Machine Learning VU - Exercise 3 Report
3.1.2 Model Inversion Attack

Nikola Dragovic, Allan Kálnay, Dušan Trnka

February 2020

Abstract

We recreate model inversion experiments on facial recognition models in the style of Frederikson et al. [1]. Our goal is to generate training images, given that we have whit-box access to a model. We try the attack on a softmax, multilayer perceptron network and a stacked denoising autoencoder model. We were successful in partly recreating the training images. We noticed that the performance of the attack depends on the variance in the training images.

1 Introduction

For this exercise, we chose to recreate experiments in the style of Frederikson et al. [1]. More specifically, we want to recreate the attacks on facial recognition models on the AT&T Laboratories Cambridge database of faces.

We consider the same models as Frederikson et al.: The first one is **Softmax regression**. Softmax regression is basically a neural network with no hidden layers and the output layer consisting of 40 neurons (1 per class), each having softmax as an activation function.

The second one is **Multilayer perceptron network (MLP)**. We use MLP with one hidden layer of 3000 sigmoid neurons and a softmax output layer. This classifier can be understood as performing non-linear transformation to the feature vector at first and then applying softmax regression.

We also tried to implement a **Stacked denoising autoencoder network (DAE)**. For this, we first trained a neural network with two hidden layers of 1000 and 300 hidden units on input images with Gaussian noise to reproduce the original images. Then, we used the first hidden layer, referred to as the encoder, as the input to a softmax regression model for classification.

The task consists of 2 steps. First, we have to train the models on our training set. Second, we have to use the *MI-Face Attack* described in [1] in order to inversely attack the models. The goal of the attack is to iteratively construct training images of each class without any knowledge of the pixels of the training images.

2 Implementation Notes

Our experiments were conducted using Python 3.6 and pytorch/torchvision as the machine learning libraries. The necessary libraries can be installed using Pipenv[1]. In order to ease the use of the model attacking, we have created a CLI interface for this purpose. The `run.py` script running behind the interface trains and attacks a model. The user may specify what model they want to attack. Moreover, the user may also specify other parameters for the attack. The parameters are following: *model type*, *attack's learning rate*, *gamma attack parameter*, *attack iterations*. All the parameters are well documented in the `help` of the CLI script. Note that there is no option to train the models with different parameters. We decided to save the models in the repository in order to save time. For example, the DAE takes multiple hours to train. Also, only one label may be attacked at a time. For the evaluation of all attacks we used the `main.py` script.
Our code submission is structured as follows: the `data` folder contains the orginial AT&T dataset, our train and test set as well as our reconstructed images for each attack and each label. The file to run for the CLI is `run.py`. The attack code is implemented in the attack module of the submission. The model implementation code can be found in the models module. The saved model files are in the resources directory of the code submission.

3 The AT&T Dataset

The link to the dataset given in Frederikson et al. [1] is not working as of June 6, 2020. Therefore, we had to resort to other resources. Thankfully, the dataset was available in a public github repository[2]. We downloaded the AT&T dataset from there.
The dataset consists of 10 pictures each of 40 people, in total 400 images. The images have a size of 112x92 pixels and are grayscale. Frederikson et al. divide their dataset into a training and test set with a ratio of 70% for training and 30% for testing [1]. We manually split the images by using the first 7 images of each person for training and the remaining 3 for testing. For comparison of the recreated images with the target person, we took the first image of each person as a representative.

4 Models

In this section we will talk about how we implemented each model and what we used to get the best accuracy and thus the best chance for a successful attack. The accuracies on the test set of each model is shown in Table 1.

[1] `https://pipenv-fork.readthedocs.io/en/latest/`, visited June 6, 2020
[2] `https://github.com/maheshreddykukunooru/Face_recognition`, accessed June 6, 2020

Model	Accuracy
Softmax	0.95
MLP	0.60
DAE	0.94

Table 1: Accuracies on the test set for each model

4.1 Softmax Model

For the softmax model, we built a model with 10304 (112x92) input nodes and 40 output nodes. We then used pytorch's Stochastic Gradient Descent optimizer to train the model. We train the model in batches of 20 images until there is no improvement for 100 epochs on the training set, just as described in Frederikson et al. [1]. We then use a Softmax layer as a final layer. In practice, this lead us to achieve a accuracy of 1 on the train set. On the test set, we achieved an accuracy of 0.94.

4.2 MLP Model

We implemented MLP as described in [1] – with one hidden layer containing 3000 sigmoid neurons and the softmax output layer. We trained it on the training data with the batches of 20 images and measured accuracy on 280 random sample of the images for every epoch. If the accuracy didn't improve for 100 evaluations or we achieved best possible accuracy (100%), we would stop the training process. We had initially problems with being repeatedly stuck in local extreme with very low accuracy at the training process. We tried to experiment with a learning rate and initial weights to get better accuracy. Relatively long training times has not allowed us to get the model's accuracy higher than $\approx$ 0.60. This later led to bad quality images obtained by attacking the model and wrong classification of the obtained images by the MLP model.

For the model's weights initialization we used `torch.nn.init.xavier_normal_` [3] function. The function randomly samples values from the normal distribution described in [3]. This function with the combination of setting seed for random number generator [4] to 666 (`torch.manual_seed` function)helped us to build our model for 0.60 accuracy on the training data. The model went through 500 epochs during the training time.

4.3 DAE Model

Frederickson et al. do not provide much information how their DAE model was trained. They only provide information about the autoencoder's structure, which is one hidden layer of 1000 sigmoid units and one hidden layer of 300 units, and that the stacked model consisted of a Softmax output layer [1]. Their stacked model was trained until there was no improvement for 100 epochs on the training set. Therefore, we had to make some assumptions about their model

in our experiments.
The training of the Stacked Denoising Autoencoder Model was done in two steps. First, we train a neural network with 10304 input units, and two hidden layers with sigmoid units as described above. We add an output layer of 10304, as we want our neural network to learn recreating the original images. We used the Adam optimizer and Binary Cross Entropy Loss for training the autoencoder model. As the name suggests, the model should also learn to denoise the input image. Hence, we add Gaussian noise with 0 mean and 0.05 variance to the images during training. We settled on this value after some initial experiments with the denoising autoencoder. The model was trained for 1000 epochs. The loss was calculated between the output of the autoencoder and the original, not noisy image. Training the model took a few hours. Therefore, we decided to save the model in order to save time in the successive steps.
We then train a model with 300 input units and 40 output units on the output of the second hidden autoencoder layer until there is no improvement on the training data for 100 epochs. We use a softmax layer for prediction. We achieved an accuracy of 0.94 on the test set.

5 MI-FACE Algorithm

The MI-Face algorithm presented in Frederikson et al. is shown in Algorithm 1. The function AUX-TERM returns a zero vector of the same size as x_i. The PROCESS function performs post-processing to a candidate, but is only relevant for the stacked DAE model. In the cases of the Softmax and MLP model it is defined as the identity function. Just like the authors, we initially conducted our experiments with $\alpha = 5000, \beta = 100, \gamma = 0.99, \lambda = 0.1$ [1]. During the implementation of the algorithm we noticed the following: The attack stops if the cost of the candidate is greater or equal γ. Since the cost of a candidate is defined as

$$c(x) = 1 - \tilde{f}_{label}(x) + \mathbf{0}$$

in our case, the cost is in [0,1]. The smaller the cost, the higher the probability the model classifies the candidate x to belong to class *label*. This would mean that the attack would be terminated if a candidate was found that the model classifies with a probability of more than 0.01 to be of class label.
We consider this to be an error in the paper since the attack would terminate almost immediately in any case. Our belief is strengthened by the explanation of the attack the authors give in the paper: "[..] if the cost is *at least* as great as γ, then descent terminates [..]"(see Frederikson et al. [1] p. 1329). So the cost would have to be greater or equal γ according to the description. This means that the attack terminates if the candidate is classified with a probability of less than 0.01 to be of class label.
We conducted some initial experiments with the adapted line that the attack should terminate if the cost is greater than γ. This lead to unsatisfactory results since the attack would then only terminate if the cost was very high (low probability for class) or no improvement was achieved for β iterations. We do not

believe that this was the authors' intention. In our opinion, the attack should terminate if the cost is low (high probability for class). Therefore, we did not change the original algorithm. Instead, we used $\gamma = 0.01$ in our experiments. For the implementation of the MI-Face algorithm we did not use gradient descent, but stochastic gradient descent available in pytorch. We used a double-ended queue of size β as a storage for the cost of the last β candidates.

Algorithm 1 Inversion attack for facial recognition models, see Frederikson et al. [1]

function MI-FACE(*label*, $\alpha, \beta, \gamma, \lambda$)
  $c(x) \leftarrow 1 - \tilde{f}_{label}(\mathrm{x})$ + AUX-TERM(x)
  $x_0 \leftarrow \mathbf{0}$
  for $i \leftarrow 1 \dots \alpha$ **do**
    $x_i \leftarrow \mathrm{PROCESS}(x_{i-1} - \lambda \cdot \nabla c(x_{i-1}))$
    if $c(x_i) \geq \max(c(x_{i-1}), \dots c(x_{i-\beta}))$ **then**
      break
    end if
    if $c(x_i) \leq \gamma$ **then**
      break
    end if
  end for
  return $[\arg\min_{x_i}(c(x_i)), \min_{x_i}(c(x_i))]$
end function

6 Performance Measures

In our evaluation of the model inversion attack, we do not consider the time it takes to train the classifiers. However, for each of the 40 labels, we measure the time it takes for the MI-Face attack to return a candidate solution. We use three similarity metrics from the `scikit-image` library[3] to compare the reconstructed images with the reference image:

- Structural Similarity Index: This is a value between -1 and 1 that measures the similarity between two images. A value of 1 indicates two identical images, and a value of 0 no structural similarities. It is defined as

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

  with the parameters c_1, c_2 as defined by Wang et al. [5]. We used the default parameters in our experiments.

[3] `https://scikit-image.org/`, accessed June 6, 2020

- Mean Squared Error: A widely used metric to define the difference between a target and a candidate:

$$\text{MSE}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$$

  where n is the length of vector x. Both x and y must have the same shape

- Normalized Root Mean Squared Error: This metric is based on the MSE. The normalization we chose was euclidean norm.

$$\text{NRMSE}(x, y) = \frac{\|x - y\|}{\|y\|}$$

7 Results

We present the results of our experiments below. We report on the average runtime (in seconds) and epochs, as well as average SSIM, MSE and NRMSE measures between the representative images and the candidates generated by the MI-Face algorithm. A summary of the results is shown in Table

Model	Runtime(s)	Epochs	SSIM	MSE	NRMSE
Softmax	1.39	1.11	-0.02	0.22	0.99
MLP	–	–	–	–	–
DAE (identity)	0.5	1.21	0.55	0.03	0.38
DAE (process)	123.4	0.7	0.50	0.03	0.39

Table 2: Average measures for each model for model inversion attack

7.1 Softmax Model

For the softmax model, the generation of target images was very quick, taking only 1.39 seconds on average per image. While the average SSIM value close to zero would indicate that the generated image is not similar to the reference image. However, we were able to generate some images that in our opinion are very close reconstructions of the target person. For an example of a successful reconstruction, we present the images in Figure 1. Nevertheless, for some images the attack was not as successful in generating an image we would recognise as the target. An example is shown in Figure 2.

(a) Reference image

(b) Generated image

Figure 1: Comparison of a successfully reconstructed image and reference image for softmax attack

(a) Reference image

(b) Generated image

Figure 2: Comparison of an unsuccessfully reconstructed image and reference image for softmax attack

7.2 MLP Model

Regarding the MLP model attack, we have approached this one individually. The reason to do this was the fact, that the attack algorithm among others requires *learning rate* parameter, which decides, how big steps the stochastic gradient descent (SGD) takes. If we used the same learning rate for all classes, we would end up waiting for the attack to finish forever. Therefore, for different classes we have used different learning rates. So, we can say, that the learning rate was a crucial parameter from time efficiency perspective and simultaneously from the perspective of reconstructing images as correctly as possible. In concrete terms, certain classes required learning rate ~ 50, whereas certain classes required learning rate ~ 15 or even lower, around 0.01. Unfortunately, at the images reconstruction process, we have not captured the cost metrics of generating the images. We were not able to reproduce this process and capture the cost metrics due to time constrains on our side.

The classes, which required high learning rate parameter value were the ones, which would take non-reasonably huge time amount till the attacking algorithm terminates. The classes, which required very small value of the learning rate parameter were the ones, whose cost function were decreasing reasonably fast with such learning rate value. This means, that the higher value of the learning rate parameter would cause, that the cost function would not converge, because the SGD steps would be too huge to find better feature values.

The comparison of the reference images and the reconstructed images are captured in the Figures 3-6. We have chosen such pairs of images, which clearly show that some images were reconstructed quite well as well as some other were reconstructed enormously poorly. The poor reconstructed image quality was probably driven by the fact, that the accuracy of our model on the test data was just 0.6.

(a) Reference image

(b) Generated image

Figure 3: Comparison of a quite successfully reconstructed image and the reference image for MLP

(a) Reference image

(b) Generated image

Figure 4: Comparison of a quite successfully reconstructed image and the reference image for MLP

(a) Reference image

(b) Generated image

Figure 5: Comparison of a quite successfully reconstructed image and the reference image for MLP

(a) Reference image

(b) Generated image

Figure 6: Comparison of a very poorly reconstructed image and the reference image for MLP

7.3 DAE Model

Different to the previous attacks, the candidate vector x does not have the same size as the training images. Instead, the attack is done on the reduced feature space of size 300 as described by Frederikson et al. [1]. The reduced input space reflects the softmax model on the encoded images. For the attack on the DAE model, we ran two different configurations.
First, we ran the attack as described by the authors with a processing function that denoises and sharpens the candidate in the original feature space. The processing function is shown in Algorithm 2. There is no further explanation on the details of the processing function. We assume that the function NL-MEANS-DENOISE denotes non-local means denoising on the image and applying a sharpening filter. In order to reproduce this, we again used functions available in scikit-image, namely the functions denoise_nl_means from the restoration module and unsharp_mask from the filters module. For the non-local means denoising, we used a patch size of 5 and a patch distance of 6. We used a cut-off distance of 1.15 of the estimated standard deviation of the Gaussian noise. For the sharpening, we used a radius of 5 for blurring and added the double of the difference of the original image and blurred image to the original image.
Our second configuration was using the identity function as the processing function i.e. no manipulation of the candidate during the attack.

Algorithm 2 Processing function for the attack on the DAE model, see Frederikson et al. [1]

function PROCESS-DAE(x)
  encoder.DECODE(x)
  $x \leftarrow$ NL-MEANS-DENOISE(x)
  $x \leftarrow$ SHARPEN(x)
  return *encoder*.ENCODE(x)
end function

The reconstruction of the images in our second setting performed considerably better than the first setting. Not only are the performance metrics SSIM, MSE and NRMSE better, but also the reconstructed images are very similar to the reference images. An example is shown in Figure 7. However, there are also instances where the attack did not reproduce a good image, as can be seen in Figure 8.

The processing function we implemented seemed to have an adverse effect on the success of the reconstruction attack. While the attack not only took longer, the attack often terminated early, leading to only a handful of successful reconstructions. Often the generated image was the same for multiple labels. We do not know if the issue is in our input, in our model or in our processing function. An example for a reconstructed image where the attack did not terminate early is shown in Figure 9. We also show an example for where the attack terminated

(a) Reference image

(b) Generated image

Figure 7: Comparison of a successfully reconstructed image and reference image for DAE attack with no processing

(a) Reference image

(b) Generated image

Figure 8: Comparison of an unsuccessfully reconstructed image and reference image for DAE attack with no processing

early in Figure 10.

8 Analysis of the Results

In general, we noticed that reconstruction was often successful when the training images for a person had low variance, i.e. the images were similar with regard to the position and orientation of the face. All of the training images also contain images with glasses for people that do not wear glasses in their first image (reference image). This did not affect the reconstruction as much as the variance of the training images for a person.
For comparison, we show the reconstructed images for a person where the training images were very different with regard to the position and orientation of the face in Figure 12.

For the softmax model and the stacked DAE, the attack was more successful than for the MLP model. The reason is that we could not recreate a MLP with the settings the authors described that achieves an accuracy on the test set of 0.95. As we have already mentioned, our MLP model was not able to achieve more than 0.6 accuracy on training data set.
For the stacked DAE, our attack without the PROCESS-DAE function was more successful than with it. We do not know where the difference in performance

(a) Reference image

(b) Generated image

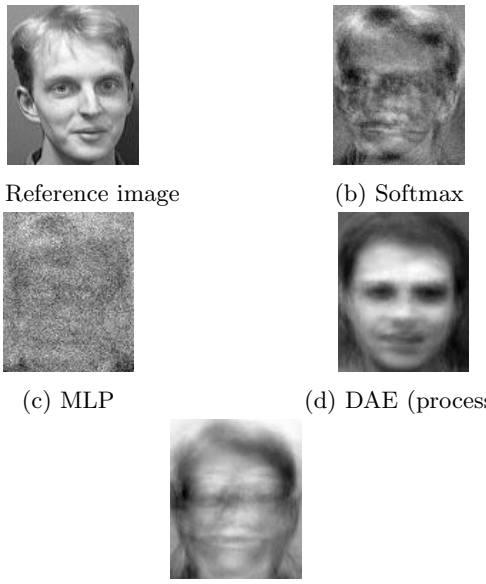
Figure 9: Comparison of a successfully reconstructed image and reference image for DAE attack with processing

(a) Reference image

(b) Generated image

Figure 10: Comparison of an early terminated attack for DAE attack with processing

stems from. Regarding runtime, the attack on the softmax model was the fastest, followed by the attack on the stacked DAE. The attack on the stacked DAE with processing took considerably longer than without it and lead to worse reconstructed images. Note that we did not perform parameter tuning for the attack and used the parameters as suggested by the original authors.
In conclusion, we were able to show that given white-box access to a facial recognition model, the training images can be recreated using the MI-FACE attack described by Frederikson et al. [1].

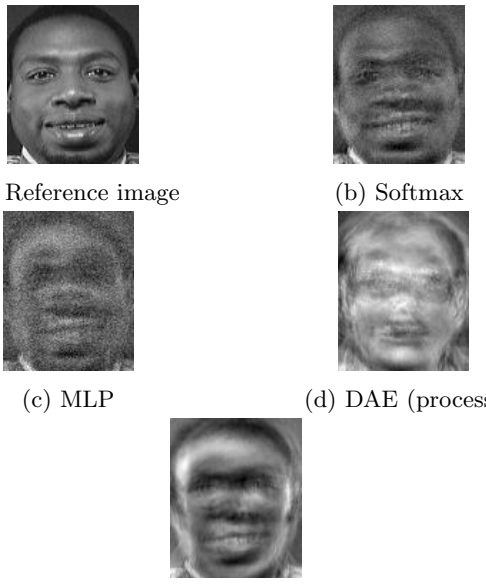


(a) Reference image (b) Softmax

(c) MLP (d) DAE (process)

(e) DAE (identity)

Figure 11: Comparison of reconstruction of an target with high training data variance



(a) Reference image (b) Softmax

(c) MLP (d) DAE (process)

(e) DAE (identity)

Figure 12: Comparison of reconstruction of an target with low training data variance

References

[1] Fredrikson, M., Jha, S. and Ristenpart, T., 2015, October. Model inversion attacks that exploit confidence information and basic countermeasures. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (pp. 1322-1333).

[2] AT&T Laboratories Cambridge. The ORL database of faces. `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`

[3] `https://pytorch.org/docs/stable/nn.init.html#torch.nn.init.xavier_normal_`

[4] `https://pytorch.org/docs/stable/torch.html#torch.manual_seed`

[5] Wang Z.,Bovik, A. C., Sheikh, H. R. and Simoncelli, E. P., "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, Vol. 13, no. 4 (pp. 600-612), April 2004