Reviving Dead Pixels

CNN-Based Image Reconstruction

Connor Malley
University of Central Florida
Orlando, FL, USA
malleyconnor@knights.ucf.edu

I. Abstract

In this paper, I develop and test a CNN-based approach to predict a missing patch of pixels in an image, given the image with the patch removed. With an accurate model, one could selectively remove unwanted small-scale anomalies from an image, such as a pimple on a face, a piece of trash from an unwanted scene, or really any unwanted ”noise” that can be bounded by some small box (relative to the size of the image). I trained, tested, and optimized the hyper-parameters of multiple CNN architectures. I found that many configurations tended to be unstable, where either the loss did not converge, or the test loss varied frequently. This was largely an effect of too high or too low of a learning rate, since even for a small patch of size $(4x4)$, the model is actually generating $4 * 4 * 3 = 48$ outputs for an RGB image, and the loss is calculated using the mean squared error between the predicted patch and the original patch. I also found that the activation function and dropout played a big factor in the stable convergence of the test loss as well. However, if the hyper-parameters are chosen carefully, I found that some models were able to achieve an average euclidean distance of about 41 from the original (4×4) patch, in RGB space.

II. Introduction

Typically, image reconstruction is thought of as converting a digital image from some sampling domain to the image domain (real two-dimensional coordinates, with a multiple possible color dimensions). One example of this is Sound Navigation and Ranging (SONAR), which takes a time-series of electric signals generated by sound waves, and uses beam-forming with discrete angles to generate the rough shape of a target object. [1] Another example is Magnetic Resonance Imaging (MRI) which creates a 2D image of the human body using rapidly rotating magnets. [2] In this paper, my idea of image reconstruction is a bit different since both the inputs and outputs are in the image domain. The idea is to remove a small patch from an image, and reconstruct the patch using the rest of the image. This would obviously fail to reconstruct the actual image if there was an anomaly contained entirely within the patch, but if you wanted to remove the said anomaly, this method may perform well. This is based on one assumption, that most of the training images do not contain anomalies within the removed patch, something that should be expected since an anomaly in this case is just a small area which is not consistent with the rest of the image.

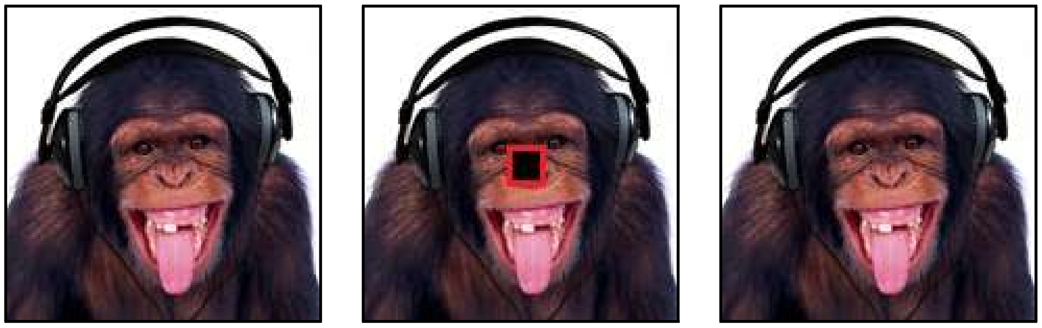


Fig. 1. Example of the image reconstruction. This higher resolution image is just used as an example. Lower resolution (32×32) images were used for training.

III. Method

In this paper, I use a subset of the MIT 80 million tiny images dataset [3] ($n_{train} = 819200$, $n_{test} = 204800$). This dataset consists of (32×32) RGB images. I chose this dataset since it is very low resolution and provides lots of data samples which could quickly train a model, compared to very high resolution images. Also, in real world images of scenes or objects, there may be little correlation between very distant pixels/chunks. For each image I remove a small square patch of size $(m \times m)$ from the middle (1 up and 1 left of the middle if the patch size is odd). I want to keep the window size close to one order of magnitude less than the actual image. Then, I scale the data from a range of $[0, 255]$ to a range of $[0, 1]$, except for the patch, which is filled with a mask value of -1. The original patch from the image is saved off as the label for the prediction which means performing regression on the $(m \times m \times 3)$ patch, where the 3rd dimension comes from the color.

Since the data is low resolution, I used only 2 convolution layers, and 3 dense layers in each of my models. In any case, I use a kernel of size (5×5) in the 1st convolution layer, and a kernel of size (3×3) in the 2nd convolution layer, with (3×3) max pooling and activation after each layer. This results in $(4 \times num\_channels)$ input features to the dense layers, where the number of channels is the same for both convolutions, and the number of output features is the same for all hidden dense layers. The output of the final layer always takes the shape $(m \times m \times 3)$. I also use a batch size of 64, and a learning rate of 0.001 with the Adam optimizer and Mean-Squared Error (MSE) Loss. I found that using any higher of a learning rate, such as 0.01, cause the model to be very unstable or not converge in most cases.

Since the input range is $[0, 1]$, and the input image is what is being predicted, the output range should also be restricted to $[0, 1]$. This is done with a Sigmoid activation after the classification layer. The output prediction can then be multiplied by 255 to generate the actual image.

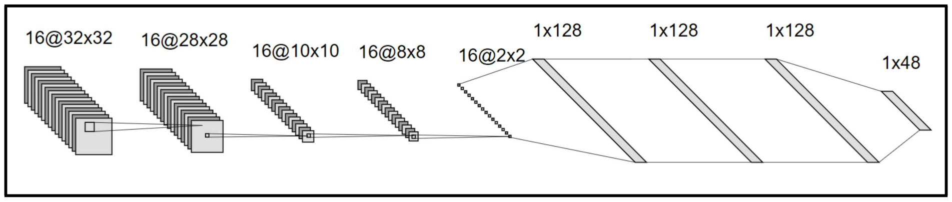


Fig. 2. Example Model with 16 convolution channels, 4*16 input features, and 128 output features, predicting a (4×4) patch. Some dropout/activation is implicit after each layer

For this experiment, many model configurations were tested for patches of each size. I tested various arrangements of $activation = [Sigmoid, ReLU]$, $conv\_channels = [16, 32, 64, 128]$, $out\_features = [128, 512, 1024]$, $cnn\_dropout = [0, 0.25, 0.5]$, and $dense\_dropout = [0.25, 0.5]$ all with a fixed patch size of (3×3) in the ”ablation studies section”. Then based on the results of these experiments, I selected a well-performing model and trained it for predictions for multiple patch sizes with $patch\_size = [1, 2, 3, 4, 5, 6]$. Note that a patch of size 1 corresponds to the simplest case of the singe pixel prediction.

IV. Results

A. Ablation Studies

1) Effect of Activation Function: I found the Sigmoid activation to perform very poorly in this model, with ReLU outperforming it in terms of convergence and stability in nearly every case. This is likely due to vanishing gradients from the Sigmoid activation causing convergence on a local minima with very poor results. Note that Sigmoid still needs to be used to ensure a range of [0,1] for the output predictions.

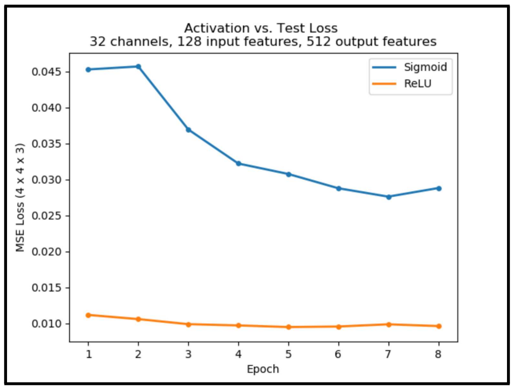
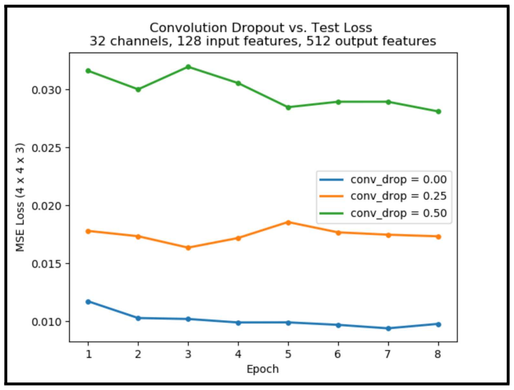


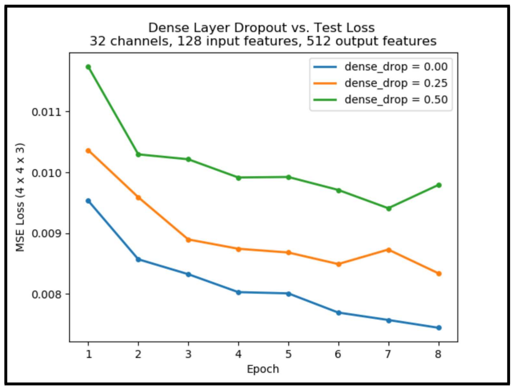
Fig. 3. Activation Function vs. MSE

From Fig. 6, it is clear that the test loss converges much quicker, and with a much lower error, when ReLU activation is used. Therefore for all future experiments in this paper, ReLU is utilized.

2) Effect of Dropout Rate: All models in this experiment utilized 32 convolution channels, and 512 output features in the dense layers, with a learning rate of 0.001 and a batch size of 64 and ReLU activation. I found that a dropout rate of even just 25% in the convolution layers was detrimental to the training process, causing the test loss to have relatively large oscillations between epochs. Obviously no dropout in the convolution layers results



(a) Convolution Dropout (Dense Dropout = 0.5)

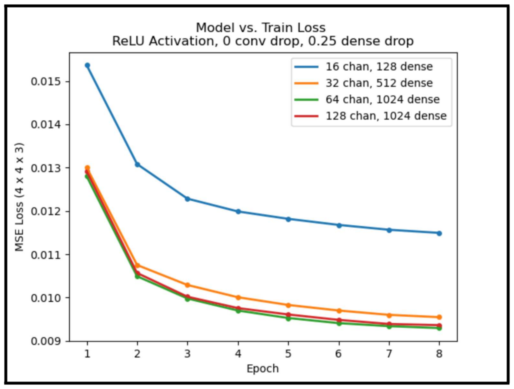


(b) Dense Layer Dropout (Conv Dropout = 0)

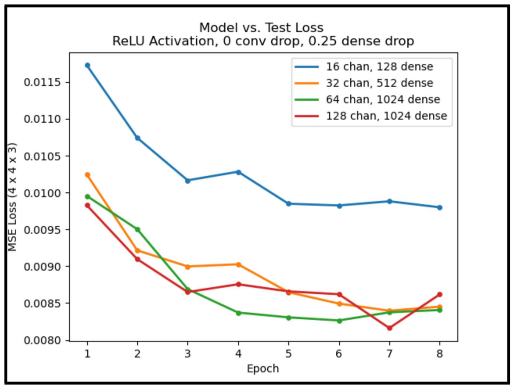
Fig. 4. Effects of dropout rate on MSE

in the best test MSE for this model. This may be expected since the convolution layers typically have many less parameters than the dense layers, and research has found that the effects of dropout in convolution layers is limited. [4] Though, the use of dropout did result in less error when used in the dense layers, partially due to the fact that the training error was also reduced by a large factor when not using any dropout.

3) Effect of Convolution Channels and Output Features: I found that models with a larger number of convolution channels and more output features in the dense layers tended to perform better than smaller models up to a certain point. Though this difference was not terribly significant in most cases. Here I tested different 4 model configurations on a (4×4) patch. Model 1 has 16 convolution channels and 128 dense output features. Model 2 has 32 convolution channels and 512 dense output features. Model 3 has 64 convolution channels and 1024 dense output features. Model 4 has 128 convolution channels and 1024 dense output features. All models use no convolution dropout, 0.25 dense layer dropout, ReLU activation, a learning rate of 0.001, and a batch size of 64.



(a) Train Loss



(b) Test Loss)

Fig. 5. Effect of model size on MSE loss

Obviously the larger models are capable of learning much more complex features than the smaller models, and here this results in lower training losses. However, there are diminishing returns after a certain point, since a 2 convolution layer model can only extract so much information from the pixel features, before a more complex set of features or more layers are needed to lower the loss. [5] Also note that the difference in the test error is negligible for larger models, so I think the model with 64 channels and 1024 dense features wins here because of its low test error and smooth convergence.

B. *Effect of Patch Size*

Here I tested multiple different patch sizes with $patch\_size = [1, 2, 3, 4, 5, 6]$ using a model with 64 convolution channels, 1024 output features in the dense layers, no convolution dropout, 0.5 dense layer dropout, and ReLU activation.

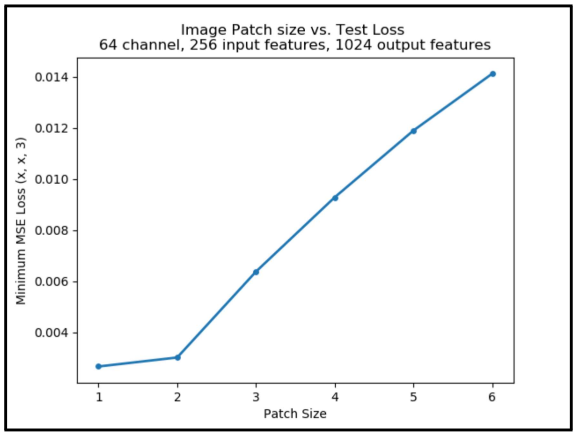


Fig. 6. Patch Size vs. MSE

It is clear that from Fig. 6 that the MSE loss increases almost linearly with an increase in the patch size. I think this is expected, as it is harder to extrapolate the center of a large patch because the center pixels' nearest (spatial) neighbors are further away than the edge pixels' nearest neighbors. Here the ”neighbors” are the pixels directly adjacent to the patch. This makes me wonder if instead predicting $(m \times 1)$ patch adjacent to the image on any side would yield the similar results for each patch size, since each pixel in the patch will be close to some input pixels.

V. Discussion and Analysis

It seems like as it stands, this method of reconstructing a patch may not be scalable to larger, high resolution patches. However, For replacing a large patch in a high resolution image, this method could theoretically be utilized on each $(m \times m)$ sub-patch of the larger patch, performing reflection wherever missing data may be needed. Alternatively, the patch could be changed from a square to a one-dimension column/row, so predictions could be generated from left to right, top to bottom, and then averaging the predictions. I think predicting a single column rather than a square may be more useful, since it was shown in the patch size study that there was a higher error for larger patches. This was likely stemming from the fact that we must extrapolate more for the center pixels than for the edge pixels. In any case, I believe this study was important to demonstrate the idea of patch based image reconstruction. Though, it may also be the case that larger patches may also require larger models, or even just deeper models. This model may also benefit from a more complex set of features, such as edges (since we expect that edges will typically be preserved in the patch unless there is some anomaly). One could also use color histograms to preserve the histograms of those blocks near the patch. Lastly, I think the images could be segmented, where the colors of each segment are fit to some spatial distribution. The segment could then be predicted for each pixel in the patch with a CNN, and the color could be generated from the fitted function for that segment. Obviously due to time constraints, I was not able to test all of these methods, but I think they would be very much worth examining in the future. However, to visualize my implemented method on the MIT 80 Million Tiny Images dataset, there are a few examples of (4×4) square patches being predicted in the appendix.

VI. Conclusion

Obviously this model is far from perfect, as the MSE for just a (4×4) patch is about 0.01. If considering the predicted pixels distance to the original pixel in the euclidean RGB space, this corresponds to an average pixel error of 41.1. This is about 10% of the maximum possible error, which is $\sqrt{3 * 255^2} = 441$. While the model may produce very small errors in some cases, it also may predict pixels that are very inaccurate in other cases. One example is the lack of edge preservation, where the model will just select from an weighted average of the local colors, and use something close to this average for the whole block. This fails to reconstruct any edges coming into or out of the patch, which can be visualized in the 1st test image in the appendix. I believe a simpler set of features is necessary to capture these patterns in images with lots of sharp edges, or many different objects. Separately segmenting the image, and predicting the segment for each patch pixel, could prove to be useful. However, the square patch model still suffers from the fact that the inner pixels are subject to more extrapolation than the outer pixels. Therefore if I were to expand on this project, I would try the proposed segmentation method with column or row based patches, possibly with the outputs of previous columns/rows as additional input. In conclusion, these results still showcase the many challenges that come with multi-label regression, or the generation of complex image data.

References

[1] B. Zerr and B. Stage, "Three-dimensional reconstruction of underwater objects from a sequence of sonar images," in *Proceedings of 3rd IEEE International Conference on Image Processing*, vol. 3, 1996, pp. 927–930 vol.3.

[2] J. A. Fessler, "Model-based image reconstruction for mri," *IEEE Signal Processing Magazine*, vol. 27, no. 4, pp. 81–89, 2010.

[3] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[4] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 648–656.

[5] Y. N. Dauphin and Y. Bengio, "Big neural networks waste capacity," 2013.

Appendix

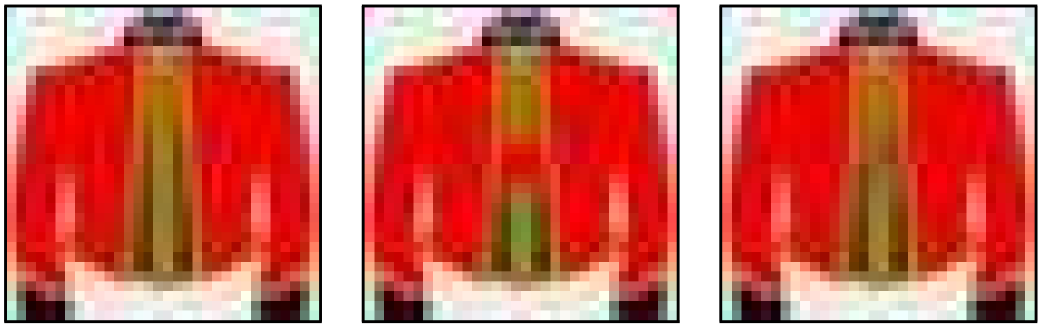

(a) Original Image (b) Masked Image (c) Predicted Image

Fig. 7. test_im0.png

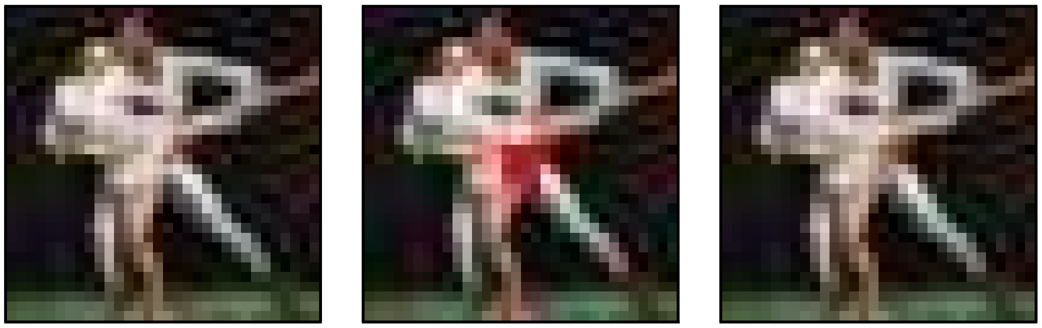

(a) Original Image (b) Masked Image (c) Predicted Image

Fig. 8. test_im1.png

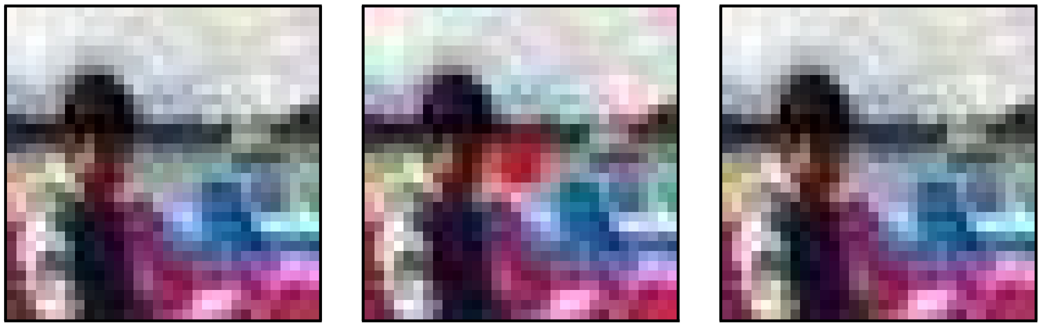
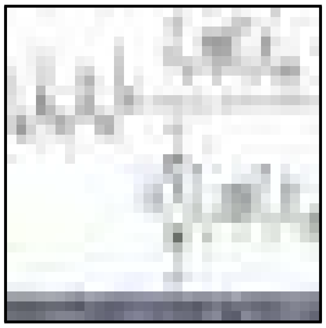
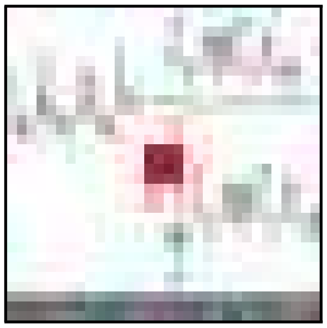

(a) Original Image (b) Masked Image (c) Predicted Image

Fig. 9. test_im2.png


(a) Original Image


(b) Masked Image

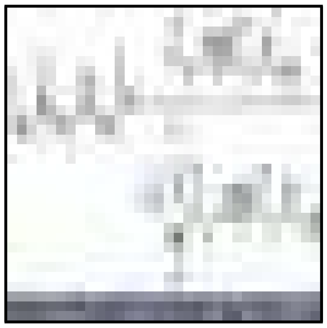
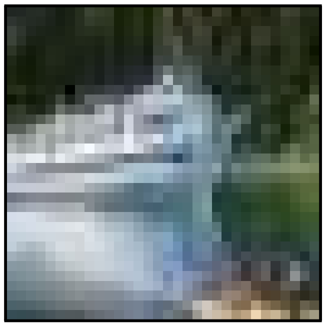
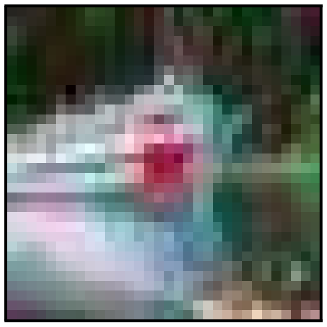

(c) Predicted Image

Fig. 10. test_im3.png


(a) Original Image


(b) Masked Image

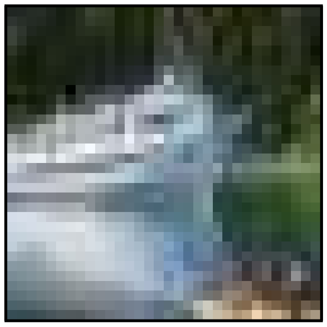

(c) Predicted Image

Fig. 11. test_im4.png