IE643: Deep Learning : Theory and Practice **Aug-Dec 2020**

End-term Project Report : The Generative Model Inversion Attack

Team Name: GloVe n Caffe *Team Members: 180070066, 180070057, 193300007*

Abstract

Machine Learning(ML) algorithms have been implemented in reconstructing private sensitive information such as facial recognition, medical diagnostics and predicting life style choices, but the inversion of a DNN is an open problem yet to be addressed. This project essentially studies and implements model-inversion attacks, in which the access to a Deep Learning model is abused to infer information about the training data, partially addressing this open problem. Specifically, we have launched a GMI, or Generative Model Inversion attack on a state of the art face recognition classifier. This report is a synopsis of the whole end-to-end implementation of the attack, and the analysis of the results, and some theory pertaining to the method used.

1 Introduction

In today's world, in which DNN (Deep Neural Network) techniques are adopted in almost all fields, from medicine to signal enhancement, and our personal data being easily exposed, concerns about data privacy are raised. It becomes critical to analyse different model inversion attacks, and update the DNN model privacy standards, to make the models as secure as possible. The gold standard of privacy now a days is the Differential Privacy or (DP), which, according to the paper, can be breached using the proposed attack.
Overall, the GMI attack algorithm will be explored and implemented, from [1], which exploits a very important piece of information, the available distributional prior, and solves an important problem of ill-posedness of the optimisation problem, and prevents the attack to give a local minima output. MI attacks are implemented as an optimization problem seeking for the sensitive feature value that achieves the maximum likelihood under the target model. For example, for a task of inversion of a face recognition classifier, which takes, for instance 64*64 face images and classifies them, for an identity we need to optimise over a 4096 dimensional space, and the error surface is not at all guaranteed to be convex. This is a horrendous task. But, in that 4096 dimensional space, face images would occupy a very small subspace. So, we are wasting time trying to optimise over the whole space, as we can do much better with a small subspace. But the question which remains is, how to identify this subspace? GANs come to our rescue! We can train a GAN such that the generator when trained, learns to produce face images, or outputs a random variable whose distribution as support only over the subspace occupied by the faces, in 4096 dimensions! Once we have this generator, we can optimise over the small space, and invert the model. In this way, we not only reduced the space of optimisation, but also avoided the possibility of being stuck in a local minima which lies outside the face subspace. We cannot have a local minima within the face-space, as that would mean the face recognition classifier is not well built. Thus, Deep Neural Networks help us solve this optimisation problem as well.
The structure of the project report is as below:

We provide a survey of existing literature in Section 2. Our proposal for the project is described in Section 3. We give details on experiments in Section 5. A description of future work is given in Section 7. We conclude with a short summary and pointers to forthcoming work in Section 8.

2 Literature Survey

The privacy attack in ML model consists of some aspects that reveal information about training data.Our particular interest lies in Membership Attack and MI attack,where Membership attack gives information whether a member used in the training data and MI attack gives information on specific feature based on the target label.
In parallel to emergence of various privacy attacks, one dominate Differential Privacy(DP),which carefully randomizes an algorithm such a way such that its output doesn't depend much on individual's training data.DP guarantees in ML models to protect information whether the data recorded is used in training the trained model. The first MI attack was demonstrated in **[2]** , where author presented an algorithm to reconstruct genetic markers,given the linear regression that uses them as input features,the response of the model, as well as other non-sensitive information of feature vector.
[3] proposes an algorithm that carries out MI attack without knowledge of non-sensitive features by poisoning training data.The algorithm presented in the above mentioned papers is restricted only to Linear Model.
[4] discussed an application of the use of MI attack for complex architecture i.e. neural network for face recognition.The algorithm can reconstruct face images with higher probability than random guessing indeed, the facial images are blurred and hard to recognize.

Figure 1: An image recovered using a new model inversion attack (left) and a training set image of the victim (right). The attacker is given only the person's name and access to a facial recognition system that returns a class confidence score

The algorithm of MI attacks is somewhat similar to orthogonal line of work on feature visualisation.**[5]** also attempts to reconstruct an image that activates target network.

Figure 2: Images synthesized from scratch to highly activate output neurons in the CaffeNet deep neural network, which has learned to classify different types of ImageNet images.

Our work differs from the above work on Feature Visualisation in that proposed algorithm adopts a novel optimization technique that results in more realistic image recovery and can introduce auxiliary knowledge of the attacker.

3 Methods and Approaches

3.1 Work done before mid-term project review

Before the midterm project review, our group started with going through the paper [1], and noting down the prerequisite knowledge we need to understand the paper. We learnt that firstly, we need to understand the Generative Adversarial Networks (GANs), as they form a very major part of our project. Through various online tutorials, we understood the working of GANs, the cost function employed in it, and other such details. This being essentially the required prerequisite, we then understood the theory of the paper, and the information about experimentation given as well. This can be broken down as follows:

- Understood the motivation behind the paper
- Performed a literature survey, read briefly about the previous work done in the fields of the paper
- Proceeded to understanding the **GMI** attack, studied the architecture of the attack, the loss functions used in it, and its performance (theoretically)
- Studied the theorem stated in the paper, which directly related the Invertibility of the model, and the predictive power of the model. It proved that the variation is direct.

Once the theory was understood, it was time to move to the implementation. For the implementation, we chose to use the PyTorch library, as recommended by the Professor. We started with going through tutorials and documentations of various components and methods of PyTorch, to gain a grasp on using it. We then proceeded to write the GAN implementation, used for generating training data. We decided to use the **CelebA dataset** for the purpose of training the GAN, as it has aligned and cropped images, with a lot of identities. We coded the two components of the GAN, but we failed to upload the dataset on colab. We tried to unzip the dataset in the code, unzip online, download unzip and upload, none of which worked. Hence, we uploaded just a few images and tried to run the GAN. After some debugging, we were able to run it, even though the generated images were of course not up to the mark. We then planned further proceedings, and there was the midterm review at this point of time.

3.2 Work done after mid-term project review

The main implementation began in this phase. The details are in the following subsection 3.3
Apart from implementation, as a part of our literature survey, and also our scribing assignment, we read and understood the paper [7] which helped us gain insights into Differential Privacy, and algorithms implemented using DP, and how do these protect the training data. The definition for privacy was established rigorously, and other details are included in the scribing assignment.

3.3 The Implementation Setup

The overall setup of the attack pipeline can be described as follows:

3.3.1 Data

We decided to use CelebA dataset, as mentioned. As we faced difficulty in loading the dataset, we requested the Professor to help, and he helped us by setting up an IEOR department server with great computational power which was useful for uploading and using the data (Using WinSCP)

3.3.2 The GAN

We used the GAN implementation similar to that proposed in [6]. The two components, Generator and Discriminator, and the loss function are briefed upon below.

- **The Generator** The generator used takes as input a latent vector of size 100, drawn from a standard normal distribution, and outputs a generated image. The architecture is well explained in the following diagram. The ConvTranspose2d performs transposed convolution, the BatchNorm2d per-

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Figure 3: The Generator Architecture

  forms batch normalization, and ReLU is a nonlinear activation layer. Finally, Tanh activation is used.

- **The Discriminator** The discriminator takes a tensor of size [1,3,64,64] as input, and outputs a real value, which denotes the probability that the input is real. The tensor is an image of [3,64,64], and the first dimension denotes the batch size. The architecture is,

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Figure 4: The Discriminator Architecture

- **The Loss Function** Here, the Wasserstein GAN loss function was used to calculate the loss. This

$$\min_G \max_D L_{\text{wgan}}(G, D) = E_x[D(x)] - E_z[D(G(z))] \quad (1)$$

Figure 5: Wasserstein GAN loss function

function can be interpreted as, the generator trying to fool the discriminator by making him classify the generated image as true, while discriminator is trying to keep up by learning to classify as good as he can.

3.3.3 The Face Recognition Classifier

This is the target model which we will try to invert for a given identity. This model will take as input an image, and return the probability distribution over the identities with which it has been trained. The probabilities will denote the likelihood of the image being of that identity, determined by the model. We designed and implemented this model on our own, the model takes as input a [1,3,64,64] tensor, and outputs a probability vector. The detailed architecture can be understood with the help of the following figure.

```
DataParallel(
  (module): TargetNetwork(
    (conv): Sequential(
      (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): LeakyReLU(negative_slope=0.2, inplace=True)
      (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (4): LeakyReLU(negative_slope=0.2, inplace=True)
      (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (7): LeakyReLU(negative_slope=0.2, inplace=True)
      (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (10): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (classifier): Sequential(
      (0): Linear(in_features=8192, out_features=512, bias=True)
      (1): ReLU()
      (2): Linear(in_features=512, out_features=64, bias=True)
      (3): ReLU()
      (4): Linear(in_features=64, out_features=20, bias=True)
      (5): Softmax(dim=0)
    )
  )
)
```

Figure 6: Target Network architecture

This architecture is inspired from the discriminator of the GAN, and we constructed the network using the same, as the discriminator is well capable of distinguishing whether an image is real or fake, hence is a good classifier architecture. We changed the last layer to accommodate all identities, using a softmax function. It is not the state of the art face recognition classifier, as recommended in the paper for better attack results, but we used this to check the model performance on general models, and see whether it works without using the fact (from the theorem in [1]) that a good target network is more vulnerable to attacks.

3.3.4 The Z vector optimizer

This part of the code performs the following: Given the whole model (Generator and Target model) and an identity, It uses SGD (Stochastic Gradient Descent) optimisation to find a latent vector, which when given as input to the generator, produces an image, which is classified as the given identity with highest probability by the target model. In the paper [1], there was a diversity loss introduced by the authors, which is used by the authors to generate as different looking faces as possible, which is not required in our implementation, as the GAN is inspired from [6] and generates quite realistic and different looking images. Also, for the SGD of the z-vector optimiser, the authors in [1] had used a prior loss, which ensures that the images generated not only are classified the best, but also have some semantic meaning, or in other words, look like real faces. All we need here is the identity loss and the prior loss. Identity loss is the Binary Cross Entropy loss between the probability outputted by the target model, and 1. The identity loss function is given by,

$$L_{id}(Z) = -log(C(G(Z))) \tag{1}$$

The prior loss is just the probability outputted by the discriminator, when input by the generated image. This ensures that the image is real (discriminator is trained for this specific purpose).

$$L_{prior}(Z) = -D(G(Z)) \tag{2}$$

The total loss is a linear combination of the two losses, and this optimiser tries to minimise this over Z. The optimisation problem is given as,

$$\hat{Z} = argmin_z L_{prior}(Z) + \lambda_i L_{id}(Z) \tag{3}$$

3.3.5 The pipeline

The steps can be listed as below

- Train the GAN
- Setup the target model so as to be able to take input from the generator
- Setup the Z-optimiser, which performs SGD on Z
- Create a Z, then perform Z-optimisation as follows:
  - Decide on an identity
  - Generate an image corresponding to Z
  - Input it into the target
  - Take the output and calculate the identity loss
  - Input the image to the discriminator and calculate the prior loss
  - backpropogate with SGD optimiser and total loss, and update Z accordingly
  - Iterate

Once we iterate for a few rounds, we would get a Z vector, which, when inputted to the Generator, would make the generator generate a face image which is very close to the identity of the person.
Thus we have inverted the target!

3.4 The Evaluation Setup

We propose a Novel Evaluation Metric here to evaluate the performance of our GMI attack, or any model inversion attack in general. The metric was termed **The Eigen Accuracy** due to obvious reasons (will be evident from the description below).
Basically, we build a *PCA* based Face Recognition system here. The step-wise methodology to calculate the *Eigen Accuracy* is described below:

1. We build identity-specific eigenspaces for *a sample* of the labels used in the training of the target model
2. We calculate the reconstruction error averaged over the sample. The reconstruction error is the standard reconstruction error as calculated in the PCA based implementations using some suitable number (k) of top eigenvectors only. Mathematically,

$$error =< ||z_p - \bar{x_p} - V_p^k \beta_p||^2 > \tag{4}$$

where,
z_p is the recovered image for the p^{th} identity in the sample
$\bar{x}$ and V_p^k are the mean image and the truncated (top k) eigenvector matrix respectively, of the thus constructed identity specific eigenspace for the p^{th} identity
β_p is the eigencoefficient matrix of the recovered image projected upon the eigenspace for his/her identity.
and the average is taken over p, the sample indices.

3. The Eigen accuracy is calculated as follows:

$$EigenAccuracy(in\%) = \frac{OptimalReconstructionError}{AverageReconstructionError} \times 100 \tag{5}$$

4. The optimal reconstruction error is calculated as the average reconstruction error of the images in the person specific database used for training the proposed classifier, averaged over the sample indices.

The motivation behind the genesis of this metric is that this metric basically provides an unbiased, generalized and unconditional evaluation of the effectiveness of the model.
If the same target network is used to evaluate the attack, the performance will be biased and may give a high accuracy even in the case of the generation of *adversarial examples*, which may be semantically meaningless.
Using another DNN based face recognition classifier for evaluation will also restrict the recovered image to be effective in a specific type of platform used for face recognition. Moreover, such a classifier would lead to incorporation of only the identity loss[1], while calculating the accuracy.
In some sense, our metric amalgamates both prior loss and identity loss, in the perfect proportion, for calculating the accuracy (it penalizes compromise in both directions, optimally).

4 Dataset Details

As mentioned before, we have used the CelebA dataset, which is a set of 2,02,599 face images of 10,177 unique celebrities, all of various ethnicities, genders and races. We had procured a zip file containing a folder having these images, from [8]. The images are RGB, and of size 64*64 pixels. They are face aligned and cropped such that they contain only the face. There are no differences in lighting conditions as well.
As a part of pre-processing, we had used $torchvision.transforms$, and had resized, centre-cropped the images. We then had converted these to the tensor datatype of torch, and had normalized all three channels of the images using 0.5 mean and 0.5 standard deviation. This was all the preprocessing done on the dataset.
To train the face recognition classifier, we used the miniCelebA dataset [9], with 20 identities, and each identity having 100 or more images. In CelebA dataset, each identity has on an average 18 images, hence the dataset trained for the target network is disjoint from the GAN dataset. The images in the miniCelebA also were unclear and of different alignment than CelebA, hence even though their names match, the data distributions were quite different. We performed the same data preprocessing on this data as well.

5 Experiments

We performed extensive experiments to actualize the proposed Generative Model Inversion Attack and implement it on one of the state of the art Face Recognition Models.

We can divide the complete actuation into the following sub-parts:

5.1 The Experimental Setup

The details of the overall experiments concerning the attack pipeline are given below:

5.1.1 The GAN

For training the GAN, we use the standard *Adam optimiser* with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The batch size used for the latent vector z was 128. The learning rate was set to 0.0002.

We executed the training loop for 5 epochs, each epoch going over all of the batches.

5.1.2 The Face Recognition Classifier

For the training of Face Recognition Classifier Model (implemented by us), we used the Adam optimizer and trained the model for 100 epochs, with each epoch going over all batches. The details of the process are as follows:

- learning rate = 0.0001
- batch size = 64
- number of workers = 4
- Loss function: Cross Entropy Loss

5.1.3 The z vector optimizer

In this part, we used a simple SGD optimizer with the following details:

- learning rate = 0.02
- batch size = 64
- momentum = 0.9
- z is initialized from a standard normal distribution

We thus randomly initialize z for 5 times and optimize each round for 1500 iterations, as proposed in the paper.
We select the batch of z with minimum mean *identity loss*[1] and then select the vector with the lowest identity loss in that batch, as the final result, which has produced the *recovered image* in our method.

5.2 The Hardware and Software Configurations

- **Software:** We used Pytorch in jupyter as the platform for most of our implementation - the GMI part. For the implementation of the Eigen Accuracy part, we used the online MATLAB.
- **Hardware:** As a part of our hardware, we used the server setup by our Guide and the Instructor, in the Deep Learning Computational Server of *The Department of Industrial Engineering and Operations Research, IIT Bombay.* The details of the server are specified below:

```
OS: Ubuntu 18.04.1 LTS
Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz, 32 cores, 96GB RAM
Nvidia GeForce(R) GTX 1080 Ti, 24GB
```

Figure 7: Server Details

It had 2 cuda GPUs enabled.

6 Results

The overall results of the project were quite satisfactory and phenomenal at the same time. We present the results according to the implementation pipeline, in a chronological order, here, as follows:

6.1 The Trained GAN

As can be seen in the following graph, five epochs were sufficient to train the GAN effectively.

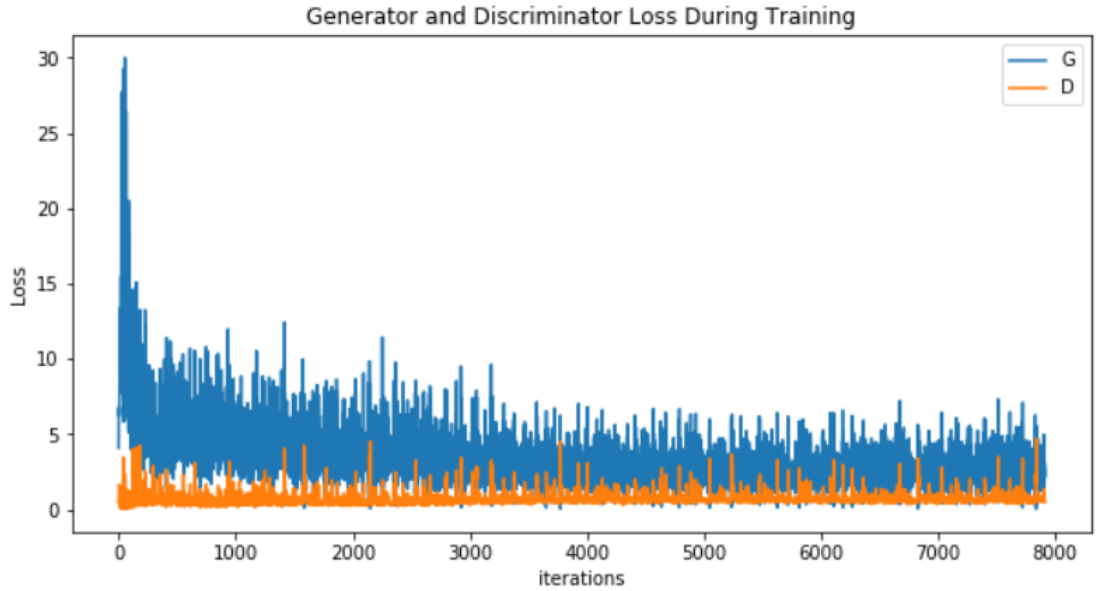

Figure 8: Loss of GAN

Here, we display a specimen image generated by the trained Generator against a random vector (z) taken from the standard normal distribution.

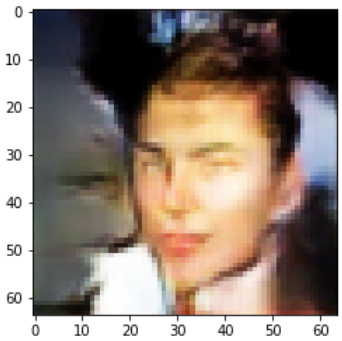

Figure 9: A specimen generated face

We see that the GAN has learnt quite effectively and is producing realistic looking images.

Here, we also display an 8x8 array of Generated images to reinforce our assumption that dcgan does implicitly promote diversity loss.

Figure 10: Faces generated by the Generator

6.2 The Secret Revelation

We performed the GMI attack for four identities out of the miniCelebA dataset used for training the target model. We display below the original and the recovered images, side by side, for the four identities described above.

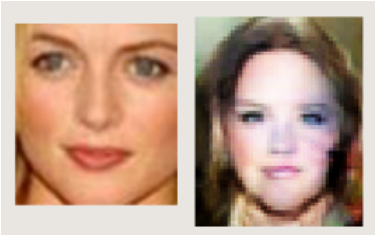

Figure 11: Target Identity 1

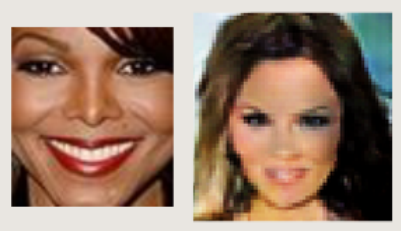

Figure 12: Target Identity 2

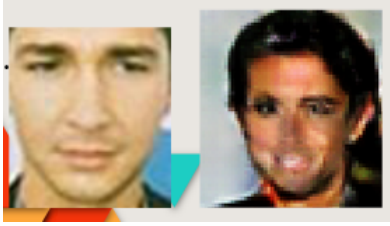

Figure 13: Target Identity 3

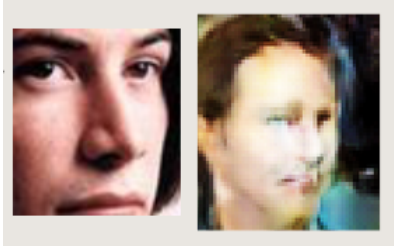

Figure 14: Target Identity 4

We see that our experimentation was quite successful and many of the important facial features were recovered in the process.
The recovered images don't seem that appealing because of the following reasons:

1. The training as well as recovered images were just 64x64 (very low resolution)
2. Our target model is very basic (as we have ourselves implemented it), and it doesn't employ any state of the art facial recognition techniques. And as we know that effectiveness of the recovery is proportional to the predictive power of the model, hence the result.
3. We don't employ any auxiliary information in the attack pipeline as used in the paper[1]. We are

generating the images from scratch.

6.3 The Performance Evaluation

In this part, we use the indigenous evaluation metric, the Eigen Accuracy, for the evaluation of our attack implementation. The details of this implementation are as follows:

- We used sample size = 4, corresponding to the four recovered identities in the secret revelation stage.
- For construction of the identity-specific eigenspaces, we used the same training dataset as used for training our target model.
- The final Eigen Accuracy achieved by our implementation is *67.4%*

We also noted the observation that the Eigen Accuracy is in fact the combined measure of the performance of the attack and model's predictive power.

7 Future Work

We propose the following potential directions which may inspire future works:

- A Generative Model Inversion Attack against a *differentially private* Face Recognition Model (which hasn't been effectively realized till date)
- We can further improve the performance of the GMI attack by employing *race* specific public datasets in the knowledge distillation task (training of the GAN) for specialized recovery of identities with prior knowledge of their race.
  This is inspired by the fact that the distributional prior being learnt by the GAN may be significantly different for the identities belonging to different races.
- The most challenging future endeavour may be the task of designing a possible defense against highly performant MI attacks such as this (GMI).
- Combination of Pure Image Inpainting and GMI may be useful for practical realizations.

8 Conclusions

We see that the proposed method is a phenomenal breakthrough in the field of Model Inversion Attacks, and largely in the field of cyber privacy as DNNs are all over the technology now. As demonstrated in the paper[1], this attack method performs radically better than the previously existing MI attack, even without any auxiliary knowledge.

To be specific, the proposed Generative Model Inversion Attack is effective even if the dataset used for knowledge distillation:

- Doesn't include the identity whose face is to be recovered
- Is unlabelled

- Is small in size
- Comes from a distribution different from that of the private dataset

The novel evaluation metric proposed, The Eigen Accuracy, has been observed and theoretically proved to be robust across platforms and seemingly articulates the attack performance in the most perfect and unbiased way, as opposed to the metrics already proposed in the paper[1].

The practical and overall implications of the theorem

$$\text{model's predicatability } \alpha \text{ model's invertability} \tag{6}$$

are quite fundamental and provide a strong theoretical base to the research going on in the field of Model Inversion Attacks.

The effectiveness of the GMI attack upon *differentially private* models, as demonstrated in the paper[1], poses an open question to design an effective defense against such attack; This points to a potential future prospect.

9 References

9.1 Research Papers

[1] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, Dawn Song: The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks. Published in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)

[2] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin,David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In 23rd USENIX Security Symposium (USENIX Security 14), pages 17–32, 2014

[3] Seira Hidano, Takao Murakami, Shuichi Katsumata, Shinsaku Kiyomoto, and Goichiro Hanaoka. Model inversion attacks for prediction systems: Without knowledge of non-sensitive attributes. In 2017 15th Annual Conference on Privacy, Security and Trust (PST), pages 115–11509. IEEE, 2017.

[4] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pages 1322–1333. ACM, 2015.

[5] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In Advances in neural information processing systems, pages 3387–3395, 2016.

[6] Alec Radford, Luke Metz, Soumith Chintala: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

[7] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 308–318. ACM, 2016.

[8] The CelebA dataset: http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[9] The miniCelebA dataset: https://drive.google.com/drive/folders/1P34eqwjcdMwClz8Wf71-xtDAHq6Jr-6Q?usp=sharing

9.2 Repositories

https://github.com/pytorch/tutorials/blob/master/beginner$_s ource/dcgan_f aces_t utorial.py$