Training Deep Networks from Zero to Hero: avoiding pitfalls and going beyond

Moacir A. Ponti, Fernando P. dos Santos, Leo S. F. Ribeiro, Gabriel B. Cavallari
ICMC – Universidade de São Paulo (USP), São Carlos, SP, Brazil
Email: ponti@usp.br, fernando_persan@alumni.usp.br, leo.sampaio.ferraz.ribeiro@usp.br, gabriel.cavallari@usp.br

***Abstract*—Training deep neural networks may be challenging in real world data. Using models as black-boxes, even with transfer learning, can result in poor generalization or inconclusive results when it comes to small datasets or specific applications. This tutorial covers the basic steps as well as more recent options to improve models, in particular, but not restricted to, supervised learning. It can be particularly useful in datasets that are not as well-prepared as those in challenges, and also under scarce annotation and/or small data. We describe basic procedures as data preparation, optimization and transfer learning, but also recent architectural choices such as use of transformer modules, alternative convolutional layers, activation functions, wide/depth, as well as training procedures including curriculum, contrastive and self-supervised learning.**

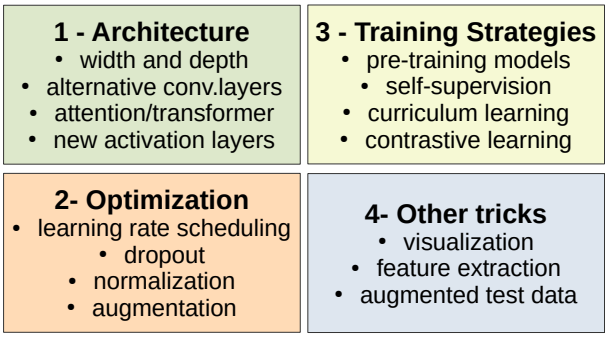


Fig. 1. Main concepts related to training of Deep Networks

I. Introduction

Different fields were revolutionized in the last decade due to the huge investment in Deep Learning research. With the curation of large datasets and its availability, as well as popularization of graphical processing units, those methods became popular in all machine learning, pattern recognition, computer vision, natural language and signal/image processing communities [1]. After becoming pervasive more broadly in related fields such as engineering, computer science and applied math [2]–[4], we observed a crescent number of projects and papers including deep learning techniques were adopted by practitioners from other fields in attempt to solve particular problems [5]–[7]. Such hype raised concerns about the pitfalls in use of machine and deep learning methods. A remarkable example is the study of Roberts et al (2021) that, in a universe of over 2,000 papers using machine learning to detect and prognosticate for COVID-19 using medical imaging, found none of the models to be of potential clinical use due to methodological flaws and/or underlying biases [8].

In fact, training deep neural networks may be challenging in real world data. Using models as black-boxes, even with transfer learning – a popular and widely used technique in this context –, can result in poor generalization or inconclusive results when it comes to small datasets or specific applications. In this paper, we focus on the main issues related to training deep networks, and describe recent methods and strategies to deal with different types of tasks and data. Basic definitions about machine learning, deep learning and deep neural networks are outside the scope of this paper. For those, please refer to the following as good starting points [1], [9], [10].

We cover the basic steps to avoid common pitfalls as well as more recent options to improve models, in particular, but not restricted to, supervised learning in visual content. Those guidelines can be particularly useful in datasets that are not as well-prepared as those in challenges, and also under scarce annotation and/or small data. Figure 1 summarizes the main options to be considered. We describe importance of basic procedures but also recent architectural choices such as use of transformer modules, alternative convolutional layers, activation functions, wide and deep networks, as well as training procedures including as curriculum, contrastive and self-supervised learning[1].

A. Notation

Let $x \in X$ be examples from a training set containing n instances, from which we may have target values or labels $y \in Y$. Such training set can be used to train a deep neural network (DNN) with multiple processing layers. For simplicity we define such neural network as a composition of functions $f_l(.)$ (related to some layer l) that has a set of parameters Θ_l, takes as input a vector x_l and outputs a vector x_{l+1}:

$$f(x) = f_L\left(\cdots f_2(f_1(x_1, \Theta_1); \Theta_2)\cdots), \Theta_L\right)$$

x_1 is the input data coming from the training set, and functions $f_l(.)$ can represent different layers such as: Dense (or Fully Connected), Convolutional, Pooling, Dropout, Recurrent, among others. Θ_l represents all learnable parameters of a given layer. For example in dense layers those are matrices W and bias values b, while in convolutional layers represent weights

[1]For an extended version containing details of the methods mentioned in thies paper see the arXiv extended version. For source-code related to this paper refer to: https://github.com/maponti/trainingdeepnetworks.

for convolutional kernels/filters. Also, let us have $z \in Z$ as examples from a test set used to evaluate the trained model.

There are also non-sequential networks having different branches containing independent or shared parameters such as siamese or triplet networks but for which at least the output layer is shared among the branches. Other models operate using more than one independent networks: a remarkable example is the Generative Adversarial Network, which contains a discriminator and a generator function.

II. How to start and common issues

Common pitfalls and issues are due to overlooked details on the design of models. In this section we present a checklist, a kind of 7 Errors Game to begin with.

A. Basic checklist (before trying anything else)

☐ *1. Input representation is fair and target patterns are present in the data*. Make sure the input data is recognizible by a human or specialist, e.g., when undersampling and trimming an audio clip the expected patterns are still audible; when resizing images the objects to be recognized are still visible. For example in Figure 2 we show two resized versions of an image to be used as input in a pre-trained neural network, however one of them clearly lost details of the cell that may be important for the task.

☐ *2. Input data is normalized accordingly*. DNNs do not work well with arbitrary ranges of numerical values. Common choices are 0-1 scaling (by computing and storing minimum and maximum values), or z-score standardization (by computing and storing mean and standard deviation). For example, in Figure 4(a-b) we compared loss and accuracy curves using normalized and non-normalized versions of the training set.

☐ *3. Data has quality (data-centric AI)*. After a decade of model frenzy, there has been a resurgence of concerns around data, that should be defined consistently, cover all important cases and be sized appropriately. Most datasets (even benchmark) have some wrong labels that hamper design of the model. In such case, recent work showed models with lower capacity (stronger bias) may be more resilient [11].

☐ *4. Both loss function and evaluation metrics makes sense*.
— loss and evaluation must be adequate to the task and to the terms involved in its computation, e.g. in a multi-class classification task make sure you are comparing probabilities (vectors with unity sum). For regression tasks, error functions (such as mean squared error) are adequate, and for object detection the intersection over union (IoU) [12] is to be considered. Note the loss function must be differentiable, i.e., have a derivative! Metrics such as accuracy, area under the curve (AUC), Jaccard and cosine distances have particular interpretations and it is paramount to understand their meaning for the task you want to learn before using it;
— check if the loss values are reasonable from the first to the last iteration, inspecting for issues such as overflow, e.g. the cross-entropy for 10 classes of a random classification result $(1/10)$ should be no more than, approximately, $-\ln(0.1) = 2.30$. Also, be sure your target (labels, range of values) matches what the network layer and its activation function outputs. For example, a sigmoid activation outputs values in the range $0-1$ for every neuron, while the softmax function outputs values so that the sum of all neurons is 1. See Figure 4(c) for the effects of using categorical vs binary cross entropy in a binary classification network in which the last layer contained only one neuron with sigmoid activation;
— Plot the loss curve for the training and validation (whenever possible) loss values along iterations (or epochs). Loss value along iterations should decrease (fairly) smoothly and converge to near zero. If not, of when the training and validation curves are too different, investigate optimization details or rethink adequacy of the chosen model for the task.

☐ *5. Projected features has reasonable structure*. It is worth visualizing the learned feature space with tSNE [13] and UMAP [14] for example, by projecting into a 2d plane the learned features, e.g. the output of the penultimate layer (often the one just before the output/prediction layer). This complements the loss curve, and may show if such space makes sense in terms of the application, or if there was no actual convergence in terms of learning an useful representation as in the case of Figure 3 in which a 10-class problem obtained a test accuracy of around 0.35, which is above random, but still far from having learning an useful representation as the same test set is projected and show no class structure.

☐ *6. Model Tuning and Validation*. The correct way to adjust a model is to use a validation set, never the test set. If you have to make any decision regarding the data preparation, neural network design, training strategies, and other, such decisions have to consider only the training data available. In this scenario you may tune the model using metrics extracted for example via a k-fold cross validation on the training set. After all choices on network topology, training strategies, hyperparameters are made, then you evaluate the final model on the test set. Otherwise, the results (even for the test set) are biased and cannot be generalized.

☐ *7. Use Internal and External Validation.* In particular for computed-aided diagnostics or deployment for decision-support, it is important to be extra careful with the data preparation, modeling and the conclusions. Methodological flaws and biases often lead to highly optimistic reported performance, but fail to be useful in practice. For example, a recent study identified 2,212 studies on COVID-19 diagnosis with chest radiographs and CT scans, from which 415 were screened, all having methodological flaws and/or underlying biases [8]. We recommend reading and checking your study using PROBAST (tool to assess risk of bias and applicability of prediction model studies) [15] and/or CLAIM (checklist for artificial intelligence in medical imaging) [16], since they may be useful not only to health data but to assess models for other applications.

After you check-listed the items above, if results are still to be improved, we now have to set ourselves to investigate: (1) how difficult the learning task is, (2) what is the nature

of the problem that makes it difficult and what options can be used to address it. Let us begin with difficult scenarios, as discussed in next sections.

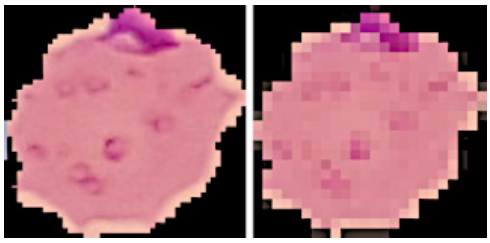


Fig. 2. Cell images resized to a size acceptable by a pre-trained network: left (128×128) still retaining structures of the cell, right (64×64) with insufficient details that would hamper learning.

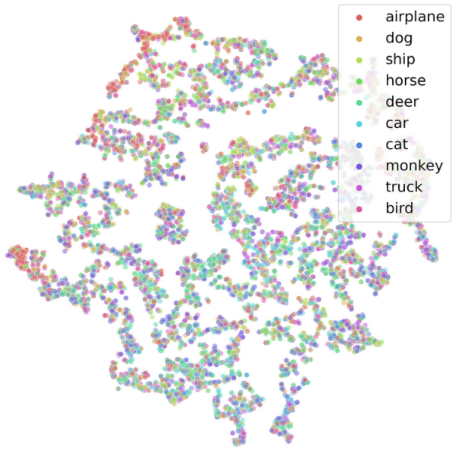


Fig. 3. t-SNE projection of the test set of STL-10 image features, extracted from the penultimate layer of a neural network that reached 35% test accuracy, but for which the learned representations shows poor class separability.

B. Small datasets and poor convergence

Learning under scarce data is known to be an issue with deep networks. For images, considering coarse-grained or category level data, i.e. the classes represent significantly different concepts such as in clothing and accessories Fashion-MNIST dataset, studies indicate a minimum of 1500 instances per class to allow learning. For fine-grained scenarios, i.e. the differences between concepts are more subtle, as in bird species CUB-200-2011 dataset (has around 60 instances per category), the problem may becomes harder if only the visual data is used in training. Therefore, if you have small data, consider transfer learning or feature extraction using DNNs (see Section V), as well as architectures with less capacity (or reduced complexity). Data augmentation is also a possibility (see Section IV-E), but if the original data is unrepresented, the augmented data will also be limited.

C. Imbalance of target data in supervised tasks

Ideally, in classification tasks, the number of examples available for each class should be similar, and for regression, training examples covering uniformly the whole range of the target data should be provided. When such supervision is not balanced with respect to the target data, one may be easily fooled by the loss function and evaluation metrics. Otherwise, one possible strategy is to weigh the classes so that the instances related to less frequent patterns will become more i the training process. Also, make sure you use metrics that evaluate how good the model along all the space of target values. In addition, data augmentation can be investigated as a way to mitigate for this imbalance (see Section IV-E).

D. Complexity of models, overfitting and underfitting

Overfitting and underfitting represent undesired scenarios of learning and are related to the complexity of the models. Although it is not the scope of this paper to explain those phenomena (for a more complete explanation refer to [17]), it is important to know how to diagnose them.

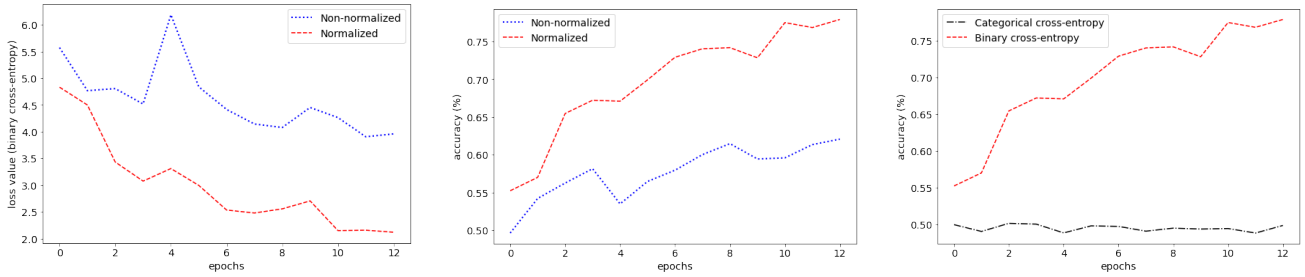
Underfitting usually occurs when the chosen architecture and training procedure are not well adapted to the task and/or the difficulty of the dataset at hand. The first symptom of this effect is a loss curve that converges to a value far from zero, or when there is no convergence at all.

Overfitting is more common for deep neural nets since those are generally high capacity models, i.e. have a large number of trainable parameters that allow for a large space of admissible functions [17]. It occurs when the network is excessively adjusted to the training set, approaching a model that memorizes the training set. Because DNNs often produce (near) zero error in the training set, it is harder to evaluate their generalization for future data.

In an attempt to measure how deep networks may memorize the training set [18] uniformly randomized the labels of examples in benchmark datasets and showed that if the network has sufficient capacity, those are able to reach near zero loss (training error) by memorizing the entire training set. More recently [11] showed lower capacity models may be practically more useful than higher ones in real-world datasets, which emphasizes the need for better data quality data and better evaluation, in particular external validation before finding a good balance between complexity of the model and its performance on a particular dataset.

E. Attacks

Deep networks learn features for a specific target task via a loss function that uses a specific training data. Because of its low interpretability, it is difficult to know which patterns from the input data were used to minimize the loss. For example, when counting white cells in blood smear images, if the purple color is present in all images with white, the optimization process has a huge incentive to use the purple color only as an indicator for white cells. Therefore, in future images, if there is purple dye in a blood smear medium (not the actual cells), the classifier may use this to incorrectly, but with a high confidence, classify the image as containing white cells. On the other hand, an image with white cells containing a different shade of purple may not be detected. The same can happen



(a) Loss values for different normalization (b) Accuracy for different normalization (c) Accuracy for different losses

Fig. 4. Comparing loss curves (a) and accuracy (b) on the training set when training the same network with the same dataset for which the instances were normalized to 0-1 (dashed red line) and not normalized (dotted blue line), and the accuracy when using different loss functions (c).

in soundscape ecology, for example when distinguishing from different bird species from its singing pattern. If there is a background noise, i.e. a critter, that usually sings at the same time of the day as some birds, the sound of the critter may be used by the network to detect the bird. In both scenarios, the features obtained after training are not the concept we wished to learn.

For example, in Figure 5 we show two test images one without attack, and the other containing a visible one-pixel attack, in which images in the training set from a given class contain a white pixel in a fixed given position, biasing the model to use that white pixel in order to predict the class, while neglecting other visual concepts. In this case we deliberately included the pixel in a visible region, but one could include that in less obvious regions such as in the border, or even add subtle features, such as gradient with similar effects [19].

Fig. 5. Example of pixel attack in which the same network is given as input two testing images (not seeing during training state), the first without attack and the second with a one-pixel attack (see the white dot on the car's door), followed by the three most probable classes output by the network. In this image the pixel was included in a visible region to facilitate visualization.

III. Architecture Options

A. Types of convolution units

Convolutions are of course the most important operation in CNNs, which means there are many studies in the literature bringing new ideas to this classic operation.

Let the *kernel size*, k, refers to the lateral size of each kernel in a convolutional layer, and we consider all those kernels to be square, so (k, k) in size. Each kernel is applied one input channel to be accumulated for one output channel, which leads to each convolutional layer having the collection of (c_{in}, c_{out}) kernels for a total of (k, k, c_{in}, c_{out}) learnable weights. The *stride* of a conv. layer refers to the step between each "application" of a kernel when it "slides" over the image, in the classic operation this step size is always one.

☐ 1×1 *convolution* can be useful for reducing computations further into networks by combining values along the channels of a single pixel. These operations do not take into account any neighbourhood, but perform the role of weighing and collecting information for each pixel on all c_{in} channels and outputting at a new channel dimensionality c_{out}.

☐ *Transposed Convolutions* play the role of a learnable, weighted upsampling operation in generative networks, autoencoders and pixel-to-pixel models (e.g. segmentation tasks). The concept simulates a *fractional stride*, so before applying the kernels a feature map is padded with zeros between spatial dimensions. When using this operation it is important to choose k as an even number to avoid the "checkerboard effect", per [20] on the effect.

☐ *Spatially Separable Convolutions* save on computation by breaking a larger convolution operation into two smaller operations. This is usually accomplished by making a convolution with $k \times k$ kernels into a $1 \times k$ followed by a $k \times 1$ operation.

☐ *Depthwise Separable Convolutions* Follow a similar principle to spatially separable ones by also breaking the traditional operation into two more efficient ones. First, the feature map is convolved with c_{in} $k \times k$ kernels, but instead of summing the resulting activations as usual, the c_{in} matrices go through a 1×1 convolution to map the output to have the desired c_{out} number of feature maps. This yields the same output shape as the traditional operation, but at a fraction of the cost.

B. Width, Depth and Resolution

Techniques for designing deep networks have evolved considerably since AlexNet [21] won the 2012 ImageNet challenge. One of the main fronts of discussion is around scaling networks up or down find a balance between accuracy and memory/computational efficiency. Width, Depth and Resolution are strategies with different pros and cons.

☐ *Wider Nets are easier to train* and are able to capture finer details in images (such as background information). Increasing width increases computational cost exponentially [22]
☐ *Deeper Nets perform better on "well-behaved" datasets* [23], such as single-object classes with "clear" objects, while wider nets did better on classes that represent scenes (e.g. "bookshop", "seashore").
☐ *Scaling Depth, Width and Resolution Together* yields the best results for a wide range of tasks and desired accuracies. Frameworks for scaling the three variables together were presented in EfficientNet [24] and MobileNet [25].

C. Pooling

Pooling layers have been a staple of CNNs since their introduction; These downsampling operations are useful both for saving on computation, memory, and for summarising feature maps as networks get deeper.
☐ *Max Pooling* is the most widely used method for classification as it enforces discriminative features within the network.
☐ *Average Pooling* was the first pooling approach and is currently used in Generative Adversarial Networks as in those models they better match the upsampling layers of generators.
☐ *Strided Convolutions*, with step size > 1 are a way of implementing "learnable pooling". Less common in classification CNNs, more common in GAN designs.
☐ *Blur Pooling* is a solution proposed by [26]. Their findings showed that current operations break the shift equivariance expected of CNNs and proposes that pooling operations are first densely evaluated, blurred by a low pass filter and only then subsampled. This improves shift equivariance.

D. Transformers and ViT

Transformers are a recent architecture created primarily for language tasks [27]. It relies on self-attention as the defining mechanism of its layers. Self-attention is very different from convolutions and from recurrent layers in that very little inductive bias is taken into account for its mechanism.

On attention layers, the relevance of one item to all other items including itself is estimated so that each item becomes a weighted average of each most relevant counterpart. This is done by learning three projection matrices W_q, W_k, W_v (similar to 3 dense layers) applied to the input items X:

$$Z = \sigma\left(\frac{(XW_q)(XW_k)^T}{\sqrt{d_q}}\right)XW_v \tag{1}$$

where σ is the softmax function that makes the resulting *attention weights* (the result of $(XW_q)(XW_k)^T$) behave as a probability function that weights the values XW_v. d_q is the dimensionality of each item.

The architecture was quickly applied to compute vision tasks as well. Notable examples being the ViT [28] for image recognition, iGPT [29] for image generation. While ongoing work with this architecture is exciting, the lack of inductive bias means that those models require much more training data than CNNs. ViT for example cannot be trained from scratch on the ImageNet dataset alone and perform well.

IV. Improving optimization

Optimization choices: algorithm, learning rate (or step size) and batch size, matters when using deep networks for learning representations. Using default options with arbitrary optimizers may lead to suboptimal results. Also, normalization, regularization and the sample size may significantly influence the optimization procedure.

A. Optimizer and batch size

The original Gradient Descent algorithm computes the gradient at one iteration using all training data. Stochastic Gradient Descent (SGD) is an approximation that allows calculating the gradient of the cost function based on a random example or a small subset of examples (minibatch). The regular SGD is a conservative but fair choice, as long as the learning rate and batch size are well defined. In fact, nearly every state-of-the-art computer vision model was trained using SGD, for example ResNet [30], SqueezeNet [31], Faster R-CNN [32], Single Shot Detectors [33].

Adaptive methods such as Adam and RAdam are good alternatives, requiring smaller learning rate (LR) values (0.001 or lower) and larger batch sizes when compared to SGD, which is less sensitive to batch size choice and LR choice is often around 0.01. Momentum can be used as an to accelerate convergence of regular SGD, however it adds another hyperparameter (the velocity weight) to be set.

B. Learning rate scheduling

A bad learning rate choice may ruin all other choices. Because the parameter adjustment is not uniform along the training process, a learning rate/step adaptation using scheduling should always be considered:
☐ *Step Decay*, decreases the learning rate by some factor along the epochs or iterations, e.g. halving the value every 10 epochs,
☐ *Exponential Decay*, reduces the learning rate exponentially.
☐ *Cosine Annealing*, continuously decreases step to a minimum value and increase it again, which acts like a simulated restart of the learning process [34].

C. Normalization

Normalizing data is a staple of classic machine learning. Since deep models are composite functions, it is beneficial to keep intermediary feature maps within some range:
☐ *Batch Norm.* (BN), a widely known technique, it was introduced by [35] to accelerate training of deep networks; it works like a layer that standardizes the feature maps across each input in a minibatch (hence the name). As learning progresses it also learns an average mean and standard deviation across the dataset that can then be used for doing single sample inference. Santurkar et. al. [36] showed that BN's advantage comes from making the optimization space smoother.
☐ *Instance Norm.* can be also designed as a layer, but instead of performing standardization across input samples, it does so for each channel of each individual sample. It's performance is worse than BN for recognition. It was designed specifically for generative and style transfer models [37].

□ *Layer Norm.* Performs standardization for each individual sample but takes mean and standard deviation from all feature maps. It was created [38] because BN cannot be applied in recurrent networks since the concept of a batch is harder to define in that context. Layer Norm. is also used on most Transformer Implementations.

D. Regularization via Dropout

Comprises mechanisms to help find the best parameters while minimizing the loss function. During the convergence process of deep networks, several combinations of parameters Θ can be found to correctly classify the training examples. Hence, **Dropout** [39] works by deactivating $p\%$ of neurons, mainly after dense layers. This avoids some neurons to over-specialize/memorize specific data. At each iteration of training, dropout provides different subsamples of activations, i.e different stages of the network. Consequently, this mechanism prevents overfitting during training. During inference dropout is turned off so that all neurons are activated.

E. Data Augmentation

Unlike the other optimization techniques mentioned before, which work to improve performances by acting on the network structure, data augmentation techniques focus exclusively on increasing the size and variability of the training set [40]. Conceptually, it generates new instances derived from the original training set by manipulating the features and replicating the original label to the generated example. Thus the training set becomes more variable and larger. Data augmentation can also be used to balance datasets (see Subsection II-C), controlling one of the drawbacks of deep learning [41]. A recurrent concern in these techniques is to ensure that the transformation performed does not alter its concept.

V. Training procedures beyond the basics

The regular approach for training deep networks is to design its topology, define its training strategies, randomly initializing all parameters and then *train from scratch*. However such networks are both data-hungry and highly sensitive to initialization. To overcome those issues, weight warmup procedures were studied, such as first training an unsupervised autoencoder [42] and then use its encoder weights as initialization. In addition, a widespread approach is to download models pre-trained using a large datasets such as ImageNet in the case of image classification [43]. This is called transfer learning, and assumes the model has generalization capability. Due to the hierarchical structure of deep networks, in which different layers provide different levels of attributes, even different image domains may benefit from pre-training [44], [45].

– **Transfer learning from pre-trained weights**:

1) remove the original output layer, design a new output layer and randomly initialize its weights;
2) freeze the remaining layers, i.e. making the layers not trainable, by not allowing their parameters to be updated;
3) train the last layer for a number of epochs.

– **Fine-tuning** after transfer learning, unfreeze and train a subset of layers using a small learning rate (often 10^{-4} or even less). As a rule of thumb, one starts by unfreezing the layers closer to the top (output) of the network and, the more data one has, more layers can made trainable. Use with care: if your dataset is small, beware not to overtrain.

– **Pre-trained nets can be used as feature extractor** in scenarios with small sets of data, in which even transfer learning would be unfeasible. For this, perform a forward pass and get the activation maps of a given layer as a feature vector for the input data. Getting the output from the penultimate layer is a fair choice since this represents input data globally [44]. However, one can also insert a global pooling layer just after a convolutional layer to summarize the data. Previous studies show that combinations coming from different layers improve the representation [46], [47].

Alternatively to the use of a global pooling layer, get all activation maps/values (often high dimensional) and carry out a separate dimensionality reduction, for instance using Principal Component Analysis (PCA). With the extracted features one can proceed with external methods such as classification, clustering, and even anomaly detection [45].

In the next sections we will cover training strategies beyond the transfer learning approach.

A. Curriculum Learning

This concept is based on the human strategy of creating a study script, in which a teacher elaborates a student's learning order, facilitating training [48], [49]. With the premise that part of the data (or the task) at hand is easier than others to be learned, instead of trying to train all model at the same time with randomly sampled data, it is possible to define an order of instances or tasks. The basic technique works with instances by defining: a scoring function and a pacing function. The scoring function is a metric to sort the training examples from the easiest to the most difficult, e.g. a shallow classifier confidence. The pacing function, e.g. linear, exponential, dictates the learning speed to incorporate more (difficult) examples into the training set. Note that unbalanced scenarios can be harder when applying curriculum learning. Also, learning rate have to be properly investigated so that not to degrade performance [49]. Curriculum learning can also be applied as a sequence of tasks, where the easiest task is performed before the most difficult ones [50].

B. Contrastive/Distance Learning

Deep learning is often used to learn representations using tasks such as classification, e.g. via CNNs, and reconstruction, e.g. via Autoencoders. An alternative way is using *contrastive learning*, which consists of using losses designed around the task of learning (dis)similarities between instances.

Commonly used losses are: contrastive, which works by optimizing distances between representations of pairs of instances, and its variant triplet loss which works using triplets of instances. While initial applications included face recognition [51] and content-based image retrieval [50], more recently self-supervised learning made use of this strategy.

C. Self-supervised Learning

Given a task and sufficient labels, supervised learning can solve it. But large amounts of manually labeled data are often costly and time-consuming to obtain. Sometimes real-world applications require concepts that are outside the scope of standard datasets. And in some cases, vast amount of unlabelled images is readily available.

Self-supervision is a form of unsupervised learning where the data itself provides the supervision. It relies on pretext tasks that can be formulated using only input data. For example one can produce surrogate (or pseudo) labels for classification or design systems that learn to compare, using a contrastive learning strategy. Those are than used to pre-train a model instead of relying only on large-scale benchmark datasets. Methods include predictions of data rotation, relative positions, maximization of mutual information, cluster-based discrimination and instance discrimination. Relevant works are SimCLR [52], SwAV [53] and Barlow Twins [54]. For a deeper understanding of those models and an extensive view of another methods, we recommend [55] and [56].

VI. Running the final mile to improve predictions

After previous strategies were explored, some other tricks may produce small but valuable improvement.

A. Activation Functions beyond ReLU

The Rectified Linear Unit (ReLU) became the standard activation function for hidden layers of deep networks because it avoids saturation which can cause training to slow down due to near-zero gradients. The problem is that values of ReLU near zero produce non-useful or bad estimates for the gradient. Numerically, it may lead to neurons that stuck completely in the negative side and always outputs zero for the training set. Swish and Mish functions were proposed to improve this.

Swish is a gated version of Sigmoid and defined as $s(x) = x \cdot \sigma(\beta x)$, where $\sigma(.)$ is the Sigmoid function and β is a hyper-parameter that can be adjusted arbitrarily or trained. Mish is defined as $m(x) = x \cdot \tanh(\ln(1 + e^x))$, which is bounded below and unbounded above and the range is approximately $[-0.31, \infty]$. Small but consistent improvement were observed when using Swish and Mish instead of ReLU in hidden layers of the network, with a slight advantage for Mish.

B. Validation and Test-Augmented Data

To assess the robustness of the trained model, one can compare its performance in the original validation set with a perturbed version containing only modified versions of the instances, i.e. by translation, noise injection, etc, one can decide to include those perturbed data in the training set.

To improve final performance after the model is trained, given a test example x, obtain augmented versions: $x^{(1)}, x^{(2)}, \ldots x^{(m)}$, and combine the network predictions $\hat{y}^{(1)}, \ldots \hat{y}^{(m)}$ (via average, majority voting or other [57]).

VII. Conclusion

This paper offers as a reference to allow researchers and practitioners to avoid major issues and to improve their models with less usual techniques. While DNNs have high generalization capacity and allow significant transfer learning, there are important concepts that require attention to allow learning. The basic recommendations for machine learning should be observed, and given the representation learning nature of deep networks, employ other practices that mainly try to improve the representations, not only the main objective function. Going beyond the basic techniques, leveraging unlabeled data and carefully designing optimizations steps beyond the basics may be the way towards more reliable models.

VIII. Acknowledgments

The authors are grateful to FAPESP grants #17/22366-8, #19/07316-0 and #19/02033-0 and CNPq 304266/2020-5.

References

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[2] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 50–59.

[3] D. M. Dimiduk, E. A. Holm, and S. R. Niezgoda, "Perspectives on the impact of machine learning, deep learning, and artificial intelligence on materials, processes, and structures engineering," *Integrating Materials and Manufacturing Innovation*, vol. 7, no. 3, pp. 157–172, 2018.

[4] C. F. Higham and D. J. Higham, "Deep learning: An introduction for applied mathematicians," *Siam review*, vol. 61, no. 4, pp. 860–891, 2019.

[5] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature medicine*, vol. 25, no. 1, pp. 24–29, 2019.

[6] S. Christin, É. Hervet, and N. Lecomte, "Applications for deep learning in ecology," *Methods in Ecology and Evolution*, vol. 10, no. 10, pp. 1632–1644, 2019.

[7] I. Chalkidis and D. Kampas, "Deep learning in law: early adaptation and legal word embeddings trained on large corpora," *Artificial Intelligence and Law*, vol. 27, no. 2, pp. 171–198, 2019.

[8] M. Roberts, D. Driggs, M. Thorpe, J. Gilbey, M. Yeung, S. Ursprung, A. I. Aviles-Rivero, C. Etmann, C. McCague, L. Beer *et al.*, "Common pitfalls and recommendations for using machine learning to detect and prognosticate for covid-19 using chest radiographs and ct scans," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 199–217, 2021.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[10] M. Ponti, L. S. Ribeiro, T. S. Nazare, T. Bui, and J. Collomosse, "Everything you wanted to know about deep learning for computer vision but were afraid to ask," in *SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T 2017)*, 2017, pp. 1–25.

[11] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," *arXiv preprint arXiv:2103.14749*, 2021.

[12] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 658–666.

[13] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[14] L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," *Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[15] R. F. Wolff, K. G. Moons, R. D. Riley, P. F. Whiting, M. Westwood, G. S. Collins, J. B. Reitsma, J. Kleijnen, and S. Mallett, "Probast: a tool to assess the risk of bias and applicability of prediction model studies," *Annals of internal medicine*, vol. 170, no. 1, pp. 51–58, 2019.

[16] J. Mongan, L. Moy, and C. E. Kahn Jr, "Checklist for artificial intelligence in medical imaging (claim): a guide for authors and reviewers," 2020.

[17] R. F. Mello and M. A. Ponti, *Machine learning: a practical approach on the statistical learning theory*. Springer, 2018.

[18] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *International Conference on Learning Representations (ICLR)*, 2016.

[19] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 427–436.

[20] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[22] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, R. C. Wilson, E. R. Hancock, and W. A. P. Smith, Eds. BMVA Press, 2016.

[23] T. Nguyen, M. Raghu, and S. Kornblith, "Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth," in *International Conference on Learning Representations*, 2021.

[24] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 6105–6114.

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[26] R. Zhang, "Making convolutional networks shift-invariant again," in *ICML*, 2019.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008.

[28] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[29] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 1691–1703.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[31] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[32] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 91–99.

[33] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," in *European Conference on Computer Vision*, 2016, pp. 21–37.

[34] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[36] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 2488–2498.

[37] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *CoRR*, vol. abs/1607.08022, 2016.

[38] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[39] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[40] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[41] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *Journal of Big Data*, vol. 5, no. 1, pp. 1–30, 2018.

[42] G. B. Cavallari, L. S. Ribeiro, and M. A. Ponti, "Unsupervised representation learning using convolutional and stacked auto-encoders: a domain and cross-domain feature space analysis," in *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 2018, pp. 440–446.

[43] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2661–2671.

[44] M. A. Ponti, G. B. P. da Costa, F. P. Santos, and K. U. Silveira, "Supervised and unsupervised relevance sampling in handcrafted and deep learning features obtained from image collections," *Applied Soft Computing*, vol. 80, pp. 414–424, 2019.

[45] F. P. dos Santos, L. S. Ribeiro, and M. A. Ponti, "Generalization of feature embeddings transferred from different video anomaly detection domains," *Journal of Visual Communication and Image Representation*, vol. 60, pp. 407–416, 2019.

[46] F. P. dos Santos and M. A. Ponti, "Alignment of local and global features from multiple layers of convolutional neural network for image classification," in *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 2019, pp. 241–248.

[47] Y. Zheng, J. Huang, T. Chen, Y. Ou, and W. Zhou, "Cnn classification based on global and local features," in *Real-Time Image Processing and Deep Learning 2019*, vol. 10996. International Society for Optics and Photonics, 2019, p. 109960G.

[48] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[49] G. Hacohen and D. Weinshall, "On the power of curriculum learning in training deep networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2535–2544.

[50] T. Bui, L. S. F. Ribeiro, M. Ponti, and J. P. Collomosse, "Sketching out the details: Sketch-based image retrieval using convolutional neural networks with multi-stage regression," *Comput. Graph.*, vol. 71, pp. 77–87, 2018.

[51] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[52] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[53] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," 2020.

[54] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," *arXiv preprint arXiv:2103.03230*, 2021.

[55] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[56] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[57] M. Ponti, "Combining classifiers: from the creation of ensembles to the decision fusion," in *2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials*. IEEE, 2011, pp. 1–10.