Epoch 27/50: 100%|██████████| 196/196 [00:07<00:00, 25.50it/s, loss=0.486]
Epoch 28/50: 100%|██████████| 196/196 [00:07<00:00, 25.41it/s, loss=0.477]
Epoch 29/50: 100%|██████████| 196/196 [00:07<00:00, 25.82it/s, loss=0.47]
Epoch 30/50: 100%|██████████| 196/196 [00:07<00:00, 25.58it/s, loss=0.452]
Epoch 31/50: 100%|██████████| 196/196 [00:07<00:00, 25.77it/s, loss=0.445]
Epoch 32/50: 100%|██████████| 196/196 [00:07<00:00, 25.64it/s, loss=0.436]
Epoch 33/50: 100%|██████████| 196/196 [00:07<00:00, 25.49it/s, loss=0.428]
Epoch 34/50: 100%|██████████| 196/196 [00:07<00:00, 25.94it/s, loss=0.413]
Epoch 35/50: 100%|██████████| 196/196 [00:07<00:00, 25.65it/s, loss=0.411]
Epoch 36/50: 100%|██████████| 196/196 [00:07<00:00, 25.30it/s, loss=0.403]
Epoch 37/50: 100%|██████████| 196/196 [00:07<00:00, 25.78it/s, loss=0.395]
Epoch 38/50: 100%|██████████| 196/196 [00:07<00:00, 25.67it/s, loss=0.385]
Epoch 39/50: 100%|██████████| 196/196 [00:07<00:00, 25.81it/s, loss=0.383]
Epoch 40/50: 100%|██████████| 196/196 [00:07<00:00, 25.55it/s, loss=0.368]
Epoch 41/50: 100%|██████████| 196/196 [00:07<00:00, 25.53it/s, loss=0.36]
Epoch 42/50: 100%|██████████| 196/196 [00:07<00:00, 25.86it/s, loss=0.356]
Epoch 43/50: 100%|██████████| 196/196 [00:07<00:00, 25.17it/s, loss=0.353]
Epoch 44/50: 100%|██████████| 196/196 [00:07<00:00, 25.57it/s, loss=0.341]
Epoch 45/50: 100%|██████████| 196/196 [00:07<00:00, 25.81it/s, loss=0.342]
Epoch 46/50: 100%|██████████| 196/196 [00:07<00:00, 25.43it/s, loss=0.33]
Epoch 47/50: 100%|██████████| 196/196 [00:07<00:00, 25.59it/s, loss=0.332]
Epoch 48/50: 100%|██████████| 196/196 [00:07<00:00, 25.51it/s, loss=0.319]
Epoch 49/50: 100%|██████████| 196/196 [00:07<00:00, 25.51it/s, loss=0.315]
Epoch 50/50: 100%|██████████| 196/196 [00:07<00:00, 25.75it/s, loss=0.309]

Accuracy of the pre-trained 'teacher' model: 80.25%

--- Step 2: Running the distillation process ---
```

```python
teacher_accuracy = evaluate(teacher_model, full_train_loader)
print(f"\nAccuracy of the pre-trained 'teacher' model: {teacher_accuracy:.2f}%\n")
print("--- Step 3: Running the distillation process ---")
```

```
Accuracy of the pre-trained 'teacher' model: 87.62%

--- Step 3: Running the distillation process ---
```

```
    #distiller = CIFAR10Distiller(model=teacher_model, full_dataset=full_train_dataset_no_aug, device=DEVICE)
    #distilled_subset = distiller.create_distilled_dataset(distillation_ratio=0.2)
    distiller = CIFAR10Distiller_Coreset(model=teacher_model, full_dataset=full_train_dataset_no_aug, device=DEVICE)
    distilled_subset = distiller.create_distilled_dataset(images_per_class=400)
    distilled_subset.dataset.transform = transform_train
    distilled_loader = DataLoader(distilled_subset, batch_size=512, shuffle=True, num_workers=4)
    print("\n--- Step 3: Training a new, scratch model on the distilled dataset ---")
    student_model = torchvision.models.resnet18(weights=None, num_classes=10)

    train(student_model, distilled_loader, epochs=200, lr=0.02)

    student_accuracy = evaluate(student_model, test_loader)

    print("\n--- FINAL RESULTS ---")
    print(f"Model trained on FULL data (50k images, 5 epochs): {teacher_accuracy:.2f}% accuracy")
    print(f"Model trained on DISTILLED data ({len(distilled_subset)} images, 50 epochs): {student_accuracy:.2f}% accuracy")
```

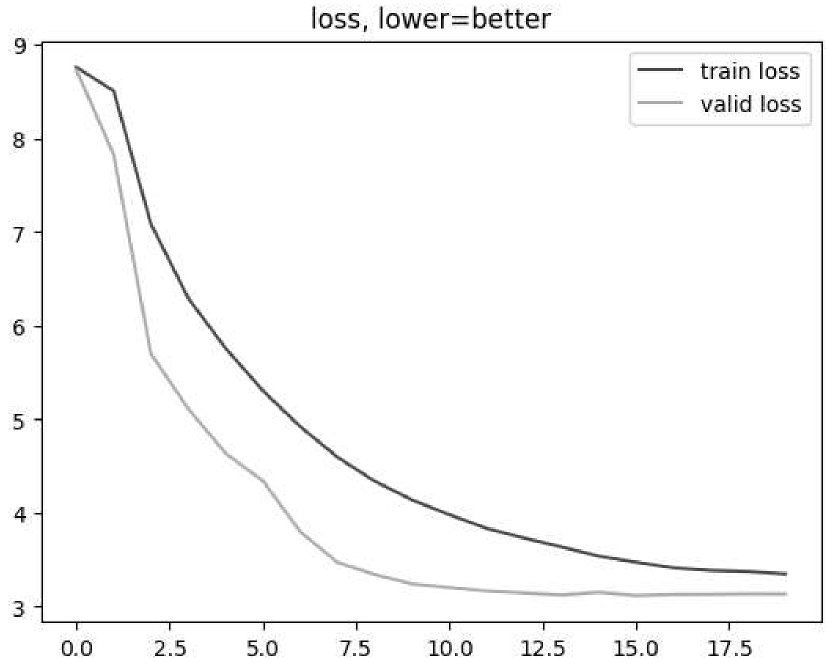
```
Epoch 147/200: 100%|██████████| 8/8 [00:00<00:00,  8.74it/s, loss=0.056]
Epoch 148/200: 100%|██████████| 8/8 [00:00<00:00,  8.61it/s, loss=0.0491]
Epoch 149/200: 100%|██████████| 8/8 [00:00<00:00,  8.84it/s, loss=0.0484]
Epoch 150/200: 100%|██████████| 8/8 [00:00<00:00,  8.89it/s, loss=0.0506]
Epoch 151/200: 100%|██████████| 8/8 [00:00<00:00,  8.61it/s, loss=0.0465]
Epoch 152/200: 100%|██████████| 8/8 [00:00<00:00,  8.76it/s, loss=0.0442]
Epoch 153/200: 100%|██████████| 8/8 [00:00<00:00,  8.80it/s, loss=0.0403]
Epoch 154/200: 100%|██████████| 8/8 [00:00<00:00,  8.55it/s, loss=0.0539]
Epoch 155/200: 100%|██████████| 8/8 [00:00<00:00,  8.73it/s, loss=0.0511]
Epoch 156/200: 100%|██████████| 8/8 [00:00<00:00,  8.76it/s, loss=0.0496]
Epoch 157/200: 100%|██████████| 8/8 [00:00<00:00,  8.78it/s, loss=0.0411]
Epoch 158/200: 100%|██████████| 8/8 [00:00<00:00,  8.77it/s, loss=0.0373]
Epoch 159/200: 100%|██████████| 8/8 [00:00<00:00,  8.51it/s, loss=0.0354]
Epoch 160/200: 100%|██████████| 8/8 [00:01<00:00,  7.97it/s, loss=0.0372]
Epoch 161/200: 100%|██████████| 8/8 [00:00<00:00,  8.56it/s, loss=0.034]
Epoch 162/200: 100%|██████████| 8/8 [00:00<00:00,  8.92it/s, loss=0.0328]
Epoch 163/200: 100%|██████████| 8/8 [00:00<00:00,  8.78it/s, loss=0.0393]
Epoch 164/200: 100%|██████████| 8/8 [00:00<00:00,  8.76it/s, loss=0.0385]
Epoch 165/200: 100%|██████████| 8/8 [00:00<00:00,  8.88it/s, loss=0.0411]
Epoch 166/200: 100%|██████████| 8/8 [00:00<00:00,  8.61it/s, loss=0.0359]
Epoch 167/200: 100%|██████████| 8/8 [00:00<00:00,  8.59it/s, loss=0.0361]
Epoch 168/200: 100%|██████████| 8/8 [00:00<00:00,  8.93it/s, loss=0.0321]
Epoch 169/200: 100%|██████████| 8/8 [00:00<00:00,  8.73it/s, loss=0.0341]
Epoch 170/200: 100%|██████████| 8/8 [00:00<00:00,  8.77it/s, loss=0.0296]
Epoch 171/200: 100%|██████████| 8/8 [00:00<00:00,  8.53it/s, loss=0.0254]
Epoch 172/200: 100%|██████████| 8/8 [00:00<00:00,  8.85it/s, loss=0.025]
Epoch 173/200: 100%|██████████| 8/8 [00:00<00:00,  8.53it/s, loss=0.0279]
Epoch 174/200: 100%|██████████| 8/8 [00:00<00:00,  8.53it/s, loss=0.024]
Epoch 175/200: 100%|██████████| 8/8 [00:00<00:00,  8.90it/s, loss=0.0272]
Epoch 176/200: 100%|██████████| 8/8 [00:00<00:00,  8.83it/s, loss=0.0247]
Epoch 177/200: 100%|██████████| 8/8 [00:00<00:00,  8.78it/s, loss=0.0181]
Epoch 178/200: 100%|██████████| 8/8 [00:00<00:00,  8.73it/s, loss=0.0228]
Epoch 179/200: 100%|██████████| 8/8 [00:00<00:00,  8.62it/s, loss=0.0273]
Epoch 180/200: 100%|██████████| 8/8 [00:00<00:00,  8.72it/s, loss=0.0215]
Epoch 181/200: 100%|██████████| 8/8 [00:00<00:00,  8.78it/s, loss=0.0231]
Epoch 182/200: 100%|██████████| 8/8 [00:00<00:00,  8.76it/s, loss=0.025]
Epoch 183/200: 100%|██████████| 8/8 [00:00<00:00,  8.66it/s, loss=0.0243]
Epoch 184/200: 100%|██████████| 8/8 [00:00<00:00,  8.84it/s, loss=0.0195]
Epoch 185/200: 100%|██████████| 8/8 [00:00<00:00,  8.63it/s, loss=0.0278]
Epoch 186/200: 100%|██████████| 8/8 [00:01<00:00,  7.95it/s, loss=0.0273]
Epoch 187/200: 100%|██████████| 8/8 [00:00<00:00,  8.31it/s, loss=0.0255]
Epoch 188/200: 100%|██████████| 8/8 [00:00<00:00,  8.36it/s, loss=0.0252]
Epoch 189/200: 100%|██████████| 8/8 [00:00<00:00,  8.87it/s, loss=0.0274]
Epoch 190/200: 100%|██████████| 8/8 [00:00<00:00,  8.76it/s, loss=0.0228]
Epoch 191/200: 100%|██████████| 8/8 [00:00<00:00,  8.75it/s, loss=0.0233]
Epoch 192/200: 100%|██████████| 8/8 [00:00<00:00,  8.85it/s, loss=0.021]
Epoch 193/200: 100%|██████████| 8/8 [00:00<00:00,  8.73it/s, loss=0.0199]
Epoch 194/200: 100%|██████████| 8/8 [00:00<00:00,  8.87it/s, loss=0.0196]
Epoch 195/200: 100%|██████████| 8/8 [00:00<00:00,  8.76it/s, loss=0.0233]
Epoch 196/200: 100%|██████████| 8/8 [00:00<00:00,  8.64it/s, loss=0.0205]
Epoch 197/200: 100%|██████████| 8/8 [00:00<00:00,  8.75it/s, loss=0.0199]
Epoch 198/200: 100%|██████████| 8/8 [00:00<00:00,  8.72it/s, loss=0.0194]
Epoch 199/200: 100%|██████████| 8/8 [00:00<00:00,  8.08it/s, loss=0.0185]
Epoch 200/200: 100%|██████████| 8/8 [00:00<00:00,  8.49it/s, loss=0.0156]

--- FINAL RESULTS ---
Model trained on FULL data (50k images, 5 epochs): 87.62% accuracy
Model trained on DISTILLED data (4000 images, 50 epochs): 54.80% accuracy
```

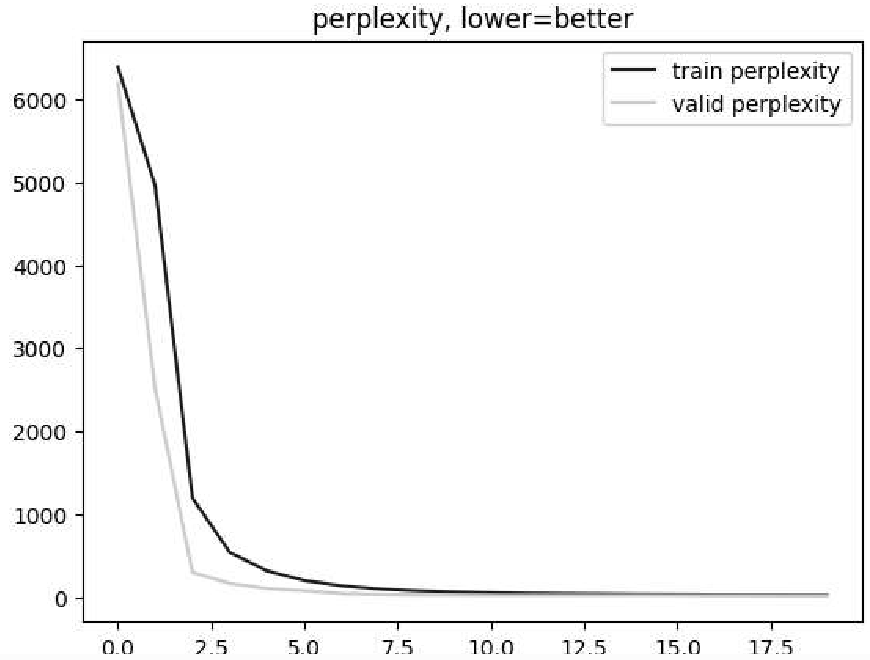
```
student_accuracy = evaluate(student_model, distilled_loader)

print(f"Model trained on DISTILLED data ({len(distilled_subset)} images, 50 epochs): {student_accuracy:.2f}% accuracy")
```

```
Model trained on DISTILLED data (4000 images, 50 epochs): 99.55% accuracy
```

Results

```
TRAIN = True
if TRAIN:
    plt.plot(trainer.metrics['train_loss'],color='red',label='train loss')
    plt.plot(trainer.metrics['val_loss'],color='orange',label='valid loss')
    plt.title('loss, lower=better')
    plt.legend()
    plt.show()
    plt.figure()
    plt.plot(trainer.metrics['train_perplexity'],color='blue',label='train perplexity')
    plt.plot(trainer.metrics['val_perplexity'],color='lightblue',label='valid perplexity')
    plt.title('perplexity, lower=better')
    plt.legend()
    plt.show()
```


loss, lower=better
train loss
valid loss
9
8
7
6
5
4
3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


perplexity, lower=better
train perplexity
valid perplexity
6000
5000
4000
3000
2000
1000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5