PAPER TITLE:

Detection of Deep Network Generated Images Using Disparities in Color Components

Haodong Li, Bin Li, Shunquan Tan, Jiwu Huang

Group Members: CED16I002, CED16I011, CED16I016

Aim:

Given an image, check if its fake or real. By fake we mean, the images are generated by a Deep Network and are called DNG images (Deep Network Generated Images).

- Well known networks are:
  DFC-VAE, DCGAN, WGAN-GP, and PGGAN
  (all in their increasing order of accuracies)
- The author spoke less of PGGAN, as the DNGs generated by the model are very much realistic.

Proposed Method:

The authors mentioned that the DNG images are more differentiable in chrominance components. They have analyzed the disparities between DNG images and real images. By measuring the similarities of DNG images and real images in different color spaces, they have found that the statistical properties of DNG images and real images are different in the chrominance components of HSV and YCbCr. I.e., a fake image has a lot of variation in the chrominance components i.e., H, S in HSV, and Cb, Cr in YCbCr spaces.

This is due to the fact that existing deep networks generate images in RGB color space and have no explicit constraints on color correlations; therefore, DNG images have more obvious differences from real images in other color spaces, such as HSV and YCbCr, especially in the chrominance components.

Besides, the DNG images are different from the real ones when considering red, green, and blue components together. The feature set consists of co-occurrence matrices extracted from the image high-pass filtering(((1 -1)), ((1),(-1))) residuals of several color components.

In order to make the feature dimension compact, binarization or truncation to the residuals is applied, and the elements of co-occurrence matrices are combined based on the symmetric property. The proposed feature set is of low dimension and achieves good detection performance even under the case of a small training set.

Practical implementation:

The order of co-occurrence matrix is set as d = 3 and the offsets as $(\Delta x, \Delta y) \in \{(0, 1), (1, 0)\}$. Therefore, we have 4 co-occurrence matrices (2 residuals × 2 offsets) for each color component, and we finally take the element-wise mean of the 4 co-occurrence matrices as the features. **In total, a 588-D feature is extracted from each image**, where the feature dimension is $(8^3 + 8^2)/2 = 288$ for $R^{\hat{}RGB}$, while the feature dimension is $(5^3 + 5^2)/2 = 75$ for each of $\check{R}^H$, $\check{R}^S$, $\check{R}^{Cb}$, and $\check{R}^{Cr}$.

Detection Strategies:

There are three detection cases. They are-

1) Sample-aware detection:

In this case, the investigator can obtain some DNG images from a known generative model. This is the most simple case for the investigator. To perform the real images and DNG images, and uses the trained classifier to predict the class labels for the given images.

2) Model-aware detection:

In this case, the investigator may know the generative process of DNG images, but he/she does not have any training images generated by the corresponding model and has no idea about the real image dataset that was used to train the model.

To perform the detection, the investigator needs to first train a generative model with the same network
architecture by using an alternative dataset and then use the trained model to produce DNG samples. With these samples, the investigator can train a binary classifier to detect DNG images. It is similar to the case of cross dataset validation in many applications. It is expected that better generalization ability brings better performance.

3) Model-unaware detection:

It is not rare that the investigator does not have any DNG image samples nor have any knowledge about the generative model. **This is the most challenging case presented to the investigator**. To cope with this case, the investigator may train a one-class classifier with real images and use the classifier to detect whether the testing image is real or DNG.

We have chosen strategy 3 and built a model for it.

DATASETS:

CelebFaces Attributes (CelebA):

This dataset consists of more than 200K celebrity face images.

Actual size: 178x218

Pre-processing: we cropped the facial region with 138x138 and resized to 64x64 and 128x128 separately as 2 new input datasets.

Generative models and DNG image datasets:

The authors borrowed the codebases of the authors of DFC-VAE, DCGAN, WGAN-GP to generate the DNG fake datasets for testing the model. The authors of PGGAN have provided their DNG images online, which the author used directly.

Note: The training phase of PGGAN took around 2 weeks for the authors of PGGAN.

Due to memory limitations and no feasible hardware, we were unable to generate any DCGANS but downloaded some of the available ones and DNGs from PGGAN drive

DNG images from PGGAN drive we downloaded = 11000 images

We have also used the sample DNG images provided by DCGAN authors: 128 images

Training Phase:

The author randomly selected 10000 real images as training images and performed the testing on the rest of the real images in and the DNG images created by different generative models. On the one hand, it can be observed that the detection performance for real images is related to the parameter nu. Specifically, when nu equals 0.10 and 0.05, the testing errors for real images are correspondingly close to 10% and 5%, respectively. It means that the trained classifier can adequately model the distribution of images in the real image set.

Note: We too trained the model with 10000 images but tested only on the few DNGs available. We could not handle large data of 2lac images.

We achieved the following results(all values are accuracy measures):

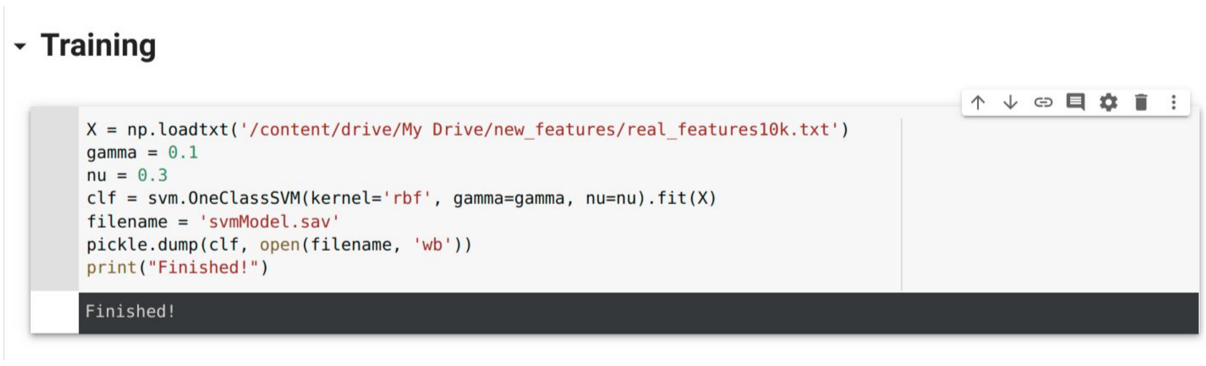
Trained on R_i	Tested on Real (64x64)
M(nu=0.10) R64	0.9106
M(nu=0.05) R64	0.9550

Trained on R_i	Tested on Real (128x128)
M(nu=0.10) R128	0.9086
M(nu=0.05) R128	0.9558

For DCGAN(64x64):

64x64:

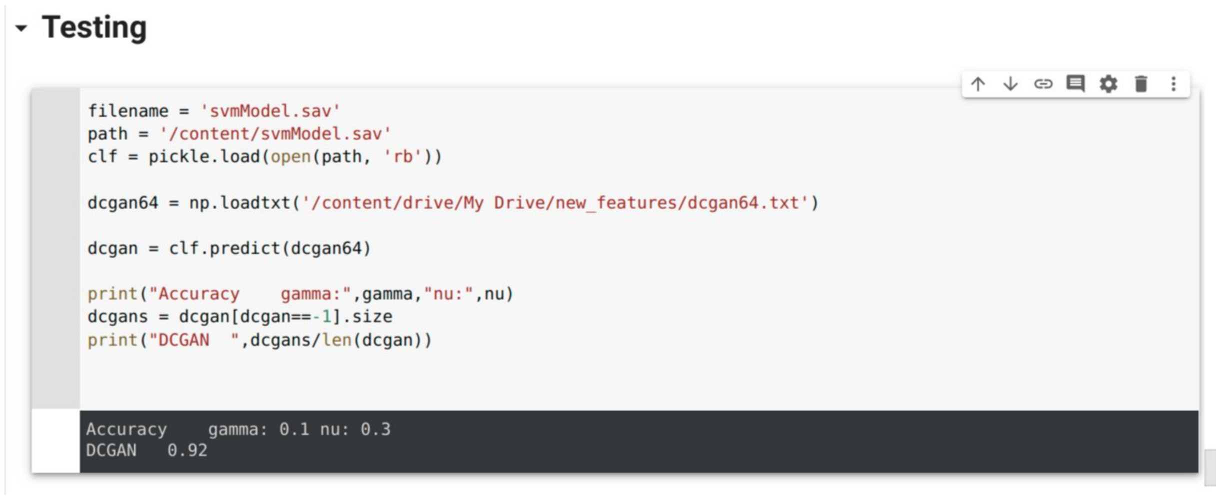
gamma	nu	Accuracy
0.1	0.3	0.92
0.1	0.25	0.88



Training

```
X = np.loadtxt('/content/drive/My Drive/new_features/real_features10k.txt')
gamma = 0.1
nu = 0.3
clf = svm.OneClassSVM(kernel='rbf', gamma=gamma, nu=nu).fit(X)
filename = 'svmModel.sav'
pickle.dump(clf, open(filename, 'wb'))
print("Finished!")
```

```
Finished!
```



Testing

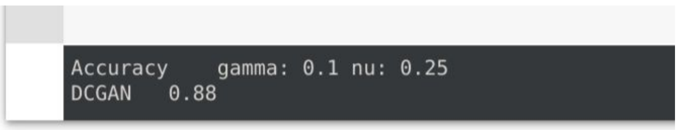
```
filename = 'svmModel.sav'
path = '/content/svmModel.sav'
clf = pickle.load(open(path, 'rb'))

dcgan64 = np.loadtxt('/content/drive/My Drive/new_features/dcgan64.txt')

dcgan = clf.predict(dcgan64)

print("Accuracy    gamma:",gamma,"nu:",nu)
dcgans = dcgan[dcgan==-1].size
print("DCGAN  ",dcgans/len(dcgan))
```

```
Accuracy    gamma: 0.1 nu: 0.3
DCGAN   0.92
```

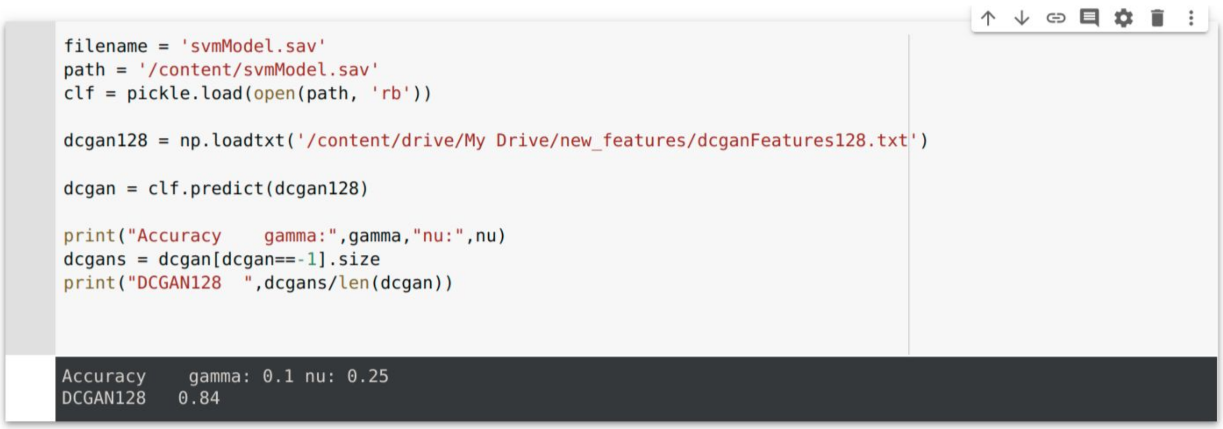


```
Accuracy    gamma: 0.1 nu: 0.25
DCGAN   0.88
```

128X128

gamma	nu	Accuracy
0.1	0.3	0.88

0.1	0.25	0.84



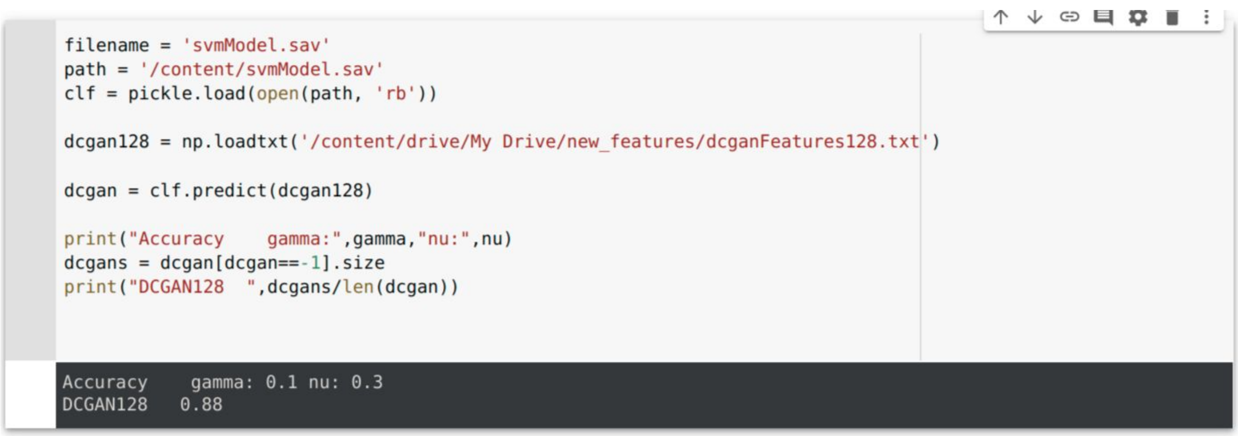
```
filename = 'svmModel.sav'
path = '/content/svmModel.sav'
clf = pickle.load(open(path, 'rb'))

dcgan128 = np.loadtxt('/content/drive/My Drive/new_features/dcganFeatures128.txt')

dcgan = clf.predict(dcgan128)

print("Accuracy    gamma:",gamma,"nu:",nu)
dcgans = dcgan[dcgan==-1].size
print("DCGAN128  ",dcgans/len(dcgan))
```

```
Accuracy    gamma: 0.1 nu: 0.25
DCGAN128   0.84
```



```
filename = 'svmModel.sav'
path = '/content/svmModel.sav'
clf = pickle.load(open(path, 'rb'))

dcgan128 = np.loadtxt('/content/drive/My Drive/new_features/dcganFeatures128.txt')

dcgan = clf.predict(dcgan128)

print("Accuracy    gamma:",gamma,"nu:",nu)
dcgans = dcgan[dcgan==-1].size
print("DCGAN128  ",dcgans/len(dcgan))
```

```
Accuracy    gamma: 0.1 nu: 0.3
DCGAN128   0.88
```

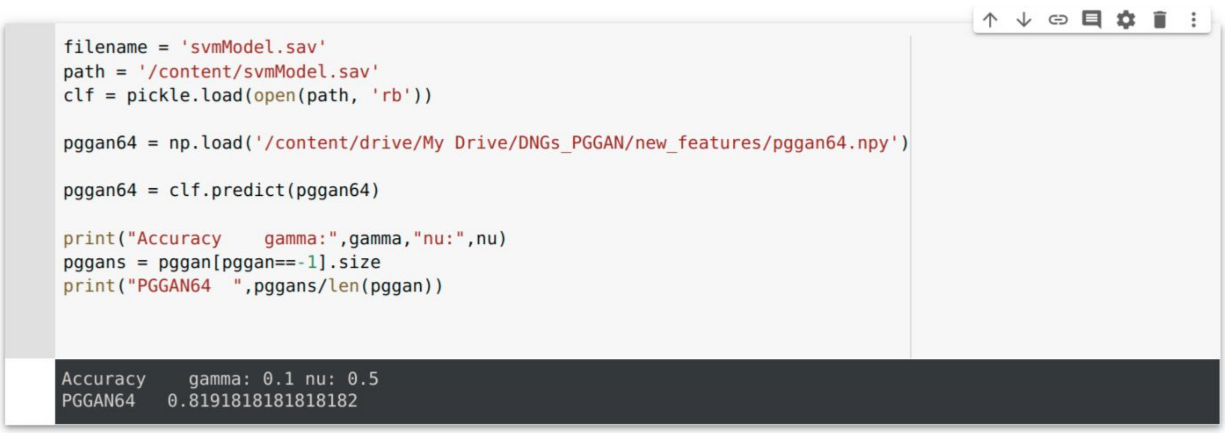
PGGAN(64x64):

We tried downloading 11K DNG images out of 100K from the authors' drive and generated features for them.

The results are as follows:

64x64:

gamma	nu	Accuracy
0.1	0.5	0.8191
0.2	0.4	0.7845



```
filename = 'svmModel.sav'
path = '/content/svmModel.sav'
clf = pickle.load(open(path, 'rb'))

pggan64 = np.load('/content/drive/My Drive/DNGs_PGGAN/new_features/pggan64.npy')

pggan64 = clf.predict(pggan64)

print("Accuracy    gamma:",gamma,"nu:",nu)
pggans = pggan[pggan==-1].size
print("PGGAN64  ",pggans/len(pggan))
```

```
Accuracy    gamma: 0.1 nu: 0.5
PGGAN64   0.8191818181818182
```