Super Resolution CNN

Chad Maycumber

November 14, 2018

Abstract

In this assignment we try to model the Zero Shot Super Resolution paper with a Convolutional Neural Network that is built from scratch. The goal of this algorithm is to improve the quality of images when images are rescaled to a different size. This works by training separate convolutional networks for a almost 7 training sets of different image sizes then compared to the final image, and applied to each rescaled image until the image is transformed into the desired size with less blur from rescaling the image.

1 Introduction

We explore the Super Resolution Algorithm to recreate images at a larger resolution with less blur. This is a practical application that could be very handy when scaling small or large images to reduce the amount of random noise in the image and produce something that should be not only easy for human eyes to understand but also for computers to get better data.

2 Hyper Parameters

Learning Rate: .01

2.1 Adam

espAdam: .000000001
beta1: .9
beta2: .999

2.2 RMSProp

espRMSProp: .000001
weightDecay: .9

2.3 Momentum

momentum: .9
weightDecay: 2.5

3 Convolutional Layer

3.1 Approaches

For Loops: The original approach I took to construct the convolutional layer class involved creating the forward and backward propagation with a series of for loops that looped over the total amount of images in the set, the height of the image, width of the image and the number of channels in the image. Within the inner loop I would create a slice of the image that needed to be convolved and apply the weight by multiplying the Kernel to with the same indices as the slice of the original image and then apply the bias, updating the final output of the next convolved layer. Although this method seemed to me as the most straightforward in practice it turned out to lead to very slow computation times, thus leading me to shift my approach to a more numpy based matrix multiplicative method that allowed for me to compute computational layers at a higher speed which speed up testing.

Numpy Matrix Multiplication: After searching the web for a long time trying to find a quicker approach for building the convolutional layers I was able to stumble upon a method that use Matrix Multiplication and less for loops. This method looped over the height and width of the image but used Numpy tensordot to compute the dot product of the given arrays over a specified axis that allowed for faster computation of the convolutional layer by being able to compute the dot product across the channels at a single time rather than looping through each individual channel. This is especially useful in our scenario being that the channels in almost all of our convolutional layers being 64 because the creation of 64 filters from the previous input.

3.2 Pseudo Code

Algorithm 1 Convolution Layer Forward

```
1:  procedure CONV FORWARD
2:      outputHeight ← int((prevOutputHeight + 2*pad - f)/stride + 1)
3:      outputWidth ← int((prevOutputWidth + 2*pad - f)/stride + 1)
4:
5:      for range(outputHeight) do
6:          for range(outputWidth) do
7:              Slice ← paddedImage[:,h*stride:h*stride+filterSize,w*stride:w*stride+filterSize,:]
8:              outputMap[:, h, w, :] ← np.tensordot(Slice, Weight, axes=([1,2,3],[0,1,2])) + bias
9:
10:     Return outputMap, cache
```

Algorithm 2 Convolution Layer Backward

```
1:  procedure CONV BACKWARD
2:      for range(outputHeight) do
3:          for range(outputWidth) do
4:              vertStart, horizStart ← h*stride, w*stride
5:              vertEnd, horizEnd ← h*stride + filterSize , w*stride + filterSize
6:
7:              Slice ← paddedOutput[:, vertStart : vertEnd, horizStart : horizEnd, :]
8:              dPaddedImage[:, vS:vE, hS:hE, :] np.transpose(np.dot(W, dZ[:,h,w,:].T), (3,0,1,2))
9:
10:             dW ← np.dot(np.transpose(Slice, (1,2,3,0)), dZ[:, h, w, :])
11:             db ← np.sum(dZ[:, h, w, :], axis=0)
12:     dAPrev ← unpadded dAPrev
13:
14:     Return dAPrev, dW, db
```

4 Relu Layer

The Relu layer was relatively easy to implement. I simply constructed a class that had both forward and backward functions which implemented a Relu and a Reverse Relu function.

The Forward function takes in the previous output from a convolutional layer and applies Relu by taking all the values in the layer that are greater than zero.

The Backward propogation function takes in the previous output from a back prop layer and computes all the outputs that are great than zero to one rather than retaining the original value.

5 Optimization

I explored a variety of different optimization functions and eventually settled on testing both the Adam Optimization algorithm and the RMSProp algorithm.

Originally I had attempted to use a momentum optimization algorithm, however I had a very difficult time getting it to converge thus leading to less than favorable results.

The RMSProp algorithm worked well when I was trying to get a quicker convergence because of its more aggressive nature this helped get output in a smaller amount of epochs.

Finally the Adam optimizer seemed to work best for overall performance of the convolutional network. It required a greater amount of epoches when compared with the RMSprop algorithm but it yielded an extremely loss after about 10 epoches for most images. This means that the error was far less than with the other optimization methods.

Adjusting the hyperparameters also had an impact on all of the optimization methods in my model. I opted for a slightly higher learning rate with the Adam method over the RMSProp. This was alright because Adam has a 'buffer' built in that accounts for the warm up of the network which didn't cause my algorithm to diverge.

5.1 Pseudo Code

Algorithm 3 Adam Optimization

```
1: procedure ADAM UPDATE
2:     $m \leftarrow beta1 * m + (1 - beta1) * dx$
3:     $mt \leftarrow m/(1 - beta1 * *t)$
4:     $v \leftarrow beta2 * v + (1 - beta2) * (dx * *2)$
5:     $vt \leftarrow v/(1 - beta2 * *t)$
6:     $x \leftarrow x - learning_rate * mt/(np.sqrt(vt) + eps)$
```

6 Super Resolution

To complete the super resolution part of the assignment I used my CNN to first train the network to recognize sharp images by down sampling images and testing them against the same down sampled re-upscaled to create a blurry image.

For each one of these images I made sure to go through nearly half the image size worth of epochs. Then I repeated the process with more re-scaled images for the number of iterations that equaled the difference between the desired output resolution and the original image.

I followed this step by retraining the image to apply sharpening with the previously trained network on each image size in the previous step. This allows for the network to get used to up-scaling images from blurry images which will help us later when applying Super Resolution on the images we want to scale.

6.1 Propogation

```
1: procedure FORWARD PROP
2:     for i in NumConvLayers - 1 do
3:         C[i] ← convForward
4:         relu[i] ← reluForward
5:     C8 ← convForward
6:     Loss ← LossForward

7:     ReturnLoss
8: procedure BACK PROP
9:     dZ ← LossBackward
10:    for i in NumConvLayers - 1 do
11:        reluBP[NumConvLayers - i] ← reluBackward
12:        CBP[NumConvLayers - i] ← convBackward
```

6.2 Training

I initially trained the algorithm on small image that was 28x28 pixels to save time and be able to iterate over the image to reduce the loss as much as possible and produce meaningful feature maps. This may have had an impact on my final results when applying the algorithm to the higher resolution image of the sunflower.

The reason I chose to train my algorithm on a smaller image size is because I though that the scaling would be more abrupt and thus creating a greater difference between the blurry and non-blurry images. Also this was very computationally friendly because my algorithm requires a series of iterations through the original image before it moves onto the testing phase.

6.3 Testing

To test my results I used two images. A 100 pixel square image and the original image used for training the 28 square pixel image.

Both images seemed to have very slight improvement in the image quality when compared to the normal resize method in scikit image. The 100 pixel image seems to have the most improvement probably because it is at a larger scale. Training on a lower res image did seem to work when applying the feature map to the larger scale image and produced relatively favorable results.

Due to computational constraints of both size and speed I had trouble processing any images that were larger than the 100 square pixel image I tested. However I do believe that the results should scale.

7 Added Elements

I chose in many cases to normalize the output of the images that the algorithm was creating because it usually provided slightly better results when compared with the un-normalized image which was often very noisy and had very extreme values. Although I debated whether this was a good approach or not, the results seemed to produce favorable outputs when compared to leaving the data un-normalized.

8 Results

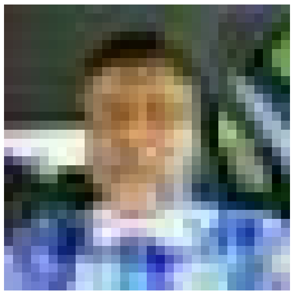
8.1 100 X 100 Pixel Image

Super Resolution Resize:

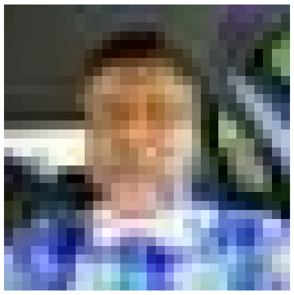
Normal Resize:

8.2 28 X 28 Pixel Image

Super Resolution Resize:



Normal Resize:



9 Conclusions

Overall I was able to improve the sharpness of images slightly when upscaling them to a greater resolution. Although my outputs don't seem as profound as the ones created in the Zero Shot Resolution paper, they do offer a slight improvement over the original rescale method reducing the blurryness and creating a slightly better looking image.

I think the affects are most notable on images of higher resolution and training with higher resolution data may improve the affect even more dramatically as the size of the images gets larger and various only slightly in resolution to the test image.

10 Run Instructions

\$ $python\ SuperResolution.py - i < image > -m < mode > -d < dimensions >$

References

[1] https://github.com/wiseodd/hipsternet

[2] http://cs231n.github.io/neural-networks-3/

[3] http://openaccess.thecvf.com/content$_cvpr_2018/papers/Shocher_Zero - Shot_Super - Resolution_Using_CVPR_2018_paper.pdf$