How Realistic is Photorealistic

CS-754 Project Report

Vivek Veer - 200050158

May 4, 2022

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. Thus there is a need for developing techniques to distinguish between photo-realistic and photographic images.

Here we have implemented a statistical model described in this paper, which seeks to achieve the same. Even though the authors managed to achieved an impressive classification rate of 99% on photo-realistic images, the paper itself was published in 2005. Thus the CGI used for training/testing were also older than 2005. The quality of CGI has increased exponentially since then and one of the objective of this project was to test how well the technique fares against a more modern data-set.

2 Paper Review

The paper exploits the statistical properties exhibited by the wavelet sub-band coefficients. They employ a four level quadrature mirror filter bank to split the frequency space into multiple scales, and orientations (a vertical, a horizontal, and a diagonal subband). The decomposition is applied independently for each colour channel, giving us three subbands for each scale and colour.

They then proceed to extract features based on these coefficients. First half of the features are just the mean, variance, skewness, kurtosis of these coefficients. These features describe the basic distribution of the coefficients, however they are unlikely to capture the strong correlations that exist between neighbourhood pixels (a property exhibited by natural images). Thus a second set of features are calculated based on the error distribution of a linear predictor (specifics are in the paper).

These features are then used to train a LDA/SVM classifier. The SVM model doesn't perform significantly better than the LDA one, while being computationally more expensive.

3 Dataset

As already mentioned, the photo-realistic images used in the original paper were quite old and don't fit well with the current state of the art computer graphics. For e.g. let's compare a still image taken from the paper's data-set and one from a latest AAA game (part of our data-set).

Figure 1: Winning image of irtc 2006

Figure 2: A still image from RE-village 2020

It is clearly visible that the second image is much more detailed as well as looks more realistic than the first one.

We collected photo-realistic images from here. It contains screenshots taken from computer games and the collection is intended for game developers to take inspiration from. For photographic images we used this. It is a data-set of high resolution photographic images, collected by 4 photographers over a period of 3 years.

The authors took about 40000 photographic images and 6000 photo-realistic images. We only took about 2000 photo-realistic and photographic images each. The reason for this was twofold,

1. Having 40000 photographic images and only 6000 photo-realistic would cause class imbalance, a common problem in classification problems.

2. We don't need as many as 40000 images. We are training a LDA classifier, a LDA model doesn't gain much from enormous data input (as opposed to say a neural network). Even having 4000 images is sufficient for a good enough model.

Finally we followed the standard 80/20 split for splitting the data into training/ validation sets.

4 Code

We have implemented the code in python. Using the following python packages,

numpy==1.22.3

opencv-python==4.5.5.64

PyWavelets==1.3.0

scikit-learn==1.0.2

scipy==1.8.0

sklearn==0.0

Following is the general code structure.

1. **train.py**
   This file contains the code to generate a model from the training data. Here we just extract the features from the images, fit a LDA model to it and save the generated model to "models/model.pkl".

2. **test.py**
   This file contains the code to test the model on a single image. When ran on an image, it'll use the model to infer whether the image is photographic or photo-realistic.

3. **val.py**
   This file contains the code to check the accuracy of the model. It runs inferences on the validation data and gives accuracy accordinlgly.

4. **utils.py**
   This file contains various utilities required for the working of the above files. Most important is the function **extractFeaturesFromImg()**. This function returns the previously mentioned features from a given image.

5 Results

We achieved a validation accuracy of **79.07%** on photographic images and **82.32%** on photo-realistic images. The total accuracy came out to be **80.65%**.

6 Drawbacks and Possible Improvements

We could possibly use a CNN instead of a LDA for a better classifier. A CNN can benefit from large amounts of data we have at our disposal. The problem with this as of now is, our code was written in python. The feature extraction code is the bottleneck and takes quite long to run. So training a CNN with a lot of data will take quite long. One way to improvise this is somehow vectorize the for loops in the algorithm to use the fast numpy bindings.

In the implementation we are only using the central 256×256 patch of the image to extract the features of the image. This is done to have a uniform size for all images without changing the aspect ratio of the image. But then we inherently end up losing some information about the image. A possible way to tackle this issue is to divide the image into patches and then train/ make inferences on each patch separately.