SGANA
Selective Generative Adversarial Network Augmentation for fine-grained classification

Fausto Conigliaro[*]
s243958
Polytechnic of Turin
s243958@studenti.polito.it

Ivan Murabito[*]
s263138
Polytechnic of Turin
s263138@studenti.polito.it

Matteo Stoisa[*]
s265542
Polytechnic of Turin
s265542@studenti.polito.it

ABSTRACT

Convolutional Neural Networks for classification purposes are ubiquitous nowadays. The most common challenge resides in obtaining a compelling dataset to solve the given problem. We were inspired by the recent progresses on Generative Adversarial Networks to propose and evaluate a new data augmentation technique based on the exploitation of sample generation for fine-graining scenarios, in order to increase the classification performance on smaller datasets. In our work, first of all we evaluated several generative approaches and qualitatively selected the best for our problem. Then we tested different implementations of our strategy on a selection of popular and state-of-the-art fine-grained classifiers, performing an in-depth analysis of the influence of the added samples. We carried out our analysis on Stanford Dogs Dataset.

1. INTRODUCTION

Classification is one of the main applications of artificial intelligence, and dataset size is a critical factor in deep learning effectiveness. Data augmentation is often used as strategy to increase the size of the training set. We think generative-based approaches to data augmentation are interesting because they could improve performance, especially on small datasets and fine-graining problems. To test our hypothesis, we will try to use a GAN in order to generate new artificial samples to reinforce the fine-graining process and to improve the classification results.

2. BACKGROUND

As starting point for our work we studied the scientific literature of this specific field, in order to choose the most appropriate neural networks and dataset for our purpose.

2.1. Generative Adversarial Networks

A GAN (Generative Adversarial Network) is a system composed by two neural networks that train each other. The target is to generate artificial samples starting from a training dataset. Generated samples are domain-similar to the train samples, so that it is difficult to distinguish between real and fake ones. Nowadays many variants have been developed but the original version of GAN, proposed by Ian J. Goodfellow [1] in 2014, is simply composed by a Generator (G) and a Discriminator (D):

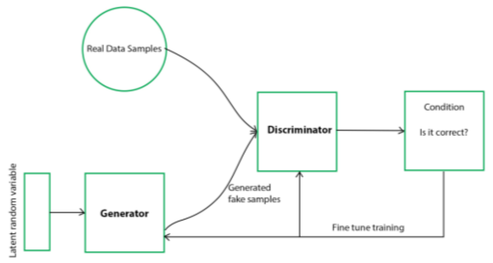


Figure 1: Schematic architecture of generic GANs

G generates an image (x) starting from an input noise variable (z) and processes it with a differentiable function represented by a multilayer perceptron. D receives an image from G's output and from the training dataset, its scalar output represents the probability that an image is real rather than fake.

D is trained in order to maximize the probability of assigning the correct label, G is antagonistically trained to minimize D's success. This is a min-max optimization problem of D and G:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

Network updates are performed using the stochastic gradient descent technique. The training procedure is composed by k epochs in which G generates an image (x) and D classifies x and many train-set images, after that networks weights are properly updated. At the end of a successful training, the GAN can generate plausible images that are similar to the training domain, various techniques have been developed to quantify both the plausibility and the proximity to the training dataset. In the last few years, the most used algorithms for this scope are Inception Score (IS) [2] and his variant Fréchet Inception Distance (FID)

[*] Equal contributor

[3]. Both use Inceptionv3 model (a pre-trained deep learning neural network model) in order to evaluate how much generated images look like a specific object (quality) and if there is a wide range of object generated (diversity: images must not be too similar to each other), FID also calculates Fréchet distance between statistic distributions of the two datasets. IS score has a range between 1 and the number of possible object classes (the best score), FID contrarily is better when score is low (0 is the best).

We decided to test the implementation of three different GANs from scratch:

- Original GAN [1]
- AC-GAN, Auxiliary Classifier GAN [4] contains in its architecture a classifier optimized to distinguish between the classes of the train in order to increase the quality of output samples.
- BigGAN [5] is focused on enlarging the GAN architecture introducing several changes:
  - Self-Attention module: as introduced in SA-GAN [6], they applied an attention map to images before passing them to D and G so that backgrounds are less relevant. The attention module has a loss function and it is updated with backward propagation too.
  - Larger batch size compared to the previous GANs
  - Many parameters: they doubled the number of parameters in both models increasing the width in each layer.
  - Truncation trick: they introduced the possibility of truncating the gaussian distribution of vector z in order to trade-off between image quality or fidelity and image variety.

2.2. Standard CNN Classifiers

A classifier is a system that starting from a sample gives the probability that the sample belongs to a certain class. A Convolutional Neural Network is basically composed by multiple layers of different kind:

- Convolutional layer: a filter that scans the whole image, it creates a feature map to predict the class probabilities for each feature
- Pooling layer: following a convolutional layer, it scales down the amount of information of features maintaining only the most important ones
- Fully connected input layer: “flattens” the outputs generated by previous layers turning them into a single vector, then passes it to other fully connected layers
- Fully connected layer: contains and applies weights over the input vector received
- Fully connected output layer: gives the final probabilities to determine the class of the input image

The performance of a classifier is quantified with the accuracy score: the percentage of correct predictions given. Since we would like to test our approach on different topologies and since we noticed they are well established as classification benchmarks in many papers, we selected the following classifiers:

- AlexNet [7], a convolutional neural network composed by five convolutional layers, three fully connected layers and non-saturating ReLU activation function.
- VGG11 [8], composed by 11 convolutional layers and uses 3x3 convolution filters.
- DenseNet121 [9], it consists of 4 dense blocks whose output is forward fed to each subsequent block. In this way for each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.

2.3. The State-of-the-Art fine-grained classifier

As final benchmark for our analysis, we utilized the State of the Art for image fine-grained classification: the WS-DAN: Weakly Supervised Data Augmentation Network [10]. The key feature of this neural network are attention maps, used to selectively apply data augmentation techniques known as attention cropping and attention dropping. In the original paper it scored 92% on Stanford Dogs Dataset.

Because of his computational heaviness, we used this network only for an initial test and for the final test with the optimal conditions we found on the other topologies.

3. DATASET

Since our goal was to test a data augmentation technique for fine-graining purposes, we decided to use a well-established dataset in this field. We chose the Stanford Dogs Dataset [11], a subset of ImageNet dataset [12]. It is composed by more than 20 thousand dog images divided into 120 different classes (the dog breeds). Each class contains approximately 150 images.

Figure. 2: Most of the Stanford Dogs Dataset images display the subject optimally.

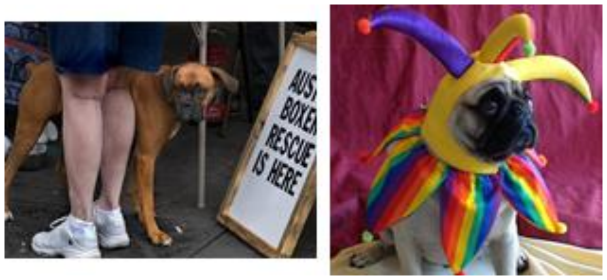


Figure 3: Other images include extensive disturbing elements, nevertheless we utilized the whole dataset without excluding the outsiders.

4. APPROACH

First, our aim was to generate the augmented dataset, to later consistently evaluate the performance of our proposed technique.

As previously anticipated, we trained from scratch three GAN implementations:

- Standard PyTorch GAN [13]: trained for 100 epochs with a batch size of 32 images
- PyTorch AC-GAN [13]: trained for 100 epochs with a batch size of 32 images
- Kaggle AC-GAN implementation [14]: trained for 100 epochs with a batch size of 32 images

We used as computational tool both Google Colab and a 1080 GeForce GPU locally available. In this three tries we analyzed the behavior of the loss functions of the discriminators and generators (and classifier in AC-GAN) making sure they had proper trends. Qualitative inspection of generated samples gave unsatisfactory results.

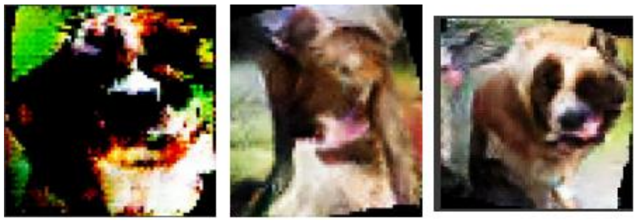


Figure 4: Example of images generated with standard GAN implementations: PyTorch GAN (Left), PyTorch AC-GAN (Center), Kaggle AC-GAN (Right)

However, increasing number of epochs was unfeasible due to the limitations of the training time and our environment. Moreover, all these implementations' outputs were limited to 64x64, not optimal for our purposes.

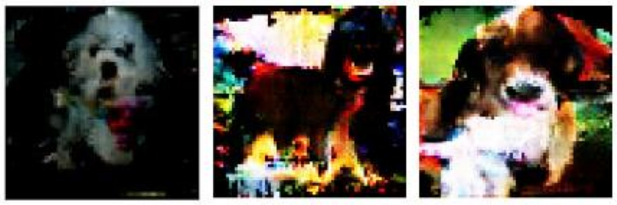


Figure 4: Some more examples of unsatisfactory images generated with standard GAN implementations

Since our goal was not to build an optimized GAN but to utilize artificial images as data augmentation, we decided to look for a better out-of-the box approach and found an open source implementation of BigGAN in PyTorch [15]. It is pretrained and optimized on ImageNet dataset. It can generate artificial images of one of each 1000 possible classes, including all dog classes we need for our purpose. It was able to produce appropriate images per for our scope. Indeed, it is able to generate up to 512x512 pixels images and qualitatively the generated images are quite realistic.

4.1. Augmented Dataset

Using the BigGAN with truncation 0.9 we generated our artificial dataset: 250 images per class with resolution 256x256, dimensionally comparable to the Stanford Dogs Dataset. The quantity of generated images to add to the standard dataset is one of the parameters we will analyze. A more in-depth qualitative analysis revealed that dogs are, on average, impressively realistic:

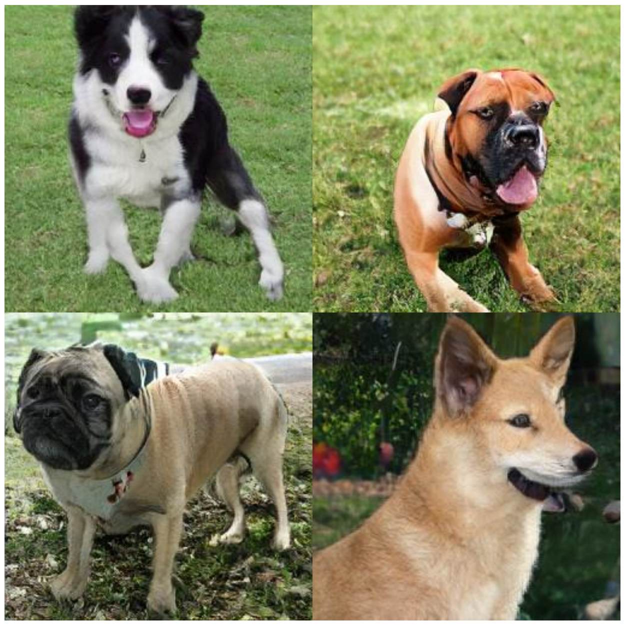


Figure 5: Examples of realistic dog images created by the BigGAN.

Other dogs appear quite distorted, but they maintain main characteristics such as color and shape so that they still could be useful to improve the classifier training:

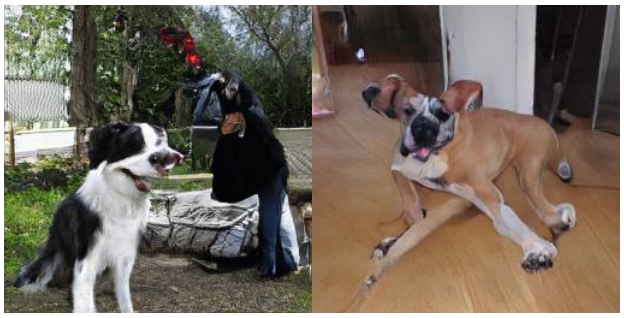


Figure 6: Examples of distorted dog images created by the BigGAN.

In other cases, background or objects near the dogs cover most of the image and are very distorted, this could negatively impact on the classifier training:

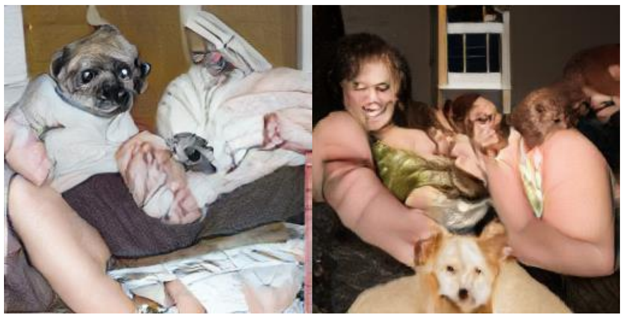


Figure 7: Examples of misleading images created by the BigGAN, anyway we considered also this kind of outsiders.

We generated 250 dogs per class, this is the artificial dataset from which in following steps we will build the augmentation dataset concatenating part of it with the stock train set. Since we will not be able to perform cross-validation due to time and environment restrictions, we decided to always use the same samples and splits to have comparable results afterwards.

4.2. Preliminary classification setup

We tested AlexNet, VGG11, DenseNet121 and the WS-DAN with baby-sitting search of learning rate (0.01, 0.001, 0.0001) and optimizer (SGD, Adam, Ada) on a validation subset of the Stanford Dogs Dataset. For all three networks implementation we used PyTorch open-source code [16] [17] [18] [19], pre-trained on ImageNet. We found out that SGD gave consistently better results on all the topologies and that the learning rates indicated in the table below were the best ones for performance and runtime, so we decided to only use these hyperparameters for all the following experiments.

	AlexNet	VGG11	DenseNet121	WS-DAN
LR	0.01	0.01	0.01	0.001
Epochs	20	10	10	10
Step size	15	7	7	7
Batch size	256	64	32	32
Optimizer	SGD	SGD	SGD	SGD

Table 1: Hyperparameters selected for each of the network used in following tests

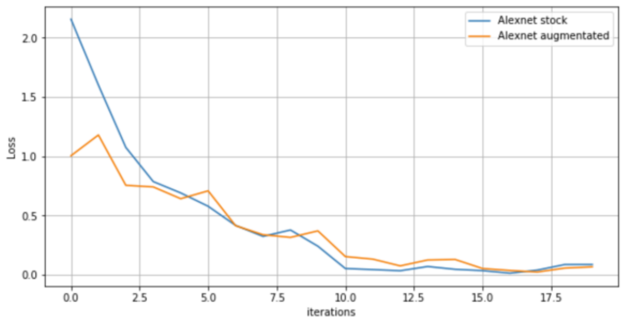
4.3. Baseline evaluations

We proceeded to do a first test of our strategy: we trained AlexNet, VGG11 and Densenet121.

We split the Stanford Dogs Dataset into train set (80%) and test-set (20%), we performed a training procedure with the stock train set (16k total samples) and with the stock train set concatenated with the full augmentation dataset (46k total samples). We qualitatively inspect the behavior of the loss functions and made sure they were appropriate. We achieved the following accuracy results:

	Stock Dataset	Augmented Dataset
AlexNet	63.99%	63.46%
VGG11	76.48%	74.98%
DenseNet121	78.09%	(out of memory)
WS-DAN	81.37%	79.23%

Table 2: Accuracy scored using the whole augmentation dataset



Graph 1: Example of the proper trend of loss for the same classifier applied to stock and augmented dataset

The accuracy scores decreased. In our reckoning one of the reasons could be that the generated dogs were much more than the original ones: an increase of roughly 200% of samples. Another possible reason was the different aspect ratio between the original and the generated images. We realized it during the qualitative inspection of the train images, and that it could be imputable to the fact that the transformations gave different results on the original and the generated samples.

We elaborated a new set of transforms to make the aspect ratio uniform and repeated the tests with half of the augmentation dataset (30k total samples). For the test and validation set we didn't use the random transforms.

Previous Transforms	New Transforms
resize(256x256)	resize(256)
random_horizontal_flip (p=0.5)	random_horizontal_flip (p=0.5)
center_crop(224)	random_apply ([color_jitter], p=0.3)
random_apply ([color_jitter, random_rotation], p=0.3)	center_crop(224)
to_tensor()	to_tensor()
Normalize ((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))	Normalize ((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))

Table 2: New transforms to achieve comparable samples

We achieved the following scores:

	Stock Dataset	Augmented Dataset
AlexNet	64.84%	66.23%
VGG11	77.45%	76.46%
DenseNet121	79.10%	80.81%
WS-DAN	81.37%	81.27%

Table 3: Accuracy scored using the half augmentation dataset

Since the accuracy scores were better than the previous ones and the new transformations proved qualitatively to be successful in providing uniform samples, we decided to focus on the impact of the number of augmented samples as a percentage of the stock dataset and to try to optimize it as an hyperparameter in a new validation run.

4.4. Optimization of Augmentation factor

During this phase we realized it could be important to make sure that the generated samples we introduce are uniformly drawn per-class instead of randomly. To do that we decided to implement the augmentation function so that the same amount of samples would be added for each class. Since on average there are 120 dogs per class in our stock training set, we defined the augmentation factor as:

$$A_f\% = \frac{100 * \#dogs\ added}{\#average\ dogs\ per\ class} = 100\frac{\#dogs\ added}{120}$$

and decided to use it to refer to the number of samples added, instead of specifying it as percentage of the whole augmentation dataset.

To have more insights about the influence of the optimization of this parameter we decided to compute and plot the accuracy per class in addition to the overall one. We created a new split of the stock dataset dividing it into training set (70%), validation set (15%) and test set (15%), in order to optimize the A_f.

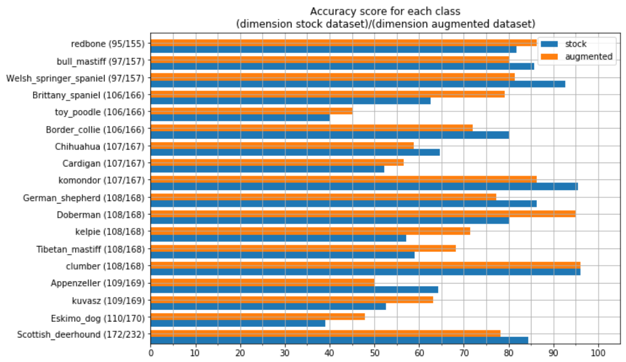
We achieved the following scores:

	AlexNet	VGG11	DenseNet121
Stock Dataset	63.20%	75.96%	77.66%
Aug. 25%	64.69%	75.83%	78.69%
Aug. 50%	64.66%	75.64%	77.29%
Aug. 75%	64.82%	76.15%	76.71%
Aug. 100%	65.44%	74.70%	77.62%

Table 4: Accuracy scored with different percentage A_f

The accuracy scores are similar, but the per-class analysis revealed a huge variance on many per-class $\Delta_{accuracy}$ (up to 30% positively and negatively).

$$\Delta_{accuracy} = accuracy_{\substack{augmented\\dataset}} - accuracy_{\substack{stock\\dataset}}$$



Graph 2: Example of accuracy per-class analysis, extracted from DenseNet121 with 50% augmentation

Further analysis on per-class delta accuracy showed that most of accuracy trends were consistent among $A_f s$. However they were not consistent among the networks.

Class	25%	50%	75%	100%	Mean
Standard schnauzer	+23.8	+28.6	+14.3	+23.8	+22.6
Norfolk terrier	+23.5	+11.8	+17.6	+23.5	+19.1
soft-coated wheaten terrier	+22.7	+13.6	+0.0	-9.0	+6.8
Chesapeake Bay retriever	+22.6	+25.8	+22.6	+25.8	+24.2
…		…			
Papillon	-15.6	-6.25	-9.38	-3.13	-8.6
Schipperke	-18.2	-9.1	-9.1	-9.1	-11.4

Siberian husky	-18.2	-9.1	-12.1	-6.1	-11.4
Standard schnauzer	-23.8	-28.6	-14.3	-23.8	-22.6

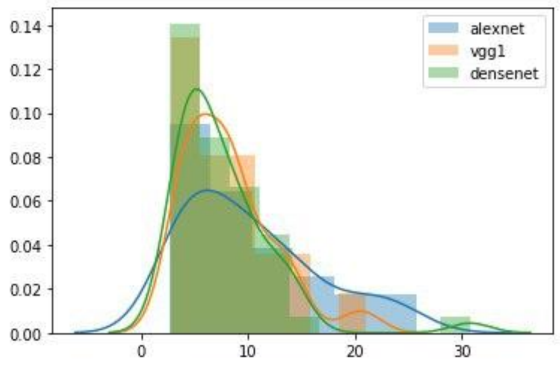
Table 5: Example of per-class $\Delta_{accuracy}$ analysis, extracted from AlexNet. Reported the top 4 positive and top 4 negative $\Delta_{accuracy}$.

Starting from these observations, we decided to only add the samples of the classes which proved to be beneficial in validation phase. If our proposed technique is working in our framework, we should see a clear improvement in the average accuracy on the test set.

4.5. Selective GAN Augmentation

We decided to do a final assessment using the best A_f found in the previous process for each network and see if the networks trained on the selectively augmented dataset were performing better on test set.

We selected only the classes which improved their accuracy in validation, regardless of A_f and for each network. We called the percentage of classes selected for augmentation $A_{c.}$



Graph 3: Frequency distribution of positive per-class $\Delta_{accuracy}$ for each run, with 100% augmentation

These are the results:

	A_f	A_c	Stock	SGANA
AlexNet	100%	50%	65.01%	65.31%
VGG11	75%	46.7%	76.90%	77.31%
DenseNet121	25%	35%	78.06%	78.96%
WS-DAN	100%	42.5%	81.36%	81.41%

Table 6: Accuracy obtained on the test set

5. CONCLUSIONS

The accuracy results we obtained on the test set show a slight improvement on the performance of the classifiers. However we think the rate of improvement does not justify the workload needed to use this method since generation is a very computationally intensive and time-consuming task. Moreover pretrained models are generally not available.

Our results are affected by great uncertainty since we were not able to cross-validate them, due to the extensive runtime and the limitations of our environment. We had to make massive compromises on the choice of some hyperparameters, such as batch size because of constrained memory, number of epochs and validation granularity due to required execution time and online computation time restrictions.
For the same reasons we were not able to repeat the same analysis on some other well-known fine graining datasets.

Tests were performed using image resolution 256x256, our approach could be more effective on different problems, probably in lower-resolution applications.

Images were generated using BigGAN with truncation 0.9 as it produced samples with an optimal trade-off between variety and fidelity. Further analysis can be done for the impact of varying this hyperparameter, as classifiers could benefit from increased variety or fidelity.

We noticed that most of the bad generated samples were due to the influence of other objects or people in the background. For this reason we selected a generator (BigGAN) and a classifier (WS-DAN) which used attention maps to achieve better results compared to the other techniques. We think that this is a smart approach in subject recognition and advice that eventual future works should either use it or pay attention to remove bad samples in some way.

It should be noted that our approach can be exploited out-the-box by different classifiers without modify existing architectures, if proven more effective.

6. CODE AND AGUMENTED DATASET

Our code, results and the full augmented dataset are available at https://gitlab.com/isnob46/doggo_gan freely for use and improvement under GNU license.

7. REFERENCES

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. (2014). Generative Adversarial Nets
[2] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. (2016). Improved Techniques for Training GANs
[3] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Sepp Hochreiter. (2018). GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium
[4] Augustus Odena, Christopher Olah, Jonathon Shlens. (2016). Conditional Image Synthesis with Auxiliary Classifier GANs
[5] Andrew Brock, Jeff Donahue, Karen Simonyan. (2019). Large scale GAN training for high fidelity natural image synthesis
[6] Han Zhang, Ian Goodfellow, Dimitris Metaxas, Augustus Odena. (2019). Self-Attention Generative Adversarial Networks
[7] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. (2012). ImageNet Classification with Deep Convolutional Neural Networks
[8] Karen Simonyan, Andrew Zisserman. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition
[9] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. (2016). Densely Connected Convolutional Networks
[10] Tao Hu Honggang Qi, Qingming Huang, Yan Lu. (2019). See Better Before Looking Closer: Weakly Supervised Data Augmentation Network for Fine-Grained Visual Classification
[11] http://vision.stanford.edu/aditya86/ImageNetDogs/
[12] http://www.image-net.org/
[13] https://github.com/eriklindernoren/PyTorch-GAN
[14] https://www.kaggle.com/jesucristo/gan-introduction
[15] https://github.com/huggingface/pytorch-pretrained-BigGAN
[16] https://pytorch.org/hub/pytorch_vision_alexnet/
[17] https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py
[18] https://github.com/pytorch/vision/blob/master/torchvision/models/densenet.py
[19] https://github.com/wvinzh/WS_DAN_PyTorch