Image Inpainting with Auto Encoder

Gilad Shotland Kfir Goldfarb

Ariel University, Ariel

Abstract

Since the Generative Adversarial Network model which was shown by Ian Goodfellow, generative models became some of the most discussed and promising tasks in the world of Artificial Intelligence and Deep Learning. One use of generative models' use is image inpainting. The need for image inpainting and feeling holes is spreads on many fields, such as general public purposes as photobombing, fixing images, or scientific as image completion of samples for researching. This paper aim is to examine the power of less complex model – The Auto Encoder, for image inpainting.
In this paper's approach, the model is a feed forward fully convolutional (and transpose convolutional) neural network, which can process a picture with a hole in it and generate the missing part with some contextual attention.

1.Introduction

In most cases, images are represented in the form of RGB matrix, which can be related as three tables of values, each entry of a table represents the strength of one of three colors, red, green and blue, each table for one color, where the values are between 0 and 255. This fact gives the ability of computing a value of a certain pixel, where its neighbors are given, assuming that there is a connection between different pixels' values.

The main challenge in filling holes, is making the computation in a contextual way. A human eye can easily understand that for a given picture, the best completion for a hole is drawing a mouth or a wheel, for a computational model, the most common thing to do is fill the missing pixels with the mean value of their surroundings. This is where deep learning can be used. The last 10 years brought growth in the use of deep learning in computer vision. This paper examines the use of Convolutional Auto Encoder. This model is based on two main parts – The Encoder, which can be the convolutional layers and a layer of fully connected neurons, and Decoder, layers of transpose convolution, which means the 'opposite' operation of convolution. The main idea of auto encoder is to try represent all the important details in a given example, and generating the example from this representation. The encoder is actually an approach of dimension reduction, made by the convolution layers and the neurons, where each neuron actually needs to represent a certain property of the example, instead of a line or a circle described in many (more than hundred, or thousand) pixels. The decoder supposes to read the encoded layer, aka the latent space, and re-build the example, with transpose convolutions.

2.Related Work

In the field of image inpainting with deep learning, the related work can be split into two main branches. First branch is the GAN based image inpainting, such as inpainting with AI (Towards Data Sciencet) , or Generative Image Inpainting with Contextual Attention . This approach takes the generative model and add a discriminator for the training session, which purpose is to classify the outputs of the generating model, that are given with some real-world examples, where the generator aims to 'fool' the discriminator and made it classify the outputs as real-world examples, and the discriminator aims to classify the outputs as fake. This has shown as a very powerful training approach, but also very difficult to maintain, although the results can be amazing, especially with high resolution images.

Second branch is the only Auto Encoder approach, such as Introduction to Image Inpainting with Machine Learning (Weights and Biases) which works for pretty low resolution images or Image Inpainting Using Auto Encoder and Guided Selection of Predicted Pixels which examines the option of completing multiple not-so-big holes.

3.Required Background

Basic knowledge in applied math such as Probability, Calculus, Linear Algebra.

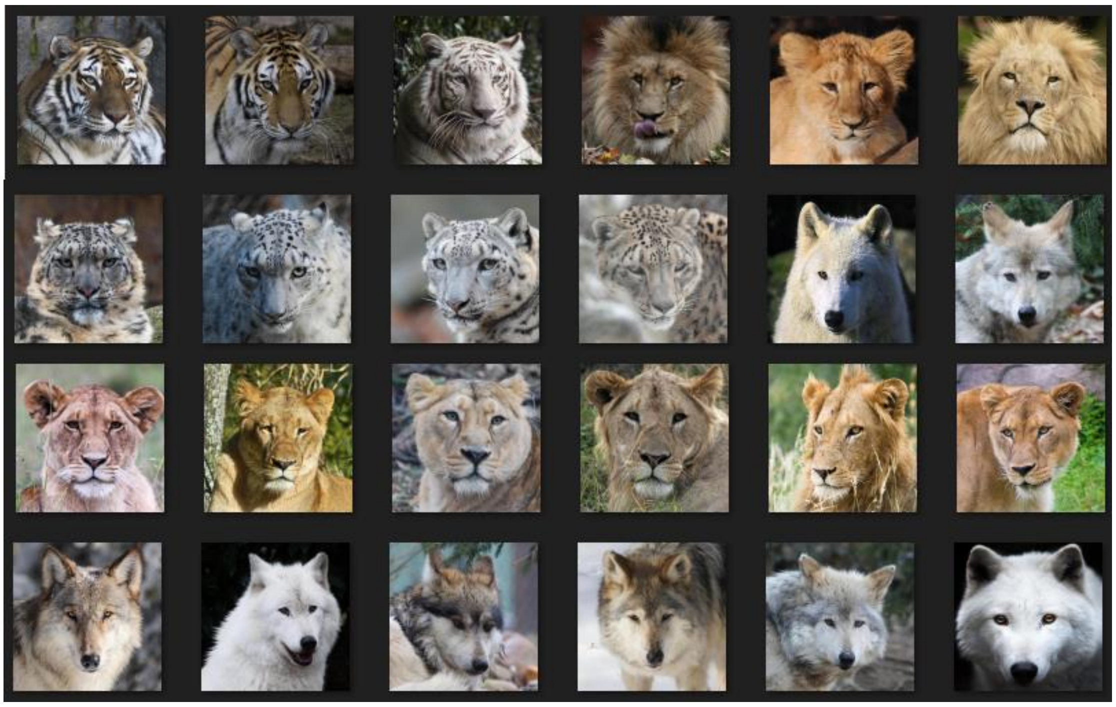
Knowledge in Machine Learning core topics such as Optimization, Convolutional Neural Networks and Auto Encoders.

4.Project Description - Image Inpainting with Auto Encoder Implementation

4.1 The Dataset

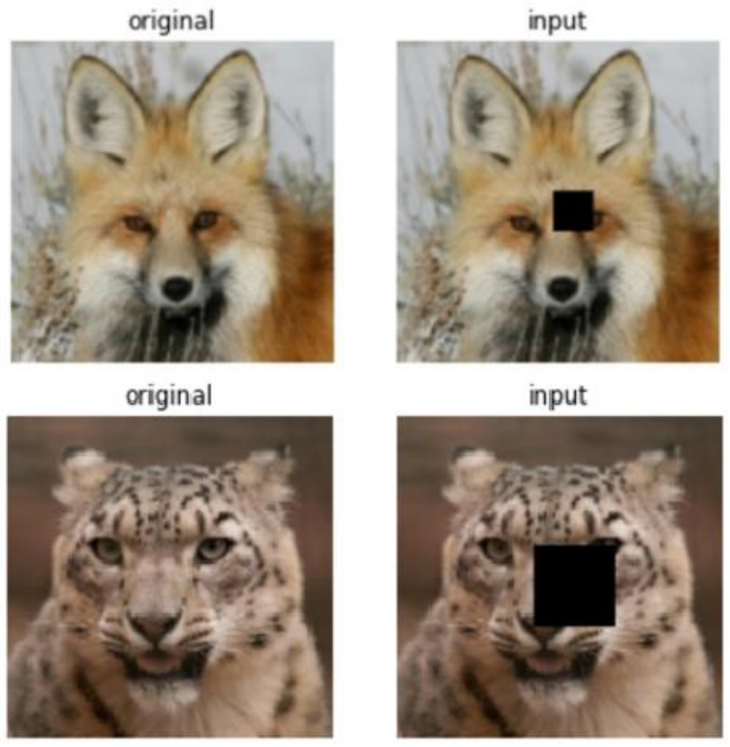
The work of this paper took a dataset of Kaggle's Animal Faces and used the 'wild' section of it. The motivation behind this data set is taking set of same 'distributed' pictures, which means they all contain a set of properties in common, such as eyes, ears, nose, mouth and train the Auto Encoder to complete a missing part of an animal face, where the interesting parts of examining the model is to watch how it tries to complete an eye, nose or leopard spots (for simplifying the model and faster learning, the images were resized from 512x512 to 128x128).

The data was split into 3341 training set and 1400 validation set.



4.2 Generating Holes in Data

For the training part of the project, the pre processing included generating holes in an exact place of each picture. Such that a given image with a hole is the input for the model, and the label is the missing part from the original image. The project took two stages, first stage of completing 16x16 holes and second stage of 32x32 holes



4.3 The Model

The main factor of the model is the convolutions and transpose convolutions. The architecture of CNN makes the effect of 'Dimension Reduction', where the matrices multiplication takes the part of representing groups of pixels in a single value, moving to representing larger groups of pixels and finally the latent space which represents what is in the image.

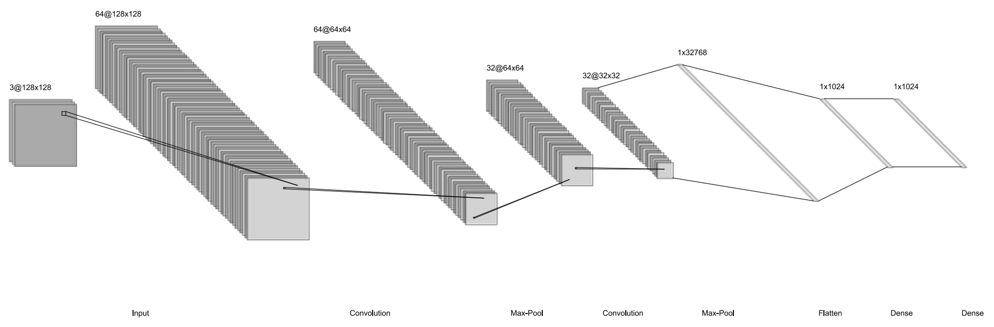
The idea behind the architecture of the Encoder-Decoder was taking the whole image, encoding it and then output the missing part, aiming that the decoder will generate the missing part with attention to the total context of the image. For example, if an eye is missing, the decoder will generate an eye.

The architecture of the Encoder, which is a simple Convolutional Neural Network, was built of 3 convolutional and max pooling layers, and 2 dense layers of 2048 neurons as narrow part of the hourglass shaped model.

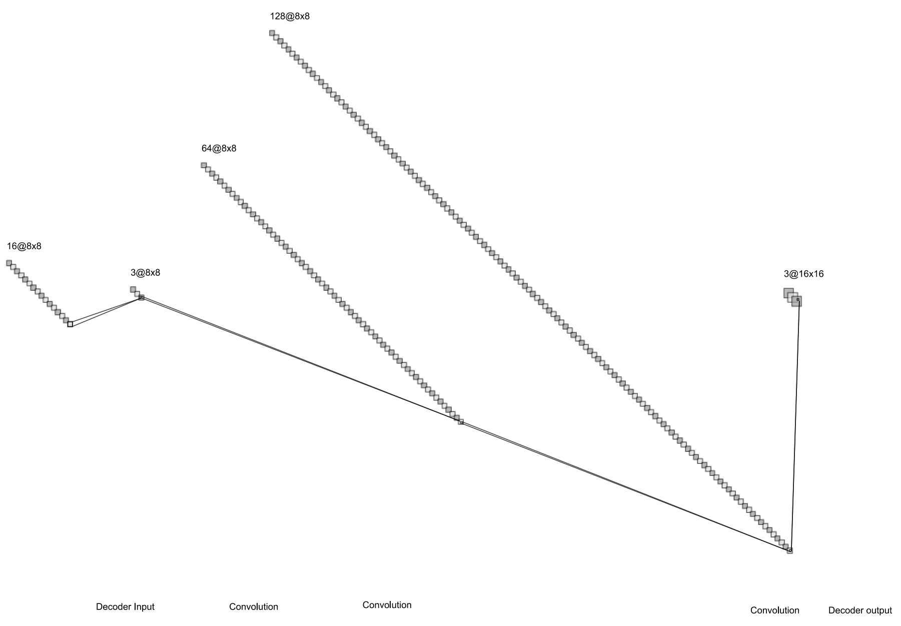
The architecture of the Decoder was built of transpose convolutional layers, where in the 16x16 generating decoder, it took 4 layers, and in the stage of 32x32 it took 5, such that the model will output 16x16 and 32x32 images respectively.

The following pictures showing the model's architecture:

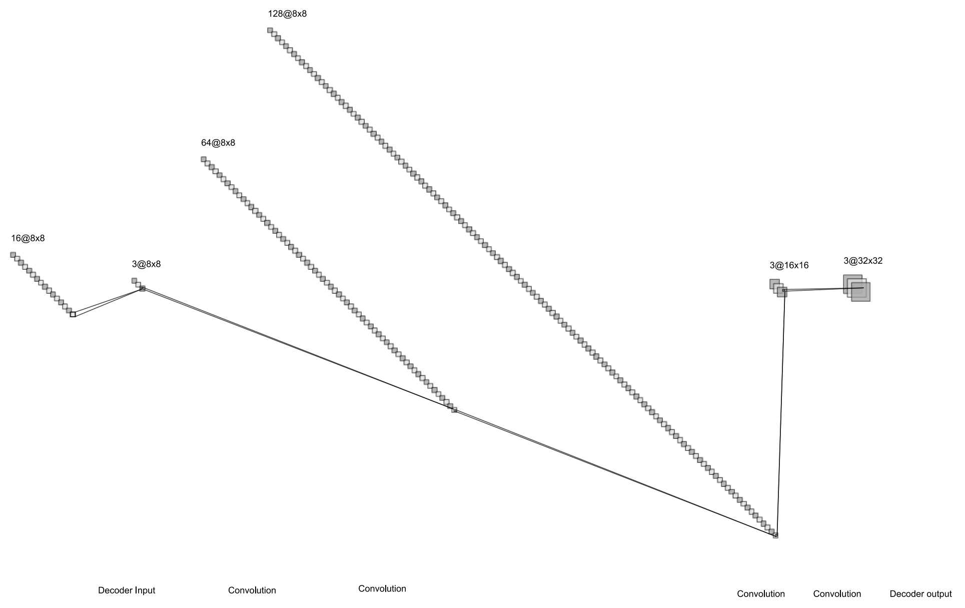
The architecture of the Encoder:



The architecture of the Decoder (output of 16X16 pixel image):



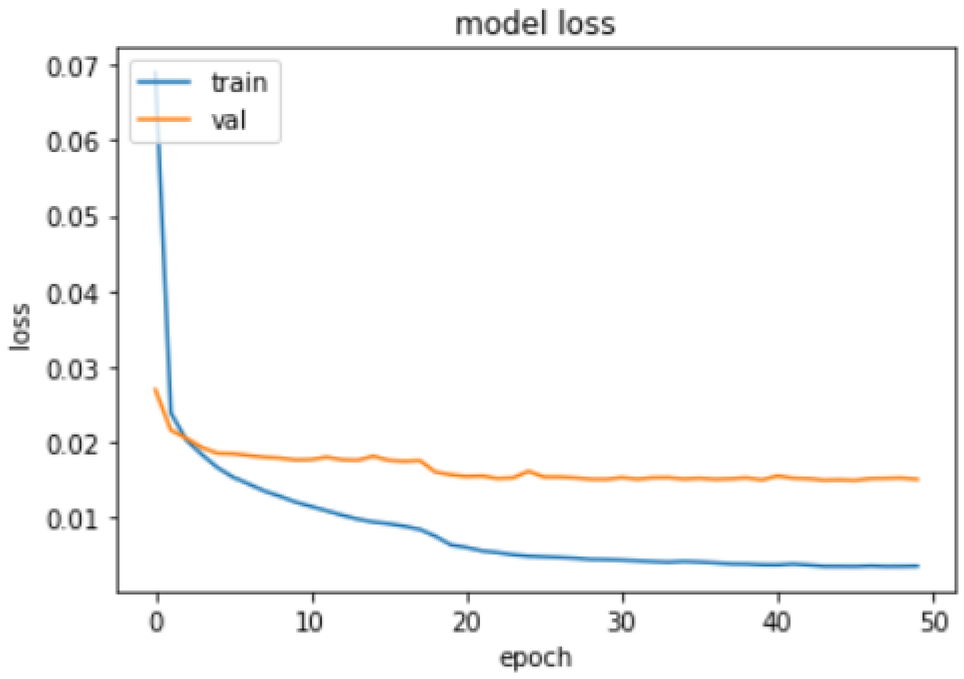
The architecture of the Decoder (output of 32X32 pixel image):



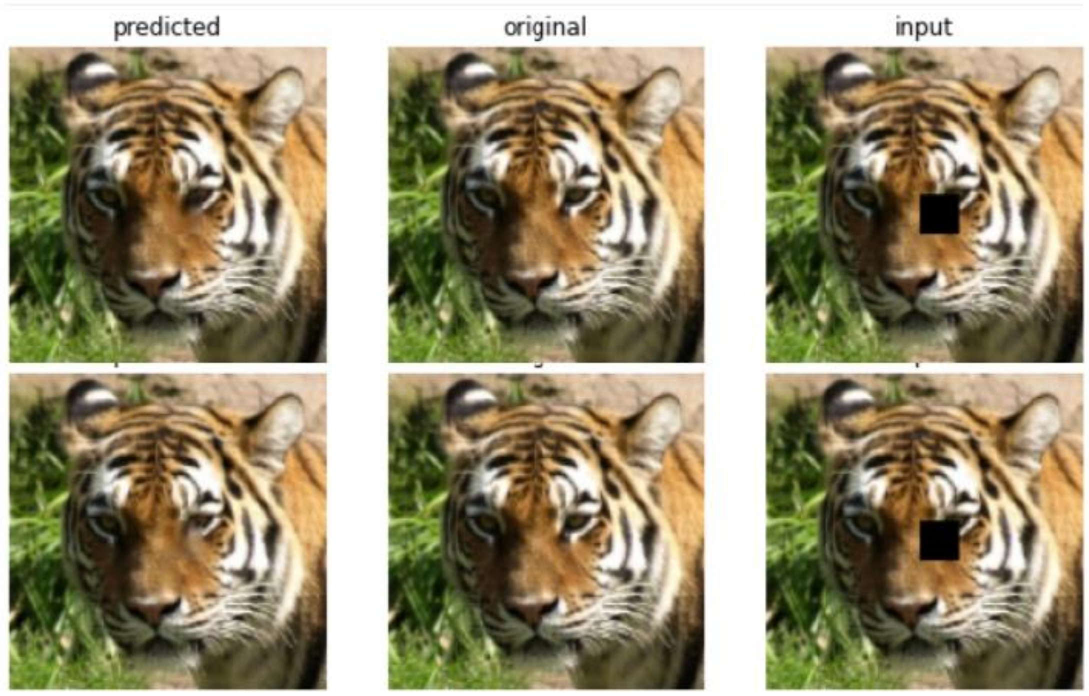
As we can see the main architecture of how our big network is built is trying to form the shape a funnel, this is for deleting the noise - some background object we don't want to learn and in the other hand learn about the main object and shape of the animal faces.

4.4 Training

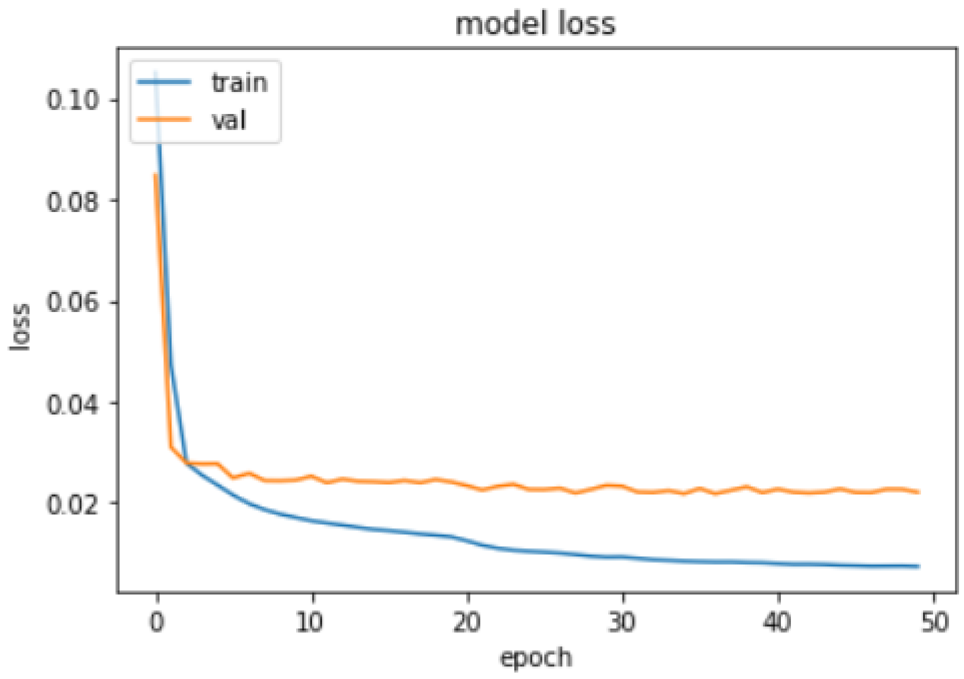
In the 16x16 stage, for the training session, the Adam optimizer was picked as conclusion from previous deliverables, with learning rate of 0.001. The training itself took 50 epochs with batch size of 10.



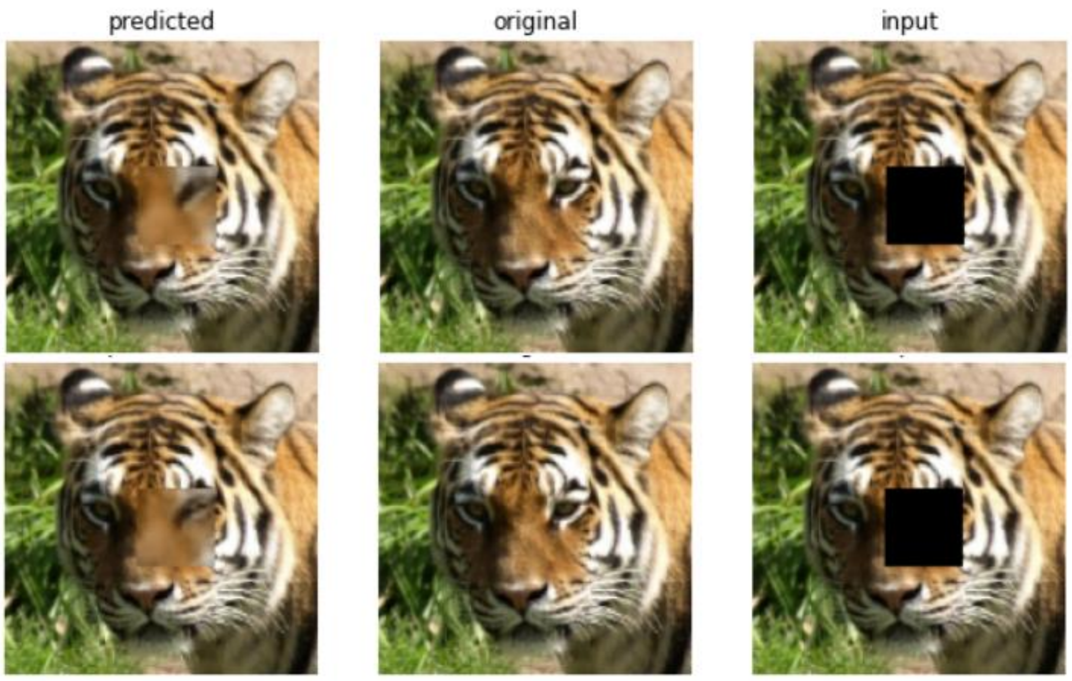
Although after 20 or 30 epochs it looks like there is no loss improvement, it is clear that the model continues in its learning of shapes, as proven in the following example that compares between 30 epochs and 50 epochs. It can easily see that after 50 epochs the model also drawing a gentle line under the eye of the tiger.



The 32x32 stage also took Adam optimizer with 0.001 learning rate and batch size of 10.



The same phenomenon of first stage, of no real loss improvement on the test set but with better visual output happened here. As can be seen that after 50 epochs the model starting to draw the 'apple in the eye' of the tiger more clearly.

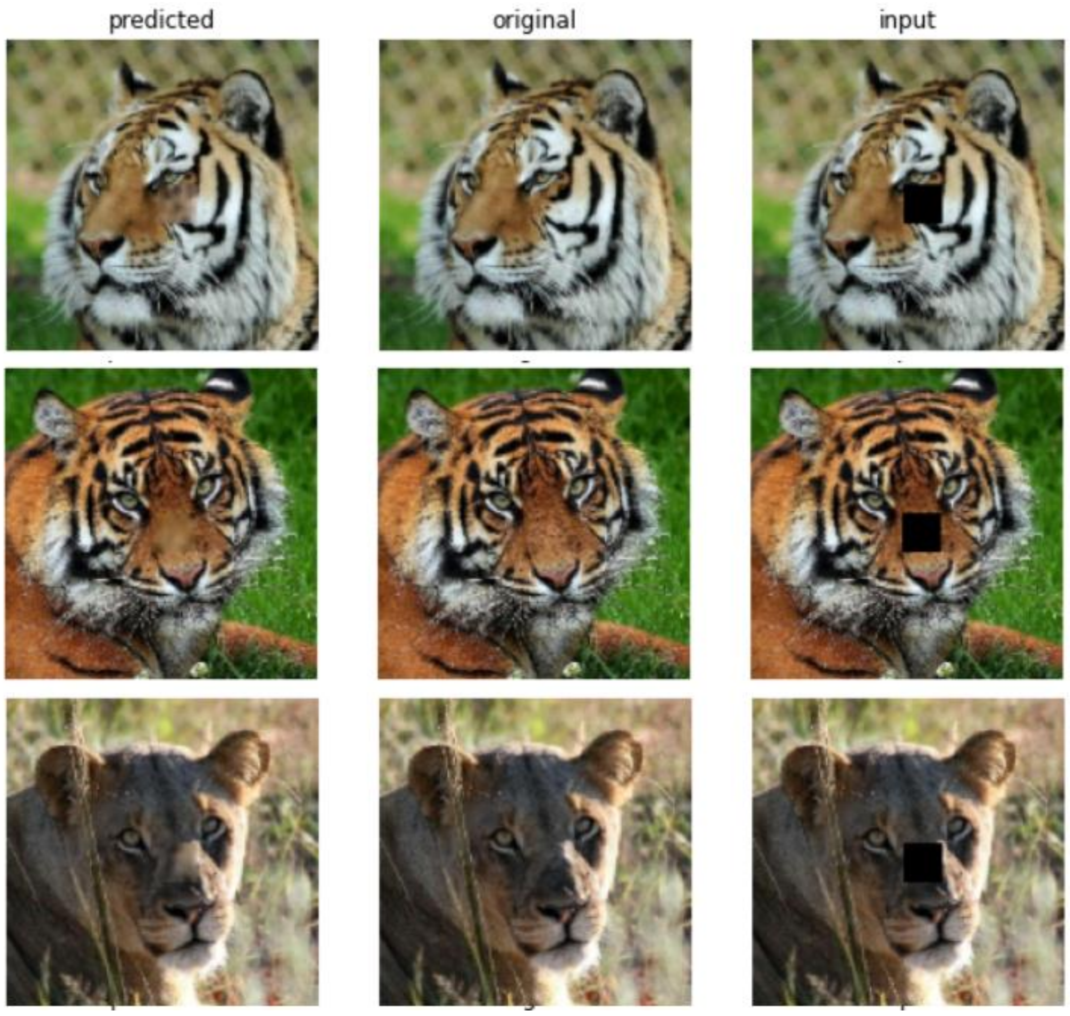


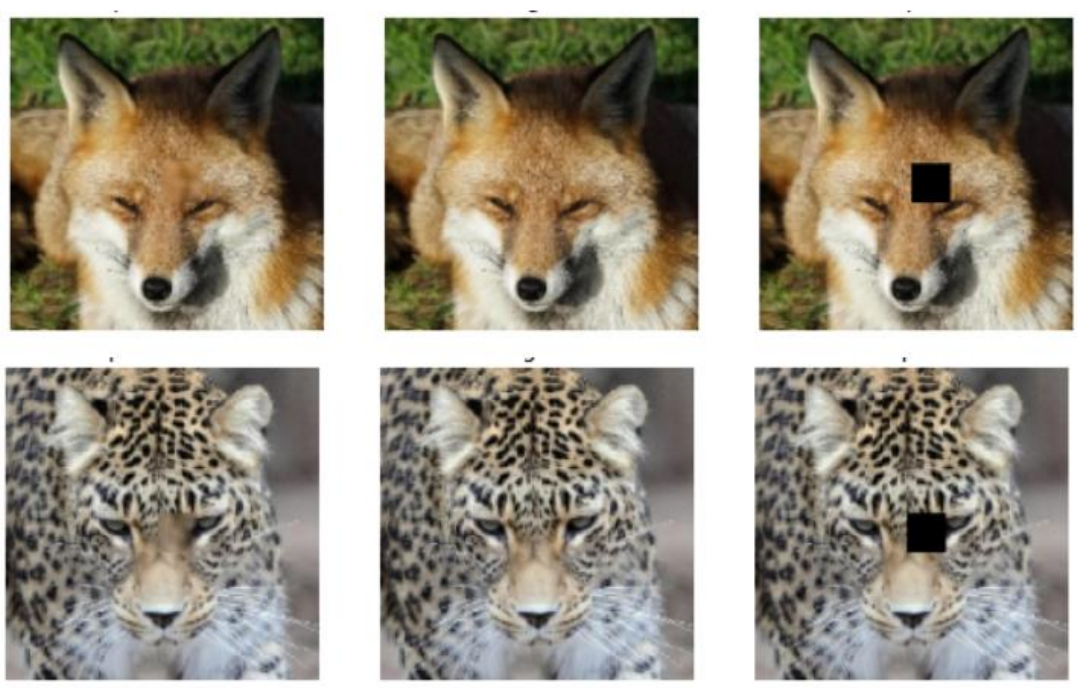
5.Simulation Results

The following examples from the test set exhibit that the model in some cases just tries to fit the missing part to the pattern of its surroundings.

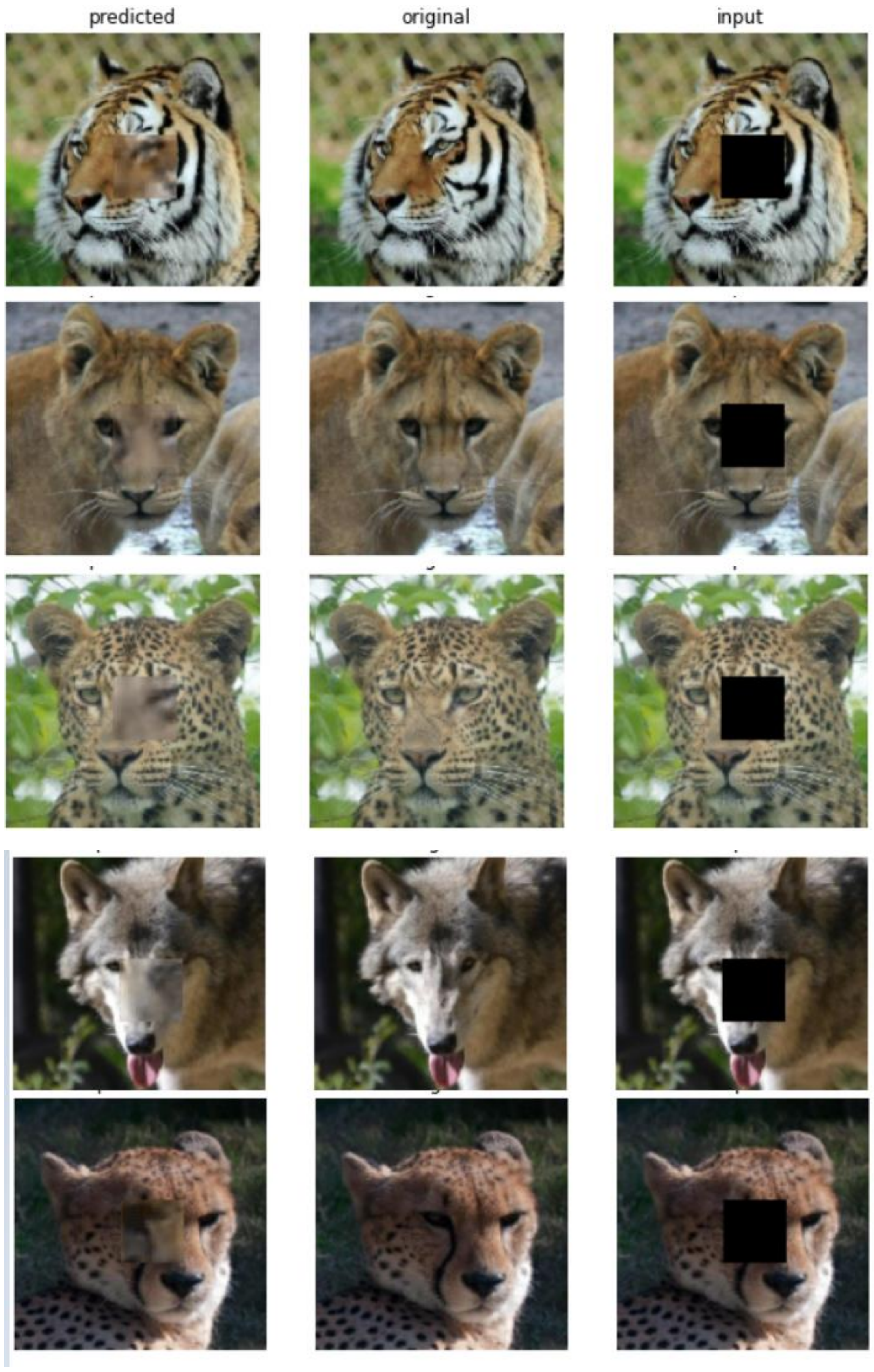
In other cases, the model completes lines and circles in the animal's face. In other cases, the model actually completed an eye or a nose. One can definitely tell that the model has hard times with tigers because of if their spots.

16x16 Test Results:





32x32 Test Results:


predicted
original
input

6.Conclusions and Discussion

The results above illustrate the power of Auto Encoder for completion of pictures. The model can restore/generate lines, circles and even part of a face in some cases, but when the model is tested with more detailed sample as a tiger with spots it fails, and there are some cases when the model just blurring the missing hole and making a blind lion with missing eye or a mute wolf with no mouth. This leads to future work of training a heavier and powerful model, when the options are using famous CNN's architecture such as AlexNet or GoogleNet, etc.

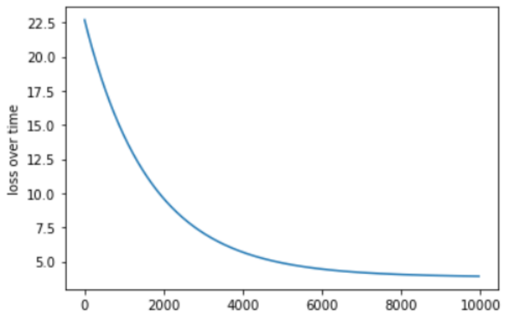
Another path of developing these achievements is generating all the holes with random size and random place for making the training and learning harder.

Additionally, the GAN model that was mentioned earlier is pretty expensive in terms of training. One can examine if training an Autoencoder firstly, then 'connecting' a discriminator and continue training like GAN will be cheaper in training time then the regular method of Generator-Discriminator from the start.

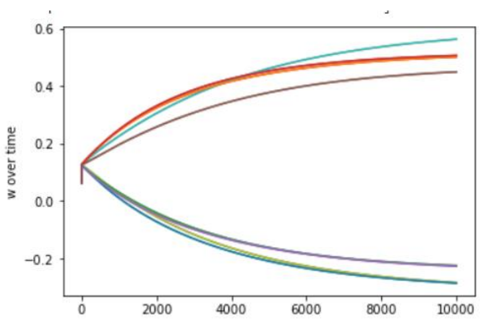
7.Previous Attempts

7.1 Linear Regression

Before we came to this approach, we developed a simple linear regression model using TensorFlow 1 that take a photo and separate it to 3X3 size metrices, each matrix has 3 channels value of RGB colours, for each matrix the model take one channel, and trying to predict the value of the middle pixel with other pixels as evidence, we assumed that all 3 channel are distribution the same, this model is built with Gradient Descent Optimizer with 0.00001 learning rate, the loss accuracy we got on our train set was 1.9817, and over learning time we could see how the loss converge to 0:



As we can see the weights are converge to the mean over learning time:

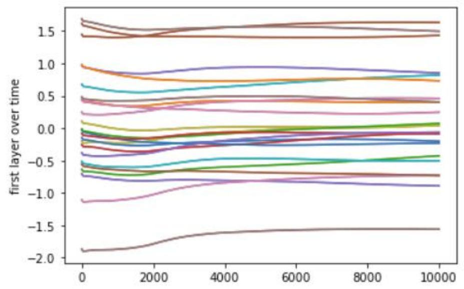


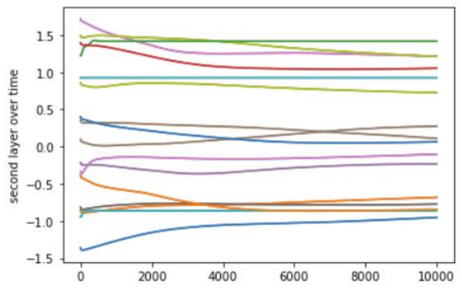
7.2 Neural Network

Another previous approach the implementation of another neural network model that predicts only one pixel value by knowing the 24-pixel values around it. We took an image and divided it into 5X5 matrices, each matrix has three values for the RGB channels, we took the middle pixel in each matrix and built a neural network model to predict the color values of the middle pixel.
First we take two random images, one for our training set and another for the test set (calculating the loss value):

Then, each photo is divided into 25X25 matrices, and each matrix is in turn divided into three different channers for each RGB colour. The model we used was built using the TensorFlow 1 library, and we used Adam Optimizer and a learning rate of 0.001.

The wights of the model over learning:





After training the model with our training set, we calculated the loss value, which is a reasonable value for a linear regression model. The loss value for the training set was 3.5912, since we assumed each colour channel is distribution the same, we calculation the Mean Square Error for each colour channel on the test set image and the results was the following:

Squared test error on the red channel: 7.946

Squared test error on the green channel: 10.499

Squared test error on the blue channel: 18.468

7.3 Current Neural Network

At the current neural network that we talk about in Project Description section, we have tried to change the number and size of the model architecture, but this damaged the loss and the prediction of shapes and object in the images.