Novel Graph Network Approach to Image Imputation

Rastko Stojsin
Vanderbilt University
rastko.stojsin@vanderbilt.edu

Anthony DeNiro
Vanderbilt University
anthony.r.deniro@vanderbilt.edu

***Abstract*—Image imputation has many applications in image processing from photo restoration and editing, to medical image correction, to applications in computer vision and object permanence. In this paper, we develop two methodologies to impute pixel values in images with the first serving as a baseline and the second serving as an improved final approach. The first approach uses linear regression to imputing selected unknown pixels in an image. We then discuss the drawbacks of this methodology and propose an original and novel alternative. This alternative is a graph networks approach that allows for higher flexibility and performance, but at a cost of high computation constraints. We demonstrate the differences of my two methodologies and their unique advantages and drawbacks.**

Keywords—graph networks, image processing, image imputation, linear regression

I. Background

We decided not to consult any other papers and to focus instead on developing a completely novel approach. Our approach was very fluid in that, it was full of trial and error and reevaluation. The reason we did it this way, is because we truly believe that the best way to learn things is to first try to do them on your own, then through trial and error try to improve on your own work. The last step is to reference the research others have done. Through this process we learned a lot about image processing and the practicality of working with network graphs.

Image imputation is an incredibly important aspect of image processing as there are many scenarios where images are incompleate or parts are unknown or damaged. We have worked closely with people who routinely deal with incomplete and imperfect medical imaging data. They have expressed two primary reasons for difficulty in obtaining reliable imaging (1) imaging the insides of things is incredibly difficult as the camera must 'know' which layer to see through and which layer to see and (2) human insides are terribly unphotogenic. In this paper we focus on two image imputation methodologies. The first is a linear regression based methodology and the second is a graph network approach. We will also have two ways to evaluate the efficacy of our approaches, one that allows us to evaluate how our predictions fare when blacking out specific areas of images and another that evaluates how well the predictions understand the image and its features overall.

II. Understanding Image Data

In order to understand our processes for imputing pixel values it is important to know what pixels are and how they are constructed as well as the data behind them. An image is composed of pixels and each pixel has three associated values a red, a green, and a blue value (figure 1). These values are on a scale from 0-255 with 0 being none of that color, and 255 being the maximum amount of that color.

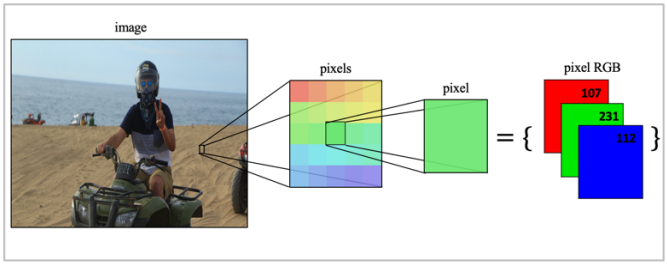

Figure 1 pixel representation

When considering an entire image it can be thought of as a (w * h * 3) matrix, where w is the width of the image in pixels, h is the height of the image in pixels, and 3 is for the three colors (figure 2).

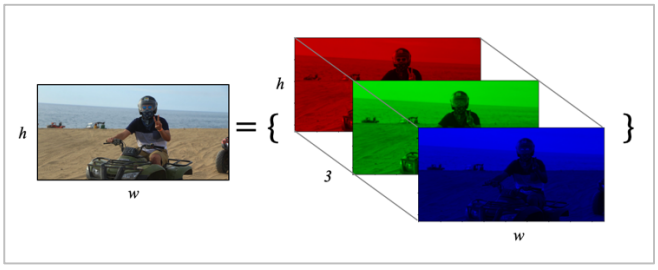

*Figure 2 (w * h * 3) matrix representation*

So now that we have a framework for how image data exists we can go about building methodologies for predicting missing pixel values in images. We start first on a pixel level and ask; what if we did not know what a pixel's RGB values are? How should we go about estimating these?

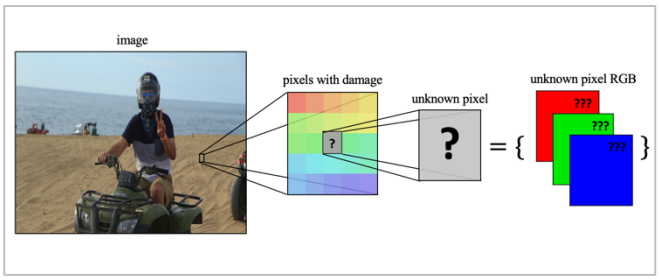

Figure 3 unknown pixel representation

we have developed two approaches to answer the question posed in figure 3 of what the RGB values for an unknown pixel. The first being a linear regression approach.

III. Linear Regression Approach

Our first approach was based on our data science background and our experience of working with tabluar data.

When working with tabluar data a common way to impute unknown values is to simply take the mean. Unfortunately for image data this would result in very conservative pixel RGB value estimates (muted colors). A better imputation methodology is by using other observations that are similar to the one we are trying to impute to predict its value. It stands to reason that a pixel should be similar to those around it. For this reason, linear regression comes to mind, and the simplest way to impliment this on tabular data is by using pixel values that are on the same row and same column as our missing pixel to estimate its value.

In figure 4 below, you can see the pixels with damage section from figure 3. I've highlighted the row and column that the damaged pixel is in and will use these cells to impute the damaged cell's RGB values. Looking at the horizontal cells and its corresponding graph at the bottom of figure 4, you can see that we have plotted the red, green, and blue pixel values of each cell in that row. Just looking at the pixels you can see that they are all very green and that the blueness increases as they move from the green gradient to the blue gradient (left to right). The scatter plot also shows consistently high green pixel values, low red pixel values, and low but increasing blue pixel values. We use these points to estimate the RGB values for our damaged pixel in the middle using linear regression. We then repeat the same process for the column the damaged pixel is in. Here you can see by the image that the pixels move from red to blue (from top to bottom). This is also reflected in the scatter plot of the RGB values, you can see that the red values sharply fall, the blue values sharply rise, and the green values bounce around. Again here we get predictions for each of the red, green, and blue values using a linear regression from the other points.

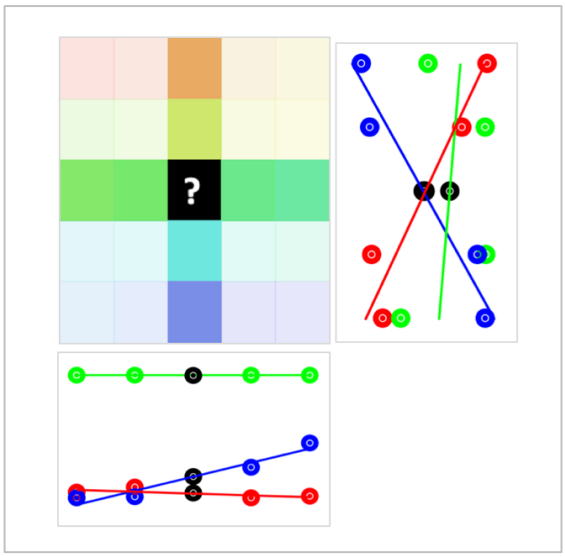

Figure 4 linear regression imputation approach

Once we have these two estimates (horizonal and vertical calculations) for the three pixel values (red, green, and blue). We take their mean to get our prediction for the damaged cell. The exact values and calcuations are given in the table below (table 1).

	Horizontal Estimates	Vertical Estimates	Final Estimate
R	116.75	167.75	**142.25**
G	232.00	193.50	**212.75**
B	129.00	165.25	**147.13**

Table 1 linear regression estimates

We can use the final estimates to impute the full RGB values of the damaged pixel. Figure 5 shows the linear regression approach's prediction against the original pixel color.

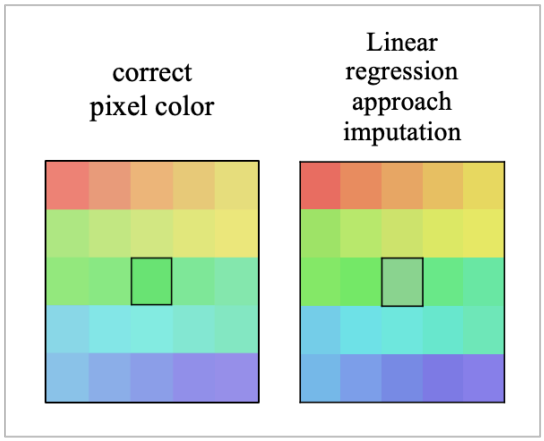

Figure 5 linear regression imputation vs actual pixel color

The linear regression approach appears to work quite well in this contrived toy example – but we were interested in looking into its performance when looking at a more real world problem.

IV. Test Case

In order to be able to evaluate this approach's performance we will use it on the following 100 by 100 pixel image that we use for all future apporaches to maintain consistency (100 by 100 pixel size was chosen because future approaches are more computationally expensive and we want to be able to directly compare results – it is also large enough of an image to contain the intricacies of large images). We chose the following image because it has a good balance of features (large features like the sky and beach; small features like my helmet and hands) (figure 6).

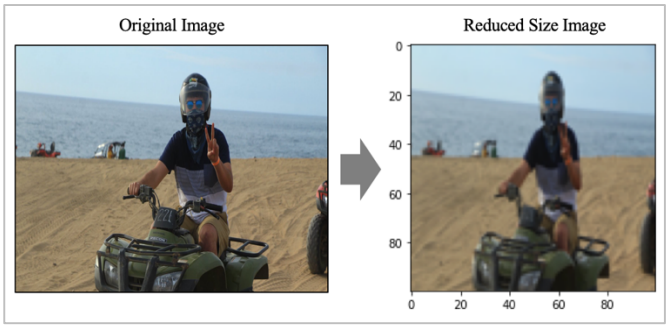

Figure 6 real world image

There are two specific evaluations we would like to conduct on this image, more specfically the reduced size image. (1) We would like to see how the approach imputes any specific area of the image. For example, in photo editing we could possibly want to edit out the red atv in the background from the image. (2) We would like to test how well the methedology understands and can recreate the image as a whole. Understanding how well any methodology understands the image as a whole can be used to help us select and/or improve an existing methodology.

Figure 7 shows the test cases for the two specific evaluations mentioned above. Using a methodology to impute the blacked out pixels given in problem one will see how well a methodology can replace certain specific areas of an image. Imputing the blacked out pixels in problem two (50% of the pixels are set to randomly blacked out) can give us an understanding of how well the approach understands the image as a whole and captures all features of the image.

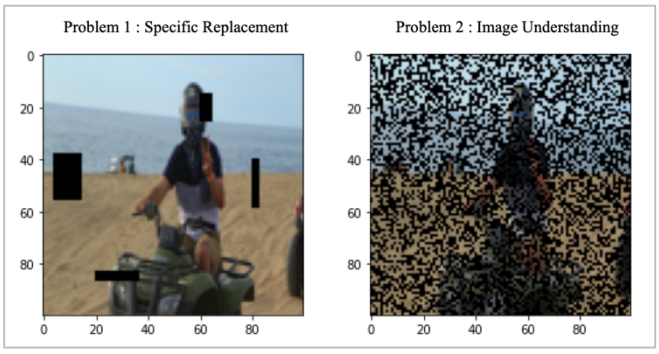

Figure 7 two test cases

Again these exact two test cases will be used for all future imputation approaches to maintain consitency and reproductability. This will also help us be able to practically compare how well different approaches perform.

V. Linear Regression Methedology's Perfromance on the Test Case

Below is the linear regression approach's predictions of problem 1 (figure 8). It is clear that the approach works better in some areas than others, but in general it seems to not be sensitive enough to its neighborhood.

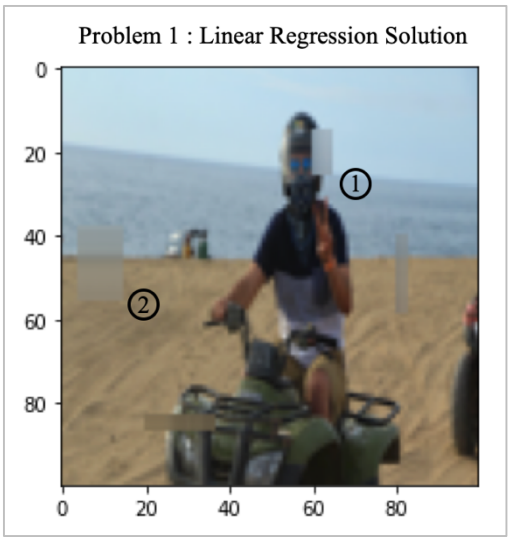

Figure 8 linear regression solution 1

In figure 9 you can see a close up of the two regions highlighted above. In region 1 it is clear that the linear regression methodology does not perform well at all. The approach does not consider pixels that are near enough. Also since half of the pixel's weight comes from the horizontal calculation – the region takes on a lot of blue color (from the sky). In region 2 this methodology performs much better – you can clearly see that it does manage to capture the large features of the beach and the sea. This is no doubt supported by the horizontal nature of the horizon

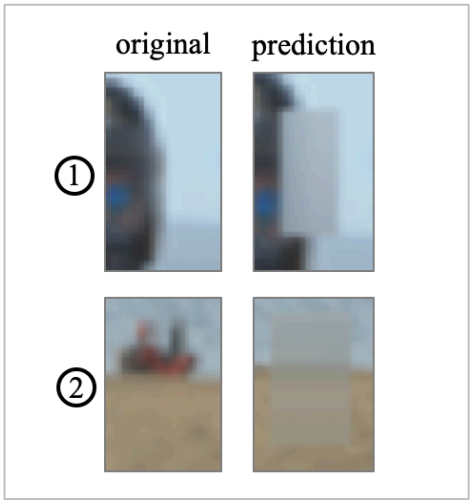

Figure 9 linear regression solution 1 exploration

It is very clear from figure 10 that the linear regression approach is not sensitive enough to its neighborhood (the pixels around it) and can only capture very large features. When looking at the second image in figure 10 you can see that the model has a lot of trouble with the middle part of the image where Rastko is on the ATV, and can only tell that there is some shadow of an object there.

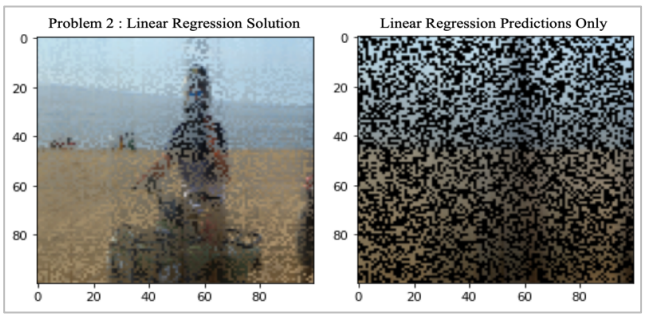

Figure 10 linear regression solution 2

VI. Graph Network Approach

The problems with the linear regression approach led to our development of a graph networks approach. This approach was designed directly to combat the issues that the linear regression approach suffered from. These issues are (1) non-sensitivity to the neighborhood, (2) non-flexible approach, and (3) only consideration of pixels directly left, right, up, and down from the pixel. The graph network framework we developed helps us solve these problems by framing it in a different way. Instead of imagining the pixels as rows and columns, a network graph structure is more appropriate where the pixels are all tied together but more smiliar to those near them, than those further away (figure 11).

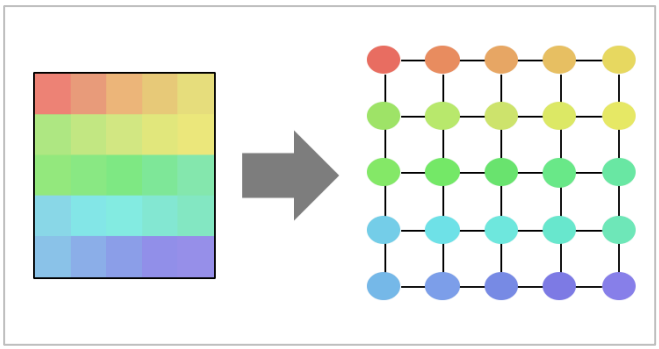

Figure 11 graph network view

If we try to use this mesh graph framework, we need to consider two factors when trying to predict any pixels RGB values from its neighbors (1) how many layers of neighbors to consider and (2) how much more should we consider neighbors that are closer than those that are further away.

We constructed two parameters (*n* and *d*) that relate to the two factors mentioned above. The neighborhood parameter *n* can be imagined as the number of hops away from the node of interest to be considered (figure 12).

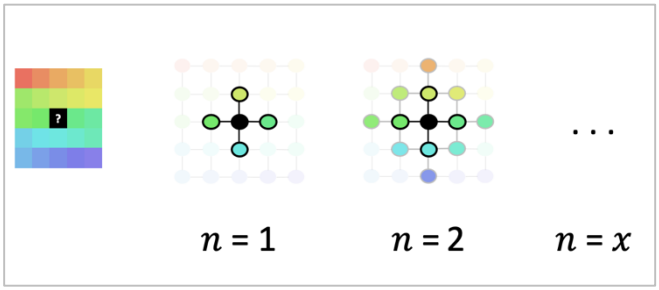

Figure 12 graph networks: n parameter

This flexible *n* parameter is the depth of neighboring nodes to be considered for the imputation – this parameter's flexibility allows the model to consider more pixel neighbors in higher pixel images.

The second parameter *d* helps specify the drop off of weight at each level *n*. The formula for getting a node *i*'s weight is given below and examples are given in figure 13 showing different values of drop off's influnces on individual drop off weights (θ) where n_i is the neighborhood of the node i.e. the number of hops from the node of interest).

$$\theta_i = d^{(n_i - 1)}$$

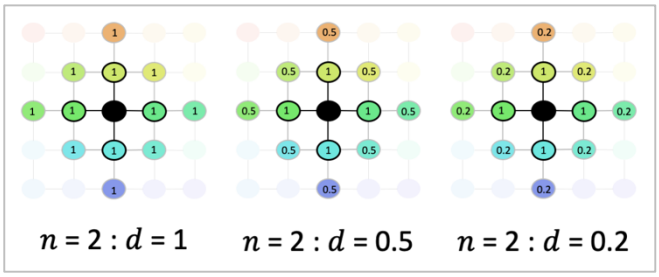

Figure 13 graph networks: d parameter

Once we have the drop off weights θ_i for each *i* we can go about calculating the final weights for each node. The weights have to sum to 1 for them to be acceptable for a graph network adjacency matrix. So the final step to get the final weights w_i is to normalize the θ_i values so that they sum to 1.

$$w_i = \frac{\theta_i}{\sum_{j \in i} \theta_j}$$

The models generally should have a $d < 0.5$. This is true because at each additional *n* the number of nodes added in the next layer is two times the nodes in the previous layer. So if the weight drop off is 0.5 these two layers will have the same weights in totallity and generally layers of nodes closer to our node of interest should weigh more.

Once you have the weight of each individual pixel in the graph in terms of how much they should affect the node of interest, you simply multiply their weights by their nodes RGB values respectivly and sum them together to get the predicted RGB values.

$$\begin{bmatrix} R_{pred} \\ G_{pred} \\ B_{pred} \end{bmatrix} = \sum_i w_i \begin{bmatrix} R_i \\ G_i \\ B_i \end{bmatrix}$$

Now that we understand the process of building our graph network and the parameters it uses for imputation, lets see how it performs on the same test case questions that we looked at for the linear regression approach.

VII. Graph Network Methedology's Perfromance on the Test Case

First off, lets see what the test image looks like when it is cast into this graph network. Below are some examples with varying parameters (*n* and *d*) and varying graphing styles from

the NetworkX package in python (figure 14). These images aren't super informative on their own, just interesting.

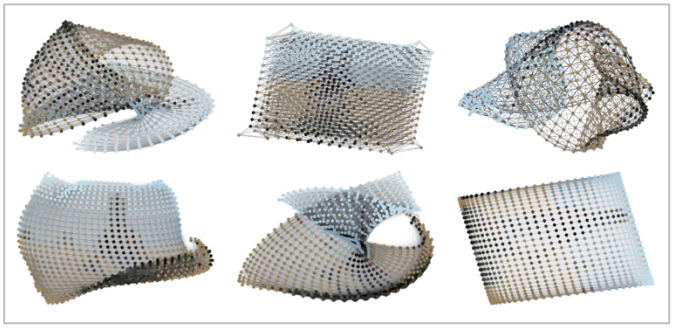

Figure 14 test image as graph network

The underlying network graphs used to generate these images above is the exact same that is used to generate our predictions.

Figure 15 shows how this graph network approach with the parameters $n = 7$ and $d = 0.2$ outperforms the previous methodology.

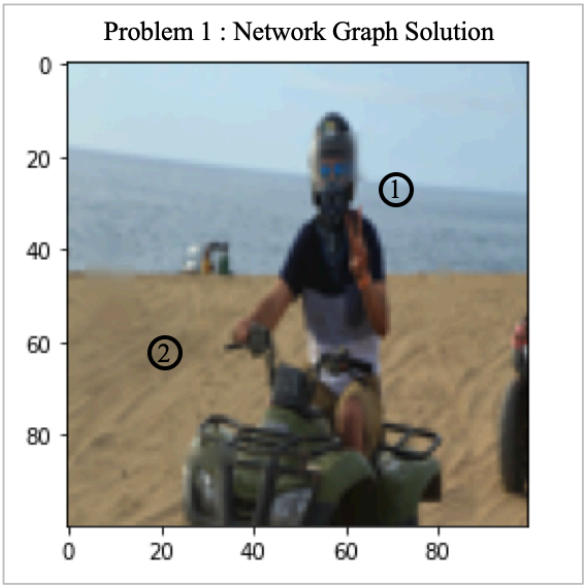

Figure 15 network graph solution 1

Overall the approach preforms much better than the linear regression approach. It is even pretty hard to notice the areas that were fixed! Looking at figure 16 we can see that the network graph approach did a really good job of capturing the local elements of the helmet and also preformed better than the linear regression approach in region 2 where the linear regression approach had its best performance.

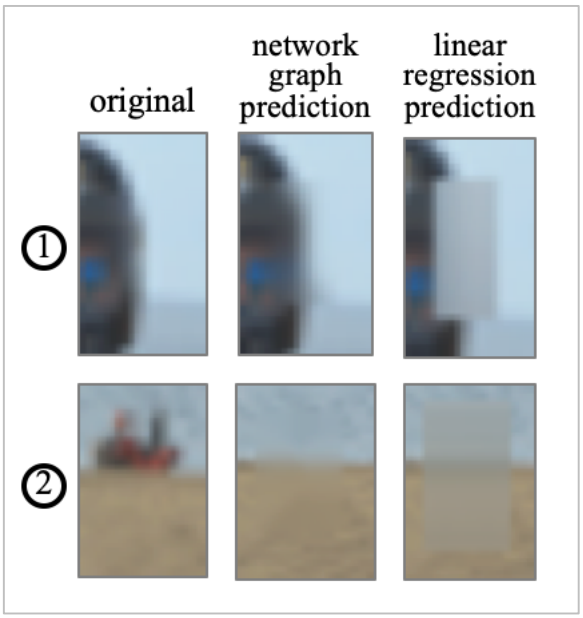

Figure 16 solution 1 comparison

Not only is the graph networks approach fantastic at imputing specified areas of an image, it also has developed a pretty good understanding of the image as a whole. In figure 17, the approach does a great job of filling in half of the image. If you consider the second image in figure 17, you can see that it captures all features of the original image. It does a good job understanding the large features like the beach and the sky, like the linear regression approach (but better), and it also can impute the small features like the sunglasses and hands.

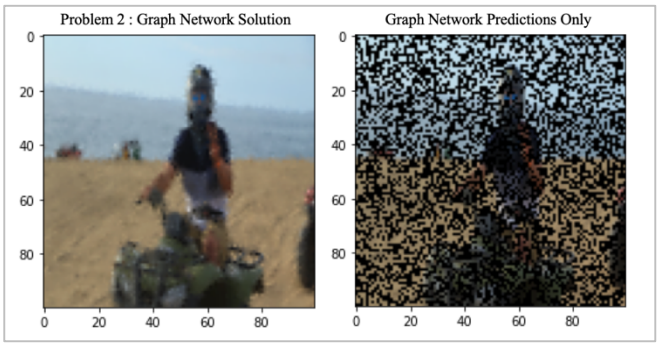

Figure 17 network graph solution 2

All in all the graph network approach is much better at both evaluations that we specified in the beginning of the paper.

VIII. Graph Network Drawbacks

Computation time is the one big drawback of the graph network approach. In fact, only the linear regression approach works on the full, non-reduce image. The graph network approach relies on building an adjacency matrix of all the pixels, and with the orginal image being a 1713 x 859 matrix image, this expands the adjacency matrix to be ~1.5 million x ~1.5 million. This adejcency matrix increases exponencially as we use higher and higher resolution images, so the graph network approach may not be possible for extreamly high resolution images, where as the linear regression approach scales really well, and will always be possible.

Another potencial problem of the graph networks approach that doesn't exist for the linear regression approach is that it is possible to black-out a region that is too large for the given *n* patch to impute completely. For example, in figure 18 you can see the orginal missing values are highlighted by the black outline, and after imputation with a *n* paramater of 7, only some of the orginal blacked out area is filled in.

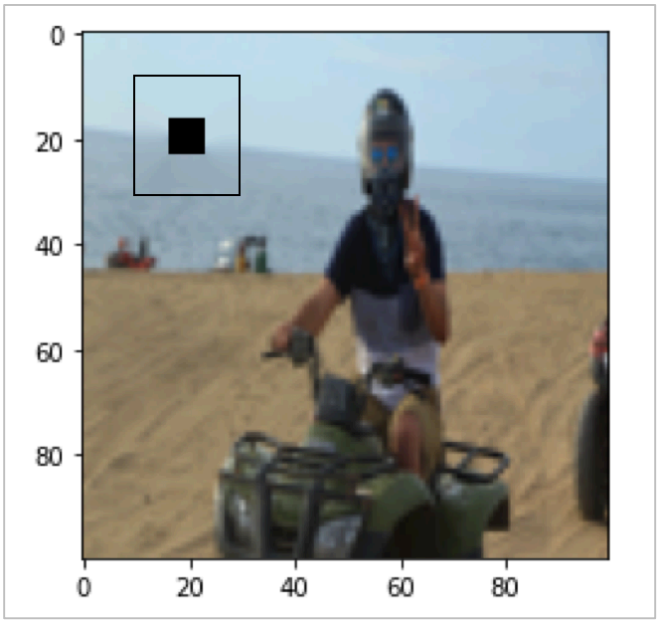

Figure 18 image imputation issue

There is two possible ways to resolve the issue in figure 18. The first and more recommended solution is to alter the parameteres. You could raise the *n* parameter so that the pixels in the middle can reach pixels with actual values. When doing this I would recommend simultaneously lowering the *d* value to limit the effect of this on the rest of the prediction (if you are satisfied with the performance of the areas it did guess). The second solution which I do not recommend is that you could rerun the the same function, this time with the image result of the first time as the input to the second function. The problems with this solution is that you are stacking imputations on imputations, that is to say you are using estimated values to estimate more values. This will end up in a snowballing error.

Another thing to note is that you generally don't want to let the *n* to be too low. The reason for this becomes clear if you examine the regions near the pixels that were not imputed in figure 18. It is not very easy to see but if you try you can sort of see it in figure 19. Right near the black box in the middle (tiny red circle in the middle) these pixel predictions consider only the values of pixels in a square around them (for the red circle – it would be the red square). This forces the prediction to be based on just a few pixels! You can see how this depresses the sea on the right side of the black middle square, and raises the sea level at the red circle, because the decision is being made on exclusivly the one nearest neighbor available which is at the edge of the black outline.

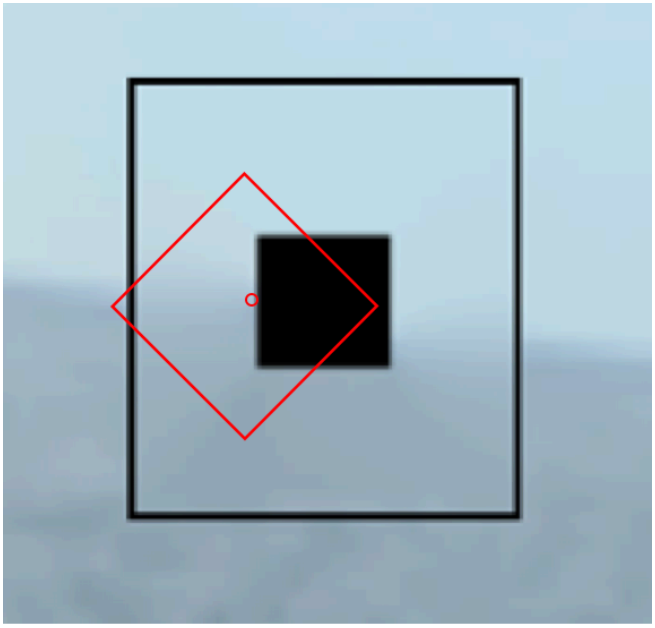

Figure 19 low n issue

Therefore, it is recommended to use a higher *n* and reduce the *d* parameter. In this way you could impute all the values in the first try and use more than just a few pixels for all the predictions, even those near the middle of the prediction area.

IX. "Enhance That Image"
– Every Movie Ever

Not directly a central part of our project but interesting nonetheless. Our graph network approach also shows why enhancing images as they love to do in movies is not really possible (you cant create something out of nothing). Our graph approach lets us test this out. If we take a 50 x 50 pixel image and cast it into a 100 x 100 pixel space (figure 20), we can use our graph network approach to try to fill in the unknown areas.

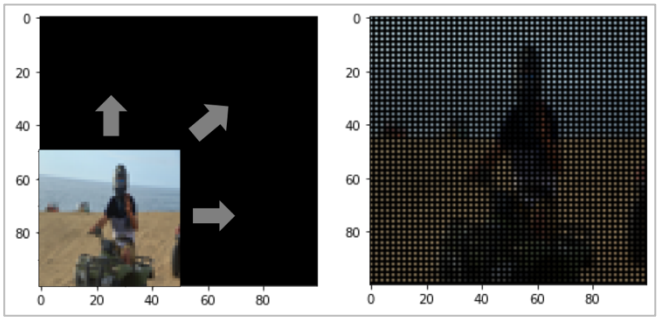

Figure 20 small pixel to large pixel projection

Figure 21 shows that the graph networks approach does give an estimate of the blacked out areas, but it isnt really adding anything to the image. Really it just ends up smooting out some of the regions between the 50 x 50 pixel image.

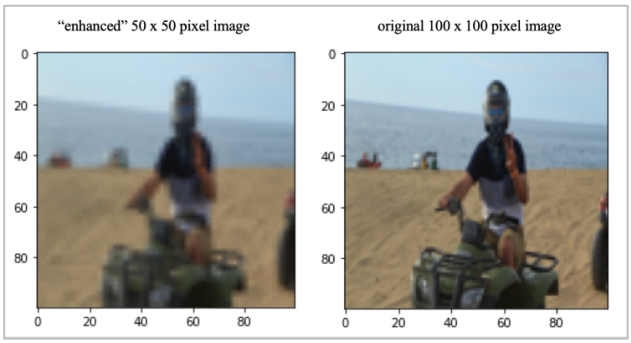

Figure 21 "enhanced" image comparison

X. Conclusion & Next Steps

We are overall happy with the graph networks approach in its ability to successfully predict missing pixel values. In both evaluations of predicting specific missing areas of an image and in understanding all image features as whole, the graph network approach performed significantly better than our baseline linear regression approach. We are further encouraged by the fact that we didn't reference any previous methodologies for image imputation and instead relied on our own processes of trial and error to develop a novel graph network-based methodology. One of our next steps will now be research how others attempted to solve this problem. We look forward to learning about how other people solved the problems we couldn't, mainly in being able to scale their methodology to a higher pixel image. Once we have a more efficient running pipeline we think immediate uses could be in photo restoration and editing. Our methodology is much further away from contributing to any feature recognition/computer vision tasks, but nonetheless it is still an interesting application to think about.

Another future step we can do is to continue to experiment with the combination of parameters for our graph network approach. We want to further understand the relationship between the parameters and the features of an image (size, complexity, etc.). Currently we are limited by computational time, but any continued experimentation could lead us to creating better tuning methodologies the depend upon the varying features of images.