Zoom^{TM} and Enhance: Super Resolution

Dylan L. Beck
University of Michigan
dlbeck@umich.edu

Ian A. Bertram
University of Michigan
ianbtr@umich.edu

Daniel L. Hoekwater
University of Michigan
dhoek@umich.edu

Katherine J. Knister
University of Michigan
kknister@umich.edu

Shriya S. Shah
University of Michigan
shriyash@umich.edu

1. Introduction

High-resolution images are aesthetically pleasing and have a natural look and feel to them. Too often, though, we are stuck with grainy low-resolution images that are inadequate for use in websites and presentations. This can be catastrophic if the only surviving photo of your subject has poor resolution.

Given the increasing importance of video communication, we also desire better video compression. A process that allows users to send downsampled video, then upsample compressed video that is received from the network, would save large amounts of network transactions. An advanced understanding of super resolution in still images might help this effort.

While it is possible to improve the resolution of images using interpolation techniques such as bicubic and nearest-neighbor, these methods do not improve the resolution as much as one may hope. This motivates the use of a convolutional neural network to improve the resolution of images.

Previous work by Wang *et al*. [3] demonstrates that a variety of neural network designs can solve the problem of implementing super resolution. Work by Dong *et al*. [1] reports that a fairly simple convolutional neural network can perform super resolution and outperform bicubic interpolation. This prior work shows us that we are not only able to solve this problem, but also solve it using computer vision.

We experimented with a neural network similar to the one presented by Dong *et al*. in [1] in order to demonstrate that it could achieve better performance than bicubic interpolation. Our final network, which we trained using a subset of the flickr2k dataset provided by [2], involved pre-upsampling, three convolution layers, and two ReLU nonlinearity layers. We ran trials with many different combinations of values for hyperparameters and slight changes in the architecture to decide that this network allowed us to achieve the best results, which indeed were better than the results achieved by bicubic interpolation.

2. Approach

2.1. Setup

Our first challenge was setting up our training pipeline such that we could effectively collaborate synchronously and asynchronously, regardless of group size. To overcome this challenge, we opted to upload our dataset into a shared Google Drive, which we mounted on Google Colab via a Jupyter Notebook in a group GitHub repository.

We based our code structure on the convolutional network used in Homework 5, and we broke the pipeline into six phases: software dependencies, dataset definition, data loader, neural network architecture, training logic, and final quantitative and qualitative testing. This structure made it possible to split into smaller teams to work on different components in parallel, speeding up the development process and reducing the coordination headache.

2.2. Architecture

As seen in Figure 1, which excludes ReLU layers for readability, our network architecture is relatively simple, using only a few convolution layers. This is in line with other super resolution architectures [1], as many practical applications such as video enhancement would require quick computation.

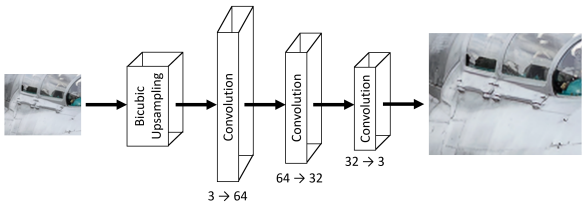

Figure 1. Our Network Architecture

Our network uses pre-upsampling: it begins by scaling the input low resolution image by a factor of 2 using bicu-

bic interpolation. Since our model's goal is to outperform bicubic interpolation, doing this first sets a baseline upon which the rest of our network architecture can improve.

After upsampling, our network has three convolution layers, with each of the first two convolution layers followed by a ReLU non-linearity layer. All three convolution layers use a 5x5 window with padding of 2 and a stride of 1. These values preserve the size of the images through each layer. The dimensions of the hidden layers for the convolutions are 3 to 64, 64 to 32, and 32 to 3. We made decisions about our network architecture through experimentation, which we will explain further in section 3.4.

2.3. Loss Functions

We used the mean squared error as our loss function because this is the most popular loss function for super resolution [3]. In particular, this is the loss function that Dong *et al*. used, and our neural network had a similar structure to theirs [1]. We can use the following calculation to use the mean squared error as our loss function:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} ||F(Y_i;\theta) - X_i||^2$$

where θ is the estimation of network parameters (weights and biases), n is the number of training samples, $\{Y_i\}$ is the set of low resolution images, $F(Y_i;\theta)$ is the reconstructed image from Y_i, and $\{X_i\}$ is the set of high resolution images.

2.4. Training Procedure

The function we used to train our network relied on comparing low resolution images to corresponding high resolution images. It was based on the training function from the Semantic Segmentation section of Homework 5. However, instead of the data consisting of images and labels, our data consisted of low-resolution and high-resolution images. We randomly selected 800 image pairs from the flickr2k dataset provided by [2] to use to train our convolutional neural network (we used 100 for validation and 100 for testing). Our training procedure kept track of the running loss, and it calculated the loss using the mean squared error, which is described in more detail above in section 2.3.

In addition to training with entire images, we experimented with training with just patches of images. We hoped that this would allow us to train with more image samples without going over the maximum GPU usage. Additionally, this allowed us to choose a consistent image size, which allowed us to easily increase the batch size to more than 1.

However, we achieved worse results with image patches compared to with entire images despite being able to train on more samples when using image patches. Because of this, the results we are reporting were generated by training with entire images, even though we tried training with image patches for a number of trials.

3. Experiments

3.1. Data

We used 1000 images that were randomly selected from the flickr2k dataset from [2]. We used 800 of these images for training, 100 for validation, and 100 for testing.

This data made sense to use because flickr2k is a commonly used dataset for image super-resolution, so we believe that using a random subset of it allowed us to have a reasonably representative sample of user images. Additionally, since it's commonly used for image super resolution, it already had low- and high-resolution image pairs that we could use for training, validation, and testing, instead of needing to generate our own image pairs.

3.2. Metrics

We measured success by comparing the peak signal-to-noise ratio (PSNR) that we achieved on the test set to the PSNR that could be achieved by bicubic interpolation. Since bicubic interpolation can generally achieve an average of around 30 PSNR, we knew that we wanted to achieve higher than 30. We also found the PSNR result that bicubic interpolation achieved on our test set specifically and made sure to achieve a higher one.

In addition to these quantitative measures of success, we also asked people to make qualitative comparisons between our results and the low resolution images, as well as between our results and the results of bicubic interpolation. To ensure our personal biases did not affect our assessment of the model's quality, we conducted an experiment.

We randomly selected five images patches from our test set. For each of these image patches, we took the original low resolution version, the version that resulted from bicubic interpolation, and the version that resulted from our super resolution method. Then, for each set of the three versions, we randomly shuffled the order of the three versions.

We showed these five image patch sets to six testers, asking them to rank the images in each set based on the resolution. We considered it a success if, for each set of images, the participants generally believed that the image version resulting from our super resolution method had a higher resolution than both the original low resolution image and the bicubic result.

PSNR is a reasonable measure of success because it is an industry standard for measuring similarity; it is currently the most widely used evaluation criteria for super-resolution methods [3]. However, a downside of PSNR is that it can result in poor performance when used to represent the reconstruction quality in real scenes. In this case, we are usually

more concerned with human perceptions [3].

With this in mind, qualitatively comparing the images is reasonable because quantitative measurements generally do not fully capture how humans perceive the level of resolution of an image. Comparing our results to bicubic interpolation makes sense because bicubic interpolation is the standard non-machine learning approach to performing super resolution.

3.3. Tuning Hyperparameters

In order to achieve better results, we experimented with the tuning of different hyperparameters (i.e. the learning rate, weight decay, number of epochs, and the size of the dataset). This, along with adjusting the network architecture, accounted for a significant portion of the time that our team spent on this project.

For the learning rate, we found that very small learning rates, on the order of 1e-4 to 1e-5, worked the best. When we used larger learning rates, our loss increased dramatically, meaning our PSNR decreased. Using smaller learning rates made the training take significantly longer without much benefit, since the loss didn't decrease by a significant amount. Our final model used a learning rate of 5e-4.

We tested multiple weight decays, and we didn't see a significant difference in the trials with the different decays. This allowed us to use a weight decay of 0 for our final model in order to try to improve our loss plot, which is discussed further in section 3.6.

Since training the data takes a long time, we decided to use 10 epochs. This meant our model took between 2 and 5 hours (depending on other hyperparameters) which allowed us to achieve desirable results while still being able to experiment with about 20 different models

We originally tried to use a dataset of 500 images since training the model takes a long time. However, we were getting unexpected results with a dataset of that size. When we increased the dataset to 1000 images, we got better results.

We also tried increasing the dataset size to 2000 and even 2650 (the size of the flickr2k dataset), but achieving results with these larger datasets was much harder because we frequently ran into issues when trying to train using Google Colab for such a long time. In particular, when we tried to use a dataset of 2650, we ran out of available GPU usage before even finishing 10 epochs, despite using an entirely new Google account. Training the model on 1000 images ended up being the most reasonable solution since we could achieve good results consistently with this size.

3.4. Finding the Optimal Architecture

As mentioned previously, our network architecture was based on the architecture used by Dong *et al*. [1], which used pre-upsampling with three convolution layers and two ReLU layers that followed the first two convolution layers. While we ultimately used this architecture to produce our best model, our team experimented with numerous architecture models.

One aspect of the architecture that we experimented with was the upsampling layer. We tried putting the upsampling layer at various locations in our network, including between convolution layers and at the end. None of these alternate locations produced better results than pre-upsampling, which led us to choose pre-upsamping. It also seemed intuitive to start with bicubic interpolation in order to beat bicubic interpolation.

Another aspect of the network we experimented with was the number of convolution layers. We tried using four and five convolution layers. However, adding convolution layers did not allow us to achieve a better PSNR regardless of other changes we made alongside this, so we decided to keep our architecture simple with only three convolutions.

We also tried various different sizes for the hidden dimensions of the various layers. We decided on our final sizes for the hidden dimensions using this experimentation.

3.5. Qualitative Results

One of the primary methods we used to evaluate our model was a qualitative comparison between the resolution of the images that resulted from our super resolution method, the original low resolution images, and the results from bicubic interpolation. Since our goal is to upscale images for humans to use and appreciate, we believe this qualitative evaluation is important.

Figure 2 below shows patches of three images: an original low resolution image that we used to test our model and the results of upscaling it with bicubic interpolation and upscaling it with our model. Looking closely at the images (particularly at edges such as the hands), one can see that the image resulting from upscaling by our model is sharper than the original low resolution image as well as the image upscaled through bicubic interpolation.

Figure 2. Comparison of image patches from original low-res image, results from bicubic interpolation, and results from our model

As described in section 3.2, we randomly selected five images patches from our test set and asked six testers to rank the original version, bicubic upscaled version, and our model's upscaled version (which were presented in random order) in terms of the quality of resolution. We found that

all six testers identified the image upscaled by our model to have the highest resolution for all five sets of images. This gave us confidence that our model not only produces results that clearly have a higher resolution than the original low resolution image, but also that our model outperforms bicubic interpolation.

We've included the complete results in Table 1 below. Each cell represents the order in which a tester (PID) ranked the perceived resolution quality of the images, in ascending order. L represents the original low resolution image, B represents the image upscaled with bicubic interpolation, and U represents the image upscaled by our model. For example, the first cell indicates that PID 1 ranked the quality of the low resolution image as the lowest, bicubic in the middle, and our model's upscaled image as the best. We want to emphasize that participants were unaware of the identity of each image while assessing their quality.

PID	Img1	Img2	Img3	Img4	Img5
1	L, B, U	L, B, U	**B, L, U**	L, B, U	L, B, U
2	L, B, U	L, B, U	L, B, U	L, B, U	L, B, U
3	L, B, U	L, B, U	L, B, U	**B, L, U**	L, B, U
4	L, B, U	L, B, U	L, B, U	L, B, U	L, B, U
5	L, B, U	L, B, U	L, B, U	L, B, U	**B, L, U**
6	L, B, U	L, B, U	L, B, U	L, B, U	L, B, U

Table 1. Summary of qualitative user testing results

3.6. Quantitative Results

The main metric we used to determine the success of our model was the peak signal-to-noise ratio, or PSNR. Our goal was to achieve a PSNR of at least 30, and we also wanted to achieve a higher PSNR with our model than that which could be achieved using only bicubic interpolation. We were able to achieve both of these goals, since the bicubic baseline achieved a PSNR of about 32.14, and our network achieved a PSNR of about 32.91.

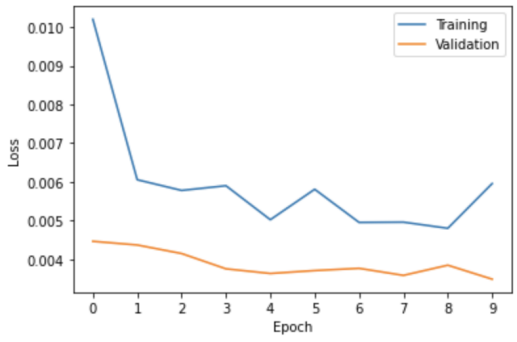

Figure 3. Training and validation loss plot for our model

As seen in Figure 3 above, the validation loss is lower than the training loss at all times. In order to mitigate this strange behavior, we tried using patches of the images instead of the full images, so that we could use batching. We also changed our loss function to compute a moving weighted average loss. By doing this, we were able to achieve a much better loss pattern, as shown in Figure 4.

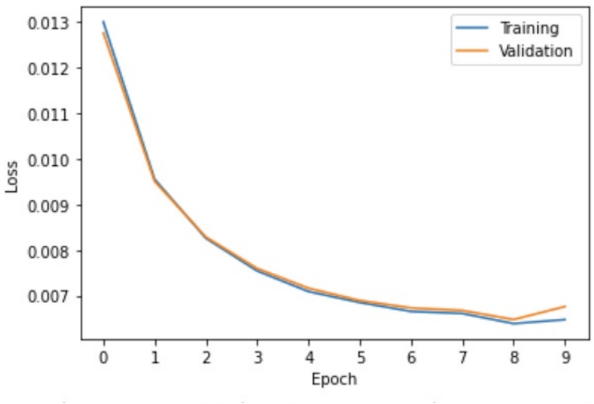

Figure 4. Training and validation loss plot for our modified model

However, the PSNR for the trial with that loss plot was much lower - the bicubic baseline had a PSNR of about 28.6 and our network had a PSNR value of about 28.7. We believe that this was due to the fact that the image patches were too small. However, due to the constraints of Google Colab, we weren't able to fully test our theory. If we had more computational resources, we would like to trying using a much larger dataset with larger image patches.

4. Implementation

The implementation of our project certainly built on the work of others. Our decision to use pre-upsampling and an architecture with three convolution layers came from [1]. Additionally, the specific code we used to train, test, and visualize the network was largely based on the starter code provided by the EECS 442 staff for Homework 5.

Despite building on the work of others, many aspects were unique to our project. On the network side, variables including the hidden layer dimensions, kernel size, and stride were chosen by us through a combination of experience from prior homework, research, and experimentation. Additionally, hyperparameters including learning rate, weight decay, number of epochs, and batch size were chosen by us through similar means. In addition, we experimented with using image patches and adding additional convolution layers.

Finally, the "glue" code (which proved to be a major challenge) was written entirely by our team. This includes code to load the dataset from Google Drive to run in Google Colab, normalize and crop the images to allow for variable batch sizes, and visualize our resulting images.

References

[1] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[2] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[3] Zhihao Wang, Jian Chen, and Steven C.H. Hoi. Deep learning for image super-resolution: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.