Applying Auto-Encoders To Make "Virtual Graphics AcceleratorUnit"

Kamran Gasimov

Abstract

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand.Computer vision works much the same as human vision, except humans have a headstart. Human sight has the advantage of lifetimes of context to train how to tell objects apart,how far away they are, whether they are moving and whether there is something wrong in an image.

Key words: Auto-Encoders, digital images, videos, Computer vision,

Computer vision trains machines to perform these functions, but it has to do it in muchless time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.Computer vision is used in industries ranging from energy and utilities to manufacturing and automotive – and the market is continuing to grow. It is expected to reachUSD 48.6 -13824 by 2022.[1]

An auto-encoder is a type of artificial neural network [2] used to learn data encodingsin an unsupervised manner.

The aim of an auto-encoder is to learn a lower-dimensional representation (encoding) for a higher-dimensional data, typically for dimensionality reduction, by trainingthe network [2] to capture the most important parts of the input image.

The architecture of auto-encoders

Let us start with a quick overview of auto-encoders' architecture.Auto-encoders consist of 3 parts:

Encoder: A module that compresses the train-validate-test set input data [2]into an encoded representation that is typically several orders of magnitude smaller than the input data.

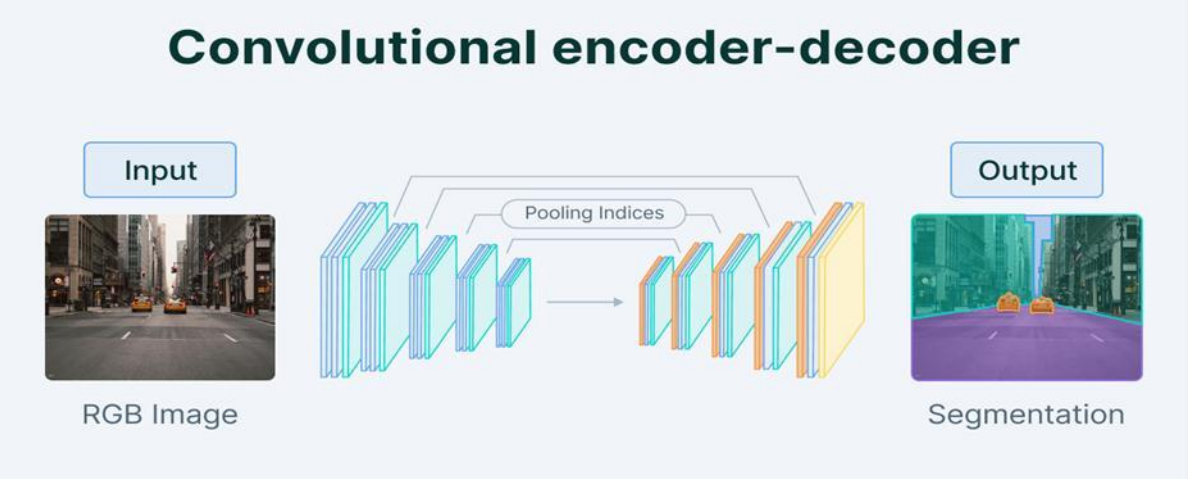
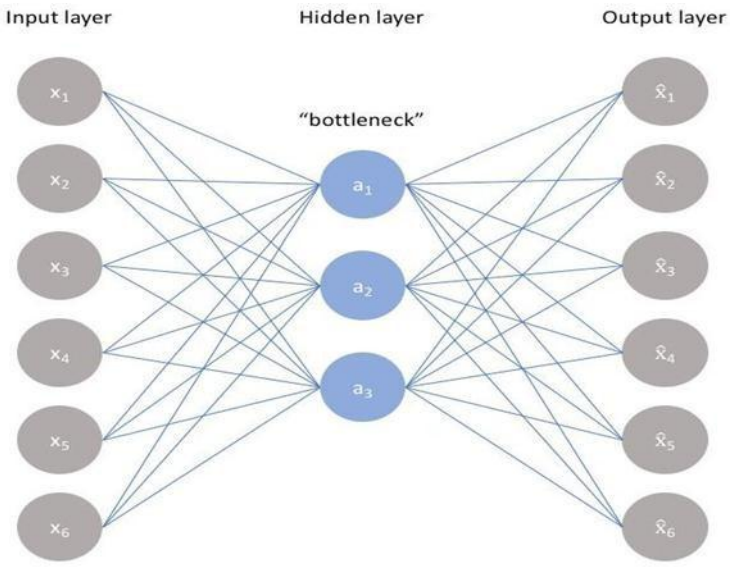


Fig.1. Convolutional encoder-decoder.

Bottleneck: A module that contains the compressed knowledge representations and is therefore the most important part of the network.

Decoder: A module that helps the network "decompress" the knowledge representations and reconstructs the data back from its encoded form. Theoutput then compared with a ground truth. The architecture as a whole looks something like this:



The relationship between the Encoder, Bottleneck, and DecoderEncoder

The encoder is a set of convolutional blocks followed by pooling modules that compress the input to the model into a compact section called the bottleneck.

The bottleneck followed by the decoder that consists of a series of upsampling modules to bring the compressed feature back into the form of an image. In case of simple autoencoders, the output iexpected to be the same as the input with reduced noise.However, for variational autoencoders it is a completely new image, formed with information the model provided as input.

Bottleneck

The most important part of the neural network, and ironically the smallest one, is thebottleneck. The bottleneck exists to restrict the flow of information to the decoder from the encoder, thus, allowing only the most vital information to pass through. Since the bottleneck designed in such a way that the maximum information possessed by an image captured in it, we can say that the bottleneck helps us form a knowledge-representation of the input. Thus, the encoder-decoder structure helps us extract the most from an image in the form of data and establish useful correlations between various inputs within the network.A bottleneck as a compressed representation of the input further prevents the neuralnetwork from memorising the input and overfitting on the data.

As a rule of thumb, remember this: the smaller the bottleneck, the lower the risk of overfitting.

However, very small bottlenecks would restrict the amount of information storable, which increases the chances of important information slipping out through the pooling layers of the encoder.

Decoder

Finally, the decoder is a set of upsampling and convolutional blocks that reconstructs the bottleneck's output. Since the input to the decoder is a compressed knowledge representation, the decoder serves as a "decompressor" and builds back the image from its latent attributes. [2] In computers, while displaying 3d objects on screen computers need to do too many complex calculations like lighting (To achieve good real lighting, computers need to project infinitely many light rays to calculate how much light bounces from the surfaces of the objects) and shadows of the scene. Due to its complexity in early and later computers can Not display realistic 3d graphics. With the invention of graphics cards, parallel computations emerged and made it possible to render realistic 3d images.

Today graphics card technologies advancing day by day and increasing in capabilities. Despite how our

technologies developed in hardware still it takes much time to render complex scenes. Also as hardware, it is big to fit small spaces. Our approach is to accelerate this process using AI Software, one of the Computer Vision method "Auto- encoders". Rendering low-resolution images are simple task for a modern computer. We can use this to build AI that can take low-resolution rendering and give an output of a high-resolution version of the image. In this way, we can utilize our cpus to boost the render performance of scenes to somewhat GPU quality and gpus to render more complex scenes at 60 FPS (Frame per second) and more. In conclusion, this method eventually leads to the utilization of the hardware wisely. We only use intensive calculations to train our model then we can use this model to predict. Once we have fully trained model we can use this to predict accurate high-resolution of the low-resolution image. To achieve video realistic graphics also Game studios can include pre-trained models in the games compatible with ”VGAU” to run the game high-resolution with out having ultra- powerful gpus. To do this we are using Auto-encoder which trained on low-resolution computer renders to generate a high-resolution version of the input. We call this “VGAU” (Virtual Graphics Acceleration Unit). To back our idea we have build one demo of “VGAU” that takes low-resolution images to upscale it. To make it easier to train our AI we need to resize all the images in our dataset to 256x256 size.

```
SIZE=256
img_data=[]
img_data_output=[]

#Input
img=cv2.imread('gul8.png', 1)    #Change 1 to 0 for grey images
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #Changing BGR to RGB to show images in true colors
img=cv2.resize(img,(SIZE, SIZE))
img_data.append(tf.keras.preprocessing.image.img_to_array(img))

img_array = np.reshape(img_data, (len(img_data), SIZE, SIZE, 3))
img_array = img_array.astype('float32') / 255.


#Output
img = cv2.imread('gul.png', 1)    #Change 1 to 0 for grey images
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #Changing BGR to RGB to show images in true colors
img=cv2.resize(img,(SIZE, SIZE))
img_data_output.append(tf.keras.preprocessing.image.img_to_array(img))

img_array_output = np.reshape(img_data_output, (len(img_data_output), SIZE, SIZE, 3))
img_array_output = img_array_output.astype('float32') / 255.
```

Fig. 4. In this code snippet, we are importing images and preparing to ready to feed in to the model.

```
model = Sequential()
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', input_shape=(SIZE, SIZE, 3)))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))


model.add(MaxPooling2D((2, 2), padding='same'))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(3, (3, 3), activation='relu', padding='same'))

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
model.summary()
```

Fig.5. Summery of Auto-encoder model we created – Code snipped written in Python.

Our model consist of:Total params: 108,675.Trainable params: 108,675.Non-trainable params:0Original forms of the images we used in our training process:

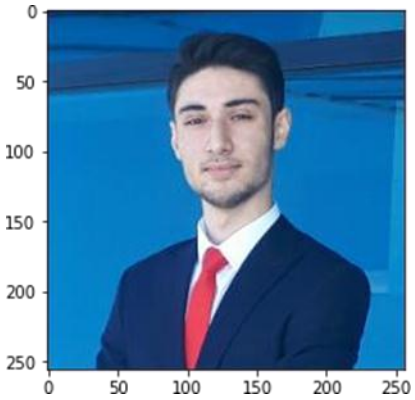


Figure. 5. Real life image.

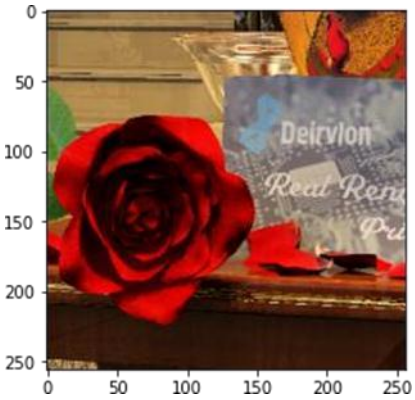
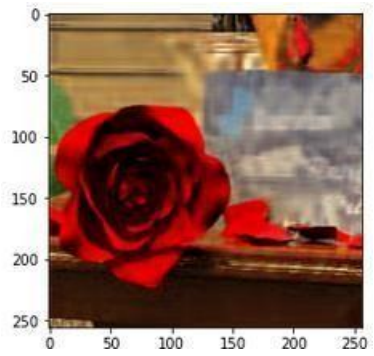


Figure.6. 3D Rendered image.

This image generated in worlds one of the advanced 3D Model creation and rendering program 3Ds Max from Autodesk.
It took ~30 Minutes rendering with size of900x900 and this image quality.
These are the results over epoch counts how our model trained:
After training 100 Epochs:loss: 0.0082; accuracy: 0.9374



Conclusion

We have achieved to acceleration of rendering 3D graphics by further processing the low resolution of the image. By this way, we can play graphic intensive games with high resolution on moderate computer or we can archive more realistic render in seconds with out waiting minutes or hours in 3D rendering applications.

References

[1] https://www.ibm.com/topics/computer-vision
[2] https://www.v7labs.com/blog/autoencoders-guide