Neural Networks and Deep Learning - Homework 2

Unsupervised Deep Learning

Davide Garbelotto

January 2021

1 Introduction

The task is to study some unsupervised deep learning problems through the implementation of different architectures. The dataset used for all the experiments is the MNIST, which contains 70.000 images of hand-written digits. The starting point is a convolutional autoencoder for image classification, whose performance have been analysed varying the size of the encoded space. Then, some noise is added to the input images to implement a denoising autoencoder. Furthermore, a classification network is built, fine-tuning the encoder previously implemented as a feature extractor. Eventually, a more complex Conditional Generative Adversarial Network (CGAN) is created, and some realistic samples have been successfully generated. An important part of the work has been dedicated to the study of the latent spaces obtained by the different architectures, in particular noticing the difference between the basic autoencoder and the advanced GAN.

2 Convolutional Autoencoder

The idea is to develop an encoder and a decoder, both made of convolutional layers, which can exploit the spatial information of the input images. The encoder takes in input the 28x28 pixels image and encodes it into a reduced latent space, whose size is one of the hyperparameters to study. This compressed code is then fed to the decoder, which should give back a reconstructed image almost equal to the input one. The most interesting feature of autoencoders is that they are completely unsupervised, since they use as labels the same input images.

2.1 Methods

The proposed structure for the encoder network is made of three convolutional layers, with 16, 32 and 64 filters of size 3x3, followed by a fully connected layer which outputs the encoded input. Exponential Linear Units (ELU) activation functions are interposed between the layers, and two batch normalization layers are used after the first two layers. The batch normalization stabilizes the training and allows to avoid strong regularization.

The loss function chosen for this task is the Mean Squared Error, which computes the difference pixel by pixel between the reconstructed image and the input one. Different optimizers have been tested, among which the SGD and the Adagrad. Eventually, the Adam outperformed the others and was therefore employed for the task. The hyperparameter optimization has been performed through grid search on k-fold cross-validation. The training set has been split into 4 folds, where each one has been used as a validation set for each set of hyperparameters. The most interesting hyperparameters to optimize, beside the standard ones like batch size, learning rate and weight decay, are encoded space dimension and activation function. Indeed the former one represents the rate of compression of the input image, since from the encoded input the decoder has to extract all the information to reconstruct the initial image. Hence, varying it means varying the compression rate of the network, and good results have been achieved also with high compression rates. The activation function, instead, influences deeply the quality of the reconstructed image. Between Relu, Leaky Relu, Tanh and Elu, the latter was the one which performed better. The number of epochs has been managed through a basic early stopping algorithm.

2.2 Results

The results strongly depend on the encoded space dimension, because a bigger encoded size allows to store more information to be used in the reconstruction process by the decoder. For this reason, the reconstruction error has been studied considering the variations of the encoded dimension in the range [2,40]. Figure 1 shows this correlation. It can be observed that the MSE loss decreases exponentially as the encoded space size increases linearly. The reconstruction results, which can be examined in Figure 2, show that the reconstruction images are of good quality, also for low space dimensions.

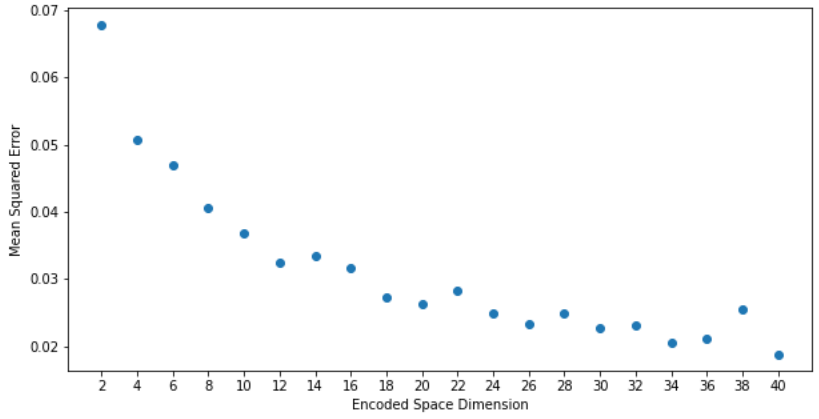


Figure 1: Reconstruction error vs encoded space dimension

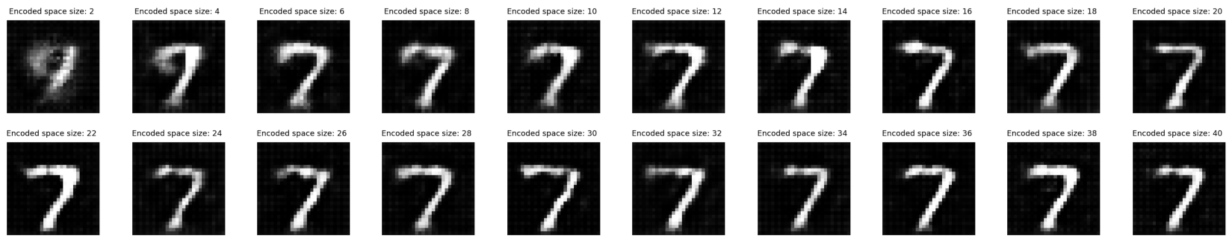


Figure 2: Reconstructed images vs encoded space dimension

The final training has been performed with an encoded dimension of 20. The learning plot and a sample of the reconstructed images are reported in the Appendix, in Figure 7 and 8. The MSE reconstruction loss reaches values around 0,011.

3 Network Analysis

The encoded space produced by the encoder, after the final training, has been analysed. Indeed through the application of the Principal Component Analysis technique, the dimensionality of the encoded space has been compressed to the two biggest dimensions. In this way it has been possible to plot them in a bi-dimensional space, observing how the different classes are represented. The plot is reported in the appendix in Figure 9.

From the plot is possible to notice that some digits are more easily distinguished by the network, whereas others like 3, 5 and 8 are mostly overlapping and are therefore often confused. Moreover, it has been tried to sample in a random way, both through uniform and normal distributions, from the encoded space to feed the decoder and reconstruct an image. Most of the times the results don't present meaningful shapes resembling the digits images. Yet, in some cases, better images are produced by the decoder. An example is shown in Figure 3. This is probably due to the fact that the encoded space produced by the autoencoder is not regularized, so a little change in the latent space can modify completely the decoded image. This problem might in theory be solved by a variational autoencoder.

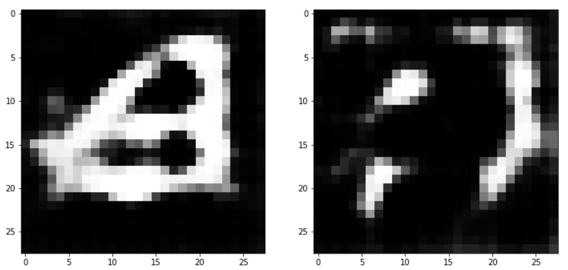


Figure 3: Random sampling from autoencoder latent space

4 Classification Task

The trained encoder is then used as a feature extractor for the digit classification problem. It's been fine-tuned by adding three fully connected layers, the third being the output layer of the classification network. ReLU has been used as activation and a batch normalization layer has been interposed between the first two. Being a classification task, the loss function, in this case, is a Cross-Entropy Loss. The optimizer is, once again, the Adam one. The fine-tuning has been implemented by freezing the trained encoder and involving in the learning process only the layers of the classifier. Although the performance of the classifier network are quite good, they are in general worse than the ones obtained in Homework 1, where a standard convolutional network has been implemented. Indeed, in this case, the Cross-Entropy Loss is around 0.1, against the 0.02 of the previous, whereas the accuracy is 0.9694 against 0.9931. Nevertheless, it has to be taken into account that the encoded space dimension of 20 is much smaller than the input image shape, which is 28x28, and if the former was increased, the classification would probably generate better results. In Figure 4 some misclassified samples are reported.

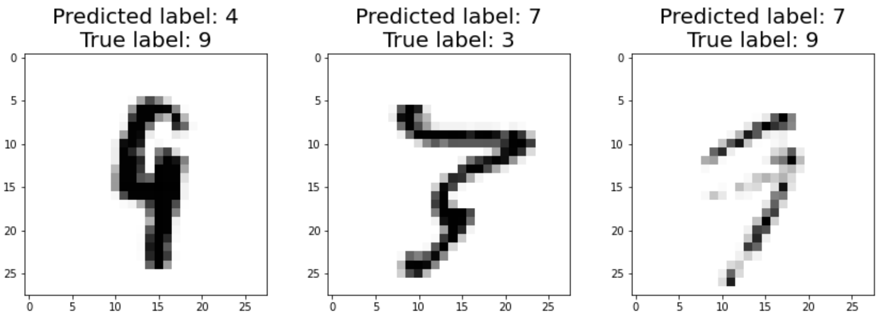


Figure 4: Misclassified images

5 Denoising Autoencoder

In this denoising task, the network structure, the hyperparameters, the loss function and the optimizers are the same as the previous autoencoder. The goal is to corrupt the input images with some noise and train the autoencoder to "clean" the image from the noise, producing the denoised input image. The corruption applied to the images is a normal random noise, with zero mean and varying standard deviation. In general, the denoising autoencoder revealed to be efficient, even with relevant levels of noise. An example is shown in the following Figure 5, where the standard deviation of the normal noise is 0.5. The input noisy images are difficult to be recognized, but the autoencoder is able to map them quite efficiently.

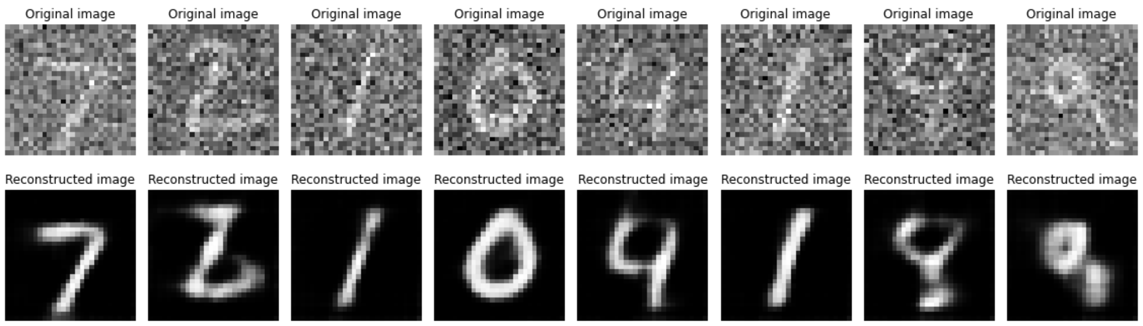


Figure 5: Denoising autoencoder reconstructed images

6 Conditional GAN

Eventually a more complicated unsupervised network has been implemented: a Conditional Generative Adversarial Network. In fact, the structure is not completely unsupervised, but can be considered semi-supervised, because the labels are partially used during the training. Indeed the Conditional version of GANs feeds in input to the networks also the labels, giving the possibility to sample in a more controlled way from the latent space.

6.1 Introduction

The main goal of GANs is to build a Generator network, producing images sampling from a latent space, and a Discriminator, which is a standard classification network, responsible for classifying whether the images are produced by the generator or are real images from the dataset. The two networks are jointly trained in a zero-sum game, where basically one gains when the other loses. In fact, the generator loss decreases when he's able to fool the discriminator, which has instead a lower loss when he's able to recognize which images are generated by the generator. GANs are indeed hard to train because the two networks have to be balanced such that none completely prevails on the other. This, in fact, can cause some issues, for example, if the discriminator reaches an accuracy of 1, the generator is likely to have a vanishing gradient problem because the gradient of the loss function goes to zero. Another issue related to the training of GANs is the so-called mode collapse, which happens when the generator finds a class of images that always fools the discriminator. It therefore starts to generate only images from that class, minimizing its loss, but collapsing its generating abilities to a single type of output.

The Conditional version of GANs is able to solve the mode collapse issue. Indeed, in this kind of networks, both the generator and the discriminator receive in input the class labels, beside the input images. In this way they learn to associate a label with a specific class of images. Once the training is completed, to generate an image from a certain class is therefore sufficient to feed the generator with the label of that class.

6.2 Methods

6.2.1 Generator Network

The generator network is made of a linear layer, followed by three *ConvTranspose* layers, and ends with a Tanh activation function, to constrain the output values to be between -1 and 1. It contains also an embedding layer, which maps the encoded labels into a bigger layer, whose dimension is an hyperparameter of the network. The generator receives as input a noise vector, sampled from a normal distribution with standard deviation equal to 0.1, which is concatenated to the embedding of the class label. This resultant vector is then passed to the *ConvTranspose* layers, which perform the opposite transformation of convolutional ones. These layers have a decreasing number of convolutional filters: 128, 64 and 32, all sized 4x4. They also have stride equal to two, performing an upsample of the activations until their size reaches the one of the output image, which is 28x28.

6.2.2 Discriminator Network

The discriminator structure presents three convolutional blocks, specular to the ones of the generator. They have indeed 32, 64 and 128 4x4 filters, and a stride of 2 which downsample the activations. In this case, the label information is encoded through an embedding layer which outputs a 28x28 tensor, which is then concatenated to the input image as a second channel, and passed to the convolutional blocks. The final linear layer, which follows the convolutional part of the discriminator network, outputs a single value, which is rescaled in the probability range through a Sigmoid activation function. It indeed represents the probability that the input image is a real image, i.e. the probability that it has not been created by the generator.

6.2.3 Hyperparameters, optimizer, loss function and initializations

Both structures present *BatchNorm* layers interposed between every convolutional block, and they both use Leaky ReLU activations.

The weights of the convolutional and batch normalization layers have been initialized through an apposite function, which assigns values from a normal distribution.

Moreover, both the discriminator and the generator use Adam optimizer.

The loss functions are of different types for the two networks. The discriminator loss is computed as a Cross-Entropy on the classification of images, whereas the generator one is a Mean Square Error loss on the discriminator activation features, as it is explained in the next paragraph.

Several combinations of learning rates have been tested, keeping in mind that the learning of the two networks has to be balanced, otherwise the losses are going to diverge. Eventually, the chosen learning rates are 0.0003, both for the generator and the discriminator.

A consistent part of the work has been dedicated to the hyperparameter setting, especially because GANs are extremely sensitive to their changes. The optimization has not been performed through a grid or random search, because the time required for the training was too large. The main hyperparameters which have been manually tuned are the latent space dimension, together with the embedding size representing the label class, and the intensity of the decreasing noise, besides the standard ones like learning rates and weight decay.

6.2.4 Advanced methods

Different techniques have been applied in order to ease the training of the GAN. The first method to facilitate the training is adding an exponentially decreasing noise on the images fed to the discriminator, both the real and the fake ones. This is done to give an advantage to the generator in learning how to produce better quality images before the discriminator is able to recognize them with perfect accuracy. In this latter case, indeed, the network would likely suffer from the aforementioned vanishing gradient issue.

Another important approach implemented is the so-called "Feature Matching" and it slightly changes the way in which the generator is trained. Indeed, in a normal GAN, the generator loss depends on how well the discriminator detected the fake images produced by the generator, and it is usually computed through a Cross-Entropy Loss. In the case of feature matching, instead the generator loss is computed as the Mean Squared Error over the mean of the intermediate activation features produced by the discriminator network when processing real images and fake images. By minimizing the distance between these two, the generator is able to produce images, which are closer to the real ones, at least from the discriminator point of view. An important aspect of this way to compute the loss is that the generator is no more trying to maximize what the discriminator minimizes, and therefore even if the discriminator reaches an accuracy of 1 on the classification, the generator can still go on learning. Indeed the generator loss is no more obtained by the classification output, but from the intermediate activations. Some tests have been done combining both this feature matching loss and the standard one, yet, the best results have been produced when only the feature matching loss is computed and backpropagated.

6.2.5 Results

Several tests, changing the hyperparameters, the network structure and implementing variations of the approaches explained above have been performed. The produced results have been mainly unstable and only some digits were learnt in a precise way. Eventually, with the right set of hyperparameters, and the network structure detailed above, the generator has been able to learn all the digits in a reliable way, even though the borders are not really precise and appear a bit blurred. This can be appreciated in Figure 6, where a sampling from the latent space has been performed. The sampling is different from the one performed in the latent space generated by the autoencoder, which presented some issues. Indeed, this time is enough to feed the generator with a normal noise vector and the label for the digit desired, and the generator will produce an image from that class.

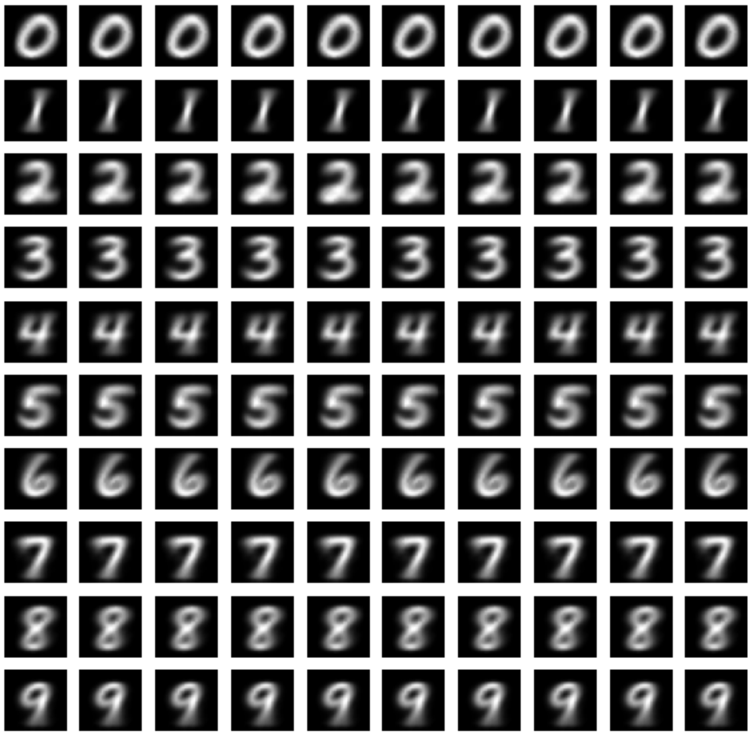


Figure 6: Caption

The learning plots with the generator and discriminator losses and the discriminator classification accuracy, both for real and fake image, are reported in Figures 11 in the Appendix. Eventually, an example of images generated by the GAN, in the case where the feature matching was not applied is reported in Figure 10 of the Appendix. It can be clearly seen that some digits, like 0, 1 or 7 have been learnt by the network in a decent way, whereas some others like 8 and 9 present more problems in being generated.

Appendices

A Convolutional Autoencoder

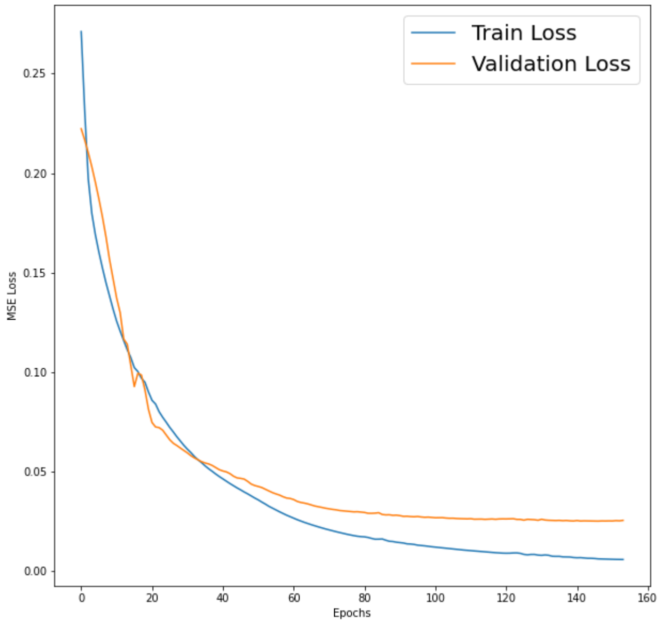


Figure 7: Training plot

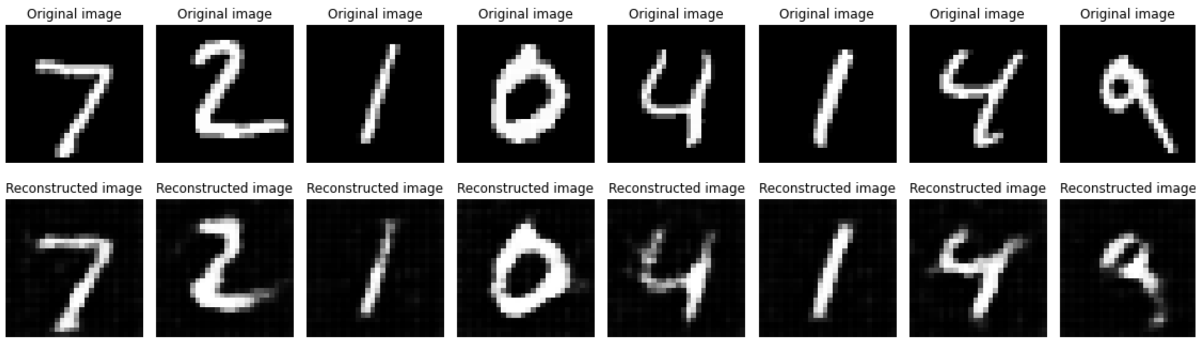


Figure 8: Reconstructed images with encoded space dimension of 20

B Network Analysis

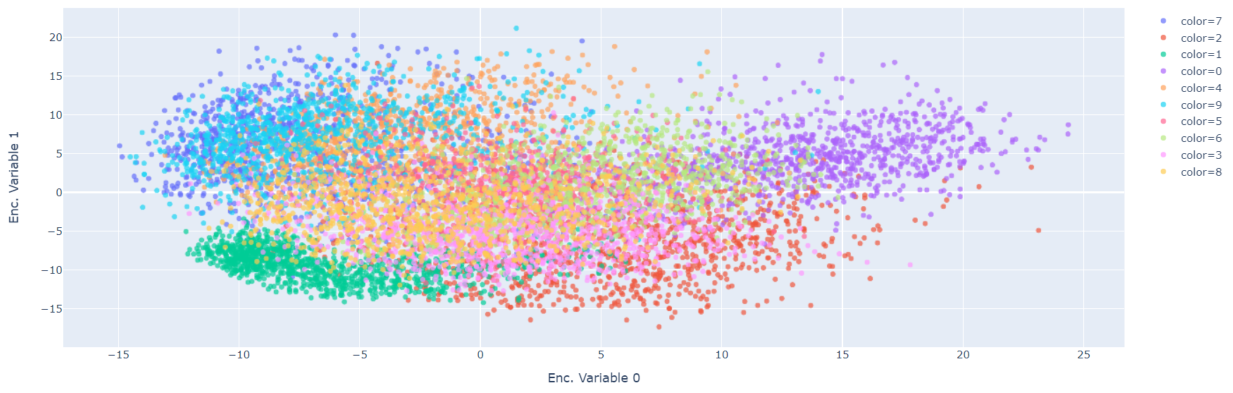


Figure 9: Encoded space PCA analysis

C Conditional GAN

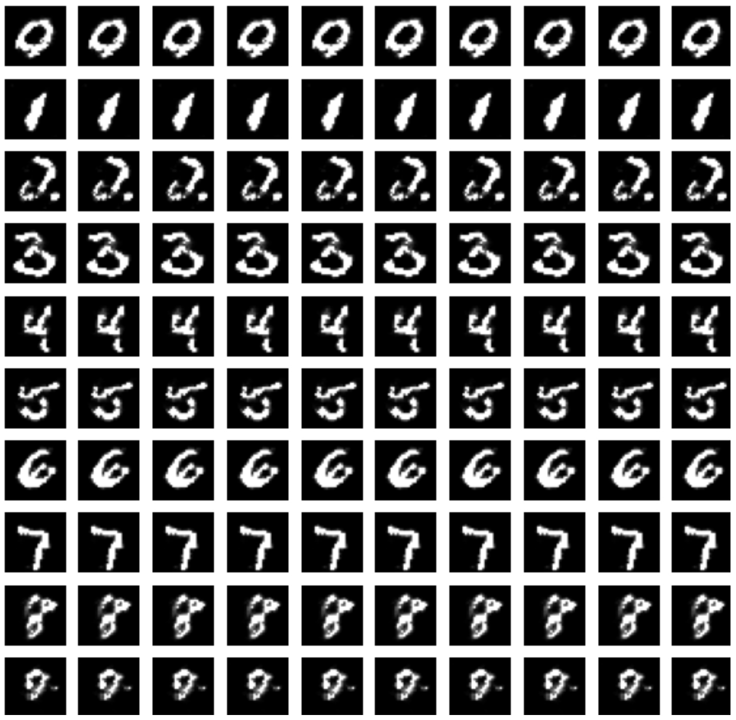
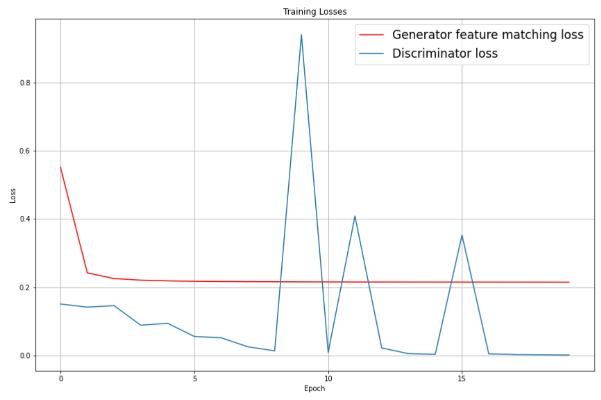
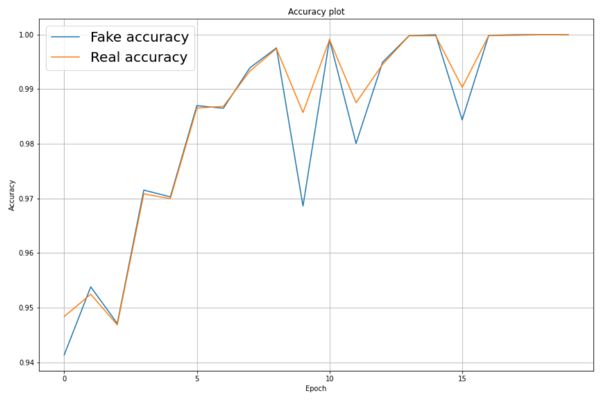


Figure 10: Generated images without feature matching



(a) Losses



(b) Discriminator classification accuracy

Figure 11: Training plots