DeepLense: Learning Mass of Dark Matter Halo

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

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

```
!tar zxf gdrive/MyDrive/task_3.tgz
```

```
import numpy as np
from os import listdir
import matplotlib.pyplot as plt
import torch
from torch.nn import MSELoss, Module, Conv2d, Linear
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, models

%matplotlib inline
```

```
files = listdir('lens_data/')

images_arr = []
halo_mass_arr = []

for f in files:
    image, mass = np.load(f"lens_data/{f}", allow_pickle=True)
    images_arr.append(image)
    halo_mass_arr.append(mass)

images_arr = np.stack(np.expand_dims(images_arr, axis=1)).astype(np.float32) # (n,
halo_mass_arr = np.stack(np.expand_dims(halo_mass_arr, axis=1)).astype(np.float32)
```

```
print(images_arr.shape)
print(halo_mass_arr.shape)
```

```
(20000, 1, 150, 150)
(20000, 1)
```

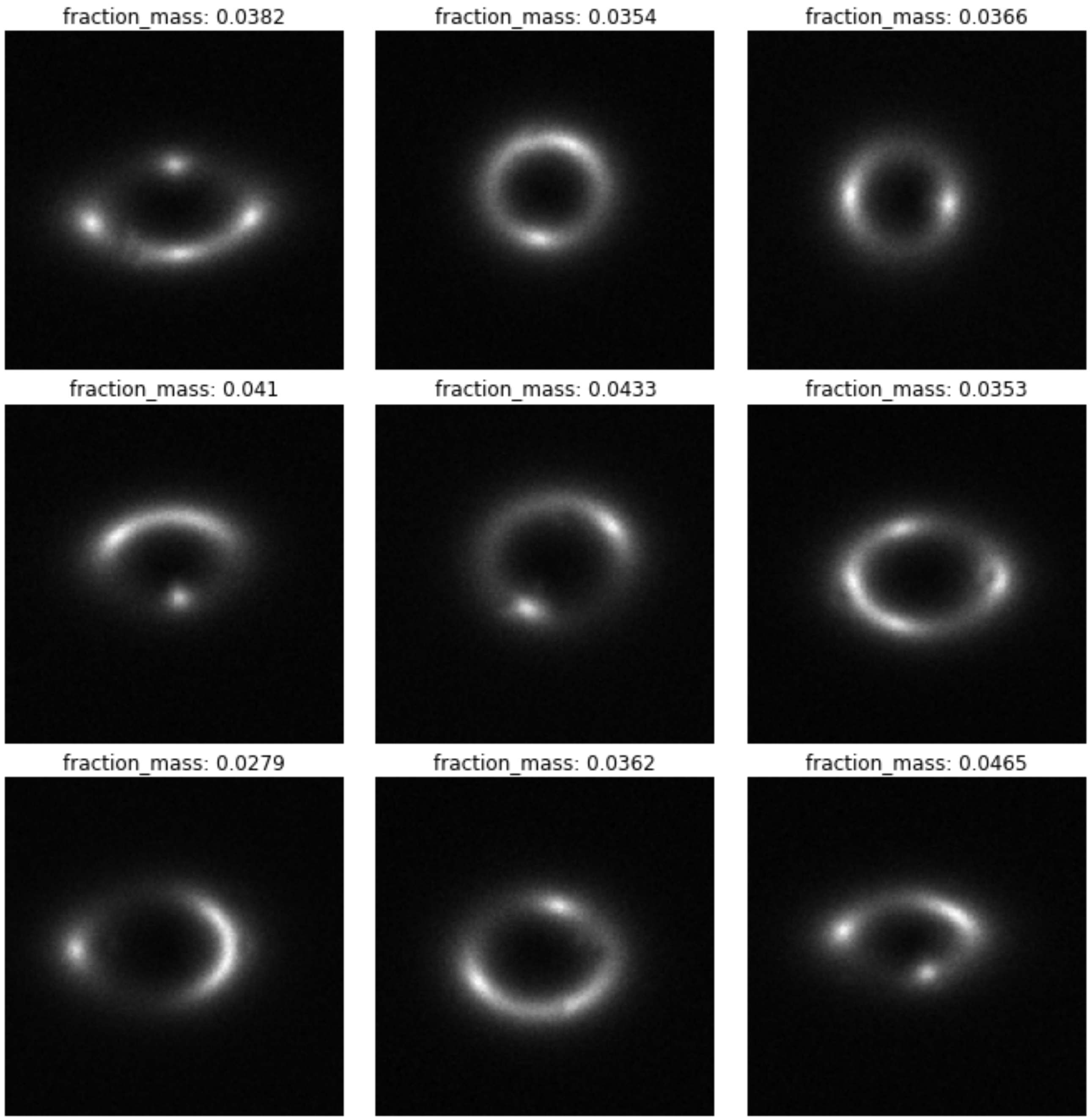
Displaying Lensing Images

```
row = 3
col = 3
```

0s completed at 2:47 PM

```
index=0

for i in range(row):
  for j in range(col):
    img = axis[i][j].imshow(images_arr[index][0], cmap='gist_gray')
    axis[i][j].set_title(f'fraction_mass: {halo_mass_arr[index][0]:.3}')
    axis[i][j].axis('off')
    index+=1
plt.show()
```



Data Augmentation

The data set consists of 20000 black and white (single channel) 150*150 unnormalized lensing images. We need to feature scale them by standardizing (z-score normalise) during the image preprocessing.
Also since the sample above shows that most images are centered, we will crop the image from the center.

```
# Calculating the respective mean and standard deviation
IMG_MEAN, IMG_STD = images_arr.mean(), images_arr.std()
```

```
print(IMG_STD, IMG_MEAN)
```

```
    95.86838 71.63925
```

```
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.Normalize(mean=[IMG_MEAN], std=[IMG_STD])
])
```

Data Loading

We use a custom dataset to store the images and the labels.

```
class ImageDataset(Dataset):
    def __init__(self, x, y, indexes=None):
        self.x = x[indexes]
        self.y = y[indexes]

    def __len__(self):
        return self.x.shape[0]

    def __getitem__(self, idx):
        image, label = self.x[idx], self.y[idx]

        image = torch.tensor(image).float()
        label = torch.tensor(label).float()

        image = preprocess(image)

        return  image, label
```

Split the dataset

```
n = len(images_arr)
t = int(0.9 * n)

train_indices = np.arange(0, t)
test_indices = np.arange(t, n)


train_dataset = ImageDataset(images_arr, halo_mass_arr, train_indices)
test_dataset = ImageDataset(images_arr, halo_mass_arr, test_indices)

batch_size = 64

train_data_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, nu
test_data_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num
```

Loss Function

We will use the Mean Squared Error as the loss function.

```
def mse(pred, true):
  return (np.abs(pred - true)**2).mean()


loss = MSELoss()
```

Creating the model

We will use the VGG13 CNN architecture modifying only the first and last layer for our custom input (single channel) and the regression output ie. 1.

```
class VGG13Regression(Module):
    def __init__(self, channels, op_size):
        super(VGG13Regression, self).__init__()
        self.vgg13 = models.vgg13(pretrained=True)
        self.vgg13.features[0] = Conv2d(
            in_channels=channels,
            out_channels=64,
            kernel_size=(3,3),
            stride=(2,2),
            padding=(2,2),
            bias=True
        )
        self.vgg13.classifier[6] = Linear(
            in_features=4096,
```

```
            out_features=op_size,
            bias=True
        )
    def forward(self, x):
        return self.vgg13(x)
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = VGG13Regression(1,1).to(device)
```

```
Downloading: "https://download.pytorch.org/models/vgg13-19584684.pth" to /root
```

100% 508M/508M [00:07<00:00, 92.5MB/s]

```
# A larger learning rate results in a relatively volatile model.
lr = 1e-4

num_of_epochs = 30

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

Training the model

```
losses = []
for epoch in range(num_of_epochs):
    print(f'Epoch {epoch}/{num_of_epochs - 1}')
    epoch_loss = 0.0
    steps_in_epoch = 0
    for _, (image, mass) in enumerate(train_data_loader):
        optimizer.zero_grad()

        image = image.to(device)
        mass = mass.to(device)

        preds = model(image)

        b_loss = loss(preds, mass)

        b_loss.backward()

        optimizer.step()

        epoch_loss += b_loss
        steps_in_epoch += 1

    w_loss = (epoch_loss/steps_in_epoch).detach().item()
    losses.append(w_loss)
```

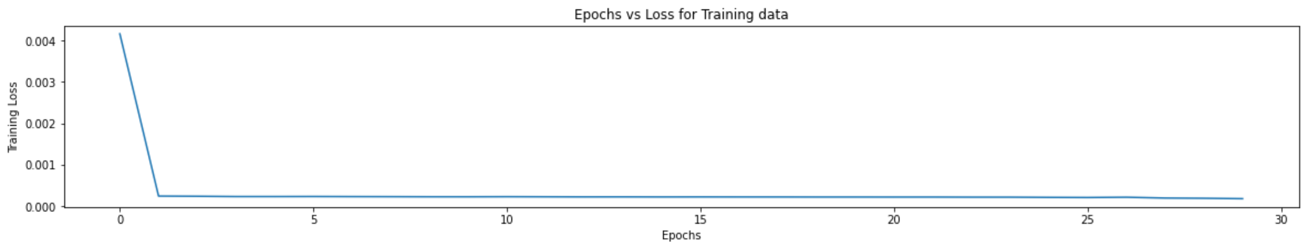
```
losses.append(w_loss)
print(f'Loss {w_loss}')
```

```
Epoch 0/29
Loss 0.00416330574080348
Epoch 1/29
Loss 0.0002425454295007512
Epoch 2/29
Loss 0.00023780424089636654
Epoch 3/29
Loss 0.00023093198251444846
Epoch 4/29
Loss 0.00023086101282387972
Epoch 5/29
Loss 0.000232241305639036
Epoch 6/29
Loss 0.00022985563555266708
Epoch 7/29
Loss 0.000228325036005117
Epoch 8/29
Loss 0.00022597268980462104
Epoch 9/29
Loss 0.00022553876624442637
Epoch 10/29
Loss 0.00022774553508497775
Epoch 11/29
Loss 0.00022548325068783015
Epoch 12/29
Loss 0.0002232967526651919
Epoch 13/29
Loss 0.00022383680334314704
Epoch 14/29
Loss 0.00022204272681847215
Epoch 15/29
Loss 0.0002229842502856627
Epoch 16/29
Loss 0.00022183143300935626
Epoch 17/29
Loss 0.00022136420011520386
Epoch 18/29
Loss 0.0002200013550464064
Epoch 19/29
Loss 0.0002199125592596829
Epoch 20/29
Loss 0.00021896294492762536
Epoch 21/29
Loss 0.0002191315870732069
Epoch 22/29
Loss 0.00021679741621483117
Epoch 23/29
Loss 0.00021632103016600013
Epoch 24/29
Loss 0.0002111839858116582
Epoch 25/29
Loss 0.0002080085687339306
Epoch 26/29
```

```
Loss 0.00021500307775568217
Epoch 27/29
Loss 0.00019500701455399394
Epoch 28/29
Loss 0.0001896429603220895
```

```
plt.xlabel('Epochs')
plt.ylabel('Training Loss')
plt.title('Epochs vs Loss for Training data')
plt.rcParams["figure.figsize"] = (10,3)
plt.plot(np.array(losses))
```

```
[<matplotlib.lines.Line2D at 0x7f694094b110>]
```



Testing

```
predicted_mf_list = []
real_mf_list = []

for step, (image_d, fm_d) in enumerate(test_data_loader):
    optimizer.zero_grad()

    image_d = image_d.to(device)
    fm_d = fm_d.to(device)

    preds = model(image_d)
    predicted_mf_list.append(preds.cpu().detach().numpy())
    real_mf_list.append(fm_d.cpu().numpy())

predicted_mf_list = np.concatenate(predicted_mf_list)
real_mf_list = np.concatenate(real_mf_list)


test_mse = mse(predicted_mf_list,real_mf_list)
print(f'Test MSE: {test_mse}')
```

```
Test MSE: 0.00017165594908874482
```

Save the model

```
torch.save(model.state_dict(), 'ct3_model.pth')
```

Generate pdf

```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('/content/drive/MyDrive/Colab Notebooks/ct3.ipynb')
```