Methods to Detect GAN-Generated Images

Aditya Agrawal, Minh Le Nguyen, Nicholas Zotalis, Sichao Yu
{adityaai,minhng99,nzotalis,sichaoy}@bu.edu

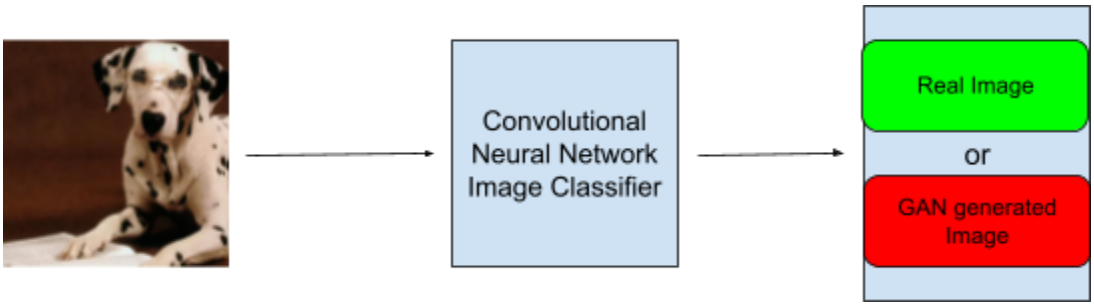


Figure 1. Simple Representation of task.

1. Task

Deep Learning has brought to humankind countless applications that we now use everyday. One of its most interesting creations is Generative Adversarial Network (GANs). Designed in 2014 by a Ph.D fellow at the University of Montreal, Ian Goodfellow, GANs has quickly become popular in the last several years among the field of Machine Learning. It opened up many new directions in research and development.

Unfortunately, GANs's potential evil is as big as its potential for good. There are GAN-based methods that generate images for malicious purposes. However, there are some peculiar traces left by the generation process that can be exploited to detect them. Some examples of this are shown in Fig. 2.

Figure 2. Examples of GAN synthetic images together with their visible and not visible artifacts. From top to bottom: color artifacts, artificial fingerprint and its averaged version, Fourier spectrum and its averaged version.

Our task was to analyze the state-of-the-art methods for the detection of GANs synthetic images, showing the key factors that create a successful approach, and compare methods' performance over existing generative architecture. Along the way, we faced difficulties in classifying unseen instances, unusual data formatting, and a minor computing resource, which will all be addressed later in this report.

2. Related Work

Early work learns the spatial domain features and exploits the intrinsic constraints of GAN. CO-Net [1] extracts co-occurrence matrices from RGB channels to train the CNN. A investigation of detectors turned out that deep pre-trained networks like Xception [2] are effective for GAN image detection. However, they fail to generalize to images generated by new unseen GAN models. Therefore, some methods have been proposed to improve models robustness and ability to generalize. Spec [3] learns frequency domain features like the presence of spectral peaks caused by the upsampling operations of most GAN architectures. One proposed solution M-Gb [4] model would be to add gaussian blurring so as to force the discriminator to learn more generalized features. A similar method was used in Wang2020 model [5], which is based on ResNet50. PatchForensics [6] proposed a different approach based on a fully-convolutional patch-based classifier. SRNet [7] describes a deep residual architecture which is able to learn both spatial-domain and JPEG steganography features and minimize hand-designed elements like heuristics. All those

methods claim to improve the generalization ability according to the performance on unseen test data.

3. Approach

We followed [8] as the reference paper for this project. We implemented a few state-of-art detectors and compared their performance on a large dataset of GANs-generated images and real images.

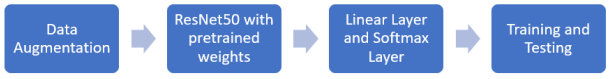
By creating the CNNs using pytorch neural network layers based on their architecture described in their respective research papers, we implemented Xception, Spec, Wang2020, PatchForensics, M-Gb, SRNet, CO-Net, and our own detector.

For all models, we used the techniques highlighted in [11] for loading and visualizing the train and test images. By using torchvision's ImageFolder class, the model training time is reduced considerably, allowing us to train them for multiple epochs.

Wang2020 [5]: This classifier uses a ResNet-50 pre-trained with ImageNet and it is trained in a binary classifier setting. The final layer of the ResNet 50 is changed to a Linear Layer with output size 1, followed by a softmax layer. Before feeding the training images to the model, some augmentation is done to the images.

1. Some images are horizontally flipped (left-right) with probability of 50%.
2. Some Images are blurred with Gaussian Blurring with probability of 10%, where $\sigma \sim$ Uniform[0.1,3]
3. All images are cropped to 224x224 pixels at a random location.
4. Some Images are JPEG Compressed with quality~Uniform[30,...,100] with probability of 10%.

During testing, the images are only center cropped to 224x224 pixels. There is no blurring, flipping or compression.
The flow of the classifier is shown in the figure below.

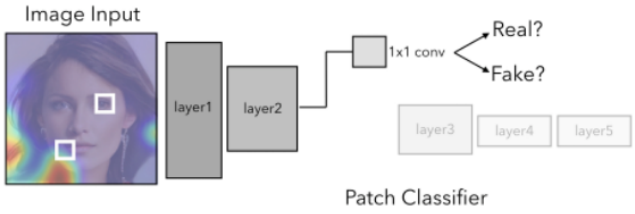


The individual image transforms are implemented using torchvision.transforms functions. The Dataloader is implemented using Pytorch modules [11]. The model is implemented using torchvisions's ResNet50 and other layers from the torch library. The training, and testing loops are all implemented from scratch.

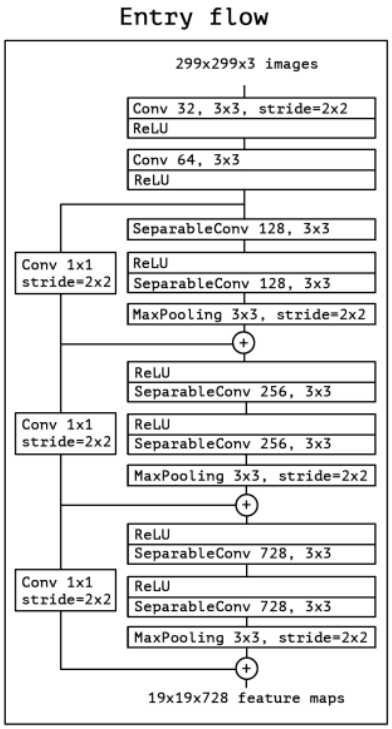
Spec [3]: This method utilizes a pretrained Resnet34 to perform binary classification on real images and gan-generated images. First a random 224x224 cropping is taken from the training images and fed forward into Resnet. The output of Resnet, which is 1000-dimensional, is then connected to the two output nodes which perform the classification. SGD was used as the optimizer with a learning rate of .001 and momentum .9. No learning rate decay was used here. In testing, a central cropping of 224x224 was taken from testing images.

Xception [2]: Compared to other neural networks, Xception features deep wise separable convolution operations. First We implemented the separable convolution operation with nn.module. In this class, we define a convolutional layer followed by a pointwise convolutional layer. Then we implement a block class which combines separable convolutional layers, ReLU and max-pooling layers. We repeatedly define the required 14 modules which contain 36 convolutional layers [12].

PatchForensics [6]: This method uses patch-based classifiers to visualize which regions of fake images are more easily detectable.

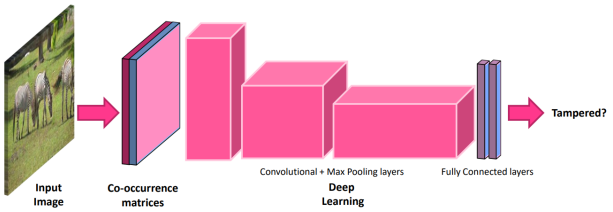


In [8], the implementation of PatchForensics is Xception first blocks trained on patch-level images. For this project, Xception entry flow blocks were implemented, with inputs as 299x299x3 images.



[Image source] [10]

Co-Net [1]: This method computes the co-occurrence matrices on the RGB Channels of an image and passes them through a convolutional neural network. For both training and testing, the images are not augmented and the co-occurrence matrix is calculated directly on the image pixels, with a distance offset of 1. This 3x256x256 tensor is passed through a multi-layer deep convolutional neural network: conv layer with 32 3x3 convs + ReLu layer + conv layer with 32 5x5 convs + max pooling layer + conv layer with 64 3x3 convs + ReLu layer + conv layer with 64 5x5 convs + max pooling layer + conv layer with 128 3x3 convs + ReLu layer + conv layer with 128 5x5 convs + max pooling layer + 256 dense layer + 256 dense layer + sigmoid layer. The framework for the Co-Net model is shown below. Taken from reference paper [1].



[Image Source][1]

The co-occurrence matrices are calculated on images using skimage.feature's greycomatrix() function. The Dataloader is implemented using Pytorch modules [11]. The model is implemented using layers from the torch library. The training, and testing loops are all implemented from scratch.

M-GB [4]:
This is the shallowest network we attempted to use. M-Gb resizes its inputs to 128x128, then randomly applies a Gaussian blur with a random kernel size of either 1, 3, 5, or 7, and a standard deviation randomly chosen between 0 and 5. It has 4 convolutional layers with kernel sizes 4 and strides 2 and one fully connected layer. It uses batch norming in between convolutional layers.

SR-Net [7]: It is a 12-layer network as described in [7]. The first two layers don't contain any residual shortcuts or pooling. Layers 3-7 have residual shortcuts without pooling. Layers 8-11 contain both residual shortcuts and average pooling. The last layer contains global pooling. The key feature of SR-Net is to avoid decreasing signal and optimize residual extraction by not using pooling in the first 7 layers.

Our Own GAN Detector: Moving beyond the approach in the paper, we believed that a neural network with only convolution layers and linear layers should be able to do the task. Thus, our own GAN detector is inspired by DCGAN from Pytorch tutorials [13]. However, instead of discarding the discriminator as normal GANs, we discard the Generator and use the Discriminator as our detector for the synthetic images. Compared to DCGAN on Pytorch, which takes in input as 64x64x3 images, we added a few more convolution layers and fully connected linear layers to our network. We enabled our network to take in 256x256 images. We also increase the number of features maps. Reasons for these modifications will be explained in detail in the results section.

4. Datasets
We will use the datasets referenced in [9]. This contains an extensive dataset of real images and GAN generated images.

For training the detectors, we use 360K real images extracted from the LSUN dataset and 360K generated images obtained by 20 ProGAN Models, each trained on a different LSUN object category (airplanes, horses, cars, cats, etc.).

For testing, we use images coming from GAN architectures never seen in training. We have 45K synthetic images generated by 13 different CNN-based

image generators (StyleGAN, ProGAN, BigGAN, CycleGAN, StarGAN, GauGAN, Deepfake etc.), including both low resolution and high resolution images. We also have 45K real images from ImageNet, RAISE dataset, etc used for generating the synthetic images.

The 13 test categories are: - {imle, stylegan, san, progan, cyclegan, stylegan2, deepfake, gaugan, whichfaceisreal, seeingdark, crn, stargan and biggan}.

Any preprocessing of the dataset is done separately for each model.

5. Evaluation Metrics

Both our training and test datasets are balanced, both in terms of category of training data and in terms of labels, real or fake. Therefore we use simple accuracy as our metric. This is the proportion of tested images that were correctly identified by our model.

All the models will be trained on the ProGAN images. Since we have 13 different GAN sources for testing images, each model will be evaluated on each test category separately. This will help us determine which models are best/worst at identifying GAN-generated images from each category, as well as which models generalize the best across all categories.

6. Results

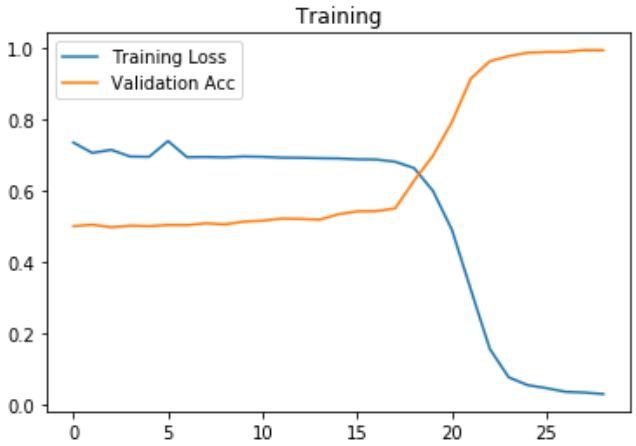
All of our methods achieved very high performance on test images from ProGAN. Five of our seven models achieved nearly 100% test accuracy. This is encouraging as our models were trained on ProGAN images so we would expect the best performance on ProGAN test images.

Most testing categories had at least one model perform well, although some were difficult for all models. GauGAN, BigGAN and WhichFaceIsReal were particularly challenging, with our best models achieving 77% test accuracy on all of these. The model which generalized the best is Spec, achieving an 81.88% test accuracy. More details about each method follow.

PatchForensics [6]:

Xception first blocks on patch-level images**:** In our first checkpoint, we successfully resized the images to 299x299x3, re-implemented the entry-flow block of Xception, and followed it by 1 fully connected layer. However, the result did not look very good, with accuracy around 50%.

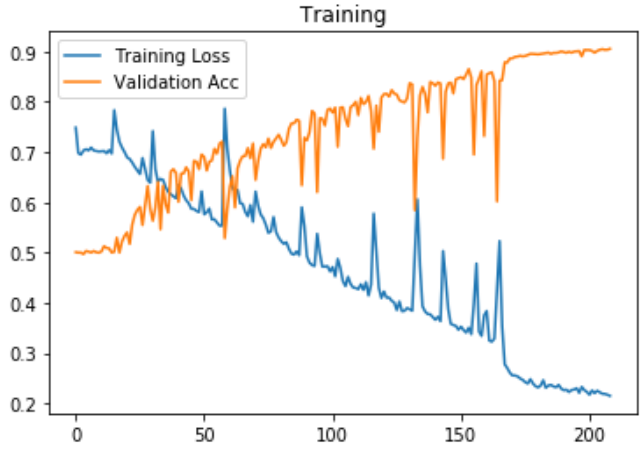
We investigated the problem and developed the solution. Instead of resizing, we do a random crop of the image to 299x299x3. With this method, the resolution of the image was reserved, while satisfying the input condition of patchForensics.



After training with 4 epochs, batchsize = 16, we got some very good results, with 99.1% accuray on the ProGAN test set, 90% on StarGAN, and 89% on StyleGAN.

M-Gb [4]:

It was trained with a batch size of 16 and learning rate .001. It was trained for a total of 10 epochs, with the learning rate decreasing to .0001 after 8 epochs.

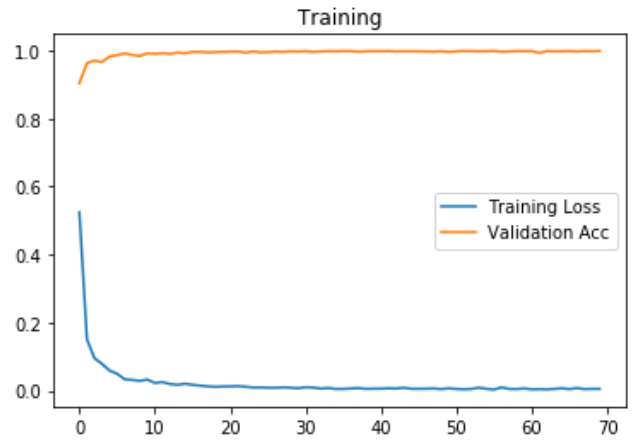


It was our worst-performing model, only reaching 91% accuracy on ProGAN images and not generalizing well to any other category. We suspect that this network is too shallow to adequately learn the differences between real and fake images.

Spec [3]

Spec fine-tunes a pre-trained ResNet18 architecture. ResNet is trained on the ImageNet database which gives our model exposure to more training data. X.

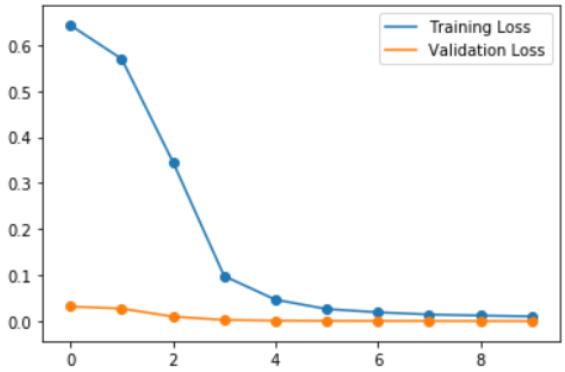
Zhang et. al's implementation of Spec replaced the last fully connected layer of ResNet18 with a fully connected layer to 2 output nodes. We found a slight improvement by keeping the original 1000 ResNet18 outputs and connecting them to a new fully connected layer with 2 output nodes. Spec was trained for 5 epochs, with batch size 16, and SGD optimizer with learning rate .001 and momentum .9.



Spec learned the difference between real and fake images very quickly and also generalized better than our other models. We suspect that its pre-training helped it in this regard.

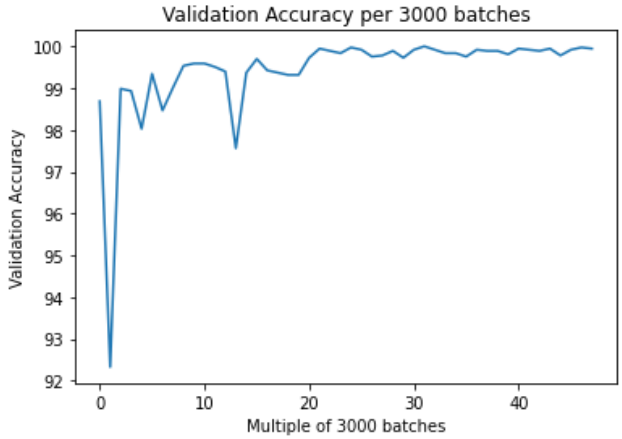
Xception [2]:

From the experiment on single category data samples we found that Xception works better with SGD optimizer. The hyperparameters are momentum=0.9, learning rate = 0.0001, weight decay = 1e-6. It is okay to run Xception without a pre-trained option, but in our case we imported the parameters from the PyTorch model zoo. The validation loss stops decreasing after 10 epochs. Increasing training epochs doesn't seem to increase the accuracy significantly. We used the trained model with 9 epoche as the final result.



The overall performance is not bad on all test datasets with an average accuracy of 76.5%. But the result is quite imbalanced. It achieves over 90% accuracy on some datasets while also getting a few results below 60%.

Wang2020 [5]: Training was done on all images in the training dataset with epochs=12 and batch size=64. For validation, 0.5% of training images are used. Adam optimizer and Binary Cross Entropy loss were used during training with a learning rate of 10^{-4}. The hyperparameters are the same as recommended in the original paper. The validation stabilizes after 10 epochs.

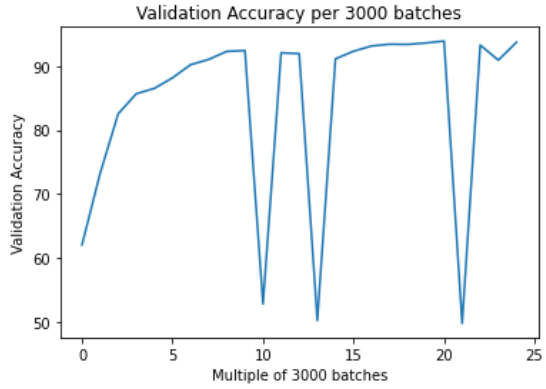


This model performs very well, with an average accuracy of 80.24 % on all datasets. It achieves an accuracy of greater than 90% on 5 datasets, with progan having 100% accuracy. It does not perform well on the "san" dataset.

Co-Net [1]:

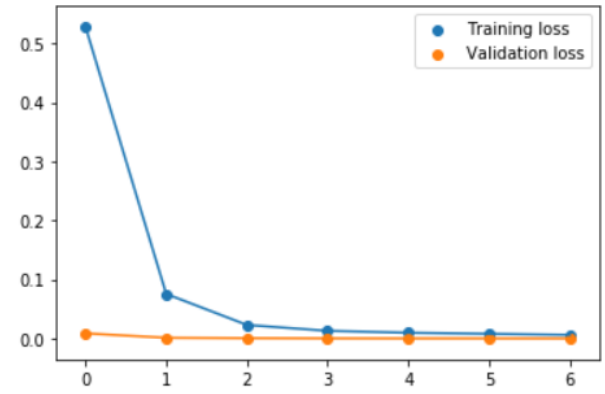
Training was done on all images in the training dataset with epochs=5 and batch size=64. For validation, 0.5% of training images are used. Adam optimizer and Binary Cross Entropy loss were used during training with a learning rate of 10^{-6}.

Multiple Learning rates were tried and with high learning rate, the model was not learning. 10^{-6} is the best learning rate found. The validation stabilizes at 5 epochs



The model achieves an average test accuracy of 65.23 % on all the test datasets. It gets good results on progan, stylegan and stylegan2, but some bad results with others like 48.8% on seeingdark and san dataset.

SR-Net [7]:
SR-Net has 12 layers, which is the deepest among the non-pre-trained models we tried. It is trained with SGD optimizer and hyperparameters are momentum=0.9, learning rate = 0.0001, weight decay = 1e-6. The validation loss stops decreasing after 6 epochs.



It performs well on most test datasets and achieves an average accuracy of 79.50%. This may imply a deeper convolutional structure will perform better on the GAN detection task.

Our Own GAN Detector:
Our GAN Detector is inspired by DCGAN. So our first step was to try running a DCGAN discriminator network on our dataset. As we expected, the DCGAN discriminator, which takes in 64x64x3 images, was not complex enough for the task. The model was only predicting all the instances as fake images, which ended up having 50% accuracy.

We decided to add a few convolution layers that enabled the network to take in 256x256 images as inputs. We then trained the network on the Airplane category in our training dataset. However, the results were still not good. The model was still predicting instances as 1 class.

We then tried adding fully connected linear layers to the network, and trained the model on the Airplane category again. The results were promising this time. The model was no longer predicting only 1 class. And it was performing equivalently well compared to PatchForensic trained on Airplane category only.

We were confident with our model performance. However, when we trained on our full dataset, it was taking around 18 hours to complete 1 epoch. We supposed that it got too complex. We wanted to fix this issue, but encountered some issues with computing resources (details explained next paragraph). Thus, we decided to stop here and leave this task for the future. The implementation of our own neural network synthetic image is in ourOwnGan.ipynb (can be found in the GitHub link)

Error encountered: we encountered some problems with the SCC. SCC did not allow one of us to save our notebook, with disk I/O error. In addition, it also prevents one of us from requesting a new SCC session, with quota full error.

7. Conclusion
We found that while it was straightforward to train a model to identify images generated by a particular GAN architecture, it is more difficult to identify GAN images in the wild. Deeper networks performed better than shallower networks in general. In addition, models that utilized pre-trained architecture such as ResNet also achieved higher accuracies and generalized better.
Obtaining more training data from different GAN sources is likely to improve generalizability. However, Obtaining such large datasets is difficult and computationally intensive. New GANs with different parameters, normalization, and regularization strategies are always being developed. We conclude that it is a difficult task to detect fake images, although the results of this experiment are encouraging as they provide a guide to further research.

Appendix A. Detailed Roles
Detailed Roles Highlighted in Table 1.

Appendix B. Code repository

Link to Github Repository: - https://github.com/MinhNguyen99AI/DeepLearning523---Synthetic-Image-Detector-Assestment

Appendix C. Detailed Results

See table 2

References

1) L. Nataraj et al., “Detecting GAN generated fake images using co-occurrence matrices,” in IS&T EI, Media Watermarking, Security, and Forensics, 2019.
2) F. Marra, D. Gragnaniello, D. Cozzolino, and L. Verdoliva, “Detection of GAN-generated fake images over social networks,” in IEEE MIPR, 2018.
3) X. Zhang, S. Karaman, and S.-F. Chang, “Detecting and Simulating Artifacts in GAN Fake Images,” in IEEE WIFS, 2019, pp. 1–6.
4) X. Xuan, B. Peng, W. Wang, and J. Dong, “On the generalization of GAN image forensics,” in Chinese Conference on Biometric Recognition, 2019, pp. 134–141.
5) S.-Y. Wang, O. Wang, R. Zhang, A. Owens, and A. Efros, “CNN-generated images are surprisingly easy to spot... for now,” in CVPR, 2020.
6) L. Chai, D. Bau, S.-N. Lim, and P. Isola, “What makes fake images detectable? Understanding properties that generalize,” in ECCV, 2020.
7) M. Boroumand, M. Chen, and J. Fridrich, “Deep residual network for steganalysis of digital images,” IEEE TIFS, vol. 14, no. 5, pp. 1181–1193, 2019.
8) D. Gragnaniello, D. Cozzolino, F. Marra, G. Poggi and L. Verdoliva, "Are GAN Generated Images Easy to Detect? A Critical Analysis of the State-Of-The-Art," in 2021 IEEE International Conference on Multimedia and Expo (ICME), Shenzhen, China, 2021 pp. 1-6.
9) https://github.com/PeterWang512/CNNDetection/tree/master/dataset
10) Tsang, Sik-Ho. “Review: Xception - with Depthwise Separable Convolution, Better than Inception-V3 (Image...” *Medium*, Towards Data Science, 20 Mar. 2019, https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568.
11) https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
12) https://github.com/tstandley/Xception-PyTorch
13) “DCGAN Tutorial¶.” *DCGAN Tutorial - PyTorch Tutorials 1.11.0+cu102 Documentation*, https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html.
14) https://github.com/brijeshiitg/Pytorch-implementation-of-SRNet

Table 1. Team member contributions

Name	Task	File names	No. Lines of Code
Sichao	Implemented Xception [2] Implemented SR-Net [7] Wrote Approach & Results for respective models Contributed equally to rest of the report	Xception.ipynb SRNet.ipynb	1912
Nicholas	Implemented Spec [3] Implemented M-Gb [4] Wrote Approach & Results for respective models Contributed equally to rest of the report	Spec.ipynb M-Gb.ipynb	1525
Aditya	Implemented Wang2020 [5] Implemented Co-Net [1] Wrote Approach & Results for respective models Contributed equally to rest of the report	Co_Net.ipynb Wang_2020.ipynb	1409
Minh	Implemented PatchForensics [6] Implemented ourOwnGan Wrote Approach & Results for respective models Contributed equally to rest of the report	PatchForensics.ipynb ourOwnGan.ipynb	1212

Table 2. Detailed results on each dataset

	spec	PatchForensics	Xception	Wang2020	M-Gb	SR-Net	CO-net
progan	99.74%	99.06%	99.80%	100 %	91.86%	99.90%	93.67 %
imle	**88.14%**	55.43%	50.50%	**90.14 %**	63.51%	50.5%	63.77 %
stylegan	86.91%	84.33%	84.13%	84.74 %	59.94%	86.00%	**93.91** %
san	58.22%	66.21%	**89.29%**	50.23 %	51.14%	**91.07%**	48.86 %
cyclegan	70.29%	75.55%	70.48%	83.61 %	68.05%	**89.76%**	64.8 %
stylegan2	**96.89%**	89.79%	87.70%	80.75 %	70.24%	**96.90%**	92.46 %
deepfake	**88.23%**	75.1%	71.01%	55.32 %	49.66%	75.44%	55.63 %
gaugan	61.54%	57.09%	59.11%	**77.08 %**	64.08%	67.52%	51.62 %
whichfaceisreal	77.15%	57.3%	50.00%	**79.10 %**	53.65%	73.20%	52.8 %
seeingdark	77.78%	58.89%	**100%**	90.28 %	71.11%	76.09%	48.89 %
crn	**91.20%**	55.02%	50.31%	**90.37 %**	55.25%	50.31%	63.84 %
stargan	**100%**	90.67%	**100%**	93.12 %	66.61%	**99.94%**	63.81 %
biggan	68.42%	65.78%	**76.40%**	68.47 %	59.95%	**77.20%**	53.9 %
Average	**81.88%**	71.55%	76.05%	**80.24%**	63.47%	**79.50%**	65.23 %