Image Prediction using Deep Learning

Jinjing Xie , Yuxuan Qi and
Qiyang Gai
Tandon School of Engineering
NYU
New York, USA

Abstract

In video compression, it is useful to be able to predict other samples from a fraction of your original samples to stay within bandwidth limits. One of the most successful methods is the multi-mode prediction approach is the H.265/HEVC mode which has needs for intra-prediction (spatial based) and inter-prediction (time based) blocks. For intra-prediction, the picture is divided into variable size blocks of 4x4,8x8,16x16. An encoder uses manually crafted filters on each block to see what mode gives the best estimation of the internal pixel. The pattern with the best result is recorded, and the data is compressed using this pattern. There have also been major leaps in the field in deep learning. We are interested to see if these leaps can be applied to create a better compression function for intra-prediction of image blocks

In order to accomplish this, we are going to first use a CNN to encode a 16x16x3 block into a much smaller block of size 2x2x8 This 2x2x8 block we hope will encode the key features. Then much like how the H.265/HEVC uses its feature mode and the border to recreate the final image. We will use the decoded version of this block + the border to create the final image. Each network will be trained separately at first to compute the optimal substructure of their tasks before combining them into a greater whole.

I. Project Overview

A. Data Overview

We took our data from open images dataset just like other research papers on intramode prediction. In order to generate the blocks, we take 10 20x20x3 subsamples from each image. We choose a total number of 10,000 images from the dataset and finally get 100,000 subsamples. Then we normalized the data to -0.5 to 0.5 by taking (pixel-128)/255.

We choose 70% of the total subsamples as training part, 20% for validation part and 10% for the test part.

B. Network Structure Overview

In order to train the network, we will divide our project into 2 parts: a encode and decoder combination and a 5-layer convolutional network with skip connections to incorporate the border information into the original block. These parts will both use the same dataset just different image block sizes. I.e. the encoder and decoder will use a 16x16x3 block as the input whereas the five-layer convolutional network will use a 20x20x3 block. You can refer to figure 2 to see how we combined the two. These functions will both be trained using stochastic gradient descent with a learning rate of 0.01 starting and an adaptive learning rate change of 0.01*0.95^(epoch number). Our batch size will be 50 images, and the network will go through 2 epochs. These parameters were all decided experimentally by Jeffrey Mao. Experimentally, there is not much benefit from adding more epochs to the network as seen in figures 4 and 5.

C. Encoder Decoder

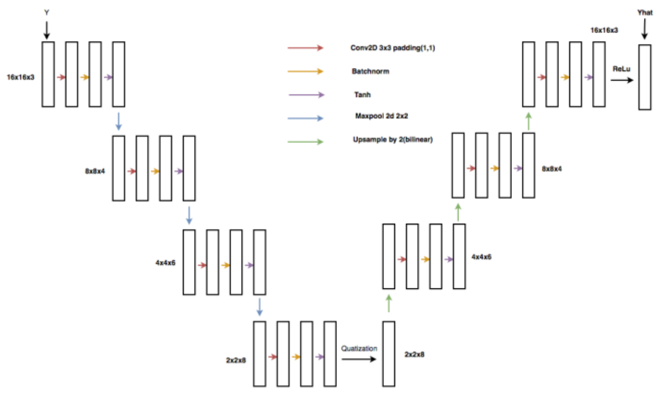


Figure 1: Encoder Decode Net

First the center part was extracted to form a subsample of 16x16x3 pixels from each of the 70000 samples in the training set. Then, our CNN network encoded each block in a 2x2x8 form and decoded it to form a different image. In each down step, the convolution was done with the following steps interpolation down 0.75 convolution2d with a 3x3 filter repeat. We then quantized the 2x2x8 block using a tanh. Similarly,we use the same way to decode it..

At the end, of the function, we will compare our output to our input and minimize the loss function.

D. Image Augmenter

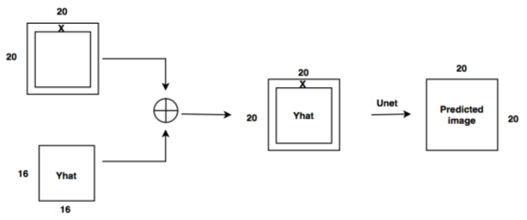


Figure 2 Adding a Border

We concatenated a border 2 pixels wide with what we get from Part 1 to create a 20x20x3 pixels.

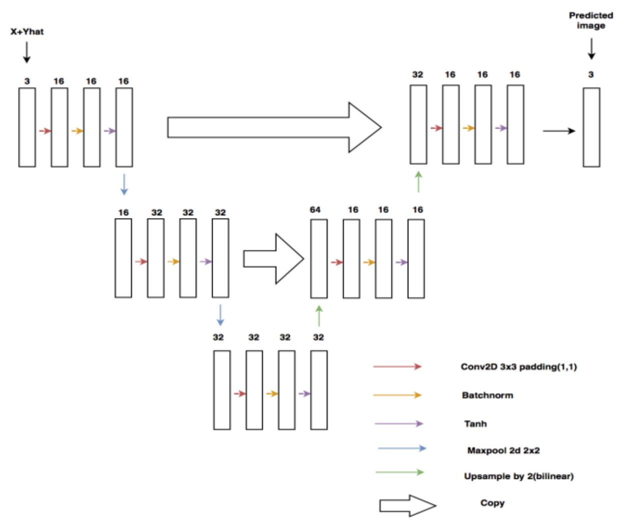


Figure 3: Final Smoothing Net

Then we send the new data to a 5-layer convolutional network with skip connections to denoise it. After adding original border to the data from Part I, we hope to get a lower loss from Part II compare to Part I.

II. PROJECT ACCOMPLISHMENTS

A. *Encoder Decoder*

The goal of this section was to compress a 16x16x3 block into a 2x2x8. This compressed block will be decompressed and combined with the border to recreate the original image. To achieve this, we attempted to match the 16x16x3 color block as well as possible by 2x2x8. In reality, it isn't necessary for the 16x16x3 block to be exactly the same as the original as long as it can be reconstructed with the border, but for the purpose of validating the structure of this part. We trained it to match the input with the output based on the mean squared error. This code was written in Python and debugged in the Jupyter Notebook environment

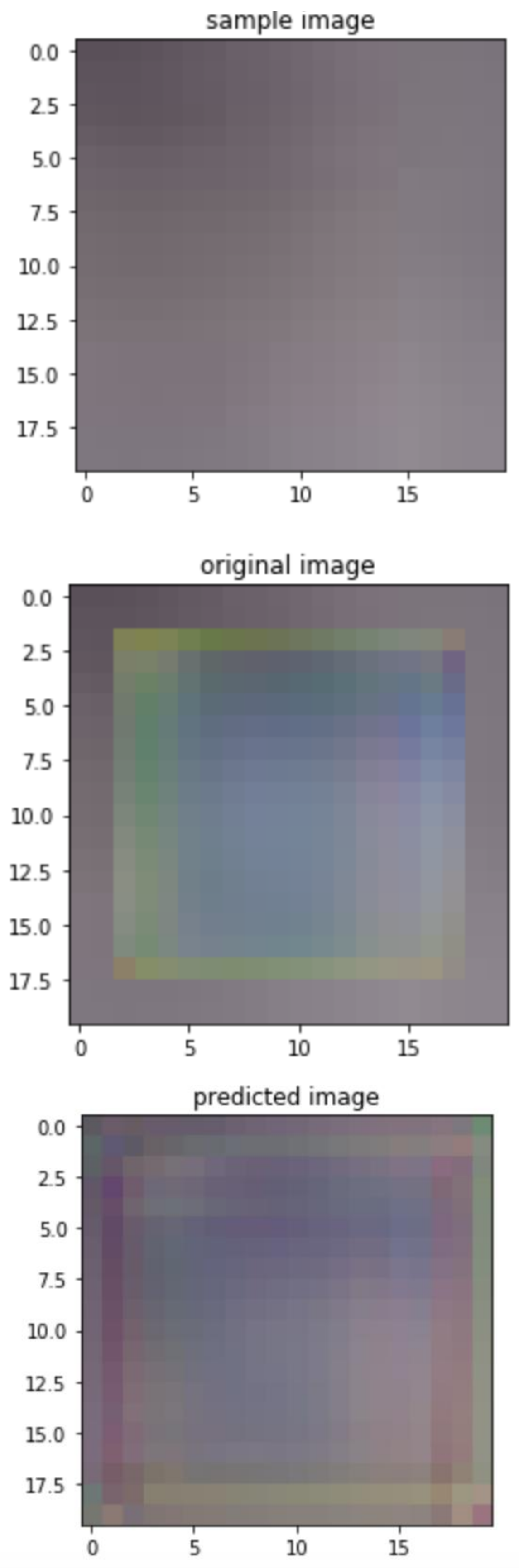
1) *Encoder Decoder Completed Works*

We created an encoder/decoder as per the figure 1 structure. Overall, the most important part was the quantization between the encode and decode which was achieved through tanh. In the end, our training loss for this system was 0.0135 and our validation loss was also 0.0128

2) *Encoder Decoder Lessons Learned*

It seems that the Encoder and decoder have a lot of trouble representing complex image patterns. Which makes sense, using less bits to represent an image will result in data lost. Unfortunately, when using a pure mean squared loss function, we can see all major patterns being lost. We believe this is cause our lose function Mean Squared error is merely trying to match colors to colors rather than edges or other patterns. So we comes up with two different approach. That is change the Image Augmenter.

Figure 4: Result using first Image Augmenter



The second approach is instead of using Image Augmenter in figure two, we use other two different image augmenter shown in figure5 and figure 7.

Figure 5

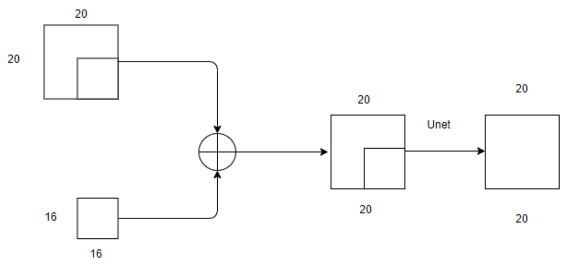
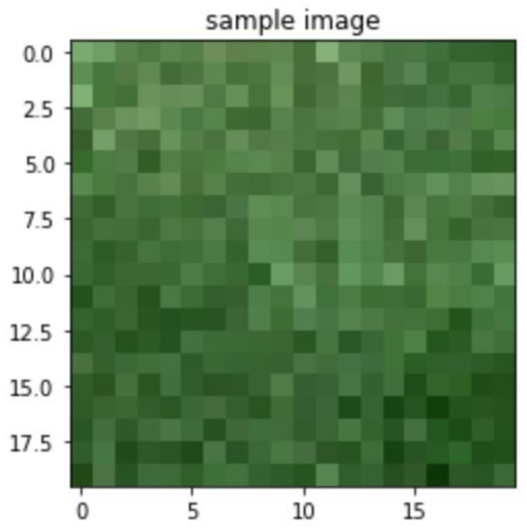
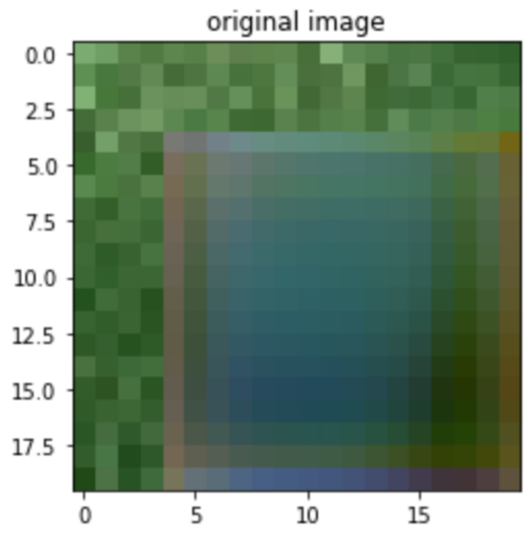


Figure 6 Results of using image augmenter in Fig.5





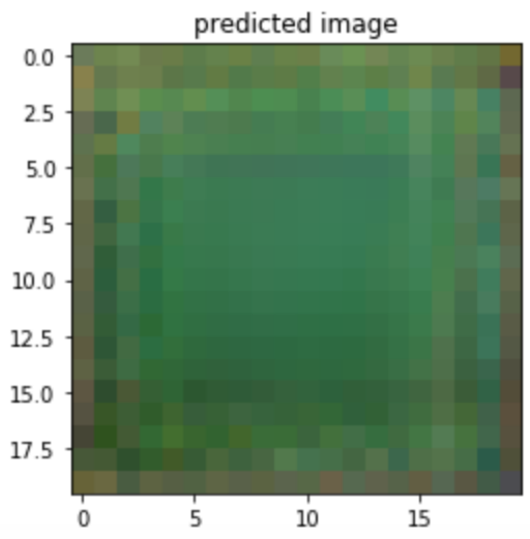


Figure 7

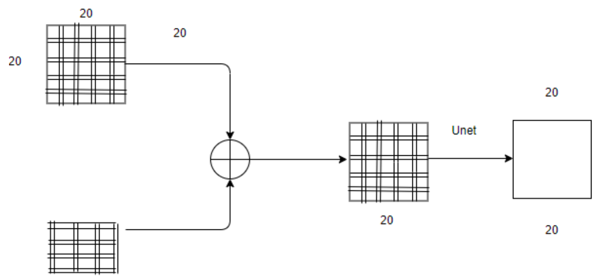
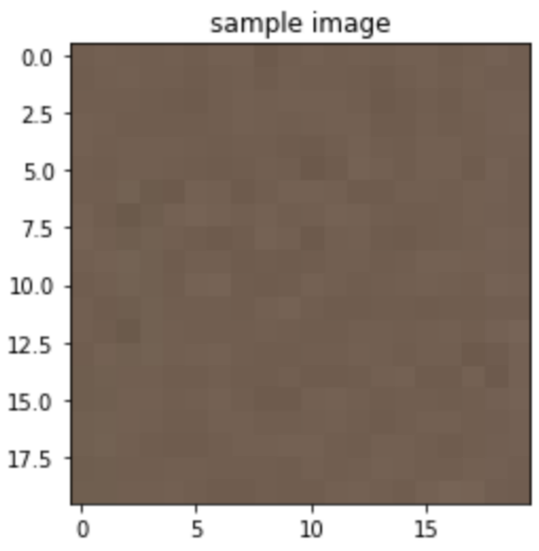
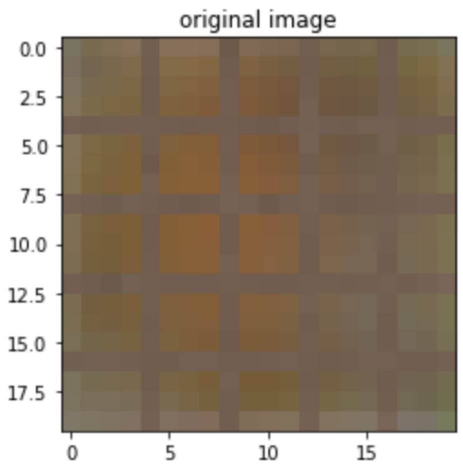
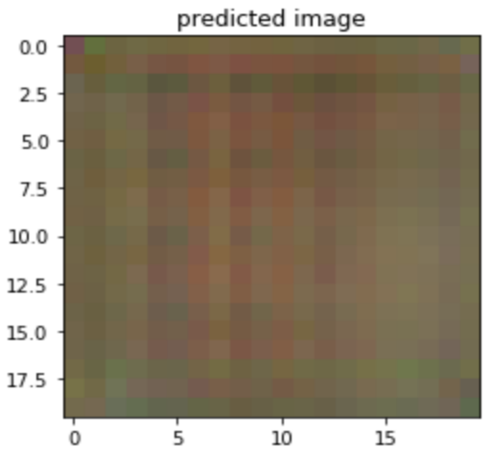


Figure 8 Results of using image augmenter in Fig.7







3) Data Augmentation Lessons Learned

Initially, we had accidentally set our learning rate to high at 0.1. This resulted in a path with lots of spikes and pits. By decreasing the initial learning rate to 0.01, we noticed a much smoother descent.

We also noticed that this neural network doesn't have a major effect on pixels very far away from the border. This is to be expected because convolution has a minimal field view. We attempted to get around this problem by using a fully connected network, but that only lead to other problems such as losing all features.

Also, something, we noticed is that that when you upsample the data and add padding the edges will be off color compared to the rest of the image. This can be solved either by not using padding when upsampling or using a fully connected network. Unfortunately, both of these methods result in quite a bit feature loss. As seen in the examples to the side, the fully connected layers don't have a weird border, but lack all the unique details from the red and black blocks. When, you merely unsampled and don't add padding during convolution. Your network is oddly shaped to meet the valid zone, such that the field view of your filter is shrunk. This results in a similar result to your fully connected layer where the system. I believe the proper way to get around this bug. Would be to replicate the final border of each image rather than zero-pad before convolution.

III. Summary

Merely computing Mean squared error is not enough for us if we want to augment the data with the border or see substantial improvements. As is, we see that the system, doesn't get much improvement from adding the border if we merely matched MSE from the original to decoded version. This network seems to be very good at replicating images with low gradients, any complex patterns get completely lost in this network.

As a result, I believe that a different loss function is required to give the border improvement greater effectiveness such as comparing the gradients between the two images. Also, I noticed the borders from the decoded image have some weakness. The idea is that by forcing the network to replicate the gradient it can more closely replicate its boundary. Another idea is that we can feed more data that includes complex patterns into this network if majority of the patterns are DC. This is less of a problem since our augmented border is the closest to our weakest point. This is likely due to us padding the image on the side to side with zeros during the unsampled side. Overall, more work can be done in combining these two networks to create a better final result. Also, it may be interesting to see if the part 1 autoencoder could be trained on a loss function comparing the gradients between the desired image and the final image.

Appendix

Dataset Link- https://github.com/openimages/dataset