INSA Rennes - Département EII

Parcours recherche 2018-2019

Rapport bibliographique

Data Augmentation for Image Classification

Étudiant:
Eduardo Fernandes Montesuma

Encadrants:
Florian Lemarchand
Maxime Pelcat

February 16, 2020

Abstract

The training of machine learning models has evolved as both processing and storing capabilities of computers have increased. With increasing complexity, deep learning models have made their way into the state-of-the-art, becoming expressive in many tasks that classical signal processing techniques have struggled to solve, such as computer vision and speech recognition. However, such complexity has a major drawback: most deep learning algorithms require a huge amount of data, which is still a constraint for many applications, even in nowadays context of big data. In that sense, data augmentation has become one of the most popular methods to enhance machine learning models, by creating new synthetic samples from existing data, and hence, increasing data availability. The present report studies how data augmentation has been employed before deep learning, how deep learning can be used to develop new methodologies and, finally, how this set of techniques can be used together, in the context of Image Classification.

Keywords— Deep Learning, Data Augmentation, Neural Style Transfer, Image Translation

Contents

1 Introduction

Nowadays, training state-of-the-art deep learning models for image classification requires a large amount of data [1]. In the context of big data, the availability of data is not an issue anymore for most applications as it was before. Nevertheless, there are still challenging factors to effectively construct a dataset, as feature extraction and data quality.

The main concern involving the training of Deep Neural Network (DNN) with small datasets is lack of generalization power, that is, once trained, the model fails to correctly classify new samples. In that case, data augmentation is proposed as an avenue to enhance learning models.

1.1 What is data augmentation?

The development of machine learning as a field of Artificial Intelligence has brought into perspective data-driven algorithms, that is, how they are trained, and how they can be used to perform predictions on new data. In that context, data augmentation can be seen as the process of artificially creating new data samples [2].

The main benefit of augmenting data in a machine learning problem is to avoid overfitting. Such phenomena can be met in complex learning problems with many degrees of freedom. A common overfitting scenario is when the model perfectly fits the training samples, while showing poor results on new predictions. As regarded in [3], training complex models with a limited amount of data often leads to overfitting.

Moreover, the amount of data needed to train a given model is application dependent. For instance, as regarded in [4], simple recognition tasks can be solved with relatively small datasets, while object recognition tasks, which require much more complex networks, requires much larger datasets.

Such a requirement, however, as suggested in [5], is often not met in specific areas where machine learning has been applied, such as image and video classification tasks. Therefore, data augmentation methods can be used to enhance the training of data-driven models by acting as regularizers [2], thus reducing overfitting. Important to mention, the generalization is improved without reducing model complexity, in constrat to other techniques, such as dropout [6].

In that sense, data augmentation methods are used for image classification [5] to enhance learned models, but are not limited to only this kind of application. For instance, data augmentation has been used to train deep learning models for Super-Resolution Imaging [7] and Image Denoising [8, 9].

Most state-of-the-art learning algorithms are DNNs, which commonly have numerous layers, enlarging not only the number of learnable parameters, but also the degrees of freedom of models. As consequence, these models require much more data to have an acceptable generalization capability, that is, predict new data correctly.

Within data augmentation methodology, yet, there are two approaches concerning the domain of its application. It can be applied to data or feature space. Such distinction comes from the common pipeline employed in machine learning systems (e.g. Image Classification), displayed in Figure 1,

Here, raw data (e.g. images) are fed into a feature extractor to generate meaningful variables to help the classifier to identify patterns within data. For instance, [10] has identified over twenty features within raw cell images to automatically detect whose were cancerous.

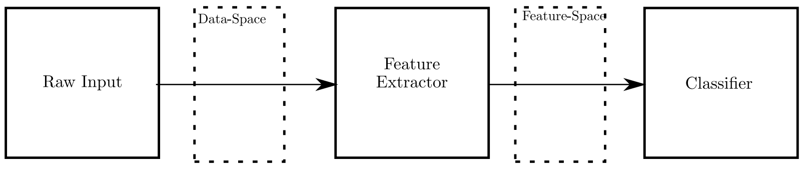

Figure 1: Data classification pipeline adapted from [2]. Raw data is fed as input to a feature extractor, to generate features so that the classifier can classify the data.

2 Data augmentation Methods

Data augmentation can be further defined regarding the domain of its application. It can be either applied to raw data inputs, or to extracted features. When applied to Data Space, one generally exploits the structure of the data types handled in the algorithm to generate new synthetic examples. For instance, when trying to recognize phonemes in audio samples, one can add noise to the samples, or change the audio speed [11].

The application of data augmentation to Feature Space, however is not so trivial as Data Space augmentation. This problem comes from the fact that, between raw data and feature-extracted data there is a feature extractor that represents samples in a different way. That feature extractor can be made by a human expert, or automatically extracted as in convolutional layers of a Convolutional Neural Network (CNN).

The decision on whether perform data augmentation on Data or Feature Space, however, is application-dependent. As [2] suggests, hypothesizing that transformations in Data Space preserve the data labels, data augmentation in Data Space has better performance in contrast with Feature Space data augmentation. In this section, we evaluate the main methodologies for synthetically generating new samples in both domains, in the context of Image Classification.

2.1 Data augmentation in Data Space

As opposed to data augmentation in the Feature Space, data augmentation in Data Space is rather dependent on application domain. Being so, here we discuss the main techniques to augment image data. As pointed out in [2], Data Space data augmentation for Image Classification is based on two types of image transformations, Geometric/Affine Transformations and Elastic Transformations. These two kinds of transformations are referred as Image Warping [12].

Concerning Geometric Transformations, and given an image $\mathbf{f}_o \in \mathbb{R}^{p \times q}$, one can find a matrix $\mathbf{A} \in \mathbb{R}^{n \times p}$ and a scalar $b \in \mathbb{R}$ so that $\mathbf{f}_w$, the warped image, may be expressed as:

$$\mathbf{f}_w = \mathbf{A}\mathbf{f_o} + b. \tag{1}$$

The group of transformations falling under such definition are: scaling, rotation, shearing and shifting. The approach to perform Elastic Deformations is similar. We first define a random displacement field $\mathbf{u}(x, y)$, for each pixel (x, y) in $\mathbf{f}_o$, so that $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, where $\mathbf{I}$ is the identity matrix. Thus, $\mathbf{u}$ follows a Gaussian distribution of zero mean and variance σ^2, hence,

$$\mathbf{f}_w = \mathbf{f}_o + \alpha \mathbf{u}, \tag{2}$$

where α is the strength in pixels displacement. To illustrate how they are applied in Image Classification problems, we use these transformations in MNIST handwritten digits dataset, as shown in Figure 2.

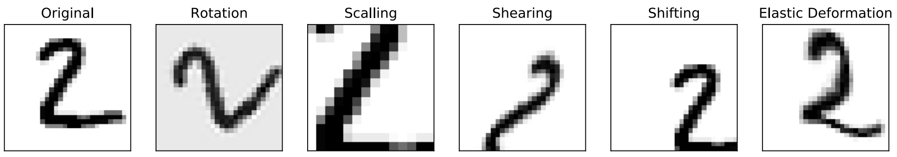

Figure 2: From left to right, the first image concerns the original sample from MNIST. From images 2, to 5, we apply geometric transforms to them. For the elastic deformation, we applied it with fine-tuned parameters $\alpha = 34$, and $\sigma = 3$.

These transformations, however, not always preserve the class of the original sample $\mathbf{f}_o$. In those cases, we say that the transformation does not preserve labels. This is a potential threat to the accuracy of the classifier. Indeed, we would feed the pipeline with a sample with incorrect class. Figure 3 shows this issue in the context of MNIST dataset.

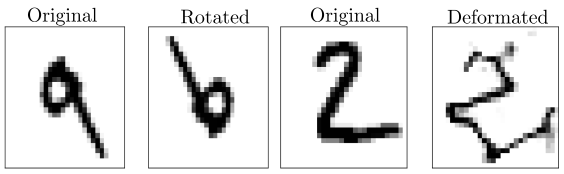

Figure 3: From left to right, the first two images address the problem where we rotate a 9 digit by 180 degrees, turning it into a 6. The later two shows the application of elastic deformation with an intensity parameter α too strong, turning the sample into an unrecognizable shape.

2.2 Data augmentation in Feature Space

The main motivation to perform Feature Space data augmentation is when Data Space methods fail to provide new samples with preserved labels. Feature Space methods have their origin in the problem of class imbalance in classification task. By definition, a classification problem is said to be imbalanced if the number of elements for each class is not equal. In consequence, classifier may be biased to classify samples as being part of the most frequent classes.

A solution to class imbalance is to synthetically generate new samples for the minority class, also known as Synthetic Minority Over-sampling Technique (SMOTE), as proposed in [13]. The algorithm is based on Nearest Neighbours method. The process is done through the selection of a point $\mathbf{x}$ within the under-sampled class $\mathcal{C}_0$, and its respective nearest neighbour within $\mathcal{C}_0$, say, $\mathbf{y}$.

SMOTE over-samples new data points in feature space by picking a random point within the line joining $\mathbf{x}$ and $\mathbf{y}$.

The SMOTE algorithm, yet, has as main drawback of over-sampling in regions where two classes overlap. This is addressed by [14], where authors have proposed an alternative algorithm for over-sampling, the Density-Based Synthetic Minority Over-sampling Technique (DBSMOTE) technique. To comprehend how sampling is done, the authors have defined the notion of Directly Reachable Points. Given ϵ and N, $\mathbf{x}$ is Directly Reachable from $\mathbf{y}$ if $||\mathbf{x} - \mathbf{y}||_2 \leq \epsilon$, and its ϵ-vicinity, $\mathcal{N}_\epsilon(\mathbf{x}) = \{\mathbf{z} : ||\mathbf{z} - \mathbf{x}||_2 \leq \epsilon\}$, has at least N points. This concept is illustrated in the left of Figure 4.

With such definition, DBSMOTE uses another method to perform clustering within the dataset: the Density-Based Spartial Clustering of Applications with Noise (DBSCAN) algorithm. This later procedure manages to divide the dataset into three regions: 1. Core points, which are those with more than N points in its $\epsilon-$vicinity; 2. Border points, which are Directly Reachable from core points, but have less than N points in their vicinity; 3. Noise Points, which are neither reachable from any core point, nor have more than N points in their vicinity. The illustration of DBSCAN is shown in the center of Figure 4.

With the clusters defined, given that $\mathcal{C}_0$ is the under-sampled cluster , DBSMOTE estimates its center by finding the nearest point $c_0 \in \mathcal{C}_0$ to the cluster average $\overline{X}_0$, that is,

$$\begin{aligned} c_0 &= min_{\mathbf{x} \in \mathcal{C}_0} ||\mathbf{x} - \overline{X}_0||_2, && (3) \\ &= min_{\mathbf{x} \in \mathcal{C}_0} \bigg|\bigg|\mathbf{x} - \frac{1}{|\mathcal{C}_0|} \sum_{\mathbf{y} \in \mathcal{C}_0} \mathbf{y}\bigg|\bigg|_2. && (4) \end{aligned}$$

For an arbitrary point p within the under-sampled cluster, DBSMOTE, then computes the shortest path γ between p and c_i, composed only by directly reachable points. Then, the algorithm selects a random edge from such path, and performs SMOTE procedure between the vertex of such edge. This is illustrated in the right of Figure 4.

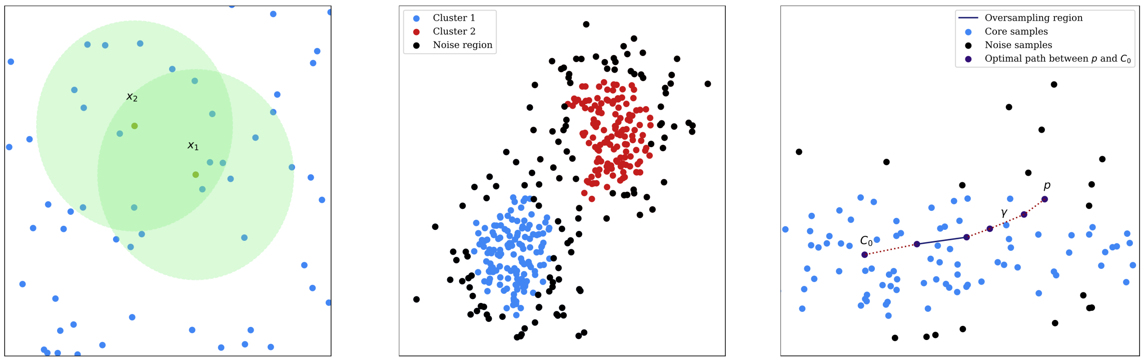

Figure 4: On the left, the green regions are the ϵ-vicinities of x_1 and x_2. If we consider $\epsilon = 0.6$, and $N = 6$, then x_1 and x_2 are directly reachable. On the center, DBSMOTE divides the dataset into cluster and noise regions. On the right, DBSMOTE performs SMOTE in a randomly selected edge of γ.

3 Deep Learning Based Data Augmentation

Nowadays, with increasing computational power and data availability, deep learning models and DNNs have received a great attention from the research community. These learning models are, today, the state-of-the-art in most of supervised and unsupervised learning applications.

Supervised learning is mainly concerned with creating models to describe the relationship between data $\mathbf{x}_i$, and its respective response y_i. In that sense, image classification may be viewed as an instance of such paradigm, where we have images as data, and its respective label as response.

Unsupervised learning, on the other hand, may be viewed as the process of learning inner structures and representations of data. Under this paradigm of learning, no label is required. For instance, a typical unsupervised problem is Dimensionality Reduction, where given data $\mathbf{x}$ in a p dimensional space, one tries to find the best representation $\tilde{\mathbf{x}}$, of original data, in a q dimensional space, where $q \leq p$.

To perform these tasks, the deep learning research community has proposed various DNNs architectures as building blocks of more complex learning systems. This is the case of CNNs, Generative Adversarial Networks (GANs) and Recurrent Neural Networks (RNNs) which give us the base for performing new data augmentation strategies in Data Space. The main goal of this section is not only to present the models used in deep learning, but also to show how they can be used to enhance data augmentation strategies.

3.1 Neural Style Transfer

CNNs are a particular type of DNNs, where one or more layers depends on the operation of convolution, rather than standard matrix multiplication. This special kind of network is the current state-of-the-art in image recognition task, approaching human performance as in MNIST handwritten digit-recognition Dataset [4], as well as computer vision tasks [4, 15].

The main interest in these kinds of network is that, having convolutional layers prepended to standard neural network architectures, these are able to learn feature representations of data as the inputs flow through the layers [16]. From Image Classification perspective, this is useful so that the classifier automatically learn to represent concepts such as edges and geometrical forms.

Not only in the context of classification, feature representations are useful for new kinds of information processing tasks, such as Texture Synthesis [17]. For this task, we consider having trained a CNN, such as VGG19 [15] to classify images. After training process, this CNN has learned a feature map at each convolutional layer. The idea of Texture Synthesis is, given a source image $\mathbf{f}_s$, to perform a gradient descent in a noisy input, so that the feature maps of the noisy image are as close as possible from those of source image.

The main contribution of such approach is to give the basis of Neural Style Transfer [16], which relies on texture synthesis, as authors have remarked. In this setting, we substitute the noise input with a target image, $\mathbf{f}_t$. The result is a synthesized image $\mathbf{f}_{sy}$, with the source form and target texture, rather than a synthesized image that retains only the texture.

To perform the style transfer between images, authors in [16] have used two kinds of losses: a style loss $\mathcal{L}_s$, and a content loss $\mathcal{L}_c$, which are defined as a sum of layer-wise losses,

$$\mathcal{L}_c(\mathbf{f}_s, \mathbf{f}_{sy}, \ell) = \frac{1}{2}\sum_{i,j}((F_s^\ell)_{i,j} - (F_{sy}^\ell)_{i,j})^2, \tag{5}$$

$$\mathcal{L}_s(\mathbf{f}_t, \mathbf{f}_{sy}, \ell) = \sum_{i,j}((G_{sy}^\ell)_{i,j} - (G_t^\ell)_{i,j})^2, \tag{6}$$

where $(F_s^\ell)_{i,j}$ are the features of the source at layer ℓ, and G_s^ℓ, the Gram matrix of these features. The usage of such matrix for computation of style loss comes from its definition, since it accounts for feature correlation at a given layer, that is,

$$(G_t^\ell)_{i,j} = \sum_k (F_t^\ell)_{i,k}(F_t^\ell)_{k,j}. \tag{7}$$

The Style Transfer process is done by applying Gradient Descent to the Synthesized image. Being initialized as a random noise, the algorithm takes the gradient of $\mathcal{L}_{total} = \alpha\mathcal{L}_c + \beta\mathcal{L}_s$ with respect to the features F_{sy}^ℓ, for each layer ℓ. Hence, the synthesized image is iteratively updated by the gradient, in order to minimize both losses. A summary of such algorithm is displayed in Figure 5.

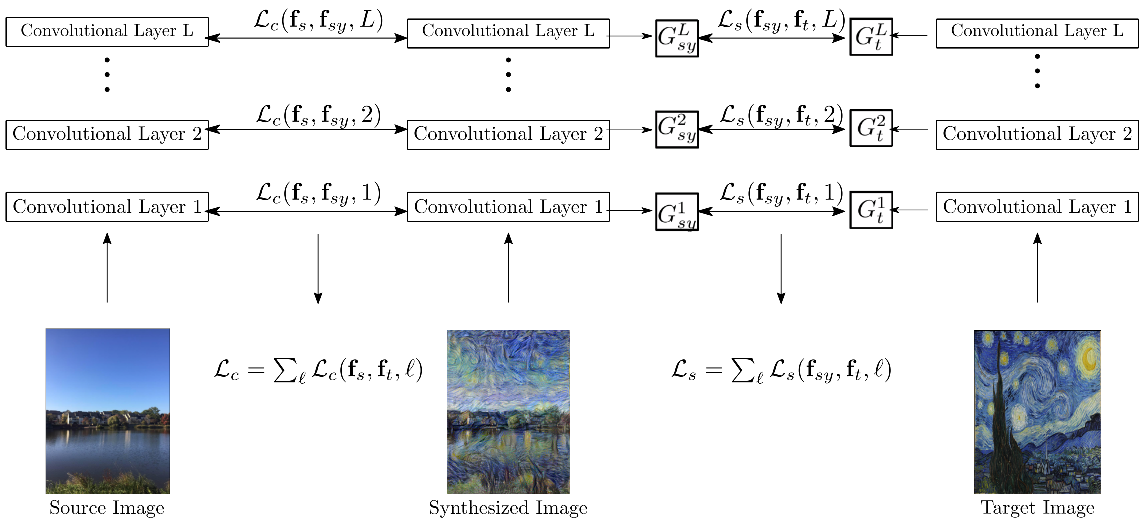
Convolutional Layer L
$\mathcal{L}_c(\mathbf{f}_s, \mathbf{f}_{sy}, L)$
Convolutional Layer L
G_{sy}^L
$\mathcal{L}_s(\mathbf{f}_{sy}, \mathbf{f}_t, L)$
G_t^L
Convolutional Layer L
Convolutional Layer 2
$\mathcal{L}_c(\mathbf{f}_s, \mathbf{f}_{sy}, 2)$
Convolutional Layer 2
G_{sy}^2
$\mathcal{L}_s(\mathbf{f}_{sy}, \mathbf{f}_t, 2)$
G_t^2
Convolutional Layer 2
Convolutional Layer 1
$\mathcal{L}_c(\mathbf{f}_s, \mathbf{f}_{sy}, 1)$
Convolutional Layer 1
G_{sy}^1
$\mathcal{L}_s(\mathbf{f}_{sy}, \mathbf{f}_t, 1)$
G_t^1
Convolutional Layer 1
$\mathcal{L}_c = \sum_\ell \mathcal{L}_c(\mathbf{f}_s, \mathbf{f}_t, \ell)$
$\mathcal{L}_s = \sum_\ell \mathcal{L}_s(\mathbf{f}_{sy}, \mathbf{f}_t, \ell)$
Source Image
Synthesized Image
Target Image

Figure 5: Diagram adapted from [16]. At each convolutional layer, the Content Loss, defined by Equation 5, and the Style Loss, defined by Equation 6, are computed. These are later summed to generate $\mathcal{L}_c$ and $\mathcal{L}_s$. These latter two, compose $\mathcal{L}_{total}$ whose gradient is used to update $\mathbf{f}_{sy}$.

3.2 Generative Adversarial Networks

Generative Models try to generate a distribution $\hat{p}_{model}$, based on available data, that represents the unknown data distribution p_{data}. Such models are important in the context of data augmentation since given a learned distribution $\hat{p}_{model}$, it is possible to sample new artificial data points from it. In fact, as [18] suggests, an ideal generative model would perfectly fit the unknown distribution p_{data}, so that we could generate perfect new artificial samples.

A particular type of generative model is GAN [19], which is composed by two entities: a discriminator D, and a generator G. The discriminator is a common classifier. The generator, on the other hand, takes noisy inputs and tries to generates outputs from the same distribution as the data.

The objective of D is to learn to discriminate between real data samples, and those generated by G, while the objective of G is to learn to deceive D. Once learning has been established, one can sample from the probability distribution learned from G to acquire new samples. The learning process can be viewed as a minmax game, where one tries to minimize the loss function

$$\boldsymbol{\theta}^{(G)*} = arg \underset{\boldsymbol{\theta}^{(D)}}{min} \underset{\boldsymbol{\theta}^{(G)}}{max} V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}), \tag{8}$$

where $\boldsymbol{\theta}^{(G)}$ and $\boldsymbol{\theta}^{(D)}$ are, respectively, the parameters of G and D. The function $V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ is defined as

$$\begin{aligned} V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) =& \frac{1}{2}\mathbb{E}_{x \sim p_{data}} \, log \, D(x) + \\ & \frac{1}{2}\mathbb{E}_z \, log \, (1 - D(G(z))). \end{aligned} \tag{9}$$

The Learning process can be viewed as a game between two adversaries: G, that tries to deceive D, and D, that tries to distinguish between true and fake inputs. We can represent such idea through the dataflow graph in Figure 6.

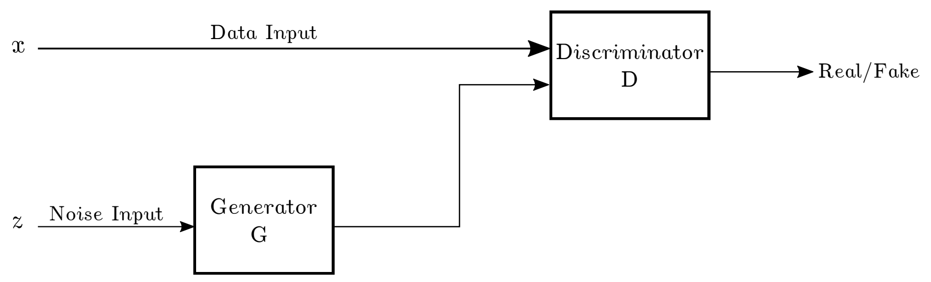

Figure 6: Dataflow graph of a GAN. After training, we can sample z from a probability distribution and give it as an input to G which outputs an image to deceive the discriminator D. If learning has been successful, the generated images should resemble those from the dataset.

Despite the attractiveness from the sampling strategy, these networks are rather difficult to train, and the samples drawn from $\hat{p}_{model}$ are rather poor when compared to the actual data. However, GANs have been used in other kinds of applications in data augmentation. For instance, [20] has applied a variation from GAN's algorithm to perform Image-To-Image Translation, which can be applied to object transfiguration or season transfer.

As authors mention, for the task of unpaired image translation, one has two sets of images, X and Y, for which there is no pairing between sources $x_i \in X$ and targets $y_j \in Y$ in the translation process.

To do so, authors in [20] have used a GAN to train a mapping $G : X \rightarrow Y$, such that the outputs $\hat{y} = G(x)$ are indistinguishable from $y \in Y$. Moreover, G induces a probability distribution on the outputs $\hat{y}$, which tries to represent $p_Y(y)$. However, due to the unpaired setting of the problem, there are infinitely many maps G that induce the same probability distribution, introducing problems in the learning algorithm.

To overcome such obstacle, they have proposed a new architecture based on GANs, the Cycle-Consistent Adversarial Network (CycleGAN), inspired in the process of translation. Instead of one mapping, such architecture has two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$ which should be inverses of each other. To encourage such property, they add to the overall loss function a cycle consistency loss, so that $F(G(x)) \approx x$ and $G(F(y)) \approx y$,

$$\mathcal{L}_{cyc}(F, G) = \mathbb{E}_X[||F(G(x)) - x||_1] + \mathbb{E}_Y[||G(F(y)) - y||_1]. \tag{10}$$

3.3 Style Augmentation

Although the technique described in Section 3.1 was a first step into establishing an algorithm to perform Style Transfer between images, it is not scalable, since for every new source and target images, the model needs to be retrained from scratch. One possible solution, instead, is to substitute the synthesized image generation process by an auxiliary network, the *Style Transfer Network* [21], T.

To deal with the problem for generalizing the style transfer process, the authors have also proposed to include a *Style Embedding Network*, P, which is another auxiliary Neural Network whose purpose is to generate an embedded representation S. Instead of modifying the synthesized image, the learning strategy changes into training the parameters of P, and T, as we can see in Figure 7

With such architecture, [22] proposes a novel technique to perform the randomization of styles within the figures of a given dataset. Specifically, they trained Figure 7 architecture in the Painters By Number[1] dataset, composed of 79.433 labeled paintings. Once the model is trained, a Gaussian Distribution is fitted to the target images style embedding, that is, the mean $\hat{\mu}$, and covariance matrix $\hat{\Sigma}$ are computed,

$$\hat{\mu} = \mathbb{E}_s[P(s)], \tag{11}$$
$$\hat{\Sigma}_{i,j} = Cov(P(s)_i, P(s)_j), \tag{12}$$

so that random style embeddings, z, are sampled from a probability distribution weighted by a hyperparameter α,

[1]Available at `https://www.kaggle.com/c/painter-by-numbers`

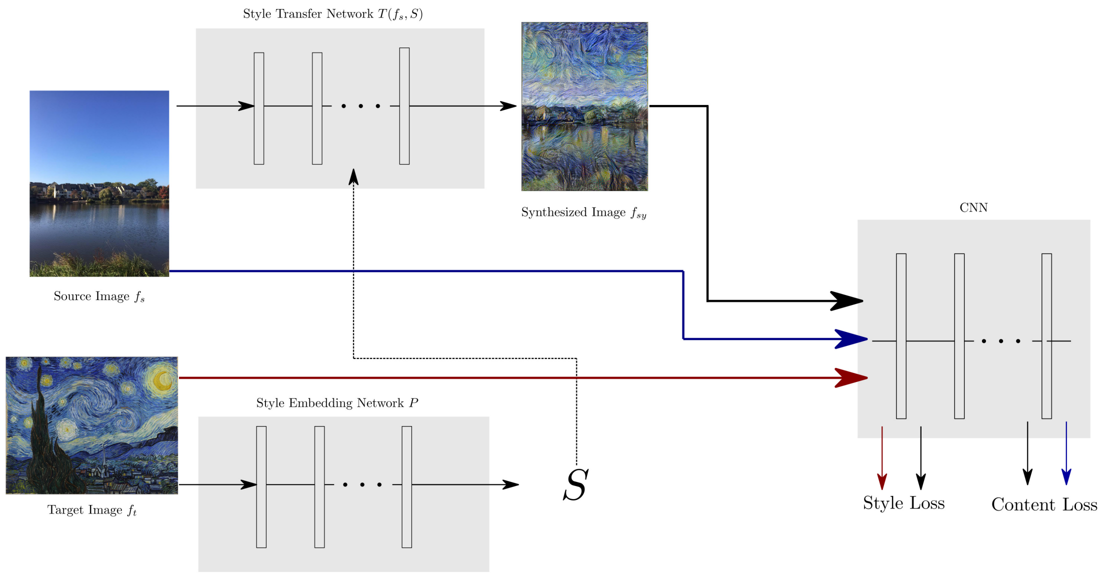
Style Transfer Network $T(f_s, S)$
Synthesized Image f_{sy}
Source Image f_s
CNN
Style Embedding Network P
Target Image f_t
S
Style Loss
Content Loss

Figure 7: Real time style transfer schematic, adapted from [21]. The Style Transfer Network T purpose is, given a source image f_s, and an embedded representation of S, to generate f_{sy}. The networks T and P are trained, to minimize the Style Loss and Content Loss already defined in section 3.1.

$$z \sim \alpha\mathcal{N}(\hat{\mu}, \hat{\Sigma}) + (1-\alpha)P(s). \tag{13}$$

In such setting, the result augmented dataset has a larger variety of different styles, as well as a greater number of samples. As the authors remarks, Data Augmentation does not limit itself to enlarging the number of available data. Indeed, Early applications, involving geometric data transformations, had the goal to provide geometric invariance to classifiers (for rotation and translation, for instance). With Style Augmentation, the motivation is similar: with training samples coming from a variety of distinct texture distributions, the trained classifier can be invariant to style perturbations.

3.4 AutoAugment

Despite the positive results in the usage of hard-coded image transformations to enlarge datasets, this approach faces huge pitfalls when applied to large scale problems, due to time consumption and domain-expertise. A rather desirable approach would be to, from a given dictionary of transformation, find those which best improve accuracy of the classifier for a given input. The idea can be formalized in the Reinforcement Learning framework, where an Agent tries to find the best policy among the possible ones, which are the data transformations among the cited dictionary.

This is the approach taken by [23], where authors have created an algorithm to learn optimal augmentation policies. They compose the Agent search space of 16 possible transformations.

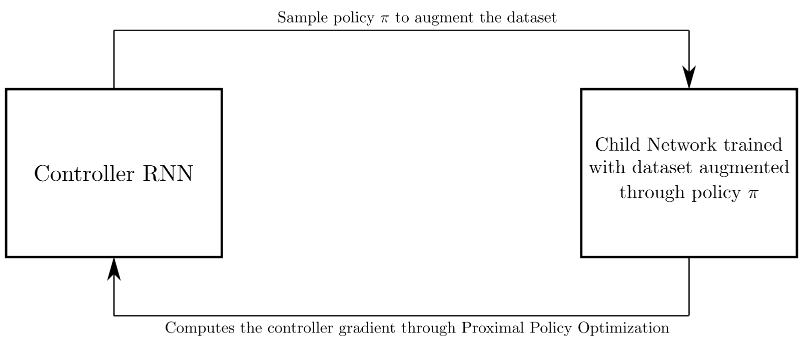
Sample policy π to augment the dataset
Controller RNN
Child Network trained with dataset augmented through policy π
Computes the controller gradient through Proximal Policy Optimization

Figure 8: The AutoAugment algorithm: first, the controller RNN samples an augmentation policy π, so that the Child network is trained in the resulting dataset. After that, the network's gradient is calculated by the Proximal Policy Optimization algorithm, so that the controller learns optimal data transformations.

Then, each policy is composed by 5 sub-policies. The sub-policies, at its turn, are composed by a given transformation, its intensity and probability of being applied.

The learning algorithm makes use of a controller network (a RNN) which generates other "child networks", with fixed structure, but being trained following the dataset augmented with the optimal policy. Such network is trained following the Proximal Policy Optimization Algorithm [24]. The overall strategy is shown in Figure 8.

4 Conclusion

With increasing complexity in state-of-the-art machine learning models, data augmentation has become an important technique to enhance the model quality, by increasing invariance and reducing overfitting. In the application of these methods, it is noteworthy how much deep learning models have to contribute with main methodologies, and how these can be later automated, so that they do not depend on human hard coding.

In this report we have reviewed most deep learning architectures used for data augmentation, as well as how they are used to perform it. Particularly, CNNs and GANs have been used for two tasks very similar in their outputs, but distinct in their settings, that is, Neural Style Transfer and Image-to-Image Translation. Another important topic discussed was how standard transformations, such as Geometric Transformations and Elastic Deformations act on data.

The future work will focus to contribute in how deep learning can be used in automatizing and optimizing these later techniques, such as discussed in Style Randomization, and in AutoAugment. These techniques share the feature that transformations are not performed by hand, saving the need for domain-expertise. AutoAugment, however, despite its rather complex search algorithm, performs optimally with respect to its policies, while Style Randomization does not. To cope with this complexity, a possible approach could rely on simpler search algorithms, such as Evolutionary Computing, or Random Search, given the search-space defined by the policies in the AutoAugment setting.

Acronyms

CNN	Convolutional Neural Network. 4, 7, 12
CycleGAN	Cycle-Consistent Adversarial Network. 10
DBSCAN	Density-Based Spartial Clustering of Applications with Noise. 6
DBSMOTE	Density-Based Synthetic Minority Over-sampling Technique. 6
DNN	Deep Neural Network. 3, 7
GAN	Generative Adversarial Network. 7, 9, 10, 12
RNN	Recurrent Neural Network. 7, 12
SMOTE	Synthetic Minority Over-sampling Technique. 5, 6

References

[1] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122. IEEE, 2018.

[2] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? *arXiv preprint arXiv:1609.08764*, 2016.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[7] Jin Yamanaka, Shigesumi Kuwashima, and Takio Kurita. Fast and accurate image super resolution by deep cnn with skip connection and network in network. In *Neural Information Processing*, pages 217–225. Springer, 2017.

[8] Peng Liu and Ruogu Fang. Wide inference network for image denoising. *arXiv preprint arXiv:1707.05414*, 2017.

[9] Weisheng Dong, Peiyao Wang, Wotao Yin, Guangming Shi, Fangfang Wu, and Xiaotong Lu. Denoising prior driven deep neural network for image restoration. *arXiv preprint arXiv:1801.06756*, 2018.

[10] Jan Jantzen, Jonas Norup, Georgios Dounias, and Beth Bjerregaard. Pap-smear benchmark data for pattern classification. *Nature inspired Smart Information Systems (NiSIS 2005)*, pages 1–9, 2005.

[11] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[12] George Wolberg. Image morphing: a survey. *The visual computer*, 14(8):360–372, 1998.

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[14] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Dbsmote: density-based synthetic minority over-sampling technique. *Applied Intelligence*, 36(3):664–684, 2012.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[16] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[17] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 262–270, 2015.

[18] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[20] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*, 2017.

[21] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *arXiv preprint arXiv:1705.06830*, 2017.

[22] Philip T Jackson, Amir Atapour-Abarghouei, Stephen Bonner, Toby Breckon, and Boguslaw Obara. Style augmentation: Data augmentation via style randomization. *arXiv preprint arXiv:1809.05375*, 2018.

[23] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.