Lighting Estimation using Generative Adversarial Networks

Bhushan Sonawane
StonyBrook, NY 11790
bsonawane@cs.stonybrook.edu

Abstract

Lighting estimation from face image in an important task and has applications in many areas ranging from image editing to image forgery detection[1]. Currently, lighting estimation can be done using on optimization based methods[2]. These methods estimates Spherical Harmonics[3] which along with face normal generates shading. Spherical Harmonics can not model all types of shading. Goal of this project is to generate latent representation in the neural networks which can model different types of lighting representation given input image. In this report, we are presenting first phase of our project, which is, understanding use of adversarial networks for domain adaptation to map synthetic image space into real image space.

1 Introduction

We studied Label Denoising Adversarial Network(LDAN) for Inverse Lighting of face images[1]. LDAN uses synthetic images to train the network for estimating lighting using spherical harmonics and uses Generative Adversarial Networks(GAN)[3] to map real images into synthetic images space in latent representation so as to be able to re-use pre-trained network for estimating lighting. LDAN approach is much faster compared to optimization based approach such as SIRFS[1].

2 Preliminary

LDAN is based on Adversarial Networks, hence it is important to build up strong foundation by experimenting with unsupervised and adversarial networks. Hence, we started off with implementing Auto-Encoder, Variation Auto-Encoder and Generative Adversarial Network.

Implementation of following preliminary work can be found at [4] [5]

2.1 Auto Encoder

We designed different Auto encoder networks to understand how different activation functions and affects the encoder.

We used MNIST dataset for this experiment.

Figure 1 shows the architecture and sample output of the basic Auto Encoder. Figure 2 shows the architecture and sample output of the basic Auto Encoder. Figure 3 shows the architecture and sample output of the basic Auto Encoder.

2.1.1 Conclusion

Based on our experimentation, AutoEncoder with ReLU activation performed better than network with Leaky Relu and Tanh activations.

```
AutoEncoder(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU(inplace)
    (4): Linear(in_features=64, out_features=12, bias=True)
    (5): ReLU(inplace)
    (6): Linear(in_features=12, out_features=3, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=3, out_features=12, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=12, out_features=64, bias=True)
    (3): ReLU(inplace)
    (4): Linear(in_features=64, out_features=128, bias=True)
    (5): ReLU(inplace)
    (6): Linear(in_features=128, out_features=784, bias=True)
    (7): Tanh()
  )
)
```

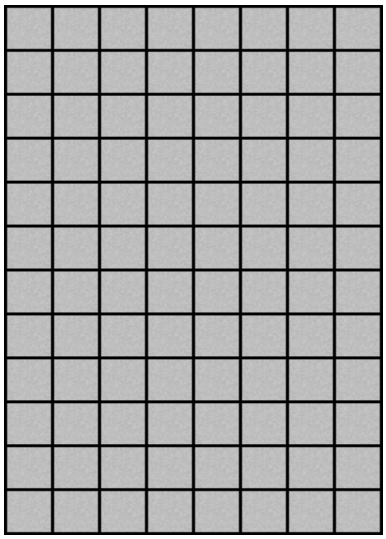
(a) AE Architecture

(b) Output of Autoencoder

Figure 1: Using ReLU

```
AutoEncoderLeaky(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): LeakyReLU(0.1)
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): LeakyReLU(0.1)
    (4): Linear(in_features=64, out_features=12, bias=True)
    (5): LeakyReLU(0.1)
    (6): Linear(in_features=12, out_features=3, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=3, out_features=12, bias=True)
    (1): LeakyReLU(0.1)
    (2): Linear(in_features=12, out_features=64, bias=True)
    (3): LeakyReLU(0.1)
    (4): Linear(in_features=64, out_features=128, bias=True)
    (5): LeakyReLU(0.1)
    (6): Linear(in_features=128, out_features=784, bias=True)
    (7): Tanh()
  )
)
```

(a) AE Architecture



(b) Output of Autoencoder

Figure 2: Using LeakyReLU

2.2 Variational Auto Encoder

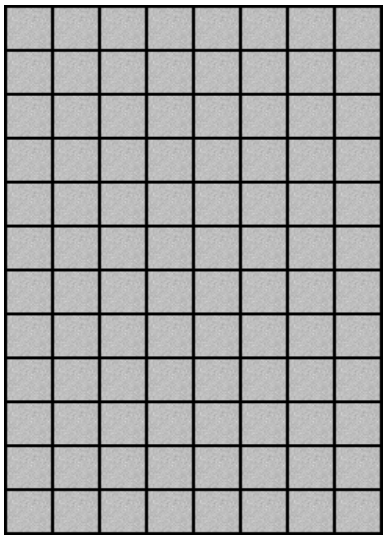
Variational Auto-Encoder makes use of reparametrization trick to train the encoder what noise needs to be fed into decoder for reconstructing the image.

We experimented with MINST dataset with following variations in the architecture:

1. Standard model - Diving encoder output in two halves - mean and variance
2. Adding New layer for mean and variance
3. Drop mean and variance to scalar
4. Dropping mean and variance to vector of 10

```
AutoEncoderT(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): Tanh()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=12, bias=True)
    (5): Tanh()
    (6): Linear(in_features=12, out_features=3, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=3, out_features=12, bias=True)
    (1): Tanh()
    (2): Linear(in_features=12, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=128, bias=True)
    (5): Tanh()
    (6): Linear(in_features=128, out_features=784, bias=True)
    (7): Tanh()
  )
)
```

(a) AE Architecture



(b) Output of Autoencoder

Figure 3: Using TanH

```
VAE(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=400, bias=True)
    (1): LeakyReLU(0.2)
    (2): Linear(in_features=400, out_features=40, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=20, out_features=400, bias=True)
    (1): ReLU()
    (2): Linear(in_features=400, out_features=784, bias=True)
    (3): Sigmoid()
  )
)
```

(a) Standard VAE

```
VAE(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=400, bias=True)
    (1): LeakyReLU(0.2)
    (2): Linear(in_features=400, out_features=20, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=20, out_features=400, bias=True)
    (1): ReLU()
    (2): Linear(in_features=400, out_features=784, bias=True)
    (3): Sigmoid()
  )
  (MuExtractor): Sequential(
    (0): Linear(in_features=20, out_features=20, bias=True)
    (1): Sigmoid()
  )
  (SigmaExtractor): Sequential(
    (0): Linear(in_features=20, out_features=20, bias=True)
    (1): Sigmoid()
  )
)
```

(b) Model2: Adding additional layerr

Figure 4: Variation Auto-Encoder architectures

Figure 4 and 5 demonstrates architectures used for the experiment.

2.2.1 Conclusion

Standard Variation-AutoEncoder outperformed other variations we tried.

2.3 Generative Adversarial Networks

Generative Adversarial Networks are known to be difficult to train, hence it is important to get familiar with training different GANs. We started with training Vanilla GAN on MNIST to training DC-GAN on CelebA datasets.
We came up with new appraoch to train GANs- using multiple Generators, where weights of better

```
VAE(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=400, bias=True)
    (1): LeakyReLU(0.2)
    (2): Linear(in_features=400, out_features=20, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=1, out_features=400, bias=True)
    (1): ReLU()
    (2): Linear(in_features=400, out_features=20, bias=True)
    (3): ReLU()
    (4): Linear(in_features=20, out_features=784, bias=True)
    (5): Sigmoid()
  )
  (MuExtractor): Sequential(
    (0): Linear(in_features=20, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (SigmaExtractor): Sequential(
    (0): Linear(in_features=20, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

(a) Dropping mean and variance to scalar

```
VAE(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=400, bias=True)
    (1): LeakyReLU(0.2)
    (2): Linear(in_features=400, out_features=20, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=10, out_features=400, bias=True)
    (1): ReLU()
    (2): Linear(in_features=400, out_features=20, bias=True)
    (3): ReLU()
    (4): Linear(in_features=20, out_features=784, bias=True)
    (5): Sigmoid()
  )
  (MuExtractor): Sequential(
    (0): Linear(in_features=20, out_features=10, bias=True)
  )
  (SigmaExtractor): Sequential(
    (0): Linear(in_features=20, out_features=10, bias=True)
  )
)
```

(b) Dropping mean and variance to 10

Figure 5: Variation Auto-Encoder architectures

performing Generator is copied over all Generators- Co-Operative GANs and will discuss in following sub-section.

2.3.1 Vanilla GAN

We implemented Vanilla GAN for generating hand written digits using MNIST dataset.

Figure 6 describes architecture used and Figure 7 shows results after 100 iterations

```
Sequential(
  (0): Linear(in_features=64, out_features=256, bias=True)
  (1): LeakyReLU(0.2)
  (2): Linear(in_features=256, out_features=256, bias=True)
  (3): LeakyReLU(0.2)
  (4): Linear(in_features=256, out_features=784, bias=True)
  (5): Tanh()
)
```

(a) Vanilla GAN: Generator

```
Sequential(
  (0): Linear(in_features=784, out_features=256, bias=True)
  (1): LeakyReLU(0.2)
  (2): Linear(in_features=256, out_features=256, bias=True)
  (3): LeakyReLU(0.2)
  (4): Linear(in_features=256, out_features=1, bias=True)
  (5): Sigmoid()
)
```

(b) Vanilla GAN: Dicriminator

Figure 6: Vanilla GAN architectures

2.3.2 DC-GAN

We implemented DC GAN for generating Celebrity face images using CelebA dataset.

Figure 8 describes architecture used and Figure 9 shows results after 100 iterations

2.3.3 Co-Operative GAN

Mode collapsing is general problem while training GANs. Many researcher have already used concept of using multiple generators for overcoming mode collapsing problem. M-GAN[6] uses Generators independently and uses random generator at the end of training.

Figure 7: Output of Vanilla GAN

```
generator(
  (deconv1): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1))
  (deconv1_bn): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
  (deconv2): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (deconv2_bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
  (deconv3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (deconv3_bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
  (deconv4): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (deconv4_bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
  (deconv5): ConvTranspose2d(128, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
)
```

(a) DC GAN: Generator

```
discriminator(
  (conv1): Conv2d(1, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2_bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
  (conv3): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv3_bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
  (conv4): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv4_bn): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
  (conv5): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1))
)
```

(b) DC GAN: Dicriminator

Figure 8: DC GAN architectures

We propose to make generators share their learning with other generators at the end of every iteration. Weights of best performing generator will be copied over all other generators - i.e. winner take all!

This method allows us to use different configuration of every generator such as learning rate and optimizer. At the end of every iteration, every generator is initialized to weights of better performing generator. Generators progressing with collaboration hence named Co-Operative GANs.

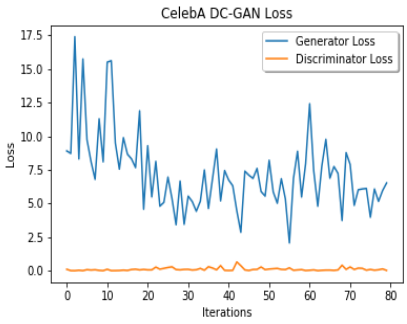
Conclusion
Co-Operative GAN worked well on MNIST dataset but experiments did not involve variation in optimizer, learning rate. It helped coming over mode collapsing, but it is of no help if Generator is badly designed.

2.3.4 Using GANs for designing Neural Networks

In deep learning, neural networks are very task specific. Lot of efforts are put in hyper-parameter tuning which ends up taking most of the researchers time. What if we could use GANs for generating new architecture?

We propose following approach for the same. Use separate Generator for controlling different parameters of the network such as number hidden of layers, optimizers to use, activation functions to use, number of iterations. Discriminator is task specific and will need manual designing for every task. Discriminator will validate the designed network.

(a) DC GAN: Generated faces



(b) DC GAN: Loss function

Figure 9: DC GAN Results

Training such network requires massive compute power as generator is coming up new network architecture and testing with it requires training new deep network during discriminator training.

Currently, this is just a thought experiment and will investigate on small tasks such as coming up with networks to learn boolean functions and distributions like gaussion.

3 Label Denoising Adversarial Network(LDAN) for Inverse Lighting of Face Images

Now, we introduce to the paper we studied for understanding usage of adversial network for domain adaptation.

3.1 Problem

As discussed in introduction, light estimation task lacks ground truth for real images. Ground truth is available for synthetic images and we can train the neural network on synthetic images. We can use this trained model to estimate lighting for real images, but real images and synthetic images are from different distribution and using network trained on synthetic images for estimating lighting for real images does not lead to accurate lighting estimation[1]. Hence, we need to map lighting feature distribution of real images into lighting features of synthetic image distribution. Hence, adversarial network comes into picture.

3.2 Training

Figure 10 shows training procedure for LDAN. Training is divided in following two steps:

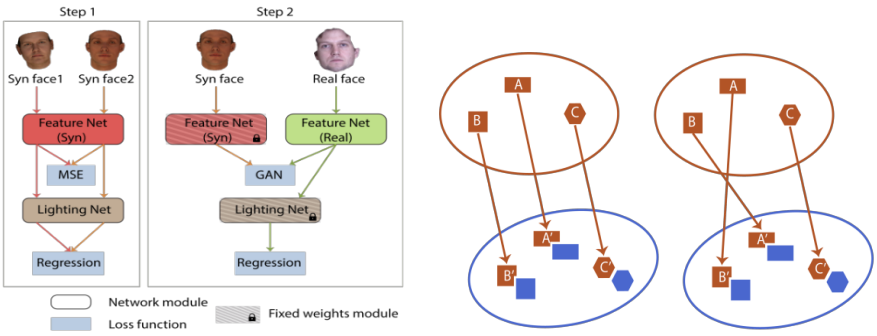
1. FeatureNet and LightingNet is trained on synthetic images using ground truth spherical harmonics.
2. Lighting net is locked and new FeatureNet is used as Generator to generator lighting features for real images. Discriminator is used to distinguish lighting nets from real and synthetic images space.

In step 2, GAN is used for training FeatureNet of real images to generate latent representation that is from same distribution as synthetic image so that, we can use lighting net trained on synthetic data for estimating lighting of real images.

4 Dataset

4.1 Dataset used

For our experiment, we used following three datasets:



(a) LDAN steps (b) LDAN mapping distribution

Figure 10: Training LDAN

Table 1: Fields of LDAN H5 dataset

Field	Comment	Dataset dimensions
$data_1$	First Synthetic Image	2500 x 64 x 64 x 3
$data_2$	Second Synthetic Image	2500 x 64 x 64 x 3
label	Ground truth sperical harmonics	2500 x 27

1. **LDAN Synthetic Dataset**: We are using Synthetic data used for LDAN experiment for training in step 1. Dataset was collected from LDAN author Hao Zhou and is stored at PATH. Images are not synthesized correctly and SIRFS cannot be applied. We were unable to get correct image mask, normal for these images hence, we only worked with ground truth SH of this data.
   Original dataset is in H5 format[7]

   Refer to table 1 which describes LDAN dataset fields.

2. **SfSNet Synthetic Dataset**: We are using Synthetic Dataset used by SfSNet experiment as Real image for one experiment and as synthetic dataset for Auto-encoder experiment. SfSNet dataset is different from LDAN Synthetic dataset and do have ground truth hence, can be used for verifying results.
   Original dataset collected from Soumyadip Sengupta is stored at [8]
   H5 file with ground truth and SIRFS data are located at [9]
   Please refer to table 2 which describes SfSNet data fields.

3. **CelebA dataset**: We are using CelebA real image dataset as real images for some experiments.
   Images are located here: [10]
   We have used crop_resize_maskout from every folder for generating SIRFS data.
   H5 file with SIRFS data are located at [11]
   Please refer to table 3 which describes h5 dataset fields.

4.2 Generating Spherical Harmonics

During GAN training, we need noisy spherical harmonics(SH) for calculating generator regression loss. We are using SIRFS implementation by Jon Barron for generating SH. Original implementation enforces use of fast c based implementation which has setup dependency, hence, we would recommend to refer to our updated version[12] which allows one to switch between faster c implementation and relatively slower but no dependency matlab based implementation.

We are generating Noisy SH for CelebA dataset for Celebtriy Dataset Experiment and for SfSNet Synthetic Dataset for SfSNet and Auto-Lighting experiment.

Table 2: Fields of SfSNet H5 files

Field	Comment	Dataset dimensions
Image	Input image	2500 x 64 x 64 x 3
Lighting	Ground truth SH	2500 x 27
Mask	Ground truth Mask	2500 x 64 x 64 x 3
Normal	Ground truth Normal	2500 x 64 x 64 x 3
SIRFS_FinalLoss	Final loss computed by SIRFS method	2500 x 1
SIRFS_Height	Height parameter output of SIRFS	2500 x 64 x 64
SIRFS_Lighting	SH computed by SIRFS method	2500 x 64 x 64 x 3
SIRFS_Normal	Normal computed by SIRFS method	2500 x 64 x 64 x 3
SIRFS_Reflectance	Reflectance computed by SIRFS method	2500 x 64 x 64 x 3
SIRFS_Shading	Shading computed by SIRFS method	2500 x 64 x 64 x 3
Albedo	Ground truth albedo of input image	2500 x 64 x 64 x 3
Shading	Ground truth shading computed by Image / Albedo	2500 x 64 x 64 x 3

Table 3: Fields of CelebA H5 files

Field	Comment	Dataset dimensions
FinalLoss	Final loss computed by SIRFS method	2500 x 1
Height	Height parameter output of SIRFS	2500 x 64 x 64
Image	Input image	2500 x 64 x 64 x 3
Mask	Input mask	2500 x 64 x 64 x 3
Lighting	SH computed by SIRFS method	2500 x 64 x 64 x 3
Normal	Normal computed by SIRFS method	2500 x 64 x 64 x 3
Reflectance	Reflectance computed by SIRFS method	2500 x 64 x 64 x 3
Shading	Shading computed by SIRFS method	2500 x 64 x 64 x 3

We are generating Spherical Harmonics, Normal, Shading of the images and storing in H5 files. We have described dataset fields in table 3 and 2 for SfSNet and CelebA dataset.

Figure 11 shows output of sirfs method:

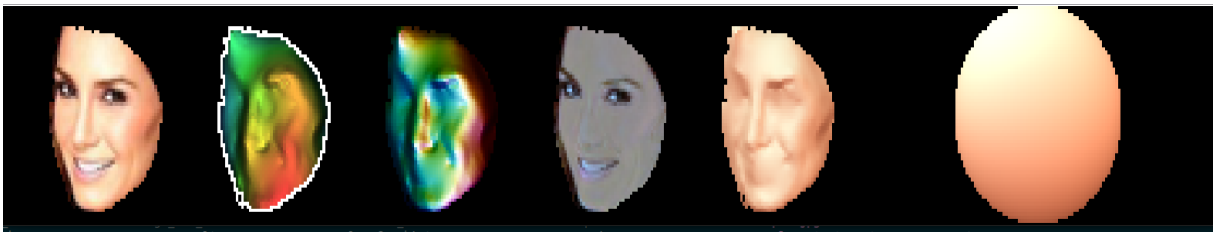


Figure 11: SIRFS method output

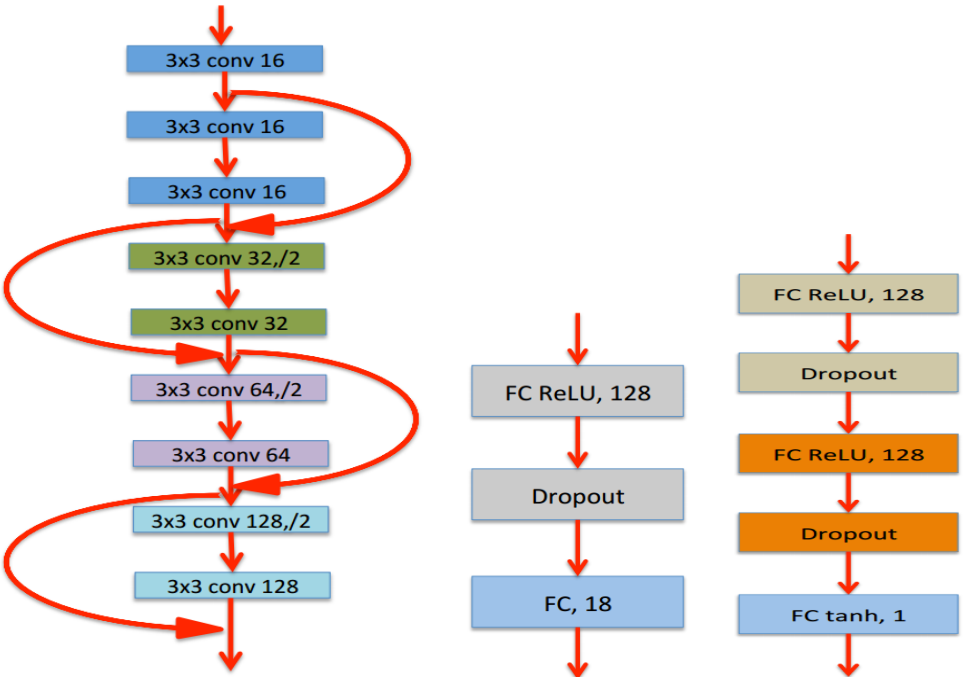
5 Network Architecture

Figure 12 shows LDAN network architecture. Generator uses Feature Net structure.

Figure 12(a) shows Feature Net which uses apply ResNet structure. 3x3 conv 16 means convolutional layer with 16 filters and size of each filter is 3x3. /2 signifies stride size is two. In ResNet, 3x3 conv 16 is followed by ReLU layer.

Figure 12(b) shows Lighting Net which uses Fully connected layer of 128x128 followed by dropout layer followed by Fully connected layer of 128x18.

Figure 12(c) shows Discriminator Net which also two Fully connected layer with input and output of size 128 followed by ReLU activation and dropout layer.



(a) Feature Net (b) Lighting Net (c) Discriminator

Figure 12: LDAN Network structure

5.1 Loss functions

Loss function are adapted from LDAN paper.

6 Experiment

In this section, we describe experiments and their results conducted on LDAN network.

Implementation of these experiment can be found here[13]

6.1 Estimating lighting of CelebA Images

In this experiment, we are estimating lighting for CelebA images.

6.1.1 Dataset

1. **Synthetic data**: LDAN Synthetic dataset
2. **Real image data**: CelebA data
3. **SIRFS SH for training GAN**: Generated using SIRFS on CelebA dataset. SIRFS SH is reasonably close to the ground truth data[1]; hence, we are using SIRFS SH as anchor point for training GAN.
4. **Ground truth for training Feature Net and Lighting Net**: Provided in LDAN synthetic dataset

6.1.2 Training Procedure

1. Train with Feature Net and Lighting Net with synthetic dataset using Regression and Feature Loss.
2. Generate lighting features for synthetic data using trained Feature Net

3. Train GAN for mapping real image lighting features into synthetic image lighting features. Note that we don't update weights of Lighting Net during GAN training.

Generator from step 3 and Lighting net from step 1 now can be used for estimating lighting of CelebA images.

6.1.3 Results

Due to lack of ground truth, we are comparing estimated shading with SIRFS generated shading. Figure 14 shows Real image and it's SIRFS estimated normal and figure 16 compares SIRFS estimated shading with LDAN estimated shading. Figure 17 shows estimated shading for real images with out domain adaption where network is trained only on synthetic data.
Plot 15 shows performance of validation data during GAN training. Note that, we are comparing with SIRFS SH and MSE decreasing signifies that SIRFS domain is being adapted.

6.1.4 Conclusion

Plot 15 and figure 16 confirms that real images are adapting Synthetic image space.

6.2 Estimating Lighting of SfSNet dataset wih SIRFS SH as anchor point during GAN training

In Experiment 1, we lack ground truth of CelebA dataset. Hence, we decided to replace CelebA images in step 3 with SfSNet dataset to validate estimated lighting with available ground truth.

6.2.1 Dataset

1. **Synthetic data**: LDAN Synthetic dataset
2. **Real image data**: SfSNet dataset
3. **Noisy SH for training GAN**: Generated using SIRFS on SfSNet dataset. SIRFS SH reasonably close to the ground truth data[1], hence, we are using SIRFS SH as anchor point for training GAN.
4. **Ground truth for training Feature Net and Lighting Net**: Provided in LDAN synthetic dataset.

6.2.2 Results

Figure 18 shows Real image and ground truth normal. Figure 20 shows ground truth shading calculated using (Real Image / Albedo) and SIRFS estimation. 21 compares estimated shading with and with out GAN training. Plot 19 shows MSE of predicted SH with respect to ground truth SH and SIRFS SH. We can see that, MSE with respect SIRFS SH is decreasing where as MSE with respect to grount truth is increasing.

6.2.3 Conclusion

Figure 20 and 21 shows that GAN quickly adapts to the SIRFS shading when SIRFS SH is used for training GAN. Although results are not better than SIRFS shading, we can see SIRFS domain was adapted by LDAN network.

6.3 Estimating Lighting of SfSNet dataset wih Ground truth SH as anchor point during GAN training

In Experiment 1 and 2, we trained GAN with SIRFS SH. In this experiment we replace SIRFS SH in GAN training with ground truth SH.

6.3.1 Dataset

1. **Synthetic data**: LDAN Synthetic dataset
2. **Real image data**: SfSNet dataset.

3. **Ground SH for training GAN**: Ground truth SH of SfSNet.
4. **Ground truth for training Feature Net and Lighting Net**: Provided in LDAN synthetic dataset.

6.3.2 Results

Due to lack of ground truth, we are comparing estimated shading with SIRFS generated shading.

Figure 22 shows Real image and ground truth normal. 24 shows ground truth shading calculated using (Real Image / Albedo) and SIRFS estimation. 25 compares estimated shading with and with out GAN training. PLot 23 compares MSE of predicted SH on validation set with respect to SIRFS SH and ground truth SH. Here, MSE with respect to ground truth is lower than MSE with respective to SIRFS SH i.e. exactly opposite of experiment 2.

6.4 AutoLighting: Denoising SIRFS SH into ground truth SH

In previous experiments, we need to map real image space into synthetic image space so that we can use existing lighting network trained on synthetic images. We propose alternative to this approach by using auto-encoders to convert SIRFS based spherical harmonics to ground truth spherical harmonics. Figure 13 shows Deep Network and Auto-Encoder architecture. We are using LDAN's FeatureNet

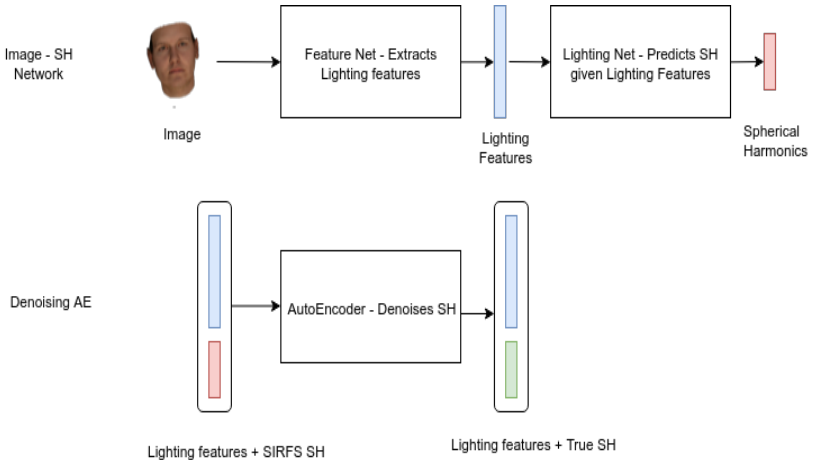


Figure 13: Auto-Lighting architecture

and LightingNet as Deep Network. For Auto-Encoder, aim is to train auto-encoder to denoise spherical harmonics given latent space feature net generated by Feature Net.

6.4.1 Input and Output of Deep Network

Input: Synthetic images are fed to Feature Net which generates Lighting Features as latent space output which is fed to Lighting Net.
Output: Lighting features are output of Feature net which is used for Auto-Encoder training. Spherical harmonics is output of Lighting Net.

6.4.2 Input and Output of Auto-Encoder

Input: Lighting features generated from Feature Net and SIRFS generated spherical harmonics are fed to Auto-encoder.
Output: Lighting features given as input and ground truth spherical harmonics.

6.4.3 Training Auto-Encoder

Goal is to train auto-encoder to convert SIRFS spherical harmonics into true spherical harmonics for given latent lighting features.
Training Auto-encoder includes following steps:

1. Generate lighting features from Feature Net by training on synthetic dataset.
2. Train Auto-Encoder to denoise synthetic SIRFS spherical harmonics to ground truth spherical harmonics.
3. Generate SIRFS spherical harmonics for real images
4. Denoise SIRFS spherical harmonics of real images using trained auto-encoder.
5. Train Deep Network (Feature Net + Lighting Net) to estimate spherical harmonics using real images as input and denoised spherical harmonics.

6.4.4 Results

We used SfSNet dataset as synthetic dataset and CelebA dataset as Real images for this experiment.

Figure 26 validation loss of predicted SH with SIRFS SH. Figure 27 shows Real image and sirfs normal. 28 compares SIRFS estimated shading and Auto-Lighting shading estimation. Figure 29 compares Auto-Lighting generated shading with LDAN network trained on synthetic data only. We have trained auto-encoder to convert SIRFS SH of synthetic images to ground truth SH of synthetic images. SIRFS SH of CelebA are indeed converted into ground truth SH space of synthetic images.

6.5 Conclusion

Generated shading with this method does not resembles with the SIRFS shading but does resembles with estimation with LDAN network trained on synthetic images only(Experiment 1). One problem could be, SIRFS SH is very important in this network and in-accurate estimation of some parameter might lead to different denoised SH. This idea needs more work for accurately estimating SH of real images.

7 Conclusion

GAN adapts to lighting features of synthetic space so that we can re-use lighting net trained on synthetic data to estimate spherical harmonics for real images.

8 Feature Work

Our experiments showed that adversarial approach can be used for domain adaptation. Now, we move our attention to need of SH for lighting estimation. We will be working on to design neural network capable of learning lighting representation itself. We expect it to be more robust that spherical harmonics as neural network is capable of modelling fine details.

References

[1] Jin Sun Hao Zhau. Label Denoising Adversarial Network (LDAN) for Inverse Lighting of Face Images. `https://arxiv.org/abs/1709.01993`, 2017.

[2] J. Malik J. T. Barron. Shape, illumination, and reflectance from shading. `https://ieeexplore.ieee.org/abstract/document/6975182/`, 2015.

[3] Mehdi Mirza Ian J. Goodfellow, Jean Pouget-Abadie. Generative Adversarial Networks. `https://arxiv.org/abs/1406.2661`, 2014.

[4] Bhushan Sonawane. Experimenting with AEs, VAEs and GANS. `https://github.com/bhushan23/GAN`, 2018.

[5] Bhushan Sonawane. DC-GAN for Celebrity face generation. `https://github.com/bhushan23/Computer-Vision`, 2018.

[6] Trung Le Dinh Phung Quan Hoang, Tu Dinh Nguyen. Multi-Generator Generative Adversarial Nets. `https://arxiv.org/abs/1708.02556`, 2017.

[7] Hao Zhou. LDAN Dataset H5 files. `/home/bsonawane/Thesis/LightEstimation/dataset/LDAN/Original_H5/`, 2018.

[8] Soumyadip Sengupta. SfSNet Synthetic dataset. home/bsonawane/Thesis/LightEstimation/dataset/SfSNet/Original/Sdata.tar.gz, 2018.

[9] Bhushan Sonawane. SfSNet Synthetic dataset with it's respective SIRFS output. /home/bsonawane/Thesis/LightEstimation/dataset/SfSNet/H5/, 2018.

[10] Zhixin Shu. Cropped CelebA dataset. /nfs/bigdisk/zhshu/FaceDatasets/Imgs2/celebA/, 2018.

[11] Bhushan Sonawane. SIRFS output of CelebA images in h5 format. /home/bsonawane/Thesis/LightEstimation/dataset/CelebA_H5/, 2018.

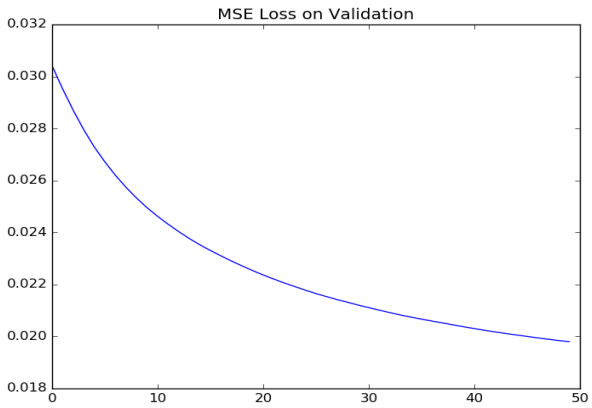
[12] Bhushan Sonawane Jon Barron. SIRFS Matlab Implementation. https://github.com/bhushan23/SIRFS, 2018.

[13] Bhushan Sonawane. Implementation of LDAN. https://github.com/bhushan23/Light-Estimation, 2018.

9 Results Figures

(a) Real Image (b) Lighting Net

Figure 14: Experiment 1: CelebA test image and normal



(a) MSE validation set

Figure 15: Experiment 1: MSE on Validation set with respect to SIRFS SH

(a) SIRFS estimation (b) LDAN estimation

Figure 16: Experiment 1: SIRFS vs LDAN estimation

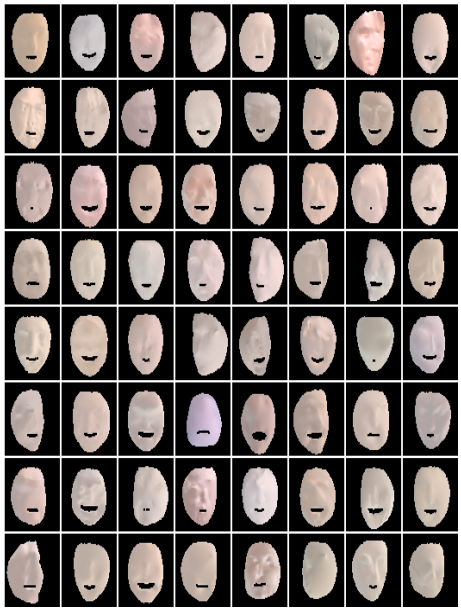
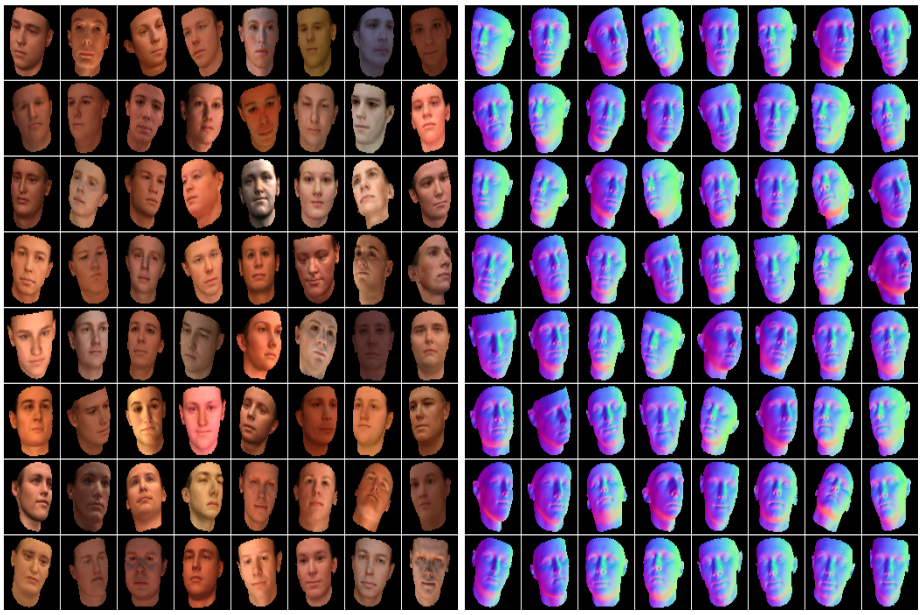


Figure 17: Experiment 1: With no domain adaptation



(a) Real Image (b) Ground truth normal

Figure 18: Experiment 2: SfSNet image used as Real image and Ground truth Normal

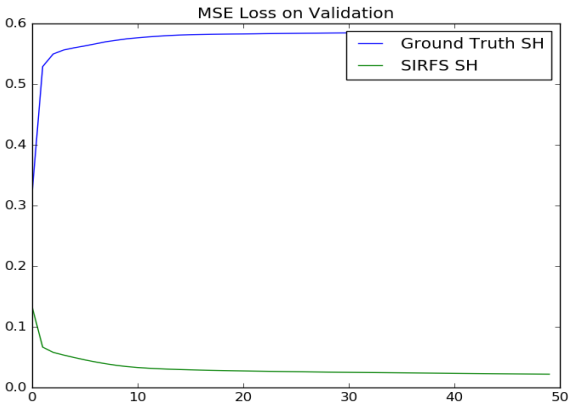


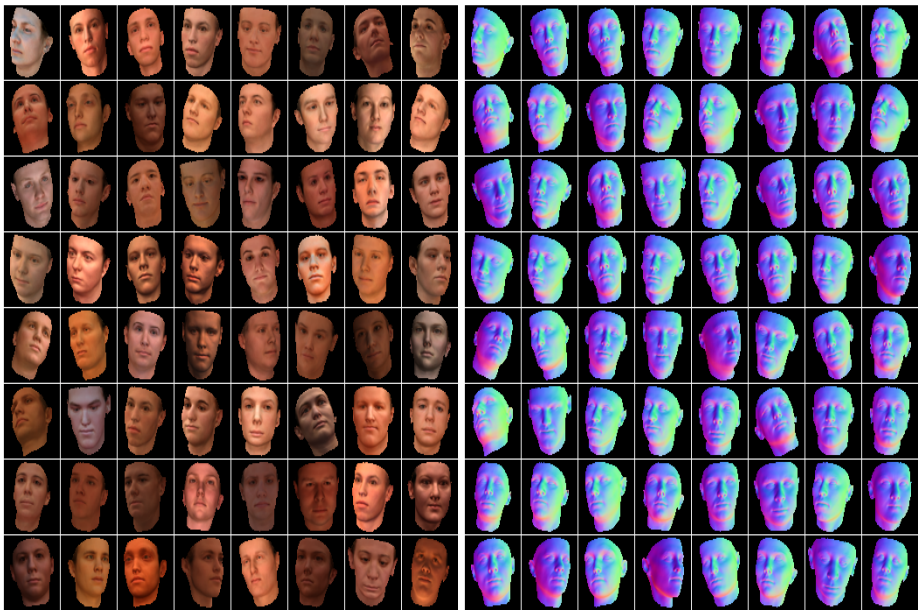
Figure 19: Experiment 2: MSE on Validation set with respect to ground truth and SIRFS SH

(a) Ground truth shading (b) SIRFS estimation

Figure 20: Experiment 2: Ground truth and SIRFS shading

(a) No Domain adaption (b) Domain adaption with ground truth SH

Figure 21: Experiment 2: No Domain adaption vs Domain Adaption with ground truth SH



(a) Real Image (b) Normal

Figure 22: Experiment 3: SfSNet image and ground truth normal

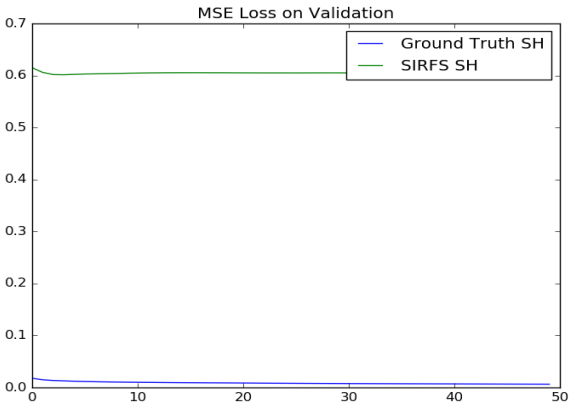


Figure 23: MSE on Validation set with respect to ground truth SH and SIRFS SH

(a) Ground truth shading (b) SIRFS estimation

Figure 24: Experiment 3: Ground truth and SIRFS shading

(a) No Domain adaption (b) Domain adaption with ground truth SH

Figure 25: Experiment 3: No Domain adaption vs Domain Adaption with ground truth SH

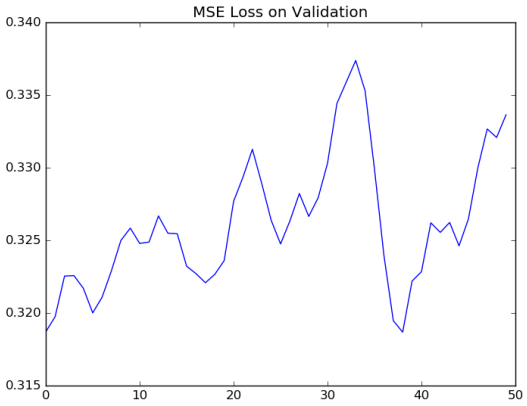


Figure 26: Experiment 4 Auto-Lighting: MSE on Validation set with respect to SIRFS SH

(a) Input Image (b) SIRFS Normal

Figure 27: Experiment 4 Auto-Lighting: CelebA Real image and SIRFS normal

(a) SIRFS estimation (b) Auto-Lighting estimation

Figure 28: Experiment 4 Auto-Lighting: SIRFS estimation vs Auto-Lighting estimation

(a) Auto-Lighting estimation (b) Experiment 1: With out domain adaptation

Figure 29: Auto-Lighting estimation s vs LDAN Experiment 1