Addressing the Texture-Bias and Domain Generalization Performance of Convolutional Neural Networks for Simulation-Based Learning

Pedro Miguel de Aguiar Coelho
pedroaguiarcoelho@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

January 2021

Abstract

Convolutional neural networks have become the de-facto standard solution for most computer vision problems. Thanks to advances in computer performance and the development of open high-end deep learning libraries, automating a simple vision task only involves gathering a reasonable dataset, that mimics the conditions in which the network will be deployed, and training a suitable network on that dataset, with some extra considerations and tuning to avoid underfitting or overfitting the dataset. This problem becomes more difficult when the dataset on which the network is trained on does not fully represent the scenarios on which the network will be immersed. For example, a network that is trained in a simulated environment may not perform well when it is tested in a real environment, due to the differences between the simulated and real environments. This situation requires networks that are able to generalize at a deeper level. The networks must be robust to changes in their environment. Therefore, in the case of a vision task, they must rely mostly on high-level object shapes rather than low-level image textures to correctly identify objects across environments. Throughout this work we will explore the inner workings of convolutional neural networks, review some of their applications and propose a novel randomized image texturization method that can make make networks rely less on texture and more on the shape of objects.

Keywords: Computer Vision, Deep Learning, Convolutional Neural Networks, Domain Generalization, Texture Bias

1. Introduction

The usage of machine learning models in the critical systems of vehicles and aircraft has long been desired to automate their functions. Good examples include the increased interest in autonomous driving [1] and the recent demonstration of full autonomous flights by an Airbus commercial aircraft [2]. One of the key steps to the deployment of a good computer vision solution, besides the choice of a trustworthy algorithm, is the construction of a large and representative dataset on which the algorithms can be successfully trained. This involves the manual annotation of thousands of images which can take more than one hour each, depending on the task, to achieve good results.

One possible solution to this problem would be the training of such algorithms in a simulated environment, like a driving simulator or a video game, where different objects in a scene can be automatically annotated, thus saving thousands of hours of human labor and allowing the creation of diversified training scenarios. Training on a simulator would allow the algorithm to learn in many different and rare scenarios that it would not observe often enough on real training images, thus guaranteeing more control over those situations.

Although the Convolutional Neural Network (CNN) architecture offers outstanding performance in image processing tasks, it does so only when the images on which it is tested on come from the same domain as the ones that were used to train the algorithm. This is the problem of out-of-domain (OOD) performance degradation. A domain shift may include simple transformations like a colour shift, noisy images, camera artifacts, camera position, image scale, or more complex differences like the fact that an image of an object can be a photograph, a drawing, a painting or the result of a render from a simulation. Furthermore, real world images themselves can exist in multiple domains, such as different times of the day, lighting conditions and weather. Any of these domain shifts can catastrophically affect the performance of CNNs. The problem of domain adaptation/generalization remains an open problem with no algorithm being capable of achieving human-level generalization to new unseen domains and with any unknown domain shift.

2. Background

The backbone of many modern computer vision and image processing algorithms are Convolutional Neural Networks. This family of architectures mimics the visual system of animals [3] and humans [4], where low-level features like points, edges and textures are combined to form mid-level features like shapes, curves and more complex textures. These mid-level features are then combined to form high-level features like faces, bodies, animals, objects, locations and backgrounds (see Figure 1). Finally the high level features are used to perform various tasks such as object classification, detection or image segmentation.

Figure 1: Feature Hierarchy leaned by GoogLeNet on the ImageNet dataset.

Gradient based CNNs were first applied to image processing by [5] in 1989 for digit recognition and considerably outperformed any previous technique. They were later used for tasks such as document recognition in 1998 [6].

It wasn't until the increase in computational power, its parallelization using Graphical Processing Units (GPUs) and the construction of large labeled image datasets such as ImageNet [7], that a breakthrough occurred in natural image processing. In 2012 [8] introduced AlexNet, a deep CNN capable of classifying images into 1000 different classes with a high degree of accuracy. Alexnet was trained by leveraging the parallel processing power of Graphical Processing Units to achieve a great increase in model capacity. From 2012 onwards, novel architectures established new state-of-the-art performances every year, such as deeper VGG (Visual Geometry Group) networks [9] and residual networks [10], which surpassed human-level performance.

Besides image classification, convolutional architectures also found application in algorithms for object detection [11] and semantic segmentation [12], in Auto-Encoders [13] and in Generative Adversarial Networks [14]. These algorithms can then be used in many types of downstream tasks such as self-driving [15], robotics [16], visual tracking [17], image captioning [18], image super-resolution [19], image colorization [20], image style transfer [21] and deep dreaming [22].

2.1. Multi-Layer Perceptron

A Perceptron (commonly called Neuron) is the most basic unit of a neural network. It receives several inputs x_i and multiplies each of them by a weight w_i. The resulting values are added together, alongside an extra bias term b, and passed through an activation function f, as seen in Figure 2. A single Perceptron can perform linear regression when a linear activation is used and can perform logistic regression when using a Sigmoid activation function.

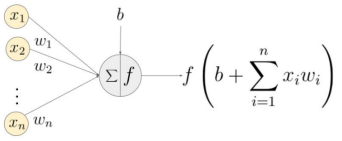

Figure 2: Schematic of a single Perceptron.

Multiple Perceptrons with the same inputs can be combined to form a fully connected layer, also called a dense layer. A Multi Layer Perceptron (Figure 3) is created by composing multiple of these layers in sequence, forming a neural network:

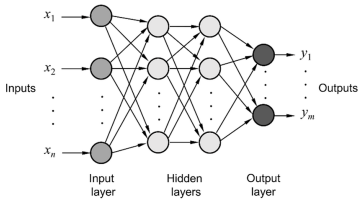

Figure 3: Schematic of a Multi Layer Perceptron.

In a regression task, the parameters from all the neurons (weights and biases) will be learned such that the neural network function approaches, as closely as possible, a set of points $(\mathbf{x}, \mathbf{y})_i$ from a training dataset. In a classification task, the weight will be learned such that, given an input, it produces vector with the likelihood of belonging to each of the possible classes. The objective is to predict a high likelihood for the true class and, by construction, a low likelihood for the incorrect classes. Since we want to estimate the likelihoods of every class as the result of the last layer, we want the sum of the components of the output vector to equal 1. To accomplish this, the activation function of the last dense layer is replaced by the Softmax function. The networks are trained by minimizing a loss function that compares the expected output and the network predictions. This loss fucntion is minimized when the network prediction match the expected output.

The process of training a neural network is comprised of three operations that are repeated through many iterations until the network performs well. The first operation is forward propagation, where the inputs are propagated through the network and

the final loss is calculated. All neuron activations are saved so that the next step, back-propagation, can occur. This second step involves using the chain rule of derivation to calculate the gradient of the loss function with respect to every single learnable parameter. The final operation consists of performing an update of the weights in the opposite direction of the gradient, thus slowly minimizing the loss function.

2.2. Convolutional Neural Networks

Convolutional Neural Networks are deep neural networks that apply at least one convolution operation. When used for image processing, two dimensional convolutions are employed to take advantage of the spatial relationship between the pixels of an image. The relationship between a pixel and its closest neighbours is far more important than its relationship with the pixels on the other side of the image. Therefore it is far more efficient to compute inter-pixel relationships using kernel convolutions. A convolution operation can be described as sliding a kernel matrix over a larger input matrix. The values of the input matrix are multiplied element-wise by the weights in the kernel matrix and finally added together to produce a value in the resulting output matrix. Each element in the resulting matrix corresponds to one position of the kernel over the input matrix, as seen in Figure 4.

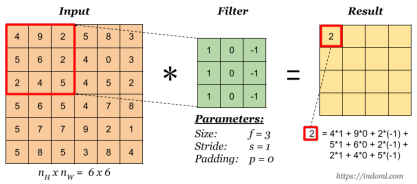

Figure 4: Convolution Operation.

A convolution layer performs multiple operations of this type. The input consists of multiple matrices, commonly called feature maps or channels. A filter is comprised a kernel matrix for each of the input channels. A convolution operation is performed for each of the kernels in the filter and the corresponding input maps. The results of these operations are added element-wise to produce a single output feature map. A convolution layer can have many of these filters, creating multiple output feature maps, as shown in Figure 5. Since convolutions are linear operations, a non-linear function is applied to each output channel to give the network non-linear regression capabilities.

A convolutional neural network consists of the composition of many of these layers to produce a complex set of features that can be easily used to produce accurate image classifications. This process is called feature extraction and is done automatically through the learnable parameters of the

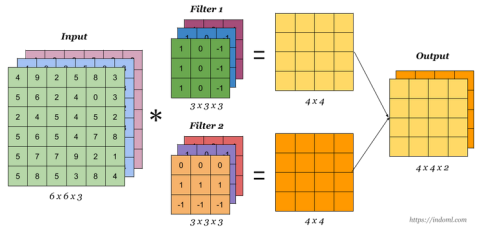

Figure 5: Multi-filter convolution with 3 input channels (RGB image), and 2 output channels.

network. Image classification CNNs are completed with a classification head. This classification head transforms the final feature maps by flattening them into a 1 dimensional vector and feeding it through a fully connected Multi Layer Perceptron, as in Figure 6. Training the full network is done with the gradient descent algorithm described in section 2.1, where now the dataset contains tuples of images and the respective labels.

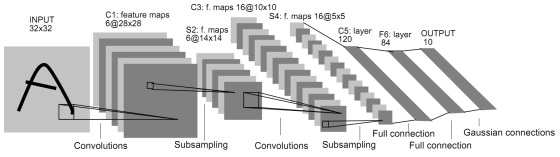

Figure 6: LeNet - Convolutional architecture used for character recognition in [6], where each plane is a feature map and the final 3 layers are fully connected.

Until now we described convolutional neural networks in the context of supervised image classification, assuming that each image only portraits a single object of interest. In typical real-world application this framework must be extended to images where one or more objects may be present, or none. The algorithms should therefore be able to identify all the objects in the images as well as their respective positions. We shall cover some of these applications since they will be the object of experiments later in this work.

2.3. Semantic Segmentation

Semantic Segmentation consists of correctly labelling each pixel in an image according to what class of objects it belongs to. The first use of a deep convolutional neural network for semantic segmentation was in [23], where fully convolutional neural networks (FCNs) were shown to be capable of semantically segmenting images. The method consists of simply taking a section of a common architecture like a VGG16 network, adding an extra convolutional layer with the number of channels equal to the number of classes, up-sampling the resulting feature maps to the input size and finally passing it through a soft-max layer. Each feature map encodes therefore the network predictions of

the probability of each pixel belonging to a class of object. These predictions are compared to the true semantic maps with a cross-entropy loss, which is minimized through a gradient descent algorithm. Unknown objects and background pixels are usually assigned their own default class. To perform inference, the class with highest probability is assigned to each pixel. This architecture is improved by adding extra prediction layers at the early convolutional layers of the encoder, upsampling the predictions to the full resolution and combining the results of all prediction layers. Figure 7 showcases the semantic segmentation task with a simple FCN architecture.

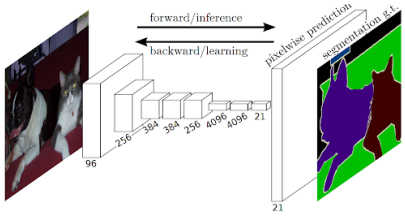

Figure 7: Schematic representation of a basic FCN architecture for semantic segmentation.

Deeplab [24] is one of the most used architectures for semantic segmentation. It employs dilated (also called atrous) convolutions. This type of convolution uses dilated (sparse) kernels, which increases the effective receptive field of each neuron without increasing the computational cost of the layer.

The predictions of the network can be evaluated using different performance metrics. The most commonly used metric is the mean Intersection over Union (mIoU) between semantic predictions and the true semantic segmentation maps. Let Ω be the set of all pixels in the image, $P_k \in \Omega$ the set of predicted pixels for a class $k \in [1, K]$, $T_k \in \Omega$ the set of true pixels for a class k, and $|.|$ the cardinality operator. The mIoU is calculated as:

$$mIoU = \frac{1}{K} \sum_{k} \frac{|P_k \cap T_k|}{|P_k \cup T_k|} \tag{1}$$

Later in this work, the FCN and Deeplab semantic segmentation architectures will be subject to different experiments in the context of domain generalization.

2.4. Object Detection

Object Detection consists of drawing a bounding box around each known object in an image and correctly predicting its class. Passing from an image classification task to an object detection task can seem trivial. To detect multiple objects in an image simply pass an image classifier in multiple patches of the input image and consider a detection when the confidence (probability prediction) for an object is above a certain detection threshold. This can be done in a rolling window fashion, at multiple scales and aspect ratios. The problem becomes apparent when we consider the number of forward passes that a classifier would have to perform to detect all the objects in a single image.

Some applications of object detection algorithms require the detection to happen in real time, so performing classification of all possible image patches becomes infeasible. Even if this process is parallelized to reduce the computation time, the shear amount of floating point operations is completely impractical.

R-CNNs [11] were the first improvement to this method. They use a region proposal step, based on a selective search algorithm [25], to select the Regions of Interest (ROIs) that are more likely to contain an object. The selected image patches are further filtered by combining patches that overlap or are included in each other. The different patches are then resized to a fixed square resolution. A convolutional neural network processes each patch into a feature vector. Finally, a SVM classifier [26] predicts the probability of each class from this feature vector. The feature vector is also used to predict an offset of the ROIs to produce the final bounding boxes.

Faster R-CNN [27] further improves this architecture by substituting the selective search algorithm by a region proposal network (RPN). The region proposal network is based on anchors, which are predefined image patches, at different scales and aspect ratios. The region proposal network calculates, for each of these anchors, a *objectedness* score indicating the probability of an object being present in that patch. This is achieved with one or more convolutional layers where the final layer has a number of channels equal to the number of anchors. The feature maps therefore represent the *objectedness* score at each pixel for each possible anchor. In the same manner as R-CNN, a final classifier layer produces the class probabilities and a final regression layer produces the bounding box coordinates from the feature vector. The full architecture is presented in Figure 8.

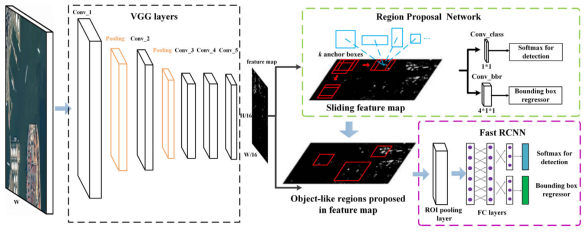

Figure 8: Schematic of the Faster R-CNN architecture.

The Faster R-CNN architecture will be tested later in this work in a domain generalization task.

The performance of object detection algorithms is typically measured using the mean Average Precision (mAP) and mean Average Recall (mAR) metrics. These are constructed by setting different Confidence C and Intersection over Union IoU thresholds, above which a detection respectively occurs and is considered correct. The performance metrics are thus the precision and recall scores averaged over a range of these thresholds.

$$AP_k = \frac{1}{N_{IoU} N_C} \sum_{IoU,C} \frac{TP}{TP + FP} \Bigg|_{IoU,C,k} \tag{2}$$

$$AR_k = \frac{1}{N_{IoU} N_C} \sum_{IoU,C} \frac{TP}{TP + FN} \Bigg|_{IoU,C,k} \tag{3}$$

Finally, the performance metrics are constructed by averaging these values across classes to produce the mean Average Precision and the mean Average Recall.

2.5. Auto-Encoders

Auto-Encoders (AEs) are a type of neural network architecture whose purpose is to learn a lower dimensional representation of the data by passing it through an information bottleneck and using that low dimensional representation to reconstruct the original data [28, 29]. An Auto-Encoder is composed of an encoder network which maps inputs to the lower dimensional latent space, and a decoder network which maps the latent space to the original higher dimensional input space (Figure 9). This procedure forces the network to learn an internal latent representation that captures as much information about the data as possible. Auto-Encoders find application in unsupervised representation learning, image denoising, image compression and image inpainting because of their ability to filter out irrelevant information. Auto-Encoders are usually trained with a reconstruction loss which can be a simple L2 distance loss, another distance metric or even a pretrained and frozen neural network based perceptual loss [30]. When applying Auto-Encoders to images the encoder and decoder networks are implemented as a vanilla CNNs with transposed convolutions or other upsampling layers for the decoder.

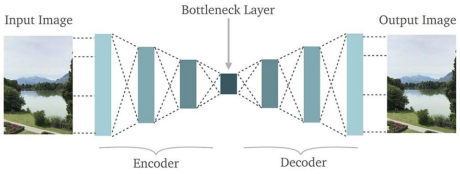

Figure 9: Example of a Fully Convolutional Auto-Encoder.

2.6. Style Transfer

Image style transfer is the task of taking the content from an input image $\mathbf{x}$ and re-producing it in the style of a style image $\mathbf{s}$. The first instance of CNN based style transfer happens in [21]. This was achieved by optimizing the pixel values of the content image to minimize a content loss and a style loss. Since this involves an optimization process, producing a single stylized image is an expensive process. Adaptive Instance Normalization (AdaIN) opened the door to arbitrary style transfer [31], where a single auto-encoder network, when fully trained, can perform style transfer between any two images in a single pass.

AdaIN is a moment-matching process. It is assumed that the style information of the images is fully represented in the first and second order moments of the feature maps produced by a pretrained encoder network. The normalization process of AdaIN is the following:

$$AdaIN(\phi^{\mathbf{x}}, \phi^{\mathbf{s}}) = \sigma(\phi^{\mathbf{s}}) \left(\frac{\phi^{\mathbf{x}} - \mu(\phi^{\mathbf{x}})}{\sigma(\phi^{\mathbf{x}})} \right) - \mu(\phi^{\mathbf{s}}) \tag{4}$$

where $\phi^{\mathbf{x}}$ and $\phi^{\mathbf{s}}$ are the latent feature maps of the content and style images respectively, $\mu()$ is the channel-wise mean and $\sigma()$ is the channel-wise standard deviation of the latent feature maps.

The content and style losses evaluate the images produced by the decoder by re-passing them through the encoder and comparing the feature maps to the original ones. The content loss is given by the mean squared reconstruction error. The style loss is given by the mean squared error in the first order and second order moments across all channels. This loss is averaged across the feature maps produced at multiple layers of the encoder. The final loss is the sum of the two losses weighted by an hyper-parameter. The network is trained to empirically minimize the expected value of the loss through gradient descent. The complete style transfer network is shown in Figure 10.

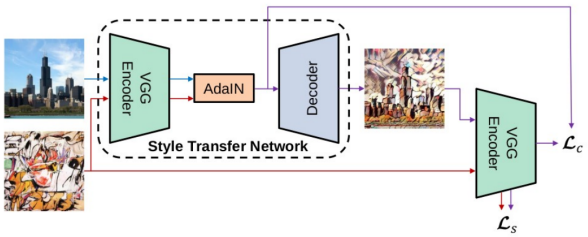

Figure 10: AdaIN architecture.

The encoder is a VGG network pre-trained on ImageNet with a classical image classification task. Learning the style transfer task consists of training a decoder that is able to inverse the encoder when

the feature map statistics are altered by AdaIN. This stylization technique is therefore an example of transfer learning, since the representations learned by the encoder are used in a different downstream task. The full style transfer network is then trained using a content dataset and a style dataset.

2.7. Texture Bias in Convolutional Neural Networks

It is commonly assumed that CNNs owe much of their performance to their ability of recognizing shapes. Many feature visualization techniques [32, 22, 33] show that CNNs effectively learn edge detectors, curve detectors, circle detectors, animal face detectors and human face detectors. The multi-layered and translation invariant architecture allows the network to combine multiple of these shapes to create more high-level shape detectors. It was thought that the structured combination of these features produced networks that were sensible to high level shape cues.

However, some recent studies showed that CNNs are in fact more texture biased than previously thought. The authors of [34] used the original iterative style transfer technique [21] to generate a Cue-Conflict dataset, based on ImageNet, with conflicting shape and textural cues. When these images are presented to CNNs and to human subjects, the CNNs are more likely to classify the image based on texture/style than their human counterparts (Figure 11). The human subjects, on the other hand, classified images mostly based on shape/content cues.

Figure 11: Classification of a standard ResNet-50 of (a) a texture image (elephant skin: only texture cues); (b) a normal image of a cat (with both shape and texture cues), and (c) an image with a texture-shape cue conflict, generated by style transfer between the first two images.

This study then showed that the texture-bias of CNNs can be attenuated by training the network in another randomly stylized version of ImageNet. This works as a data augmentation and regularization technique. By making the textural cues arbitrary with respect to the content, the network must learn to classify the content of the image purely based on the shape patterns. The data augmentation thus renders the network texture invariant. Furthermore, networks that are trained on the stylised dataset show more resistance to class-preserving image transformations like, changes in contrast, filtering and different forms of noise.

2.8. Generalization in Convolutional Neural Networks

In deep learning tasks it is expected that the training and testing data come from the same distribution. When this is not the case, we are presented with an out-of-distribution (OOD) generalization problem. This task consists of learning a source distribution and being able to perform inference in a target distribution with different characteristics. This problem can be further decomposed depending on our access to unlabeled data from the target distribution during training.

In a Domain Adaptation (DA) framework, a network is trained using the source distribution and its generalization performance is boosted by taking advantage of unlabeled samples from the target distribution. In a Domain Generalization (DG) framework, only the source distribution is accessible and the objective is to generalize to any other distribution without any prior knowledge of its nature. Furthermore, each framework can be subdivided into single-source or multi-source scenarios, according to the number of source distributions that are available during training.

A recent massive experiment [35] performed by the Facebook AI Research (FAIR) group tested many single-source and multi-source domain generalization algorithms and model selection (validation) strategies using the same network architecture across different domain generalization benchmark datasets. It was concluded that a vanilla supervised learning algorithm, when trained with a good data augmentation strategy can generalize just as well or even better than most algorithms purposely built for domain generalization. It was also found that the model selection strategy was far more impactful than the choice of algorithm. This indicates that the generalization performance is strongly dependent on the network's capacity, which can be maximally used with simple data augmentation techniques, suggesting that the key to achieving good domain generalization results can simply be a good data augmentation strategy.

This work continues the research effort on domain generalization by randomized domain shifting and data augmentation, which could potentially be effective in reducing the texture bias of CNNs and improve their effectiveness in domain generalization tasks.

3. Domain Shifting Experiments

Given the recent research work in Domain Generalization based purely on data augmentation, we wish to construct a method that randomly alters image textures, such that networks trained on those images become less sensitive to textures and more sensitive to the shapes of objects.

A solution to this problem could be the AdaIN

stylization method, which is based on an auto-encoder architecture with a VGG16 encoder and decoder. Although this architecture may be altered to generate random styles, these styles are heavily conditioned on a style dataset of paintings. Training this architecture without a style dataset generates styles that are much less expressive and varied, only consisting of slight colour variations. These results, coupled to the fact that this method is also considerably expensive computationally, make this method unfit for the purpose of data augmentation.

In this work we propose a second solution, also based on an auto-encoder architecture, which solves the mentioned problems.

Ideally, a random stylization method should be able to reconstruct the input image with a high degree of fidelity when no noise is introduced in the auto-encoder network. The AdaIN stylization method achieves this when the same image is used for content and style. A data augmentation method should be lightweight enough to not cause a considerable computational overhead on the main method. It should also be as general as possible, in a sense that, it should produce all possible variations of the transformation that it applies.

With these considerations in mind, a different approach is therefore devised where a simple convolutional auto-encoder is used to learn a latent representation of the textures in an image. The auto-encoder has no information bottleneck. Instead an encoder converts spatial information into channel-wise information using down-sampling with strided convolutions. This means that each pixel in the latent feature maps encodes the texture of a patch in the input image. A decoder based on transposed convolutions is then tasked with reconstructing the input image. A Mean Squared Error (MSE) reconstruction loss is used between the input and output images. An extra KL Divergence loss is used in the latent space to softly constrain the representations to have $\mu = 0$ and $\sigma = 1$. The encoder and decoder can have 2 to 4 layers each, depending on how wide we want the texture patches to be and how non-linear we want the transformation to be. The auto-encoder is trained on a set of content images using these two losses. Since there is no information bottleneck, the auto-encoder should be able to perfectly reproduce the input images and have an MSE loss that tends to 0.

The purpose of the encoder-decoder architecture is not to compress the information of the images. It is rather to encode the images into a texture-space, where they are transformed and then decoded back. Therefore the total volume ($C \times W \times H$) of the feature maps should remain constant at every layer of the encoder. When a down-sampling is performed, the number of channels should increase by the same ratio i.e. dividing the resolution by 3 in height and width, should be accompanied by a 9 times increase in channel number. The information is therefore encoded channel wise and not pixel wise, creating the aforementioned texture-space.

To perform randomized texturing of images the auto-encoder is frozen and a transformation is introduced in the latent space. The full architecture, presented in Figure 12, consists of an encoder E, a transformation T that introduces the noise in the feature maps created by the encoder, and a decoder D that generates the transformed images.

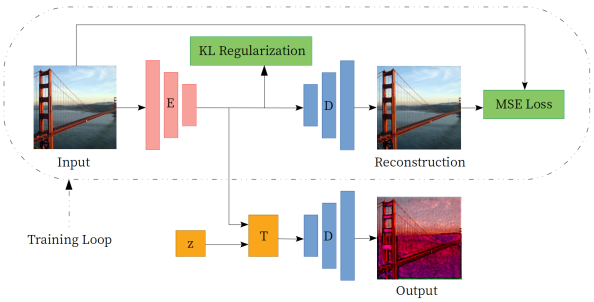

Figure 12: Encoder-Transformer-Decoder architecture. During training, the Encoder (E) and Decoder (D) are trained together, using the MSE Loss and a KL Divergence regularizer. At inference time, a Transformer (T) transforms the latent space, according to a noise vector z, before decoding the image.

The key element of this architecture is the choice of transformation T, which is responsible for introducing the noise vector z in a meaningful manner.

The following experiments showcase the type of randomized texturizations that is possible to obtain with the previously mentioned techniques. The encoder has 2 convolutional layers with a Hyperbolic Tangent activation in between. The decoder has 2 transposed convolutional layers with the same activation in between.

This Auto-Encoder is trained on the MS-COCO dataset for 25000 iterations with a batch size of 16 and a learning rate of 1×10^{-4} with Pytorch's default ADAM optimizer. All images are resized to 505×505 pixels during training. The latent feature maps have 243 channels and 57×57 resolution ($((505 + 2)/3 + 2)/3$). The two convolutions with a stride of 3 generate latent representations that encode the texture in a 9×9 patch.

Once the Auto-Encoder is fully trained, a transformation can be performed in the space of textures which results in an output image that is transformed in a spatially consistent manner (Figure 13). The transformation can be a simple translation in the space of textures, a rotation or an AdaIN transformation. This technique allows the augmentation of images with randomized textures with a very small computational overhead when compared to

the Style Transfer method.

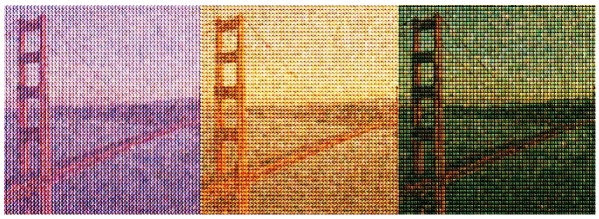

Figure 13: Textured versions of the same image with a translation transformation of the latent space.

4. Domain Generalization Experiments

The proposed texturization technique was then tested as a data augmentation method in multiple computer vision tasks, in multiple scenarios and with multiple architectures of the main network. Figure 14 shows examples of textured images from the STL10 dataset.

Figure 14: Examples from the STL10 dataset with a random texturization applied. Each column represents one class in the following order: bird, car, cat, deer, dog, plane, ship, truck. These images were used for training the networks.

A first round of experiments measured the texture bias of a ResNet34 network on the STL10 classification dataset. The texture bias was measured using mixed-cue images, with the content of one class and the style of another, as seen in Figure 15.

Figure 15: Examples of style transfer between different classes of STL10. Top row: content images, Middle row: style images, Bottom row: resulting images. These images were used to evaluate the networks.

These images were generated using the aforementioned AdaIN Style Transfer technique applied to images of different classes on the STL10 dataset. The texture bias is the proportion of times the network predicts the class that was used as style instead of the class that was used for content, when one of them is predicted.

The texture bias was measured on the baseline training setting, without textured images, and then with textured images at different levels of texturization strength. Each experiment was reproduced three times, to produce the mean and standard deviation results of Figure 16.

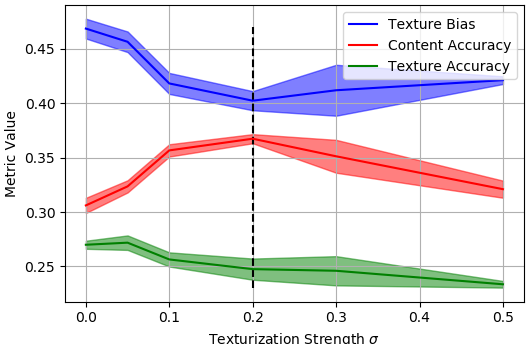

Figure 16: Texture Bias, Content Accuracy and Texture Accuracy while varying the values of the texturization strength σ. The dashed line represents the optimal value of σ.

It is concluded that using textured images during training of the ResNet34 network produces a network that is less texture-biased and that achieves a better accuracy in identifying the content of the mixed-cue images. This phenomenon happens until a point at which the texturization is too strong and network becomes incapable of learning due to underfitting.

In a second round of experiments, this method is tested in a semantic segmentation domain generalization task. Two network architectures, a VGG16 Feature Pyramid Network and a ResNet101 DeeplabV2, are trained to segment road scene images in one simulated environment (GTA5 or Synthia) and are tested in real road scenes (Cityscapes), as seen in Figure 17.

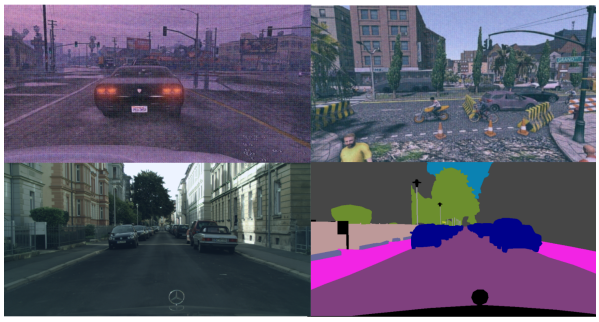

Figure 17: Textured textured images from GTA5 and Synthia (top), an unchanged image from Cityscapes and the respective segmentation map (bottom).

In this more challenging setting, using textured images during training always produces worst results than the baseline training scenario (Figure 18).

This indicates that challenging tasks like semantic segmentation domain generalization do not benefit from the use of this data augmentation strategy. The performance in the training dataset also decreases as the texturization strength increases, which leads to the conclusion that the use of textured images during training produces a severe underfitting phenomenon. This could have been expected in the semantic segmentation task since this task highly benefits from textural information and requires the predictions of the network to be accurate to the pixel level.

Training Dataset	GTA5				Synthia			
Architecture	ResNet101		VGG16		ResNet101		VGG16	
Data Augmentation	V	V+T	V	V+T	V	V+T	V	V+T
mIoU	**38.4**	16.7	**37.1**	32.1	**27.9**	16.5	**29.7**	23.6

Figure 18: Mean Intersection over Union (mIoU) results of the semantic segmentation experiments, with Vanilla data augmentation (V) and Textured images (T).

A final round of experiments measured the effect of this technique on the domain generalization performance in an object detection task. Three Faster R-CNN architectures, with VGG11, VGG19 and ResNet50 backbones, are trained on the VirtualKitti2 simulated road scene dataset and tested on the Kitti real road-scene dataset. The task consisted of detecting all the vehicles in an image, as shown in Figure 19.

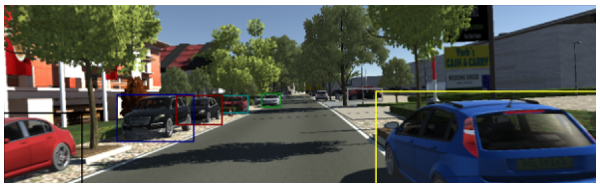

Figure 19: Example image from the Virtual Kitti 2 dataset with respective vehicle bounding boxes.

Once again, this task proved too difficult when training the networks with textured images and the overall performance decreased. However, the performance is mostly impacted when detecting small objects (far away vehicles). The proposed method increases the domain generalization performance in the case of vehicles that occupy regions with more than 96×96 pixels, in an image that is more than 1000 pixels wide (Figure 20). This indicates that this technique performs as intended but it renders the task of detecting small objects much more difficult and thus the overall result is negative.

5. Conclusions

In section 3 we explored different domain shifting methods. Until now, image domain augmentation used to be performed either with simple transformations like image translations, rotations,

Metric	IoU	Area	MaxDets	VGG11		VGG19		ResNet50-FPN	
				N	T	N	T	N	T
mAP	0.50 : 0.95	Small	100	**0.115**	0.070	**0.170**	0.140	**0.245**	0.170
	0.50 : 0.95	Medium	100	**0.333**	0.307	**0.413**	0.368	**0.440**	0.358
	0.50 : 0.95	Large	100	0.479	**0.498**	0.525	**0.546**	**0.541**	0.538

Figure 20: Mean Average Precisions (mAP) results of the object detection experiments, with no data augmentation (N) and Textured images (T).

scalling, additive noise, other kinds of noise, or with more computationally expensive techniques like style transfer. We first tried generalizing the AdaIN based style transfer method to work without a style dataset, this method generated images that had different textures but the diversity of those textures did not justify the computational resources required just to run the data augmentation process. This work proposed a novel technique for randomly altering the texture of an image. This is achieved with a Fully Convolutional Auto-Encoder architecture, where no information bottleneck exists. The image is down-sampled while being encoded and the number of channels is increased in the same proportion. The inverse process is performed by the decoder. This allows the Auto-Encoder to perfectly reconstruct the images once fully trained. The resulting latent space therefore encodes the square image patches in a single vector per patch, encoding the textures at each patch of the image. This technique is simpler and performs data augmentation in a manner that is very suited CNNs, bringing only a small computational overhead.

In section 4 we showed that the proposed technique successfully renders CNNs less texture-biased in a classification task with mixed textural and shape cues. We then tested our method in more demanding scenarios. The semantic segmentation experiments in domain generalization from a simulated world to a real world revealed that introducing our data augmentation technique during training leads to severe underfitting. This shows that CNN architectures do not have good and efficient mechanisms to filter out textural information. In an object detection scenario the same phenomenon was observed. Interestingly, the technique was shown to work for large object instances but the performance in smaller detections remained severely affected.

The final conclusion is that the proposed data augmentation technique, although it renders CNNs less texture-biased, it does not help their domain generalization performance due to an inefficiency of CNNs when filtering out irrelevant textural information.

Acknowledgements

I would like to thank my family, friends, Conceição Amado, Fernando Lau, IST, ISAE-Supaero, David Vigouroux, DEEL and IRT Saint-Exupéry.

References

[1] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *arXiv:1906.05113*, 2019.

[2] Airbus. Airbus concludes attol with fully autonomous flight tests. *Airbus Press Release*, Jun 2020.

[3] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.

[4] D Marr. Vision: A computational investigation into the human representation and processing of visual information. *MIT Press*, 1982.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[6] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[8] 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.

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

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[12] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, 2012.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[14] 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.

[15] Eder Santana and George Hotz. Learning a driving simulator. *arXiv:1608.01230*, 2016.

[16] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[17] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *Advances in Neural Information Processing Systems*, pages 809–817, 2013.

[18] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[19] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2015.

[20] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666, 2016.

[21] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv:1508.06576*, 2015.

[22] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks.

[23] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[24] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.

[25] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[26] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[27] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[29] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[30] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711, 2016.

[31] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

[32] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833, 2014.

[33] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017.

[34] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv:1811.12231*, 2018.

[35] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. *arXiv:2007.01434*, 2020.