PR2 – Deep Learning

Submit one zip file that contains three folders for each part.

Part 1 – Image Classification:

In this part, you will train a VGG16 network on the CIFAR100 dataset. The following is a brief tutorial to start with. You should complete the code (some parts are missing).

Import the required modules and libraries:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
```

Resize the size of all images to a unanimous value (224, 224). Convert PIL image objects into Tensors. Normalize the tensor values based on the mean and standard deviation of the RGB values of all the images:

```
data_transforms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])
```

Create an object of `torchvision.datasets.CIFAR100` to get the training and testing set:

```
trainset = datasets.CIFAR100(dataset_root, train = True, transform = data_transforms,
download=True)
testset = datasets.CIFAR100(dataset_root, train=False, transform = data_transforms,
download=True)
```

Create a data loader.

```
torch.utils.data.DataLoader(YOUR_DATASET, batch_size=, shuffle=)
```

Load a VGG16 network with pretrained ImageNet weights:

```
model = models.vgg16(pretrained = True)
```

Extract the number of input features for the last fully connected layer of the model:

```
num_in_ftrs = model.classifier[6].in_features
```

Replace the last fully connected layer with a new layer. The new layer has the same number of input features as the original network but the number of outputs should be equal to the number of classes in the CIFAR100 dataset.

```
model.classifier[6] = nn.Linear(num_in_ftrs, num_cls)  # num_cls is the number of
classes.
```

We are using pretrained weights from the ImageNet dataset. The last layer of VGG16 has been replaced for fitting with our dataset (CIFAR100). Except for the new last layer, weights from other layers need to be frozen. It means that these weights will not be updated during the training.

```
for param in model.parameters():  # freeze all the layers
    param.requires_grad = False
for param in model.classifier[6].parameters():  # unfreeze the last linear layer.
    param.requires_grad = True
```

Set the number of epochs:

```
num_epochs = 10
```

Move the model to GPU (if available):

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
```

Define a loss function for evaluating the trained model:

```
criterion = nn.CrossEntropyLoss()
```

Create an optimizer with an initial learning rate and momentum:

```
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Create a scheduler to control the way that learning rate changes during the training process:

```
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

Iterate over the epochs and save the best model weights. Basically, the best model can achieve the best accuracy during the iteration. In every iteration, we get a mini-batch of images and their corresponding labels. Use `zero_grad()` to reset the calculated gradients. Use the current model weights for predication and backpropagate the prediction loss. After iterating over all batches and if we are in the training phase, we need to run `scheduler.step()` to update the scheduler status.

So, for each batch, the pseudo-code is:

```
images, labels = data  # data is a mini-batch input
optimizer.zero_grad()
outputs = model(images)  # here the model is the pretrained VGG16
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

Save the model weights:

```
torch.save(model.state_dict(), 'best_model.pth')
```

The testing process is very similar to the training process except that there is no need to backpropagate the loss. For testing the model, first, you need to prepare the model in the same way that we prepared it for the training process and load the best model that we saved in the training process.

```
model.load_state_dict(torch.load('best_model.pth'))
```

After loading the model weights, set the model to evaluation mode. Then go through the test set, and predict the category of images, and compute the number of correctly classified images and the accuracy.

So, for each batch, the pseudo-code is:

```
images, labels = data
outputs = model(images)
_, preds = torch.max(outputs, 1)
```

Finally, print the accuracy.

What to submit:

Submit your Google Colab **notebook file** along with its **python code** file as follows:

```
Go to File menu → Download → Download .ipynb
Go to File menu → Download → Download .py
```

Part 2 – Custom Network:

Build your own convolutional neural network for image classification. Use the CIFAR100 dataset to train and test your model (similar to Part 1). The following are the requirements for your network architecture:

1. It must have at least 3 convolutional layers.
2. It must have at least 1 fully connected layer.
3. It must have at least 1 max-pooling layer.
4. It must use ReLU as the activation function.

Since it is a new architecture, there is no pre-trained model to use. So, it should be trained from scratch. You are required to train the network by at least 30 epochs. Save the best model which achieves the best accuracy during training. Next, load your best model to predict the labels for the test set. Print the testing accuracy of your model.

What to submit:

Submit your Google Colab **notebook file** along with its **python code** file as follows:

```
Go to File menu → Download → Download .ipynb
Go to File menu → Download → Download .py
```

Part 3 – Semantic Segmentation:

In this part we are working on the FCN network which is a fully convolutional network for semantic segmentation.

Download 5 (or more) pictures that each one includes at least 3 different objects from these classes: Aeroplan, Bicycle, Bird, Boat, Bottle, Bus, Car, Cat, Chair, Cow, Dining table, Dog, Horse, Motorbike, Person, Potted plant, Sheep, Sofa, Train, Tv/monitor

Similar to part 1, use PyTorch to load a pretrained FCN network (either `fcn_resnet50` or `fcn_resnet101`).

For each image:

- Feed the image to the network.
- The network outputs 21 feature maps. Save all the feature maps (not segmentations) in 1 image (as tiles).
- Create the final segmentation image such that each color represents 1 class.
- You might need to try different input sizes to get the best segmentation.

What to submit:

Submit your Google Colab **notebook file** along with its **python code** file as follows:

`Go to File menu → Download → Download .ipynb`
`Go to File menu → Download → Download .py`

Also, submit the **input images**, **feature maps**, and **final segmentations** (either as a PDF or separate files).