Smarter Prototyping for Neural Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract
Deep Neural Networks have become commercially viable in fields like machine vision, speech/language processing, data acquisition among other applications. Convolutional Neural Networks (CNNs), Recurrent Neural Network (RNNs) and their variants in some conditions have achieved performance better than human experts. However, existing deep network models are incompatible with low power devices or mission-critical applications due to either high computational & latency cost or memory storage, which makes them unfit to scale. Moreover, less effort has been put in making the architectural improvements modular or model-agnostic. In the developing regions of the world, efficient and frugal learning frameworks will have a huge socio-economic impact. AI can be a game-changer by enabling unique strategies to facilitate social good through domain-experts if ML-prototyping is intuitive. Thus, this paper serves a dual purpose, first is to present easily implementable structural modifications, and second is to provide a comparative overview of prevalent compression techniques. Finally, we conclude this paper discussing and proposing possible challenges in these areas.

1 Introduction
The deep learning networks which are being deployed commercially are GPU-hungry. State-of-the-art models in-lieu of delivering superb performance often contain billions of parameters. Such requirements are a critical hindrance in low-resource applications. While model quality has been shown to scale with model and data-set size [Hestness et al., 2017], the heavy resources required to train them can be prohibitive, especially in regions with low research budgets. More importantly, the current motivation for network design stems from beating the 'state-of-the-art'. These metrics are un-intuitive, and don’t provide actionable feedback towards improvement [Anderson-Cook et al., 2019]. As Goodhart’s Law states- “When a measure becomes a target, it ceases to be a good measure.”, they instead lead to models that are incomprehensible for further development [Lipton and Steinhardt, 2019]. Various methodologies have been used to reduce the architectural complexity of such models [Simard, Steinkraus, and Platt, 2003]. Residual Networks (ResNets) [He et al., 2016] and SqueezeNet [Iandola et al., 2017] achieve better classification results despite very small parameter count. Although, as shown in [Dubey, Chatterjee, and Ahuja, 2018], these compact-nets also have remnant redundancies.

In this work we shed light on some methodologies for giving a jump-start to development pipelines aimed at leveraging the power of deep networks. This compilation should provide an intuitive platform for designing an existing model while also providing directions to conduct ablative studies.

2 Deep Learning for Good
Most non-profit organizations do not have a specialized workforce of data scientists, and engineers. With all the modern technology available to humankind, humble farmers across the world are still
at the mercy of the environment for their livelihood. Remote Health Diagnostics in parts of the
developing world still hasn’t hit a critical point. While no silver bullet, machine learning can be an
invaluable tool in a wide array of applications. It is quite evident that these problems already have
baseline solutions using AI, although from a realistic perspective these are still out of reach from the
populace. Neural Networks which will not be a luxury (Ferreira et al., 2019) to deploy can empower
local officials and regional scientists. AI can be truly transformative if it receives contributions from
all over the globe. Its significance as a tool depends on technology-penetration and adoption scale.

Domain-experts can use AI tools to harness innovations in Agriculture (Rußwurm et al., 2019),
Climate Science (Kim et al., 2019), Healthcare (Koushik, Amores, and Maes, 2018) (Rawat, Li, and
Yu, 2019) etc. However, these solutions will not come to fruition unless machine learning models are
proficient both in terms of cost-effectiveness and ease of development. We address this two-pronged
requirement by listing ideas to facilitate prototyping for real-world applications and also giving a
gentle introduction to compact/compressed networks.

3 Prototyping-based improvements for Applied ML

In the ML community, it is well-known that data augmentation can be very beneficial for model
performance (Goodfellow, Bengio, and Courville, 2016). Similarly, adding Gaussian Noise during
training enhances validation and reduces overfitting (An, 1996). Also, cyclic learning rates have
been shown to enhance training of the neural-nets to achieve faster convergence (Smith, 2015).
Such tricks-of-the-trade are immensely useful during early stages of model design, especially for AI
practitioners from eclectic backgrounds.

While there is a huge demand for AI-based solutions in other fields such as medical, disaster relief
and management, such cross-disciplinary efforts will only succeed when deep learning models are
frugal and easily customizable. The current trend of shipping black-box or non-interpretable neural
networks is a serious bottleneck for custom-model development since they’re counter-intuitive and
unreliable. Thus, ease of prototyping is a crucial step forward.

Due to a meteoric rise in the general area of machine learning research, it is difficult to keep track of
all the remarkable contributions while working on well-established theories. Hence, here we will
focus on other such emerging techniques. In Table 1 (3.6), we summarize various techniques (Swish,
OctConv, SwapOut, ZoneOut, Population-based Augmentation), and give an overview in Figure 1.

![Figure 1: Modular Prototyping Techniques](image)

3.1 Swish

Swish (Ramachandran, Zoph, and Le, 2018) is an activation function (as shown in Fig.2):

\[ f(x) = x \sigma(\beta x) \]

where \( \sigma(z) = \frac{1}{1+\exp(-z)} \). Also, \( \beta \) can be defined as a constant (generally, 1) or a trainable parameter
(especially helpful when encountering many dead ReLUs). Swish is non-monotonic but, like ReLU,
bounded from below and unbounded from above. Its advantages are observable in deeper networks
since it better handles vanishing gradients. It has a tendency to speed-up learning (despite using
higher dropout), but may suffer from over-fitting in simpler tasks. It is highly recommended for dealing with complex data sets.

Figure 2: Swish instead of ReLU in a DNN

Can we replace ReLU with Swish everywhere? This is a tricky question to answer since ReLU has been extensively studied and is currently the most deployed activation function. The original paper (Ramachandran, Zoph, and Le, 2018) includes a comparative study of Swish’s performance on standard benchmarks such as CIFAR-10 and CIFAR-100. It revealed that swish performs better than most of the other variants of ReLU and SELU (Klambauer et al., 2017) at the cost of an increased training time. But there are no comparative results on adversarial training and generative models. A recent work on flow-based generative models (Chen et al., 2019a) coined a new swish function called Lipswish to address the vanishing gradient function in generative models. Although it produces promising results, the present literature does not offer enough case studies to conclude that swish would outperform other linear activation functions in every other scenario.

3.2 Octave Convolution (OctConv)

There are a lot of spatial redundancies in CNN frameworks, thus OctConv (Chen et al., 2019b) enhances efficiency by leveraging low/high-frequency features independently. This characteristic also boosts classification performance due to better global context knowledge from a widened receptive field. It is a plug-and-play, orthogonal unit to substitute regular convolutions (2D and 3D) without any modifications to the network structure. To the best of our knowledge, OctConv has been used to stabilize GAN training (Durall, Pfreundt, and Keuper, 2019) and also reacts well with model compression (Zhou et al., 2019). It can be combined with techniques to decrease channel-wise redundancy (ex. depth-wise convolutions) and even for topological-improvements.

3.3 Swapout

Swapout (Singh, Hoiem, and Forsyth, 2016) is a stochastic training method that shows stable improvements using efficient parameter utilization. It can be seen as a clever merge of the two regularization techniques that is, dropout (Srivastava et al., 2014) and stochastic-depth (Huang et al., 2016), outperforming both in stand-alone comparisons. Also, linear decay of parameters (less dropping on early layers, more on later ones) significantly improves its results. Relatively shallow Swapout networks give similar performance to extremely deep ResNets.

JumpOut (Wang, Zhou, and Bilmes, 2019) is an orthogonal method which improves regularization and generalization by actively normalizing dropout rate based on active neurons. The compute overhead is relatively negligible, and it can be used with other versions of dropout.

3.4 Zoneout

In the hidden units of RNNs, instead of drop-out, one can use zoneout (Krueger et al., 2016). In this method, unit activations are stochastically replaced with the preceeding time-step activations instead of removing them. This modification achieves higher performance than alternative regularizers, especially in speech processing applications. It seems to be inherently robust to changes in hidden state while streamlining information-flow through the network.

3.5 Population-based Augmentation (PBA)

Data augmentation has proved to be a valuable resource during training of deep learning models (Perez and Wang, 2017). Although, still there are no well-defined rules to select augmentation policies for a DNN. Population-based Augmentation or PBA (Ho et al., 2019) searches from a set
of several operations and seeks out an effective combination. A carefully chosen policy enhances generalization and robustness but is a time-consuming process. AutoAugment (Cubuk et al., 2019) is the state-of-the-art method but is extremely compute-intensive. PBA is different to AutoAugment in a sense that it (dynamically) learns a schedule of policies, each dedicated for a specific training-epoch.

3.6 Adversarial Training for Free

Adversarial attacks can cause malfunctioning in other intelligent models through malicious inputs (Papernot et al., 2016), similar to optical illusions in human-vision. A natural defence is Adversarial training of a deep network which is very compute-intensive (Xie et al., 2018). This "free" technique (Shafahi et al., 2019) proposes an effective solution where instead of using separate gradient computations for each update during one simultaneous backward pass, simultaneous updates are applied on the model parameters and image-perturbations. This method also differs from previous approaches since it is orthogonal to other defenses, faster and applicable on big networks including high-resolution data sets ex. Imagenet (Krizhevsky, Sutskever, and Hinton, 2012).

Table 1: Summary of prototyping-based improvements for DNNs (*GRUs= Gated Recurrent Units)

<table>
<thead>
<tr>
<th>Technique</th>
<th>Group</th>
<th>Brief</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swish</td>
<td>Activation Function</td>
<td>Non-monotonic, small bump for -ve input</td>
<td>Trains Deeper-Nets, faster convergence. Works with GANs</td>
<td>Over-fitting, slower epochs and inference (~15%)</td>
</tr>
<tr>
<td>OctConv</td>
<td>Convolution Operation</td>
<td>It bifurcates feature map tensors into low &amp; high frequencies</td>
<td>Saves computation and memory. Boosts accuracy. Stabilizes GAN training</td>
<td>More hypertuning. Compatibility with other compression methods untested</td>
</tr>
<tr>
<td>SwapOut</td>
<td>Regularizer</td>
<td>Dropout + random skipping connections to generalize, Bonds parameters across layers</td>
<td>Prevents co-adaptation in neurons(units) &amp; across network layers</td>
<td>Testing for best training schedules,</td>
</tr>
<tr>
<td>ZoneOut</td>
<td>Regularizer (RNNs)</td>
<td>Preserves hidden units (unlike dropout), regularize transition dynamics</td>
<td>Robust to Hidden-state changes, boosts gradient propagation</td>
<td>Not suitable for ResNets, More tests required for variants ex. GRU*</td>
</tr>
<tr>
<td>PBA</td>
<td>Augmentation</td>
<td>Hyperparameter search using evolutionary algos &amp; random search to seek adaptive augmentation policy</td>
<td>Low compute-cost and memory. Effective optimization.</td>
<td>Not the most optimal result, Lacks extensive testing</td>
</tr>
<tr>
<td>Adversarial Training for Free</td>
<td>Model Robustness</td>
<td>Reuse the gradient info for concurrent updates to parameter &amp; perturbation</td>
<td>Low compute and memory required. Multiple adversarial updates possible</td>
<td>Untested with other adversarial defense methods</td>
</tr>
</tbody>
</table>

4 Compact Networks

In a cloud-based environment with abundant computational capabilities, enabled by multiple graphical processing units (GPUs), such massive memory requirements may not be considered a restriction (Chollet, 2016). However, in case of mobile devices (ex. Robots, Internet of Things) with limited computational capabilities, such resource intensive deep neural networks cannot be readily applied. The exploitation of deep learning on sensory-devices, including smartphones (Lane, Georgiev, and Qendro 2015), (Haffari et al., 2018), has pointed out this as a major hurdle to wide spread use.

Nowadays, sparsity is also used to refer to the proportion of a neural networks weights that are zero valued. Moreover, it has been shown empirically that DNNs can tolerate high levels of sparsity (Narang et al., 2017), and this property has been leveraged to significantly reduce the cost associated with implementing deep-nets, and to enable the deployment of state-of-the-art models in severely

4
resource constrained environments (van den Oord et al., 2016). Higher sparsity corresponds to fewer weights, and smaller computational and storage requirements (Gale, Elsen, and Hooker 2019). Thus, the design of deep neural networks that require less storage and computation power has established itself as a new research direction. Particularly, the modification of large cumbersome models that reduces the memory requirements while retaining as much of its performance as possible is referred to as compression of neural networks (Han et al. 2015a). Another direction is the design of more memory efficient network architectures from scratch (Hinton, Vinyals, and Dean 2015).

In the following segment, we trace the foundational ideas of the prevalent methods for model-compression, briefly comment on their key characteristics and tabulate their Pros & Cons. In Table 2 (3.6), we summarize four types of compression methods (Pruning, Quantization, Knowledge Distillation and Tensor Vectorization/Low-Rank Factorization).

### 4.1 Pruning

Despite the utilization of powerful regularization techniques like dropout or weight decay, some weights of a network will always contribute more to the prediction than others (Han et al. 2015b). The process of removing the less contributing weights to compress (and/or further regularize) the network is called pruning (Han, Mao, and Dally 2016). After some weights have been pruned, the network typically has to be fine tuned again to have it adapt to the change (Wen et al. 2016).

### 4.2 Quantization

Quantization is decreasing the (dispensable) numerical precision of a model. Its a popular approach for data compression and several methods have emerged for neural network acceleration as well. The most popular methods are: scalar/vector quantization, and fixed-point quantization. However, experimentally (Google 2019) it is revealed that methods like k-means quantization do not improve speed or storage requirements in practice. Since the weight matrix has to be reconstructed to 32-bit floats from 8 or 16 bit floats during inference, the process is often referred as pseudo quantization in the community. It is different from real quantization where each weight is permanently encoded using fewer bits.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Pruning</th>
<th>Quantization</th>
<th>Distillation</th>
<th>Vectorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Sharing or removing redundant parameters</td>
<td>Reducing the model’s numerical precision</td>
<td>Learning smaller models from big ones (mimic)</td>
<td>Approximate a weight matrix by sum minimization</td>
</tr>
<tr>
<td>Application*</td>
<td>Conv/FC-layer. (on synapses and/or neurons)</td>
<td>Post-Training (or quantization-aware training)</td>
<td>Conv &amp; FC-layer (across network)</td>
<td>Conv/FC-layer (Low Rank Factorization)</td>
</tr>
<tr>
<td>Pros</td>
<td>Reduces size, complexity and over-fitting</td>
<td>Faster Inference (Even better when Activations are quantized too)</td>
<td>Significant reduction in computational-cost and size</td>
<td>High compression and speed-up</td>
</tr>
<tr>
<td>Cons</td>
<td>Longer Training, more hyper-tuning</td>
<td>Hardware dependent benefits</td>
<td>Overfitting, works only for (Softmax) classifications</td>
<td>Rigorous Retraining, Small ranks may hurt model</td>
</tr>
</tbody>
</table>

### 4.3 Knowledge Distillation

The thought in knowledge distillation is to "mimic" knowledge of a (large) teacher model into a smaller and efficient (student) model by learning the softmax-based class distributions. The cross entropy difference in the predictions from both the models is calculated while doing a forward pass during training, and is added to the student-model’s loss value. In this method, the student also learns “dark knowledge” (Hinton, Vinyals, and Dean 2014) of the closely associated categories from the teacher-model in addition to learning from ground-truth labels.
Although knowledge-distillation based techniques are very promising, and achieving impressive results in reducing compute-cost and complexity, they are currently severely restrictive in a sense that only models having softmax-based classification tasks can be distilled.

4.4 Vectorization/Tensor Decomposition (Low-Rank Factorization)

The core thought behind Matrix factorization (MF) is that by exploiting latent structures inherent to the data set we can obtain compressed feature representations ([Denton et al., 2014]). Tensor Decompositions are higher order matrix decomposition, leveraging redundancy in 4D tensors (Conv Kernels). Thus, any reductions in the convolutional layers (Conv operations are heavy) would improve inference speed. This method is controversial for requiring robust re-training since the model’s learning is altered, but many consider the gains worthwhile. In its various forms, it is a popular tool for dimensionality-reduction, unsupervised clustering, and can be used in combination to other compression methods.

5 Conclusion

The first-half of the paper is oriented towards discussing various techniques in the recent literature that would expedite early-stage prototyping and favour AI adoption in diverse domains and applications. It also gives pragmatic insights about improving network performance (training time, accuracy etc.) and thus could be of interest to the ML community as well. These techniques would bring maximum benefit when built with efficiency at their core. So, in the second-half we gave an overview of the main concepts and approaches to model compression. This paper presents the precursory tool-kit necessary for building and training an efficient network from scratch. In the following section we discuss on the prevalent issues related to implementing these various tools and how that may impact the future direction of research.

6 Discussion and Future Directions

We discussed the feasibility of compressing models for disk size and memory usage at negligible accuracy losses. Compressing for speed, on the other hand, is tricky in practice. It can depend on faster integer multiplication or sparse matrix multiplication, here machine learning research hits the reality of computing (hardware) architecture. As evident in recent literature, there’s a growing interest in Compact Networks as there is an urgent of compatible for small scale embedded systems.

While we tried to provide a brief and yet comprehensive overview on the recent advancements made in network architecture design to the readers, we also discovered huge gaps in the existing literature on detailed comparative studies on various prototyping techniques in this rapidly evolving field. For example, we have presented a detailed discussion on the merits of using swish activation function compared to other linear activation functions for both generative and classification models. But while we explored the literature for similar analysis on other techniques such as swapout or zoneout, we could not find enough empirical analysis to comment about the generalizability of these approach across domain and model specifications. Exploring detailed analysis on these techniques would be beneficial to the deep learning community and might lead to exciting new findings. We believe such studies will also reveal the limitation of these various prototyping techniques and help us establish more elaborate standards when it comes to building deep networks from the scratch.

Another interesting area we did not explore in this paper is Interpretable and Explainable Network Architectures ([Chen et al., 2016], [Zhou et al., 2017]). This field has a lot of potential and will be essential for building trust in AI Systems as well as setting-up benchmarks for future research. There is also a dire need of developing mathematical framework that would help us analyse and understand behaviour of neural networks under the influence of various activation or regularization functions. The existing literature relies heavily on empirical case studies for performance analysis and are often designed specifically for a particular problem. Often these kind of practices do not transfer well across different domains and lead to a lot of trial-and-error runs before one can narrow down the specifications that works well for their problem.
References


