Video processing using autoencoders

Svetozar Stojković
Neural networks
Faculty of Technical Sciences
Email: svetozar.cvele.stojkovic@gmail.com
GitHub: https://github.com/svetozarstojkovic/video-processing

Abstract: In this project couple of video processing related problems were attempted to be solved and hey are: denoising problem which removes noise from video and problem of converting grayscale to RGB video Each problem. Each problem is solved using two kinds of autoencoders and the results are compared.

I. Introduction

This project was created for the subject Neural networks on Faculty of Technical Sciences on Master studies.
Neural network was made in *Python* language using *Keras* library.

II. Artificial neural networks

An artificial neural network (ANN) is an information processing system that is based on a mathematical model inspired by the complex non-linear and parallel neural structures of information in the brain of intelligent beings that acquire knowledge through experience.

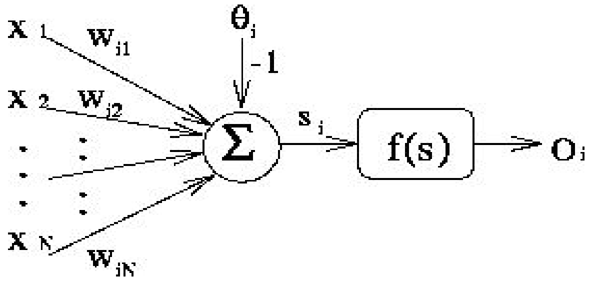

Image 1: Perceptron

A large ANN may contain hundreds and event the thousands of processing units, while the mammalian brain contains several billions of neurons. In the human brain neural networks are able to organize their neurons and find solutions for very complex problems, such as recognizing standards. With only a few stimuli the human brain processes neural structures responsible for very fast and high quality responses, which it may never be possible to replicate artificially. However, the progress achieved in studies involving ANNs in a very short time and in several fields is amazing. Using neurobiological analogy as inspiration and the wealth of accumulated theoretical and technological tools, it is likely that our understanding of ANNs will soon be much more sophisticated than today (Haykin, 2001). ANNs are similar to the brain in at least two respects: knowledge is acquired by the network from its environment through a learning process; and the connection strength among neurons, known as synaptic weights, is used to store acquired knowledge. Neurocomputers, connectionist, and parallel distributed processing are also called ANNs.

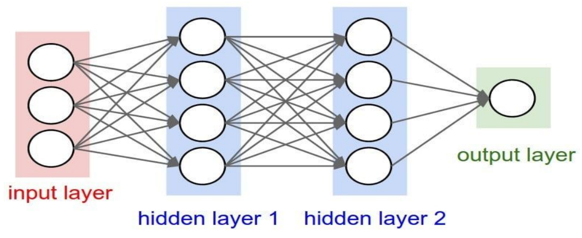

Image 2: Artificial neural network

III. Autoencoders

"Autoencoding" is a data compression algorithm where the compression and decompression functions are:

1. data-specific;
2. lossy;
3. learned automatically from examples;

rather than engineered by a human. Additionally, in almost all contexts where the term "autoencoder" is used, the compression and decompression functions are implemented with neural networks.

1. Autoencoders are data-specific, which means that they will only be able to compress data similar to what they have been trained on. This is different from, say, the MPEG-2 Audio Layer III (MP3) compression algorithm, which only holds assumptions about "sound" in general, but not about specific types of sounds. An autoencoder trained on pictures of faces would do a rather poor job of compressing pictures of trees, because the features it would learn would be face-specific.

2. Autoencoders are lossy, which means that the decompressed outputs will be degraded compared to the original inputs (similar to MP3 or JPEG compression). This differs from lossless arithmetic compression.

3. Autoencoders are learned automatically from data examples, which is a useful property: it means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input. It doesn't require any new engineering, just appropriate training data.

To build an autoencoder, you need three things:

1. Encoding function
2. Decoding function
3. Distance function between the amount of information loss between the compressed representation of your data and the decompressed representation (i.e. a "loss" function).

The encoder and decoder will be chosen to be parametric functions (typically neural networks), and to be differentiable with respect to the distance function, so the parameters of the encoding/decoding functions can be optimized to minimize the reconstruction loss, using *Stochastic Gradient Descent.* It's simple! And you don't even need to understand any of these words to start using autoencoders in practice.

Autoencoders are not really good at data compression They are used in practical applications. In 2012 they briefly found an application in greedy layer-wise pretraining for deep convolutional neural networks, but this quickly fell out fashion as we started realizing that better random weight initialization schemes were sufficient for training deep networks from scratch. In 2014, batch normalization started allowing for event deeper networks, and from late 2015 we could training arbitrarily deep networks from scratch using residual learning.

Today two interesting practical applications of autoencoders are **data denoising** and **dimensionality reduction for data visualization.** With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

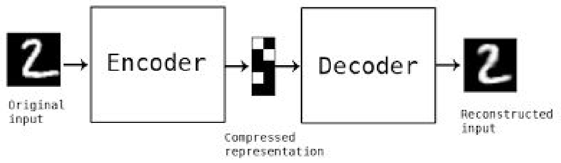

Image 3: Basic autoencoder

IV. Training data

The data this system uses are the frames of some video file. I used an episode of American TV show *New Girl S01E01* which has 720p quality.

Each frame is split into grid with each cell is (10x10x3) pixels which is flatten into one dimensional array with shape (300, 1) and also all values are scaled on 0 to 1.

V. Denoising

Input data for denoising is some video frame where specific pixels are changed and that acts like the noise in the video.

In each frame noise is put using this python code:

```
noise_factor = 0.2

temp = temp + noise_factor *
np.random.normal(loc=0.0,
scale=1.0,size=temp.shape)

temp = np.clip(temp, 0., 1.)
```

Output data is original video frame split into a grid, without any changes.

Training deep autoencoder is done using this python code:

```
# defining autoencoder
# encoder
input = Input((300, ))
encoder = Dense(100)(inpt)
encoder = Activation('relu')(encoder)
encoder = Dense(10)(encoder)
encoder = Activation('relu')(encoder)
```

```
# decoder
decoder = Dense(10)(encoder)
decoder = Activation('relu')(decoder)
decoder = Dense(100)(decoder)
decoder = Activation('relu')(decoder)
decoder = Dense(300)(decoder)
decoder = Activation('sigmoid')(decoder)

# encoder and decoder
model = Model(input=input,
output=decoder)

model.compile(loss='mse',
optimizer='adam', metrics=['accuracy'])
batch_size = 32
nb_epoch = 300

# training autoencoder
model.fit(input_array, output_array,
batch_size=batch_size,
nb_epoch=nb_epoch,
shuffle=True,
validation_data=(test_input_array,
test_output_array))

model.save('denoising.h5')
```

Training convolutional autoencoder starts by splitting input and output data by channels. Instead of having data in format (10, 10, 3) now we have three matrix with shape (10, 10). Training is done using this code:

```
input_img=Input(shape=(factor,factor,1))

x = Conv2D(16, (3, 3), activation='relu',
padding='same')(input_img)
x = MaxPooling2D((2, 2),
padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu',
padding='same')(x)
x = MaxPooling2D((2, 2),
padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu',
padding='same')(x)
encoded = MaxPooling2D((2, 2),
padding='same')(x)
```

```
# at this point the representation is (4,
4, 8) i.e. 128-dimensional

x = Conv2D(8, (3, 3), activation='relu',
padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu',
padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3),
activation='relu')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3),
activation='relu')(x)
decoded = Conv2D(1, (3, 3),
activation='relu', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam',
loss='mse', metrics=['accuracy'])

autoencoder.fit(input_array,
output_array,
epochs=50,
batch_size=32,
shuffle=True,
validation_data=(test_input_array,
test_output_array))

autoencoder.save('gray_to_color_convoluti
onal.h5')
```

Testing is done at the same time for deep autoencoder and convolutional autoencoder and that is saved into "output.avi". That file is just representative example how the system works.

- Top row shows how the deep autoencoder works
- Bottom row represents convolutional autoencoder.

- Left frame is frame with noise;
- Middle frame is output of neural network;
- Right frame is original frame.

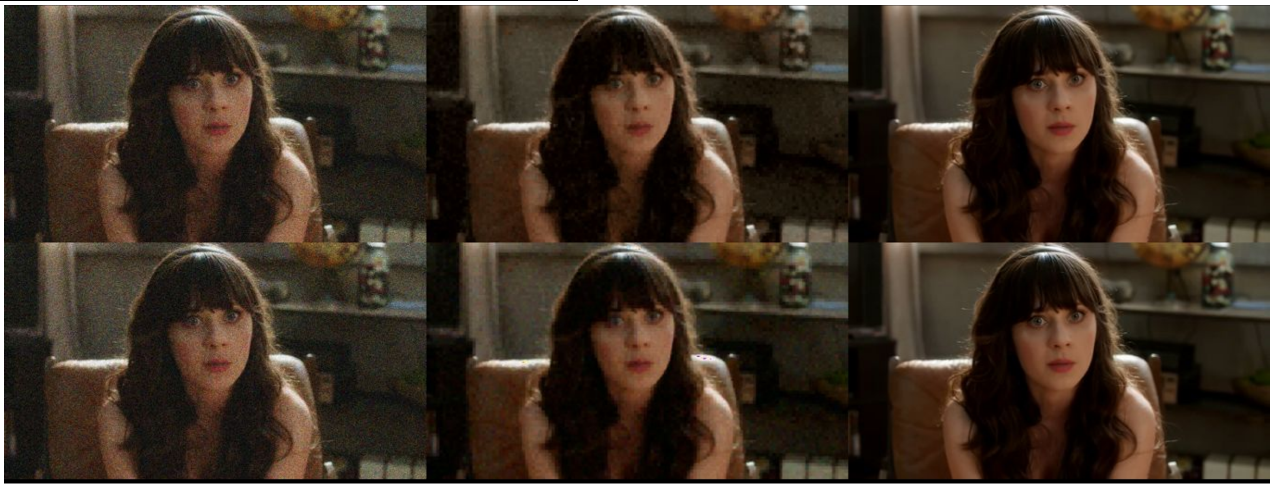

Image 4: Output of denoising algorithms

VI. Grayscale to color

Input data for denoising is some video frame on which OpenCV's method rgb to grayscale is applied.
Now the grid cells have shape (10, 10) pixels.

```
gray=cv2.cvtColor(rgb,cv2.COLOR_BGR2GRAY)
```

Output data is original video frame split into a grid, without any changes.

Training deep autoencoder is same as it was in denoising part except instead of encoder input shape being (300,) now is (100,).

Training convolutional autoencoder is the same as it was in denoising part except now the input is put in encoder in format (10, 10) and the output is put in format (10, 10, 3) since we have to generate three times bigger matrix on output comparing to input.

Testing is done at the same time for deep autoencoder and convolutional autoencoder and that is saved into "output.avi". That file is just representative example how the system works.

- Top row shows how the deep autoencoder works
- Bottom row represents convolutional autoencoder.

- Left frame is grayscale frame;
- Middle frame is output of neural network;
- Right frame is original frame.

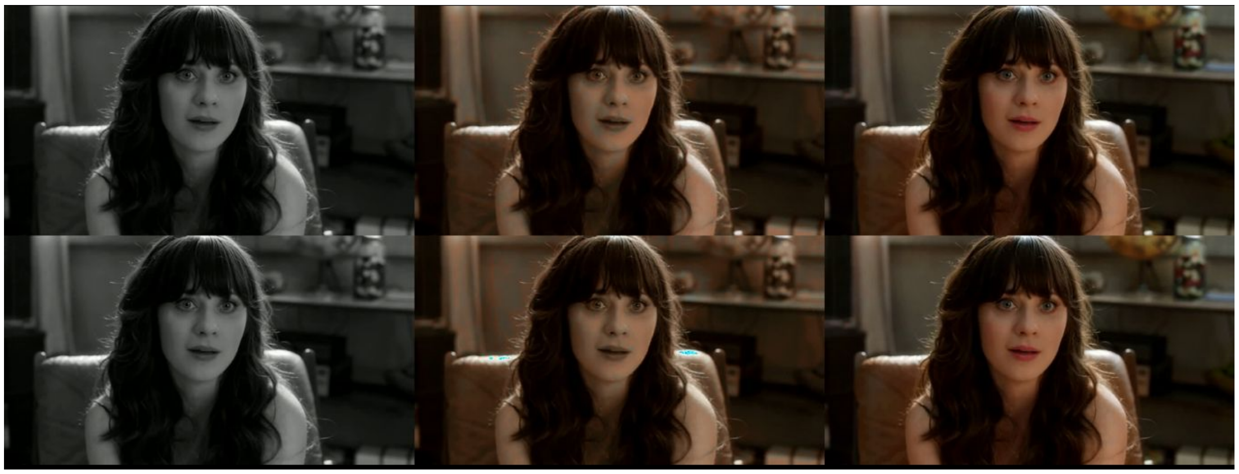

Image 5: Output of gray to color algorithm

VII Conclusion

For this project two problems are solved and each has two ways of doing it. One way is ordinary deep autoencoder and another one is convolutional autoencoder.

Outputs do not represent the original data exactly because autoencoders are "lossy".

For denoising problem deep autoencoder gives better results because convolutional sometimes leaves some of the noise.

For gray to rgb to problem convolutional autoencoder gives better results comparing to deep autoencoders in terms of color transitions because gray area can be seen around girls lips in deep autoencoder. But on the other hand there is also blue shine above girl's left shoulder in convolutional autoencoder.

References

[1] - Nutritional Modeling for Pigs and Poultry - N.K. Sakomura; R.M. Gous; I. Kyriazakis; L. Hauschild;
- Artificial Neural Networks - A.S. Ferraudo

[2] - Building Autoencoders in Keras