```
import numpy as np
from torchvision.datasets import CIFAR10
from torch.utils.data import DataLoader, random_split
from sklearn.model_selection import train_test_split
import torchvision.transforms as transforms
import os
from tqdm.notebook import tqdm
import torch
from torch import nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

```
target_size = ([32,32])
num_classes = 10
learning_rate = 0.01
num_epochs = 10
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(target_size),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.Normalize(mean=(0.491, 0.482, 0.446), std=(0.247, 0.243, 0.261))
])

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.491, 0.482, 0.446), std=(0.247 ,0.243, 0.261)),
])

train_data = CIFAR10(root='./data', train=True, download=True, transform=train_transform)
test_data = CIFAR10(root='./data', train=False, download=True, transform=transform)
valid_data, test_data = random_split(test_data, [6500,3500])

train_load = DataLoader(train_data, batch_size=64, shuffle=True)
valid_load = DataLoader(valid_data, batch_size=64, shuffle=False)
test_load = DataLoader(test_data, batch_size=64, shuffle=False)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
class ConvNeuralNet(nn.Module):
  def __init__(self, num_classes) -> None:
    super(ConvNeuralNet, self).__init__()
    # conv layers, maxpool, relu, fully connected
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride =1, padding=1)
    self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride = 1, padding = 1)
    self.conv3 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, stride = 1,padding = 1)
    self.conv4 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride = 1,padding = 1)
    self.conv5 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride = 1,padding = 1)
    self.conv6 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride = 1, padding = 1)
    self.bn1 = nn.BatchNorm2d(16)
    self.bn2 = nn.BatchNorm2d(32)
    self.bn3 = nn.BatchNorm2d(64)
    self.bn4 = nn.BatchNorm2d(128)
    self.maxPool = nn.MaxPool2d(kernel_size=2, stride=2)

    self.fc1 = nn.Linear(128 * 8 * 8, 512)
    self.fc2 = nn.Linear(512, num_classes)

  def forward(self, x):
    x = F.relu(self.bn1(self.conv1(x)))
```

```
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn2(self.conv3(x)))
        x = self.maxPool(x)

        x = F.relu(self.bn3(self.conv4(x)))
        x = F.relu(self.bn3(self.conv5(x)))
        x = F.relu(self.bn4(self.conv6(x)))
        x = self.maxPool(x)

        x = x.view(-1, 128*8*8)

        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
model = ConvNeuralNet(num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
print("training model...")
for epoch in range(num_epochs):
  running_loss=0.0
  for data in train_load:
    # Here we initialize the model and pass in the batches one by one
    # this is repeated for however many epochs we specify
    inputs, labels = data
    # zero out the gradients before passing in a new batch
    optimizer.zero_grad()

    outputs = model(inputs.cuda())

    # Our loss function is a cross entropy loss function, we pass in the outputs from our model
    # and the labels to the corresponding data -> test how close it is
    loss = criterion(outputs.to(device), labels.to(device))
    # back propagate to calculate gradients -> by default all input tensors are
    # set to requires_gradient = true
    loss.backward()
    # takes one optimization step at the end of the batch iteration
    optimizer.step()

    running_loss += loss.item()
  print(f"epoch {epoch + 1}, loss {running_loss/len(train_load)}")
print("finished training")
saved_model = model.state_dict()
torch.save(saved_model, './saved_model.pth') # saving the model
```

```
training model...
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default value of the antialias param
  warnings.warn(
epoch 1, loss 0.7475551733046847
epoch 2, loss 0.6659289847706895
epoch 3, loss 0.6261889767425749
epoch 4, loss 0.5979353812192102
epoch 5, loss 0.5811216227919854
epoch 6, loss 0.5544099824507828
epoch 7, loss 0.5382589554161672
epoch 8, loss 0.5240938409476938
epoch 9, loss 0.5017522856250138
epoch 10, loss 0.48528751929092895
finished training
```

```
model.eval()

total_correct = 0
total_loss = 0.0
```

```
total_images = 0

# to measure how many were right given the real labels
y_true = []
y_pred = []

# to plot accuracy per class
correct_per_class = np.zeros(num_classes)
total_per_class = np.zeros(num_classes)

with torch.no_grad():
  for i, (input,labels) in enumerate(test_load):

    input, labels = input.to(device), labels.to(device)
    outputs = model(input)

    #loss
    loss = criterion(outputs, labels)
    total_loss += loss.item()*len(labels)

    # accuracy
    _, predicted = torch.max(outputs.data, 1)
    total_correct += (predicted == labels).sum().item()
    total_images += labels.size(0)

    labels_numpy = labels.cpu().numpy()
    predicted_numpy = predicted.cpu().numpy()

    y_true.extend(labels.cpu().tolist())
    y_pred.extend(predicted.cpu().tolist())

    for i in range(num_classes):
      correct_per_class[i] += (predicted_numpy[i] == labels_numpy[i])
      total_per_class[i] += 1


val_accuracy = total_correct/total_images
print(f"validation accuracy {val_accuracy}")
average_val_loss = total_loss/total_images
print(f"average validation loss {average_val_loss}")
conf_mat = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10,7))
sns.heatmap(conf_mat, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.show()

class_accuracy = 100 * correct_per_class/total_per_class
plt.figure(figsize=(12,6))
plt.bar(range(num_classes), class_accuracy, color = 'lightblue')
plt.xticks(range(num_classes), classes)
plt.ylabel('Accuracy')
plt.xlabel('Class')
plt.show()
```

```
validation accuracy 0.7408571428571429
average validation loss 0.8145537434305463
```