Applied Data Mining Coursework 1

Dimitrios Priovolos, 36086817
School of Computing and Communications
MSc Data Science

***Abstract*—In this report, three different deep autoencoder models are presented. Multiple pix2pix models with different types of layers are created and tested to evaluate which one works best. The data set that is utilized for the training and testing of the generated models is part of the CUHK Sketch Database (CUFS), namely the photos and sketches folders each containing 188 images. The model with the highest performance is the Convolutional Layers AE, with the Self-Implemented AE that was designed a close second. Performance was judged, using different measures, like Mean Absolute Error (MAE) for face images and Structural Similarity score (SSIM) for sketches.**

I. Introduction

Face image processing has been one of the most important problems in computer vision since it is closely related to our daily lives. It has a wide range of applications from simple entertainment purposes to face recognition and law enforcement. In circumstances when only sketches of criminal suspects are available, face to sketch synthesis can considerably improve the accuracy and efficiency of identity verification as you can see in Figure 1. Some already existing approaches produce impressive results, but they usually contain hazy effects and significant deformation of various facial components.[2]

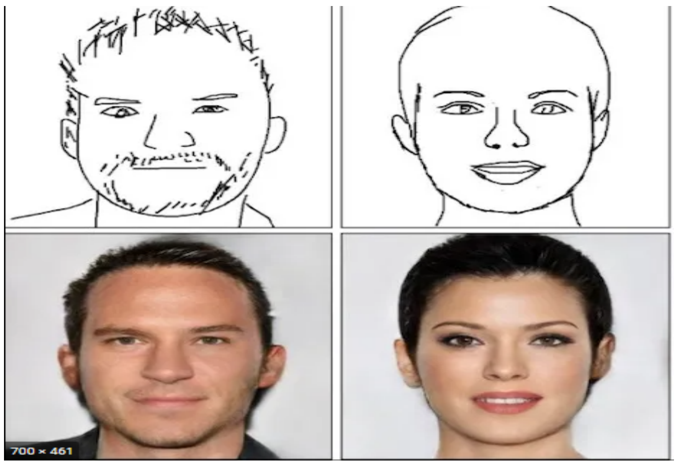


Fig. 1. Example of how sketch to face prediction can be used

In this report, three different deep learning approaches on two face image processing tasks will be explored, namely face to sketch synthesis and sketch to face synthesis. Using deep autoencoders, an attempt to design a model that can accurately predict new images using face or sketch images as input will be made.[1] Depending on which task needs to be achieved, a face image or a sketch from the CUHK Sketch Database [3] will be used as input, to produce the opposite one as an output.

The first autoencoder model that was implemented, is a Deep Autoencoder (DAE) model of dense architecture, meaning that it consists of multiple dense layers of different sizes. A dense layer is a layer that is fully connected with its preceding layer, meaning the neurons of the layer are connected to every neuron of its preceding layer. So for the first approach, a deep fully-connected Autoencoder was selected. [4] This model performed the worst out of the three because dense layers fail to recognise different patterns in the images.

The second autoencoder model is a DAE that is composed of many Convolutional 2D (two dimensional) layers. Similarly to the previous model, the model consists of only Conv2D layers. For our image data, the designed Convolutional Neural Network (CNN) with multiple layers was found to perform the best out of the three models. After being trained for each specific task, this model showed a good level of pattern and features recognition, as shown in the results section. [5]

The last autoencoder model that was implemented, is an attempt to combine dense and convolutional layers, in order to create a model with higher performance than them being used separately. Although the final model that was created did perform much better than the only dense deep autoencoder, its performance falls a little short of the CNN model that was introduced. This is mainly because of the GPU and RAM restrictions that applied in the creation of the models. Since the mixed model consumes much more GPU RAM, the limit in its complexity was easily met. Had stronger hardware been available so the model could be more complex, there is a good chance it would outperform the CNN model.

In this report, some of the recent works and methods of image to image generation will be reviewed before explaining and comparing the three separate methodologies that have been used. Finally, the training process and the results of each method will be displayed before discussing the research progress achieved.

II. Literature Review

The CUHK Sketch Database has already been used by other researchers, in their study of face or sketch generation. Pallavi, Sannidhan, Sudeepa and Abhir Bhandary [6], introduce an innovative approach for generating face sketches from Mugshot Photographs. They demonstrate some very realistic results, by utilizing the idea of using Gaussian blur to first blur the given image and after converting the input to greyscale image, then they use the Unsharp masking technique to enhance the sketch.

For face and sketch synthesis using GANs, during the Twenty-Seventh International Joint Conference on Artificial Intelligence, Shengchuan Zhang, Rongrong Ji, Jie Hu, Yue Gao and Chia-Wen Lin [7] have introduced a GAN model that can deal with uncontrolled conditions (variations of illumination, pose, background and ethnic origin). They too used the CUHK Sketch Database. More specifically, they design a stacked structure that connects a Resnet based generator [8], with an illumination layer and an Unet generator [9] together. Finally, they used a fully convolutional based discriminator [10], to achieve generative adversarial training. Their experiments show that the pGAN they created, can generate satisfactory results at a low computational cost.

For text to image task, Reed [11] based on deep convolutional generative adversarial networks (DCGAN) [12] proved that GAN can generate images effectively conditioned on text descriptions, by developing a simple and effective GAN architecture and training strategy that enables compelling text to image synthesis of bird and flower images from human-written descriptions. A more recent approach to that task has been introduced by Stanislav Frolov [13], who mainly focuses on the development and evaluation of T2I methods. In his research, he goes beyond the only other existing T2I survey by incorporating more approaches, thoroughly discussing the current state of evaluation techniques in the T2I field and systematically examining open challenges.

More recently, Jian Zhao [14] introduced a new approach in generating a semantic and photographic face. Inspired by Create Anime girl [15], which is a kids game in which you get to design your girl character, he understood that if the detailed attributes of the face can be known from the text description, then the sketch can provide the corresponding positional composition of the text. So he proposed a method that can use GAN to generate a photographic face from the sketch guided by attribute.

During the IEEE conference of 2019, Aron Yu and Kristen Grauman [16], challenge the approach that more labelled image data is always better, and obtaining labels is the bottleneck. They base their mindset on the fact that even though there are virtually unlimited unlabeled photos on the Web, the visual variety and information content of the images amassed for labelling eventually reach a limit. The current active selection methods scan the pool of manually curated unlabeled images when choosing which ones to label [17][18]. They propose to address the curation challenge with active image generation. The main idea of their project is to jointly learn the target visual task while also learning to generate novel realistic image samples that, once manually labelled, will benefit that task.

Finally, Han Zhang, Ian Goodfellow, Dimitris Metaxas and Augustus Odena [19], proposed the Self-Attention Generative Adversarial Network (SAGAN), which introduced a self-attention mechanism into convolutional GANs. The idea came upon the realisation that simple convolutional Gans have more difficulty in modelling some image classes than others when trained on multi-class data sets. One example of this case is that while Takeru Miyato and Masanori Koyamait GAN [20] has an amazing performance at synthesizing image classes with few structural constraints, it fails to capture geometric or structural patterns that occur consistently in some classes.

Looking at the related work done in face and sketch synthesis, it is clear that most of the already implemented approaches use GANs for image generation. Fewer researchers have attempted to face the image generation task using autoencoders. Some of them are Koumudi Panguluri and Kishore Kumar [21], who attempted image generation using a variational autoencoder, and Wenju Xu, Shawn Keshmiri and Guanghui Wang, who have presented an adversarially approximated autoencoder, which learns the latent code space of a manifold structure and generates high-quality samples given latent codes. The lack of approaches using autoencoders to the face and sketch generation is what lead to this report's experiments.

III. PRELIMINARY ON METHODOLOGIES

Autoencoders are feedforward neural networks that try to get an output similar to their input. They compress their input into a lower-dimensional form and then use that to reconstruct their input as similar as possible to the original one. They are comprised of three parts, the encoder, the code and the decoder. The basic purpose of an encoder is to compress the input data while preserving its key features. The code is the output of the encoder and the input of the decoder. The decoder is tasked with reconstructing the original input from the data it takes from the code. In order for the error between the reconstructed data and the original input to be minimized, backpropagation, which is an algorithm that used gradient descent is utilized. Deep autoencoders differ from simple ones because they have more than one hidden layer. The structure of a deep autoencoder with 2 hidden layers for both the encoder and decoder is displayed below in Fig 2.

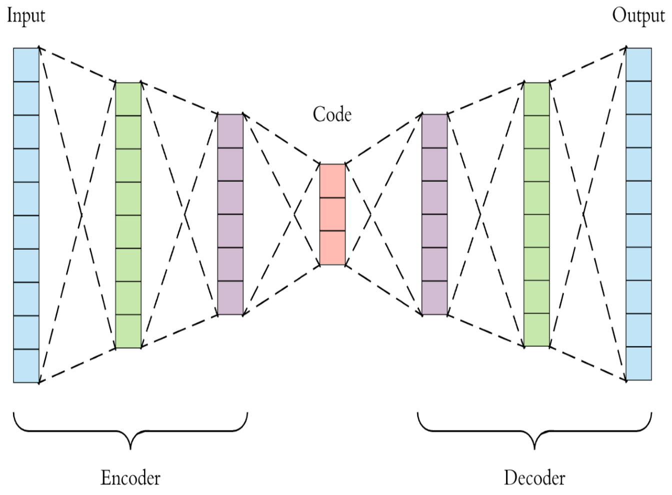


Fig. 2. Deep autoencoder with 4 hidden layers

In this report, the deep autoencoders introduced are distinguished from the kind of hidden layers they have. One consists of only dense layers, the second one consists of only convolutional 2D layers (and convolutional 2D transposed ones for the up-scaling) and the last one which is a self-implemented one is consisted of both. The fundamental distinction between the Convolutional and Dense layers is that the Convolutional layer employs fewer parameters because input values are forced to share its parameters.

The Dense Layer employs a linear operation, which means that the function generates each output based on each input. To put it another way, it ”forces” every input into the function and then let the Neural Network figure out how it relates to the output. As a result, n*m connections (or weights) arise, with n indicating the number of inputs and m indicating the number of outputs. The way dense layers are connected in a model is shown in Fig 3, wherein the right side of each layer is the input and the left side is the output.

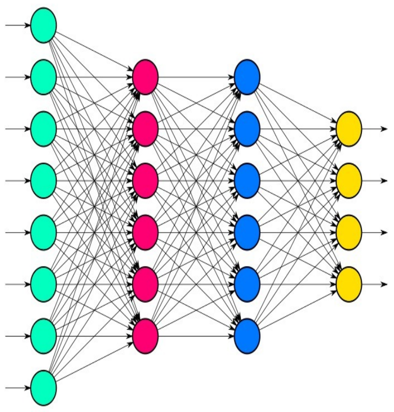


Fig. 3. Dense layers in NN

The Convolutional layer, on the other hand, employs a small size filter to perform the convolution operation. Its output is made up of a small number of inputs that are proportional to the filter’s size, and the weights are shared across all pixels. The result is highly specific features that can be detected anywhere on input images. The process that each convolutional layer goes through is shown in Fig 4.

The first two models are tested to compare the effects that those two kinds of layers have on image data, and how well can they can reconstruct it individually.

In the self-implemented model, both dense layers and convolutional2D (simple and transposed) layers are used, combined with some Upsampling layers used for the reconstruction of the input in the decoder and some Batch Normalization layers used for stabilizing the learning process and reducing the number of training epochs required to train deep networks. There is a reason that makes the combination of dense layers with convolutional layers useful. The convolutional layers are only used when it is certain that all the neighbouring pixels have a strong relationship. If the relationship of adjacent pixels is uncertain, then the model must ”force” the pixels into a dense layer, so it can establish the pixel’s current relationships.

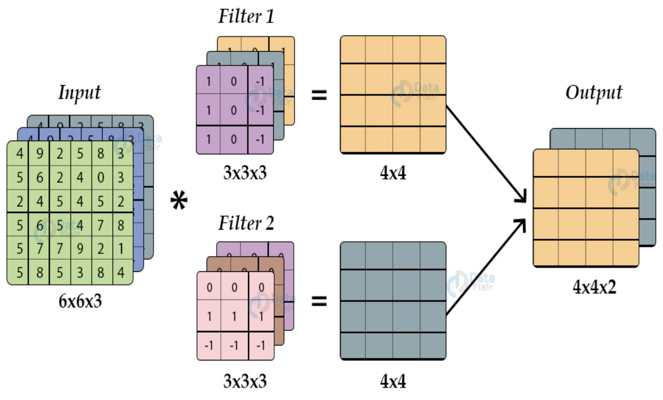


Fig. 4. Convolutional layer in NN

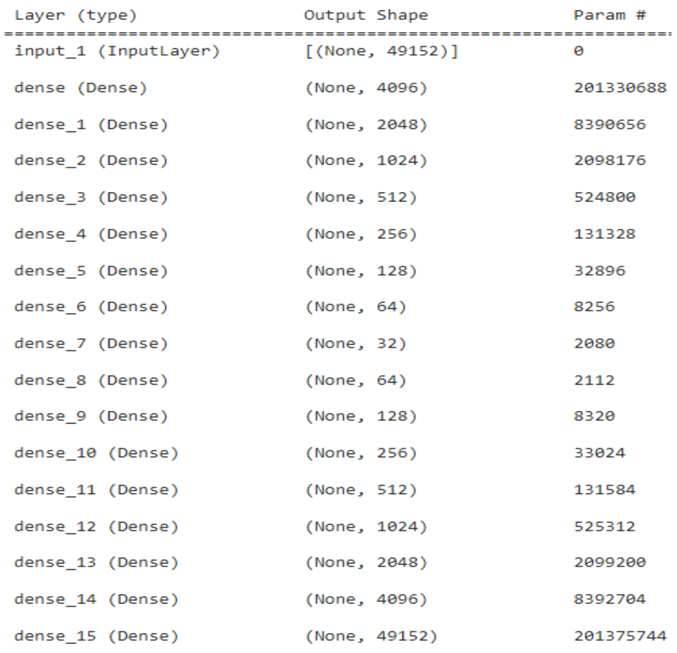
This usually happens in deeper layers, where prior information about data is not accessible.

A. Dense Layer Model

The Dense Layer Model consists of sixteen hidden layers, eight of them belonging to the encoder and the rest belonging to the decoder. The weights of each layer are initialized with the default statistical distribution that Keras [23] provides, which is glorot_uniform. This distribution is a random uniform distribution that’s bounded between

$$\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{1}$$

where n_i is the number of incoming network connections, or “fan-in,” to the layer, and $n\_i + 1$ is the number of outgoing network connections from that layer.



```
Layer (type)                 Output Shape              Param #
==================================================================
input_1 (InputLayer)         [(None, 49152)]           0
dense (Dense)                (None, 4096)              201330688
dense_1 (Dense)              (None, 2048)              8390656
dense_2 (Dense)              (None, 1024)              2098176
dense_3 (Dense)              (None, 512)               524800
dense_4 (Dense)              (None, 256)               131328
dense_5 (Dense)              (None, 128)               32896
dense_6 (Dense)              (None, 64)                8256
dense_7 (Dense)              (None, 32)                2080
dense_8 (Dense)              (None, 64)                2112
dense_9 (Dense)              (None, 128)               8320
dense_10 (Dense)             (None, 256)               33024
dense_11 (Dense)             (None, 512)               131584
dense_12 (Dense)             (None, 1024)              525312
dense_13 (Dense)             (None, 2048)              2099200
dense_14 (Dense)             (None, 4096)              8392704
dense_15 (Dense)             (None, 49152)             201375744
```

Fig. 5. Architecture of the Dense DAE Model

All the layers, except the last one which uses sigmoid, use the rectified linear activation function ”ReLU”, which is a linear function that will output the input directly if it is positive, otherwise, it will output zero. Using it helps the model overcome the vanishing gradient problem, allowing it to learn faster and perform better.

In Fig. 5, the Architecture of the Dense DAE Model is shown, where the output nodes of each layer are displayed. The output shape style in each layer is (None, Output Nodes), where None represents the batch_size of the training input. The ”Param #” column indicates the summed number of non-trainable and trainable parameters in each layer. The final layer, which is the layer that calculates the final output, is activated using the ”sigmoid” function, which is displayed below.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - S(-x) \tag{2}$$

B. Convolutional Layer Model (CNN)

The Convolutional Layer Model consists of 12 connected ”sub-models”, which are connected to two transposed convolutional layers at the output of the decoder. Each ”sub-model”’s structure varies, depending on which part of the autoencoder it is and the values of its parameters. The main structure of the encoder ”sub-models” is that they are composed of a convolutional layer, sometimes followed by a batch normalization layer and always ending with a LeakyReLU activation layer, which allows a small gradient when the unit is not active. For the decoder ”sub-models”, they consist of a transposed convolutional layer, sometimes followed by a dropout layer, which prevents overfitting by randomly setting input units to 0 with a frequency of a given rate at each step during training time, and always ending with a LeakyReLU activation layer. Each convolutional and transposed convolution layer’s weights are initialized with the ”he_normal” kernel initializer. This initializer draws samples from a truncated normal distribution centred on 0 with

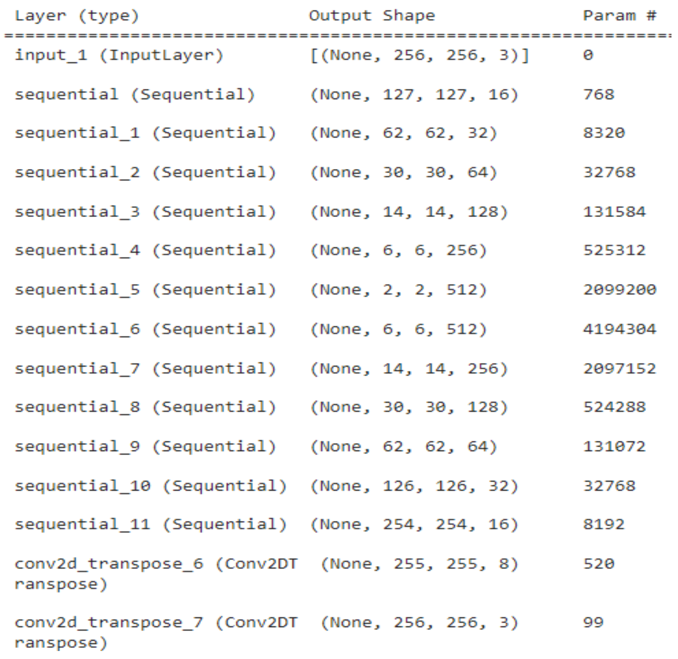
$$Standard\,Deviation = \sqrt{\frac{2}{fan_i}} \tag{3}$$

where fan_i is the number of input units in the weight tensor.

In Fig. 6, the Architecture of the Convolutional DAE Model is shown. In the layer column, there are only sequential type layers, except from the last two convolutional ones. Those sequential layers are the ”sub-models” referenced before. Here, the output shape is different from the previous model’s, because the input of a convolutional layer is not flattened before it passes through the layer, contrary to the dense layers that were used in the first model. So the input shape’s style for each convolutional layer is (Batch size, image rows (in pixels), image columns (in pixels), number of channels).

C. Self-Implemented Model

The self-implemented model, like all autoencoders, is composed of the encoder and the decoder. For its encoder, six 2D convolutional layers were deployed. Each convolutional layer had possibly different ”padding” parameters. Between each convolutional layer, sometimes a 2D max-pooling layer is introduced to create a downsampled (pooled) feature map, followed by a LeakyReLU activation layer and a batch normalisation layer. At the bottom of the encoder, the data is flattened and then passed through a dense layer with more outputs than the number of inputs.



```
Layer (type)                  Output Shape              Param #
==================================================================
input_1 (InputLayer)          [(None, 256, 256, 3)]     0

sequential (Sequential)       (None, 127, 127, 16)      768

sequential_1 (Sequential)     (None, 62, 62, 32)        8320

sequential_2 (Sequential)     (None, 30, 30, 64)        32768

sequential_3 (Sequential)     (None, 14, 14, 128)       131584

sequential_4 (Sequential)     (None, 6, 6, 256)         525312

sequential_5 (Sequential)     (None, 2, 2, 512)         2099200

sequential_6 (Sequential)     (None, 6, 6, 512)         4194304

sequential_7 (Sequential)     (None, 14, 14, 256)       2097152

sequential_8 (Sequential)     (None, 30, 30, 128)       524288

sequential_9 (Sequential)     (None, 62, 62, 64)        131072

sequential_10 (Sequential)    (None, 126, 126, 32)      32768

sequential_11 (Sequential)    (None, 254, 254, 16)      8192

conv2d_transpose_6 (Conv2DT   (None, 255, 255, 8)       520
ranspose)

conv2d_transpose_7 (Conv2DT   (None, 256, 256, 3)       99
ranspose)
```

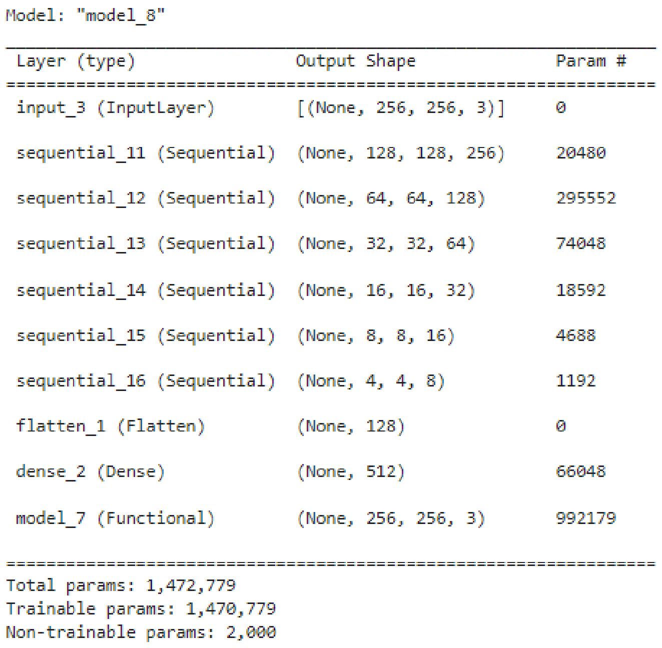
Fig. 6. Architecture of the Convolutional DAE Model

At the start of the decoder, there is a dense layer, that takes the output of the encoder’s dense layer as input, and has as many output nodes as the flattened shape of the encoder dense layer input. The dense layer output is then reshaped, to take the same shape as before it was flattened. The processed data then goes through 5 2D transposed convolutional layers. Between each of those layers, there is an Upsampling2D layer that is used to repeats the rows and columns of the data so it can double in size, followed by a ReLU activation layer and possibly a batch normalisation layer. After those, there is one last Upsampling layer, which makes the length and width of the data the same as the original, and a final 2D transposed convolutional layer, which changes the number of channels to the original. After that, the shape of the data is the same as the input. The architecture of the model is shown in Fig 7.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

Colaboratory, or “Colab” for short, is a free to use product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. Colab resources are not guaranteed and not unlimited, and the usage limits sometimes fluctuate.[24] All the models were trained and tested in the Google Colab platform,



```
Model: "model_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 256, 256, 3)]     0

 sequential_11 (Sequential)  (None, 128, 128, 256)     20480

 sequential_12 (Sequential)  (None, 64, 64, 128)       295552

 sequential_13 (Sequential)  (None, 32, 32, 64)        74048

 sequential_14 (Sequential)  (None, 16, 16, 32)        18592

 sequential_15 (Sequential)  (None, 8, 8, 16)          4688

 sequential_16 (Sequential)  (None, 4, 4, 8)           1192

 flatten_1 (Flatten)         (None, 128)               0

 dense_2 (Dense)             (None, 512)               66048

 model_7 (Functional)        (None, 256, 256, 3)       992179

=================================================================
Total params: 1,472,779
Trainable params: 1,470,779
Non-trainable params: 2,000
```

Fig. 7. Architecture of the Self-implemented Model

with GPU hardware accelerator and High Ram Runtime shape. The setup for the model implementation is described below.

1) Loading the data: The process of loading the data, pre-processing it and splitting it into train and tests sets were the same for all three models. The data was accessed from google drive (a link will be shared in the Conclusions section). Once the drive was mounted, the paths for the photos and sketches files were used to load the files into the program in two separate variables. Before they were loaded, the data from each file was sorted alphabetically. After that, the sorted data is loaded using cv2, which returns a NumPy array in BGR (Blue-Green-Red) format.

2) Pre-Processing the data: The format is then changed to RGB (Red-Green-Blue) because most image software and cameras use RGB nowadays. After that, the data is normalized in (0,1) range by dividing it by 255 (0-255 is the range of values for every pixel). The provided data only contain 188 photos and sketches for the training and testing of the models. That is very low, so in order to overcome this weakness, data augmentation is used to make different variations of the images by flipping and rotating them in different ways (7 times for every image). That makes the new total of available images 1504 photos and the same for sketches. The new data is then loaded in two NumPy arrays (one for each). Both face and sketch data is then split into 2 parts, the training data and the testing data. The testing data consists of 7% of the data sets and the rest 93% goes towards training the models. After splitting the data, they are resized in the desirable dimensions ((256,256) in this case).

3) Similarity Measures used: The performance of each model will be determined by using various similarity measures taken from the image-similarity-measures 0.3.5 library[25]. The ones that will be used in particular are Structural Similarity Index (SSIM), Root mean square error (RMSE), and Median Absolute Error (MAE). Root Mean Square Error (RMSE) measures the amount of change per pixel due to the processing. RMSE values are non-negative and a value of 0 means the image or videos being compared are identical.[26] The RMSE between an original image, image1-K and the enhanced or predicted image, image2-I(i, j) is given by:

$$RMSE = \sqrt{\frac{1}{M*N}\sum_{i=0,j=0}^{M-1,N-1}[I(i,j)-K(i,j)]^2} \quad (4)$$

where M and N are the number of rows and columns in the input images.

Structural Similar Index Measure (SSIM) quantifies image quality degradation caused by processing, such as data compression, or by losses in data transmission. SSIM is based on visible structures in the image. In order words, SSIM measures the perceptual difference between two similar images. The algorithm does not judge which of the two is better. However, that can be inferred from knowing which is the original image and which has been subjected to additional processing, such as data compression. The SSIM value is between -1 and 1 with 1 indicating perfect structural similarity.[26]

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (5)$$

In the above equation, μ_x is the average of x, μ_y is the average of y, σ_x is the variance of x, σ_y is the variance of y, σ_xy is the covariance of x and y c1 and c2 are two variables that stabilize the division with a weak denominator, L is the dynamic range of pixel values and k1 = 0.01 and kk2 = 0.03 by default.

MAE is the Euclidean pixel-based difference between images. it is calculated by comparing a pixel colour with another image's same pixel colour by comparing the distance between the different components in the pixels. It is calculated with the following equation.

$$\mathrm{MAE} = \frac{1}{\mathrm{n}}\sum_{\mathrm{j}=1}^{\mathrm{n}}|\mathrm{y_j} - \mathrm{x_j}| \quad (6)$$

where n is the total number of image pixels, j is the current pixel, y is the predicted image and x the original one.

B. Training of the Models

Once the loading and pre-processing of the data was achieved, the models had to be compiled and trained for each task. The optimizer used for all the training processes is the Adam optimizer from the Keras library[23]. Adam is a deep learning model training algorithm that replaces stochastic gradient descent. Adam combines the finest features of the AdaGrad and RMSProp methods to create an optimization technique for noisy issues with sparse gradients.[27] The learning rate chosen for all training processes is 0.001, which is a common lr, since it has been observed to perform very well in many cases.

1) Dense Layers AE: Both the face to sketch part and the sketch to face of the Dense Layered AE used the same training methods and process. The loss function that was used in the training, was the Mean Squared Error (MSE) loss function, which calculates pixel loss. Pixel loss is a measure that judges how far are the target image's pixels from the predicted image pixels. The model was trained for 100 epochs with the faces as inputs and the sketches as outputs and then 100 more epochs for the opposite. The loss functions for both those training processes over the epochs is shown in Fig 7.

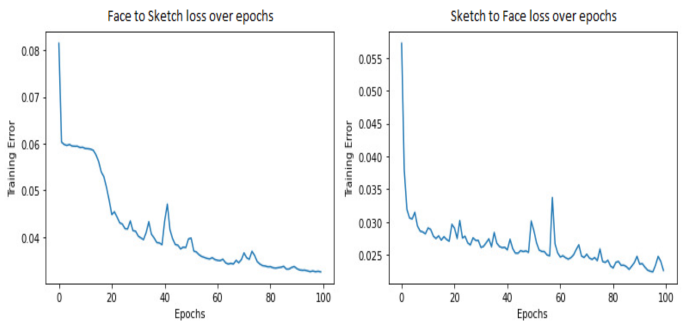


Fig. 8. Training Loss over epochs for Dense Layers AE

2) Convolutional Layers AE: Same as the Danse Layers AE, both the face to sketch part and the sketch to face of the Convolutional Layered AE used the same training methods and process. This time, the loss function that was used in the training, was the Mean Absolute Error (MAE), which is a loss function that is used when performing regression and don't want outliers to play a big role. Contrary to MSE, where the mean value is the optimal value, in MAE the median value is the optimal prediction. The model was trained for 100 epochs with the faces as inputs and the sketches as outputs and then 100 more epochs for the opposite. The loss functions for both those training processes over the epochs is shown in Fig 8.

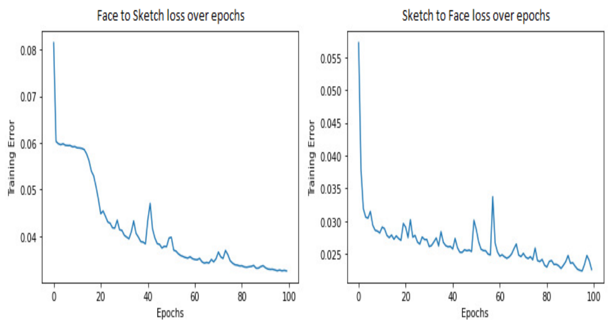


Fig. 9. Training Loss over epochs for Convolutional Layers AE

3) Self-Implemented AE: For the Self-Implemented AE, the training process was a little more complex than the other two, because it underwent multiple training processes. First, the model was trained using the MSE loss function for 100 epochs with face photos as inputs and 100 more for sketches as inputs. After that, the already trained model was once again trained for 50 epochs with face photos as inputs and 50 more for sketches as inputs, but this time the MAE loss function was used. Below, the loss over epochs is displayed only for the first part of the training process. If the other part's loss is needed, it is easily accessible by running the code.

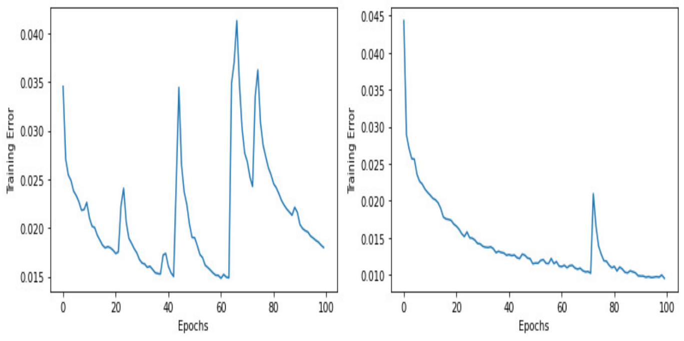


Fig. 10. First part of Training Loss over epochs for Self-Implemented AE

C. Test Results and Evaluation

After training all the models, it is time to test them and evaluate their performance. In Fig 11. the results of the face to sketch task are shown. The first image is the face input, the second one is the original sketch and then the generated sketch of each model follows. It is visibly clear, that the Dense AE performed the worst out of the three models. After that, the results of the other two models are similar, but the Convolutional AE seem to perform a little better. But to properly evaluate the performance of every model, the previously mentioned similarity measures will be calculated and compared. In Table 1, the three different similarity measures are shown for each model.

Similarity Measure	Dense AE	Convolutional AE	Self-Implemented AE
Average MAE	0.162	0.171	0.150
Average RMSE	0.266	0.274	0.269
Average SSIM	0.642	0.739	0.727

TABLE I
FACE TO SKETCH SIMILARITY MEASURES FOR EACH MODEL

There, it is clear that even if the Convolutional AE surpasses both other models in average structural similarity, the self-implemented model has a smaller value for both average MAE and average RMSE. Out of the three models, it can be concluded that for the face to sketch task, the Self-Implemented AE performs best, closely followed by the Convolutional AE, with the Dense AE having the worst performance. This is to be expected because Dense layers, despite being universal approximators, continue to underperform, as far as identifying and generalizing raw image pixels is concerned.

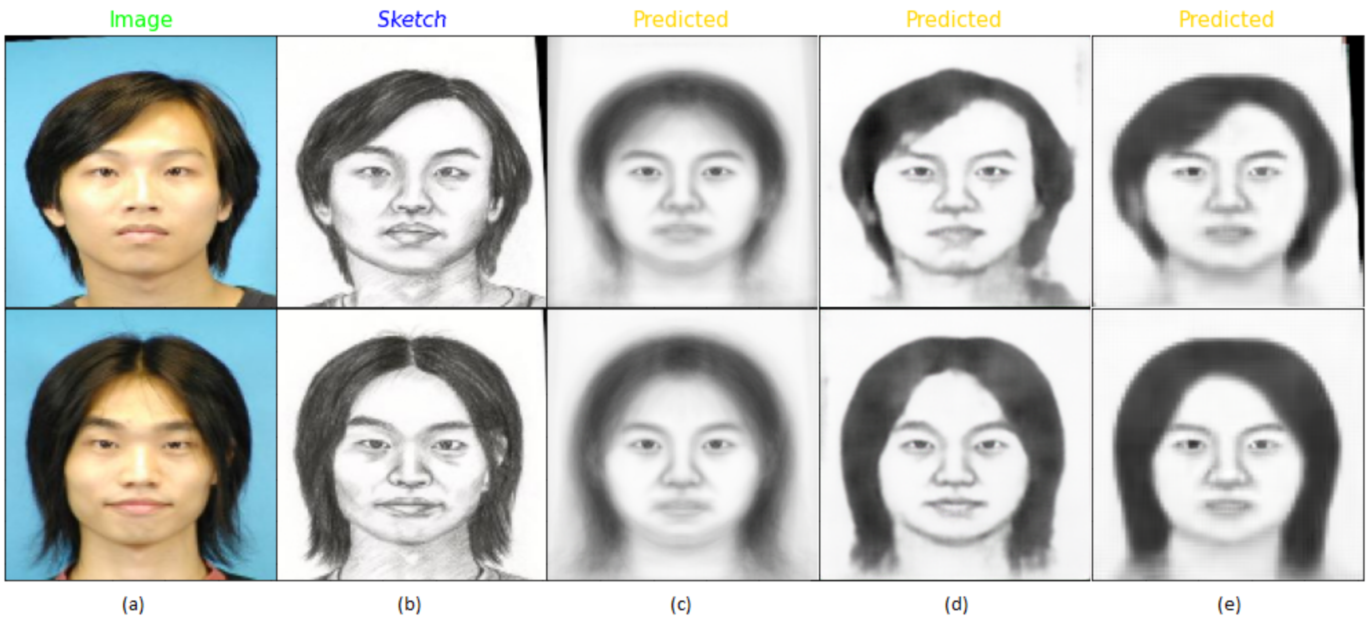


Fig. 11. Comparison of original image (a) and sketch (b) with the predicted sketches of the Dense AE (c), the Convolutional AE (d) and the Self-Implemented AE (e)

Now it's time to look at the sketch to face task results, which are displayed in Fig 12. The first image is the sketch input, the second one is the original face, and then the generated faces of each model follows. It is again clear, that the Dense AE performed the worst out of the three models. After that, the results of the other two models are similar, but the Convolutional AE seem to perform a little better. In Table 2, the three different similarity measures are shown for each model to determine which one performed the best.

Similarity Measure	Dense AE	Convolutional AE	Self-Implemented AE
Average MAE	0.064	0.051	0.055
Average RMSE	0.136	0.121	0.130
Average SSIM	0.735	0.824	0.801

TABLE II
SKETCH TO FACE SIMILARITY MEASURES FOR EACH MODEL

This time, for the sketch to face task the results are much clearer. The Convolutional AE has less average mean absolute error and root mean squared error than both other models and a better average structure similarity index. This compared with the visible results from Fig 12, lead to the conclusion that the Convolutional AE performs better than the other two in the sketch to face task. Although the Self Implemented AE has worse values for every similarity measure, the difference is small. For the MAE and RMSE errors, the difference is lower than 0.01, and from the SSIM, the difference is 0.023, which is a little bigger. This makes the self-implemented model a close second as far as performance is concerned. The Dense AE has the worst performance of all three for this task as well, with all its similarity measure values notably worse than the other two.

V. DISCUSSION

The comparison of the models produced some logical and positive results. The Convolutional AE and the Self-Implemented AE performed well in both tasks, with the Convolutional AE being slightly superior overall. In the face to sketch task, both of them identified most facial features and hair outlines. Both the models were confused in the second example of the long-haired man, probably because they have both associated long hair with woman facial features since long-haired men were an outlier of the given data. In the sketch to face task, both models did a good job in detecting and designing the facial features, but the Convolutional AE did a much better job it the details. It is also notable that the Convolutional AE and the Self-Implemented AE replicate the background of each photo almost perfectly. This seems positive at first, but it may introduce issues in the robustness of the models if different kinds of images need to be processed. The Dense AE managed to vaguely detect the faces shape and position of facial features, but overall the results were too blurry and with no details. One more factor that probably has hindered the models performance is that the sketches data that was provided is hand-drawn, and not digitally created to accurately represent the corresponding face data. There is also the big problem of the insufficient data samples to feed our models. Even though data augmentation was performed to make the training process more viable, the fact is that our models have not seen many different variations of the human face. If the above problems could be solved, it is most likely that the models would perform much better.

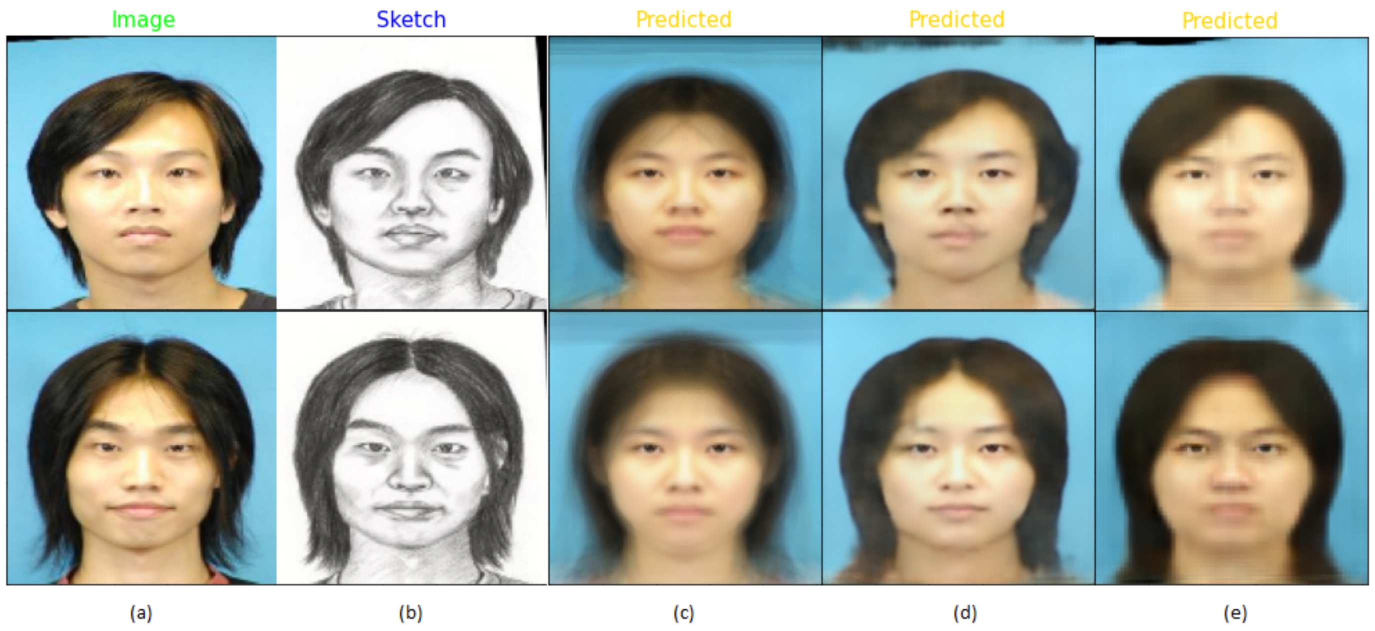


Fig. 12. Comparison of original sketch (a) and desired image (b) with the predicted sketches of the Dense AE (c), the Convolutional AE (d) and the self-made AE (e)

VI. Conclusion

In this report, three different models are presented, trained and evaluated. The designing of the Dense AE and Convolutional AE was remade by already existing models. For the Self-Implemented model, both the training process and the structure and hyper-parameters of every layer, although they have been influenced by many different pix2pix techniques, are mostly original. The model performing the best overall is the Convolutional AE, with the Self-Implemented model as a close second. The shortcomings of the Self-Implemented model can be attributed to the fact that it requires notably larger memory (both RAM and GPU) in order to operate and train. This made it difficult to build a more complex model on the same basis and mindset as this one. Had better equipment been provided, it is highly probable that further modifications would provide a better model, able to compose higher quality faces and sketches than the current one. It would also be interesting to apply various refinement algorithms on the better results that were generated, to observe how close to a realistic image is it possible to get. Finally, the bottom part of all images, which contains information about people's clothes, could be removed, so the models would not struggle to learn the correlations between a human face and their clothes.

The code for each model, the data needed to reproduce the results and the already trained models are all uploaded in GoogleDrive. You can gain access to them through the public share link here: Dimitrios Priovolos CW1 Files.

Acknowledgements

The idea of the Dense AE was taken from Lancaster University's SCC.413 Lab 2. The Convolutional AE, was taken from "Photo to Sketch Using Autoencoders".

References

[1] Chen Chaofeng. 2020. Face sketch synthesis andface super resolution in the wild with deep learning. https://hub.hku.hk/handle/10722/297490
[2] Jun Yu, XingxinXu, Fei Gao, Shengjie Shi, Meng Wang, Dacheng Tao, and QingmingHuang. 2020. Towards Realistic Face Photo-Sketch Synthesis viaComposition-Aided GANs. https://arxiv.org/abs/1712.00899
[3] iaogang Wang and Xiaoou Tang. 2008. Face Photo-Sketch Synthesis and Recognition. https://ieeexplore.ieee.org/document/4624272
[4] Alexandrine Ribeiro, Luis Miguel Matos, Pedro Jose Pereira, Eduardo C.Nunes, Andre L. Ferreira, Paulo Cortez, and Andre Pilastri. 2020. DeepDense and Convolutional Autoencoders for Unsupervised AnomalyDetection in Machine Condition Sounds.
[5] GabrielB.Cavallari, Leonardo Sampaio Ferraz Ribeiro, and Moacir Antonelli Ponti. Unsupervised representation learning using convolutional and stacked auto-encoders: a domain and cross-domain feature space analysis.https://arxiv.org/abs/1811.00473
[6] SPallavi, M.S. Sannidhan, Abhir Bhandary and K.B. Sudeepa. 2018. A Novel Approach for Generating Composite Sketches from Mugshot Photographs. https://ieeexplore.ieee.org/document/8554564
[7] Heng Chuan Zhang and Rongrong Ji. 2018. Robust face sketch synthesis via generative adversarial fusion of prior sand parametric sigmoid — Proceedings of the 27th International Joint Conference on Artificial Intelligence. https://dl.acm.org/doi/abs/10.5555/3304415.3304580
[8] Jun-Yan Zhu, Taesung Park, Phillip Isola and Alexei A. Efros. 2017. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In2017 IEEE International Conference on Computer Vision (ICCV). 2242–2251. https://doi.org/10.1109/ICCV.2017.244
[9] Phillip Isola, Jun-Yan Zhu,Tinghui Zhou and Alexei A.Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. https://arxiv.org/abs/1611.07004
[10] SergeyIoffeandChristianSzegedy. BatchNormalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

[11] Scott Reed and Zeynep Akata. 2016. Generative adversarial text to image synthesis — Proceedings of the33rd International Conference on International Conference on Machine Learning - Volume 48. https://dl.acm.org/doi/10.5555/3045390.3045503
[12] Alec Radford, Luke Metz and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. https://arxiv.org/abs/1511.06434
[13] Stanislav Frolov, Andreas Den-gel and Federico Raue. 2021. Adversarial Text-to-Image Synthesis: A Review. https://arxiv.org/abs/2101.09983
[14] Jian Zhao, Lin Wang, MengCao and Miao Zhang. 2019. Generating Photographic Faces From the Sketch Guided by Attribute Using GAN.https://ieeexplore.ieee.org/document/8642316
[15] https://www.girlgames.com/anime-girl.html
[16] Aron Yu and Kristen Grauman.2019. Thinking Outside the Pool: Active Training Image Creation for Relative Attributes. https://arxiv.org/abs/1901.02551
[17] Sudheendra Vijayanarasimhan and Kristen Grauman. 2011. Large-scale live active learning: Training object detectors with crawled data and crowds. InCVPR 2011.1449–1456. https://doi.org/10.1109/CVPR.2011.5995430
[18] Liyue Zhao, Gita Sukthankar and Rahul Sukthankar. 2011. Incremental Relabeling for Active Learning with Noisy Crowd sourced Annotations. In 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011IEEE Third International Conference on Social Computing. 728–733. https://doi.org/10.1109/PASSAT/SocialCom.2011.193
[19] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-Attention Generative Adversarial Networks. https://arxiv.org/abs/1805.08318
[20] Takeru Miyato and Masanori Koyama. 2018. cGANs with Projection Discriminator. https://arxiv.org/abs/1802.05637
[21] Koumudi Panguluri and Kishore Kumar Kamarajugadda. 2020. Image Generation using Variational Autoencoders. https://www.researchgate.net/publication/344224573Image Generation using Variational Autoencoders
[22] Wenju Xu, Shawn Keshmiri, and Guanghui Wang. Adversarially Approximated Autoencoder for Image Generation and Manipulation. https://arxiv.org/abs/1902.05581
[23] https://keras.io/
[24] https://research.google.com/colaboratory/faq.html
[25] https://pypi.org/project/image-similarity-measures/
[26] https://up42.com/blog/tech/image-similarity-measures
[27] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/