CS230

Realistic Image Synthesis and Classification

Chris Fontas cfontas@stanford.edu
Wendy Li liwendy@stanford.edu
Emanuel Mendiola emanuelm@stanford.edu

Abstract

As techniques for creating photo realistic imagery evolve, it becomes increasingly difficult to differentiate between real photos and computer generated (CG) images. We utilize custom high pass filters and a CNN framework to take CG and photo inputs and classify them as CG or real. Building on past projects for photo vs. CG differentiation, we achieved a 99.7% accuracy on 100x100 resolution image blocks, outperforming current state of the art published results.

1 Introduction

Today film and game industries are constantly pushing the limits of CGI technology to please increasingly discriminating audiences. In the era of fake news, being able to pinpoint photos that have been computer generated can be a useful tool in determining which articles are trustworthy. A CG vs. photo classifier could be used as a benchmark to rate the realism of CG images. Furthermore, adding realistic CGI characters and effects to live action film is a painstaking and time-consuming process. A realistic image synthesizer using a conditional GAN has the potential to automatically refine CG images to look more realistic and reduce production costs.
Our project focused on addressing the problem of CG vs. photo differentiation through creating a binary classifier for differentiating between the two classes. The input to our algorithm is a realistic CG or high resolution photo image which is then passed into a CNN that outputs a CG or real photo prediction. Using our classifier as a discriminator within a conditional GAN, we focused on transforming CG image inputs into photo like outputs.

2 Related work

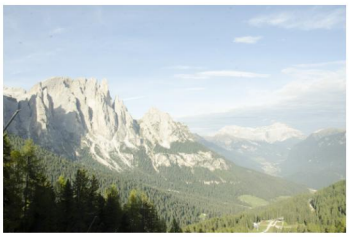
Our project was inspired by the work of Rahmouni et al, who worked on distinguishing photo realistic computer graphics from natural images using CNNs.[5] Images were broken up into small 100x100 tiles that used a custom pooling layer to extract simple statistical features from each tile and used a CNN framework for the classification. The best accuracy reported was 93.2% which was achieved by using simple statistics and a 2 layer CNN. While this accuracy is high, there is still room for improvement as only the most intricate CG generated images can fool the human eye.
Building on from Rahmouni's work, Yao et al proposed a custom image preprocessing and CNN-based model training approach.[9] Yao et al clipped images into smaller nxn patches that were then passed into three high pass filters. The filtered image patches were then fed to a 5 layer CNN-based model for training. Yao et al were able to achieve 100% accuracy but were restricted to image patches of size 650x650 pixels or larger. Due to the large size of the image patches, many of their training images were too small to meet the minimum size requirement.
Based on the work of Rahmouni et al and Yao et al, we consider Rahmouni's results to be state of the art for lower resolution smaller images and Yao's to be state of the art for larger image sizes. Since we cannot beat 100% accuracy, the goal of our work will be to achieve near 100% accuracy for smaller image patch sizes of 100x100 pixels.

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

3 Dataset and Features

Our dataset consists of images from two databases. The first is the RAISE database [1], which houses a collection of over 8000 raw images taken from various DSLR cameras at resolutions of 3000-5000 x 3000-5000 pixels. The second is the Level Design Reference Database [3], which contains over 60000 screenshots of photorealistic video games at resolutions of 1000-1800 x 600 pixels.
To maintain a fair comparison to the work of Rahmouni et al and Yao et al, we used the same training, testing, and validation images specified by 3 csv files provided by Rahmouni et al. We implemented a script that downloads the images from RAISE and converts the .NEF raw format the images come in to PNG through the LibRaw[8] and SFML[2] image libraries. Our training set consisted of roughly 2500 images from each database, evenly split between real and CG images. These images were then decomposed into 100x100 pixel patches, for a total of 40,000 patches. The validation set contained 546 images and the test set contained 723 images with the same distribution as our training set.

(a) Example CG image from Level Design Reference database



(b) Example real photo from RAISE database

Figure 1: Sample images from the RAISE database and Level Design Reference database

4 Methods

Our approach was to take code provided by Rahmouni et al to train a baseline model for comparison. After running Rahmouni's pre-trained model and weights on the provided test dataset (361 CG and 360 real images), we achieved the baseline performance shown on table 1.
While Rahmouni reported a final accuracy of 93.2%, we were unable to reproduce his results with the pretrained weights and data set he provided. Rahmouni did not provide any source code for compressing and formatting RAW images from the RAISE dataset to JPEG besides mentioning a 95% compression rate. In converting our RAW images to JPEGs, differences in our compression method may have influenced the lower accuracy score. Since we were unable to replicate his exact results, we will be using our proven model performance findings as our baseline model.

Accuracy: 0.875	Precision: 0.808	Recall: 0.98	**F1 Score: 0.8875**

Table 1: Baseline Performance

4.1 Model Description

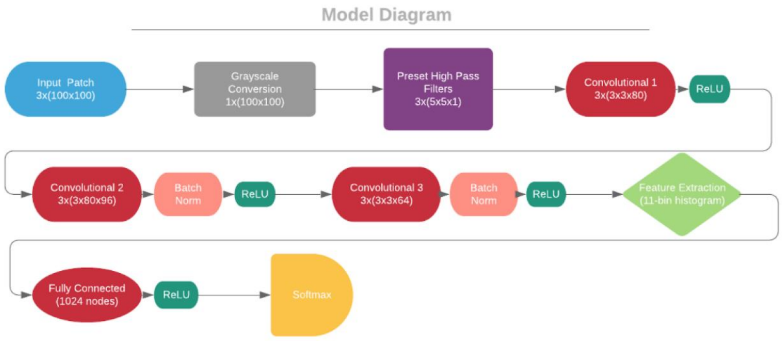


Figure 2: Diagram of model implementation

Our model is a modification on top of the original Rahmouni model that achieved the highest accuracy score (simple stats feature extraction + 2L MLP). After compressing RAW images into PNG form

and dividing them into image patches, we apply an additional greyscale and high pass filter to the input before proceeding with the original Rahmouni model seen above. The loss function that we are minimizing is the cross entropy function. Below we will go into further detail about the reasoning and steps taken in our approach to improve the original model.

4.2 JPEG to PNG

While both reference papers Rahmouni et al and Yao et al used JPEGs for their image classification, we chose to implement image classification using PNGs because JPEG compression is lossy. File compression for PNGs is lossless and has the potential to retain useful information that a JPEG would otherwise lose. In switching over from compressed JPEG to PNG images using the pretrained weights, we achieved the following model performance. Even though the original weights were trained on JPEG images, adding further information by using a PNG file improved accuracy by 6.5

Accuracy: 0.94	Precision: 0.91	Recall: 0.98	**F1 Score: 0.943**

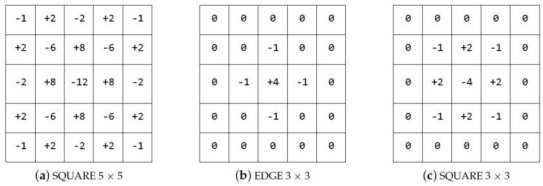
Table 2: JPEG to PNG results

4.3 High Pass Filters and Grayscale

In the paper by Yao et. al, it was discovered that better results could be achieved by removing low frequency content from the training data and bringing out the sensor pattern noise (SPN) in the images. This was done by converting each RGB image to a grayscale format and convolving them by three high pass filters (HPF). The high pass filters are given in figure 3. We replicate this same approach by inserting two new hidden layers at the front of the model, the first to convert each input to grayscale and the second to apply the HPF. With the new high pass filtering and grayscale, we were able to improve the accuracy and precision of the model significantly. Despite a slightly lower recall rate, the new changes increase the F1 score of the model from 0.943 to 0.954.

Accuracy: 0.96	Precision: 0.98	Recall: 0.93	**F1 Score: 0.954**

Table 3: High Pass Filter and Grayscale results



(a) SQUARE 5 × 5 (b) EDGE 3 × 3 (c) SQUARE 3 × 3

Figure 3: High Pass Filters

4.4 Error Analysis

Initially, we used the original baseline model's pre-trained weights to perform error analysis on our validation set. For reference, a false negative means we incorrectly predicted an image was a photo and a false positive means we incorrectly predicted an image was CGI. After analyzing a few categories we suspected could contribute to reduced accuracy, these were our findings:

Image Type	Number of False Positives	% of Total	Number of True Negatives	% of Total
Had Drawings in Image	8	9.411764706	14	5.07246377
Architecture Subject	33	38.82352941	64	23.1884058
Nature Subject	27	31.76470588	146	52.8985507
Person/ People Subject	21	24.70588235	20	7.24637681
High Color Saturate	11	12.94117647	15	5.43478261

Table 4: Error Analysis results

Our original classifier misclassified images that contain drawings (e.g. mosaics or paintings) more often than it correctly classified them as photos. It also struggled with classifying architecture and people. Lastly, it performed worse when classifying highly saturated images. To address this problem, we decided to add in the high pass filters from the Yao et. al, paper. High pass filters reduce the contributions of low-frequency pixels which contribute heavily to the overall appearance

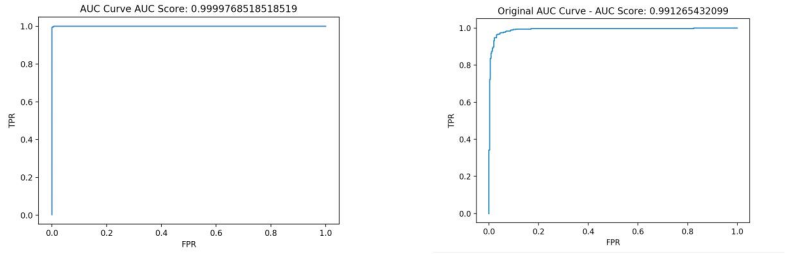
of the content, and exaggerate the high-frequency noise (generated generated from camera sensors) between pixels, which is more prominent in photographs than in CG images, regardless of the actual content. The high pass filters worked well, and after applying them along with grayscale, most of the misclassified photos of architecture and hand drawn images became correctly classified.
One adverse effect of these changes was that most subsequent misclassified CG images were heavily blue-saturated. We believe this is due to the way the grayscale component is calculated. The standard grayscale formula, as it is implemented in TensorFlow[7] (I= 0.21R + 0.72G + 0.07B) seeks to mimic the human eye's sensitivity to light of different wavelengths, assigning the least weight to blue. This means that images that are saturated with blue result in very small grayscale values that might hinder the model's ability to learn those values. However, the percentage of images that suffered from this effect was not significant enough to warrant further model architecture modifications. Therefore, we proceeded to hyper-parameter tuning.

5 Results and Discussion

Using a random grid approach to tune the learning rate, dropout, and filter sizes hyper-parameters. Table 5 lists our best results and hyper-parameters along with Rahmouni et al's for comparison. For our model an adaptive learning rate approach was taken. The model was trained for 15000 iterations at a learning rate of 0.001 and then for 1200 more iterations at a learning rate of 0.0001. Since we wanted our model to have a balanced recall and precision values, we tuned our hyperparameters to optimize for the F1 score. Figure 4 shows the AUC curve for our final best model and for the Rahmouni model.

Model	Filters	Patch Size	Learning Rate	Dropout	Recall	Precision	Accuracy	F1
Rahmouni	[32,64]	100	0.001	0.65	0.98	0.808	0.875	**0.8875**
Yao	[8,16,32,64,128]	650	0.001 γ=0.001	1.0	1.0	1.0	1.0	**1.0**
Our Model	[80,96,64]	100	0.001 and 0.0001	0.79	0.99722	0.99722	0.99722	**0.99722**

Table 5: Best results after hyperparameter tuning



(a) AUC curve of our final highest performance model

(b) AUC curve of Rahmouni baseline model model

Figure 4: Figure a of our final model in comparison to figure b, the baseline model generated by Rahmouni's model

5.1 Model Generalization

(a) (b)

Figure 5: Example hyper-realistic CG images taken from the Octane Renderer

To test our model for overfitting to our dataset, we sampled 245 hyper-realistic CG images taken from galleries from both the POV Ray [4] and Octane path-tracing rendering engines [6] (figure 5). These images possess a level of photo-realism that exceeds those taken from the Level-Set Design Database. When run through our model, we were able to achieve a 96% recall rate. To gauge the usefulness of this percentage, we selected a subset of 30 images to show to a dozen volunteers and ask them which images they thought were CG and which were real. To avoid bias based on the subject matter of the images, only those CG photos depicting real-world phenomena were chosen and those depicting fanciful creatures or science fiction were left out. The average recall rate of our human volunteers was roughly 80%, a significantly worse performance than our model.

5.2 Using our Classifier as a Discriminator for conditional GAN

Since our newly trained classifier is designed for differentiating between CG images and photos, we were curious to see if it would be an effective discriminator when used within a conditional GAN. Pushing our entire model in as the discriminator to our GAN was not successful. Upon closer inspection, we noticed that the D loss when training was approaching 0, signalling that the discriminator was not misclassifying any images and effectively not allowing the generator to learn what features it needed to produce to trick the discriminator. As a result, we continuously stripped away layers of our original classifier, removing our grayscale filter, custom high pass filters and finally our feature extraction and fully connected MLP layers until our GAN was able to converge and produce recognizable outputs.
The code for our conditional GAN was obtained from work by Zhu et al [10]. We trained on 400 images of landscape and nature scenes from the RAISE database [1] and 400 images of realistic landscape and nature images from video games like the Witcher, Skyrim and Tomb Raider, pulled from online desktop background albums.

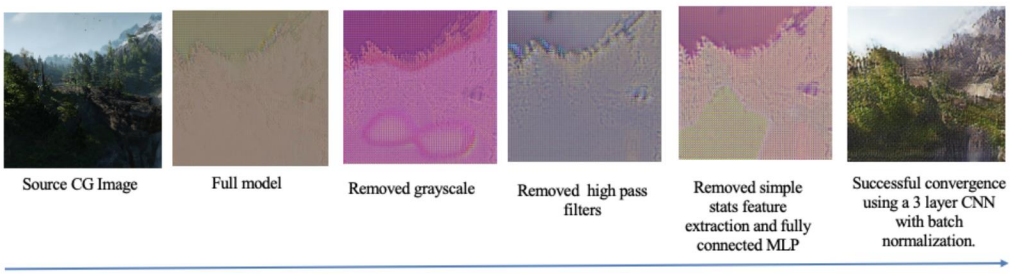


Figure 6: From left to right, our initial source image for transformation then the results of training with our classifier as our GAN discriminator; the layers that were successively removed from each attempt are described underneath each image

6 Conclusion and Future Work

In conclusion, we believe our model was able to exceed published state of the art results by outperforming the F1 score reported by Rahmouni et al as well as getting virtually the same score as Yao et al (0.997 F1 score vs. 1.00) while only using 100x100 image tiles as opposed to their 650x650 tiles. As the quality of computer graphics imaging has been increasing rapidly over the past decade, it is likely that our neural net would become obsolete quickly unless it is regularly updated with new data. The CG images used to train were chosen due to their photorealistic qualities, but there already exist CG imagery whose photorealism exceeds those we used to train, specifically those generated by engines which use offline Monte Carlo methods, such as POV Ray or Octane Renderer mentioned above. Thus with more time and resources we would like to include these images in future training. Additionally, our model currently assumes a clear distinction between photography and CGI, however CGI is often blended together with real footage as can be seen in any Hollywood blockbuster film. One possible avenue of improvement then could be to train a model to detect which aspects of a given image are CG and which are real, instead of focusing on classifying whole images as one or the other.

7 Contributions

Chris wrote the scripts for downloading and converting images from raw NEF to PNG, implemented the high pass filters, grayscale for our model and the bicubic interpolation to shrink images to test with CycleGAN, and did additional testing and analysis on the hyper-real images. Wendy compiled

together the CycleGAN dataset, refitted our model to be used as a discriminator in the GAN, and conducted error analysis. Emanuel handled setting up the GPUs on AWS and getting our model running on their platform, and worked on the hyperparameter tuning that gave us our final model with the best performance.

8 Code

The code for our model can be viewed and downloaded from Github using the following link: `https://github.com/wendyli/CS230_DL_Project`.
The code for our modified version of CycleGan, using our model as the discriminator, can be found using this link instead: `https://github.com/wendyli/finalproject`

References

[1] Duc-Tien Dang-Nguyen, Cecilia Pasquini, Valentina Conotter, and Giulia Boato. Raise: a raw images dataset for digital image forensics. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 219–224. ACM, 2015.

[2] Laurent Gomila. Simple and fast multimedia library. https://www.sfml-dev.org/, 2018.

[3] Piaskiewicz M. Level design reference database. level-design.org/referencedb, 2017.

[4] The POV-Team. Pov ray hall of fame. www.povray.org, 2018.

[5] Nicolas Rahmouni, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Distinguishing computer graphics from natural images using convolution neural networks. In *Information Forensics and Security (WIFS), 2017 IEEE Workshop on*, pages 1–6. IEEE, 2017.

[6] Refractive Software. Octane renderer showcase. home.otoy.com/render/octane-render, 2018.

[7] Google Brain Team. Tensorflow. www.tensorflow.org, 2018.

[8] Alex Tutubalin. Libraw: Raw image decoder. https://www.libraw.org/, 2018.

[9] Ye Yao, Weitong Hu, Wei Zhang, Ting Wu, and Yun-Qing Shi. Distinguishing computer-generated graphics from natural images based on sensor pattern noise and deep learning. *Sensors*, 18(4):1296, 2018.

[10] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.