ResNet Digit Classifier

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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

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


DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

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

class CIFAR10Distiller_Coreset:
    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 = [], []
        pbar = tqdm(temp_loader, desc="Extracting features")
```

```
        prog = tqdm(temp_loader, desc="Extracting features")
        for images, labels in prog:
            images = images.to(self.device)
            features = self.feature_extractor(images).view(images.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 tqdm(range(num_classes), desc="Finding prototypes per class"):
            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"\nDistilled coreset size: {len(distilled_indices)}")
        return Subset(self.full_dataset, distilled_indices)


class DistilledModelTrainer:
    """
    Verilen bir modeli, verilen bir veri yükleyici (DataLoader) ile eğitir ve değerlendirir.
    """
    def __init__(self, model, train_loader, test_loader, device):
        self.model = model.to(device)
        self.train_loader = train_loader
        self.test_loader = test_loader
        self.device = device
        self.criterion = nn.CrossEntropyLoss()

    def train(self, epochs, lr=0.01):
        print(f"\n--- Starting training for {epochs} epochs with LR={lr} ---")
        optimizer = optim.SGD(self.model.parameters(), lr=lr, momentum=0.9, weight_decay=5e-4)
        scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

        for epoch in range(epochs):
            self.model.train()
            running_loss = 0.0
            prog = tqdm(self.train_loader, desc=f"Epoch {epoch+1}/{epochs}")
            for images, labels in prog:
                images, labels = images.to(self.device), labels.to(self.device)
                optimizer.zero_grad()
                outputs = self.model(images)
                loss = self.criterion(outputs, labels)
                loss.backward()
                optimizer.step()
                running_loss += loss.item()
                prog.set_postfix(loss=f"{running_loss / len(prog):.4f}")
            scheduler.step()

    def evaluate(self, description=""):
        self.model.eval()
        correct, total = 0, 0
        with torch.no_grad():
            for images, labels in self.test_loader:
                images, labels = images.to(self.device), labels.to(self.device)
                outputs = self.model(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        accuracy = 100 * correct / total
        print(f"Accuracy of the model {description}: {accuracy:.2f} %")
        return accuracy


if __name__ == '__main__':
    transform_train_augmented = 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_no_aug = 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_no_aug)
    test_dataset = Cifar10Raw(X_test, y_test, transform=transform_test_no_aug)
    test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False, num_workers=2)

    print("--- Pre-training a teacher model on the full dataset ---")
    teacher_model = torchvision.models.resnet18(weights=None, num_classes=10)
    full_train_loader = DataLoader(Cifar10Raw(X_train, y_train, transform=transform_train_augmented), batch_size=256, shuffle=True, num_workers=2)
    teacher_trainer = DistilledModelTrainer(teacher_model, full_train_loader, test_loader, DEVICE)
    teacher_trainer.train(epochs=50, lr=0.02)
    acc = teacher_trainer.evaluate(description="No Distillation")

    print("\n--- Running the Coreset distillation process ---")
    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)

    print("\n\n===== EXPERIMENT 1: TRAINING ON CORESET WITHOUT AUGMENTATION =====")
    distilled_loader_no_aug = DataLoader(distilled_subset, batch_size=256, shuffle=True, num_workers=2)
    student_model_no_aug = torchvision.models.resnet18(weights=None, num_classes=10)
    trainer_no_aug = DistilledModelTrainer(student_model_no_aug, distilled_loader_no_aug, test_loader, DEVICE)
    trainer_no_aug.train(epochs=50, lr=0.02)
    acc_no_aug = trainer_no_aug.evaluate(description="on Coreset WITHOUT augmentation")
```

```
print("\n\n===== EXPERIMENT 2: TRAINING ON CORESET WITH AUGMENTATION =====")
distilled_subset.dataset.transform = transform_train_augmented

distilled_loader_with_aug = DataLoader(distilled_subset, batch_size=256, shuffle=True, num_workers=2)
student_model_with_aug = torchvision.models.resnet18(weights=None, num_classes=10)
trainer_with_aug = DistilledModelTrainer(student_model_with_aug, distilled_loader_with_aug, test_loader, DEVICE)
trainer_with_aug.train(epochs=50, lr=0.02)
acc_with_aug = trainer_with_aug.evaluate(description="on Coreset WITH augmentation")


print("\n\n========== FINAL COMPARISON ==========")
print(f"Accuracy WITHOUT distillation: {acc:.2f} %")
print(f"Accuracy WITHOUT augmentation: {acc_no_aug:.2f} %")
print(f"Accuracy WITH augmentation:    {acc_with_aug:.2f} %")
print("======================================")
```

```
Epoch 1/50: 100%|██████████| 16/16 [00:00<00:00, 20.10it/s, loss=2.2256]
Epoch 2/50: 100%|██████████| 16/16 [00:00<00:00, 20.50it/s, loss=1.9363]
Epoch 3/50: 100%|██████████| 16/16 [00:00<00:00, 20.53it/s, loss=1.7832]
Epoch 4/50: 100%|██████████| 16/16 [00:00<00:00, 20.46it/s, loss=1.7017]
Epoch 5/50: 100%|██████████| 16/16 [00:00<00:00, 20.78it/s, loss=1.5763]
Epoch 6/50: 100%|██████████| 16/16 [00:00<00:00, 20.70it/s, loss=1.4961]
Epoch 7/50: 100%|██████████| 16/16 [00:00<00:00, 20.70it/s, loss=1.4031]
Epoch 8/50: 100%|██████████| 16/16 [00:00<00:00, 20.50it/s, loss=1.3674]
Epoch 9/50: 100%|██████████| 16/16 [00:00<00:00, 20.24it/s, loss=1.2883]
Epoch 10/50: 100%|██████████| 16/16 [00:00<00:00, 20.60it/s, loss=1.2641]
Epoch 11/50: 100%|██████████| 16/16 [00:00<00:00, 20.24it/s, loss=1.2153]
Epoch 12/50: 100%|██████████| 16/16 [00:00<00:00, 20.83it/s, loss=1.1524]
Epoch 13/50: 100%|██████████| 16/16 [00:00<00:00, 20.80it/s, loss=1.1280]
Epoch 14/50: 100%|██████████| 16/16 [00:00<00:00, 20.63it/s, loss=1.0920]
Epoch 15/50: 100%|██████████| 16/16 [00:00<00:00, 19.91it/s, loss=1.0710]
Epoch 16/50: 100%|██████████| 16/16 [00:00<00:00, 20.43it/s, loss=0.9979]
Epoch 17/50: 100%|██████████| 16/16 [00:00<00:00, 19.45it/s, loss=0.9833]
Epoch 18/50: 100%|██████████| 16/16 [00:00<00:00, 20.65it/s, loss=0.9351]
Epoch 19/50: 100%|██████████| 16/16 [00:00<00:00, 20.30it/s, loss=0.9324]
Epoch 20/50: 100%|██████████| 16/16 [00:00<00:00, 20.35it/s, loss=0.8768]
Epoch 21/50: 100%|██████████| 16/16 [00:00<00:00, 20.59it/s, loss=0.8437]
Epoch 22/50: 100%|██████████| 16/16 [00:00<00:00, 20.41it/s, loss=0.8210]
Epoch 23/50: 100%|██████████| 16/16 [00:00<00:00, 20.74it/s, loss=0.7605]
Epoch 24/50: 100%|██████████| 16/16 [00:00<00:00, 20.64it/s, loss=0.7202]
Epoch 25/50: 100%|██████████| 16/16 [00:00<00:00, 20.31it/s, loss=0.6517]
Epoch 26/50: 100%|██████████| 16/16 [00:00<00:00, 20.78it/s, loss=0.6525]
Epoch 27/50: 100%|██████████| 16/16 [00:00<00:00, 20.59it/s, loss=0.6379]
Epoch 28/50: 100%|██████████| 16/16 [00:00<00:00, 20.72it/s, loss=0.5660]
Epoch 29/50: 100%|██████████| 16/16 [00:00<00:00, 20.97it/s, loss=0.5160]
Epoch 30/50: 100%|██████████| 16/16 [00:00<00:00, 19.92it/s, loss=0.5178]
Epoch 31/50: 100%|██████████| 16/16 [00:00<00:00, 20.17it/s, loss=0.4861]
Epoch 32/50: 100%|██████████| 16/16 [00:00<00:00, 20.37it/s, loss=0.4666]
Epoch 33/50: 100%|██████████| 16/16 [00:00<00:00, 20.34it/s, loss=0.4242]
Epoch 34/50: 100%|██████████| 16/16 [00:00<00:00, 20.92it/s, loss=0.3825]
Epoch 35/50: 100%|██████████| 16/16 [00:00<00:00, 21.11it/s, loss=0.3543]
Epoch 36/50: 100%|██████████| 16/16 [00:00<00:00, 20.96it/s, loss=0.3265]
Epoch 37/50: 100%|██████████| 16/16 [00:00<00:00, 20.59it/s, loss=0.3120]
Epoch 38/50: 100%|██████████| 16/16 [00:00<00:00, 20.44it/s, loss=0.2902]
Epoch 39/50: 100%|██████████| 16/16 [00:00<00:00, 20.79it/s, loss=0.2815]
Epoch 40/50: 100%|██████████| 16/16 [00:00<00:00, 20.74it/s, loss=0.2413]
Epoch 41/50: 100%|██████████| 16/16 [00:00<00:00, 20.68it/s, loss=0.2391]
Epoch 42/50: 100%|██████████| 16/16 [00:00<00:00, 20.66it/s, loss=0.2477]
Epoch 43/50: 100%|██████████| 16/16 [00:00<00:00, 20.44it/s, loss=0.2165]
Epoch 44/50: 100%|██████████| 16/16 [00:00<00:00, 20.97it/s, loss=0.2151]
Epoch 45/50: 100%|██████████| 16/16 [00:00<00:00, 20.68it/s, loss=0.1995]
Epoch 46/50: 100%|██████████| 16/16 [00:00<00:00, 19.56it/s, loss=0.2046]
Epoch 47/50: 100%|██████████| 16/16 [00:00<00:00, 18.52it/s, loss=0.2044]
Epoch 48/50: 100%|██████████| 16/16 [00:00<00:00, 19.65it/s, loss=0.1960]
Epoch 49/50: 100%|██████████| 16/16 [00:00<00:00, 20.52it/s, loss=0.1891]
Epoch 50/50: 100%|██████████| 16/16 [00:00<00:00, 20.92it/s, loss=0.1914]
Accuracy of the model on Coreset WITH augmentation: 55.66 %


========== FINAL COMPARISON ==========
Accuracy WITHOUT distillation: 83.39 %
Accuracy WITHOUT augmentation: 48.97 %
Accuracy WITH augmentation:    55.66 %
======================================
```

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()
```