CS230

Honey, I Shrunk the Image: Image Compression with Deep Convolutional Neural Networks

Kiko Ilagan **Neel Yerneni** **Varun Nambikrishnan**

Abstract

Image compression is an important task for the media-heavy Internet of the present and future. In this project, we apply a compression framework implemented using deep convolutional neural networks with an alternating optimization scheme to the problem of image compression. The approach is compatible with existing image codecs and usable with many types of images. While the task of image decompression still proves challenging, the approach has been proven to work for grayscale images and shows promise for higher quality color images.

1 Introduction

The rapid scaling of modern technology has driven demand for more compact ways to store media. In addition, development of higher and higher quality content has rendered modern standards for image encoding inefficient and lacking in quality. Fundamentally, there are two types of image compression: lossy and lossless. Briefly, lossless compression allows for complete reconstruction of the original image whereas lossy compression generally involves approximations and loss of original information in order to save more storage space. Our aim is to use neural networks to achieve the space savings of lossy compression while maintaining the detail of lossless compression.

We apply a deep learning approach to the task of image compression. The input to our algorithm is a color image. We then use 2 back-to-back Convolutional Neural Networks. The first network, coupled with an image codec such as JPG, produces a compressed, space-efficient encoding of the image. The representation can then be decoded and fed into the second network. The second network de-compresses the encoding, outputting an image that ideally looks nearly identical to the original image.

2 Related work

Image compression has recently captured the attention of the deep learning community. As such, there has been a variety of different neural network architectures produced. Toderici et al.[1] use a RNN encoder/decoder combination along with a binarizer and entropy coder neural network for image compression. Different types of RNN cells are tried, including a new hybrid GRU/ResNet cell. Rippel et al.[2] use a GAN architecture for image compression. The authors use a multiscale adversarial model to allow for the reconstructions of images to match the ground truth inputs even at low bitrates, while traditional codecs produce increasingly more noticeable distortions (blurriness, pixelation, etc.) as the bitrate is lowered.

CNNs, a natural architecture for dealing with images, have also found success in image compression and this is the architecture we choose for our own model. Our own model implementation closely follows that of Jiang, Tao et al.[3] in their paper An End-to-End Compression Framework Based on Convolutional Neural Networks (2017).

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

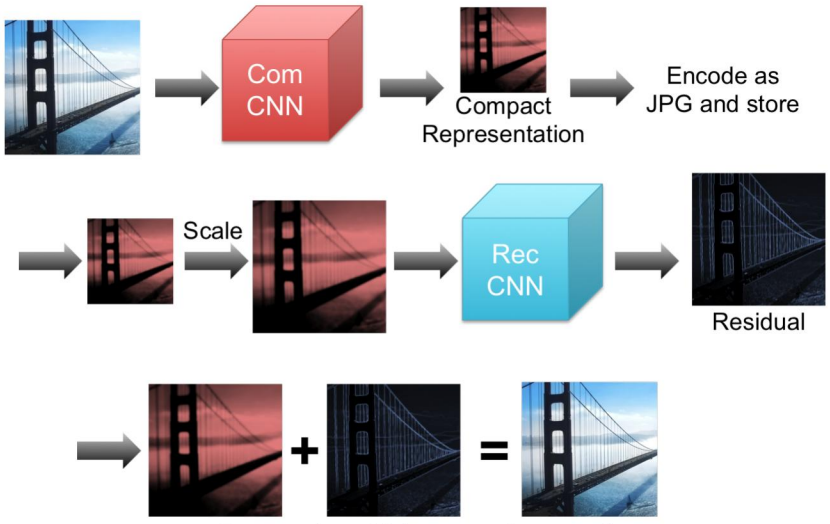
3 Dataset

Due to the nature of the image compression task, we don't need images of specific objects or even data labels, as long as the images are of reasonable quality. Thus, we sampled our images from the Food Image Dataset[4] published by the Multimedia Signal Processing group. It is a general image dataset containing pictures of food, people, animals, and landscapes, ranging from low-resolution amateur photos to high-quality professional images. We followed the suggested split of Jiang, Tao et al. and trained on four hundred images in the training set, fifty images in the holdout development set, and ten images on the test set. The authors of the Jiang, Tao et al. paper state that more training examples did not lead to significant gains in performance. Prior to training, images were all resized to be 180x180 pixels.

4 Methods

4.1 Approach

We propose a system comprised of two neural networks, working collaboratively. Images are first fed to a Compact Representation Convolutional Neural Network (ComCNN), which outputs a compact representation of the image scaled to half its original size.
This compact representation is then encoded through an image codec (JPG in our case). For training, the image is then immediately decoded and scaled back up its original size using Bicubic Interpolation.
The scaled-up image is then fed into the Reconstruction CNN (RecCNN), which outputs the residual between the scaled-up image and the original, uncompressed image. The residual can then be added to the scaled-up image to restore the original image. In practical application, ComCNN and RecCNN could be separated. The compact representation output by ComCNN that is subsequently compressed by the image codec could be stored efficiently. Then, when the image needs to be decompressed, it can be retrieved, scaled up, and put through RecCNN to obtain a high-quality image with minimal loss of original detail.



Proposed model layout and approach

Towards the beginning of the project, we experimented with more end-to-end approaches that eschewed the image codec altogether. Originally, we used a deep convolutional network that learned a compressed version of the image, coupled with a deep deconvolutional network that would restore the image. Unfortunately, the results of initial experiments suggested that while compression was easy to learn, the deconvolution was much more difficult. The reconstruction network only learned to output blurry images, even after reducing the size of the training set to purposely overfit. We suspect that either the loss function we formulated for the end-to-end architecture was insufficient, or the architecture itself is unsuited to the task. Regardless, such an end-to-end system is an avenue of study

worth considering in future work.

4.2 Architecture

ComCNN is comprised of three weight layers. First is a convolutional layer that convolves sixty-four 3x3 filters over the inputs with stride one and same padding, followed by a ReLU activation. The next layer convolves sixty-four 3x3 filters with stride two and same padding and a ReLU activation - besides learning spatial and content information, the added purpose of this layer is to downscale the image to create a more compact representation. Finally, the last layer convolves three 3x3 filters with stride one and same padding - this layer outputs a 90x90 representation that can be interpreted as an RGB image.
This image is then encoded through an image codec, in our case JPG. The compressed image is then decoded. The 90x90 RGB image is then scaled to the original size of 180x180 using Bicubic Interpolation. The resulting upscaled image is then fed into RecCNN.
RecCNN is comprised of twenty weight layers. First, the upscaled image is convolved with sixty-four 3x3 filters with stride one and same padding, followed by a ReLU activation. This layer is then followed by eighteen identical layers, each of which convolve the input with sixty-four 3x3 filters with stride one and same padding then perform a ReLU activation. Finally, the result is fed through one last convolutional layer with three 3x3 filters to output a 180x180 RGB image, which should be the residual between the original image and the upscaled image. The residual can then be added to the upscaled image to restore the original image.

4.3 Loss

Since optimize two separate CNNs, we have a separate loss functions for ComCNN and RecCNN. Let x be the image, $Rec(*)$ be the output of RecCNN, $Com(*)$ be the output of ComCNN, and $Cod(*)$ be the output of the image codec. The loss for ComCNN is:

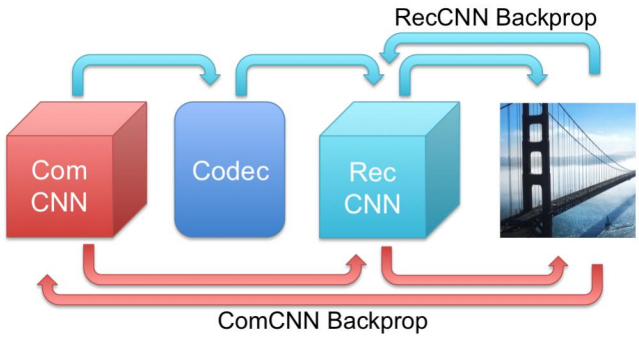
$$L_C(\theta_C) = \frac{1}{2N}\sum_{k=1}^{N} \|Cod(Com(\theta_C, x_k)) + Rec(\theta_R, Cod(Com(\theta_C, x_k))) - x_k\|^2 \tag{1}$$

$$\approx \frac{1}{2N}\sum_{k=1}^{N} \|Com(\theta_C, x_k) + Rec(\theta_R, Com(\theta_C, x_k)) - x_k\|^2 \tag{2}$$

Note that the codec function involves a non-differentiable rounding function, meaning that a single continuous backpropagation is impossible. To combat this, we make the approximate assumption that the codec can perfectly encode and decode the image with no loss of detail, which is a reasonable upper bound to the performance for the purposes of the loss function. Thus, ComCNN is optimized using equation (2) above.
The loss for RecCNN is:

$$L_R(\theta_R) = \frac{1}{2N}\sum_{k=1}^{N} \|Rec(\theta_R, Cod(Com(\theta_C, x_k)) - (Cod(Com(\theta_C, x_k)) - x_k)\|^2 \tag{3}$$

We are free to use the codec for RecCNN's loss function since it is not applied to RecCNN's output. Since equations (2) and (3) are interdependent, we use an alternating optimization scheme - we hold RecCNN parameters fixed while updating ComCNN with equation (2), then hold ComCNN fixed while updating RecCNN with equation (3).



The training/optimization cycle. The networks are optimized over two separate forward/backward propagations. ComCNN skips the codec for training.

4.4 Training

We initially trained on a single image to sanity check (to see if it was able to learn) our model, then proceeded with training on the full data. Following the iterative alternating approach detailed above, a single iteration consists of getting a compressed output from ComCNN and the codec, forward and backward propagating through RecCNN, and using the output from RecCNN to backpropagate through ComCNN.
The networks were trained for 50 epochs using the Adam Optimizer with learning rate .001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We used a batch size of 20 images and experimented with L_2 regularization. Training was performed on an AWS GPU, and takes roughly four days to complete.
Models were evaluated using a metric called the Multi-Scale Structural Similarity Index (MS-SSIM) between the original and reconstructed images. MS-SSIM is an image metric that better captures human intuitions of similarity than other distance metrics such as pixel difference[5]. It ranges from 0 to 1, where an MS-SSIM of 1 between two images means they are identical.
Due to the limited time and budget allotted to use, we were unable to perform a satisfactory cross-validation and fully combat the overfitting that we tended to see in the model.

5 Experiments/Results/Discussion

Our compression CNN performed quite well, and was able to produce a compact representation that was easily compressible by the codec. However, the compression CNN did struggle to effectively compress heavily polychromatic images. We hypothesize that Jiang et al. chose to train and develop their network on grayscale images for this reason.

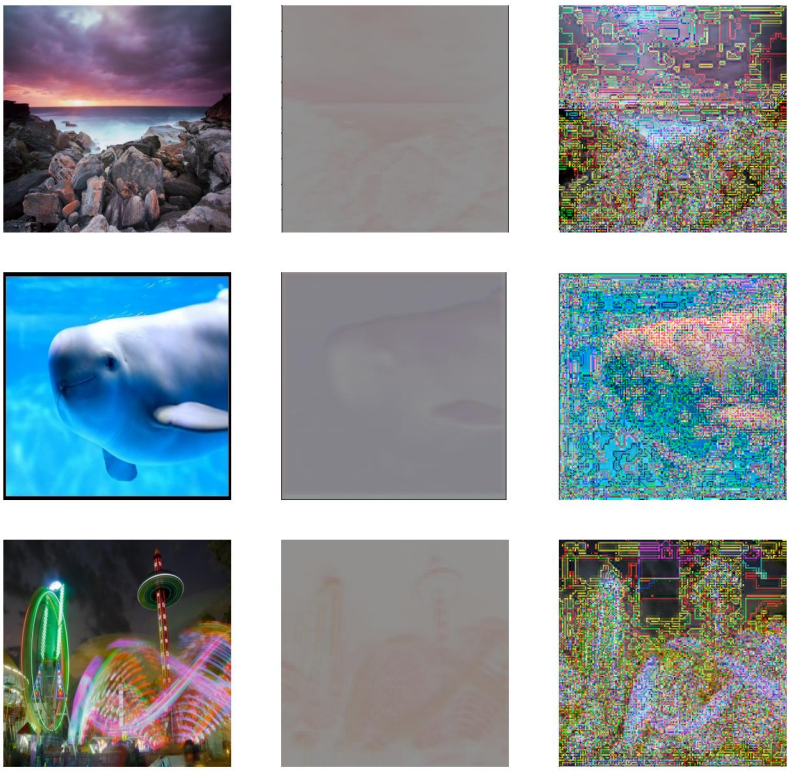
Average MS-SSIM by Cohort

	Train	Dev	Test
Model	.8895	.4948	.4628
JPG	-	-	.82

The average MS-SSIM of the model over the images in each cohort. Performance of JPG image standard is presented for comparison

Our reconstruction CNN did not perform as well. Specifically, the model heavily overfit the training set (even with L_2 regularization) and thus the reconstruction loss on the dev/test sets were quite high. Additionally, while the reconstructed images were able to capture the general coloration/shapes found in the original image, the reconstructed images were quite noisy.
We experimented with different levels of batch normalization and found that performing no batch normalization led to better quality reconstructed images for color photos.

Example Image Output



Example images from the test set. Original images in left column, compact representation in the middle, reconstructed image on the left. MS-SSIM scores and qualitative inspection suggested that there was overfitting, but the approach shows promise

As mentioned, the training time proved to be a significant challenge that hampered our efforts at more comprehensive cross-validation. When overfitting on a single image, we noticed that the model tended to learn highly exaggerated values for the image residual, leading to many reconstructions that saturated pixel values and white/black blocks and patches. ComCNN managed to minimize the loss fairly quickly however, so it is likely the case that reconstruction is a much more difficult task than compression.

6 Conclusion/Future Work

Our compression CNN was able to produce a good compact representation for the codec to encode, and we felt that part of our model was reasonably successful. However, our RecCNN overfits heavily while still not being able to reconstruct the finer details in the original images, which is why JPEG produces a better MSSSIM score. One way we could combat this would be to use more aggressive regularization; we used L_2 loss but we could experiment other regularization techniques as well. A high dropout rate may allow the networks to generalize much better to the dev/test sets. Another direction may be to play around with the losses as well. Mean squared error losses often produce blurry images[6] so playing around with the loss functions (using L_1 or the adversarial loss found in GANs) may help our reconstruction CNN.
Training for longer (after combating the overfitting problem, of course) may have also helped produce clearer reconstructed images. Additionally, using transfer learning using the pre-trained weights of VGG-18[7] may allow the networks to spend less time learning the shapes/colors of the images and instead allow them to learn optimal compact representations of these features.

7 Contributions

Varun - Set up input pipeline, set up single example training, set up msssim metrics/visualization of data, contributed to paper, general debugging of model
Kiko - Created model/training algorithm, created poster, contributed to paper, general debugging of model
Neel - Set up input pipeline, found dataset, AWS training of model, tuned hyperparameters, contributed to paper, general debugging of model

References

(1) Toderici, George, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. "Full resolution image compression with recurrent neural networks." arXiv preprint (2016).

(2) Rippel, Oren, and Lubomir Bourdev. "Real-time adaptive image compression." arXiv preprint arXiv:1705.05823 (2017).

(3) Jiang, Feng, Wen Tao, Shaohui Liu, Jie Ren, Xun Guo, and Debin Zhao. "An end-to-end compression framework based on convolutional neural networks." IEEE Transactions on Circuits and Systems for Video Technology (2017).

(4) Ecole Polytechnique Fédérale de Lausanne (EPFL), “Food-5k”, Dataset

(5) The TensorFlow Authors, Multi-Scale Structural Similarity, GitHub Repository, https://github.com/tensorflow/models/blob/master/research/compression/image_encoder/msssim.py (2016)

(6) Mathieu, Michael, Camille Couprie, and Yann LeCun. "Deep multi-scale video prediction beyond mean square error." arXiv preprint arXiv:1511.05440 (2015).

(7) K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556 (2014).

(8) The Tensorflow Authors, Tensorflow, Deep Learning Framework (2018)

Link to code repository: https://github.com/ilaganf/deep-shrinks/tree/master