Politechnika Śląska

Biologically Inspired Artificial Intelligence
Wydział Automatyki Elektroniki i Informatyki

Gliwice, 20.06.2022 r.

Subject:

Generation of human faces basing on a set of photos

Authors:

Piotr Hasiec

Michał Rabsztyn

Supervisor:

Dr inż. Grzegorz Baron

1. Introduction

In the modern era of high computational power and advanced technological advances, we are able to perform many complex tasks which would once be considered impossible. Generative networks are one of the strongest talking points of the past decade because of the gravity of the fabulous results they have been able to successfully generate. The task, which was also once considered quite complex to achieve, is the generation of realistic faces with the help of generative adversarial networks. There have been multiple successful deployments of models and different neural architectural builds that have managed to accomplish this task effectively.

2. Analysis of the task

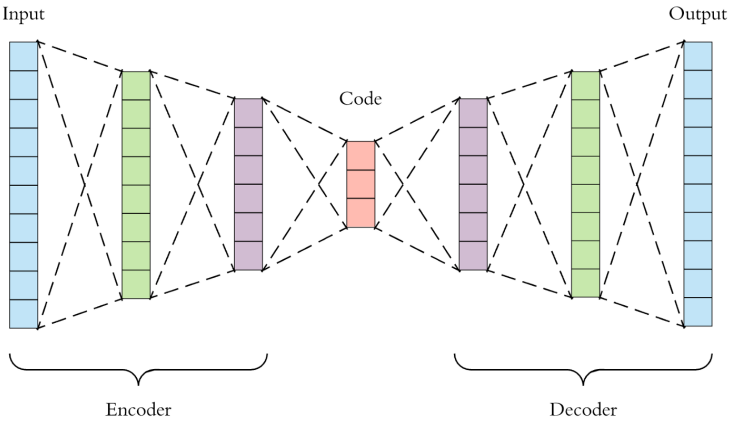
a. Approach

Among the most commonly used generative models there are:

- Variational AutoEncoders (VAE),
- Generative adversarial networks (GAN),
- AutoEncoders (AE).

	VAE	GAN	AE
Pros	- Clear and recognized way to evaluate the quality of the model	- Good modeling of data distribution - nicer images	- simple structure - clear and recognized way to evaluate the quality of the model
Cons	- Because of the injected noise and imperfect reconstruction, and with the standard decoder (with factorized output distribution), the generated samples are much more blurred than those coming from GANs. - complicated structure	- Hard to train, unstable - Mode Collapse issue. The learning process of GANs may have a missing pattern, the generator begins to degenerate, and the same sample points are always generated, and the learning cannot be continued - even more complicated structure	- similar to VAE but the quality of generated images is usually lower

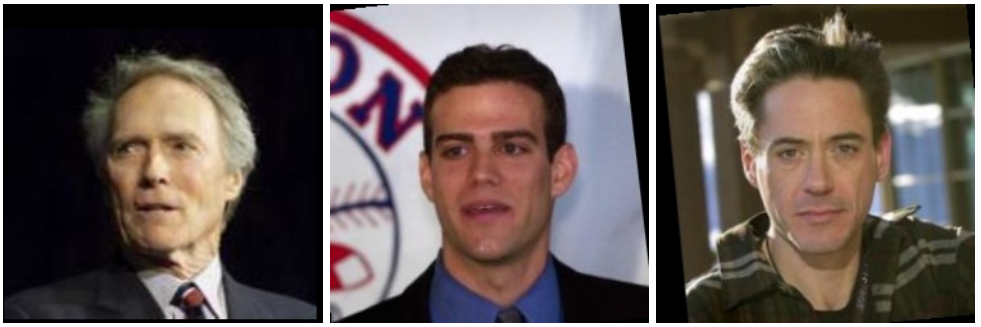
Considering all the approaches we decided for AutoEncoder as the least complicated generative model.



An AutoEncoder is a type of artificial neural network used for deep learning. It consists of an encoder that maps the input into the code, and a decoder that maps the code to reconstruct the input.
It's not simply duplicating the signal, but rather is forced to recreate the input data, keeping only the most relevant aspects of the data in the copy.
The encoder and decoder are constructed by stacking multiple RBM layers. The architectural parameters were originally trained layer-by-layer: after learning one layer of feature detectors, they have enough visible variables to teach the appropriate RBMs. Current approaches typically include end-to-end training with gradient descent methods. Training can be repeated until certain stop criteria are met.

b. Dataset

The dataset we chose contains 13 233 images presenting human faces of which 5 749 are of unique people. Most of the faces belong to famous people, but some of them are random people's faces. The size of an image is 256x256 px.



c. Tools

The tools we used were:

Python 3.10 - a high-level, interpreted, general-purpose programming language in version recommended as stable (as for June 2022),

TensorFlow - a free and open-source software library for machine learning and artificial intelligence,

Matplotlib - a plotting library for the Python programming language,

OpenCV - a library of programming functions mainly aimed at real-time computer vision.

Keras - an open-source software library that provides a Python interface for artificial neural networks.

3. Specification

a. Solution

The idea of using the autoencoder as a generative model is to separate the decoder fragment from the entire network. Due to the learning process, the decoder is able to convert the code into images representing faces. Due to the relatively small number of code variables, faces must be densely represented in the code space of the autoencoder. We can also assume that small changes in the code value cause small changes in the image, so the transition between the faces from the training set will be smooth. Knowing the expected values, the spread and possibly the correlation of individual features (code variables), we can create a random vector given the same distribution and it should represent a face that does not exist in the training set.

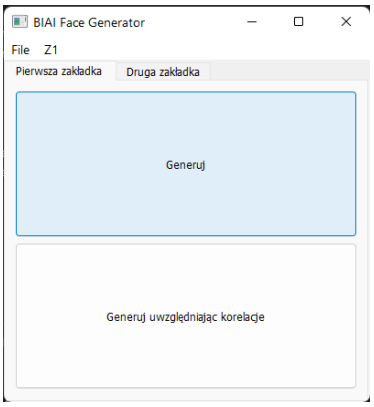
b. Data structures

The solution was divided into 3 main files:

- Learning.py - contains a script in python responsible for training the neural network,
- Keras_Model_functions.py - contains methods responsible for interacting with an already learned neural network,
- MainApp.py - main application file.

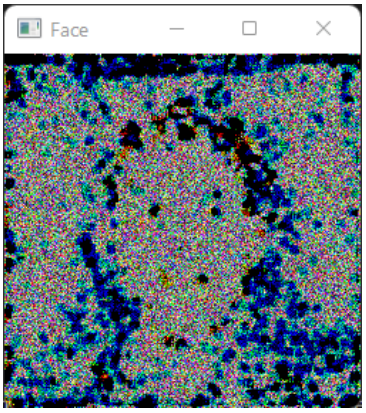
c. User interface

i. Main application's window



It allows you to generate a face using two methods:
with correlation between code variables (which should give better results) or without. In practice, the difference is almost imperceptible.

ii. Auxiliary window for displaying the face



4. Experiments

a. Background

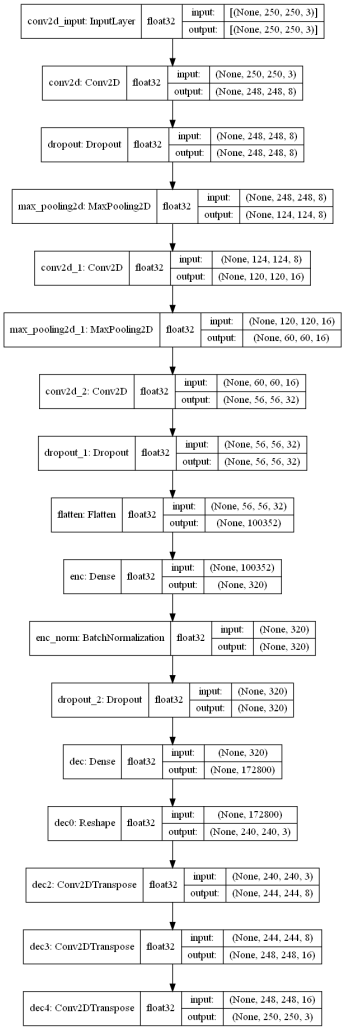
In our network model experiments, we used the following types of neural network layers:

- conv2D - its 2 dimensional convolution layer whose structure is inspired by the visual cortex of animals
- gaussianNoise - layer which adds random gaussian noise to the output of the previous layer. It is possible to define expected value of noise
- batchNormalisation - layer which normalizes output of the previous layer.

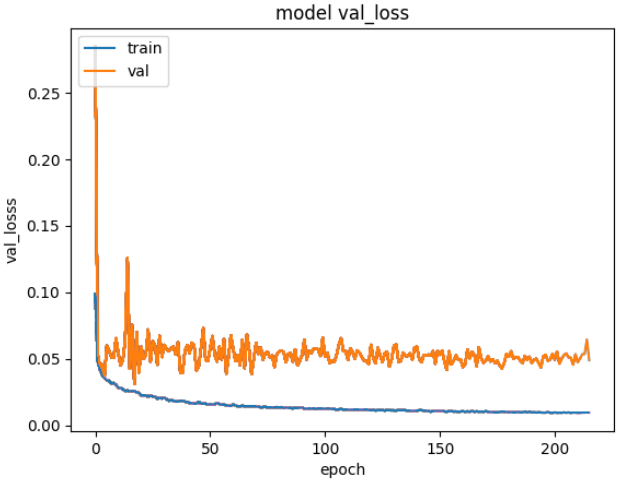
- maxPooling2D - layer used to reduce the vector of the outputs of the previous layer by selecting the maximum values
- dropout - randomly excludes certain neurons from learning to prevent overfitting of the neural network
- flatten - flattens multivariate data to univariate
- dense - standard dense layer, i.e. one in which each neuron is connected to all the neurons of the previous layer
- reshape - a layer that allows you to change the shape of, in our case, one-dimensional to multi-dimensional data
- conv2DTranspose - deconvolution layer

i. Model I

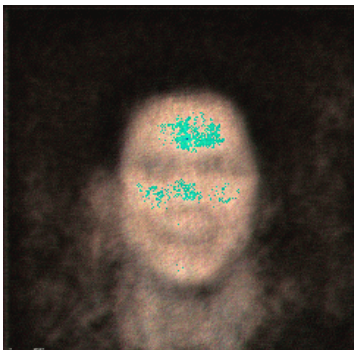
1. Model structure:
   For the sake of speed, we used the ReLu function as the activation function for neurons in all layers. The network learned quite quickly, but unfortunately it often gave artifacts on the output resulting from the non-normalized range of neuron output in the last layer - a given pixel may assume a value outside the <0.1> range. The code size was 320 variables.



2. Learning curve

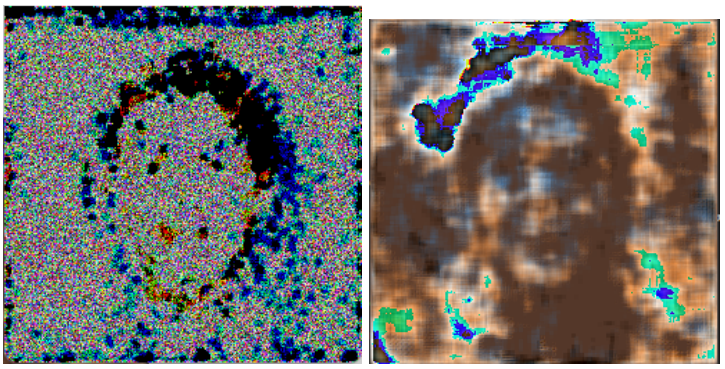


3. Results
   Face reconstruction from the training set in following epochs:



This picture is a gif and is available at GitHub

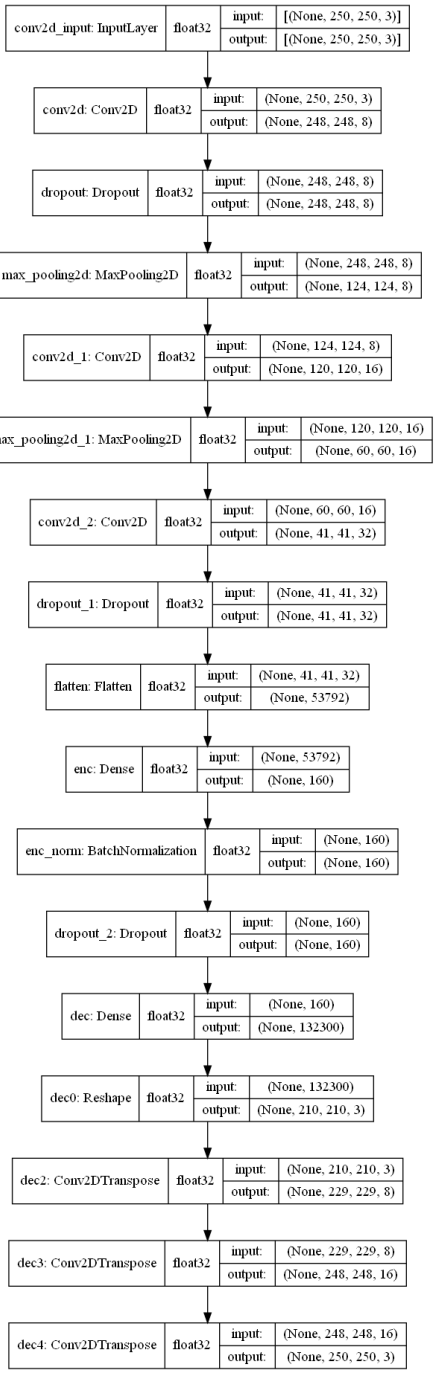
New face generation results:



As you can see, the results were not satisfactory, so the network structure was modified and the ReLu activation function was replaced with a sigmoidal function in the last layer.

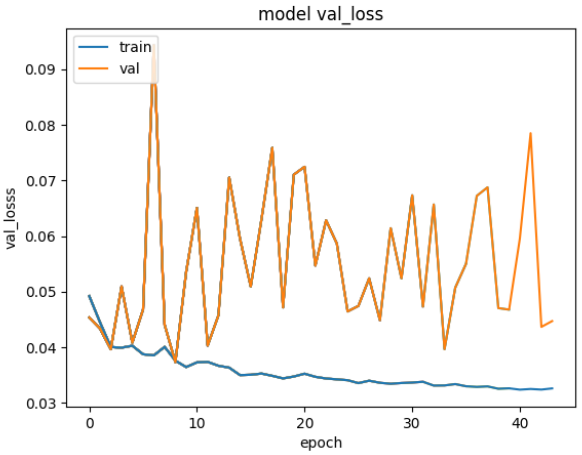
ii. Model II

1. Model structure:
   In this model, in all layers except the output layer, we used the ReLu function as a function of neuron activation due to the learning speed. Due to the output normalized from 0 to 1 for the output layer neurons, we used the sigmoid function.

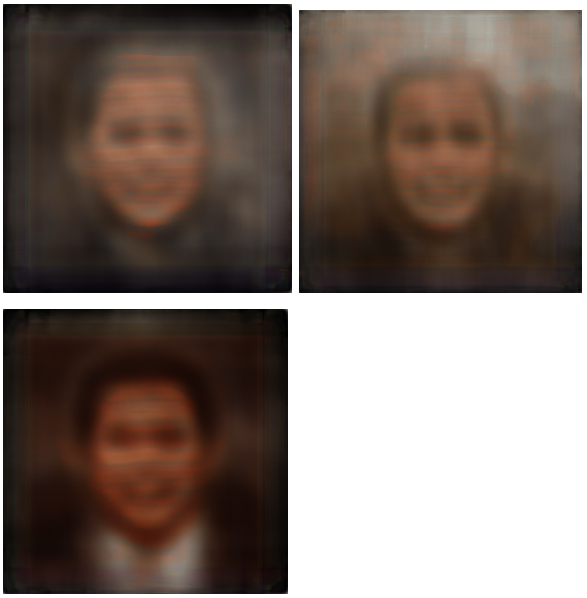


The code size was 160 variables.

2. Learning curve



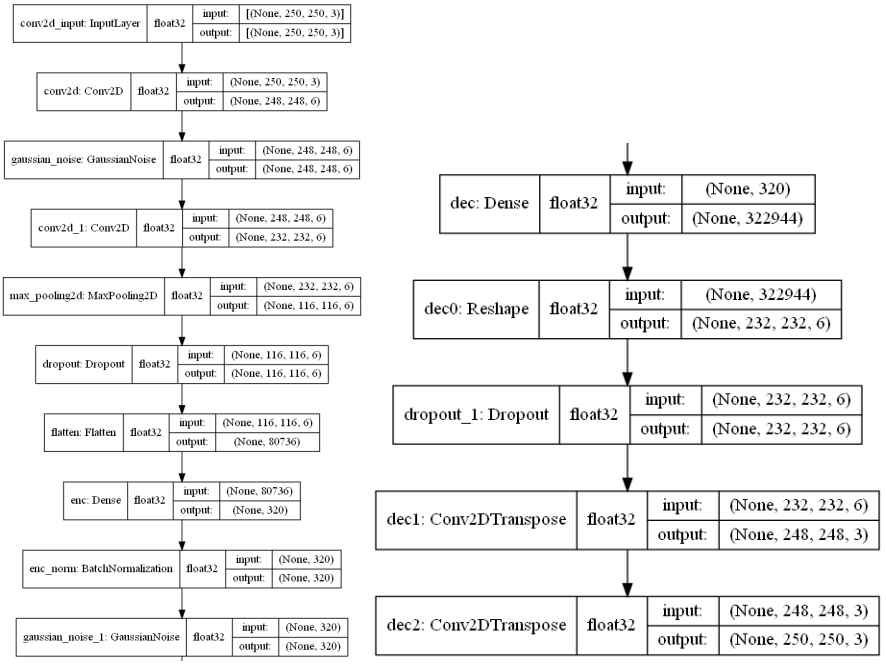
3. Results



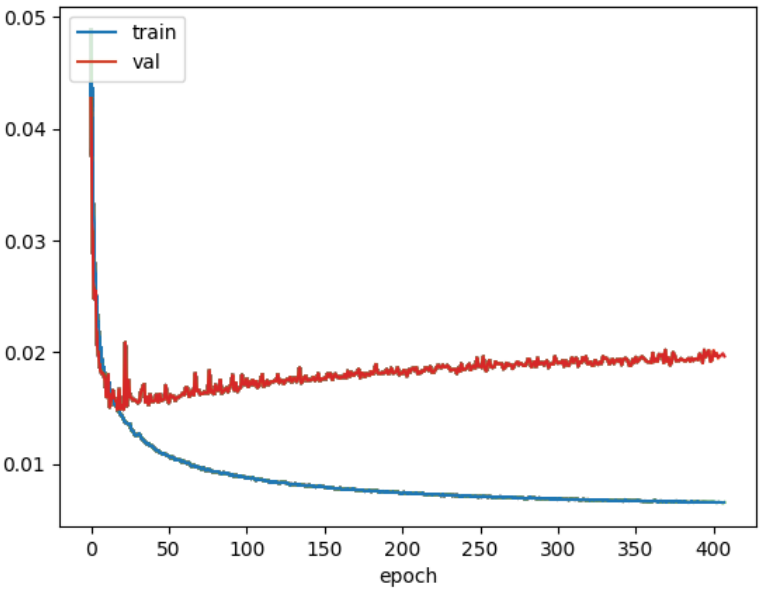
At some point the network stopped learning and we had to prepare another model.

iii. Model III

1. Model structure:
   In this model, in all layers except the output layer, we used the ReLu function as a function of neuron activation due to the learning speed. Due to the output normalized from 0 to 1 for the output layer neurons, we used the sigmoid function



2. Learning curve



3. Results

Face reconstruction from the set in following epochs:

This picture is a gif and is available at GitHub

5. Summary

Not all the results of the program are fully satisfactory, and those that meet the minimum standards often resemble oil paintings or the Turin Shroud. To improve the results, better hardware may be necessary despite the fact that we used a relatively powerful 2GB GTX 1050 Ti card.

We are partially satisfied with the effect, but we wanted to get more real faces, which unfortunately did not work.

The structure of the network has a very large impact on the entire learning process and outcomes, and unfortunately we do not know how to create a good one other than by trial and error.

6. References

https://en.wikipedia.org/wiki/Autoencoder
https://faroit.com/keras-docs/1.2.0/
https://keras.io/api/
https://www.tensorflow.org/api_docs

7. Implementation

https://github.com/PiotrHasiec/BIAI