ct1

March 20, 2022

1 DeepLense: Multi-Class Classification

[1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[2]:
```
!unzip -qq gdrive/MyDrive/dataset.zip
print('Extraction done.')
```

```
Extraction done.
```

[3]:
```
import os
import time
import copy
import numpy as np
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, roc_auc_score, auc
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
import torchvision
%matplotlib inline

cudnn.benchmark = True
plt.ion()    # interactive mode
```

1.1 Data Augmentation

The dataset consists of strong lensing images of three classes:

1. no substructure
2. subhalo substructure
3. vortex substructure

The dataset is already normalized so to an extent its already in a standard form.
Since the data is in the form of a list of images each having a single channel and a size of 150x150. We create a simple list of transforms to apply using transforms.Compose. We add a random horizontal flip with 0.5 probability and also enable random rotations up to 90 degrees. This allows our network to learn the image and with a new perspective each time and understand the underlying structure. Since the images have a single channel, we can either copy the same image 3 times or we modify our model's first convolution layer to account for the single channel in our image.

[4]:
```
# Data augmentation
data_transforms = {
    'train': transforms.Compose([
        transforms.ToPILImage(),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(90),
        transforms.Resize(224),
        transforms.ToTensor(),
        #transforms.Normalize([0.5], [0.5]),
        #transforms.Lambda(lambda x: x.repeat(3,1,1))
    ]),
    'val': transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize(224),
        transforms.ToTensor(),
        #transforms.Normalize([0.5], [0.5]),
        #transforms.Lambda(lambda x: x.repeat(3,1,1))
    ]),
}
```

1.2 Data Loading

The dataset is structured as :

```
dataset
  train
      no
            1.npy
            2.npy
         | ...
      sphere
            1.npy
            2.npy
         | ...
      vort
            1.npy
```

```
            2.npy
        | ...
    val
        no
            1.npy
            2.npy
        | ...
        sphere
            1.npy
            2.npy
        | ...
        vort
            1.npy
            2.npy
        | ...
```

Which means we can use the DatasetFolder from the datasets module in the torchvision library with the original directory structure to generate our dataset.

[5]:
```python
def npy_loader(path):
    sample = torch.from_numpy(np.load(path))
    return sample

data_dir = 'dataset/'

image_datasets = {x: datasets.DatasetFolder(
    root=os.path.join(data_dir, x),
    loader=npy_loader,
    transform=data_transforms[x],
    extensions=('.npy')
) for x in ['train', 'val']}
```

We create an iterable dataloader from the generated dataset.

[6]:
```python
dataloaders = {x: torch.utils.data.DataLoader(
    image_datasets[x], batch_size=8,
    shuffle=True, num_workers=2) for x in ['train', 'val']}

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}

device = "cuda" if torch.cuda.is_available() else "cpu"
class_names = image_datasets['train'].classes
```

1.3 Display the image data

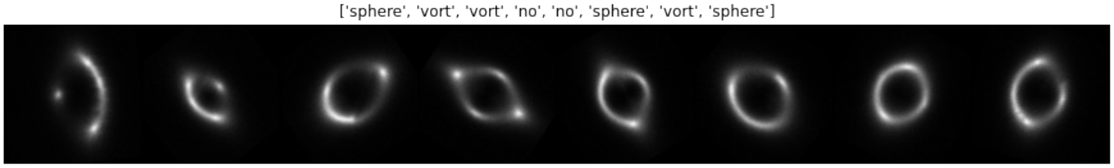
Displaying the augmented images along with their class name.

[41]:
```
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    inp = np.clip(inp, 0, 1)
    plt.figure(figsize = (20,2))
    plt.imshow(inp, cmap='binary')
    if title is not None:
        plt.title(title)
    plt.axis('off')
    plt.pause(0.001)

inputs, classes = next(iter(dataloaders['train']))

out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```



1.4 Training the model

We prepare the model for training. We can transfer learn from a pretrained ResNet model for this task.

[8]:
```
# A function for the training loop.
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        model.train() # Set to training mode
        train_acc = 0.0
        train_loss = 0.0
        for inputs, labels in dataloaders['train']:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad() # Parameter gradients set to zero

            # forward pass and tracking history
            with torch.set_grad_enabled(True):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
```

```
                loss = criterion(outputs, labels)

                # backward pass
                loss.backward()
                optimizer.step()
            train_loss += loss.item() * inputs.size(0)
            train_acc += torch.sum(preds == labels.data)
        scheduler.step()
        train_loss = train_loss / dataset_sizes['train']
        train_acc = train_acc / dataset_sizes['train']
        print(f'Train accuracy : {train_acc} Train Loss : {train_loss}')
```

[9]:

```
# Using the resnet18 model with the pretrained=True
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features

# Modifying the in_channels = 1 to account for single channel in our image
model_ft.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=(7, 7),␣
 ↪stride=(2, 2), padding=(3, 3), bias=False)

# Modifying our final connected layer's out_features to 3 corresponding to our␣
 ↪3 classes of lensing images.
model_ft.fc = nn.Linear(in_features=num_ftrs, out_features=3)
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.Adam(model_ft.parameters(), lr=1e-4, weight_decay=1e-5)

# Using a learning rate scheduler to decay the learning rate of model␣
 ↪parameters by 0.1 every 10th epoch.
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to
/root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
```

```
  0%|          | 0.00/44.7M [00:00<?, ?B/s]
```

[10]:

```
train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=20)
```

```
Epoch 0/19
Train accuracy : 0.4572666585445404 Train Loss : 1.0041208835204443
Epoch 1/19
Train accuracy : 0.7534999847412109 Train Loss : 0.6004982976575692
Epoch 2/19
Train accuracy : 0.8216666579246521 Train Loss : 0.4583210413351655
Epoch 3/19
Train accuracy : 0.8476999998092651 Train Loss : 0.3954284671537578
Epoch 4/19
```

```
Train accuracy : 0.8642666935920715 Train Loss : 0.3593075161462029
Epoch 5/19
Train accuracy : 0.876966655254364 Train Loss : 0.3346447165094316
Epoch 6/19
Train accuracy : 0.8844000101089478 Train Loss : 0.31271330162882804
Epoch 7/19
Train accuracy : 0.8911666870117188 Train Loss : 0.29608962479159234
Epoch 8/19
Train accuracy : 0.8948667049407959 Train Loss : 0.2833737640510003
Epoch 9/19
Train accuracy : 0.899233341217041 Train Loss : 0.27320004466436804
Epoch 10/19
Train accuracy : 0.924833357334137 Train Loss : 0.20700984241887926
Epoch 11/19
Train accuracy : 0.9325667023658752 Train Loss : 0.18997424170672894
Epoch 12/19
Train accuracy : 0.9321333169937134 Train Loss : 0.18686800953478863
Epoch 13/19
Train accuracy : 0.9357333183288574 Train Loss : 0.17656188220499705
Epoch 14/19
Train accuracy : 0.9369333386421204 Train Loss : 0.17597715077990045
Epoch 15/19
Train accuracy : 0.9385666847229004 Train Loss : 0.1722651528686285
Epoch 16/19
Train accuracy : 0.9404333233833313 Train Loss : 0.16808178955633193
Epoch 17/19
Train accuracy : 0.9397667050361633 Train Loss : 0.16874635211390754
Epoch 18/19
Train accuracy : 0.9390333294868469 Train Loss : 0.17005677118225335
Epoch 19/19
Train accuracy : 0.9431999921798706 Train Loss : 0.16044263968675707
```

1.5 Testing

We test the model on our validation data.

[11]:
```
y_score = []
y_test = []

for inputs, labels in dataloaders['val']:
    model_ft.eval() # Setting to eval mode
    inputs = inputs.to(device)
    labels = labels.to(device)
    y_test.append(labels.cpu().detach().numpy())
    y_score.append(nn.functional.softmax(model_ft(inputs), dim=1).cpu().
 ↪detach().numpy())
```

[12]:
```
y_test = np.concatenate(y_test)
y_test_orig = y_test
y_test = label_binarize(y_test, classes=[0, 1, 2])
y_score = np.concatenate(y_score)
```

[43]:
```
print(f'Accuracy on the test set : {(y_score.argmax(axis=1) == y_test_orig).
↪sum() / len(y_test) * 100}')
```

```
Accuracy on the test set : 94.70666666666666
```

1.6 Plot ROC curve

[13]:
```
n_classes = y_test.shape[1]
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

mean_tpr = np.zeros_like(all_fpr)

for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

plt.rcParams['figure.figsize'] = [7, 5]

lw = 2

plt.figure()

plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average (area = {})'
               ''.format(round(roc_auc["micro"],5)),
```

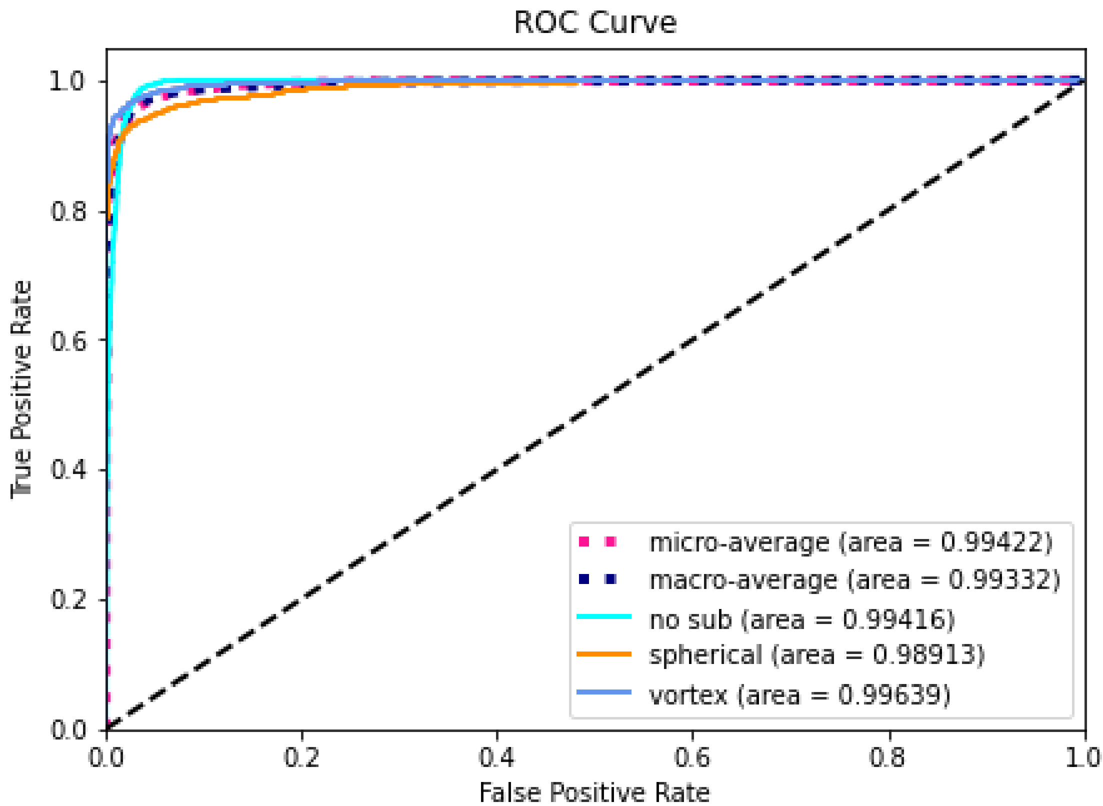
```
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average (area = {})'
               ''.format(round(roc_auc["macro"],5)),
         color='navy', linestyle=':', linewidth=4)

labels = ['no sub', 'spherical', 'vortex']
colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='{} (area = {})'
             ''.format(labels[i], round(roc_auc[i],5)))

# Plot the ROC
plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right", prop={"size":10})
```

[13]: <matplotlib.legend.Legend at 0x7fcb900da650>



1.7 Generate pdf

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('ct1.ipynb')
```

```
--2022-03-20 16:11:21--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.110.133, 185.199.108.133, 185.199.111.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.110.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: 'colab_pdf.py'

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2022-03-20 16:11:21 (14.9 MB/s) - 'colab_pdf.py' saved [1864/1864]
```

```
Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```

[ ]:

```

```