Re-implementation of CNN Detection via TensorFlow 2.0

E4040.2020Fall.ACNN.report
Jingyuan Liu jl5742, Bo Feng bf2477, Joseph Yang zy2431
Columbia University

Abstract

In this work, our main goal is to 1) re-implement the paper "CNN-Generated Images Are Surprisingly Easy to Spot... for Now" (Wang et al., 2020 [2]) in TensorFlow 2.0 (original code was implemented in Pytorch) and 2) try to verify (improve if possible) the results of the original paper by utilizing various training strategy and finding more public datasets (e.g. available datasets from public GAN models published on Github). In our experiments, we faced several challenges: 1) low experiment efficiency due to the huge amount of images in the dataset and limited computing resources available 2) limited public datasets 3) needs to develop customized data pipeline for training our model. Through carefully designed repetitive experiments, we verified the original paper's conclusion with our results. Also, we confirmed the network structure and training strategy, which were proposed by the original paper, have amazing generalization ability, by testing on the extra datasets we found (StyleGAN2 and whichfaceisreal, WFIR). Besides, we concluded that the random data augmentation used in the original paper might cause final performance to fluctuate by a certain extent.

1. Introduction

Recently, the advances in image synthesis using deep network techniques (e.g. Generic Adversarial Networks, GAN) have gained huge public attention. People are wondering how the world might change after the computer-synthesized images generated by deep neural networks flood into our life. The main concern of the original paper, and our project, is whether there's a simple way to tell which images are real and which are fake (generated by NNs).

The original paper pointed out that, while traditional machine learning classifier is highly likely to overfit on training dataset due to image dataset bias (e.g. a classifier trained on a face dataset performs horribly on a car dataset since these two datasets are very different), classifiers trained to detect CNN-generated images *can* exhibit a surprising amount of generalization ability across datasets, architectures, and tasks, contrary to current understanding.

So the goal of *our* project is to:

1) Build the model from original paper from scratch with TensorFlow 2.0
2) Try to find extra data (images generated by other GANs that are not included in the original code datasets) to do further tests on the generalization ability of the model.
3) Utilizing various training strategies to compare output results and find out the factors that affect experiment performance.

In our project, we have faced the following challenges:

1) The original paper used ResNet-50 as the base model and built further structures based on that. They chose the PyTorch library in their released code. So the first task for us is to independently build the whole structure in TensorFlow, based on the description of the model given in the original paper.
2) When we tried to find some extra datasets on public GitHub, we noticed that most released GANs either did not release their datasets, or they simply used the datasets that were released by some famous papers, which had already been covered in the original paper. In the end, we found two extra datasets, StyleGAN2 and WFIR (Which Face Is Real), for our model's extra testing.
3) In our experiment, we are dealing with considerably large datasets (700k+ images, over 70GB) with limited computing resources. To solve this problem, we chose to first verify our code structure and training design on smaller dataset and then train each final model for 8 epochs while manually set experiment configurations (directory hierarchy, file names, evaluation functions)

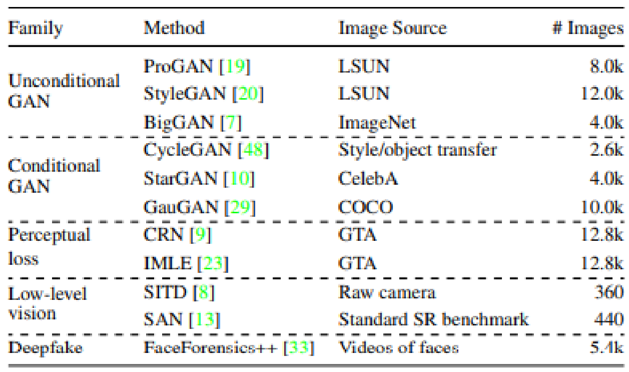
2. Summary of the Original Paper

Here, we summarize the original paper by the methodology it uses and the key results it produces.

2.1 Methodology of the Original Paper

2.1.1 Dataset Generation of the Original Paper

To study the transferability of classifiers trained to detect CNN-generated images, the authors collected a dataset of images created from 11 synthetic models. Their methods of data collecting spanned a variety of CNN architectures, datasets, and losses. All 11 models have an upsampling convolutional structure, as this is by far the most common design for generative CNNs. The statistics of each model can be found in Table 1.



Family	Method	Image Source	# Images
Unconditional GAN	ProGAN [19]	LSUN	8.0k
	StyleGAN [20]	LSUN	12.0k
	BigGAN [7]	ImageNet	4.0k
Conditional GAN	CycleGAN [48]	Style/object transfer	2.6k
	StarGAN [10]	CelebA	4.0k
	GauGAN [29]	COCO	10.0k
Perceptual loss	CRN [9]	GTA	12.8k
	IMLE [23]	GTA	12.8k
Low-level vision	SITD [8]	Raw camera	360
	SAN [13]	Standard SR benchmark	440
Deepfake	FaceForensics++ [33]	Videos of faces	5.4k

Table 1 (Selected from the original paper): Generative models. They evaluate forensic classifiers on a variety of CNN-based generative models. Notice that this is a screenshot from the original paper. We re-number the reference in the following way: ProGAN[4], StyleGAN[5], BigGAN[6], CycleGAN[7], StarGAN[8], GauGAN[9], CRN[10], IMLE[11], SITD[12], SAN[13], FaceForensics++[14]

Here the original paper considered 4 families of synthetic models: Unconditional GANs: three state-of-the-art unconditional GANs trained on either LSUN or ImageNet datasets; Conditional GANs; Perceptual Loss: models that directly optimize a perceptual loss, with no adversarial training; Low-level Vision: models like SITD; and Deepfakes: models that perform face-replacement tasks, as face recognition is one of the most complicated high-level tasks in Computer Vision.

They collected images from the models by generating without applying additional post-processing. To pair real images with fake images, real images went through the same preprocessing pipeline to better match with fake images. Also, the file names and directory hierarchies were taken care of.

Since 256*256 resolution is the most commonly shared output size among most state-of-the-art image synthetic models, for models that originally return 256*256 resolution, they did not perform postprocessing and for models that do not, they performed rescaling on the model outputs to match the 256*256 resolution.

Finally, the last step of dataset preparation was to perform a general image processing step: for all datasets, the authors made their real/fake predictions from 224*224 resolution crops of the original 256*256 resolution images (random-crop at training time and center-crop at the testing time). We believe that this step adds noise to the dataset and makes the model more unlikely to overfit since the original paper was trying to do a generalization work.

2.1.2 Model Training of the Original Paper

The authors emphasized that not all models are suitable for classifier training, due to limitations in data size. So, as the unconditional GAN models can synthesize arbitrary numbers of images, they chose ProGAN (one of the three unconditional GANs which produces high-quality images and whose structure is simple) to train the detector on.

The main idea of their experiments was to train a binary real-vs-fake classifier on the pre-generated large ProGAN dataset, and evaluate how well the model generalizes on other state-of-the-art GAN models. They chose ResNet-50, pre-trained with ImageNet, as the classifier for their task.

During training, all training images were randomly left-right flipped and cropped to 224*224 resolution, as described in the previous section. The original paper proposed further data augmentation in the following ways: **(1)**No further augmentation **(2)**Gaussian blur with 50% probability before cropping **(3)**JPEG-ed by OpenCV and PIL with 50% probability **(4a)**Blur and JPEG-ed both with 50% probability **(4b)**Blur and JPEG-ed both with 10% probability.

The model's performance was evaluated using *average precision* because it's threshold-less and non-sensitive on the rate of the real and fake images in the dataset. The average precision (AP) score was computed separately on each dataset. This is because images in one dataset are likely to contain similar semantic contents, while images in different datasets are likely to contain different semantic contents.

2.2 Key Results of the Original Paper

We conclude the original paper's key results in the following sections.

2.2.1 Augmentation improves generalization

The authors concluded that in general, augmentation improves the generalization ability of the classifier. After adding augmentation, performance (i.e. the AP score) of the classifier on all conditional GANs and unconditional GANs improved dramatically (at least by 10%, GauGAN improved from 67.0 → 98.1).

SAN and DeepFake were the two models where augmentation hurt performance. The authors gave the following explanations: **(1)**SAN is a super-resolution model, and only high-frequency components can differentiate between real and fake images. Augmentation tricks like blur would remove such high-frequency features, hence hurting performance. **(2)**DeepFake, on the other hand, served only '*as an out-of-distribution test case*', directly quoting the original paper [2]. Since the DeepFake model has a complex structure, it does not generate images by CNN architectures alone.

2.2.2 Augmentation improves robustness

Images from real-world applications might have undergone many unknown post-processing steps, so during the testing phase, the authors performed several post-processing steps like blurring and JPEG compression on real and fake images. They concluded that after training on augmented datasets, their model became more robust to unknown image post-processing steps.

2.2.2 Image diversity improves performance

The authors varied the number of classes in the training dataset for the classifier. They concluded that image diversity improves performance, but such improvement has a rough limit. Specifically, they trained multiple classifiers, each one on a subset of the full training dataset. When the number of training image classes increases from 2 to 16, the AP score improved consistently, but they noticed diminished improvement when the number of training image classes increased from 16 to 20. Hence they concluded that there might be an 'enough' size of the training dataset for practical generalization.

3. Methodology (of the Students' Project)

3.1. Objectives and Technical Challenges

Objectives: To construct a classifier that classifies real images and CNN-generated (fake) images.

Technical Challenges:

1. Considering the fairly large training dataset (700k+ images), we have limited computing resources. Training each epoch on a full training set takes approximately 1.5 hours. Thus, to make the best of our resources, we trained each model for 8 epochs.
2. The generic data importing mechanism of TensorFlow 2.0 does not support the training set structure we use and does not meet our needs of customizable real-time random data augmentation. Therefore, we re-design and implement a new data importing pipeline based on our requirements.
3. In test time, the TensorFlow version 2.0's default model.evaluate() does not provide average precision (AP) score and it introduces randomness in evaluation by computing the results batch by batch. As a result, we implement our test procedure by feeding in the whole test set to model and record the raw output straight out of it, thus we can compute AP manually afterwards.

3.2.Problem Formulation and Design Description

Problem Formulation: We are interested in whether the fake images generated by main-stream CNN models despite their different structures share some common flaws that could be detected by a classifier that trained on one CNN and generalize to other models.

Design Description: In our project, we construct the classifier based on Res-Net 50 pre-trained with ImageNet. On top of that, we train it as a binary classifier. In the original paper, the model is implemented using the PyTorch package while in our project, we use TensorFlow 2.0 for model realization.

In the original paper, the authors trained models with different data augmentation settings and various training set sizes. In our project, we follow the same settings to train the models (see 4.1 and 5.1 for details). In test time, we introduce images from two more sources (StyleGAN2, whichfaceisreal.com) to evaluate, adding to the original paper's 11 test sets.

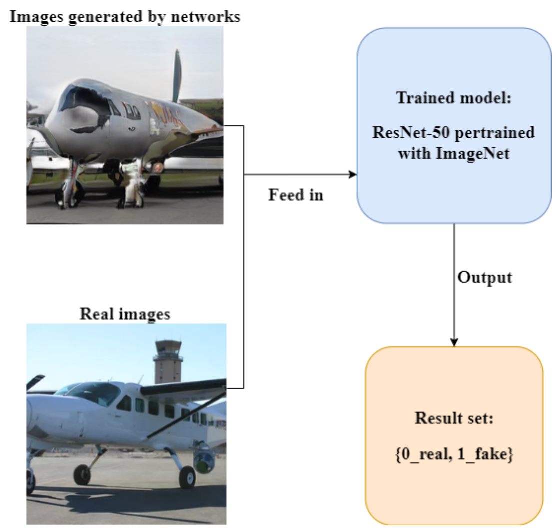


figure 1: **Research problem schematic diagram**

4. Implementation

4.1. Deep Learning Network

Architecture: The architecture of our network is shown in the diagram below. As described in 3.2, we train the model based on Res-Net 50 pre-trained with ImageNet. On top of that, we add a global average pooling layer and a fully connected layer for binary classification.

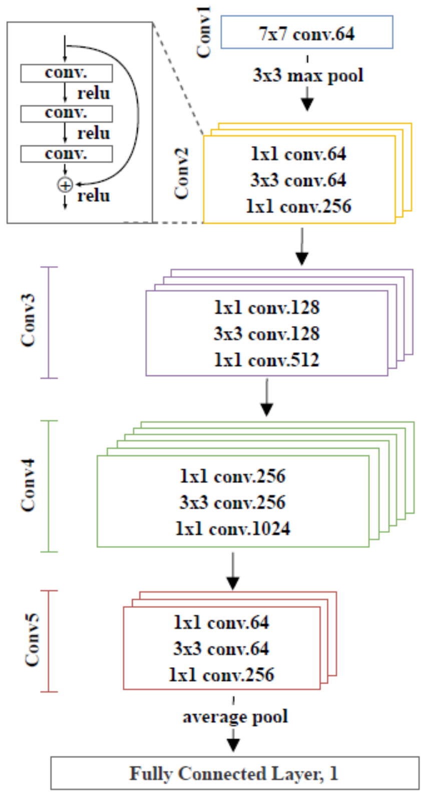


figure 2: **Network structure**

As in the plot, at each convolution stage, we have 1, 3, 4, 6, and 3 residual network blocks respectively. An example of the residual network block is shown on the top left.

Training Algorithm: In training, we choose the **Adam algorithm** as the optimizer with an initial learning rate of 1e-4. We use **binary cross-entropy** to evaluate the loss of our models. We set from_logits to be true in the loss function to output the logits values directly. During training, we save a checkpoint of the model after each epoch that can be used to restore training later.

Dataset: Following the original paper, we feed our model with a large set of real images and fake images. Fake images are generated using ProGAN, a high-performance unconditional GAN model. In the training and validation set, we assign label 0 for real images and 1 for fake images.

Data Augmentation: In training and validation, we implement data augmentation with the following methods: all images are randomly right-left flipped and cropped to 224 by 224 pixels. We implement additional data augmentation including Gaussian blur and JPEG with probability chosen by pass-in hyperparameter.

Training strategy: According to the original paper, we train a total of 7 models with different settings of real-time data augmentation and use different proportions of training sets. As described, all models are trained with images of randomly left-right flipped and cropped to 224 by 224 pixels. The first 5 models using the full training dataset are given as **(1) Baseline:** with no further data augmentation, **(2) Gaussian blur:** apply Gaussian blur of σ~Uniform[0,3] to the image with 50% probability, **(3) JPEG**: we JPEG-ed images with two libraries OpenCV and Python Imaging Library (PIL) with quality ~ Uniform{30, 31, …, 100}, **(4) Blur+JPEG (0.5)** combined (2) and (3), both with 50% probability, **(5) Blur+JPEG (0.1)**: similar to (4) but with 10% probability instead [2]. We train two additional models in data augmentation setting **(4)** as described above but with only 2 and 8 classes of the training set (total 20).

4.2. Software Design

Flow charts: The top-level flow chart is given below.

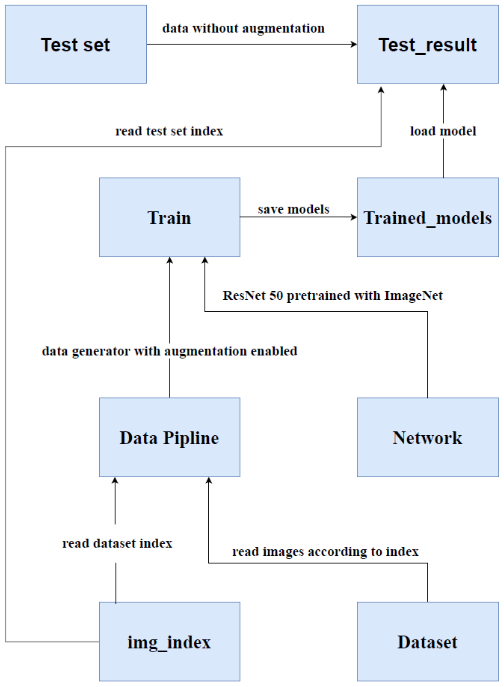


figure 3: **Top-level software design**

A more detailed structure for each module is shown here.

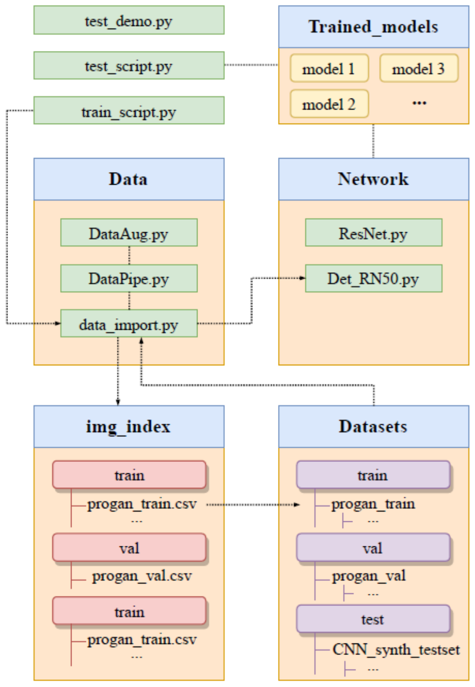


figure 4: **Detailed module structures**

Algorithms/Pseudocode: (link provided)

Data pipeline module: In this module, we implement our data pipeline that reads from the dataset as well as image index and returns a data generator compatible with TensorFlow's fit method.

In [DataAug.py](), we perform a random crop, random left-right flip, Gaussian blur, and JPEG to images passed in.

```
Algorithm 1: DataAug.py
  Input: image img, a dictionary of augmentation probabilities opt
  Output: Augmented image
  img ← array(img) ;
  img ← random_crop(img) ;
  img ← random_hflip(img);
  img ← gassian_blur(img,opt[blur_prob], $\sigma \sim [0, 3]$;
  img ← jpeg(img, opt[jpeg_prob], method ← CV2, quality ← $[30, 100]$) ;
  return img
```

In [DataPipe.py](), we pass in an image index table and dataset directory and return a data generator that is compatible with TensorFlow's Model.fit().

```
Algorithm 2: DataPipe.py
  Input: image index csv file idx, dataset directory dir,
     batch size batch, augmentation options opt
  Output: a data generator gen
  initialize gen ← tf.sequence object;
  gen.batch_size ← batch ;
  gen.index ← idx ;
  gen.data ← read_img_from_csv(idx, dir) ;
  return gen
```

In [data_import.py](), the function returns a training data generator and a validation data generator provided the directory of training and validation set as well as the image index associated.

```
Algorithm 3: data_import.py
  Input: training index csv file train_idx, training directory train_dir,
     validation index csv file val_idx, validation directory val_dir, batch size batch
  Output: data generators training_gen, val_gen
  train_idx ← train_idx.shuffle();
  val_idx ← val_idx.shuffle();
  train_gen ← DataPipe(train_idx, train_idx, batch);
  val_gen ← DataPipe(val_idx, val_idx, batch);
  return train_gen, val_gen
```

Network module:
In [Det_RN50.py](), we load ResNet-50 pre-trained with ImageNet. We do not include top layers of ResNet-50 in our model. Instead, we put a global average pooling and a fully connected layer with 1 node on the top of that.

```
Algorithm 4: Det_RN50.py
  Output: a ResNet-50 network in binary classification setting model
  base = ResNet50(weights ← imagenet, include_top ← False);
  x ← GlobalAveragePooling2D()(x);
  prediction ← Dense(1)(x);
  model = tf.Model(base.input, predictions);
  return model;
```

Training module:
In [train_script.py](), we pass in the training parameters and train our model. For a complete list of parameters, please refer to the pseudocode below.

```
Algorithm 5: train_script.py
  Input: training set directory, train_dir, training image index, train_idx,
     validation set directory, val_dir, validation image index val_idx,
     restore checkpoint ckpt, Gaussian blur probability blur, JPEG probability jpeg,
     starting epoch start, epoch epoch, batch size batch
  Output: trained model model
  train_gen, val_gen ← data_import( train_dir, train_idx, val_dir,val_idx );
  model = Det_RN50();
  model.load(ckpt);
  model.compile(Adam, BinaryCrossentropyLoss);
  model.fit(train_gen, val_gen, batch, start, epoch);
  return model
```

Test module:
In [test_script.py](), we pass in the model checkpoint, image index, test set directory, and batch size and return the evaluation of the model on the test set.

```
Algorithm 6: test_script.py
  Input: test set directory, test_dir, test image index, test_idx,
     restore checkpoint ckpt, batch size batch
  Output: trained model model
  test_gen ← data_import( test_dir, test_idx);
  model = Det_RN50();
  model.load(ckpt);
  model.compile(Adam, BinaryCrossentropyLoss);
  evaluation ← model.evaluate(test_gen, batch, start, epoch);
  return evaluation
```

Model name	Training settings			Individual test generators													Total
	Training set	Augments		Pro-GAN	Style-GAN	Big-GAN	Cycle-GAN	Star-GAN	Gau-GAN	CRN	IMLE	SITD	SAN	Deep-Fake	Style-GAN2	WFIR	mAP
		Blur	JPEG														
model-2c	2-class	0.5	0.5	97.6	82.3	66.4	82.8	85.5	88.1	96.2	97.9	79.2	58.1	61.7	81.1	99.1	82.2
model-8c	8-class	0.5	0.5	100.	95.2	68.3	87.1	100.	63.4	98.4	95.3	97.5	87.1	94.8	95.4	94.0	90.5
model1	20-class	N/A	N/A	100.	96.6	72.1	76.0	**100.**	59.3	97.8	94.1	98.9	**91.1**	**96.4**	**99.8**	97.5	**90.7**
model2	20-class	0.5	N/A	100.	96.5	76.7	84.4	**100.**	65.1	92.6	93.8	**99.6**	49.1	67.3	99.4	98.9	86.4
model3	20-class	N/A	0.5	100.	96.2	74.7	86.1	95.9	90.3	**99.4**	**99.6**	85.0	69.3	88.5	94.6	95.3	90.4
model4	20-class	0.5	0.5	100.	**97.7**	**78.1**	**91.9**	98.2	**92.4**	96.7	98.1	99.2	56.2	65.9	98.5	**99.7**	90.2
model5	20-class	0.1	0.1	100.	93.6	72.1	81.7	99.2	73.1	87.7	81.5	98.8	54.5	68.0	97.4	97.5	85.0

Table 2: **Cross-dataset generalization results from our model.**

5. Results

5.1. Project Results

Test procedure: Following the original paper, we evaluate our models' performance on the test set using average precision (AP) by computing this score separately on each test set. As for test set selection, we include all 11 datasets used in the original paper, together with images from 2 extra sources (StyleGAN2, whichfaceisreal.com), to conduct comparison as well as further generalization ability tests. During testing, for each image set from the 13 test sets mentioned above, we collect their raw scores predicted by our 7 different models (trained with different settings) and calculate the AP score. Speaking of test settings, we apply no data augmentation during test-time, per the original paper.

Results (by AP scores) from our overall 91 tests (7 models x 13 test sets) are all listed in table 2. In this table, we compare the generalization ability of models trained with different training settings as the original paper did. To begin with, we discover that the increment in training set diversity (i.e. the number of classes we include) does help improve the general performance of the resulted model, as mentioned in the paper. Furthermore, we also observe that, once the amount of training data is settled, using rather aggressive train-time data augmentation can significantly improve the resulting model's generalization ability as well as average precision on most of our test sets, similar to the original paper.

Augmentation (usually) improves generalization In accordance with the paper, we first trained 5 models with different train-time data augmentation implementations. For starters, we trained our baseline model without any augmentation (model 1). As it was found in the paper, we observe that our baseline model also works perfectly on the held-out ProGAN test set (100.0 AP). Take one step further, we can clearly see that our baseline model claims a very impressive generalization ability across most of our test sets, like StyleGAN, StarGAN, CRN, etc.

When augmentations are added during train-time, the performance on test images generated from conditional GAN models like CycleGAN and GauGAN are significantly improved, 76.0 → 91.9, 59.3 → 92.4, respectively. For test sets that already had a decent score with our baseline model, some are slightly improved with the implementation of augmented data, like StyleGAN, 96.6 → 97.7. However, it is very interesting that certain test sets actually get a much worse score with models trained with data augmentation, like SAN and DeepFake, 91.1 → 49.1, 96.4 → 65.9, per the paper's argument.

Diversity promotes performance Similar to the paper, we also train our classifier with varied settings of the number of classes in the train set, while applying the same amount of data augmentation as we did with model 4 (Blur 0.5 & JPEG 0.5). As can be seen in table 2, we notice an obvious performance improvement as we increase the train set's diversity. Most of our test sets gain instant better performance after we expand the classes contained in the train set from 2 to 8. Moreover, although there wasn't enough time and hardware resources for us to repeat all tests implemented in the paper, we can still see the improvement tend to slope off after we have about 8 classes in the train set (see figure 8).

In addition, while for most results we are getting an almost identical trend to the paper as expected, some AP scores we get with our model are even higher than the original paper. We'll discuss the comparison with the original results later in section 5.2 and 5.3.

5.2. Comparison of the Results Between the Original Paper and Students' Project

Here from table 3 to table 9, figure 5 to figure 8 we list test set AP scores acquired through all our 7 different models and their corresponding original results from the paper side by side. In tables, we highlight the better side in black where the scores from our model are obviously better or worse than that of the original paper (difference greater than 4 numerically).

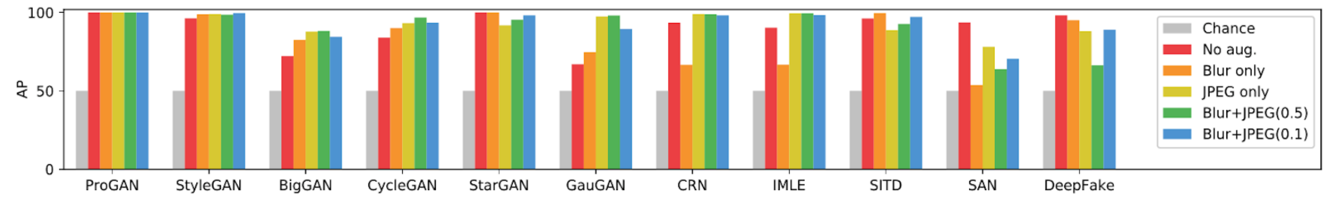


figure 5: **Effect of augmentation** (from the paper)

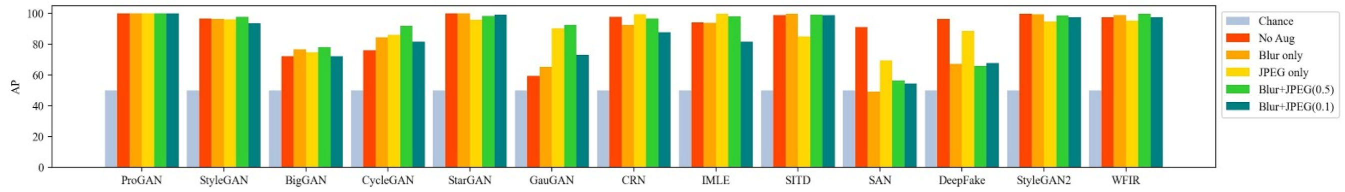


figure 6: **Effect of augmentation** (generated by us)

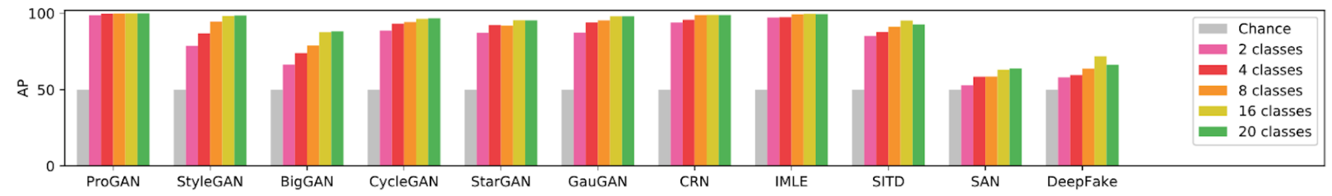


figure 7: **Effect of data diversity** (from the paper)

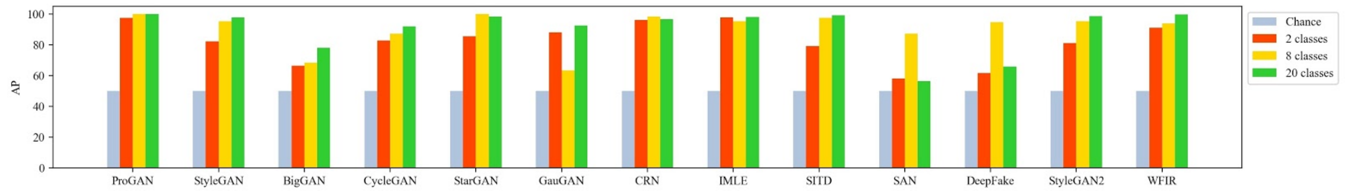


figure 8: **Effect of data diversity** (generated by us)

Training settings	**Training set:** 2-class Pro-GAN	
	Augmentation Blur: 0.5 JPEG: 0.5	
Individual test generators	**Average Precision**	
	Original	Ours
ProGAN	98.8	97.6
StyleGAN	78.3	**82.3**
BigGAN	66.4	66.4
CycleGAN	**88.7**	82.8
StarGAN	87.3	85.5
GauGAN	87.4	88.1
CRN	94.0	96.2
IMLE	97.3	97.9
SITD	**85.2**	79.2
SAN	52.9	**58.1**
DeepFake	58.1	61.7
StyleGAN2	N/A	81.1
WFIR	N/A	99.1
Total mAP	81.3	82.2

Table 3**: model-2c comparison** (trained on 2-class proGAN)

Training settings	**Training set:** 8-class Pro-GAN	
	Augmentation Blur: 0.5 JPEG: 0.5	
Individual test generators	**Average Precision**	
	Original	Ours
ProGAN	99.9	100.0
StyleGAN	94.2	95.2
BigGAN	**78.9**	68.3
CycleGAN	**94.3**	87.1
StarGAN	91.9	**100.0**
GauGAN	**95.4**	63.4
CRN	98.9	98.4
IMLE	99.4	95.3
SITD	91.2	**97.5**
SAN	58.6	**87.1**
DeepFake	98.2	94.8
StyleGAN2	N/A	**95.2**
WFIR	N/A	**94.0**
Total mAP	87.9	90.5

Table 4: **model-8c comparison** (trained on 8-class proGAN)

Training settings	**Training set:** 20-class Pro-GAN **Augmentation** Blur: N/A JPEG: N/A	
Individual test generators	**Average Precision**	
	Original	Ours
ProGAN	100.	100.
StyleGAN	96.3	96.6
BigGAN	72.2	72.1
CycleGAN	**84.0**	76.1
StarGAN	100.	100.
GauGAN	**67.0**	59.3
CRN	93.5	**97.8**
IMLE	90.3	94.1
SITD	96.2	98.9
SAN	93.6	91.1
DeepFake	98.2	96.4
StyleGAN2	N/A	99.8
WFIR	N/A	97.5
Total mAP	90.1	90.7

Table 5: **model1 comparison** (trained with no augmentation)

Training settings	**Training set:** 20-class Pro-GAN **Augmentation** Blur: 0.5 JPEG: N/A	
Individual test generators	**Average Precision**	
	Original	Ours
ProGAN	100.	100.
StyleGAN	99.0	96.5
BigGAN	**82.5**	76.7
CycleGAN	**90.1**	84.4
StarGAN	100.	100.
GauGAN	**74.7**	65.1
CRN	66.6	**92.6**
IMLE	66.7	**93.8**
SITD	99.6	99.6
SAN	53.7	49.1
DeepFake	**95.1**	67.3
StyleGAN2	N/A	99.4
WFIR	N/A	98.9
Total mAP	84.4	86.4

Table 6: **model2 comparison** (blurriness is applied with 50% probability, no JPEG compression applied)

Training settings	**Training set:** 20-classPro-GAN **Augmentation** Blur: N/A JPEG: 0.5	
Individual test generators	**Average Precision**	
	Original	Ours
ProGAN	100.	100.
StyleGAN	99.0	96.2
BigGAN	**87.8**	74.7
CycleGAN	**93.2**	86.2
StarGAN	91.8	**95.9**
GauGAN	**97.5**	90.3
CRN	99.0	99.4
IMLE	99.5	99.6
SITD	88.7	85.0
SAN	**78.1**	69.3
DeepFake	88.1	88.5
StyleGAN2	N/A	94.6
WFIR	N/A	95.3
Total mAP	93.0	90.4

Table 7: **model3 comparison** (JPEG compression is applied with 50% probability, no blurriness is applied)

Training settings	**Training set:** 20-class Pro-GAN **Augmentation** Blur: 0.5 JPEG: 0.5	
Individual test generators	**Average Precision**	
	Original	Ours
ProGAN	100.	100.
StyleGAN	98.5	97.7
BigGAN	**88.2**	78.1
CycleGAN	**96.8**	91.9
StarGAN	95.4	98.2
GauGAN	**98.1**	92.4
CRN	**98.9**	96.7
IMLE	**99.5**	98.1
SITD	92.7	**99.2**
SAN	**63.9**	56.2
DeepFake	66.3	65.9
StyleGAN2	N/A	98.5
WFIR	N/A	99.7
Total mAP	90.8	90.2

Table 8: **model4 comparison** (Both JPEG compression and blurriness are applied with 50% probability)

Training settings	**Training set:** 20-class Pro-GAN	
	Augmentation Blur: 0.1 JPEG: 0.1	
Individual test generators	**Average Precision**	
	Original	Ours
ProGAN	100.	100.
StyleGAN	**99.6**	93.6
BigGAN	**84.5**	72.1
CycleGAN	**93.5**	81.7
StarGAN	98.2	99.2
GauGAN	**89.5**	73.1
CRN	**98.2**	87.7
IMLE	**98.4**	81.5
SITD	97.2	98.8
SAN	**70.5**	54.5
DeepFake	**89.0**	67.7
StyleGAN2	N/A	97.4
WFIR	N/A	97.5
Total mAP	**92.6**	85.0

Table 9: **model5 comparison** (Both JPEG compression and blurriness are applied with 10% probability)

5.3. Discussion of Insights Gained

Amazingly well generalization ability To verify results from the paper, we made the best use of our time and ran nearly 100 independent tests in total on 7 models, which covers most scenarios mentioned by the author. Based on what we learned from our test scores, we can confirm that the network structure and training strategy proposed by the author works really well, as described in their paper, and showed amazing generalization ability. Without any additional fine-tuning, these models present almost perfect performance on our two extended test sets, StyleGAN2 and WFIR, 99.8, 99.7, respectively. With the relatively limited time and resources we had, we are able to get quite good results without any tweaks or tricks, and our model works even better than the original on some test sets following the same training procedure stated in the paper.

Randomness introduced in training While the overall total mAPs we are getting from different train settings are very close to the corresponding original results, and the way our results shifted among models are similar to the paper as can be seen from figure 5 and figure 8, we do observe some noticeable differences on several datasets when implementing certain models. From comparison tables in section 5.2, we can clearly pick up that, for models with less aggressive data augmentation like model1, the AP we got from our tests are almost identical to the paper on nearly every test set. When it comes to models with heavier data augmentation implementations, the outcome fluctuates more often and our results are usually slightly worse than the paper's. Since the data augmentation process is highly randomized, where we only set the probability of applying blur/JPEG and the range in which their parameters can vary, it is reasonable that this randomness introduced in training eventually leads to most of the differences we are looking at. It is highly likely that the authors of the paper trained their models multiple times and ran more epochs in each training so that they can let randomness do its job and select the best model possible. On the other hand, we had limited time and had to run as many different train and test experiments as possible, so we ended up getting the correct result instead of the best result for train settings with more randomness.

6. Conclusion

In this work, we re-implement the paper *"CNN-Generated Images Are Surprisingly Easy to Spot... for Now"* (Wang et al., 2020 [2]) using TensorFlow 2.0 framework. We verify most of the experiment results in the original paper by developing 7 different models and testing them on the original paper's 11 test data sets provided. Despite some random fluctuations, we confirm the remarkable solidness of the original paper's work on detecting CNN-generated fake images. Given the limited time and computing resources, our models achieve an average precision of over 90% on most test sets generated by the main-stream networks which match up with the result of the original paper. If time and computational power permit, the results could be further improved.

Moreover, we test on extended data sources (StyleGan2, whichfaceisreal.com) and find the model generalizes well on the new test set. On these datasets, we manage to achieve nearly-perfect 99.8 and 99.7 average precision respectively. We think that these outstanding results on extended data sources indicate the great potential of the model structure and training strategy brought up by the original paper. All in all, our results suggest that CNN-generated images still share some common detectable traits that allow us to generalize from one model to others. However, though we showed our method is robust with current main-stream CNN models and certain levels of data augmentation, the generalization of the model to other fake image data sources and data transformation is still in question. In the future, we think more work can be done in evaluating the effect of other more aggressive data transformations such as resizing.

Last, we note that an effective model in detecting fake images can be used to fight the increasing threat of abusing computer image techniques. We suggest that in the future more work is needed to evaluate the detection models in real-life scenarios. We hope this work can shed some light on solving social problems caused by technology misuses such as social media disinformation.

7. Acknowledgement

We'd like to thank Sheng-Yu Wang and his team from UC Berkeley for a detailed description of their methodology in the paper and provided clear results and plots to follow. Also, we would like to thank the TAs and Professor of this course for helping us and answering our questions on Piazza efficiently.

8. References

[1] B. Feng, J. Liu, J. Yang, GitHub repository, https://github.com/ecbme4040/e4040-2020fall-project-acnn-jl5742-bf2477-zy2431
[2] S. Wang et al., "CNN-generated images are surprisingly easy to spot... for now", UC Berkeley, *CVPR 2020*
[3] What face is real? https://www.whichfaceisreal.com/
[4] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In ICLR, 2018. 1, 2, 3, 4
[5] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In CVPR, 2019. 1, 2, 3
[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In ICLR, 2019. 1, 3
[7] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In ICCV, 2017. 1, 3
[8] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In CVPR, 2018. 1, 3
[9] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In CVPR, 2019. 1, 3
[10] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In ICCV, 2017. 1, 3
[11] Ke Li, Tianhao Zhang, and Jitendra Malik. Diverse image synthesis from semantic layouts via conditional imle. In ICCV, 2019. 1, 3
[12] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. Learning to see in the dark. In CVPR, 2018. 1, 3
[13] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In CVPR, 2019. 1, 2, 3
[14] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. FaceForensics++: Learning to detect manipulated facial images. In ICCV, 2019. 1, 2, 3, 5, 6

9. Appendix

9.1 Individual Student Contributions in Fractions

	jl5742	bf2477	zy2431
Last Name	Liu	Feng	Yang
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1 **Prework**	Large dataset management	Cloud server management	Literature review
What I did 2 **Code**	Train module, network modeling	Test module, data pipeline github repo structure	Dataset structure and generation
What I did 3 **Report**	Report writing (section 3, 4, 6)	Report writing (section 5, 6)	Report writing (section 1, 2, 7)