Computer Vision & AI - CSC3831

Practical Session: *Implementation of Convolutional Neural Networks.*

Programming Language: Python. You can choose any form of tool and code-writing platform. However, we recommend using Jupiter Notebook and running them on open online Python compilers such as Google Colab or similar online compilers. There are numerous online compilers for Python.

Dataset: The dataset used in this session is CIFAR-10. CIFAR-10 is a seminal pattern recognition (i.e., image classification) task that has been consistently used as benchmarking Neural Networks since it was first released:

- Original: https://www.cs.toronto.edu/~kriz/cifar.html
- Benchmarking: https://paperswithcode.com/dataset/cifar-10

Library: There are two popular libraries for computer vision and AI.

(1) pytorch (https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html)

PyTorch gives you access to much deeper-level functions and modules for solving the tasks and offers an in-depth understanding of how the tasks are being solved.

A typical implementation of the CIDAR 10 is available here: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

(2) TensorFlow (https://www.tensorflow.org/tutorials).
TensorFlow offers access to a simplified modular approach to solving tasks. It offers an in-depth understanding of high-level abstraction of solving the tasks.

Using this dataset and the library options (you may choose one or both. However, most instruction in this document is largely related to PyTorch), you are asked to solve problems in the following topic:

Pre-requisite: From Part I and Part II sessions, you may be aware of loading CIFAR-10 datasets and playing with this dataset's dimensions and features. You may also be aware of training, validation, and test set splits of datasets for solving *supervised machine learning* problems. The following two tasks are supervised machine learning (i.e., computer vision tasks).

Problem

Load the CIFAR-10 dataset using your preferred library. Most libraries already have a training and test split of this dataset. If not, split the dataset into a Training and Test sets. The CIFAR-10 dataset has the first 60,000 samples as a training set and the next 10,000 as a test set.

Once you have loaded the dataset, ensure the following by printing:

- Each image in the dataset has images of size `32x32x3` pixels where 3 indicates channel (color) and other dimensions represent the images' height and width.
- There are 10 target classes representing different objects.
- Check the image by using plot (e.g., `imshow`) for either of these characters using your preferred `cmap`

- You can import the following libraries

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import datasets
import numpy as np
import matplotlib.pyplot as plt
```

Note: you may require importing more libraries and modules as they may be necessary

```
# choose the training and test datasets
train_data = datasets.CIFAR10('data', train=True,
download=True, transform=transform)
test_data = datasets.CIFAR10('data', train=False,
download=True, transform=transform)
```

Contract a simple CNN with 1 Conv – ReLU, module. In PyTorch, you can create a class (e.g., `class ConvNet(nn.Module)`) using `nn.Module` and in TensorFlow, you can create this either by using `tf.Module`, or you can use the Sequential model approach to stack `Conv Pool, and dennse` layers. Design the module so that you can save the constructed model. For example, you can do this in PyTorch as follows:

```
class ConvNet(nn.Module):
    def __init__(self, imgHeight, imgWidth, categories):
        super().__init__()
            self.conv1 = nn.Conv2d(in_Ch, out_Ch_x, Filters_Size)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(out_Ch_x, out_Ch_y, Filters_Size)
            self.fc1 = nn.Linear(out_Ch_y * Filters_Size * Filters_Size,
            No_Fc1_Nodes)
            self.fc2 = nn.Linear(No_Fc1_Nodes, No_Fc2_Nodes)
            self.fc3 = nn.Linear(No_Fc2_Nodes, Out_Classes)

    def forward(self, x):
         x = self.pool(F.relu(self.conv1(x)))
         x = self.pool(F.relu(self.conv2(x)))
         x = torch.flatten(x, 1) # flatten all dimensions except batch
         x = F.relu(self.fc1(x))
         x = F.relu(self.fc2(x))
         x = self.fc3(x)

        return x

model = ConvNet (imgHeight, imgWidth, categories).to(device)
print(model)
```

Create a training module `def train (…)` for training the model you constructed. You need to choose an optimizer from a set of optimizers (`Adam, sgd, Adagrade, etc`). Adam is the most popular and effective optimizer; you can choose it by default. However, it is recommended to experiment with other optimizers. You use an optimizer.

```
optimizer = torch.optim.Adam(model.parameters())
```

Your training must perform the following:

- it should set a minimum 10 epochs of training to execute the train module

```
epochs = 10
```

- it should use and track `nn.CrossEntropyLoss()` that you can use as in pytorch as

```
loss_fun = nn.CrossEntropyLoss()
```

- it should track training classification accuracy.

```
def correct (output, target):
    pred = output.argmax(1)
    correct_pred = (prediction == target).type(torch.float)
    return correct_pred.sum().item()
```

- it should, at the end of the training, plot per epoch training loss and accuracy

You can write a training module that reports the training history.

```
def train(dataset, model, loss_fun, optimizer):
    model.train()

    num_batches = len(dataset)
    train_len = len(dataset.training)

    total_loss = 0
    total_correct = 0
    for data, target in dataset:
        data = data.to(device)
        target = target.to(device)

        # Do a forward pass
        output = model(data)

        # Calculate the loss
        loss = loss_fun(output, target)
        total_loss += loss

        # Count total correct prediction
        count_correct += prediction (output, target)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    train_loss = total_loss / num_batches
    accuracy = count_correct/ train_len
```

Finally, you create test module `def test(…)` to test the accuracy on the test set and print the test accuracy.

```
def test(test_loader, model, lossfunt):
    model.eval()
    # Other necessary variables
    with torch.no_grad():
        for data, target in test_loader:

            # Do a forward pass
            # Calculate the loss
            # Count the number of correct digits
    test_loss = test_loss/num_batches
    accuracy = total_correct/num_examples

    print(f"Testset accuracy: {100*accuracy:>0.1f}%, average loss:
{test_loss:>7f}")
```

Further Experiments.: *Construct a VGG 16 Net for an Image Classification Task by modifying the ConvNet class and (optionaly) experimenting with it using the TensorFlow library.*

A ConvNet Solution on MNIST dataset is provided along with this instuctions.

VGG 16 architecture is explained in Lecture Slides that has 13 Conv layers and 3 fc layers.