Self-Competitive Neural Networks

Iman Saberi and Fathiyeh Faghih

October 2, 2020

Self-Competitive Neural Networks

Iman Saberi and Fathiyeh Faghih

College of Engineering
Department of Electrical and Computer Engineering
University of Tehran, Tehran, Iran
{iman.saberi,f.faghih}@ut.ac.ir

Abstract. Deep Neural Networks (DNNs) have improved the accuracy of classification problems in lots of applications. One of the challenges in training a DNN is its need to be fed by an enriched dataset to increase its accuracy and avoid it suffering from overfitting. One way to improve the generalization of DNNs is to augment the training data with new synthesized adversarial samples. Recently, researchers have worked extensively to propose methods for data augmentation. In this paper, we generate adversarial samples to refine the Decision boundaries of each class. In this approach, at each stage, we use the model learned by the primary and generated adversarial data (up to that stage) to manipulate the primary data in a way that look complicated to the DNN. The DNN is then retrained using the augmented data and then it again generates adversarial data that are hard to predict for itself. As the DNN tries to improve its accuracy by competing with itself (generating hard samples and then learning them), the technique is called Self-Competitive Neural Network (SCNN). To generate such samples, we pose the problem as an optimization task, where the network weights are fixed and use a gradient descent based method to synthesize adversarial samples that are on the boundary of their true labels and the nearest wrong labels. Our experimental results show that data augmentation using SCNNs can significantly increase the accuracy of the original network. As an example, we can mention improving the accuracy of a CNN trained with 1000 limited training data of MNIST dataset from 94.26% to 98.25%.

Keywords: Deep Neural Networks · Data Augmentation · Computer Vision

1 Introduction

Deep learning models have performed remarkably well on many classification problems. With the advent of Convolutional Neural Networks (CNNs) [9], significant improvements have been reported in computer vision tasks. One of the main challenges in training a deep neural model is providing a big dataset in order to prevent model from overfitting. The challenge is more significant in small datasets, such as medical image analysis. Data augmentation is a known solution in the literature to improve model generalization.

One of the well-known techniques for data augmentation are Generative Adversarial Networks (GANs) [6]. They are used to generate new data in order to inflate the training dataset [10,6,3,1]. There are, however, challenges in data generation using these networks. First, the aim of GANs is to find a Nash equilibrium of a non-convex game with continuous and high-dimensional parameters, while they are typically trained based on a gradient descent based technique designed to minimize a cost function (instead of finding a Nash equilibrium of a game). Therefore, these algorithms may fail to converge [7], and hence, synthesis of high resolution data may be very difficult using this technique. Another limitation of GANs is their need to substantial amount of primary data in order to train the discriminator well, and hence, they are not practical in small datasets [13].

In this paper, we propose a novel approach in this field that concentrates on the weaknesses of the functional structure of DNNs in order to improve their accuracy. The non-polynomial architecture of DNNs consists of deep linear and nonlinear operations, and hence, the Decision boundary of each class cannot be easily determined. We present a method to refine the DoAs of the network by synthesizing harder samples from the primary input dataset. The synthesizer network modifies each sample in a way that it is located on the boundary of its true label and its nearest wrong label in the trained embedded network. The synthesizer tries to generate more complicated samples from the primary data, and then the embedded network tries to learn them correctly. This cycle is repeated as many times as the accuracy of the embedded network increases. Similar to GANs, our approach consists of two networks, a synthesizer and an embedded network. However, unlike GANs that try to play a minimax game and converge to a Nash equilibrium, in our approach, the two networks separately try to minimize their loss function based on a gradient descent method in order to improve the accuracy and robustness of the embedded network.

Our experimental results demonstrate that our proposed technique can improve the accuracy of networks, especially in small datasets. We selected a limited set of 1000 training data from the MNIST dataset, and fed them to an SCNN. We observed that the accuracy of the baseline well-trained embedded network was increased from 94.26% to 98.25% using our technique. We also did experiments on Fashion MNIST and Cifar10 datasets as harder datasets. The results show 1.35% increase in the accuracy of the Fashion MNIST dataset and 4.85% increase in the accuracy of the Cifar10 dataset, compared to the baseline Resnet18 Model.

2 Related Work

Mining hard examples was previously studied in the literature [15,14]. The idea is to select or generate optimal and informative samples in order to enrich the dataset. Data augmentation techniques in the literature can be categorized into data wrapping and oversampling methods. Data wrapping augmentations transform the existing samples, such that their labels are preserved. Geometric and color transformations, random erasing, adversarial training, and style transfer networks are examples of data wrapping techniques. Oversampling augmentation methods generate new instances and add them to the training set. Oversampling encompasses augmentations, such as mixing images, feature space augmentation, and GANs. These two categories do not form a mutually exclusive dichotomy [13]. Our approach is a data wrapping augmentation technique that tries to manipulate each sample, such that it is located on the boundary of its true label and its nearest wrong label.

In [5], the authors propose a method to seek small transformations that yield maximal classification loss on the transformed sample based on a trust region strategy. This work is similar to our idea in the criterion of generating informative augmented data. However, our strategy in synthesizing augmented samples is totally different. The proposed algorithm in [5] selects a set of possible transformations, where each one has a specific degree of freedom. The algorithm applies a set of transformations that make the cost function have the most value. Our technique is different in that a gradient descent method tries to move each sample in a direction to be located on the decision boundary for that sample.

The closest work to this paper is [12], where the authors designed an augmentation network that competes against a target network by generating hard examples (using GAN structure). The algorithm selects a set of augmentations that have the maximal loss against random augmentation samples. The idea is to apply a reward/penalty strategy and formalize the problem as a minimax game for effective generation of hard samples. Our paper is different in that we do not make the generator network to select the best strategy from a set of predefined strategies. Instead, a gradient descent method decides what is the best transformation parameters in order to manipulate each sample. Also, we pose the problem as minimizing two cost functions separately, instead of playing a minimax game.

3 Self-Competitive Neural Network

3.1 SCNN Architecture

The architecture of a Self-Competitive Neural Network (SCNN) contains two main parts:

1. An embedded neural network: The goal of SCNN is to improve the accuracy of this network. The internal design of this network is crucial in order to have a powerful model. The embedded neural network can be of any type, such as Convolutional Neural Network (CNN).
2. An Adversarial Data Synthesizer (ADS) network: This network is constructed by concatenation of a number of differentiable components and the embedded network. The aim of this network is synthesizing difficult data for the embedded network. The differentiable components take the input data and try to learn their parameters in a way that the synthesized data is predicted as the nearest wrong label by the embedded network. The details of the internal structure of this component will be discussed in Section 4.

The training life cycle of an SCNN has three main phases (Fig. 1):

1. Training with primary data: In this phase, the embedded network is trained by the main input dataset.
2. Adversarial data synthesis: After training the embedded network by the primary data, the weights of the embedded network are set as immutable, and the primary input data is manipulated in a way that the embedded network predicts the synthesized data by a wrong label. In other words, the ADS network manipulates the input data in a way that they are more difficult for the embedded network to predict correctly.
3. Training with adversarial data: In this phase, the SCNN uses the synthesized data from the previous phase, sets the weights of the embedded network as mutable, and the embedded network is trained by the synthesized adversarial data.

You can see the embedded network in the both the training phases and the data generation phase. The difference is that in the training phases, the weights of the embedded network are mutable and learned during these steps. However, in the data generation phase, the weights are an immutable part of the ADS network, and hence, they are not changed.

The three phases are repeated in a cycle as many times as the accuracy of the embedded network improves. In each cycle, the weights of the embedded network are tuned by the primary data (phase 1), the ADS synthesizes new data from the primary input data that seem harder for the embedded network to predict correctly (phase 2), and the embedded network is trained by the synthesized harder boundary data (phase 3). This procedure can be considered as *online* strategy, where the underlying embedded network is improved gradually by harder data. Another approach is *offline*, where the model is trained from scratch by the primary and harder data generated in the training life cycle of an SCNN. It is obvious that the offline approach is more expensive, but as we will show in Section 5, the resulting accuracy of this approach is better than the online strategy.

4 Details of SCNN

The main idea of SCNN can be thought of as a competition between the embedded neural network and the adversarial data synthesizer, where during the training phases (both training with primary data and adversarial data), the embedded network tries to correctly predict the labels of the primary and adversarial data. On the other hand, during the adversarial data synthesis, the SCNN tries to manipulate the input data in a way that the embedded network predicts the manipulated data with wrong labels.

For simplicity in explanation and without loss of generality, consider a MultiLayer Perceptron (MLP) as an embedded network. A loss function is defined in a way that explains how much the outputs of the network are different from the correct labels. The optimizer uses gradient descent to minimize this function.

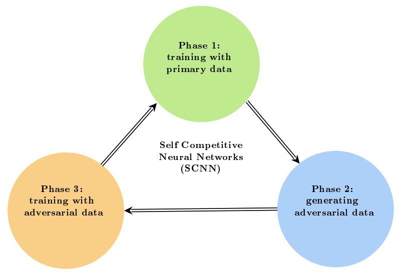


Fig. 1: The life cycle of training a SCNN

The main idea of SCNN is to fix the weights of the embedded neural network, and use the gradient descent technique in order to manipulate the input data and generate a set of adversarial data. As an example, the architecture of the ADS network with MLP as the embedded network is presented in Fig. 2. In this structure, the input vector $X = [x_1, x_2, ..., x_n]$ is fed to a (a set of) differentiable component(s) that try to manipulate the input data to a vector $X' = [x'_1, x'_2, ..., x'_n]$, which is fed to the embedded network. To do that, we define an optimization problem, where the parameters of the differentiable components are trained based on a cost function that is defined in a way that the manipulated data is predicted as the nearest wrong label by the embedded neural network.

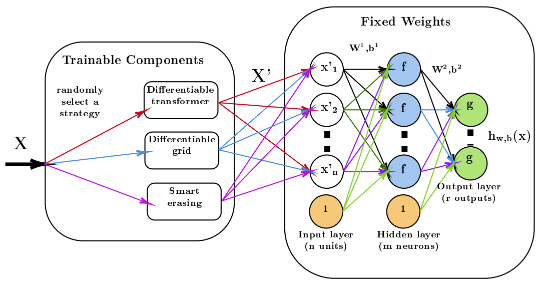


Fig. 2: Adversarial data synthesizer network

The trainable components are designed based on the type of the input dataset. In this paper, we consider three main trainable components for the *image* samples:

- **Differentiable transformer:** We use Spatial Transformer Network (STN) layer introduced in [8] for this transformation. It takes a vector of size 6 per image to perform an affine transformation (such as rotation, zoom, shearing) on the input image. For example, if we have an input image and the coordinate of each pixel is identified by (x_i^s, y_i^s), and (x_i^t, y_i^t) represents the coordinate of the pixel after the affine transformation, we have:

$$G_t = \tau_\theta(G_i) = \begin{pmatrix} x_i^t \\ y_i^t \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^s \\ y_i^s \\ 1 \end{pmatrix} \tag{1}$$

where G_i is the initial grid of the input feature map, and G_t is the final grid of the feature map after applying the transformation τ_θ on G_i. The most important feature of this component is that it is differentiable with respect to its transformation parameters, which means the gradient descent can learn them based on a defined cost function.

– **Differentiable grid:** We design this component to be more flexible to perform grid manipulation in the input feature map. It is a differentiable component that manipulates the grid of the input image in three main steps:
  1. Convolution on a null input space: For each input image, we generate a null input space (δ) based on sampling from uniform distribution $U(-1, +1)$ with the same size as the input image size, denoted by (W, H) . This space is the starting point for grid manipulation, and will be trained during the adversarial data synthesis phase. After that, we define a Gaussian kernel window, which is convolved with this null space, and as a result, we will have a smoothed null input space, as depicted in Fig. 3a. The convolved null space (μ) can be computed as follows:

$$\begin{aligned} &\delta_{ij} \sim U(-1, +1) : \; i \in [1...W] \;\; j \in [1...H] \\ &\mu_{ij} = \delta_{ij} \circledast Kernel_{ij} = \sum_{m=1}^{w_K} \sum_{n=1}^{h_k} Kernel_{m,n}(\Phi_\mu, \Phi_\sigma)\delta_{i-m,j-n} : \\ &\forall i \in [1...W] \;\; \forall j \in [1...H] \end{aligned} \tag{2}$$

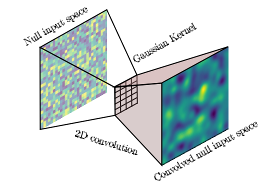
  where $Kernel$ is a predefined 2D Gaussian kernel with size (w_K, h_K), and mean and standard deviation equal to Φ_μ and Φ_σ, respectively. Note that this kernel is immutable during the training phase of the input null space.
  2. Generating the manipulated grid: In this step, the convolved null space (μ) is added to the initial grid (G_i) of the input image, and as a result, the manipulated grid (G_t) is generated, as shown in Fig. 3b.

$$G_t = G_i + \mu \tag{3}$$

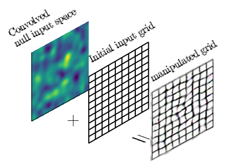
  3. Image sampling: The main operation of this step is an image interpolation based on the manipulated grid generated in the previous step. As depicted in Fig. 3c, the image sampler takes an image and a manipulated grid as input, and computes the interpolated output image based on the manipulated grid. The Final output image can be computed as follows:

$$\begin{aligned} &O_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} I_{nm}^c k(x_i^t - m; \Phi_x) k(y_i^t - n; \Phi_y) \\ &\forall i \in [1...HW] \; \forall c \in [1...C] \\ &(x^t, y^t) \in G_t \end{aligned} \tag{4}$$

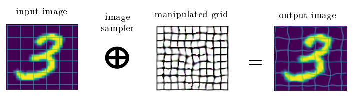
where k is a generic sampling kernel function that performs the image interpolation. It can be any image interpolation function (e.g bilinear), and Φ_x and Φ_y are the parameters of this kernel function. I^c_{nm} is the input value at location (n, m) in channel c of the input image, and O^c_i is the output value for pixel i at location (x^t_i, y^t_i).



(a) Null input-space convolved with predefined Gaussian kernel



(b) summation of convolved input space and initial grid of input image



(c) Image sampling

Fig. 3: Three main steps of the differentiable grid component

– **Smart erasing:** Random erasing is one of the most effective data augmentation techniques which is mostly used in DNNs [11]. This component exploits erasing technique in order to smartly erase some parts of the input data in a way that the erased samples maximize the loss function. This component has four main steps:

1. Defining an NxN trainable mask (M): this mask covers the input data and responsible for distinguishing which parts of the input have the most significant effect on classifying the input data. Parameter N could be varied based on input size(for example, for a 32x32 input size, N could be 4 or 8).
2. A 3D-Upsampling operation: this operation fits the size of the grid to the size of the input image. for instance, for a 4x4 grid, we use an 8x8x3 Upsampling operation in order to make a 32x32x3 grid(same size as the input data). The elements of the upsampled grid are multiplied to input data.
3. Finding attention area: From the previous step, we prepare a trainable grid that covers the input data. Now, we train this grid in order to distinguish which parts of the input data have the most significant effect on classifying tasks (Fig. 4.b).
4. Erasing the most effective grids: after finding attention areas, grids with the most impact on the classifying task are deleted or replaced with noisy data.(Fig. 4.d, Fig. 4.e). There are two possibilities, if the removed area is a part of the background, it will help classifying task that decreases the importance of this part. If the removed area is an essential part of the input, the algorithm tries to identify the input without considering this part.

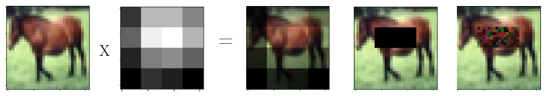

(a) Input Image (b) Attention area(4x4 grid) (c) Output Image (d) Smart Erasing1 (e) Smart Erasing2

Fig. 4: Applying smart erasing on a specific image of cifar10 database

We defined three components, the transformer component which is differentiable with respect to its transformation parameters θ, and the grid manipulation component, which is differentiable with respect to its null input space δ and the smart erasing component which is differentiable with respect to its defined mask (M). Let's denote the output of the transformer component with $V_{(X_j|\theta)}$ and the output of the grid manipulation component with $O_{(X_j|\delta)}$ and the output of the random erasing component with $E_{(X_j|M)}$ for input sample X_j. X' (the generated adversarial data to be fed to the embedded network) can be computed as follows (Fig. 2):

$$X' = \begin{cases} V_{(X|\theta)} \ \ or \\ O_{(X|\delta)} \ \ or \\ E_{(X|M)} \end{cases} \tag{5}$$

The corresponding function of the adversarial data synthesizer network is as follows:

$$h^{ADS}(X|\theta,\delta) = h^{MLP}(X') \tag{6}$$

where h^{ADS} is the corresponding function of the ADS network, and h^{MLP} is the corresponding function of the embedded network (MLP in this example). The goal is to optimize parameters θ, δ and M in a way that the output of the ADS becomes the nearest wrong label for each input data. For defining an appropriate loss function, we need to define the following variables:

$$\begin{aligned} \hat{y}_j &= \arg\max_i h_i(X_j|\theta,\delta)^{ADS} \\ \check{y}_j &= \arg\max_i h_i(X_j|\theta,\delta)^{ADS} : i \neq \hat{y}_j \\ \bar{y}_j &= \begin{cases} (y_j + \alpha\check{y}_j) \ if \ \hat{y}_j = y_j \\ (y_j + \alpha\hat{y}_j) \ if \ \hat{y}_j \neq y_j \end{cases} \end{aligned} \tag{7}$$

where $\hat{y}_j$ is the index of the largest output of the ADS network, $\check{y}_j$ is the index of the second largest output of the ADS network, and $\bar{y}_j$ is the desired output of the ADS network for generating harder samples. Note that $\bar{y}_j$ is constructed by a linear combination of the correct label y_j, and the nearest wrong label for the input data X_j. Also, α is the strength rate of moving the input data to the boundary of its true label and the nearest wrong label. Now, we can define the loss function of the ADS, as follows:

$$loss^{ADS} = -\sum_{i=1}^{n}\sum_{k=1}^{K} \bar{y}_{ik} log(\hat{y}_{ik}) \tag{8}$$

The goal is to optimize parameters (θ, δ, M), such that the manipulated input data is predicted with a wrong label, or becomes closer to the boundary of its true label and its nearest wrong label.

5 Experimental Results

We implemented SCNN on three datasets; MNIST, Fashion MNIST, and Cifar10. The underlying embedded network is a CNN for MNIST and Fashion MNIST datasets, consisting of a convolutional, a BatchNormalization, and a pooling layer, repeated three times in a stack. Size of all filters is (3x3) and number of filters in convolution layers are 64, 128 and 256 respectively. We also add a fully-connected hidden layer(with 512 neurons) to the end of last convolution layer. All activation functions are ReLu to have less intensive computation overhead in the training phase. For Cifar10 dataset we choose Resnet18 structure as our underlying embedded network. Number of filters in resnet18 start with 64 and we use a dropout layer(mask ratio:0.2) after the fully connected layer of Resnet18.

5.1 Limited MNIST

We randomly selected 1000 samples of the MNIST handwritten dataset to have a more challenging problem. SCNN generated 1,000,000 adversarial hard samples in 50 cycles. For comparison with random augmentation, we use the Augmentor tool [2] that applies random augmentation operations, such as rotation, scaling, shearing, flipping, random erasing, and elastic transformations. The results based on the online strategy for baseline model, Augmentor tool, and SCNN are shown in Fig. 8a. It can be observed that the SCNN approach experiences less overfitting compared to other approaches. The final accuracy results are depicted in Table 1. In Fig. 5, we illustrate the process of synthesizing a hard sample by applying gradient descent on the trainable parameters of the transformer, where a sample with true label 2 is transformed, such that its label is predicted as 8.

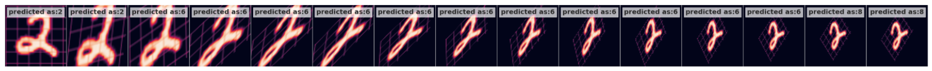


Fig. 5: Data manipulation based on differentiable transformer on MNIST sample data

5.2 Fashion MNIST

We use the entire 60,000 samples in the training dataset of Fashion MNIST, and generated 600,000 adversarial hard samples in 10 cycles. Once the baseline model was trained by the primary data, and another time, it is trained by the adversarial and primary data. The results are shown in Fig. 8b, and the final accuracy after 100 epochs is shown in Table 2. It can be observed that SCNN experiences less overfitting compared to the baseline model. Fig. 6 illustrates the synthesis of a hard sample by applying gradient descent on the trainable parameters of the differentiable grid, where the goal is to manipulate a sample with true label 0 (T-shirt), such that it is predicted as label 3 (Dress).

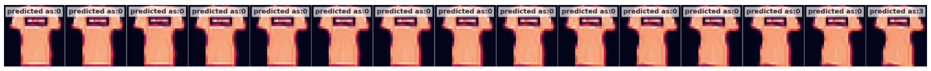


Fig. 6: Data manipulation based on differentiable grid on Fashion-MNIST sample data

5.3 Cifar10

This dataset is more complicated, consisting of 50,000 training data. We generated 100,000 adversarial data in 5 cycles. The baseline model was trained by 1.the primary data, 2.the primary data with random augmentation, 3.the primary data with AutoAugment approach which is introduced in [4] and 4.the primary data with the SCNN. The results are shown in Fig. 8c, after epoch 150 we retrain our network with generated adversarial data and primary data, as a result we firstly observed a decrease in the accuracy after epoch 150. The final accuracy of these approaches is shown in Table 3. Fig. 7 illustrates synthesizing a hard sample by applying gradient descent on the trainable parameters of the differentiable grid, where the generated sample is finally predicted correctly, but with less confidence. So we can see that after multiple cycles, the label of the adversarial data may still be predicted correctly.

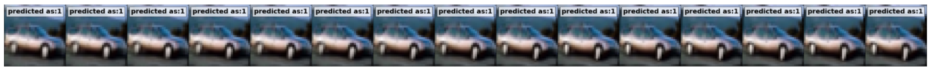
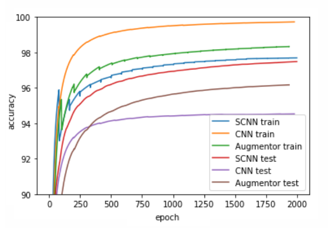


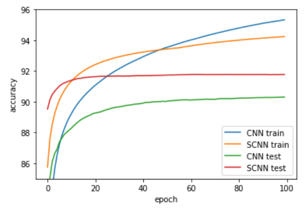
Fig. 7: Data manipulation based on differentiable grid on Cifar10 sample data

6 Conclusion and Future Work

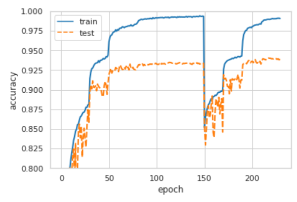
We introduced a new approach for generating harder samples for DNNs. An SCNN contains two sub-networks, an embedded network, and a synthesizer



(a) Online Strategy for limited MNIST dataset



(b) Offline Strategy for Fashion-MNIST dataset



(c) Online Strategy for Cifar10 dataset

Fig. 8: Performance of SCNN for different datasets

Table 1: Test accuracy of different approaches on the Limited MINST(1000 train data)

Network Type	test accuracy(online strategy)	test accuracy(offline strategy)
Baseline	94.26	94.26
Baseline + Augmentor	96.20	96.64
Baseline + SCNN	97.63	**98.25**

Table 2: Test accuracy of CNN baseline model and SCNN on the Fashion MNIST dataset

Network Type	test accuracy(online strategy)	test accuracy(offline strategy)
Baseline CNN	90.11	90.11
Baseline + Augmentor	90.54	90.74
SCNN	91.03	**91.45**

Table 3: Test accuracy of different approaches on the Cifar10 dataset

Network Type	test accuracy
Resnet18	89.2
Resnet18 + Random Augmentation	93.3
Resnet18 + AutoAugment	93.83
Resnet18 + SCNN	**94.11**

network. We also proposed three differentiable image manipulator components which parameters can be learned in a way that can transfer a sample to the boundary of its true label and its nearest wrong label. We observed that the accuracy and the robustness of SCNNs are improved in comparison with their baseline models. As a future work, this approach can be extended by designing differentiable manipulator components for other types of input data, such as texts, graphs, and voice.

References

1. Antoniou, A., Storkey, A., Edwards, H.: Data augmentation generative adversarial networks. arXiv preprint arXiv:1711.04340 (2017)
2. Bloice, M.D., Stocker, C., Holzinger, A.: Augmentor: an image augmentation library for machine learning. arXiv preprint arXiv:1708.04680 (2017)
3. Bowles, C., Chen, L., Guerrero, R., Bentley, P., Gunn, R., Hammers, A., Dickie, D.A., Hernández, M.V., Wardlaw, J., Rueckert, D.: Gan augmentation: Augmenting training data using generative adversarial networks. arXiv preprint arXiv:1810.10863 (2018)
4. Cubuk, E.D., Zoph, B., Mané, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) pp. 113–123 (2019)
5. Fawzi, A., Samulowitz, H., Turaga, D., Frossard, P.: Adaptive data augmentation for image classification. In: 2016 IEEE International Conference on Image Processing (ICIP). pp. 3688–3692. Ieee (2016)
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
7. Goodfellow, T.S., Zaremba, W., Ian, V.C.: Improved techniques for training gans. arXiv preprint arXiv:1606.03498 (2016)
8. Jaderberg, M., Simonyan, K., Zisserman, A., et al.: Spatial transformer networks. In: Advances in neural information processing systems. pp. 2017–2025 (2015)
9. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
10. Nielsen, C., Okoniewski, M.: Gan data augmentation through active learning inspired sample acquisition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 109–112 (2019)
11. O'Gara, S., McGuinness, K.: Comparing data augmentation strategies for deep image classification. IMVIP 2019: Irish Machine Vision & Image Processing (2019)
12. Peng, X., Tang, Z., Yang, F., Feris, R.S., Metaxas, D.: Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2226–2234 (2018)
13. Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of Big Data **6**(1), 60 (2019)
14. Shrivastava, A., Gupta, A., Girshick, R.: Training region-based object detectors with online hard example mining. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 761–769 (2016)
15. Uijlings, J.R., Van De Sande, K.E., Gevers, T., Smeulders, A.W.: Selective search for object recognition. International journal of computer vision **104**(2), 154–171 (2013)