# Image Inpainting

David Voihanski Shay Naor

## 1. Abstract

Image inpainting is the process of restoring lost or damaged parts of an image. Today, there are many new ways to use computers to achieve this goal very effectively.
In this project we propose a Convolutional Neural Network for Image inpainting that does not receive information about the location of the abstraction. We will also describe our previous attempts using Linear Regression and Multilayered Perceptron with information about the location of the abstraction.

## 2. Introduction

The objective of image inpainting is to use pixel information from the image to fill an abstracted part of it. It has many real-life applications such as: Removal of elements from images, image restoration, image noise reduction etc.
There are algorithms that receive information about the location of the abstracted part to fill the hole using this information. Our method does not rely on information about the abstracted location, the thesis is that preliminary information may limit the results. We will describe mainly our best result using a Convolutional Neural Network. Then we will show our previous attempts using weaker models and how we improved the results.

## 3. Related work and required background

- Auto-Encoder with GAN
  Use a Convolution Neural Network to encode an image with abstractions, then a fully connected layer to decode it without the abstractions. Then use a classifier that receives both real images and images decoded by the fully-connected network and determines whether they are real or fake. This way both networks improve over time and result in visually pleasing generated images.

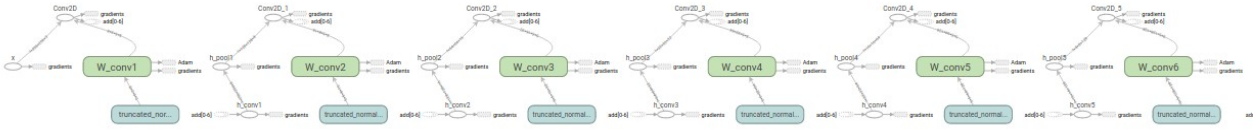
# 4. Project description

- Dataset description
  We used Places365's dataset.
  Taken from: http://places2.csail.mit.edu/download.html
  Our dataset contains 365,000 256X256 resolution images, 328,500 for training and 36,500 for testing.

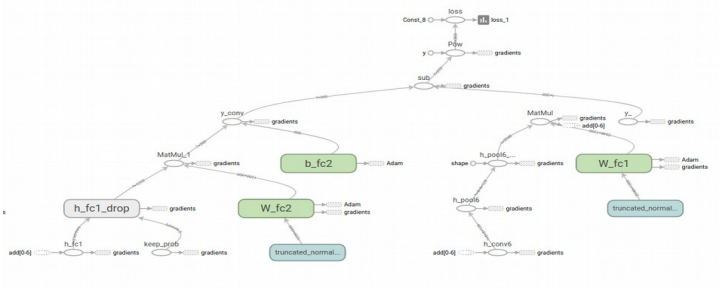
- Pre-processing
  We split both the test and the train folders to sub-folders of size 50, a total of 6570 folders in the train folder and 730 in the test. Each batch we read a folder and shuffle the images. Then for each image we generate 2 random indexes from the range [31, 215], create a copy of the image, and make a 10*10 hole starting from those indexes.

- Labels
  The label for each image is a vector of 300 expected pixel values taken from the image.

- Our model
  Our input is a batch of 50 RGB images- 256X256 resolution with a 10X10 hole in a random position. The images are inputted to the convolutional layers for feature extraction. We have a total of 6 convolution layers with "same" padding and stride of 1X1.
  The first convolution layer has 8 kernels of size 5X5X3.
  The second convolution layer has 16 kernels of size 5X5X8.
  The third convolution layer has 32 kernels of size 5X5X16.
  The fourth convolution layer has 64 kernels of size 5X5X32.
  The fifth convolution layer has 128 kernels of size 5X5X64.
  The sixth convolution layer has 128 kernels of size 5X5X128.
  Between each layer we use ReLU activation function and do 2X2 max pooling.
  The convolutions' output is 128- 4X4 matrices.




  We then flatten the convolutions' output and use it as input for the fully connected layers.
  We have 2 fully connected layers; the first layer has 1200 neurons each taking the convolutions' output as input.
  Then we pass the first layer's output to the 300 neurons in the second layer and use linear regression to predict 300 pixel values which correlate to the 10X10X3 hole we made.




For the loss function we used Mean-Squared-Error, comparing the expected pixel values from the model's prediction.
We use Adam Optimizer for learning with a learning rate of 0.0001.
We have:
(5*5*3*8+8)+(5*5*8*16+16)+(5*5*16*32+32)+(5*5*32*64+64)+(5*5*64*128+128)+(5*5*128*128+128) = 682,576 weights in the convolution layers and (4*4*128*1200+1200)+(1200*300+300) = 2,819,100 weights in the fully connected layers.
That's a total of 5,638,200 weights.
We let the models train for 4 epochs.

# 5. Experiments/simulation results

In the beginning we let the model train for 2 epochs, and we noticed it was underfitted, so we let it train for 4 epochs. First we tried 3 convolution layers beginning with 32 kernels and multiplying by 2 every layer. We experimented a lot with the number of convolution layers, kernels and pooling. In the end we found that initiating with a smaller number of kernels and adding more convolution layers gives better results. We also noticed that the fully connected layers have a strong impact on the results. We started with 2 layers- 400 neurons in the first layer with 50% dropout probability and 300 in the second layer. We then tried increasing the number of layers to 3- and noticed no significant improvement in the results. On the other hand, increasing the number of neurons in the first layer significantly improved the results.
We noticed the predictions were monotonous and lacked textures, so we tried changing the dropout rate to 0%, which greatly improved the results.

- CNN results:

- Train Tensorboard loss graph:

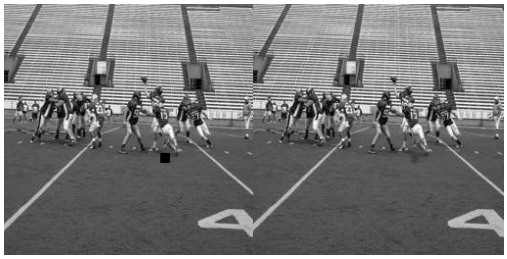

The average loss for all test images was 26.8.

# 6. conclusion

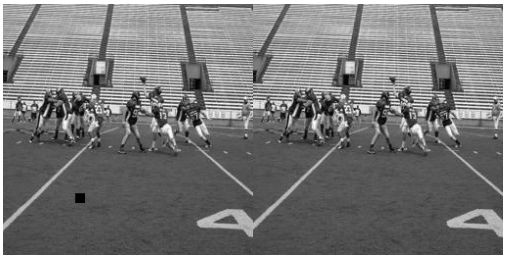
We found that Convolution Neural Network is an effective model for image inpainting, which gives good visual results. We tested a very wide variety of pictures, and got good results in many pictures, and found it fails with complicated textures.
Future research idea for improvement- Use Auto Encoder and GAN, handle multiple holes in different sizes.

# 7. Comparison to Linear Regression and MLP

- Linear Regression results:



MLP results:



In the Linear Regression and MLP the features we took were dependent on the location of the whole, thus making it “weaker”.
We only took 780 pixel values around the hole as features, compared to the whole image in CNN.
In these models we predict each pixel value in iterations, this meant we took our previous predictions as features, in CNN the output was 300 pixel values correlating to the hole.
With these models we only tried to fill gray-scale images, compared to RGB in CNN.