Towards Discovery and Attribution of Open-world GAN Generated Images: Supplementary Material

Sharath Girish* sgirish@cs.umd.edu
Saksham Suri* sakshams@cs.umd.edu
Saketh Rambhatla rssaketh@umd.edu
Abhinav Shrivastava abhinav@cs.umd.edu
University of Maryland, College Park

1. Additional experimental details

For our feature extractor, we use 3 fully-connected layers. Each fully connected layer is followed by the ReLU activation unit and Dropout with a drop probability of 0.5 during train phase for regularization. The first layer maps the input 2048 dimensional vector to 512 dimension. The second layer maintains the number of activation units at 512 while the third one downsamples it to 128 which is the final dimension of the feature vector we use for subsequent stages. For all our experiments in the supplementary, we train on 128×128 sized images. We set a percentile threshold of 0.9 for our out-of-distribution detection stage. Our cluster merging algorithm using the 1-Nearest Neighbour Graph is a 2 staged setup which initially merges the newly obtained clusters from K-Means in the previous stage and then merges the entire clustered set. We adopt this 2 staged setup as K-Means overclusters the discovery set and requires merging before merging with the clusters in the clustered set. For training SVMs, we use the GPU-accelerated library of ThunderSVM [1].

2. Additional dataset details

Table 1 summarizes the class-wise train and test splits used across our experiments. We use a variety of images for multiple image sources to more closely simulate a real world setup. Note that some train images are not used depending on the dataset setup where the image sources could only belong in the discovery set.

Our online dataset setup defined in Section. 4.4.3 of the paper consists of GANs in the chronological order they were published or introduced. We have an initial labeled and discovery set as defined in Table 2. After running our pipeline for 2 iterations on this set, we add 3 more GANs to the discovery set: BigGAN and SSGAN [2], both trained on ImageNet, StyleGAN trained on CelebA-HQ. We run our pipeline for 2 more iterations on the new discovery set, and add 3 more GANs: ResNet19 and StarGAN-v2 [3], both trained on CelebA-HQ, S3GAN trained on ImageNet. This is followed by 2 more iterations of network training resulting in a total of 6 iterations for the full online setup. The numbers are as reported in Section 4.4.3 of the main paper.

*First two authors contributed equally

3. Additional baseline comparisons

Section 4.2 of the paper provides comparisons with baselines derived from the works of [4] and [5] by training their methods on our dataset in a multiclass manner. We additionally provide baselines by using features from the pre-trained models provided by them which were trained on their datasets. We provide results by performing K-means clustering on the features for $k = 20$ and $k = 500$ similar to the baselines derived in Section 4.2. The results are shown in Table 3 (denoted by *). It can be seen that the features don't generalize across datasets and does worse than the baselines reported in the paper on both the metrics of Average Purity and NMI.

Also, as [5] primarily deals with only real-fake classification, we train their method on our dataset but only on the binary real-fake classification task and extract their features. We show the results based on clustering the features for $k = 20$ and $k = 500$ and reporting results in Table 3 (denoted by †). We see that features generated from the binary classification problem do worse than the multiclass case. This is because the binary classification problem only discriminates between real and fake image sources while grouping the different fake image sources together. This causes less discrimination between the fake image sources harming the clustering performance. We also provide a baseline (denoted by #) using our approach but adding JPEG and blur augmentations as used by [5]. We see that this degrades the performance compared to our original approach likely because these augmentations destroy valuable high frequency information used for discriminating between GAN sources. Since the baseline performance was lower in these evaluations we did not include them in the main paper.

Table 1. List of GANs trained on the corresponding 4 real datasets used in our labeled and discovery set. Note that the same GAN can be trained on multiple datasets.

Dataset	Image Source	# of Images (Train)	# of Images (Test)
CelebA	Real	$20k$	$10k$
	StarGAN	$20k$	$5k$
	AttGAN	$20k$	$10k$
	BEGAN	$20k$	$10k$
	ProGAN	$20k$	$10k$
	SNGAN	$20k$	$10k$
	MMDGAN	$20k$	$5k$
	CramerGAN	$20k$	$10k$
CelebA-HQ	Real	$20k$	$10k$
	ProGAN	$20k$	$10k$
	StyleGAN	$20k$	$5k$
	ResNet19	$20k$	$10k$
ImageNet	Real	$20k$	$10k$
	BigGAN	$20k$	$5k$
	S3GAN	$20k$	$10k$
	SNGAN	$15k$	$10k$
LSUN-Bedroom	Real	$20k$	$5k$
	ProGAN	$20k$	$10k$
	MMDGAN	$20k$	$5k$
	SNGAN	$20k$	$3k$
	CramerGAN	$20k$	$10k$
DTD	Real	-	$10k$
	ProGAN	$20k$	$5k$
FashionGen	Real	$20k$	$10k$
	DCGAN	$20k$	$5k$
Night	Real	$15k$	$5k$
	Pix2Pix	$15k$	$10k$
Shoes	Real	$20k$	$3k$
	Pix2Pix	$20k$	$10k$

Table 2. Initial labeled and discovery set for our online setup.

Dataset	**Labeled GANs**	**Discovery GANs**
CelebA	BEGAN, MMDGAN, CramerGAN, ProGAN	BEGAN, MMDGAN, CramerGAN, ProGAN, StarGAN, AttGAN, SNGAN
CelebA-HQ	ProGAN	-
ImageNet	SNGAN	-
LSUN Bedroom	ProGAN, MMDGAN, CramerGAN	ProGAN, MMDGAN, CramerGAN,SNGAN

Table 3. Comparing the proposed approach with additional baselines from [4, 5]. * represents the pretrained features used for clustering while † denotes the features obtained from binary classification, the original task of [5]. We also provide a baseline (denoted by #) using our approach but with JPEG and blur augmentations as used by [5]. This does worse on both clustering metrics compared to our original approach.

# of clusters	**Method**	**Avg. Purity**	**NMI**
20	Wang *et al*. [5]*	0.1946	0.2042
	Yu *et al*. [4]*	0.4529	0.4543
	Wang *et al*. [5]†	0.3841	0.4434
500	Wang *et al*. [5]*	0.2929	0.2004
	Yu *et al*. [4]*	0.5947	0.3916
	Wang *et al*. [5]†	0.6082	0.4334
258	Ours$^{\#}$	0.7696	0.6249
266	Ours	0.8216	0.6552

4. Out-of-distribution detection

In this section, we provide more details on the WTA hash and also a comparison between cosine based distance and the WTA hashing based hamming distance for out-of-distribution detection. Additionally, we compare our approach with another popular out-of-distribution algorithm [6] and show that our algorithm performs well on their reported benchmarks. Finally, we analyze the effect of the percentile threshold used in our approach.

4.1. WTA hash details

The WTA hashing algorithm proceeds as follows. Suppose a single feature vector $\boldsymbol{x}$ has a dimension d. We generate H different permutations $\boldsymbol{p}_i$, $i \in \{1, ..., H\}$ of indices $\{1, ..., d\}$ and then apply each of these permutations to $\boldsymbol{x}$ to get a set of vectors $\{\boldsymbol{x}'_i\}_{i=1}^{H}$. For each vector $\boldsymbol{x}'_i$, we take the first K elements, for a window size K, and obtain the index of the max element. The set of these H indices (one for each permutation) yields a new vector $\boldsymbol{x}_H$. Note that $\boldsymbol{x}_H$ is a H dimensional vector with its elements taking integral values in $[0, K-1]$. The distance between two feature vectors is then defined as the hamming distance between their corresponding hashes.

4.2. Cosine based distance details

We compare the in-distribution, out-distribution and overall accuracy of our algorithm for the 12 seen classes (as described in Table 1 of the paper) using the WTA hash distance and a cosine-based distance. The results are shown in Table 5. As the number of samples in our in-distribution is roughly the same as number of samples in our out-distribution, we use standard accuracy as our metrics for

Table 4. Comparison of our approach using WTA hash with ODIN [6]. The in-distribution dataset is CIFAR-100 which is used to train a DenseNet. We evaluate our method on the same metrics reported in [6]. The numbers reported are in the format of "ODIN/Ours". All values are in percentages. ↑ implies that the larger value is better while ↓ implies smaller value is better. We outperform ODIN on all OOD datasets and metrics excluding LSUN (crop).

Out-distribution dataset	FPR at 95% TPR ↓	Detection error ↓	AUROC ↑	AUPR In ↑	AUPR Out↑
Tiny-ImageNet (crop)	26.9/**18.8**	12.9/**10.2**	94.5/**96.4**	94.7/**96.6**	94.5/**96.3**
Tiny-ImageNet (resize)	57.0/**20.2**	22.7/**10.6**	85.5/**96.2**	86.0/**96.3**	84.8/**96.1**
LSUN (crop)	**18.6**/32.1	**9.7**/14.0	**96.6**/93.8	**96.8**/94.2	**96.5**/93.7
LSUN (resize)	58.0/**17.3**	22.3/**9.6**	86.0/**96.8**	87.1/**97.0**	84.8/**96.7**
iSUN	64.9/**28.3**	24.0/**12.6**	84.0/**94.8**	85.1/**95.2**	81.8/**94.6**
Gaussian	100.0/**0.0**	17.9/**0.1**	99.5/**100.0**	87.5/**100.0**	65.1/**99.8**
Uniform	100.0/**0.0**	38.0/**0.0**	40.5/**100.0**	60.5/**100.0**	40.9/**99.9**

Table 5. Comparison of our OOD step using WTA hash or cosine distance. We see that the WTA hash consistently outperforms the cosine-based distance at all 4 iterations of training even though it drops slightly on the out-distribution accuracy.

Iteration	In-distribution Accuracy (%)	Out-distribution Accuracy (%)	Net Accuracy (%)
1	86.02/**91.74**	**93.26**/89.35	89.33/**90.65**
2	83.49/**88.33**	**98.36**/97.63	90.22/**92.58**
3	81.14/**85.37**	**99.32**/98.34	89.45/**91.3**
4	79.11/**82.94**	99.10/**99.12**	88.27/**90.33**

comparison. In-distribution accuracy refers to the accuracy on all the samples in the discovery set which belong to the 12 seen classes while out-distribution corresponds to those belonging to the 8 unseen classes. Net accuracy is the overall accuracy on the full discovery set. We see that using the hash outperforms cosine based distance in terms of the net accuracy and in-distribution accuracy. It performs lower than the cosine-based distance in terms of the out-distribution accuracy but with only a small difference. This is because of an inherent tradeoff between in-distribution and out-distribution accuracy based on the percentile threshold.

4.3. Related works comparison for OOD

We now compare our approach with the popular out-of-distribution (OOD) approach called ODIN [6] on their benchmark. We show results using features extracted from DenseNet trained on CIFAR-100. We evaluate on the various out-of-distribution datasets provided by the authors of [6] and report our results in Table 4. We see that we outperform their algorithm on almost all datasets except LSUN (crop). Additionally, our algorithm has very few hyperparameters which require careful tuning and does not require a validation set. This shows that our approach generalizes well to other dataset setups and can be used in general for the problem of out-of-distribution detection.

Table 6. Comparison between using the WTA hash based hamming distance or the cosine based distance for computing the 1-NN graph during merge step. We analyze the performance directly at iteration 1 and also at the end of 4 iterations for both stages of merge and refine.

#iter.	Stage	Avg. Purity	NMI	% Samples Discovered	# of Sources Discovered	# of clusters
1	Merge	0.791/**0.793**	0.642/**0.646**	-	-	433/**383**
	Refine	0.776/**0.780**	**0.671**/0.666	74.70/**76.54**	4/**5**	**167**/180
4	Merge	0.820/**0.825**	0.635/**0.647**	-	-	618/**432**
	Refine	0.819/**0.823**	0.651/**0.655**	91.80/**94.76**	**8**/8	294/**266**

Table 7. Reducing number of clusters for K-Means (K) by almost half at each iteration. Average Purity and NMI does not change drastically compared to our default setup.

Iteration	K	Avg. Purity	NMI
1	500	0.695	**0.6756**
2	250	0.7877	0.6572
3	125	**0.8086**	0.6484
4	60	0.8056	0.6399
Ours (Default)		0.8216	0.6552

Table 8. We Naïvely recluster the test set after each training step and use them as pseudolabels for retraining. Compared to our original approach, a significant drop in Average Purity and NMI is observed.

Step	Avg. Purity		NMI	
	Reclustering	Ours	Reclustering	Ours
1	0.7858	0.7803	0.5402	0.6658
2	0.7803	0.8183	0.5383	0.6625
3	0.7597	0.8211	0.5289	0.6595
4	0.7303	0.8216	0.5112	0.6552

4.4. Threshold analysis

We evaluate the performance of our pipeline when varying the percentile threshold which was set by default to 0.9. We run our full pipeline iteratively for 4 iterations and report the numbers for 4 different values of 0.7, 0.8, 0.9, 0.95. The results are summarized in Table 9. Increasing the

Table 9. Effect of the percentile threshold for OOD on the final performance. The default value for our experiments is 0.9. For all the thresholds, all sources were discovered.

β	Avg. Purity	NMI	# of Clusters	% Samples Discovered
0.7	0.7952	0.5842	449	0.9226
0.8	0.8087	0.6135	376	0.9234
0.9	0.8216	0.6552	266	0.9476
0.95	0.8268	0.6985	181	0.9358

threshold has the expected result of decreased clusters as more samples in the discovery set are called in-distribution and are attributed to existing clusters. However, the resulting cluster metrics do not vary drastically in the neighbourhood of the default value of 0.9. This shows that our pipeline is fairly robust to the percentile threshold which also intuitively transfers across datasets. Our approach thus has a single easy to tune scalar which automatically varies the threshold for the different seen classes or clusters.

5. Clustering

We analyze the effect of the clustering, merge and refine stages of the pipeline, varying a number of hyperparameters and show our pipeline's robustness to these values with respect to the final performance.

5.1. Effect of different number of clusters and number of rounds of merge and refine

The k value chosen for K-Means changes the number of clusters that are passed on to the merge step. Additionally, we try performing multiple rounds of merge and refine within a single iteration, which decreases the number of clusters and improve purity. Table 11 summarizes the effect of changing k and the number of rounds of merge and refine, r, for the pipeline. We see that for a fixed r and different k's, even though the number of clusters changes in the clustering stage, the final number of clusters at the end of 4 iterations are similar and the performance on Average Purity and NMI does not vary drastically. However, when we fix k and vary r we see that final number of clusters show a visible drop and as expected NMI increases slightly while Average Purity decreases.

5.2. Cosine distance based merging

Instead of using the hamming distance between the WTA hashes of the feature vectors, we use a cosine based distance between feature vectors and compare the performance at the end of the first iteration and also at the end of 4 iterations. Table 6 compares cosine and WTA based distance at both points in the pipeline at the end of both merge and refine steps. We see that the WTA hash based distance marginally outperforms the cosine based distance at almost all points in the pipeline. It also discovers a higher percentage of samples with fewer clusters demonstrating its effectiveness in our pipeline.

Table 10. Effect of varying the size threshold (τ) and SVM fire fire threshold (ϵ) on performance. The default setting corresponds to $\tau = 100$ and $\epsilon = 0.5$.

τ	ϵ	Avg. Purity	NMI	Sources Discovered	# of Clusters	% Samples Discovered
50	0.3	0.8178	0.6356	20/20	407	0.9695
	0.5	0.8212	0.6326	20/20	434	0.9729
	0.7	0.8223	0.6551	20/20	308	0.9549
100	0.3	0.8238	0.6451	20/20	293	0.9590
	0.5	0.8216	0.6552	20/20	266	0.9476
	0.7	0.823	0.6573	20/20	245	0.9237
200	0.3	0.8265	0.6731	20/20	166	0.8919
	0.5	0.8197	0.67	20/20	161	0.9015
	0.7	0.8233	0.686	19/20	138	0.8841
300	0.3	0.7849	0.7124	14/20	70	0.8643
	0.5	0.8009	0.6992	13/20	89	0.8340
	0.7	0.8055	0.7161	18/20	85	0.8638

5.3. Size threshold and SVM firing threshold

Next, we evaluate the effect of varying the size threshold, τ for discarding clusters at the end of refine step, as well as varying the SVM fire threshold, ϵ, which also controls the number of clusters (likely impure) being discarded. The results are summarized in Table 10. As expected increasing τ or ϵ decreases the number of clusters as more samples are discarded. This comes at the cost of fewer samples and GANs being discovered. However, the clustering metrics do not vary drastically showing that our pipeline is robust to these hyperparameter values and can generalize across varying datasets and sizes.

5.4. Reclustering

We now evaluate the effect of removing the merge and refine steps from our pipeline. Our pipeline consists of the usual network training followed by clustering. For each iteration, we discard the old clusters and perform clustering again using K-Means on all the test set samples. We then use the new clusters as pseudo-labels and retrain our network for improving the features. It should be noted there is no OOD detection, merge or refine steps. This method is similar to ClusterFit [7] who also use cluster pseudo-labels for network training. We analyze the results in Table 8 by comparing with our original pipeline. We see that the performance drops by a significant margin showing that it is crucial to maintain existing clusters and iteratively merge and refine them while simultaneously improving the feature representations of the existing clusters.

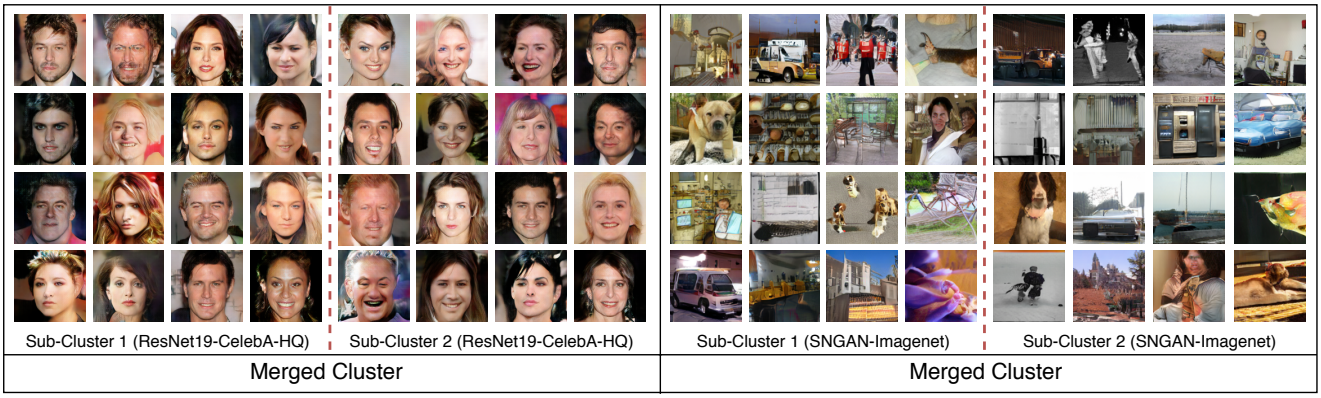

Figure 1. Some example clusters merged by our approach during the merge step.

Table 11. Effect of varying the number of clusters k for K-Means along with the number of times merge and refine (Additional Iters) is performed for each step. In the default setting Additional Iters is 0 as merge and refine are performed once per step while $k = 500$.

K	Additional Iters	Avg. Purity	NMI	Sources Discovered	% Samples Discovered	# of Clusters
100	0	0.7948	0.6722	20/20	0.9839	133
	1	0.7952	0.6938	20/20	0.9876	104
	2	0.7889	0.7016	19/20	0.9904	89
	3	0.7627	0.7255	19/20	0.9835	66
200	0	0.8086	0.657	20/20	0.9762	199
	1	0.8125	0.6798	20/20	0.9847	154
	2	0.7944	0.6981	19/20	0.9749	105
	3	0.7944	0.7085	20/20	0.9757	95
300	0	0.8142	0.652	20/20	0.9665	238
	1	0.8051	0.6696	20/20	0.9486	163
	2	0.802	0.6975	20/20	0.9602	121
	3	0.7669	0.7046	17/20	0.9569	92
400	0	0.8175	0.6472	20/20	0.9494	281
	1	0.814	0.6639	20/20	0.9637	196
	2	0.8105	0.6817	20/20	0.9687	158
	3	0.8005	0.7078	20/20	0.9679	113
500	0	0.8216	0.6552	20/20	0.9476	266
	1	0.8249	0.6736	20/20	0.9532	195
	2	0.8118	0.6887	20/20	0.9536	136
	3	0.7862	0.691	20/20	0.9485	114
600	0	0.8264	0.6535	20/20	0.9396	279
	1	0.8229	0.667	20/20	0.9545	212
	2	0.8176	0.6854	20/20	0.9470	153
	3	0.8039	0.6936	20/20	0.9459	121
700	0	0.8279	0.6572	20/20	0.9137	266
	1	0.8294	0.6837	20/20	0.9115	172
	2	0.8348	0.6887	20/20	0.9473	166
	3	0.8233	0.6923	20/20	0.9382	139

5.5. Effect of varying number of clusters

For most of our experiments we run the clustering using K-Means at a fixed value of k, which is used for all iterations. As the number of undiscovered samples reduce as number of iterations increases, we evaluate our pipeline's performance by decreasing k after each iteration. We thus, approximately halve the value of k after each iteration. The results are reported in Table 7. We see that the performance does not change drastically compared to the default setup which shows that our network does not heavily rely on the number of clusters used during K-Means.

5.6. Qualitative Analysis

We now qualitatively show the effect of our merge step for a few clusters. The merge step of our approach merges clusters belonging to the same class but are actually fragmented due to overclustering from K-Means. We visualize two such clusters in Fig. 1. As we showed in the main paper, our clusters focus on the GAN source rather than image semantics and the merge step successfully combines clusters having the same majority GAN source.

6. Network Training

6.1. Effect of faster training

By default, we retrain all feature extractor weights every iteration. To reduce the cost of full network retraining, we analyze finetuning only the final residual block of the ResNet-50 backbone along with the subsequent fully connected layers of the feature extractor. We also analyze using a lighter network such as MobileNet [8] for our network training and compare it with our original setup. Additionally, we try to see the effect of removing the merge step from the pipeline The results are summarized in Table 12. We see that there is a small drop in network performance in terms of both Average Purity and NMI and it also fails to discover a single unseen source. Constraining the network is likely to have restricted the network's capability of improving the discovery set features although it doesn't have a significant impact. On the other hand, MobileNet obtains

Table 12. Results on our setup with slight variations in our training.

Experiment	# of Clusters	Avg. Purity	NMI	# Sources Disc.
Ours (Original)	209	0.861	0.724	20/20
Ours (w/o merge)	257	0.841	0.712	19/20
Ours (Freeze)	229	0.850	0.691	19/20
MobileNet	70	0.846	0.773	15/20

Table 13. Results on our setup with varying image sizes.

Image Size	# of Clusters	Avg. Purity	NMI	# Sources Disc.
64	169	0.655	0.579	19/20
128	266	0.822	0.673	20/20
256	209	0.861	0.724	20/20

a higher NMI because of much fewer clusters, albeit at the cost of not discovering most of the unseen sources. This shows that very light networks are not as effective in obtaining discriminative representations for discovering new sources. Also the performance without merge is sub-par to our original approach which shows the importance of performing merge step to group similar clusters together after over-clustering.

6.2. Effect of image size

By default, we resize and center crop all images to 256×256 in most of our experiments. We compare results when image sizes are varied from 64 to 256 for network training. From Table 13, we see that increasing image size shows a marked improvement in all metrics. Therefore, we hypothesize that model fingerprints are likely more detectable and distinguishable when the image is resized to a higher resolution. However, this comes at the cost of increasing network training times and memory requirements (quadratically) which is infeasible in an online setup or for very large scale datasets.

7. Multiple seed sources

[4] showed that training generators with different random seeds generate different distinguishable fingerprints in their images. We analyze whether we can discover new separate sources when a single generator architecture is trained on the same dataset but with different seeds. Table 14 shows results comparing this setup with our original setup. We add 2 different seeds for ProGAN for both CelebA and LSUN-Bedroom providing 2 new sources. Note that only a single seed of ProGAN trained on LSUN-Bedroom is present in the labeled set while the other 3 sources are unseen. The remaining classes are same as our original setup as described in Table 1 of the main paper. We see that there is only a small drop in Average Purity and NMI although it fails to discover a single unseen source.

Table 14. Results on our setup with variations in training.

Experiment	# of Clusters	Avg. Purity	NMI	# Sources Disc.
Ours (Original)	209	0.861	0.724	20/20
Ours + Unseen seeds	216	0.842	0.702	21/22

References

[1] Zeyi Wen, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. ThunderSVM: A fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research*, 19:797–801, 2018. 1

[2] Ting Chen, Xiaohua Zhai, Marvin Ritter, Mario Lucic, and Neil Houlsby. Self-supervised gans via auxiliary rotation loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12154–12163, 2019. 1

[3] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8188–8197, 2020. 1

[4] Ning Yu, Larry S Davis, and Mario Fritz. Attributing fake images to gans: Learning and analyzing gan fingerprints. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7556–7566, 2019. 1, 2, 6

[5] Sheng-Yu Wang, Oliver Wang, Richard Zhang, Andrew Owens, and Alexei A Efros. Cnn-generated images are surprisingly easy to spot... for now. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 7, 2020. 1, 2

[6] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017. 2, 3

[7] Xueting Yan, Ishan Misra, Abhinav Gupta, Deepti Ghadiyaram, and Dhruv Mahajan. Clusterfit: Improving generalization of visual representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6509–6518, 2020. 4

[8] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5