# TResNet: High Performance GPU-Dedicated Architecture

Tal Ridnik Hussam Lawen Asaf Noy Emanuel Ben Baruch Gilad Sharir Itamar Friedman
DAMO Academy, Alibaba Group
{tal.ridnik, hussam.lawen, asaf.noy, emanuel.benbaruch, gilad.sharir, itamar.friedman}
@alibaba-inc.com

## Abstract

*Many deep learning models, developed in recent years, reach higher ImageNet accuracy than ResNet50, with fewer or comparable FLOPs count. While FLOPs are often seen as a proxy for network efficiency, when measuring actual GPU training and inference throughput, vanilla ResNet50 is usually significantly faster than its recent competitors, offering better throughput-accuracy trade-off. In this work, we introduce a series of architecture modifications that aim to boost neural networks' accuracy, while retaining their GPU training and inference efficiency. We first demonstrate and discuss the bottlenecks induced by FLOPs oriented optimizations. We then suggest alternative designs that better utilize GPU structure and assets. Finally, we introduce a new family of GPU-dedicated models, called TResNet, which achieves better accuracy and efficiency than previous ConvNets* [1]. *Using a TResNet model, with similar GPU throughput to ResNet50, we reach* 80.8% *top-1 accuracy on ImageNet. Our TResNet models also transfer well and achieve state-of-the-art accuracy on competitive single-label classification datasets such as Stanford Cars (96.0%), CIFAR-10 (99.0%), CIFAR-100 (91.5%) and Oxford-Flowers (99.1%). TResNet models also achieve state-of-the-art results on a multi-label classification task, and perform well on object detection.*

## 1. Introduction

The seminal ResNet models [10], introduced in 2015, revolutionized the world of deep learning. ResNet models use repeated well-designed residual blocks, allowing training of very deep networks to high accuracy while maintaining high GPU utilization. ResNet models are also easy to train, and converge fast and consistent even with plain SGD optimizer [45]. NVIDIA Volta tensor cores [27] further improved ResNet models GPU utilization, up to quadrupling their GPU throughput on mixed-precision training and inference [44]. Among the ResNet models, ResNet50 established itself as a prominent model in terms of speed-accuracy trade-off, and became a leading backbone model for many computer vision tasks [8, 21, 42, 14].

Since ResNet50, many modern deep learning models were developed, which achieve better ImageNet accuracy with fewer or comparable FLOPs. Surprisingly, even though most deep learning models are trained, and sometimes deployed, on GPUs, few models try explicitly to find an optimal design in terms of GPU throughput. Since FLOPs are not an accurate proxy for GPU speed [2], sub-optimal design for GPUs might occur. This is especially true for GPU training speed, which is rarely measured and documented in academic literature, and can be severely hindered by some modern architecture design tricks [26].

Table 1 compares ResNet50 to popular newer architectures, with similar ImageNet top-1 accuracy - ResNet50-D [11], ResNeXt50 [43], SEResNeXt50 [13], EfficientNet-B1 [36] and MixNet-L [37]. We see from Table 1 that the reduction of FLOPs and the usage of new tricks in modern networks, compared to ResNet50, is not translated to improvement in GPU throughput. This is especially evident for GPU training speed, where ResNet50 gives by a large margin better speed-accuracy trade-off. We identify two main reasons for this throughput gap:

1. Modern networks like EfficientNet, ResNeXt and MixNet do extensive usage of depthwise and 1x1 convolutions, that provide significantly fewer FLOPs than 3x3 convolutions. However, GPUs are usually limited by memory access cost and not by number of computations, especially for low-FLOPs layers. Hence, the reduction in FLOPs is not translated well to an equivalent increase in GPU throughput [26].
2. Modern networks like ResNeXt and MixNet do extensive usage of multi-path. For training, this creates lots of activation maps that need to be stored for backward propagation, which reduces the maximal possible batch size, thus hurting the GPU throughput. Multi-path also limits the ability to use inplace operations [32], and can lead to network fragmentation [26].

[1] Source code is available at: https://github.com/mrT23/TResNet

| Model | Training Speed [img/sec] | Inference Speed [img/sec] | Top1 Accuracy [%] | Flops [G] |
|---|---|---|---|---|
| ResNet50 [10] | **805** | 2830 | 79.0 | 4.1 |
| ResNet50-D [11] | 600 | 2670 | 79.3 | 4.4 |
| ResNeXt50 [43] | 490 | 1940 | 79.4 | 4.3 |
| EfficientNetB1 [36] | 480 | 2740 | 79.2 | 0.6 |
| SEResNeXt50 [36] | 400 | 1770 | 79.9 | 4.3 |
| MixNet-L [37] | 400 | 1400 | 79.0 | 0.5 |
| TResNet-M | 730 | **2930** | **80.8** | 5.5 |

Table 1. **Comparison of ResNet50 to top modern networks, with similar top-1 ImageNet accuracy**. All measurements were done on Nvidia V100 GPU with mixed precision. For gaining optimal speeds, training and inference were measured on 90% of maximal possible batch size. Except TResNet-M, all the models' ImageNet scores were taken from the public repository [41], which specialized in providing top implementations for modern networks. Except EfficientNet-B1, which has input resolution of 240, all other models have input resolution of 224.

Following our analysis of Table 1, we want to design a new family of networks, TResNet, aimed at high accuracy while maintaining high GPU utilization. TResNet models will contain the latest published design tricks available, along with our own novelties and optimizations. Unlike previous works, which measure only the FLOPS proxy or just GPU inference speed, we will directly focus on both GPU inference and training speed. For a proper comparison to previous models, one network variant (TResNet-M) is designed to match ResNet50 GPU throughput, while the rest match modern larger architectures.

We will show that for all tested datasets, TResNet models offer an improved speed-accuracy trade-off. Specifically, they reach ImageNet top-1 accuracy of $80.8\%$ with GPU throughput similar to ResNet50 ($79.0\%$), and top-1 accuracy of $84.3\%$ with better GPU throughput than EfficientNet-B5 ($83.7\%$). Besides ImageNet, TResNet models also achieve state-of-the-art accuracy on 3 out of 4 widely used downstream single-label datasets, with x8-15 faster GPU inference speed. They also excel on multi-label classification and object detection tasks.

## 2. TResNet Design

TResNet design is based on the ResNet50 architecture, with dedicated refinements, modifications and optimizations. It contains three variants, TResNet-M, TResNet-L and TResNet-XL, that vary only in their depth and the number of channels. TResNet architecture contains the following refinements and changes compared to plain ResNet50 design:

- SpaceToDepth Stem
- Anti-Alias Downsampling
- In-Place Activated BatchNorm
- Novel Block-type Selection
- Optimized SE Layers.

Previous works usually offer refinements to ResNet50 which increase the accuracy at the cost of reducing the GPU throughput [11, 20, 13, 46]. Differently, in our design some refinements increase the models' throughput and some decrease it. All-in-all, for TResNet-M we chose a mixture of refinements that provide a similar GPU throughput to ResNet50, for a fair comparison of the models' accuracy.

### 2.1. Refinements

**SpaceToDepth Stem** - Neural networks usually start with a stem unit - a component whose goal is to quickly reduce the input resolution. ResNet50 stem is comprised of a stride-2 conv7x7 followed by a max pooling layer [10], which reduces the input resolution by a factor of $4$ ($224 \rightarrow 56$). ResNet50-D stem design [11], for comparison, is more elaborate - the conv7x7 is replaced by three conv3x3 layers. The new ResNet50-D stem design did improve accuracy, but at a cost of lowering the training throughput - see Table 1, where the new stem design is responsible for almost all the decline in the throughput.

We wanted to replace the traditional convolution-based downscaling unit by a fast and seamless layer, with little information loss as possible, and let the well-designed residual blocks do all the actual processing work. The new stem layer sole functionality should be to downscale the input resolution to match the rest of the architecture, e.g., by a factor of $4$. We met these goals by using a dedicated SpaceToDepth transformation layer [33], that rearranges blocks of spatial data into depth. Notice that in contrast to [33], which mainly used SpaceToDepth in the context of isometric (single-resolution) networks, in our novel design SpaceToDepth is used as a drop-in replacement for the traditional stem unit. The SpaceToDepth layer is followed by simple convolution, to match the number of wanted channels, as can be seen in Figure 1.

**Anti-Alias Downsampling (AA)** - [46] proposed to replace all downscaling layers in a network by an equivalent AA component, to improve the shift-equivariance of deep

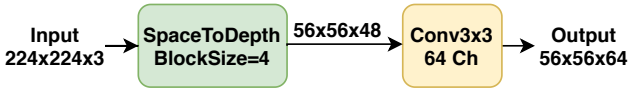


Figure 1. **TResNet-M SpaceToDepth stem design.**

networks and give better accuracy and robustness.

We implemented an economic variant of AA, similar to [20], that provides an improved speed-accuracy trade-off - only our stride-2 convolutions are replaced by stride-1 convolutions followed by a 3x3 blur kernel filter with stride 2, as described in Figure 2.

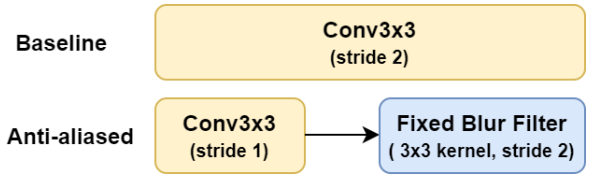


Figure 2. **The AA downsampling scheme of TResNet architecture**. All stride-2 convolutions are replaced by stride-1 convolutions, followed by a fixed downsampling blur filter [46].

**In-Place Activated BatchNorm (Inplace-ABN)** - Along the architecture, we replaced all BatchNorm+ReLU layers by Inplace-ABN [32] layers, which implements BatchNorm with activation as a single inplace operation, allowing to reduce significantly the memory required for training deep networks, with only a small increase in the computational cost. As an activation function for the Inplace-ABN, we chose to use Leaky-ReLU instead of ResNet50's plain ReLU.

Using Inplace-ABN in TResNet models offers the following advantages:

- BatchNorm layers are major consumers of GPU memory. Replacing BatchNorm layers with Inplace-ABN enables to significantly increase the maximal possible batch size, which can improve the GPU utilization.
- For TResNet models, Leaky-ReLU provides better accuracy than plain ReLU. While some modern activation, like Swish and Mish [28], might also give better accuracy than ReLU, their GPU memory consumption is higher, as well as their computational cost. In contrast, Leaky-ReLU has exactly the same GPU memory consumption and computational cost as plain ReLU.
- The increased batch size can also improve the effectiveness of popular algorithms like triplet loss [19] and momentum-contrastive learning. [9]

**Novel Block-Type Selection** - ResNet34 and ResNet50 share the same architecture, with one difference: ResNet34 uses solely 'BasicBlock' layers, which comprise of two conv3x3 as the basic building block, while ResNet50 uses 'Bottleneck' layers, which comprise of two conv1x1 and one conv3x3 as the basic building block [10]. Bottleneck layers have higher GPU usage than BasicBlock layers, but usually give better accuracy. However, BasicBlock layers have larger receptive field, so they might be more suited to be placed at the early stages of a network.

We found that the uniform block selection of ResNet models is not optimal, and a better speed-accuracy trade-off can be obtained using a novel design, which uses a mixture of BasicBlock and Bottleneck layers. Since BasicBlock layers have a larger receptive field, we placed them at the first two stages of the network, and Bottleneck layers at the last two stages.

To create larger variants of TResNet, we modified the number of initial channels, and the number of residual blocks in the 3rd stage, similarly to [10] and [36]. Full specification of TResNet networks, including block type, width and number of residual blocks in each stage, appears in Table 2.

**Optimized SE Layers** - We added dedicated squeeze-and-excitation [13] layers (SE) to TResNet architecture. In order to reduce the computational cost of the SE blocks, and gain the maximal speed-accuracy benefit, we placed SE layers only in the first three stages of the network. The last stage, which works on low-resolution maps, does not get a large accuracy benefit from the global average pooling operation that SE provides.

Compared to standard SE design [13], TResNet SE placement and hyper-parameters are also optimized: For Bottleneck units, we added the SE module after the conv3x3 operation, with a reduction factor of $8$, and for BasicBlock units, we added SE module just before the residual sum, with a reduction factor of $4$. This change aims to reduce the number of parameters and computational cost of SE layers: since BasicBlock units are placed on the first two stages of the network, they contain relatively low number of input channels, so only a small reduction factor ($4$) is needed. The Bottleneck units are placed in later stages of the network, with more input channels, so a higher reduction factor ($8$) is needed. Placing the SE layers after the reduction phase of the Bottleneck layer further reduces the computational cost. The complete blocks design, with SE layers and Inplace-ABN, is presented in Figure 3.

## 2.2. Code Optimizations

In this section we will describe code optimizations we did to enhance the GPU throughput and reduce the memory footprint of TResNet models. While code optimizations are sometimes overlooked and seen as 'implementation details', we claim that they are crucial for designing a modern network with top GPU performance.

We designed TResNet using the PyTorch library [30],

| Layer | Block Type | Output | Stride | TResNet | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | M | | L | | XL | |
| | | | | Repeats | Channels | Repeats | Channels | Repeats | Channels |
| Stem | SpaceToDepth | 56×56 | - | 1 | 48 | 1 | 48 | 1 | 48 |
| | Conv1x1 | | 1 | 1 | 64 | 1 | 76 | 1 | 84 |
| Stage1 | BasicBlock+SE | 56×56 | 1 | 3 | 64 | 4 | 76 | 4 | 84 |
| Stage2 | BasicBlock+SE | 28×28 | 2 | 4 | 128 | 5 | 152 | 5 | 168 |
| Stage3 | Bottleneck+SE | 14×14 | 2 | 11 | 1024 | 18 | 1216 | 24 | 1344 |
| Stage4 | Bottleneck | 7×7 | 2 | 3 | 2048 | 3 | 2432 | 3 | 2688 |
| Pooling | GlobalAvgPool | 1×1 | 1 | 1 | 2048 | 1 | 2432 | 1 | 2688 |
| #Params. | | | | 29.4M | | 54.7M | | 77.1M | |

Table 2. **Overall architecture of the three TResNet models.**

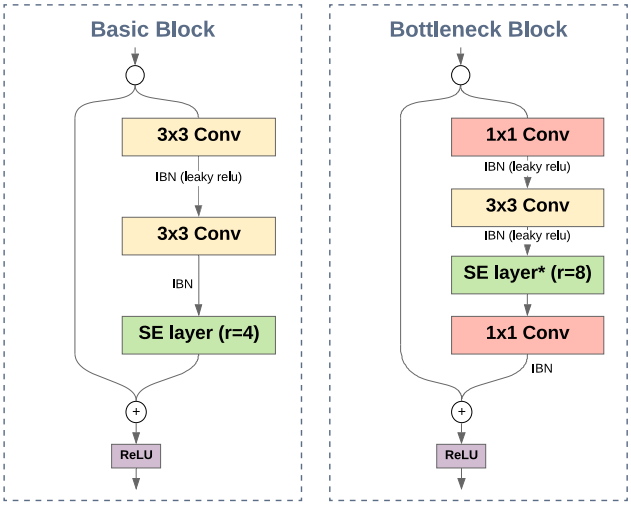


Figure 3. **TResNet BasicBlock and Bottleneck design** (stride 1). IBN = Inplace-BatchNorm, r = reduction factor, * - Only for 3rd stage.

due to its popularity and ability for easy code prototyping. However, all the optimization described below are also applicable to other deep learning libraries, such as TensorFlow [1] and MXNet [4]

#### 2.2.1 JIT Compilation

JIT compilation enables, at execution time, to dynamically compile a high-level code into highly-efficient, optimized machine code. This is in contrast to the default Pythonic option of running a code dynamically, via an interpreter. We used JIT compilations for network modules that don't contain learnable parameters - the AA blur filter and the SpaceToDepth modules. For modules without learnable parameters, JIT compilation is a seamless process that accelerates the network GPU throughput without imposing limitations on the actual training and inference - for example, the input size does not need to be fixed and pre-determined, flow control statements are still possible.

For the AA and SpaceToDepth modules, we found that JIT compilation reduces the GPU cost by almost a factor of two.

#### 2.2.2 Inplace Operations

Inplace operations change directly the content of a given tensor, without making a copy. They reduce the memory access cost of an operation, and also prevent creation of unneeded activation maps for backward propagation, hence increasing the maximal possible batch size. In TResNet code, inplace operations are used as as much as possible. In addition of using Inplace-ABN, there are also inplace operations for the residual connection, SE layers, blocks' final activation and more. This is a key factor in enabling large batch size - TResNet-M maximal batch size is almost twice of ResNet50 - 512, as can be seen in Table 1. For full review of TResNet inplace operations, see our public code.

#### 2.2.3 Fast Global Average Pooling

Global average pooling (GAP) is used heavily in TResNet architecture - both in the SE layers, and before the final fully connected. GAP can be implemented via a general boilerplate method, AvgPool2d, that does average pooling of a tensor to a given spatial dimension, which in the GAP is (1,1). However, usually using this boilerplate methodology is usually not optimal, and leading to sub-optimal performances, mainly due to bad memory utilization.

We found that a simple dedicated implementation of GAP, with optimized code for the specific case of (1,1) spatial output, can be up to 5 times faster than the boilerplate implementation on GPU, since it can bring data from memory more efficiently. Our full TResNet code implementation, include the fast GAP, appears in the github page cited in the abstract.

#### 2.2.4 Deployment Optimizations

There are dedicated optimizations for enhancing (frozen) model inference speed during deployment. For example, BatchNorm layers can be fully absorbed into the convolution layers before them, significantly accelerating the model. In addition, there are dedicated libraries for GPU deployment, such as TensorRT [39]. However, we wanted to provide a fair comparison of TResNet to other architectures, while focusing on all aspects of GPU efficiency (training speed, inference speed and maximal batch size), so we avoided doing inference-tailored optimizations. In practice, the inference speed reported for TResNet can be improved via such optimizations.

## 3. ImageNet Results

In this section, we will evaluate TResNet models on standard ImageNet training (input resolution 224), and compare their top-1 accuracy and GPU throughput to other known models. We will also perform an ablation study to better understand the effect of different refinements, show results for fine-tuning TResNet to higher input resolution, and do a thorough comparison to EfficientNet models.

### 3.1. Basic Training

Our main benchmark for evaluating TResNet models is the popular ImageNet dataset [18]. We trained the models on input resolution 224, for 300 epochs, using a SGD optimizer and 1-cycle policy [34]. For regularization, we used AutoAugment [6], Cutout [7], Label-smoothing [35] and True-weight-decay [25]. We found that the common ImageNet statistics normalization [20, 6, 36] does not improve the training accuracy, and instead normalized all the RGB channels to be between 0 and 1. For comparison, we repeated the same training procedure for ResNet50. Results appear in Table 3.

| Models | Top Training Speed (img/sec) | Top Inference Speed (img/sec) | Max Train Batch Size | Top-1 Acc. [%] |
|---|---|---|---|---|
| ResNet50 | **805** | 2830 | 288 | 79.0 |
| TResNet-M | 730 | **2930** | **512** | 80.8 |
| TResNet-L | 345 | 1390 | 316 | 81.5 |
| TResNet-XL | 250 | 1060 | 240 | **82.0** |

Table 3. **TResNet models accuracy and GPU throughput on ImageNet, compared to ResNet50**. All measurements were done on Nvidia V100 GPU, with mixed precision. All models are trained on input resolution of 224.

We can see from Table 3 that TResNet-M, which has similar GPU throughput to ResNet50, has significantly higher validation accuracy on ImageNet ($+1.8\%$). It also outperforms all the other models that appear in Table 1, both in terms of GPU throughput and ImageNet top-1 accuracy.

Note that our ResNet50 ImageNet top-1 accuracy, $79.0\%$, is significantly higher than the accuracy stated in previous articles [10, 11, 15], demonstrating the effectiveness of our training procedure. In addition, training TResNet-M and ResNet50 models takes less than 24 hours on an 8xV100 GPU machine, showing that our training scheme is also efficient and economical.

Another strength of the TResNet models, as reflected by Table 3, is the ability to work with significantly larger batch sizes than other models. In general, large batch size leads to better GPU utilization, and allows easier scaling to large inputs. For distributed learning, it also reduces the number of synchronization needed in an epoch between the different GPUs.

### 3.2. Ablation Study

#### 3.2.1 Network Refinements

We performed an ablation study to evaluate the impact of the different refinements and modifications in TResNet-M model on the validation accuracy, inference speed, training speed and maximal batch size. Results appear in Table 4. We can better understand from Table 4 the contribution of each refinement:

**SpaceToDepth**: The SpaceToDepth module provides improvements to all the indices. Notice that while we are not the first to use this innovative module (see [33]), we are the first to integrate it into a high-performance network as a drop-in replacement for the traditional convolution-based stem, and get a meaningful improvement. While the GPU throughput improvement is expected, the fact that also the accuracy improves (marginally) when replacing the ResNet stem cell by a "cheaper" SpaceToDepth unit is somewhat surprising. This result supports our intuition that there could be "information loss" within convolution-based stem unit - details from the original image are not propagated well due to the aggressive downscaling process. Although simpler, a SpaceToDepth module can minimizes this loss, and enables to process the data via the residual blocks, which are protected from information loss by the skip connections. We will further investigate this issue in future works.

**Block-Type selection**: Our novel block-type selection scheme, which uses both 'BasicBlock' and 'Bottleneck' blocks in the network, provides significant improvements to all indices, and shows that the uniform block-type design of the ResNet models is not optimal. Notice that this refinement also includes changing the number of residual blocks in the 3rd stage, from 6 to 11. In practice, the actual number (11) was chosen after all the other refinements were finalized. Its goal was to bring TResNet-M to a similar GPU

| Refinement | Top1 Accuracy [%] | Inference Speed [img/sec] | Training Speed [img/sec] | Maximal Batch size |
|---|---|---|---|---|
| ResNet50 | 79.0 | 2830 | 805 | 288 |
| + SpaceToDepth | 79.1 (+0.1) | 2950 (+120) | 830 (+25) | 312 (+24) |
| + Novel Block-Type Selection | 79.4 (+0.3) | 3320 (+370) | 930 (+100) | 424 (+112) |
| + Inplace-ABN | 79.5 (+0.1) | 3470 (+150) | 880 (-50) | 624 (+200) |
| + Optimized SE | 80.3 (+0.8) | 3280 (-190) | 815 (-65) | 596 (-32) |
| + AA | 80.8 (+0.5) | 2930 (-350) | 730 (-85) | 512 (-80) |

Table 4. **Ablation study - The impact of different refinements in TResNet-M model on ImageNet top-1 accuracy, inference speed, training speed and maximal batch size**. All measurements are done on Nvidia V100 GPU, with mixed precision.

throughput as ResNet50.

**Inplace-ABN**: As expected, Inplace-ABN enables us to significantly increase the possible batch size, by 200 images. In addition, using Leaky-ReLU as activation function, instead of plain ReLU, marginally improved the accuracy. However, in terms of contribution to the actual GPU throughput, the impact of Inplace-ABN is mixed: while the inference speed improved, the training speed was somewhat reduced. While Inplace-ABN enables us to increase the batch size (which should be translated to better GPU throughput), it is also a more complicated module than a simple BatchNorm layer, so there is an inherent trade-off for using it. Since the ability to work with larger batch is highly beneficial for multi-GPU training and some dedicated loss functions, we chose to include this refinement.

**Optimized SE + Anti-Aliasing layers**: As expected, these layers significantly improve the ImageNet top-1 accuracy, with a price of reducing the model GPU throughput. We were able to compensate for this decrease with the previous refinements, and all-in-all get a better speed-accuracy trade-off than ResNet50.

#### 3.2.2 Code Optimizations

In Table 5 we evaluate the contribution of the different code optimizations. We can see from Table 5 that among the optimizations, dedicated inplace operations give the greatest boost - not only it improves the GPU throughput, but it also significantly increases the maximal possible batch size, since it avoids the creation of unneeded activation maps for backward propagation.

### 3.3. High-Resolution Fine-Tuning

We tested the scaling of TResNet models to higher input resolutions on ImageNet. We used the pre-trained TResNet models that appear in Table 3 as a starting point, and did a short 10 epochs fine-tuning to input resolution of 448. The results appear in Table 6.

| Refinement | Inference Speed [img/sec] | Training Speed [img/sec] | Maximal Batch size |
|---|---|---|---|
| TResNet-M No Code Optimizations | 2790 | 670 | 420 |
| + JIT | 2830 (+40) | 695 (+25) | 420 (0) |
| + Inplace Operations | 2910 (+80) | 715 (+20) | 512 (+92) |
| + Fast GAP | 2930 (+20) | 730 (+15) | 512 (0) |

Table 5. **Ablation study - The impact of code optimizations in TResNet-M model on inference speed, training speed and maximal batch size**.

| Model | Input Resolution | Top-1 Accuracy [%] |
|---|---|---|
| TResNet-M | 224 | 80.8 |
| TResNet-M | 448 | 83.2 |
| TResNet-L | 224 | 81.5 |
| TResNet-L | 448 | 83.8 |
| TResNet-XL | 224 | 82.0 |
| TResNet-XL | 448 | **84.3** |

Table 6. **Impact of the input resolution on the top1 ImageNet accuracy for TResNet models**. All TResNet 448 input-resolution accuracies are obtained with 10 epochs of fine-tuning.

We see from Table 6 that TResNet models scale well to high resolutions. Even TResNet-M, which is a relatively small and compact model, can achieve top-1 accuracy of 83.2% on ImageNet with high-resolution input. TResNet

largest variant, TResNet-XL, achieves 84.3% top-1 accuracy on ImageNet.

### 3.4. Comparison to EfficientNet Models

EfficientNet models, which are based on MobilenetV3 architecture [12], propose to balance the resolution, height, and width of a base network for generating a series of larger networks. They are considered state-of-the-art architectures, that provide efficient networks for all ImageNet top-1 accuracy spectrum [36]. In Figure 4 and Figure 5, we compare the inference and training speed of TResNet models to the different EfficientNet models respectively.

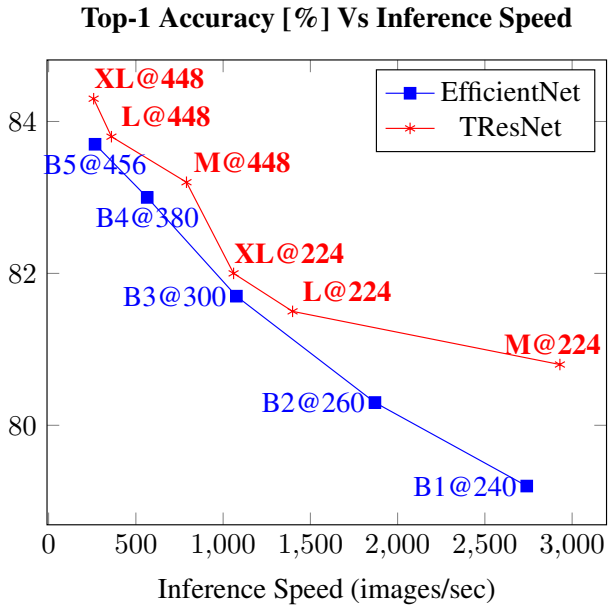


Figure 4. **TResNet Vs EfficientNet models inference speed comparison.** Y label is the accuracy[%]

We can see from Figure 4 and Figure 5 that all along the top-1 accuracy curve, TResNet models give better inference-speed-accuracy and training-speed-accuracy tradeoff than EfficientNet models. Note that each EfficientNet model was bundled and optimized to a specific resolution, while TResNet models were trained and tested on multi-resolutions, which makes this comparison biased toward EfficientNet models; Yet, TResNet models show superior results. Also note that EfficientNet models were trained for 450 epochs and not for 300 epochs like TResNet models, and that EfficientNet training procedure included more GPU intensive tricks (RMSProp optimizer, drop-block) [36], so the actual gap in training times is even higher than stated in Figure 5.

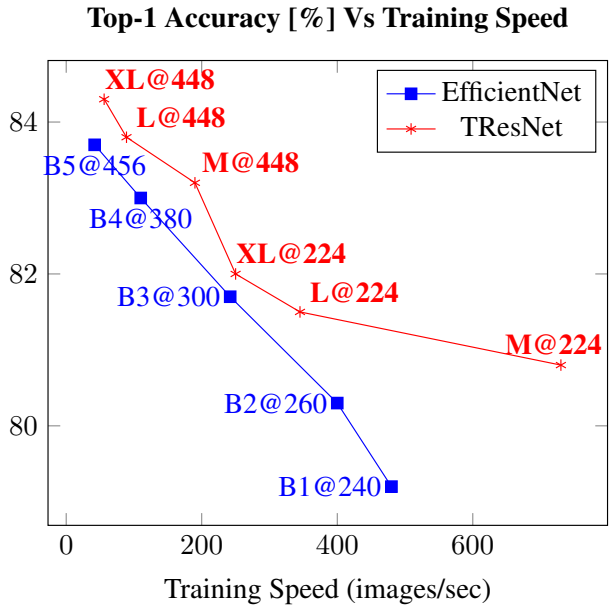


Figure 5. **TResNet Vs EfficientNet models training speed comparison.** Y label is the accuracy[%]

## 4. Transfer Learning Results

In this section, we will present transfer learning results of TResNet models on four well-known single-label classification downstream datasets. We will also present transfer learning results on multi-label classification and object detection tasks.

### 4.1. Single-Label Classification

We evaluated TResNet on four commonly used, competitive transfer learning datasets: Stanford-Cars [16], CIFAR-10 [17], CIFAR-100 [17] and Oxford-Flowers [29]. For each dataset, we used ImageNet pre-trained checkpoints, and fine-tuned the models for 80 epochs using 1-cycle policy [34] . For the fine-grained classification tasks (Stanford-Cars and Oxford-Flowers), in addition to cross-entropy loss we used weighted triplet loss with soft-margin [31, 19], which emphasizes hard examples by focusing of the most difficult positives and negatives samples in the batch. Table 7 shows the transfer learning performance of TResNet, compared to the known state-of-the-art models.

We can see from Table 7 that TResNet surpasses or matches the state-of-the-art accuracy on 3 of the 4 datasets, with x8-15 faster GPU inference speed. Note that all TResNet's results are from single-crop single-model evaluation.

### 4.2. Multi-Label Classification

For multi-label classification tests, we chose to work with MS-COCO dataset [23] (multi-label recognition task). We used the 2014 split, which contains about 82K images

| Dataset | Model | Top-1 Acc. | Speed img/sec | Input |
|---|---|---|---|---|
| CIFAR-10 | Gpipe | **99.0** | - | 480 |
| | TResNet-XL | **99.0** | **1060** | 224 |
| CIFAR-100 | EfficientNet-B7 | **91.7** | 70 | 600 |
| | TResNet-XL | 91.5 | **1060** | 224 |
| Stanford Cars | EfficientNet-B7 | 94.7 | 70 | 600 |
| | TResNet-L | **96.0** | **500** | 368 |
| Oxford-Flowers | EfficientNet-B7 | 98.8 | 70 | 600 |
| | TResNet-L | **99.1** | **500** | 368 |

Table 7. **Comparison of TResNet to state-of-the-art models on transfer learning datasets (only ImageNet-based transfer learning results)**. Models inference speed is measured on a mixed precision V100 GPU. Since no official implementation of Gpipe was provided, its inference speed is unknown.

for training and 41K for validation. In total, images are involved with 80 object labels, with an average of 2.9 labels per image. Our training scheme is similar to the one used for single-label training. The main difference is the loss function, which is adapted for a multi-label setting - we implemented a variant of the well known focal-loss [22], where two different gamma values are used for positive and negative sample. This enables to better tackle the highly imbalanced nature of a multi-label dataset.

Following the conventional settings [5, 40], we report the main performance evaluation metric, mean average precision (mAP), but in addition state average per-class precision (CP), recall (CR), F1 (CF1) and the average overall precision (OP), recall (OR), F1 (OF1). In Table 9, we present the transfer learning results of TResNet model and compare it to the known state-of-the-art model.

| Backbone | mAP | CP | CR | CF1 | OP | OR | OF1 |
|---|---|---|---|---|---|---|---|
| KSSNet[40] | 83.7 | 84.6 | 73.2 | 77.2 | 87.8 | 76.2 | 81.5 |
| TResNet-L | **86.4** | **87.6** | **76.0** | **81.4** | **88.4** | **78.9** | **83.4** |

Table 9. **Comparison of TResNet to state-of-the-art model on multi-label classification on MS-COCO dataset**. KSSNet [40], is the known SOTA, based on ResNet101 backbone.

We can see from Table 9 that the TResNet-based solution significantly outperforms previous top solution for MS-COCO multi-label dataset, increasing the known SOTA by a large margin, from 83.7 mAP to 86.4 mAP. All additional evaluation metrics also show improvement.

### 4.3. Object Detection

While our main focus was on various classification tasks, we wanted to further test TResNet on another popular computer vision task - object detection.

We used the known MS-COCO [23] dataset (object detection task), with a training set with 118k images, and an evaluation set (minival) of 5k images. For training, we used the popular mm-detection [3] package, with FCOS [38] as the object detection method and the enhancements discussed in ATSS [47].

We trained with SGD optimizer for 70 epochs with 0.9 momentum, weight decay of 0.0001 and batch size of 24. We used learning rate warm up, initial learning rate of 0.01 and 10x reduction at epochs 40, 60. We also implemented the data augmentations techniques described in [24]. For a fair comparison, we used first ResNet50 as backbone, and then replace it by TResNet-M (both give similar GPU throughput). Comparison results appear in Table 10.

| Method | Babkbone | mAP % |
|---|---|---|
| FCOS | ResNet50 | 42.8 |
| FCOS | TResNet-M | **44.0** |

Table 10. **Comparison of TResNet-M to ResNet50 on MS-COCO object detection task**. Results were obtained using mm-detection package, with FCOS as the object detection method .

We can see from Table 10 that TResNet-M outperform ResNet50 on this object-detection task, increasing COCO mAP score from 42.8 to 44.0. This is consistent with the improvement we saw in the single-label ImageNet classification task.

## 5. Conclusion

In this paper, we point out a possible blind-spot of latest developments in neural network design patterns. They tend not to consider actual GPU utilization as one of the measurements for a network quality. While GPU inference speed is sometimes measured, GPU training speed and maximal possible batch size are widely overlooked. For many real-world deep learning applications, training speed, inference speed and maximal batch size are all critical factors. To address this issue, we propose a carefully selected set of design refinements, which are highly effective in utilizing typical GPU resources - SpaceToDepth stem cell, economical AA downsampling, Inplace-ABN operations, block-type selection redesign and optimized SE layers. We combine these refinements with a series of code optimizations and enhancements to suggest a family of new models, dedicated for GPU high-performance, which we call TResNet. We demonstrate that on ImageNet, all along the top-1 accuracy curve TResNet gives better GPU throughput than existing models. In addition, on three commonly used downstream single-label classification datasets it reaches new state-of-the-art accuracies. We also show that TResNet generalizes well to other computer vision tasks, reaching top scores on multi-label classification and object detection datasets.

## References


[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[4] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[5] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. Multi-label image recognition with graph convolutional networks, 2019.

[6] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123, 2019.

[7] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[8] Shanghua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip HS Torr. Res2net: A new multi-scale backbone architecture. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[9] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2019.

[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] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019.

[12] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[14] Jeremiah W Johnson. Adapting mask-rcnn for automatic nucleus segmentation. *arXiv preprint arXiv:1805.00500*, 2018.

[15] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Large scale learning of general visual representations for transfer. *arXiv preprint arXiv:1912.11370*, 2019.

[16] Jonathan Krause, Jia Deng, Michael Stark, and Li Fei-Fei. Collecting a large-scale dataset of fine-grained cars. 2013.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[19] Hussam Lawen, Avi Ben-Cohen, Matan Protter, Itamar Friedman, and Lihi Zelnik-Manor. Compact network training for person reid. *Proceedings of the 2020 International Conference on Multimedia Retrieval*, Jun 2020.

[20] Jungkyu Lee, Taeryun Won, and Kiho Hong. Compounding the performance improvements of assembled techniques in a convolutional neural network. *arXiv preprint arXiv:2001.06268*, 2020.

[21] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *arXiv preprint arXiv:1804.06215*, 2018.

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, pages 21–37, 2016.

[25] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[26] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[27] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[28] Diganta Misra. Mish: A self regularized non-monotonic neural activation function, 2019.

[29] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[31] Ergys Ristani and Carlo Tomasi. Features for multi-target multi-camera tracking and re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6036–6046, 2018.

[32] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[33] Mark Sandler, Jonathan Baccash, Andrey Zhmoginov, and Andrew Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution. *arXiv preprint arXiv:1909.03205*, 2019.

[34] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[35] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[36] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[37] Mingxing Tan and Quoc V Le. Mixnet: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595*, 2019.

[38] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection, 2019.

[39] Han Vanholder. Efficient inference with tensorrt, 2016.

[40] Ya Wang, Dongliang He, Fu Li, Xiang Long, Zhichao Zhou, Jinwen Ma, and Shilei Wen. Multi-label classification with label graph superimposing. *ArXiv*, abs/1911.09243, 2019.

[41] Ross Wightman. pytorch-image-models, 2019. https://github.com/rwightman/pytorch-image-models.

[42] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 466–481, 2018.

[43] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[44] Rengan Xu, Frank Han, and Quy Ta. Deep learning at scale on nvidia v100 accelerators. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 23–32. IEEE, 2018.

[45] Masafumi Yamazaki, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, and Kohta Nakashima. Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds. *arXiv preprint arXiv:1903.12650*, 2019.

[46] Richard Zhang. Making convolutional networks shift-invariant again. In *ICML*, 2019.

[47] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z. Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection, 2019.