Title:

Assignment 3,

Image to Sketch by AutoEncoders

Department of Computer and Data Sciences

Advisor:

Dr. Kheradpisheh

Student:

Fatemeh Saberi Khomami

Student Number:

400422114

1. Introduction

Autoencoders are used for denoising data, image colorization, data compression, image generation and many more tasks. Through the process of compressing input data, encoding it, and then reconstructing it as an output, autoencoders allow you to reduce dimensionality and focus only on areas of real value. Every Autoencoder have two main parts. An encoder and a decoder.

The encoder part converts the input data into a latent vector which is only comprehendible by machines.

The decoder part reconstructs the desired output out of the latent vector.

In this project we are keen to use Autoencoders as a system which converts images to sketches.

Before we proceed, it is worth to mention that there were so many trials and runs of different strategies for preparing the data, different architectures, and ideas which most of them lead to failure, but it taught us to gradually find a better model. For the sake of the efficiency, we will exclude our vast experiences of failure from this report, but you can find their corresponding notebooks in a folder called 'experience notebooks' in the repository of this report.

The models inside this report were run on GTX 1050ti with 2.4 GB available GPU ram.

2. Getting to know the Data

We have two sets of data, one with 188 images of Chinese students and the other has 1194 images. Each of them has their corresponding sketches. The second one is called the Feret dataset.

Although in the notebooks located in 'experience notebooks' folder, you can find some cases that we merged the datasets and fed the merged data to various models, but we are going to talk about two separate works on two separate datasets. Let's start with the Chinese students' dataset.

2.1. Chinese Dataset

Some of the Chinese images were paired with the wrong sketch which we corrected it manually.

We imported the data with the shape of 144*120. Then, as an augmentation technique, we added the equalized images to the dataset. Histogram Equalization is a computer image processing technique used to improve contrast in images. Now we doubled our data from 188 to 376.

After that, we added the flip version of the images to the dataset.

You can see some of the images and sketches along their flipped version in the figure below.

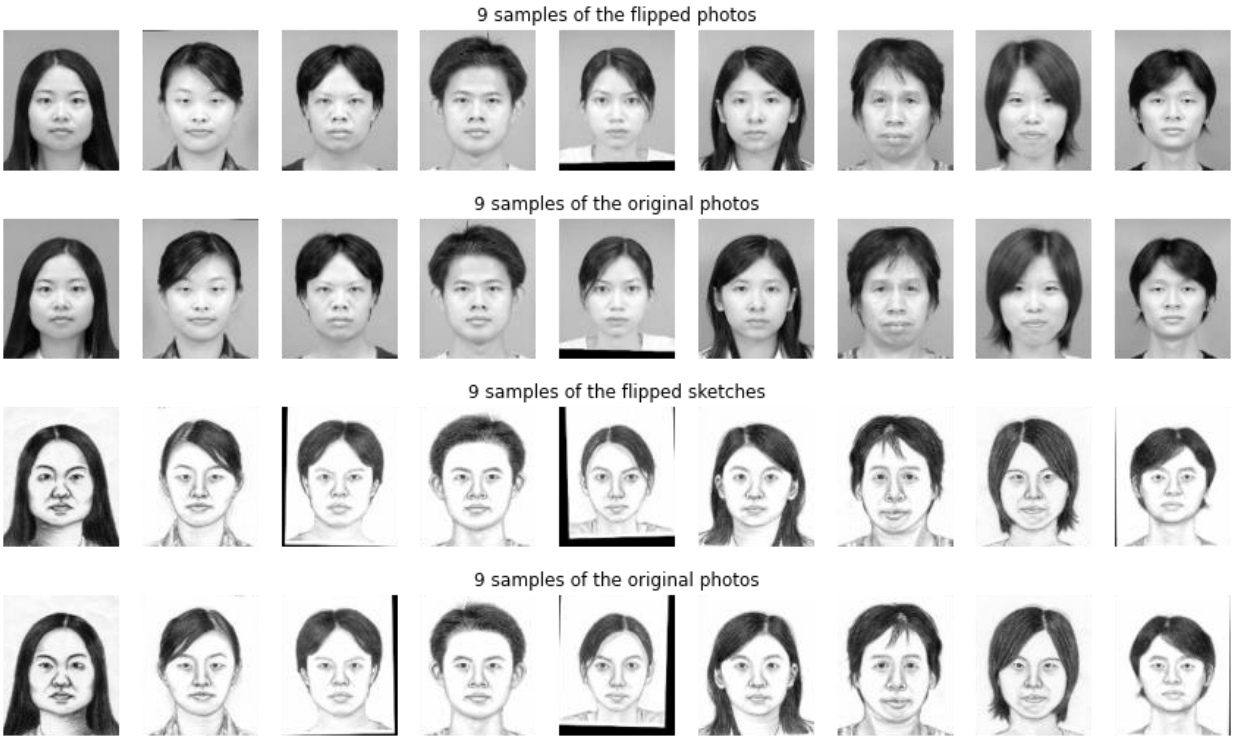

Figure (1): Visualizing some of the images and sketches alongside their flipped versions

To further augment the data, we added clockwise and counter-clockwise $\frac{\pi}{8}$ rotation to the data.

Our final shape of data is 2256. Now we normalize the images by dividing them by 255.

Now we split the data into test and train sets. Here are the shapes:

	SHAPE
TRAIN_PHOTO	(2233, 144, 120, 1)
TRAIN_SKETCH	(2233, 144, 120, 1)
TEST_PHOTO	(23,144,120,1)
TEST_SKETCH	(23,144,120,1)

Table (1): Shape of train and test images and sketches

2.2. Feret Dataset

This dataset has two kinds of sketches, cropped sketches and original sketches. In the prior notebooks located in 'experience notebooks' folder, you can see that we proceed with the cropped sketches, but this report is about using the original sketches. We imported the images in shape 128*128 and sketches in shape 168*160, so that we can crop the margins a little bit. Then, by OpenCV's cv2.normalize, we increased the contrast and brightness of the face images. Some of the results are depicted in the figure below.

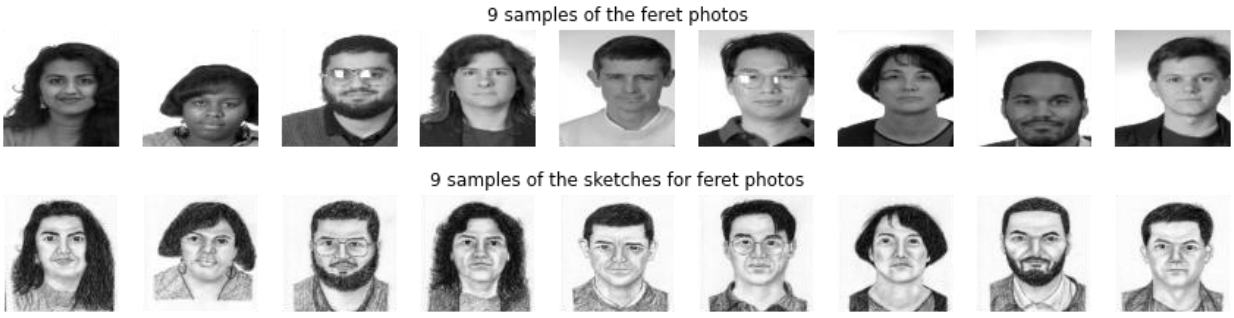

Figure (2): Visualizing some of the improved images along with their corresponding sketches

As an augmentation technique, we added the filliped photos to the images and sketches. To further augment the data, we added clockwise and counter-clockwise $\frac{\pi}{8}$ rotation to the images and sketches.

At last, for normalizing the images, we divided the matrices of photos and sketches by 255.

Now we split the data into test and train sets. Here are the shapes:

	SHAPE
TRAIN_PHOTO	(7020, 128, 128, 1)
TRAIN_SKETCH	(7020, 128, 128, 1)
TEST_PHOTO	(144,128,128,1)
TEST_SKETCH	(23,128,128,1)

Table (2): Shape of train and test images and sketches

3. Begin Training

3.1. Chinese Dataset

The successful model had the below architecture:

Encoder:		
Conv2D	Kernel size: 5*5	32 filters
Conv2D	Kernel size: 3*3	32 filters
Conv2D	Kernel size: 3*3	32 filters
Conv2D	Kernel size: 3*3	32 filters
Maxpooling	Kernel size: 2*2	
Conv2D	Kernel size: 3*3	64 filters
Conv2D	Kernel size: 3*3	64 filters
Maxpooling	Kernel size: 2*2	
Conv2D	Kernel size: 3*3	128 filters
Conv2D	Kernel size: 3*3	128 filters
Maxpooling	Kernel size: 2*2	
Conv2D	Kernel size: 3*3	256 filters
Decoder:		
Conv2DTranspose	Kernel size: 3*3	128 filters
Conv2D	Kernel size: 3*3	128 filters
Conv2DTranspose	Kernel size: 3*3	64 filters
Conv2D	Kernel size: 3*3	64 filters
Conv2DTranspose	Kernel size: 3*3	32 filters
Conv2D	Kernel size: 3*3	32 filters
Conv2D	Kernel size: 5*5	1 filter

Table (3): Model architecture

From the input image with shape 144*120*1, the information converted into a latent vector with shape 18*15*256. From there, we start upsampling to reconstruct an output sketch with shape 144*120*1. For the training we used Adam algorithm as the optimizer with learning rate 0.001. Then we compiled the model with binary_crossentropy loss. After 1,000 epochs, here is the result of the last epoch:

loss: 0.2549 - val_loss: 0.2606

In the following figure, you can see the performance of the model.

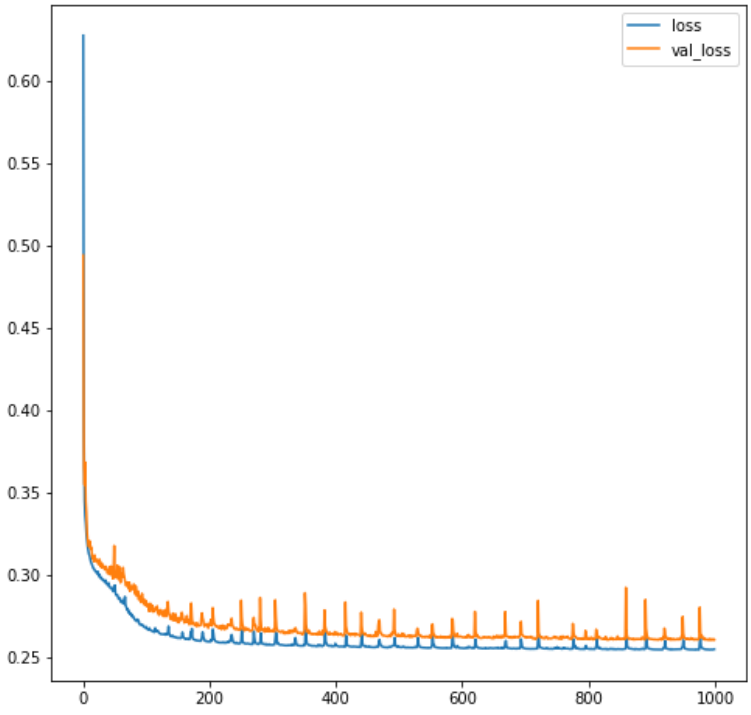

Figure (3): Validation loss vs training loss

Here are some sample predictions of the model on the test set:

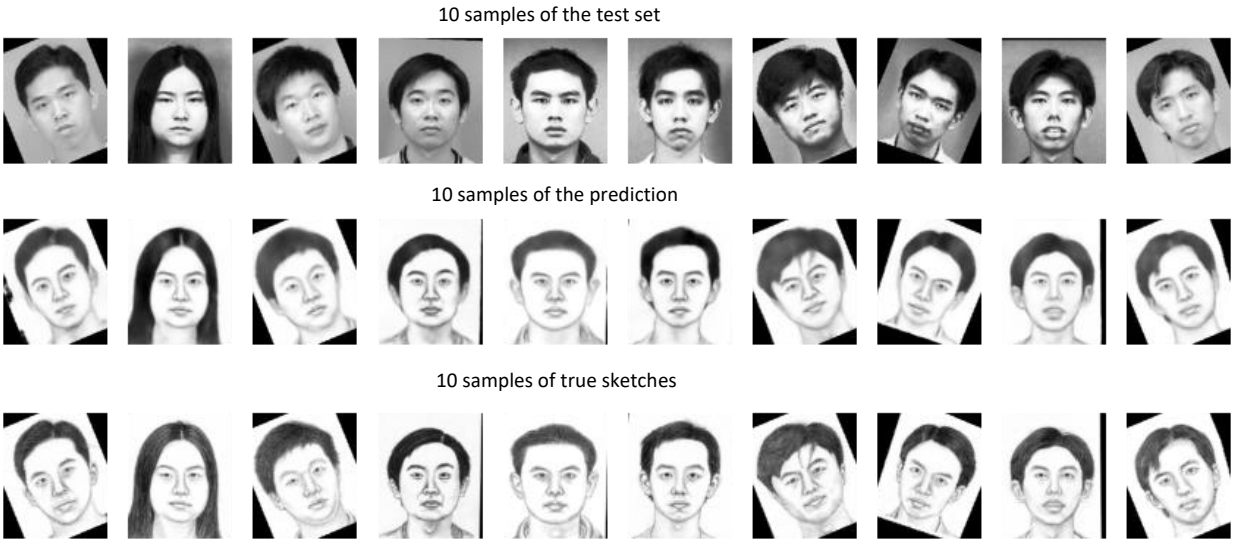

Figure (4): Predictions made by the model alongside the real images and sketches

You can see that after 1,000 epochs, our model has a satisfying performance.

3.2. Feret Dataset

The first model had the below architecture:

Encoder:		
Conv2D	Kernel size: 4*4	16 filters
Conv2D	Kernel size: 4*4	32 filters
Conv2D	Kernel size: 4*4	64 filters
Conv2D	Kernel size: 4*4	128 filters
Conv2D	Kernel size: 4*4	256 filters
Conv2D	Kernel size: 2*2	512 filters
Decoder:		
Conv2DTranspose	Kernel size: 2*2	512 filters
Conv2DTranspose	Kernel size: 4*4	256 filters
Conv2DTranspose	Kernel size: 4*4	128 filters
Conv2DTranspose	Kernel size: 4*4	64 filters
Conv2DTranspose	Kernel size: 4*4	32 filters
Conv2DTranspose	Kernel size: 4*4	16 filters
Conv2DTranspose	Kernel size: 2*2	8 filters
Conv2DTranspose	Kernel size: 2*2	1 filter

Table (4): Model architecture

From the input image with shape 128*128*1, the information converted into a latent vector with shape 1*1*512. From there, we start upsampling to reconstruct an output sketch with shape 128*128*1. For the training we used Adam algorithm as the optimizer with learning rate 0.001. Then we compiled the model with binary_crossentropy loss. After 100 epochs, here is the result of the last epoch: loss: 0.3607 - val_loss: 0.4052

In the following figure, you can see the performance of the model.

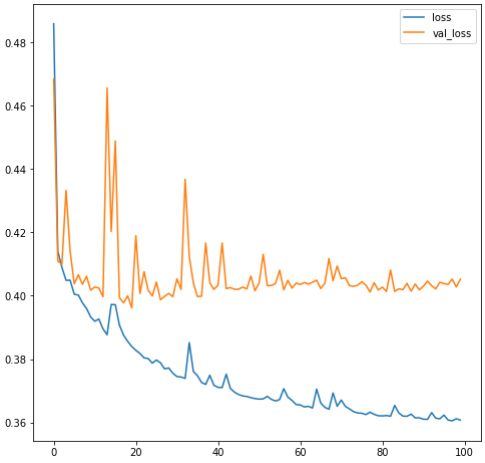

Figure (5): Validation loss vs training loss

As you can see, from the 20th epoch, model starts to overfit.

Here are some sample predictions of the model on the test set.

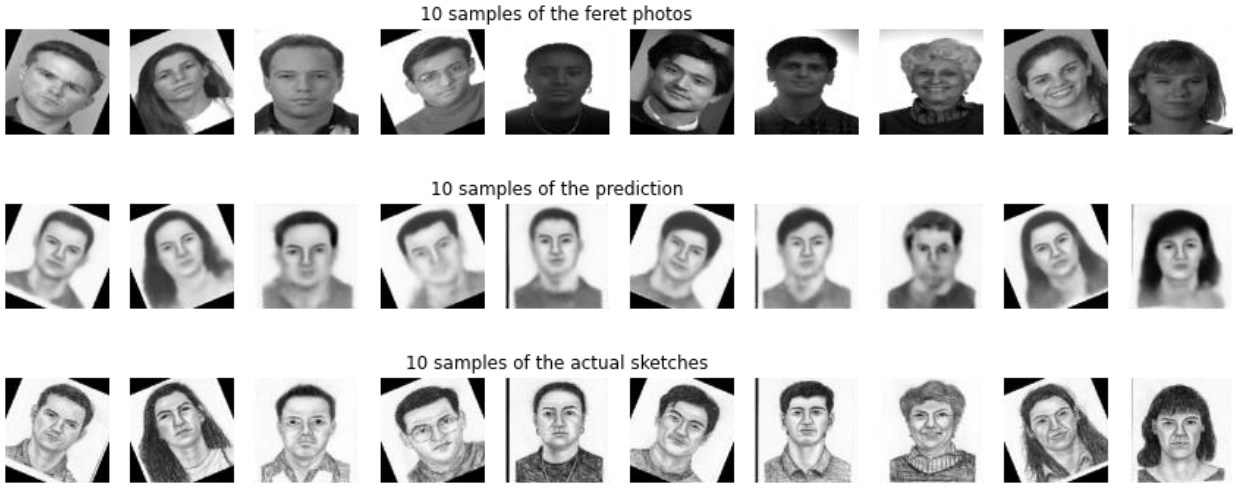

Figure (6): Predictions made by the model alongside the real images and sketches

Let's proceed to the second model.

The second model had the below architecture:

Encoder:		
Conv2D	Kernel size: 4*4	32 filters
Conv2D	Kernel size: 4*4	32 filters
Conv2D	Kernel size: 4*4	64 filters
Conv2D	Kernel size: 4*4	128 filters
Conv2D	Kernel size: 4*4	256 filters
Conv2D	Kernel size: 2*2	512 filters
Decoder:		
Conv2DTranspose	Kernel size: 2*2	512 filters
Conv2DTranspose	Kernel size: 4*4	256 filters
Conv2DTranspose	Kernel size: 4*4	128 filters
Conv2DTranspose	Kernel size: 4*4	64 filters
Conv2DTranspose	Kernel size: 4*4	32 filters
Conv2DTranspose	Kernel size: 4*4	16 filters
Conv2DTranspose	Kernel size: 2*2	8 filters
Conv2DTranspose	Kernel size: 2*2	1 filter

Table (5): Model architecture

From the input image with shape 128*128*1, the information converted into a latent vector with shape 1*1*512. From there, we start upsampling to reconstruct an output sketch with shape 128*128*1. For the training we used Adam algorithm as the optimizer with learning rate 0.001. Then we compiled the model with mean_absolute_error loss. After 100 epochs, here is the result of the last epoch: loss: 0.0622 - val_loss: 0.0925

In the following figure, you can see the performance of the model.

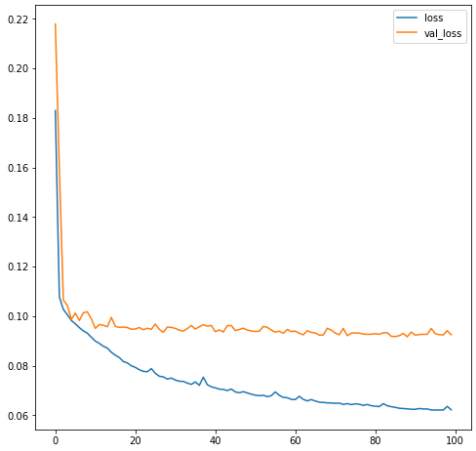

Figure (7): Validation loss vs training loss

As you can see, the overfitting of this model is less than be previous model.

Here are some sample predictions of the model on the test set.

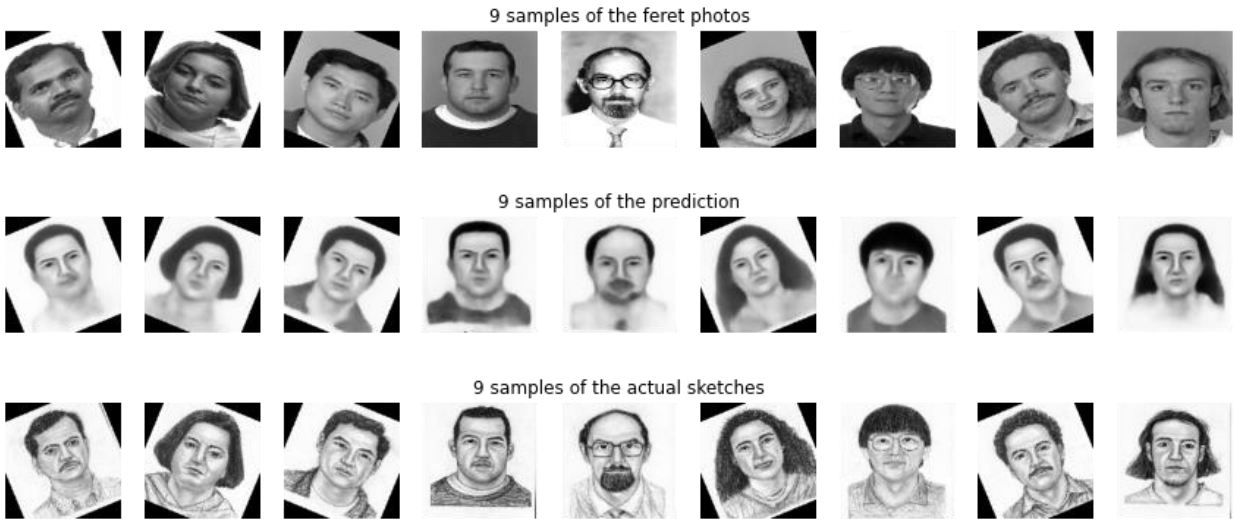

Figure (8): Predictions made by the model alongside the real images and sketches

As you can see, the outcome of this model is better than the previous model.

Let's proceed to the third model.

We used the same architecture and configuration, but increased the learning rate to 0.005. We also added a dropout layer after the latent vector and began the training with 100 epochs. The last epoch had the following results: loss: 0.0716 - val_loss: 0.0956

In the following figure, you can see the performance of the model.

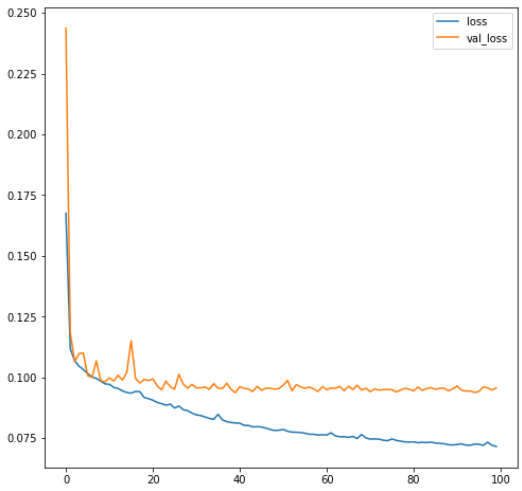

Figure (9): Validation loss vs training loss

Here are some sample predictions of the model on the test set.

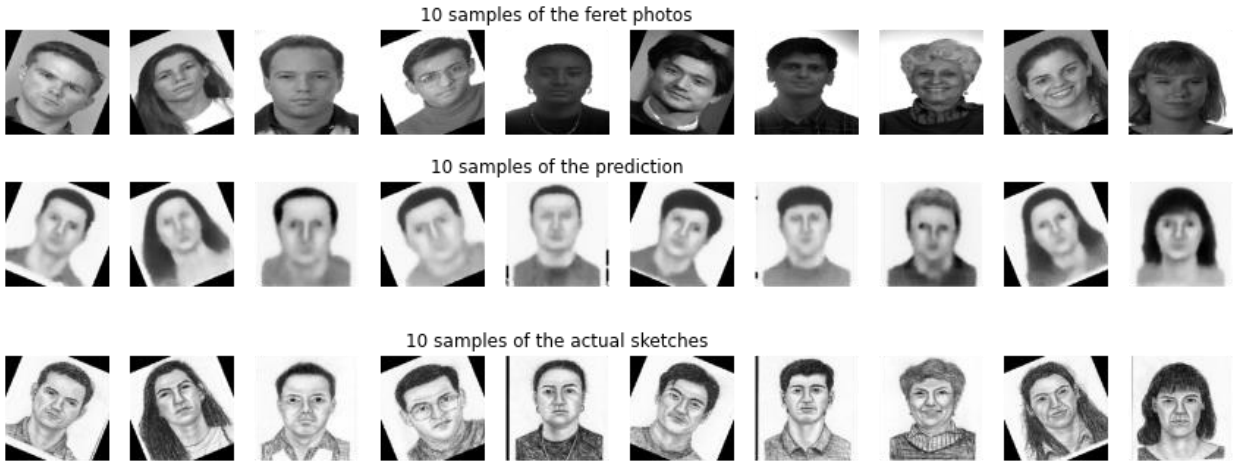

Figure (10): Predictions made by the model alongside the real images and sketches

Comparing to previous models, this model is performing poorly.

Let's proceed to the fourth model.

The second model had the below architecture:

Encoder:		
Conv2D	Kernel size: 4*4	64 filters
Conv2D	Kernel size: 4*4	64 filters
Conv2D	Kernel size: 4*4	128 filters
Conv2D	Kernel size: 4*4	128 filters
Conv2D	Kernel size: 4*4	256 filters
Conv2D	Kernel size: 2*2	512 filters
Decoder:		
Conv2DTranspose	Kernel size: 2*2	512 filters
Conv2DTranspose	Kernel size: 4*4	256 filters
Conv2DTranspose	Kernel size: 4*4	128 filters
Conv2DTranspose	Kernel size: 4*4	64 filters
Conv2DTranspose	Kernel size: 4*4	32 filters
Conv2DTranspose	Kernel size: 4*4	16 filters
Conv2DTranspose	Kernel size: 2*2	8 filters
Conv2DTranspose	Kernel size: 2*2	1 filter

Table (6): Model architecture

From the input image with shape 128*128*1, the information converted into a latent vector with shape 1*1*512. From there, we start upsampling to reconstruct an output sketch with shape

128*128*1. For the training we used Adam algorithm as the optimizer with learning rate 0.001. Then we compiled the model with mean_absolute_error loss. After 100 epochs, here is the result of the last epoch: loss: 0.0627 - val_loss: 0.0914

In the following figure, you can see the performance of the model.

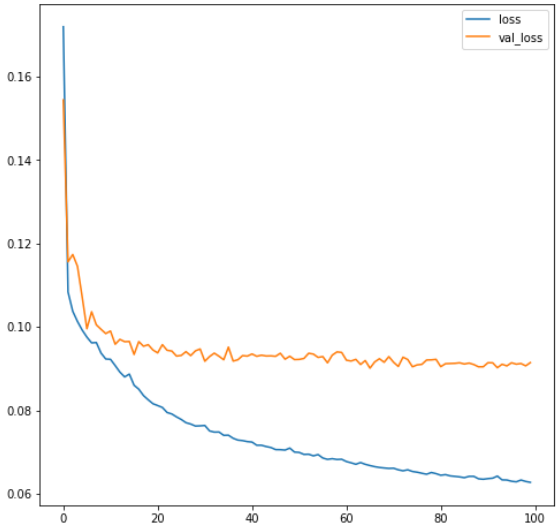

Figure (11): Validation loss vs training loss

Here are some sample predictions of the model on the test set.

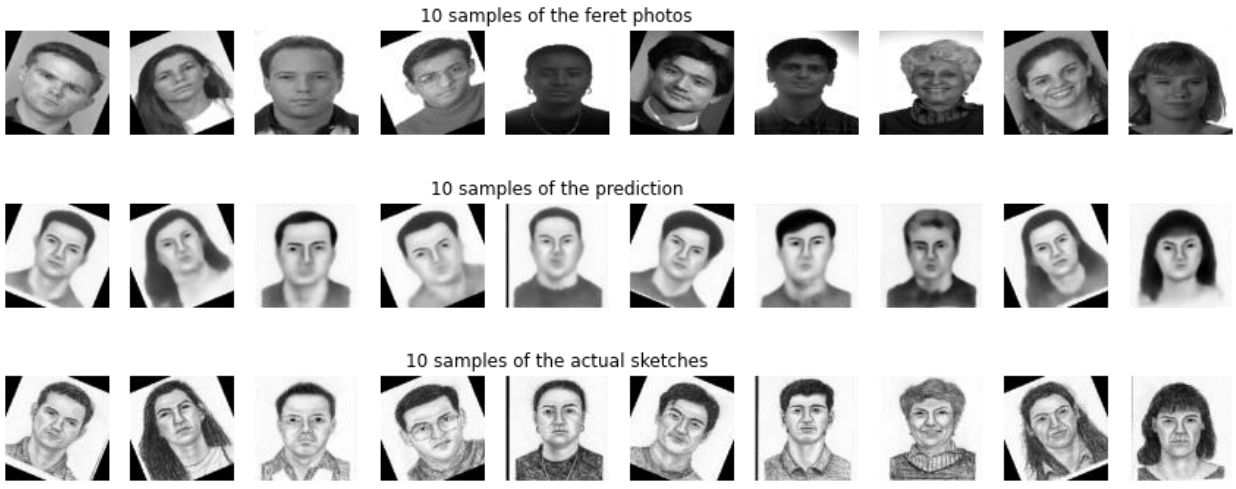

Figure (12): Predictions made by the model alongside the real images and sketches

Now we want to test the models 2 and 4 on a new image.

Let's see what will happen.

Original Image:

Output of autoencoder 2:

Output of autoencoder 4:

Figure (12): Predictions made by models 2 and 4 on a new image

As you can see, these two models have learned the crucial details of a face, but they failed to draw a smile.