EasyDeep: An IoT Friendly Robust Detection Method for GAN Generated Deepfake Images in Social Media

Alakananda Mitra[1] [0000−0002−8796−4819], Saraju P. Mohanty[1] [0000−0003−2959−6541],
Peter Corcoran[2] [0000−0003−1670−4793], and Elias Kougianos[3] [0000−0002−1616−7628]

[1] Department of Computer Science and Engineering, University of North Texas, USA.
alakanandamitra@my.unt.edu, saraju.mohanty@unt.edu
[2] School of Engineering and Informatics, National University of Ireland, Galway, Ireland.
peter.corcoran@nuigalway.ie
[3] Department of Electrical Engineering, University of North Texas, USA.
elias.kougianos@unt.edu

Abstract. Advancements in artificial intelligence, and especially deep learning technology have given birth to a new era of multimedia forgery. Deepfake takes it to a whole new level. This deep learning based technology creates new images with features which have been acquired from a different set of images. The rapid evolution of Generative Adversarial networks (GANs) provides an available route to create deepfakes. They generate highly sophisticated and realistic images through deep learning and implement deepfake using image-to-image translation. We propose a novel, memory-efficient lightweight machine learning based deepfake detection method which is successfully deployed in the IoT platform. A detection API is proposed along with the detection method. To the best of the authors' knowledge, this effort is the first ever for detecting highly sophisticated GAN generated deepfake images at the edge. The novelty of the work is achieving a considerable amount of accuracy with a short training time and inference at the edge device. The total time for sending the image to the edge, detecting and result display through the API is promising. Some discussion is also provided to improve accuracy and to reduce the inference time. A comparative study is also made by performing a three-fold textural analysis - computation of Shannon's entropy, measurement of some of Haralick's texture features (like contrast, dissimilarity, homogeneity, correlation,) and study of the histograms of the generated images. Even when generated fake images look similar to the corresponding real images, the results present clear evidence that they differ significantly from the real images in entropy, contrast, dissimilarity, homogeneity, and correlation.

Keywords: Deepfake · IoT · Edge Computing · Generative Adversarial Networks (GANs) · Image-to-Image Translation · Texture Analysis · Shannon's Entropy · Haralick Texture Feature · CycleGAN · StarGAN

1 Introduction

Recently, there was a lot of excitement generated by Hollywood actor Tom Cruise's TikTok videos. Those videos went viral. They have more than 10M views in March 2021. The technology behind these videos is deepfake. The real Mr. Cruise was not present in those videos. They were not shot with a real camera. The videos were synthetically generated from thousands of his video clips or images by deepfake technology. Features of Mr. Cruise were learned by deep neural networks from those photos and videos and deepfake videos were created through image-to-image translation. It is illegal to impersonate someone in social media. But there is a disparity between the way technology is progressing and the way researchers and companies are trying to combat it. As a result, newly advanced deepfake videos are spreading everyday in social media like Facebook, Twitter, Instagram etc. Generative Adversarial Networks (GANs) create such sophisticated images/videos, that it is hard to detect them. The Defense Advanced Research Projects Agency (DARPA) of the U.S. government is collaborating with various institutions to fight image forgery, especially deepfake [4]. The tech giants Facebook, Google, Microsoft, and Amazon are also combating it.

In today's world, social media play a crucial role in everyday life. Frequently checking social media anywhere, anytime has become a habit. People can connect to the world from a small hand held device easily. On the other side, multimedia forgery has spread immensely. These altered images and videos are often being uploaded in social media. They can defame a person, create political tension or spread rumors [29].

This motivates us to propose a machine learning (ML) based deepfake detection system, which can be deployed in an end device of an IoT setting so that people can check the authenticity of any image anytime, anywhere. This will help to stop circulating misinformation or rumors. The overall system overview of the detection method in an IoT environment is shown in Fig.1.

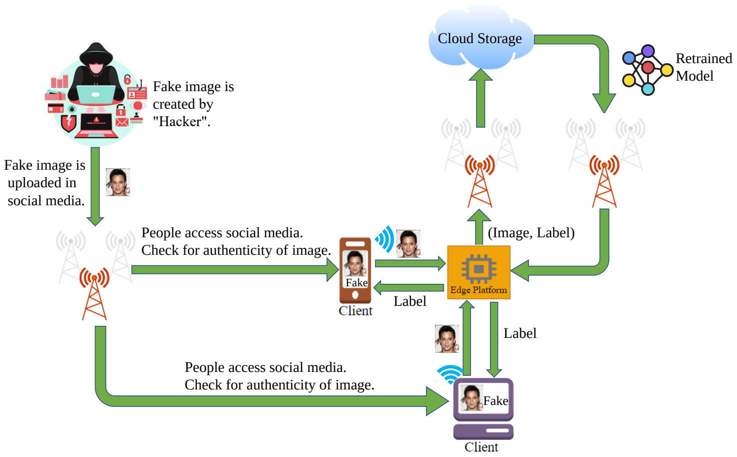

Cloud Storage
Retrained Model
Fake image is created by "Hacker".
Fake image is uploaded in social media.
People access social media. Check for authenticity of image.
Fake
Client
Label
(Image, Label)
Edge Platform
Label
People access social media. Check for authenticity of image.
Fake
Client

Fig. 1. System Overview of the Detection Method.

In general, GANs [9] have a generator and a discriminator as sub models, but they differ in overall structure and working principles. Hence, fake images generated by different GANs will differ from each other. Initially, we investigate fake images generated by two GANs - CycleGAN [41] and StarGAN [7]. As they have different structures and different working principles, as detailed in Table 1, deepfake image/video produced by CycleGAN will differ from that of a StarGAN. We analyze GAN generated images at pixel level and compute several textural features to obtain more information about the deepfake image textures. This textural analysis helps us to propose a memory efficient Machine Learning (ML) based detection method which adapts very well in an IoT environment.

Table 1. Comparison Between CycleGAN And StarGAN

	CycleGAN [41]	**StarGAN** [7]
Network Structure	2 Generators + 2 Discriminators	2 Generators + 1 Discriminator
Number of Domains	2	≥ 2
Loss	Adversarial Loss + Forward Cycle Consistency Loss + Backward Cycle Consistency Loss	Adversarial Loss + Domain Loss + Reconstruction Loss
Data for Training	Unpaired Datasets	Dataset with labeled attributes

The rest of the paper is organized as follows. Section 2 presents challenges and the motivation behind this work. Section 3 focuses on the novel contributions of the paper. Section 4 is a survey of related works in this field. GAN generated images are analyzed in detail in Section 5. Our proposed detection method is described in Section 6. Experimental verification is discussed in Section 7 while Section 8 presents the results. Section 9 draws conclusions and suggests directions for future work.

2 Challenges of GAN Generated Deepfake Images

GANs have improved the quality of the generated images [6, 10, 17, 33]. Presently, GAN approaches have achieved monumental success in creating synthetic images [15–17, 36] and in transferring image styles between different domains [41]. Image-to-image translation can be used in changing seasons in a photo, photo enhancement, object transfiguration, etc. [41]. But, these applications can also be used in negative ways. It is difficult for people to distinguish between a GAN generated deepfake image and a real image with bare eyes. These fake images spread misinformation through social media or news channels. Faking someone's identity ("deepfake") in social media could have a socio-political impact along with financial and security hazards.

For more than a century, audio-visual media have presented the truth, recording the time and history. But fake images, videos and audios change the perception of truth or reality. Thus it is important to look into the threats posed by GAN-generated deepfake images or videos.

3 Novel Contributions of the Current Paper

This paper proposes a ML based technique of detecting GAN generated deepfake images, implementable at an edge device. The novelties of this work are:

- To the best of our knowledge, this work is the first ever effort to detect GAN generated deepfake images at an edge device.
- As edge devices are of limited resources (memory, architecture, storage etc.), high accuracy, heavy computing models can not be deployed. We propose a novel ML based technique which detects deepfake images at an edge device using texture analysis. This is done with lighter computational load. Training time is much shorter and we achieve a considerate amount of accuracy.
- A detection API is proposed to make the overall process automated. Fig. 2 shows the overall Detection API diagram.
- We also suggest different ways of achieving higher accuracy.
- Some discussion is provided on strategies to improve inference time.

- Various textural features like Shannon's entropy, and Haralick's texture features of GAN generated deepfake images have been explored to understand the textural difference between generated and real images. It helps us to propose a memory efficient detection method.
- A comparative study on histograms between StarGAN generated images and corresponding real images has been performed to understand the effect of manipulation on color.

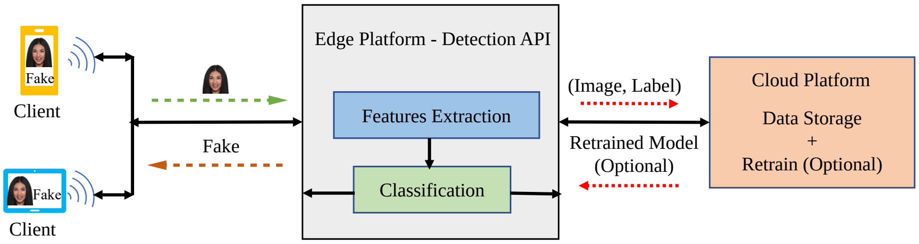


Fig. 2. Detection Method API Diagram.

4 Related Prior Works

In the last decade, due to the availability of GPUs, the research on GAN generated images has received a huge boost. Various areas of image manipulation such as high quality GAN generated images [5,10,17,20,27,31,33,38], image-to-image translation [7,14,39,42], face completion [13,21], various facial expression and attributes [8,23], domain transfer [19,34], and style transfer [15,16] have received the attention of the computer vision community.

Researchers in image forensics and computer vision have been working to develop methods in detecting those GAN generated fake images. In our previous work [28,29] we detected mostly auto-encoder generated social media deepfake videos. An ensemble deep learning technique via a Random Forest classifier has been proposed in [12]. Three shallow CNN structures have been used to extract features from the images in YCbCr, HSV, and Lab color spaces. Two fully connected (FC) networks with 2048 and 1024 nodes increase the total number of trainable parameters. Parallel processing of the same image in three different color spaces makes the process resource intensive. No IoT implementation effort has been made here. Fake faces have been detected in [37] using a shallow neural network as a classifier and neuron activation has been monitored using deep neural network. The model has been evaluated for four perturbation attacks. This model is also resource intensive. It is not implemented in IoT settings. Gram blocks have been added in ResNet structure in detecting fake faces in [22]. This is not an IoT friendly network either, due to its heavyweight structure. Gray level co-occurrence matrices (GLCM) have been calculated separately on RGB channels and used as the inputs of a DNN structure [30]. GLCM calculation over three channels for an image makes it memory intensive. No effort has been made to deploy it at edge either. Both GAN generated and man made fake images have been detected in a DNN based ensemble network [35].

How a GAN generated image is different from a camera shot image from a color cue perspective, has been investigated in [26]. GAN fingerprints on image attribution have been noted in [40]. Some of the textural properties of fake images generated by StyleGAN and

PGGAN have been explored in [24] along with their detection network. All these works claim to have high accuracy, but no effort has been made to deploy them in IoT environments.

5 Analysis of GAN Generated Deepfake Images

In this section, we explore several first and second order statistical texture features of GAN generated fake images, before detecting them in Section 6. A comparative study has also been performed between the generated deepfake images and corresponding real images.

When a photo is shot in a digital camera, light from the object gets passed through the lens and falls on the CMOS sensors which breaks the image into pixels after measuring the light intensity and brightness. When a real image is passed through a GAN generator, it transforms the image into a latent space vector. After a zero sum game with the discriminator it generates a fake image. Hence, the way a digital camera takes a photo differs from the process of generation of fake images by GANs. This motivates us to explore the textural features of GAN generated images. We tested CycleGAN and StarGAN generated images.

5.1 Entropy Computation

In information theory, Shannon's Entropy measures the uncertainty of a random variable's possible outcomes. For an image, it is more related to the texture or image information. The difference in entropy between a generated image and its corresponding real image gives some information on texture difference between them. Calculating the entropy of each pixel in an image and summing over all possible gray scale pixels gives a scalar value E of the entropy of that image according to Eq. 1, and makes it easily comparable:

$$E = -\sum_{i=0}^{n-1} p_i \log_b p_i, \tag{1}$$

where n denotes the number of gray levels, p_i is the pixel probability for gray level i, and base b is the base.

To calculate the entropy of an gray scale image E_{gray}, the space of any *RGB* image is first changed to *Gray* scale. With same radiance of *red (R)*, *green (G)*, and *blue (B)* colors, green always looks brightest among those three because the luminous efficiency tops at the green zone of the visible light spectrum, red looks less bright and blue is the darkest. So when the luminance (*Y*) of an *RGB* image is expressed as a function of *R*, *G*, and *B*, different weights are applied in them to represent the color perception of real life as in Eq.2 [3]. During conversion of *RGB* image to *Gray* scale image, Eq.2 is considered as the luminance of the image.

$$Y = 0.2125R + 0.7154G + 0.0721B \tag{2}$$

To compare the entropies of generated and real images, the process described in Algorithm 1 has been used.

5.2 Haralick's Texture Features Analysis

Texture is a spatial property of an image. Computing the Gray-Level Co-Occurrence Matrix (GLCM) of an image is a well known method to capture the spatial dependence of gray level values. It calculates the likelihood of a pixel value and its relationship with other pixels [2].

Algorithm 1 Process of Comparing Entropy of Generated and Real Images for Section 5.

```
**Input:** Fake Image $I1$ and Real Image $I2$
**Output:**$min$, $max$, $mean$, and $standard\ deviation$ of entropy difference
SAVE the real images with name ending real and fake images with ending word fake
SET input image directory $input\_dir$
DECLARE two Dictionaries $ent\_fake\_dict$ and $ent\_real\_dict$ with keys $filename$ and $entropy$ for fake and real images respectively
DECLARE a variable $comp\_entropy$ and initialize it to 0
DECLARE a list $comp\_entropy\_list$ for storing the entropy difference and initialize it to $comp\_entropy\_list \leftarrow NIL$
ASSIGN a particular fake image to $index\_f$ and a real image to $index\_r$
**for** $file \in input\_dir$ **do**
    Entropy is calculated.
    **if** $file$ contains the word real **then**
        STORE the entropy in $ent\_real\_dict$ along with $filename$
    **else**
        STORE the entropy in $ent\_fake\_dict$ along with $filename$
    **end if**
**end for**
**for** $index\_r \in ent\_real\_dict$ **do**
    **for** $index\_f \in ent\_fake\_dict$ **do**
        GET the fake image ent_fake_dict[$filename$] corresponding to the real image ent_real_dict[$filename$].
        COMPUTE $comp\_entropy$ by taking the difference between entropies of the
         corresponding fake and real images
        STORE $comp\_entropy$ in $comp\_entropy\_list$
    **end for**
**end for**
COMPUTE $min$, $max$, $mean$, and $standard\ deviation$ of $comp\_entropy\_list$
```

To explore the fake images more closely, we calculate several Haralick's Texture Features [11] from the GLCM. The features we calculate are contrast, correlation, homogeneity, and dissimilarity.

The GLCM is defined as a square matrix with elements as the frequency of occurrence of pixels with gray levels at a certain distance and angle. Contrast, homogeneity, dissimilarity, and correlation are defined in Eq. 3, 4, 5, and 6 respectively.

$$CON = \sum_{i,j=0}^{n-1} p(i,j)(i-j)^2 \tag{3}$$

$$HOM = \sum_{i,j=0}^{n-1} \frac{p(i,j)}{(1+(i-j)^2)} \tag{4}$$

$$DIS = \sum_{i,j=0}^{n-1} p(i,j)|i-j| \tag{5}$$

$$COR = \sum_{i,j=0}^{n-1} p(i,j) \left[\frac{(i-\mu_i)(j-\mu_j)}{\sqrt{(\sigma_i^2)(\sigma_j^2)}} \right] \tag{6}$$

In the above expressions, n denotes number of gray levels, $p(i,j)$ is the element of GLCM for the distance between gray level values i and j, and μ and σ are the mean and variance of the intensities of all gray values present, respectively.

We have followed the process in Algorithm 2 for comparing contrast, dissimilarity, homogeneity and correlation among real and generated images.

Algorithm 2 Process of Comparing Contrast, Dissimilarity, Homogeneity, and Correlation of Generated and Real Images for Section 5.

```
Input: Fake Image $I1$ and Real Image $I2$
Output:$min$, $max$, $mean$, and $standard\ deviation$ of GLCM properties difference
SAVE real images with name ending real and fake images with fake.
SET input image directory $input\_dir$
DECLARE two Dictionaries $glcm\_prop\_fake\_dict$ and $glcm\_prop\_real\_dict$ with keys $filename$, $contrast$, $dissimilarity$, $homogeneity$, and $correlation$ for fake and real images, respectively.
DECLARE a list $comp\_glcm\_prop$ for storing the GLCM properties and initialize it to $comp\_glcm\_prop \leftarrow NIL$.
DECLARE another list $comp\_glcm\_prop\_list$ for storing the GLCM properties differences and initialize it to $comp\_glcm\_prop\_list \leftarrow NIL$.
ASSIGN a particular fake image to $index\_f$ and a real image to $index\_r$.
for $file \in input\_dir$ do
    GLCM is calculated.
    $contrast$, $dissimilarity$, $homogeneity$, and $correlation$ are calculated.
    if $file$ contains the word real then
        STORE $contrast$, $dissimilarity$, $homogeneity$, and $correlation$ in $glcm\_prop\_real\_dict$ along with $filename$.
    else
        STORE $contrast$, $dissimilarity$, $homogeneity$, and $correlation$ in $glcm\_prop\_fake\_dict$ along with $filename$.
    end if
end for
for $index\_r \in glcm\_prop\_real\_dict$ do
    for $index\_f \in glcm\_prop\_fake\_dict$ do
        GET $glcm\_prop\_fake\_dict[filename]$ corresponding to $glcm\_prop\_real\_dict[filename]$.
        COMPUTE $comp\_glcm\_prop$ by taking the difference between contrast, dissimilarity, homogeneity, and correlation of the corresponding fake and real images.
        STORE $comp\_glcm\_prop$ in $comp\_glcm\_prop\_list$
    end for
end for
COMPUTE $min$, $max$, $mean$, and $standard\ deviation$ of $comp\_glcm\_prop\_list$.
```

5.3 Histogram Analysis

We explore the histograms of the generated images and compare them with those of real images. Histograms for the R, G, and B channels are plotted separately. They show a comparative graphical representation of the distribution of intensity of a generated image and corresponding real image for each channel.

6 The Proposed Novel Deepfake Detection Method at Edge Computing Platform

In this section, we propose a novel machine learning based model for detecting GAN generated images at an edge device using textural features of images. The process is performed automatically through our proposed detection API. The reasons behind this approach are multi-fold:

- The computation cost is very low, therefore it is a good fit for less resource IoT settings.
- As the the approach is based on gray level co-occurrence at pixel locations, it is generic. It can be applied to any type of GAN.

6.1 System Level Representation

The system level overview of the detection method is shown in Fig. 1. In this IoT environment, end users or clients are connected to the edge device. The edge device is connected to the cloud for data storage. Retraining of the model can be done at the cloud with the stored images and corresponding predictions at a later stage, if needed. The detection process is initiated when an image from social media (uploaded by any person of bad intent [1]), to be detected, is sent to the edge platform through our proposed 'Detection API'. Once the image is detected by the model, the detection score goes back to its source. The detection score is also stored in the cloud along with the image to be used for future retraining of the model. Fig. 2 shows a detailed representation of the concept when a client sends the image to the edge device.

6.2 Detection Methodology at the Edge Platform

Fig. 3 shows the overall diagram of the detection model at edge. It is an automatic workflow. When an user wants to check any picture from social media accounts for authenticity, he calls the Detection API and sends the image to the edge device. Once the image reaches the edge device it is saved in a folder and the GLCM followed by Haralick's texture features are calculated from the corresponding gray level image of the colored image.

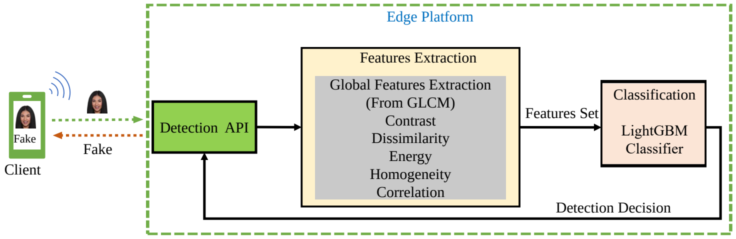


Fig. 3. Overall Workflow Diagram at Edge Platform.

A features set is created from those features. We compute five Haralick's texture features. The texture features calculated at this stage are contrast, dissimilarity, homogeneity, energy, and correlation. We calculate those features from the GLCM for four distances at $d = 1,2,3,5$ and three angles $\theta = 0, \pi/4, \pi/2$ to generate the feature vector. After the feature extraction, a machine learning algorithm is used to classify the image. As in IoT environment, the resources are limited, we carefully choose our classifier as LightGBM with boosting type 'Gradient Boosting Decision Tree (gbdt)'. It is a tree based algorithm. The advantage of using this classifier over others are:

- The algorithm [18] uses histograms to learn. It is cost effective because for a histogram based algorithm the time complexity is proportional to the number of bins, not to the data volume once histograms are made.
- Use of discrete bins reduces the memory usage which is a limiting factor at an edge device.
- Training is very fast as it is distributed.

6.3 Phases of Detection Method

The overall detection process has two phases: Training and Testing or Inferring.

- *Training Phase:* In the training phase, the classifier learns how to detect the fake images. Initial training has been done on a PC. The details are mentioned in Section 7. At a later stage, if retraining is needed, it can be done in the cloud.
- *Testing Phase:* Testing phase is where the unknown samples are tested. This is implemented at the edge device. The testing phase is performed through our proposed API.

6.4 Detection API

We propose a Detection API hosted at the edge device. The workflow of the API is shown in Fig. 4. The goal is to make the API lighter and faster to work at an edge device.

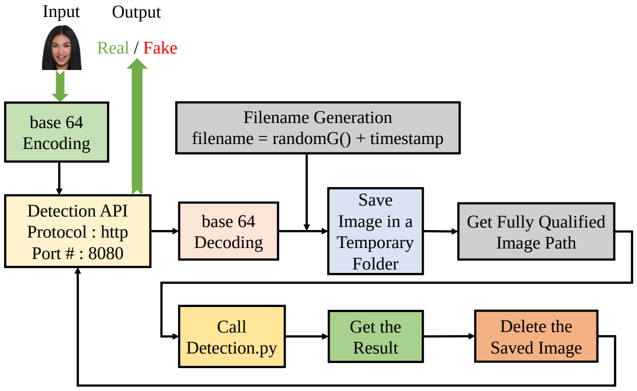

Input
Output
Real / Fake
base 64 Encoding
Detection API Protocol : http Port # : 8080
Filename Generation filename = randomG() + timestamp
base 64 Decoding
Save Image in a Temporary Folder
Get Fully Qualified Image Path
Call Detection.py
Get the Result
Delete the Saved Image

Fig. 4. Detection API Workflow.

6.5 Detection Metrics

To visualize the classification performance of our detection model, we define the *Confusion Matrix (CM)* as in Table 2 [29]. Detecting real or GAN generated fake images is a binary classification problem, the corresponding *CM* is a 2×2 matrix. To evaluate the detection model, various metrics have been computed from *CM* according to the following expressions:

$$Accuracy = \left(\frac{TP{+}TN}{TP{+}TN{+}FP{+}FN}\right)\times 100\% \tag{7}$$

$$Precision = \left(\frac{TP}{TP{+}FP}\right)\times 100\% \tag{8}$$

$$Recall = \left(\frac{TP}{TP{+}FN}\right)\times 100\% \tag{9}$$

$$F1{-}score = \left(\frac{2}{\frac{1}{Precision}+\frac{1}{Recall}}\right)\times 100\% \tag{10}$$

Table 2. Confusion Matrix

	Predicted Label	
True Label	True Positive (**TP**): Reality : **Fake** Model predicted : **Fake**	False Negative (**FN**): Reality : **Fake** Model predicted : **Real**
	False Positive (**FP**): Reality : **Real** Model predicted : **Fake**	True Negative (**TN**): Reality : **Real** Model predicted : **Real**

7 Experiments

7.1 Datasets

StarGAN and CycleGAN datasets have been chosen to compare the fake images generated by different GANs. The CycleGAN dataset consists of images which are generated by translating from one image domain to another image domain, whereas StarGAN generates multi-domain images on the fly by changing physical attributes and with different expressions.

StarGAN Dataset: To generate this dataset we follow the process as in [7]. Five different physical attributes, such as different hair color (black, blond, brown), gender, and age have been chosen. No expression or mood change has been done. To generate the images by StarGAN, the first 6,000 images from CelebA [25] dataset have been chosen. Each real image generates five images, so a total of 30,000 images are generated. 500 fake images along with corresponding 100 real images have been tested to compare textural properties of fake images with those of real images. For the detection model a total of 60,000 images (30,000 fake and 30,000 real) have been used for training and validation. Up-sampling of the minority class has been done to provide a balanced dataset. Fig. 5 shows some sample StarGAN generated images used in the experiment.

CycleGAN Dataset: To make this dataset, the GitHub page of the original paper has been followed [14, 41]. Real images for CycleGAN have been collected from ImageNet, Wikiart, and CMP Facades dataset [32], as suggested by the original paper. A total of 9,809 images have been generated from 9,812 real images. Table 3 presents the detailed list of the dataset generated. Some of the generated CycleGAN images are shown in Fig. 6.

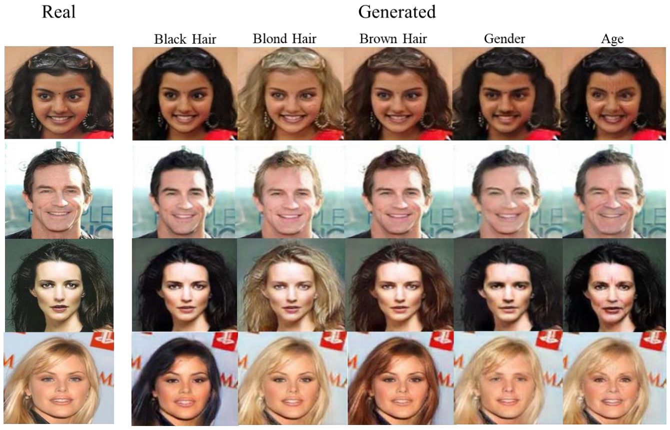


Fig. 5. Sample StarGAN Generated Images.

Table 3. Generated CycleGAN Dataset

	apple2orange	**horse2zebra**	**monet2photo**	**vangogh**	**ukiyoe**	**cezanne**	**facades**
Real	2237	2349	671	1738	194	295	343
Generated	2239	2348	672	1736	193	294	342

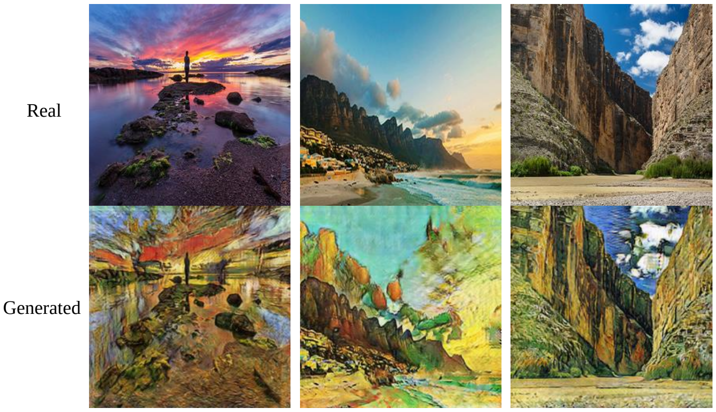


Fig. 6. Sample CycleGAN Generated Images.

7.2 GAN Generated Image Analysis

Once the datasets are generated, we evaluate various texture statistics for those GAN generated fake images and their corresponding real images. We perform three different experiments with the images.

In the first part, we compute the entropy of the fake images generated by both GANs and compare the result with corresponding real images. Minimum, maximum, mean and standard deviation of the entropy difference between the fake images and real images are also calculated.

We evaluate 2,239 “apple2orange”, 2,348 “horse2zebra”, 672 “monet2photo”, 1,736 “vangogh”, 193 “ukiyoe”, 294 “cezzane”, 342 “facades”, and 500 StarGAN images. This provides a comparative idea of fakeness of the two sets of fake images.

For the same set of images, GLCM is calculated. Then four Haralick’s texture features - contrast, homogeneity, dissimilarity and correlation - are computed and compared with those of corresponding real images. To provide an inter-GAN comparison, the same statistical parameters of the dataset are calculated for both GANs.

Histograms for the R, G, and B channels are observed separately for generated and real images. Histograms of generated image (which has same feature as the real image) are compared with the corresponding histograms of real image e.g., if the sample input image is a black haired young male, comparison is made only when the generated image has the same features (‘black haired young male’). For CycleGAN this part is skipped, as the generated image is different than the real image. This has been implemented in Python.

7.3 Implementation of Proposed Detection Method of GAN Generated Images at an Edge Computing Platform

Single Board Computer Platform: The detection model has been implemented on a 4GB Raspberry pi 4, a single board computer. The input image has been provided through the proposed Detection API and the detection result has been given back through the API.

Initially, the training has been done on a PC with 16GB total memory and Intel Core i7-9750 processor. No GPU was used. $48{,}000$ images ($24{,}000$ deepfake images generated by Star GAN and $24{,}000$ real images) have been utilized for training and $12{,}000$ images have been used for validation of the model. Total time for training and validation of the model was 27 minutes. Before constructing the features set, the image is converted to gray level and then resized to 256×256. The features set is constructed from Haralick’s texture features. The feature set is of size $48{,}000 \times 30$ for the training data. The learning rate of the classifier is kept at 0.05, the maximum depth of 600 trees is 13, and the number of leaves of each tree is kept at 8,500. The boosting algorithm is chosen to be ‘Gradient Boosting Decision Tree’. The detection part has been implemented in Python and the API part in Java.

8 Results

8.1 Analysis of GAN Generated Images

In this section, the observations after analyzing and comparing two GAN generated images with real images are reported in detail.

Entropy: The entropy difference of the generated image and its corresponding real image for both GANs is shown in Table 5. The abbreviated terms of Table 5 are explained in Table 4.

The entropy difference has been computed by taking the difference of generated images from its corresponding real images. Some results give positive entropy difference and some results negative.

- Negative entropy difference means that the entropy of the generated image is greater than its corresponding real image and is denoted by $\varDelta E_2$ in Table 5.
- Positive entropy difference means that the entropy of the generated image is lower than its corresponding real image and is denoted by $\varDelta E_1$ in Table 5.

Table 4. Definition of Abbreviated Terms Of Table 5

Term	**Meaning**
ΔE	Entropy difference of real image and generated image $E_R - E_F$
ΔE_1	ΔE when $E_F < E_R$
ΔE_2	ΔE when $E_F > E_R$
ΔE_{mean}	Mean Entropy Difference of Test Data Distribution
ΔE_{std}	Standard Deviation of Entropy Difference of Test Data Distribution

Table 5. Entropy Differences between Real Images and GAN Generated Fake Images

GAN	**Data**	$\mathbf{\Delta E_1}$	$\mathbf{\Delta E_2}$	$\mathbf{\Delta E_{mean}}$	$\mathbf{\Delta E_{std}}$
Cycle GAN	apple2orange	1.8177	-4.6379	0.3965	0.7221
	horse2zebra	0.7999	-2.3821	0.0514	0.2862
	monet	0.5779	-0.83646	0.01031	0.2428
	vangogh	0.5779	-2.3413	0.1046	0.3519
	ukiyoe	0.4335	-0.8343	0.0217	0.2016
	facades	1.448	-3.4454	0.4901	1.3729
	cezanne	0.6253	-1.6657	-0.0171	0.3158
StarGAN		0.4579	-1.3998	-0.1062	0.2310

We tested entropy difference over a certain number of images. Table 5 shows the trend of entropy distribution of a particular GAN generated fake dataset, but changing the images will definitely change the values of maximum, minimum, mean, and standard deviation of entropy difference. StarGAN generates images with lower $|\Delta E|$ on average. The standard deviation of entropy difference in Table 5 also follows the same trend. Other than the ukiyoe dataset, all cases of CycleGAN have larger standard deviation of entropy difference than StarGAN. It means that CycleGAN generates a wide variety of fake images with greater ΔE than StarGAN.

Entropy of an image is the randomness around the pixels of an image. It gives certain information on image texture. Lower $|\Delta E|$ of StarGAN generated images prove that the texture of fake images varies lesser in StarGAN generated images than CycleGAN generated images. StarGAN generates more robust (less varied entropy than real images) fake images than CycleGAN.

Haralick's Texture Features: Table 6 and Table 7 show the differences of texture features between generated image and the corresponding real image. The average difference of contrast, dissimilarity, correlation, and homogeneity are much larger in CycleGAN than StarGAN because CycleGAN generated images are very different from the original images. But StarGAN generated images have varied texture features than real images too. Therefore, we choose these textural features along with energy (inversely proportional to entropy) in forming the features set for detecting fake images.

Histogram: Third, we observe histograms for the StarGAN generated images. They are shown in Figs. 7(a), 7(b), and 7(c). In each case, histograms for red, green, and blue channels of the generated images differ from those of the real images with the same attributes. It means that even if the generated image looks the same as the real image with bare eyes, the color distributions are not the same for those two images.

So, GAN generated images vary in textures and colors from the real images. This observation has been utilized in detecting GAN generated images.

Table 6. Difference of Texture Properties of GAN Generated Images and Real Images - I

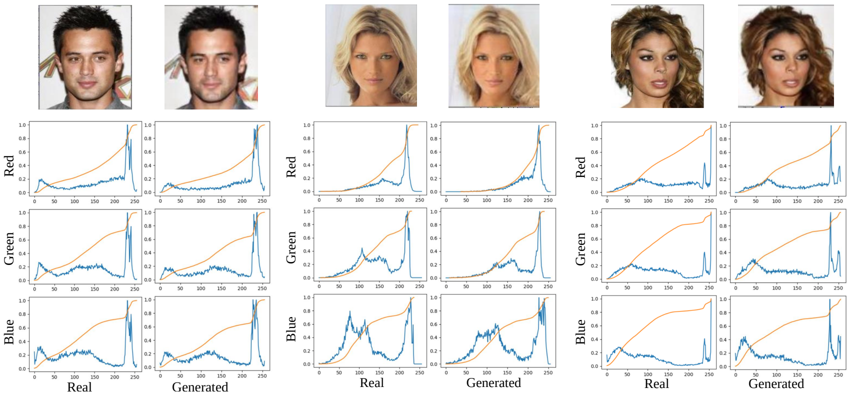
GAN	**Data**	**Contrast**		**Dissimilarity**		**Homogeneity**		**Correlation**	
		Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}
Cycle GAN	apple2orange	0.1759	4067.51	0.0025	19.2035	0.6417	4.0771	0.6303	3.3104
	horse2zebra	0.0089	16649.09	0.0002	65.8922	0.00001	0.3731	0.8593	6.9169
	monet	0.9867	2559.67	0.0026	18.8634	0.00001	0.4919	0.0004	0.3023
	vangogh	0.6193	2532.19	0.0025	27.3638	0.0002	0.6615	0.0004	0.6378
	ukiyoe	1.2918	2343.93	0.03824	23.4140	0.00005	0.4478	0.0003	0.4959
	facades	1.2550	1982.35	0.0061	21.2044	0.0014	0.6379	0.00005	0.3006
	cezanne	1.0449	1964.87	0.0071	19.0935	0.00003	0.4021	0.0016	0.4070
	Average	**0.7689**	**4585.66**	**0.0085**	**27.8621**	**0.0919**	**1.0131**	**0.2132**	**1.7673**
StarGAN		0.0003	3175.85	0.0026	24.2543	0.0001	0.4029	0.0001	0.3212

$\Delta \rightarrow$(Difference of a texture property)

Table 7. Difference of Texture Properties of GAN Generated Images and Real Images-II

GAN	**Data**	**Contrast**		**Dissimilarity**		**Homogeneity**		**Correlation**	
		Δ_{mean}	Δ_{std}	Δ_{mean}	Δ_{std}	Δ_{mean}	Δ_{std}	Δ_{mean}	Δ_{std}
Cycle GAN	apple2orange	353.3008	339.2468	3.6820	3.0643	0.0964	0.0108	0.0547	0.0522
	horse2zebra	1241.3559	1579.4719	8.7607	8.4865	0.0364	0.0482	0.0001	0.1554
	monet	259.5098	297.9325	2.9529	2.4031	0.0553	0.0605	0.0455	0.0401
	vangogh	663.8957	433.3669	10.1373	5.8673	0.1118	0.0958	0.1661	0.1022
	ukiyoe	935.4261	510.1807	9.7288	4.7253	0.0787	0.0849	0.1719	0.0871
	facades	479.9753	447.9612	6.2992	4.1170	0.2592	0.1790	0.0609	0.0524
	cezanne	542.8389	340.8045	7.1889	4.1682	0.0918	0.0806	0.1110	0.0781
	Average	**639.4718**	**513.8523**	**6.9642**	**4.6902**	**0.1042**	**0.0800**	**0.0872**	**0.0811**
StarGAN		813.0842	679.7914	6.8765	4.9921	0.0729	0.0663	0.0787	0.0579

$\Delta \rightarrow$(Difference of a texture property)

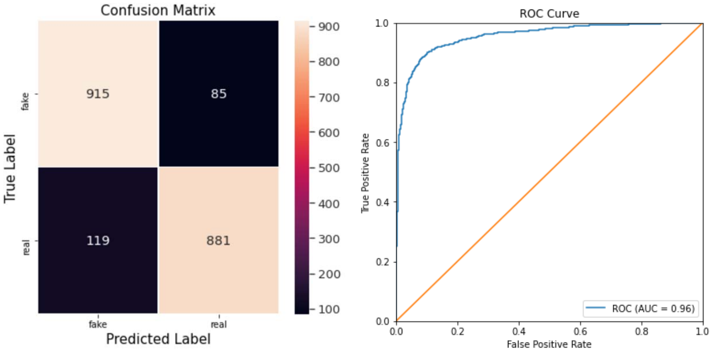


(a) Black Haired Male (b) Blond Haired Female (c) Brown Haired Female

Fig. 7. Histogram Comparison of StarGAN Generated Images and Real Images.

8.2 Evaluation of Edge Detection Model

The performance of the model has been evaluated over 2,000 total test images. 50% of the images are fake and the rest are real. The *Confusion Matrix* is generated from these images, as shown in Fig. 8(a). We calculate *Precision*, *Recall*, and *F1-score* from *CM* using Eqs. 8, 9, and 10. Table 8 shows the classification report of the model.



(a) Confusion Matrix.

(b) ROC Curve

Fig. 8. Histogram Comparison of StarGAN Generated Images and Real Image.

Table 8. Classification Report of Test Images.

Test Images	**Precision**%	**Recall**%	**F1-score**%
1000 Fake	88.0	92.0	90.0
1000 Real	91.0	88.0	90.0
Macro Average	90.0	90.0	90.0
Weighted Average	90.0	90.0	90.0
Total 2000	Accuracy %	90.0	
Total 2000	**AUC Score** %	96.0	

To evaluate the model more accurately, we draw the *Receiver Operating Characteristic (ROC)* curve, as shown in Fig. 8(b). *AUC score* for the model is 96%.

Table 9 shows the relation between the accuracy, tree structure, boosting type algorithm and size of the model. We vary the parameters to have a model which can be deployed at the Raspberry pi and would also have high accuracy. The final structure is chosen with an accuracy of 90%, 600 trees, and each tree with a maximum depth of 13 and number of leaves 8,500. Training time for our model is 27 minutes. Accuracy is improved by increasing the number of trees in the algorithm. Maximum tree depth and number of leaves also influence the accuracy.

Table 9. Accuracy Variation with Tree Structure

Number of Trees	Max Tree Depth	Number of Leaves	Algorithm Boosting	Accuracy %	Model Size (MB)
100	8	255	dart*	79.4	3.2
100	10	1000	dart	80.4	6.7
100	11	2500	dart	81.8	12.4
100	12	4200	dart	82.1	15.8
100	13	8500	dart	82.9	19.0
100	14	17000	dart	82.7	22.2
100	13	8500	gbdt*	85.5	14.3
100	14	17000	gbdt	85.9	16.4
200	13	8500	gbdt	87.4	21.5
300	13	8500	gbdt	88.2	27.3
400	13	8500	gbdt	89.0	32.8
600	13	8500	gbdt	90.0	43.7

dart* (Dropouts meet Multiple Additive Regression Trees)
gbdt* (Gradient Boosting Decision Tree)

9 Conclusion and Future Work

It is challenging to implement a computation-intensive computer vision problem like deepfake detection in an IoT environment. We tried to keep the computation as light as possible. We chose only 30 features for each image so that we could infer at a limited resource IoT device with considerable accuracy. Accuracy can be improved by increasing the number of trees and also by changing the feature set. With more features, the accuracy will be higher and the generalization of the model will be achieved. To improve the inference time instead of sending the image in base64 format, binary image can be sent. As a future work, generalization of the model and higher accuracy can be achieved along with improved inference time.

References

1. image ”hacker”. image: freepik.com, Accessed on 07 June 2021
2. Mathworks. https://www.mathworks.com/help/images/texture-analysis-using-the-gray-level-co-occurrence-matrix-glcm.html, Accessed 28 January 2021
3. scikit-image. http://poynton.ca/PDFs/ColorFAQ.pdf, Accessed 02 Febrauary 2021
4. DARPA News. https://www.darpa.mil/news-events/2021-03-02 (Mar 2021), Accessed 07 May 2021
5. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. arXiv: 1701.07875 (2017)
6. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv abs/1809.11096 (2018)
7. Choi, Y., Choi, M., Kim, M., Ha, J., Kim, S., Choo, J.: Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8789–8797 (2018)
8. Dogan, Y., Keles, H.Y.: Semi-supervised image attribute editing using generative adversarial networks. Neurocomputing 401 pp. 338 – 352 (2020)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K.Q. (eds.) Proceedings of Advances in Neural Information Processing Systems. vol. 27, pp. 2672–2680. Curran Associates, Inc. (2014)
10. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of Wasserstein GANs. In: Proceedings of Advances in Neural Information Processing Systems. p. 5767–5777 (2017)

11. Haralick, R.M., Shanmugam, K., Dinstein, I.: Textural features for image classification. IEEE Transactions on Systems, Man, and Cybernetics **SMC-3**(6), 610–621 (1973)
12. He, P., Li, H., Wang, H.: Detection of fake images via the ensemble of deep representations from multi color spaces. In: Proceedings of IEEE International Conference on Image Processing. pp. 2299–2303 (2019)
13. Iizuka, S., Simo-Serra, E., Ishikawa, H.: Globally and locally consistent image completion. ACM Transanctions on Graph. **36**(4) (Jul 2017)
14. Isola, P., Zhu, J., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. p. 1125–1134 (2017)
15. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 4396–4405 (2019)
16. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of stylegan. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 8107–8116 (2020)
17. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv: abs/1710.10196 (2017)
18. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.: Lightgbm: A highly efficient gradient boosting decision tree. In: Proceedings of Advances in Neural Information Processing Systems (2017)
19. Kim, T., Cha, M., Kim, H., Lee, J.K., Kim, J.: Learning to discover cross-domain relations with generative adversarial networks. arXiv: abs/1703.05192 (2017)
20. Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A.P., Tejani, A., Totz, J., Wang, Z., Shi, W.: Photo-realistic single image super-resolution using a generative adversarial network. arXiv: abs/1609.04802 (2016)
21. Li, Y., Liu, S., Yang, J., Yang, M.: Generative face completion. arXiv: abs/1704.05838 (2017)
22. Liu, M., Breuel, T., Kautz, J.: Unsupervised image-to-image translation networks. arXiv:abs/1703.00848 (2017)
23. Liu, M., Tuzel, O.: Coupled generative adversarial networks. arXiv: abs/1606.07536 (2016)
24. Liu, Z., Qi, X., Torr, P.H.S.: Global texture enhancement for fake face detection in the wild. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 8057–8066 (2020)
25. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of International Conference on Computer Vision (December 2015)
26. McCloskey, S., Albright, M.: Detecting gan-generated imagery using color cues. arXiv:abs/1812.08247 (2018)
27. Mirza, M., Osindero, S.: Conditional Generative Adversarial Nets. arXiv: 1411.1784 (2014)
28. Mitra, A., Mohanty, S.P., Corcoran, P., Kougianos, E.: A novel machine learning based method for deepfake video detection in social media. In: Proceedings of IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS). pp. 91–96 (2020)
29. Mitra, A., Mohanty, S.P., Corcoran, P., Kougianos, E.: A machine learning based approach for deepfake detection in social media through key video frame extraction. SN Computer Science **2**(2), 1–18 (2021)
30. Nataraj, L., Mohammed, T.M., Manjunath, B.S., Chandrasekaran, S., Flenner, A., Bappy, J.H., Roy-Chowdhury, A.: Detecting GAN generated fake images using co-occurrence matrices. arxiv:abs/1903.06836 (2019)
31. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv: 1511.06434 (2016)
32. Radim Tyleček, R.Š.: Spatial pattern templates for recognition of objects with regular structure. In: Proceedings of German Conference on Pattern Recognition. Saarbrucken, Germany (2013)
33. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. p. 2234–2242 (2016)

34. Taigman, Y., Polyak, A., Wolf, L.: Unsupervised cross-domain image generation. arXiv: abs/1611.02200 (2016)
35. Tariq, S., Lee, S., Kim, H., Shin, Y., Woo, S.S.: Detecting both machine and human created fake face images in the wild. In: Proceedings of the 2nd International Workshop on Multimedia Privacy and Security. p. 81–87 (2018)
36. Varkarakis, V., Bazrafkan, S., Costache, G., Corcoran, P.: Validating seed data samples for synthetic identities – methodology and uniqueness metrics. IEEE Access **8**, 152532–152550 (2020)
37. Wang, R., Juefei-Xu, F., Ma, L., Xie, X., Huang, Y., Wang, J., Liu, Y.: Fakespotter: A simple yet robust baseline for spotting ai-synthesized fake faces. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. pp. 3444–3451 (7 2020)
38. Wang, T., Liu, M., Zhu, J., Tao, A., Kautz, J., Catanzaro, B.: High-resolution image synthesis and semantic manipulation with conditional gans. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 8798–8807 (2018)
39. Yi, Z., Zhang, H., Tan, P., Gong, M.: Dualgan: Unsupervised dual learning for image-to-image translation. arXiv: abs/1704.02510 (2017)
40. Yu, N., Davis, L., Fritz, M.: Attributing fake images to gans: Learning and analyzing gan fingerprints. In: Proceedings of IEEE International Conference on Computer Vision. pp. 7555–7565 (2019)
41. Zhu, J., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of IEEE International Conference on Computer Vision. pp. 2242–2251 (2017)
42. Zhu, J.Y., Zhang, R., Pathak, D., Darrell, T., Efros, A.A., Wang, O., Shechtman, E.: Toward multimodal image-to-image translation. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 465–476. Curran Associates, Inc. (2017)