ResNets for detection of computer generated images

Gauri Bhagawantrao Jagatap
gauri@iastate.edu

***Abstract*—Computer generated images (CGI) are rendered by current 3D image modeling software, which is becoming increasingly good at generating photorealistic images. This can be a challenge, specially if such technology is used for misinformation and forgery. Hence the design of an effective classification technique is essential. In recent years, residual networks (ResNets) have shown superior performance in classification tasks. We present a residual network (ResNet) for the problem of classification of CGI and natural images and report empirical results on the same.**

***Index Terms*—Classification, CNN, CGI, Forgery detection**

I. Introduction

COMPUTER generated images (CGI) are images generated by computer software for a variety of purposes, such as modeling manufacturing designs, realistic looking imagery for computer games and animation for movies.

Image rendering software is becoming increasingly adept at generating photo-realistic computer generated images (PRCG), which also improves the likelihood of misuse of technology, particularly if used for misrepresentation.

In fact, recent literature suggests that even experienced photographers find it difficult to distinguish between natural and photo-realistic CGI [7].

With the advent of Generative adversarial networks (GANs), the production of photo-realistic images has become easier. NVIDIA [8] was able to produce thousands of fake celebrity images (see Figure 1).

Fig. 1. Photorealistic celebrity images generated by NVIDIA.

Therefore, the development of a detection mechanism to distinguish between CGI images versus real images is of importance.

Classification models for detecting Computer generated images (CGI) and real or natural images (NI), typically involve extracting appropriate features of the images, such as via Linear Discriminant Analysis (LDA) which is followed by a two-class classification using Support Vector Machines (SVMs) which are linear classifiers [5].

In the recent few years, the field of computer vision has seen great success in modeling and incorporating properties of natural images into the classifier structure, leading to improved classification accuracies. This was primarily realized via Convolutional Neural Networks (CNNs).

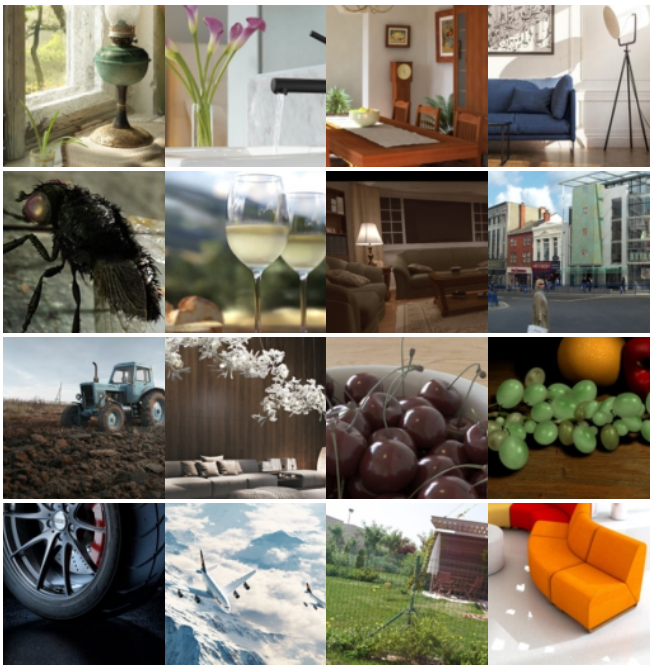

Fig. 2. Examples of photorealistic computer generated images used in this paper, from Columbia PRCG dataset and Google Images.

Recently, there has been increasing interest in Convolutional Neural Network (CNN) based approaches for classifying NI and CGI images [1], [3], [4].

II. Related work

Convolutional networks (CNNs) have been used with great success for the problem of CGI versus natural image classification. Specifically, in [1], the authors develop a five layer network, with the first three layers being convolutional in nature, to extract key features of the images and the last two layers being fully connected ones, representing a multi-layer perceptron (MLP).

The authors minimize the cross entropy loss of the effective network, and use an l1 regularization on the loss function. The dataset used in [1] is the Columbia Photographic Images and PRCG Dataset [2], which is open source.

A similar approach is adopted in [3]; they use two convolutional layers for feature extraction followed by a MLP and dropout for classification. The authors minimize the cross entropy of the effective network.

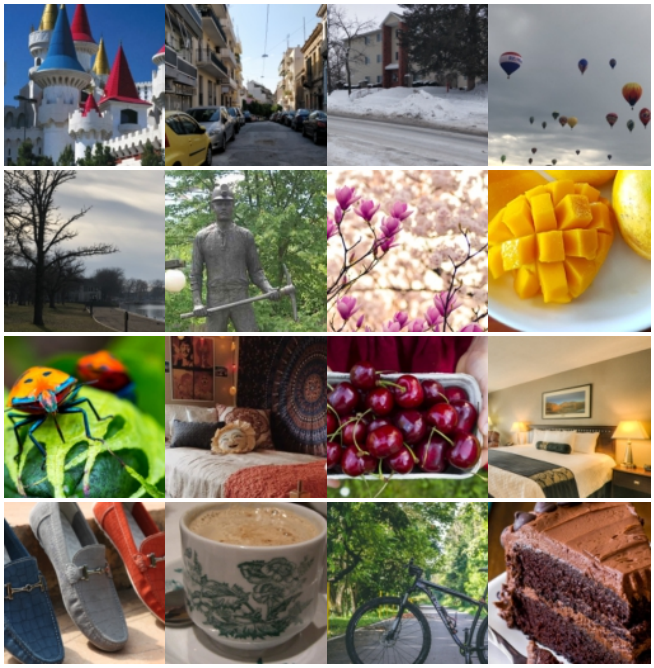

Fig. 3. Examples of natural images used in this paper, from OnePlus, iPhone cameras and Google Images.

In [4], the authors use a similar CNN framework, however, they add an extra pre-processing step where they extract sensor pattern noise information with the help of high pass filters. The basis of this step is the fact that natural images are typically captured by cameras which are prone to sensor pattern noise. Meanwhile, computer generated images are rendered virtually, hence there is no such noise. This modelling assumption can be used as a useful pre-processing step to distinguish between CGI and NI images.

III. This Paper

The application of CNNs to classification of CGI vs NI images is relatively new. There have been several advances in the formulation of Convolutional Neural Networks, which may be used to improve classification accuracy. Residual networks, or ResNets [6] were introduced for classification tasks, which utilized skipped connections to improve classification performance of CNNs. Hence, structuring a network with 3 or more convolutional layers, along with skipped connections, is a potential direction that can be pursued.

Moreover, a sensor pattern noise identification, in terms of high pass filtering operations, similar to [4] will be tried out, prior to the CNN classification stage.

The dataset that used for this project is from [2] to test our hypothesis and will be used across different neural network architectures to determine which architecture gives the best result.

A. Dataset and pre-processing

The Columbia PRCG dataset was used for obtaining Photo-realistic Computer Generated (PRCG) images. CNNs typically require large amounts of data to achieve good accuracies.

-1	+2	-2	+2	-1
+2	-6	+8	-6	+2
-2	+8	-12	+8	-2
+2	-6	+8	-6	+2
-1	+2	-2	+2	-1

Fig. 4. High pass filter for pre-processing inputs to CNN.

For this project, I have used 200 images which are photo-realistic computer graphic (PRCG) and 163 images which are either obtained via OnePlus5 phone camera or iPhoneX phone camera or natural images from Google Images. These images are then reshaped such that the maximum dimension (either height or width) is 256 pixels. After this, 5 patches of 128x128 size are extracted, from four corners and one from center, via the `five_crop` function from the `torchvision.functional` library. Thus the total number of images is 1815, belonging to either class 1 (PRCG) or class 2 (natural).

This dataset is then randomly split into 70% training (1271) and 30% (544) validation data. In each epoch of training, a composition of transformations is applied to each image:

A random horizontal flip is applied with probability $p = 0.5$. Random crops of size 64x64 are made from the 128x128 patches. All 3 channels of images are normalized.

Sample images from both categories are displayed in Figures 3 and 2.

B. Sensor pattern noise

In [4], the authors employ a pre-processing step that leverages the fact that CGI images will have no sensor-pattern noise. Since noise is high frequency information, this information is extracted out in the first step, using the high pass filter shown in Figure 4, which is applied on each individual channel of all training images. Note that this pre-processing step dramatically improves classification accuracy, as can be seen in Table I.

C. CNN architecture

A CNN architecture with residual blocks is considered. We vary freeze number of convolutional layers between 3 and 11 layers, while a two layer MLP is considered for the final layer. The architecture of the best performing setup is as follows:

The convolutional layers are referred to as `conv1`, `conv2`, and so on, depending on the number of convolutional layers. The last two layers constituting the multilayer perceptron are called `fc1` and `fc2`.

The Python code for the network is shown in the code snippet in Table II in the Appendix. The first two arguments of the `conv2d` module are the number of input channels and

TABLE I
TABLE TO COMPARE CLASSIFICATION ACCURACIES ACROSS VARIOUS TYPES OF NETWORKS.

Sr. No.	Pre-process with HPF	Conv Layers	MLP Layers	Residual blocks	Train Accuracy	Test Accuracy
1	No	4	2	None	57.73%	56.55%
2	No	4	2	1	62.98%	61.90%
3	Yes	4	2	None	64.06%	61.38%
4	Yes	4	2	1	68.12%	64.48%

number of output channels respectively. Stride length, padding and kernel size information is also fed to the function `Conv2d` which creates an object variable storing weights of the neural network.

Residual connections are made between the output of the first convolutional layer and the output after the third convolutional layer. Dropout is used for regularization.

IV. EXPERIMENTS

The experiments were run on NVIDIA GeForce GPU with 8GB RAM. The code was implemented on a Jupyter Notebook with Python 3.6 under the PyTorch framework.

Cross entropy loss is considered for training the network. Stochastic gradient descent (SGD) is chosen as optimizer with learning rate of 0.05. The batch-size of training set is chosen to be 128. Thus in each iteration of SGD, roughly 36 batches are passed to the optimizer. The SGD algorithm is run for 200 epochs for all experiments and weight decay parameter is set to 0.2 (ℓ_2 regularization).

The classification accuracies with three different network architectures are shown in Table I. We show a benefit of adding a residual connection in the comparisons between architecture 1 and 2, and between 3 and 4. We show the benefit of adding a high-pass filtering preprocessing step by comparing between architecture 1 and 3, and between 2 and 4.

Misclassifications with the best performing CNN (architecture 4) are shown in Figure 5

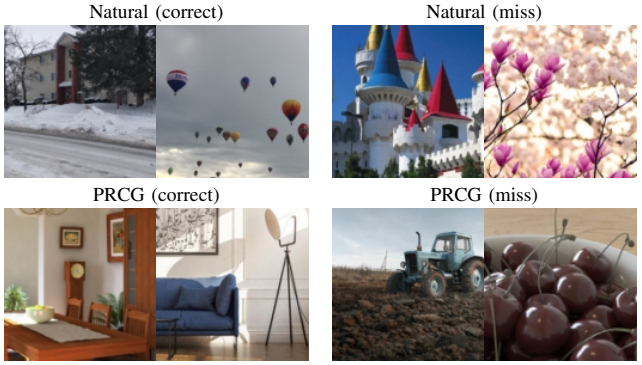

Fig. 5. Displaying images from both classes which were (a) classified correctly, (b) misclassified by architecture 4 from Table I.

V. DISCUSSION

The problem of classifying between natural and PRCG images is extremely challenging. For this project, the main challenge was obtaining the dataset, and pre-processing it.

Several network architectures were tried out. It was observed that adding residual connections improves the training performance of the neural network and reflects improved classification accuracies. Using sensor pattern noise information significantly improves training performance.

The classification performance can perhaps be further improved upon by trying more tweaks to the network architecture; additionally, the training was terminated at 1000 epochs, due to time constraints. Therefore the results from architectures 1 and 2 can possibly be further improved upon and this direction is reserved for future work.

VI. FUTURE DIRECTIONS

The problem of classifying between natural and PRCG images is a tough one. To solve this problem, residual connections have been leveraged. One of the main challenges for this project was the unavailability of a large enough dataset; even though we used data-augmentation techniques, the addition of more training samples will definitely help improve accuracy of classification.

Using prior information, such as image formation pipeline to either the preprocessing step or CNN architecture itself may improve training performance of the CNN.

REFERENCES

[1] W. Quan, K. Wang, D.M. Yan and X. Zhang, Distinguishing between natural and computer- generated images using convolutional neural networks. IEEE Transactions on Information Forensics and Security, 13(11), pp.2772-2787, 2018.

[2] T.T. Ng, S.F. Chang, J. Hsu, and M. Pepeljugoski, Columbia photographic images and photorealistic computer graphics dataset, Columbia Univ., New York, NY, USA, Tech. Rep. pp 205-2004-5, 2004.

[3] N. Rahmouni, V. Nozick, J. Yamagishi and I. Echizen, Distinguishing computer graphics from natural images using convolution neural networks. In 2017 IEEE Workshop on Information Forensics and Security (WIFS) (pp. 1-6), December 2017.

[4] Y. Yao, W. Hu, W. Zhang, T. Wu, Y.Q. Shi. "Distinguishing computer-generated graphics from natural images based on sensor pattern noise and deep learning". Sensors. 2018 Apr;18(4):1296.

[5] S. Lyu, and H. Farid. How realistic is photorealistic?. IEEE Transactions on Signal Processing 53.2 (2005): 845-850.

[6] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778), 2016.

[7] J. Ktsyri. "Those virtual people all look the same to me: Computer-rendered faces elicit a higher false alarm rate than real human faces in a recognition memory task". Frontiers in psychology, 9, 1362, (2018).

[8] T. Karras, T. Aila, S. Laine, J. Lehtinen."Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018.

APPENDIX

TABLE II
CODE SNIPPET OF NETWORK ARCHITECTURE

```
class SimpleCNN(torch.nn.Module):
def __init__(self):
super(SimpleCNN, self).__init__()
self.conv1 = torch.nn.Conv2d(3,16,
    kernel_size=4,stride=1,padding=0)
self.bn1 = torch.nn.BatchNorm2d(16)

self.pool = torch.nn.MaxPool2d(kernel_size=2,
    stride=2, padding=0)

self.conv2 = torch.nn.Conv2d(16,32,
    kernel_size=2,stride=1,padding=2)
self.bn2 = torch.nn.BatchNorm2d(32)

self.conv3 = torch.nn.Conv2d(32,16,
    kernel_size=4,stride=1,padding=0)
self.bn3 = torch.nn.BatchNorm2d(16)

self.conv4 = torch.nn.Conv2d(16,3,
    kernel_size=4,stride=1,padding=0)
self.bn4 = torch.nn.BatchNorm2d(3)

self.fc1 = torch.nn.Linear(3*29*29,1000)
self.fc2 = torch.nn.Linear(1000,2)
self.dropout = torch.nn.Dropout(p=0.7)

def forward(self,x):

##Convolutional block
x = self.bn1(F.relu(self.conv1(x)))
res = self.bn2(F.relu(self.conv2(x)))
res = self.bn3(F.relu(self.conv3(res)))
x = res + x#residual connection

x = self.pool(self.bn4(F.relu(self.conv4(x))))
x = self.dropout(x)

##MLP
#first layer of MLP
x = x.view(-1,3*29*29)
x = self.dropout(F.relu(self.fc1(x)))
#second layer of MLP
x = self.fc2(x)
return(x)
```