CS221: Artificial Intelligence: Principles and Techniques

CNNs for Photorealistic Computer-Generated Imagery Detection

Gabriel Mukobi
gmukobi@stanford.edu

Shawn Zhang
szhang22@stanford.edu

https://github.com/mukobi/CS221-Project

1 Introduction

In this past decade, computer-generated imagery has improved drastically — so much so that the line between fantasy and reality is greatly blurring. Especially in this time of fake news, it is absolutely critical that we protect against disinformation and similar threats to authenticity. Especially noted, if CGI was used in fields like justice and journalism, then CGI would undermine trust in everything that we watch [1]. Thus to counteract this danger, our project seeks to build a binary classifier that is able to detect whether an image is a real photographic image (PIM) or just a photorealistic computer-generated image (CG). Our inputs are RGB images of "real" and "fake" scenes, where we output both our classification as well as our confidence.

Note: Several others have built DeepFake detectors to combat faked facial animations. We believe that more generalized synthesized image detectors like ours could be valuable for ensuring people the authenticity of any kind of image.

2 Literature Review

Ng et al. (2005) [2] proposed a geometry-based image model, which was motivated by the physical image generation process. With their physics approach, they were able to reveal particular physical differences between PIM and CG, such as gamma correction and/or sharp structures. Their technique was notably able to achieve a classification accuracy of 83.5%. Note, this research was conducted before the widespread use of convolutional neural networks (CNNs), which we mark as Krizhevsky et al. (2012)'s ImageNet [3].

The most recent, state-of-the-art work comes from Yao et al. (2018) [4]. For their method, they first clipped an image into patches, used three high-pass filters to enhance the image's sensor pattern noise, and finally fed the image through a five-layer CNN. In doing so, they were able to achieve an accuracy of 100%. However, since the resolution of each image patch is 650×650, they noted that their proposed CNN cannot work for such smaller images. Hence, there is this tradeoff between accuracy and computational limitations.

In consideration of the above two papers, we have decided to also adopt CNNs due to (1) the ability to use spatial context, (2) translational invariance, and (3) the ability to use less total parameters from weight-sharing – these reasons should boost our classification accuracy. However, we have decided to not use the image patches approach in order to make our pipeline more general and flexible to account for all sort of images.

,

3 Datasets

3.1 Initial Dataset

As an initial starting point, we used the Columbia Photographic Images and Photorealistic Computer Graphics Dataset, a dataset for multimedia forensics research [5]. This dataset included 3600 images split into 4 different classes. The classes were:

Personal: 800 photographic images (PIM) from the Columbia dataset authors' personal collections and 400 images from the personal collection of Philip Greenspun. These have reliable sources but somewhat limited diversity in content, camera, and photographer style.

Google: 800 photographic images (PIM) from Google Images searches for the categories from the CG set. These have more diverse image content, but the ground truth may not be as reliable as the Personal class.

CG: 800 photorealistic computer-generated (CG) images taken from mostly 40 3D-graphics websites.

Recaptured CG: 800 of the same images from the CG class, but displayed on an LCD monitor and recaptured with a physical camera to restore the photographic effect.

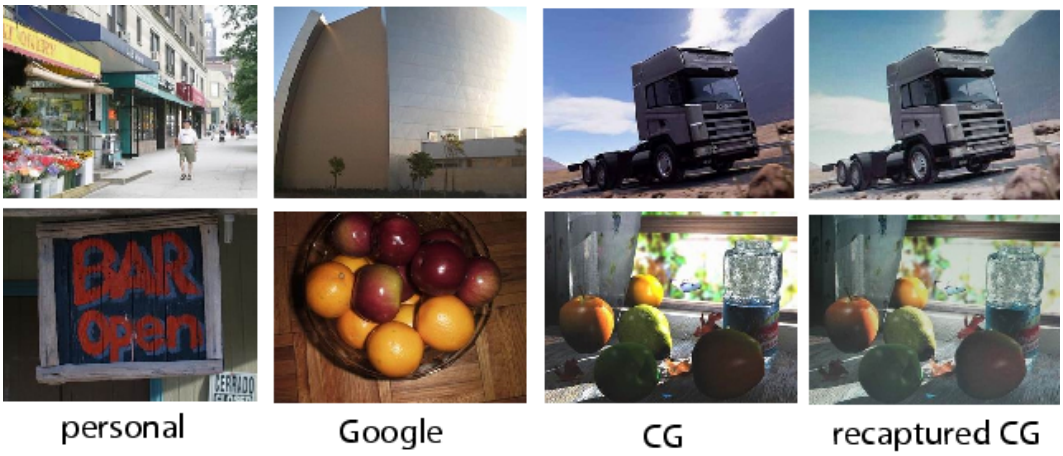

Figure 1: Example images from the Columbia Dataset [5].

Between these 4 classes, we primarily only care about the binary distinction between "real" PIM images (Personal, Google) and "fake" CG images (CG, Recaptured CG) for the purposes of this project. For any given input image, we wish to predict the corresponding output classification as either photographic images (PIM) or computer-generated (CG).

We used this dataset to train our baseline model and early stages of our actual model. However, it quickly became evident that there were several issues with this dataset, the most relevant being the fact that it was compiled in 2005 and thus the quality of the CG images in it were not up to the same standards as today. Additionally, this led to other problems of overfitting, where the model was not general enough to reasonably be put to use on modern CG images.

Because of this, we decided to expand our data by adding two new, modern datasets. These were the Level Design Reference Database (LDRD) [6] for CG images and the RAISE Raw Image Dataset for PIM images [7].

3.2 Updated Datasets

The Level Design Reference Database (LDRD) contains a large and growing number of mid-resolution screenshots from modern video games that highlight level-design and in turn, high graphical fidelity. At the time of writing, the database contains over 60,000 images from more than 170 different titles, of which we used 1832 for our CG dataset. Two examples from this dataset can be seen in Figure 2.

Figure 2: Example CG images from the Level Design Reference Database dataset [6].

Meanwhile, the RAISE Raw Image Dataset contains 8156 photographic images taken by 4 photographers on various cameras. These are uncompressed, high-resolution, camera-native RAW images, and the full dataset is over 350 GB! Below are two examples from this dataset.

Because of the large size of this dataset and because the files were in a high-resolution RAW .TIF files, we had to do some preprocessing. First we selected to use RAISE 2K for our dataset which only contains a subset of 2000 images. We wrote a script to download RAISE 2K from the list of image URLs that the dataset provides.

Then, we needed to convert the raw images to jpg compressed files (smaller and better compatibility with Python's Pillow library). We used ImageMagick to convert the .TIF raw images to 80% quality jpg images and resize them to a resolution where the minimum dimension is 650 pixels but preserving the aspect ratio of the total image by proportionally scaling the other dimension. This brought this dataset down to 259 MB.

Figure 3: Example PIM images from the RAISE Raw Image Dataset [7].

4 Methodology

4.1 Preprocessing

Before feeding our images into the model, we performed preprocessing to fit our network architecture and reduce the variance in our inputs.

Hence, we resized each image to a square of 256×256. Additionally, we normalized each image's color channels relative to the total dataset, centering the data and encouraging faster learning.

4.2 Model

Advancing from our baseline's simple logistic regression, we propose a new CNN to better distinguish between CGI and natural photos.

Here we breakdown our best and final proposed neural network:

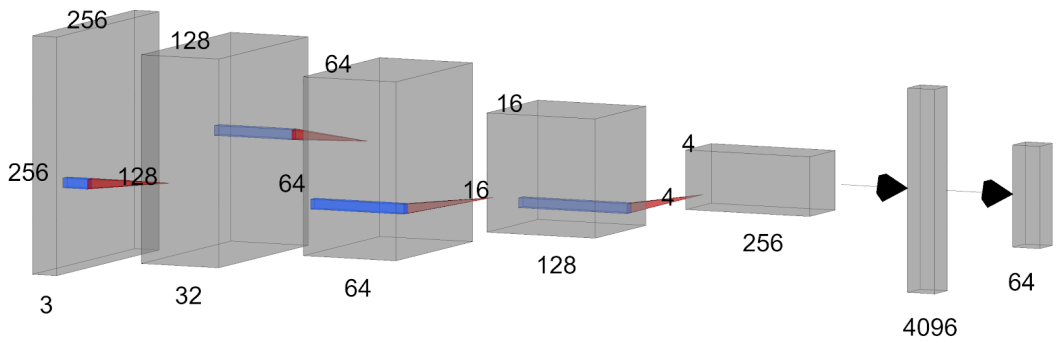

Figure 4: Diagram of our final CNN model. Created via NN-SVG [8].

1. Resizing of the input image to (3 x 256 x 256) and normalization of the color channels
2. First convolutional layer
   2.1. Convolution with 32 (5 x 5) kernels, stride of 1, and padding of 4
   2.2. Batch normalization and ReLU
   2.3. Max-pooling with (5 x 5) kernel and stride of 2
3. Second convolutional layer
   3.1. Convolution with 64 (5 x 5) kernels, stride of 1, and padding of 4
   3.2. Batch normalization and ReLU
   3.3. Max-pooling with (5 x 5) kernel and stride of 2
4. Third convolutional layer
   4.1. Convolution with 128 (5 x 5) kernels, stride of 2, and padding of 6
   4.2. Batch normalization and ReLU
   4.3. Max-pooling with (5 x 5) kernel and stride of 2
5. Fourth convolutional layer
   5.1. Convolution with 256 (5 x 5) kernels, stride of 2, and padding of 6
   5.2. Batch normalization and ReLU
   5.3. Max-pooling with (5 x 5) kernel and stride of 2
6. Flatten to (4096 x 1) vector
7. Fully-connected to (64 x 1) vector
8. ReLU
9. Fully-connected to (1 x 1) output
10. Sigmoid of output

Digression: For matching dimensions in our CNN, the following formula was immensely useful for us to calculate output shape post-convolution:

$$\lfloor \frac{n+2p-f}{s} \rfloor \times \lfloor \frac{n+2p-f}{s} \rfloor, \tag{1}$$

for an $n \times n$ image, $f \times f$ filter, padding p, and stride s.

Note: Whenever the sigmoid's output was > 0.5, we would classify the image as CGI (+1). Otherwise, we would classify it as real (0).

4.3 Training

For our training, we split our comprehensive dataset into 80% training and 20% validation.

The loss was calculated via Binary Cross Entropy (BCE):

$$Loss_{BCE}(\hat{y}^{(i)}, y^{(i)}) = y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}), \tag{2}$$

where $y^{(i)}$ is the i-th example's ground truth and $\hat{y}^{(i)}$ is the corresponding output of the sigmoid (i.e. prediction) from the model.

This loss was then optimized stochastically from the Adam algorithm with varying learning rates of 0.001, 0.0001, or 0.00001.

5 Experimentation

In this section, we justify our choice of model architecture and parameters. We experimented with the following hyperparameters: learning rate, preprocessing procedures (resizing or cropping the input dimension), model size (filter size and stride, number of convolutions, fully-connect layer neurons), batch normalization, L2 regularization, and dropout (both in the fully-connected layers and in the convolutional layers).

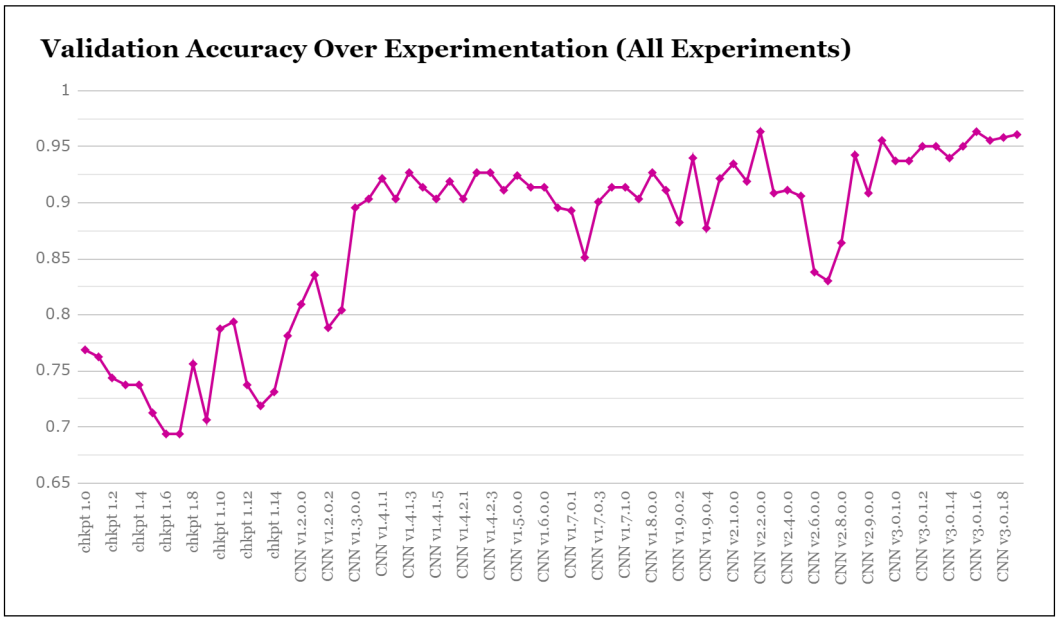

Figure 5: The validation accuracy of the best performing model from each experiment.

Note: For Figure 5, experiments are ordered chronologically from first on the left to last on the right. Additionally, "failed" experiments where the best validation accuracy was below 60% are not included.

5.1 Learning Rate and Preprocessing

First, we experimented both with the learning rate of our optimizer as well as the input dimension of our data. Here, input dimension refers to the side dimension of the square RGB image size that we resized all images to during preprocessing before feeding them into our network. For example, an input dimension of 128 means all of our input images were resized to 128x128 pixel RGB images or size 3x128x128 tensors.

Notable, since we were modifying the input size of our CNN, the output size of a batch after passing through the convolutional layers varied with input dimension. This means changes in input dimension correlate to changes in the number of neurons in our fully connected layers after flattening the final convoluted layers.

5.1.1 Learning Rate

From our experimentation, we quickly determined that a learning rate of 0.0001 is optimal. Higher learning rates weren't quite able to reach the optimum weights and sometimes exhibited exploding

Input Dim \ Learning Rate	1.0E-5	0.0001	0.001
32	0.6937500238	0.7125000358	0.6937500238
64	0.78125	0.71875	0.7312499881
128	0.7624999881	0.7687500119	0.5
256	0.7375000119	0.7437500358	0.7375000119
325	0.7875000238	0.7562500238	0.7062500119
512	0.7375000119	0.7937499881	0.4875000119

Figure 6: The best validation accuracy of models trained with varying learning rates and input dimensions.

gradients. Lower learning rates provided similar accuracy but took longer to learn. Because of this, we stuck to a learning rate of 0.0001 for most of our future experimentation.

5.1.2 Preprocessing

As for input dimension, results were more mixed. There was a balance to be found between low values like 32 pixels which lost so much semantic information from the image that learning became difficult and high values like 512 pixels which took longer to train on and potentially exhibited overfitted from the more complex model. We decided to settle on an input dimension of 128 pixels as a good balance between speed and performance for most of our experimentation but an input dimension of 256 pixels for our final model to squeeze out a little more performance.

Additionally, we experimented with different methods of preprocessing our images into the chosen input dimensions. These were resizing the whole image which distorts the aspect ratio but preserves all the semantic content of the image, randomly cropping the image which doesn't distort the image but loses part of it, and a combination of the two where we resized the image down so the minimum dimension was our input dimension (preserving the aspect ratio) and then cropping the other dimension to the input dimension (losing some semantic content but less than full cropping).

We tested this throughout various stages of our experimentation and found that resizing performed the best, followed closely behind by resizing with cropping, and significantly ahead of cropping with a change in validation accuracy between 5-10%. This might indicate that all of the image content is important to our classifier which might reflect potential bias in the dataset. We decided to use resizing in preprocessing for our final model.

5.2 Model Size

In determining the optimal model structure size, we mostly experimented with filter size and stride, the number of convolutions, and the number of neurons in our fully-connected layers.

For filter size, we found 5x5 filters to work fine in both our convolutional and max pooling layers with not much benefit from larger filters. Also, we hypothesized that image noise at the pixel-level might be useful [4], so we kept the stride of our first 2 convolutional layers to 1 but set it to 2 for the other 2 convolutional layers and the max-pooling layers in order to reduce the dimensionality of the data for faster training.

We also experimented with the number of convolutional layers. Our first models used only 2 convolutions and achieved mediocre performance with 70-80% validation accuracy at best. When we increased this to 3 convolutional layers (along with some regularization mentioned below), we suddenly were able to reach validation accuracies around 88-93%. We only tested up to 4 convolutional layers which reached our highest validation accuracy and which we used for our final model, though we hypothesize more than 4 convolutions might yield even more performance.

Last, we played around with the number and size of our fully-connected layers. This wasn't as extensively tested, but our testing indicated that 1 hidden layer with 64 neurons performed acceptably, so we kept it. This is potentially a source for model complexity and could be further tweaked to avoid overfitting.

5.3 Regularization

Early on, we ran into issues of overfitting with training accuracy above 90% but validation accuracy near 70%. To address this, we implemented three different forms of regularization.

First, we implemented batch normalization. Batch normalization reduces the internal covariance shift in each layer's inputs, accelerating learning and providing some regularization [9]. We applied batch normalization to each of our convolutional layers after the convolution and before activation with ReLU, and this helped significantly in reducing overfitting.

Next, we tried dropout, both in the convolutional layers and on each fully-connected layer. Dropout in the convolutional layers didn't do much, and conceptually, it perhaps doesn't make much sense to have dropout in between convolutions, so we left it out. Dropout on our fully connected layers did provide a little bit of regularization, but it was mitigated when we introduced batch normalization simultaneously. Thus, it seemed like batch normalization eliminated the need for dropout as sometimes occurs [9], so we left out dropout entirely on our final model.

Finally, we added L2 Regularization to our loss function. Again, we didn't see any significant positive change with batch normalization, so we kept batch normalization and left our L2 regularization on our final model.

6 Results

Using our best model outlined in the previous 2 sections, we were able to achieve **99.837% train accuracy** and **96.084% validation accuracy**. With more training, our models were quickly able to achieve 100% train accuracy, but validation accuracy didn't get much better than this. A graph of it's training is included below.

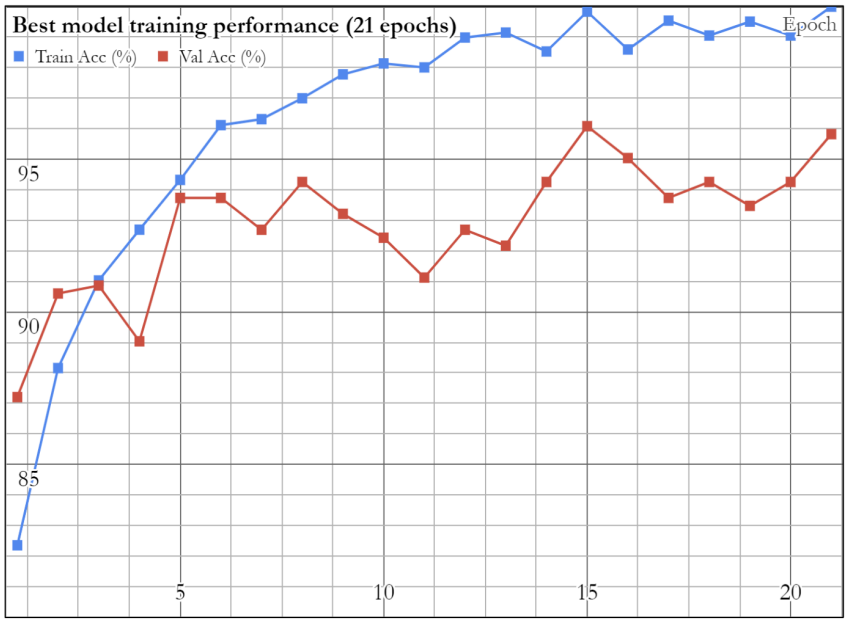

Figure 7: The training and validation accuracy of our best model over 21 epochs. The best performing model was reached after 15 epochs

In framing our problem and design our research proposal, we also tested the performance of a simple baseline model as well as a high-performing oracle model.

Our baseline model involved resizing each image to 32x32 RGB pixels or a size (3x32x32) tensor, flattening the image into a vector of length 3072, and running a single fully connected logistic regression followed by a sigmoid activation for our output. This baseline achieved a classification accuracy of 69% on the validation set.

Conversely, our oracle was human performance where a human was randomly given and equal number of computer-generated and photographic images in a random order and had to classify them one at a time. Our human oracles on average achieved a classification accuracy of approximately 90% on the validation set.

Our final model with a classification accuracy of 96.084% on the validation set clearly outperforms our baseline model and even beat our human oracle.

7 Analysis

From our best model, we then performed error analysis to obtain a better idea of our model's performance.

7.1 Confusion Matrix

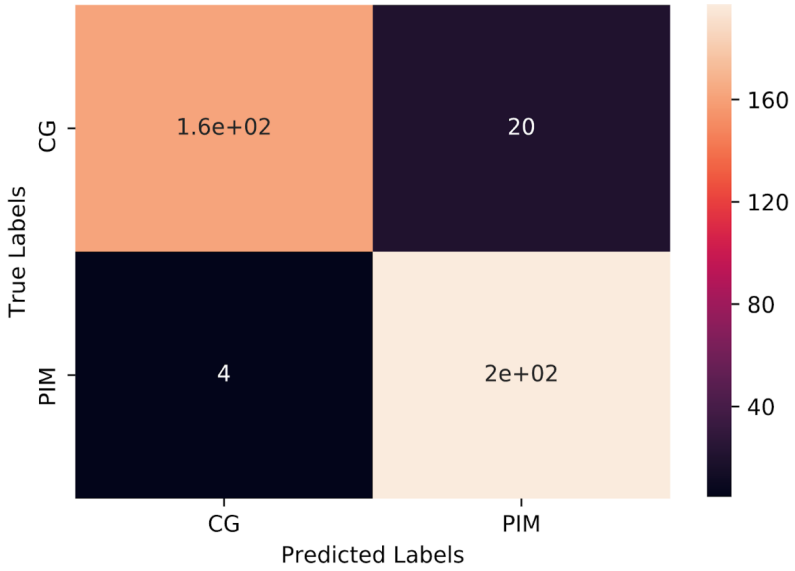

Figure 8: Validation confusion matrix.

From this confusion matrix on our validation data, we can see that the model is significantly more likely to misclassify CG as PIM than the other way around. However, this is also what we would expect – the confusion matrix is an indication on CGI's advancements in photorealism.

7.2 Saliency Mapping

In addition to the confusion matrix, we also constructed saliency maps in order to infer what our model was actually seeing. Please refer to Figure 9 below.

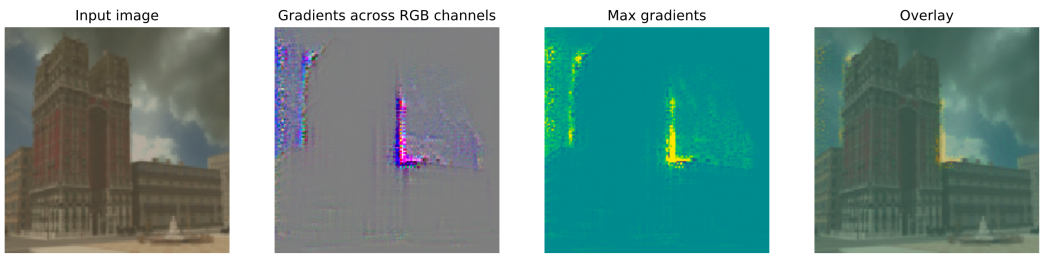

Figure 9: Saliency map for sample CG image. Created via FlashTorch [10].

Note: The input image has its color channels normalized. Analyzing the saliency map, we can see that the most influential pixels seem to outline edges. The model could be distinguishing PIM vs. CG from edge aliasing (i.e. CG images likely have more pristine edges).

With that in mind, we wished to then observe the saliency map of the image with the highest loss:

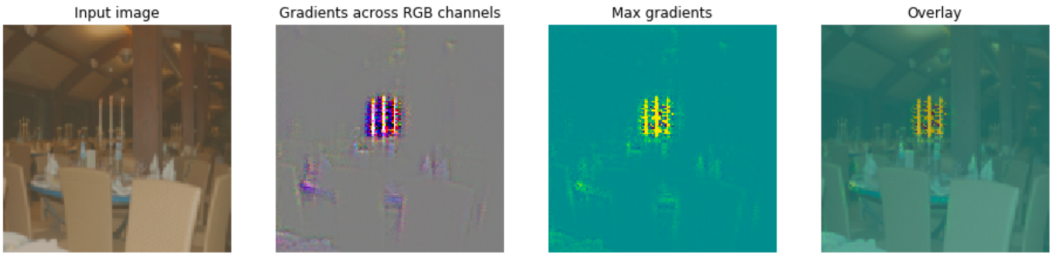

Figure 10: Saliency map for highest-loss PIM image. Created via FlashTorch [10].

Interestingly, this saliency map shows that the model really emphasized the candles centered in the picture. We hypothesize that those candles are being interpreted perhaps as gun crosshairs, which were prevalent in the Level Design Reference Database. By using so much video game data, our model may "cheat" and base its decisions too eagerly on the presence of guns or in-game UI. We may need to refine our dataset as a result.

8 Conclusions

Our work demonstrates a relatively simply-engineered method for building a computer-generate image detector using a convolutional neural network.

Related papers have been able to achieve similar or higher accuracy through techniques like convolving images through pre-built filters for detecting camera noise and/or training on image patches [4]. However, our model operates on the entire content of each image and requires no pre-engineered filters while still achieving quite a high validation accuracy of around 96.1%. That said, we believe our methods could be even further optimized to reduce overfitting and train a better classifier.

9 Future Work

Our data currently consists of CG images from 2005-2014. We wish to collect more recent CG images, challenging our model and demonstrating CGI's evolution over time. Additionally, we wish to diversify our dataset more. As of now, our dataset contains an abundance of video game screenshots due to CGI's prevalence in the industry. As a result, our model may be overfitting too much to video game features (guns, crosshairs, etc.). By collecting more movie CGI and computer art, our model should become more flexible for applicable settings.

Also, our dataset only contains strictly CG or strictly PIM images. However, modern-day graphics often contain a mix of the two. To extend our project, we may include CG/PIM hybrid images in efforts to perform segmentation and isolate the computer-generated portions. Such an application would be greatly beneficial to image forensics.

Finally, CGI usually comes with animation. Hence, we may enhance our model with a recurrent neural network (RNN), feeding in clips of CGI. With more data fed, our classifier ought to achieve even better accuracy.

10 Code

The code for our model, the compiled datasets with preprocessing scripts, data files containing training logs, and the best performing models can be viewed and downloaded on GitHub at the following link: `https://github.com/mukobi/CS221-Project`.

References

[1] Rocha, A., Scheirer, W., Boult, T., & Goldenstein, S. *Vision of the unseen: Current trends and challenges in digital image and video forensics.* ACM Computing Surveys 2011, 43, 26–40.

[2] Ng, T.-T., Chang, S.-F., Hsu, J., Xie, Lexing., & Tsui, M.-P. (2005). *Physics-Motivated Features for Distinguishing Photographic Images and Computer Graphics. ACM Multimedia*, Singapore.

[3] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks.* Communications of the ACM, 60(6), 84–90. doi: 10.1145/3065386.

[4] Yao, Ye., Hu, Weitong., Zhang, Wei., Wu, Ting., & Shi, Yun-Qing. (2018). *Distinguishing Computer-Generated Graphics from Natural Images Based on Sensor Pattern Noise and Deep Learning*. Sensors. 18. 10.3390/s18041296.

[5] Ng, T.-T., Chang, S.-F., Hsu, C., & Pepeljugoski, M. (2005). *Columbia Photographic Images and Photorealistic Computer Graphics Dataset.* ADVENT Technical Report, 205-2004-5.

[6] (2017). *Level-Design Reference Database*. [Online]. Available: http://level-design.org/referencedb.

[7] Dang-Nguyen, D.T., Pasquini, C., Conotter, V., & Boato, G. *RAISE - a raw images dataset for digital image forensics.* ACM Multimedia Systems, Portland, Oregon, March 18-20, 2015. 219–224.

[8] LeNail, A. (2019). *NN-SVG: Publication-Ready Neural Network Architecture Schematics. Journal of Open Source Software*, 4(33), 747, https://doi.org/10.21105/joss.00747.

[9] Ioffe, S.; Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. CoRR, abs/1502.03167

[10] Ogura, M. (2019). *MisaOgura/flashtorch*: 0.1.1 (Version v0.1.1). Zenodo. http://doi.org/10.5281/zenodo.3461737.