ME AND CHATGPT: MAKING AN IMAGE PROCESSING CLASS DEMO SUITE

Al Bovik and ChatGPT

Colorado's Laboratory for Image and Video Engineering (LIVE)
Department of Electrical, Computer, and Energy Engineering
University of Colorado Boulder

ABSTRACT

I discuss my adventures in substantially recreating my class (again) Digital Image Processing to "catch up" to the wave of Artificial Intelligence that is sweeping all of technology space. This time, a complete destruction and recreation, in modernized form, of my software for live image processing demos. Rather than utilizing my programming skills to program these, I spent the Summer of 2025 working with ChatGPT (4.0 and 5.0) to create them. In the end, the courseware "we" created has been a roaring success (as proven in the classroom), but the process was hardly without hiccups! To summarize, I found these language models to be both amazingly capable and surprising incapable. But perhaps the most important thing I can report is that a domain expert, at least one having near-Jobian patience, can create sophisticated educational software *today* with effort matching what might have been needed in writing the code. This portends that not far off, continuously evolving AI models will become less fraught with limitations and become transformative at assisted courseware creation.

Index Terms— Digital Image Processing, courseware, image processing educational demos, large language models, ChatGPT

1. INTRODUCTION

In the Year 2020, just before the Covid epidemic smashed everyone's lives, I wrote a small paper entitled "Weeping and gnashing of teeth: Teaching deep learning in image and video processing classes," which I presented at this conference in March 2020 [1]. I had an interesting response to this paper. While it has cited little, I have received email communications on it from more people than any other paper I have written, and was even encouraged by David Donoho to "go on the road" to encourage other to take heart, and embrace the AI revolution. These e-letters generally commiserated with what I had felt when I was forced to reassess and erase half of my image processing class, replacing much suddenly obsolete material by deep learning methods. The current paper may be viewed as a follow-up to that prior paper.

2. RENEWED ANGST AND FEARS

I must say that it was to some degree, dragged kicking and screaming into this significant effort. While I had nicely upgraded my course material in years prior to reflect modern deep learning (as suitable for an upper-division class), the class was held back by my old digital courseware: dozens of Labview demonstrations of image processing algorithms. These had served me admirably for decades with only occasional updating, and included excellent, highly visual, interactive demonstrations of image processing methods, which the users could modify as they liked to see the effects of parameter choices instantly. It was excellent, but around the time that I wrote [1], I was realizing that yes, I could use the existing class demos for all the classical material still, and show *pictures* of the outcomes of deep learning algorithms. This was acceptable, since no one really expected a suite of live, sophisticated deep learning image processing algorithms to be collected together as a courseware. But I knew that day would come.

My only plan was to slowly bring deep learning based models into the class as picture and video demoes and eventually seek academic funding to create entirely new courseware. However, just about the time that I transitioned from UT Austin to CU Boulder, the decision was made for me. This happened when Microsoft updated the Windows OS back in 2024, and National Instrument made the decision to longer support older Labview formats. National Instruments was changing, had greatly reduced their interest in image processing, and soon would be purchased by Emerson Electric even as their share price stagnated.

In short, none of my suite of time-honored Labview demos that has helped me teach thousands of students the principles of Digital Image Processing operated anymore, nor could they be made to work by any modification. With considerable trepidation and misgivings, I starting mentally planning new courseware, realizing that it appeared as though I would have to spend a lot of time coding it up. For a busy fellow like me, the idea of finding that much time, even during summers, was pretty terrifying. It meant developing suitable GUIs, interactivity, coding models both classical and deep, gathering and using large image datasets to train deep models, and much more. Sheesh! This was not what someone at my career stage should be doing!

Of course, LLMs are being used increasingly to assist programmers of all varieties, and there were public statements from the likes of Mark Zuckerberg to the effect that human programmers at Meta Platforms were being replaced [2], with much of the work to be conducted by language models, ostensibly under the guidance, control, and prompting of human "super-programmers."

It was clear that language models would be very good at some things, such as creating GUIs. After all, building windows, displaying images, placing menus, dropdowns, and radio buttons should be easy. However, I planned to create dozens of demos (as it turns out, nearly 100), and many would have unique GUI requirements. Writing code for so many algorithms, including deep models, appeared a foreboding task. I wasn't sure I could even accomplish it over the temporal gap created by my transition to CU Boulder, a pretty clear Summer of 2025 to give it a try. As it turned out, it required two months of significant labor, occupying most of my days during that span.

3. "IT ALWAYS SEEMS IMPOSSIBLE UNTIL IT'S DONE"

Nevertheless, I felt intimidated by the effort required for this task. Not only because of all the programming, but also because I wanted to significantly expand the course material at the same time. I want to include deep models for denoising, compression, deblocking, inpainting, object detection and recognition, and much more. Using modern deep CNN and attention models, and yes, foundation models and (visual) language models. I did not have much time to accomplish this, just the summer, which I wanted to also enjoy hiking the mountains of my new home, Colorado! Along with everything else, it seemed an impossible task. As N. Mandela stated in his above quote, in a famous speech addressing far larger problems, tasks may seem impossible, but when effort is made, may turn out not to be so. But clearly, I needed help, and lots of it!

So, I turned to the famous language model ChatGPT which I had been using for all kinds of Q&A, and who would soon become my companion, AI-friend, co-worker, and often utterly frustrating, obtuse, numbskull.

Given my ambitions for the course material revision and for creating a comprehensive and (hopefully) amazing set of image processing demo courseware, while also seeking a learning challenge for myself, I made a decision that I hoped might make my life easier. I would create the courseware entirely without using my programming skills. I would work with "Chat" via the conversational interface, or prompting, to achieve the entire outcome, matching the need to demonstrate all the surviving material to be covered, as well as newer, more modernistic image processing demos of deep learning based image processing, suitable for upper division undergraduate ECE and CS students.

4. VIBING AND PROMPTING

I starting working on this project sometime around June 1, 2025. I was immediately impressed by Chat's ability to handle the creation of simple GUIs for image processing, including menus for algorithm variations and browsing for image files, buttons for choices, and for display of pictures. The first Module of the class has some very introductory Demos to give the students the feel of the dimensionality and parameters of images. With some simple prompts, I was quickly able to have Chat generate python scripts to demonstrate image down-sampling (with a slider to select factors of 2, 4, 8, or 16), image quantization (slider for bit depth reduction from 1 to 8 bits/channel), and for RGB to YUV conversion. In creating these demos, I realized there could be many similar repetitive commands going forward.

My solution was to ask Chat to remember "protocols" that be remembered for future scripts. For example, SPLIT PROTOCOL generates an interface having a browsing menu (to select an image), displays the image on the left half of the display window, *without* changing its aspect ratio, while displaying the outcome on the right side under the same constraints. This worked beautifully and saved me a lot of time! Chat is, after all, a very sophisticated computer program; defining operations and variables is natural to its operation, and I was delighted that Chat would remember these, even months later, and even today.

Significantly emboldened, I moved forward to Module 2, where I worked with Chat to develop interactive scripts for image histogram, binary thresholding (both of gray-scale only), and then too a step forward with morphological operators which required the definition of structuring elements and the basic binary morphological correction methods (DILATE, ERODE, OPEN, CLOSE, and so on). While these methods have a limited ranges of uses these days, I find they are an excellent way to introduce students to the concepts of spatiality and shape, separate from grey-scale variation. I was not surprised to find that Chat knew about all these operations and could import available routines from public libraries with ease, and along with the SPLIT protocol command, I developed these Demo programs with ease. I should say Chat and I, since while I gave the directives, Chat implemented them efficiently. In all of these there would be a few corrections, such as ensuring histograms were sized nicely, with tick marks.

I then moved on to point operations: full-scale contrast stretch, nonlinear point operations such as logarithms and gamma. Easy as pie, and all familiar to Chat.

I should mention that this is very much a deep learning for image processing class. But since I assume the students know nothing about machine learning (ML), I begin with Rosenblatt's Perception, move on to MLPs and the basics of ML, including backpropagation, and at this point introduce the first ML Demo: a simple MLP trained on a 1719-picture Day-Night image dataset (available to all that access this course, as are all the datasets) that learns to classify images and “Day” or “Night.” Not surprisingly, elaborately trained AI that he is, Chat was able to follow my instructions, to build a 2-layer MLP accepting 5 basic luminance features.

At this point I was becoming rather confident, and for Module 3 Chat and I developed special-purpose Fourier transform demonstration routines, with the ability to show reverse transforms of frequency bands and slices, and of special functions important in image processing. We handled these pretty effortlessly, as defining radial frequency and orientation bands is easy, and the rest was SPLIT PROTOCOL or similar (such as splitting the right half screen into two or three sub-displays) with some simple graphics. Module 4 covered linear filtering, filter banks, and introduced CNNs. Again, things went smoothly although I encountered some difficulties in creating the scripts for the Wiener Filter and Gabor Filter Bank Maker. Both these required multiple steps of implementation. Chat became confused at times, and left out steps, but we finally arrived. My patience at that time was pretty strong. The Filter Bank Maker was more difficult, as Chat had difficulty creating many objects (filters) satisfying frequency-intersection constraints. Towards the end of Module 4, I introduce CNNs and their construction, using VGG-16 [3] as an exemplar. Finally, we build a small and fun VGG-7 classifier (with around 2.7M parameters), trained on about 7500 exemplars, able to classify Dogs vs Cats mostly accurately!

Module 5 (Image Denoising) began easily again – I felt, I’m on a roll! With classical denoisers including the median filter, nonlocal means, and BM3D. These are again easy before-and-after Demos, using available routines. As we delve into more powerful networks – ResNets and Denoising Autoencoders, I find that again, prompting Chat through the creation of deep models is one of its strongest points. However, when training a ResNet-18 to conduct image denoising I encountered a difficulty. Since much of the course is devoted to perception-based image processing, I was interested in showing the power of using image quality models as loss functions. SSIM [4] and MS-SSIM [5] are ideal for this purpose, as they are differentiable and quasi-convex [6]. However, I discovered that existing Pytorch implementations of SSIM/MS-SSIM are unstable, at least in the context of using them as losses during training. There are a variety of reasons for this, related to vanishing gradients. However, I got past this, and developed ResNet-18 denoisers optimized using both MSE and MS-SSIM, with outcomes clearly showing the visual superiority of using MS-SSIM as a loss.Only three Modules to go, and less than a month had passed. A piece of cake!!

5. NOT A PIECE OF CAKE

At this point, I felt things were moving very well and I would have an earlier summer vacation that I thought. However, the last three Modules are the largest, and cover more difficult and complex topics like image compression, image quality prediction, and image analysis. Module 6 began with easy Demos of DPCM, BTC, and JPEG, for which there are available routines. An important thing to note is that I had been creating all of these easier scripts within a single long session. I began to notice that Chat began to make small mistakes or forget little things with increasing frequency. I inquired, and Chat confessed that its efficacy tended to be reduced with session length. I therefore began new sessions at suitable breakpoints. Later, I would do this more often! We then created a simple image de-blocker using the same ResNet-18 architecture as earlier used for denoising. This was easy to do, the model being trained on 5000 before-and-after JPEG images.

We then moved to deep image compression. A favorite model of mine, and influential, is the Ballé *et al* autoencoder [7] architecture that uses divisive normalization (DT) instead of ReLu or other common activations, and using the classic additive uniform noise model [8] to approximate / linearize quantization to allow backprop. We created our own Deep Compressor Demo using 9 layers of ResNet encoding to a bottleneck, and 8 layers of ResNet decoding. Instead of DT, we used ReLu, and left out quantization approximation entirely. Optimization was over MSE + L1 loss, combined with an entropy term on the bottleneck code.

During the creation of the Deep Compressor Demo, I began to encounter something I had several times earlier, but more severely: indent errors, and Chat’s inability to fix them. Indenting is fundamental to Python formatting, and misplaced indents produce errors. Recall my intention *not* to open the programs and fix them: hence I asked Chat to find them and fix them. Upon compiling the script and executing, the same error would arise again. Ask to correct, compile, and then again, the same error. One late night I counted 20 consecutive error messages, each fixing one error then allowing another. Naturally I prompted with “fix all indent errors” and “loop through and check your work for any indent errors.” Each time Chat would return with a cheery message that all is well, and *this time* our beautiful program would be perfect! Once I demanded Chat loop 100 times, and find all indent errors, before finishing. Chat cheerily replied that it surely would, once it had “passed my 100-loop gauntlet,” which seemed to pass for dry humor!

Keep in mind that I encountered this error with both ChatGPT 4.0 and 5.0, and when using the special coding mode. The jolly reassurances seem to be part of Chat’s nature, perhaps a programming team’s response to problems encountered by other users. However, this false cheer lost its impact on me pretty quickly. In the end, I found the best solution was to feed the model my last indent-error-free copy of the script, and tell it to try from there as a reference.

In the end, the Deep Compressor Demo works quite well, delivering compression in the range 40:1 – 100:1 with pretty good (given only 750K trainable parameters) de-compressed outcomes. Fig. 1 illustrates it in action, achieving 82:1 compression, with pretty good quality.

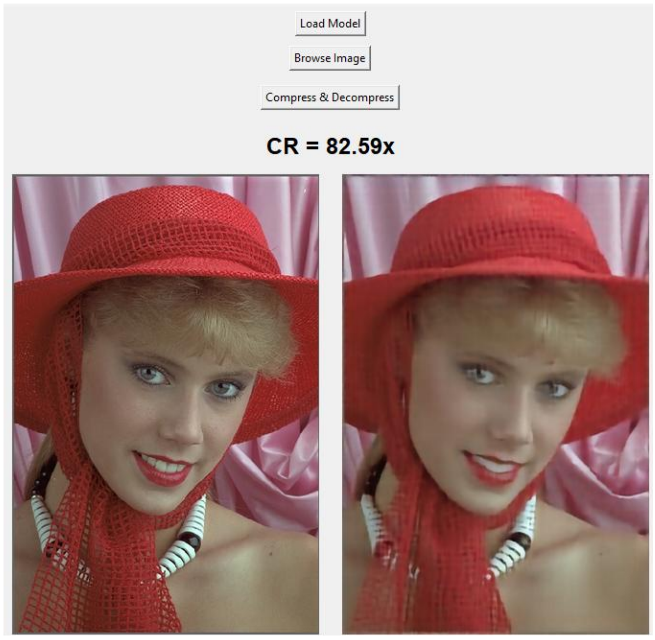

Fig. 1. The Deep Compressor Demo in action.

Things calmed down after that, and Module 6 concludes with self-attention models and a successful Transformer based Inpainting Model, able to fill small masked regions. This simple "before and after" model was easy to create.

Module 7 covers image quality prediction: SSIM, MS-SSIM, VIF [9], NIQE [10], PaQ-2-PiQ [11], and more. With the increased, multi-step sophistication of the Demos, greater problems were encountered. The first three were easy enough, as Python code exists, but NIQE presented important difficulties and more lessons. NIQE is a no-reference "blind" image quality prediction model related to BRISQUE [12], but with no ML component. There aren't any "correct" implementations of NIQE available in Python (there are some, but using ineffective approximations), so building a NIQE demo was necessary.

NIQE consists of multiple steps of computing local mean subtraction, and contrast normalization (MSCN), followed by histogram fitting to MSCN and products (correlations) of MSCN coefficients, over two scales, ultimately yielding 36 fitting parameters. These are expressed as a 36-D Gaussian model, which is compared to a similar model computed on a dataset of naturalistic images, the Mahalanobis Distances between them constituting the quality predictions.

Even given a long and very detailed expert prompt, Chat was unable to follow through at all on creating NIQE, leaving out entire steps, or critical portions of stages, and we could never get there by that approach. So instead, I sequenced the stages, allowing Chat to first conduct MSCN while plotting histograms, then computing best fits to various generalized gaussian models, and so on. Aside from a touch stretch of displaying 4 plots next to an MSCN image (depicting univariate and product histograms, over a choice of scales), with labels, this approach worked quite well, and had the benefit of allowing Demos of each of the steps. In the end, the NIQE Demo is quite successful, and is probably the best Python implementation available.

Following NIQE, we created a couple of Deep IQA demos, the first called "Colorado-QA" based on a large, pretrained ResNet-50 backbone (23.5 frozen parameters), which supplies "semantic awareness" with a Transformer-based head (8.5M learnable parameters), and trained on 9000 images from the FLIVE IQA Dataset [11]. The model performs quite well, apparently matching the prediction capability of the SOTA Feedback version of PaQ-2-PiQ [11]. Then I introduce a CLIP-based version of a similar model, called ColoradoCLIP-QA, using the ViT-B/32 version (151M fixed parameters) of CLIP to feed a similar but slightly smaller Transformer head (7.5M parameters), trained on the same data as Colorado-QA. Interestingly, the CLIP model converged in fewer than 5 epochs as compared to the ResNet (~40 epochs), delivering similar IQA performance, owing no doubt to exposure to vast troves of highly diverse images of all kinds of perceptual qualities.

Module 8 is the final one, starting with edge detection – which I still cover, despite the fading relevance of "edge maps," if only to hammer home the usefulness of the image gradient. Gradient, LoG [13], and Canny [14] edge detector Demos were easy-peasy, as was SLIC [15], the neat superpixel algorithm. The next Demo is another foundation model, Segment Anything (SAM) [16], an interesting comparison against the tiny and heuristic SLIC. Of course, SAM was originally trained on millions of images and billions of segmentation masks! This is followed by a various feature detectors including the hoary Hough Transform and the still-ubiquitous SIFT [18], the classic Viola-Jones face detector [19], and finally the classic YOLOv5 object detector [20], a masterwork of handcrafted deep learning! These Demos were pretty easy to create, given public-domain Python code. Somehow, I was done!

6. TIPS AND TRICKS

Thanks for reading! I am happy to share this courseware with any image processing educators that are interested. At some point I will make them publicly available. A few things which made this effort possible and which resulted in excellent courseware.

1. Produce and distribute compilable rather than executable code, so students can fiddle with and learn from it.
2. Create only small models (<10M parameters) using small datasets (<10K) so students' machines can handle them.
3. Define PROTOCOLS for repetitive Demo set-ups.
4. Limit the lengths of ChatGPT sessions to avoid fatigue
5. Ignore all of Chat's cheery assurances – it's probably hiding something!
6. When Chat produces errors repeatedly, restart the process from a known reference script without that error. Of course, store prior versions.
7. Break complex, multiple-stage processes into pieces, then assemble the completed pieces afterwards.

7. REFERENCES

[1] A.C. Bovik, “Weeping and gnashing of teeth: Teaching deep learning in image and video processing classes,” *IEEE Southwest Symposium on Image Analysis and Interpretation*, Las Vegas, Nevada, March 30-31, 2020.

[2] G. Marks, “Business tech news: Zuckerberg says AI will replace mid-level engineers soon,” *Forbes*, online at:https://www.forbes.com/sites/quickerbettertech/2025/01/26/business-tech-news-zuckerberg-says-ai-will-replace-mid-level-engineers-soon/, Jan 26, 2025.

[3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv:1409.1556. April 10, 2015.

[4] Z. Wang, A.C. Bovik, H.R. Sheikh and E.P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, April 2004.

[5] Z. Wang, E. Simoncelli, and A.C. Bovik, “Multi-scale structural similarity for image quality assessment,” *Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, November 9-12, 2003.

[6] S.S. Channappayya, A.C. Bovik, C. Caramanis and R.W. Heath, “Design of linear equalizers optimized for the structural similarity index,” *IEEE Transactions on Image Processing*, vol. 17, no. 6, pp. 857-872, June 2008.

[7] J. Ballé, V. Laparra, and E.P. Simoncelli, “End-to-end optimized image compression,” arXiv preprint arXiv:1611.01704. 2016 Nov 5.

[8] B. Widrow, “Statistical analysis of quantized systems” *IRE Transactions on Circuit Theory*, vol. CT-3, pp. 266–276, 1956.

[9] H.R. Sheikh and A.C. Bovik, “Image information and visual quality,” *IEEE Transactions on Image Processing*, vol. 15, no. 2, pp. 430-444, February 2006.

[10] A. Mittal, R. Soundararajan, and A.C. Bovik, “Making a ‘completely blind’ image quality analyzer,” *IEEE Signal Processing Letters*, vol. 21, no. 3, pp. 209-212, March 2013.

[11] Z. Ying, H. Niu, P. Gupta, D. Mahajan, D. Ghadiyaram, and A.C. Bovik, “From patches to pictures (PaQ-2-PiQ): Mapping the perceptual space of picture quality,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Seattle, Washington, June 13-19, 2020.

[12] A. Mittal, A.K. Moorthy, and A.C. Bovik, “No-reference image quality assessment in the spatial domain,” *IEEE Transactions on Image Processing*, vol. 21, no. 12, pp. 4695-4708, December 2012.

[13] E. Hildreth and D. Marr, “Theory of edge detection,” *Proceedings of the Royal Society of London*, vol. 207(1167), pp. 187-217, Feb. 1980.

[14] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 679-98, January 2009.

[15] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, “SLIC superpixels compared to state-of-the-art superpixel methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274-2282, May 2012.

[16] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland L. Gustafson, T. Xiao, S. Whitehead, A.C. Berg, W.Y. Lo, and P. Dollár, “Segment anything,” *IEEE/CVF International Conference on Computer Vision* (CVPR), pp. 4015-4026, Vancouver, British Columbia, June 2023.

[17] P.C.V. Hough, “Method and means for recognizing complex patterns,” U.S. Patent 3,069,654, December 18, 1962.

[18] D.G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[19] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR), Kauai, Hawaii, December 2001.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), Las Vegas, Nevada, June 2016.