Shape Adaptor: A Learnable Resizing Module

Shikun Liu*[1], Zhe Lin[2], Yilin Wang[2], Jianming Zhang[2], Federico Perazzi[2], and Edward Johns[1]

[1]Department of Computing, Imperial College London
[2]Adobe Research

Abstract

We present a novel resizing module for neural networks: *shape adaptor*, a drop-in enhancement built on top of traditional resizing layers, such as pooling, bilinear sampling, and strided convolution. Whilst traditional resizing layers have fixed and deterministic reshaping factors, our module allows for a learnable reshaping factor. Our implementation enables shape adaptors to be trained end-to-end without any additional supervision, through which network architectures can be optimised for each individual task, in a fully automated way. We performed experiments across seven image classification datasets, and results show that by simply using a set of our shape adaptors instead of the original resizing layers, performance increases consistently over human-designed networks, across all datasets. Additionally, we show the effectiveness of shape adaptors on two other applications: network compression and transfer learning. The source code is available at: `github.com/lorenmt/shape-adaptor`.

1 Introduction

Deep neural networks have become popular for many machine learning applications, since they provide simple strategies for end-to-end learning of complex representations. However, success can be highly sensitive to network architectures, which places a great demand on manual engineering of architectures and hyper-parameter tuning.

A typical human-designed convolutional neural architecture is composed of two types of computational modules: i) a *normal layer*, such as a stride-1 convolution or an identity mapping, which maintains the spatial dimension of incoming feature maps; ii) a *resizing layer*, such as max/average pooling, bilinear sampling, or stride-2 convolution, which reshapes the incoming feature map into a different spatial dimension. We hereby define the *shape* of a neural network as the composition of the feature dimensions in all network layers, and the *architecture* as the overall structure formed by stacking multiple normal and resizing layers.

To move beyond the limitations of human-designed network architectures, there has been a growing interest in developing Automated Machine Learning (AutoML) algorithms [13] for

[*] Corresponding Author: shikun.liu17@imperial.ac.uk.

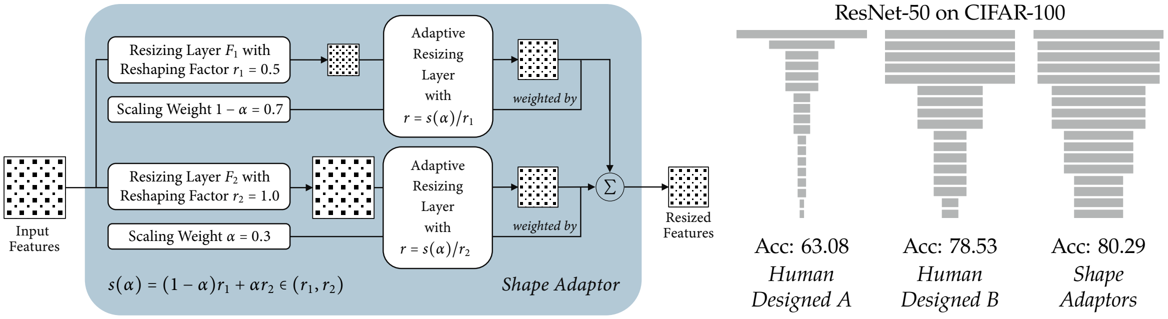

Resizing Layer F₁ with Reshaping Factor r₁ = 0.5
Scaling Weight 1 − α = 0.7
Adaptive Resizing Layer with r = s(α)/r₁
weighted by
Input Features
Resizing Layer F₂ with Reshaping Factor r₂ = 1.0
Scaling Weight α = 0.3
Adaptive Resizing Layer with r = s(α)/r₂
weighted by
Σ
Resized Features
s(α) = (1 − α)r₁ + αr₂ ∈ (r₁, r₂)
Shape Adaptor
ResNet-50 on CIFAR-100
Acc: 63.08
Human Designed A
Acc: 78.53
Human Designed B
Acc: 80.29
Shape Adaptors

Figure 1: Left: Visualisation of a shape adaptor module build on top of two resizing layers. Right: Different network shapes in the exact same network architecture ResNet-50 can result a significantly different performance.

automatic architecture design, known as Neural Architecture Search (NAS) [30, 2, 21, 20]. However, whilst this has shown promising results in discovering powerful network architectures, these methods still rely heavily on human-designed network shapes, and focus primarily on learning connectivities between layers. Typically, reshaping factors of 0.5 (down-sampling) and 2 (up-sampling) are chosen, and the total number of reshaping layers is defined manually, but we argue that network shape is an important inductive bias which should be directly optimised.

For example, Figure 1 Right shows three networks with the exact same design of network structure, but different shapes. For the two human-designed networks [11], we see that a ResNet-50 model designed specifically for CIFAR-100 dataset (Human Designed B) leads to a 15% performance increase over a ResNet-50 model designed for ImageNet dataset (Human Designed A). The performance can be further improved with the network shape designed by the shape adaptors we will later introduce. Therefore, by learning network shapes rather than manually designing them, a more optimal network architecture can be found.

To this end, we propose *Shape Adaptor*, a novel resizing module which can be dropped into any standard neural network to learn task-specific network shape. A shape adaptor module (see Figure 1 Left) takes in an input feature, and reshapes it into two intermediate features. Each reshaping operation is done using a standard resizing layer $F_i(x, r_i), i = 1, 2$, where each resizing layer has a different, pre-defined reshaping factor r_i to reshape feature map x. These two reshaping factors then define the search space (r_1, r_2) (assuming $r_1 < r_2$) for a shape adaptor module. Finally, the two intermediate features are softly combined with a scalar weighting $1 - \alpha$ and α respectively (for $\alpha \in (0, 1)$), after reshaping them into the same spatial dimension via a learned reshaping factor in the search space $s(\alpha) \in (r_1, r_2)$. The module's output represents a mixed combination over these two intermediate features, and the scalar α can be learned solely based on the task-specific training loss with stochastic gradient descent, without any additional supervision. Thus, by simply optimising these scaling weights for every shape adaptor, the entire neural architecture is differential and we are able to learn network shape in an automated, end-to-end manner.

We evaluated shape adaptors on seven standard image classification datasets of various complexities. Our results show that shape adaptors can consistently improve human-designed networks, and notably achieve up to 10% relative performance gains on fine-grained classification datasets. Further experiments show that shape adaptors are robust to initialisations and hyper-parameters, and for a given dataset, they consistently result in the same overall network shape, suggesting that shape adaptors are able to achieve a globally optimal shape. Finally, we further show the effectiveness of shape adaptors in two additional applications: automated neural shape compression, and architecture-level transfer learning.

2 Related Work & Background

Neural Architecture Search Neural architecture search (NAS) presents an interesting research direction in AutoML, in automatically discovering an optimal neural structure for a particular dataset, alleviating the hand design of neural architectures which traditionally involves tedious trial-and-error. NAS approaches can be highly computationally demanding, requiring hundreds of thousands of GPU days of search time, due to intensive techniques such as reinforcement learning [43] and evolutionary search [31]. Several approaches have been proposed to speed up the search, based on parameter sharing [30], hyper-networks [1], and gradient-based optimisation [21]. But despite their promising performance, these approaches come with controversial debate questioning the lack of reproducibility, and sensitivity to initialisations [19, 39]. Whilst NAS methods learn network structures based on pre-defined network shapes, shape adaptors are designed in an orthogonal direction, and instead search network shapes in pre-defined network structures. Nevertheless, shape adaptors could be potentially incorporated into modern NAS frameworks, which we consider as future work.

Architecture Pruning & Compression Network pruning is another direction towards obtaining optimal network architectures. But instead of searching from scratch as in NAS, network pruning is applied to existing human-design networks and removes redundant neurons and connectivities. Such methods can be based on $\mathcal{L}_0$ regularisation [23], batch-norm scaling parameters [22], and weight quantization [9]. As with our shape adaptors, network pruning does not require the extensive search cost of NAS, and can performed alongside regular training. Our shape adaptors can also be formulated as a pruning algorithm, by optimising the network shape within a bounded search space. We provide a detailed explanation of this in Section 5.1.

Design of Resizing Modules A resizing module is one of the essential components in deep convolutional network design, and has seen continual modifications to improve performance and efficiency. The most widely used resizing modules are max pooling, average pooling, bilinear sampling, and strided convolutions, which are deterministic, efficient, and simple. But despite their benefits in increasing computational efficiency and providing regularisation, there are two issues with current designs: i) *lack of spatial invariance*, and ii) *fixed scale*. Prior works focus on improving spatial robustness with a learnable combination between max and average pooling [38, 18], and with anti-aliased low-pass filters [41]. Other works impose regularisation and adjustable inference by stochastically inserting pooling layers [40, 17], and sampling different network shapes [42]. In contrast, shape adaptors solve both problems simultaneously, with a learnable mixture of features in different scales, and with which reshaping factors can be optimised automatically based on the training objective.

3 Shape Adaptors

In this section, we introduce the details of the proposed shape adaptor module. We discuss the definition of these modules, and the optimisation strategy used to train them.

3.1 Formation of Shape Adaptors

A visual illustration of a shape adaptor module is presented in Figure 1 Left. It is a two-branch architecture composed of two different resizing layers $F_i(x, r_i)_{i=1,2}$, assuming $r_1 < r_2$, taking the same feature map x as the input. A resizing layer F_i can be any classical sampling layer, such as max pooling, average pooling, bilinear sampling, or strided convolution, with a fixed reshaping factor r_i. Each resizing layer reshapes the input feature map by this factor, which represents the ratio of spatial dimension between the output and input feature maps, and outputs an *intermediate feature*. An adaptive resizing layer G with a learnable reshaping factor is then used to reshape these intermediate features into the same spatial dimension, and combine them with a weighted average to compute the module's output.

Each module has a learnable parameter $\alpha \in (0, 1)$, parameterised by a sigmoid function, which is the only extra learnable parameter introduced by shape adaptors. The role of α is to optimally combine two intermediate features after reshaping them by an adaptive resizing layer G. To enable a non-differential reshaping factor in G to be learned, we use a monotone function s, which monotonically maps from α into the search space $s(\alpha) \in \mathcal{R} = (r_1, r_2)$, representing the scaling ratio of the module's reshaping operation. With this formulation, a learnble reshaping factor $s(\alpha)$ allows a shape adaptor to reshape at any scale between r_1 and r_2, rather than being restricted to a discrete set of scales as with typical manually-designed network architectures.

Using this formulation, a shape adaptor module can be expressed as function:

$$\texttt{ShapeAdaptor}(x, \alpha, r_{1,2}) = (1-\alpha) \cdot G\left(F_1(x, r_1), \frac{s(\alpha)}{r_1}\right) + \alpha \cdot G\left(F_2(x, r_2), \frac{s(\alpha)}{r_2}\right), \quad (1)$$

with reshaping factor $s(\alpha)$, a *monotonic* function which satisfies,

$$\lim_{\alpha \to 0} s(\alpha) = r_1, \quad \text{and} \quad \lim_{\alpha \to 1} s(\alpha) = r_2. \quad (2)$$

We choose our adaptive resizing layer G to be a bilinear interpolation function, which allows feature maps to be resized into any shape. We design module's learnable reshaping factor $s(\alpha) = (r_2 - r_1)\alpha + r_1$, a convex combination over these pre-defined reshaping factors, assuming having no prior knowledge on the network shape.

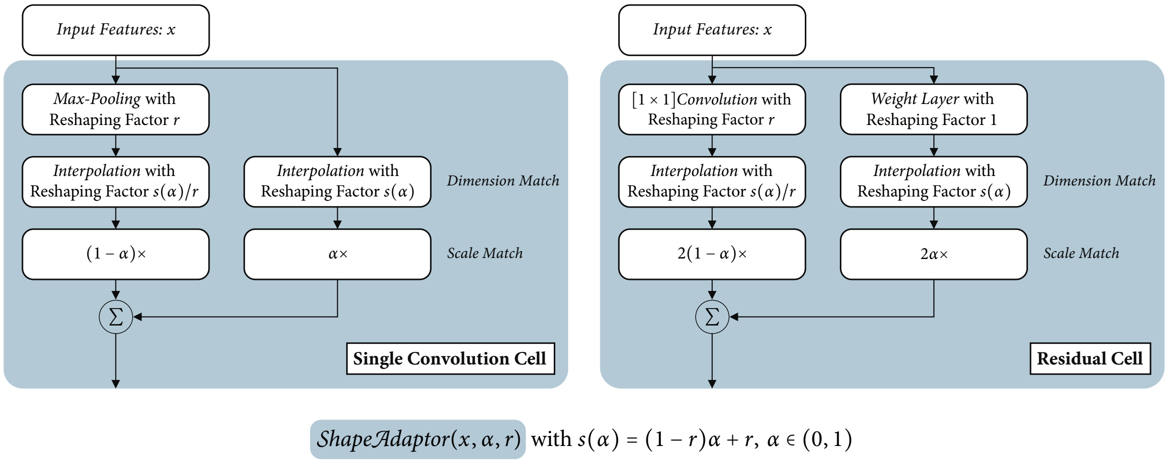

Input Features: x
Max-Pooling with Reshaping Factor r
Interpolation with Reshaping Factor s(α)/r
Interpolation with Reshaping Factor s(α)
Dimension Match
(1 − α)×
α×
Scale Match
Σ
Single Convolution Cell
Input Features: x
[1 × 1]Convolution with Reshaping Factor r
Weight Layer with Reshaping Factor 1
Interpolation with Reshaping Factor s(α)/r
Interpolation with Reshaping Factor s(α)
Dimension Match
2(1 − α)×
2α×
Scale Match
Σ
Residual Cell
ShapeAdaptor(x, α, r) with s(α) = (1 − r)α + r, α ∈ (0, 1)

Figure 2: Visualisation of a down-sampling shape adaptor built on a single convolutional cell and a residual cell with a reshaping factor in the range $\mathcal{R} = (r, 1)$.

Each shape adaptor is arranged as a soft and learnable operator to search the optimal reshaping factor $s(\alpha^*) = r^* \in \mathcal{R}$ over a combination of intermediate reshaped features $F_i(x, r_i)$. Thus, it can also be easily coupled with a continuous approximate of categorical distribution, such as Gumbel SoftMax [14, 24], to control the softness. This technique is commonly used in gradient-based NAS methods [21], where a categorical distribution is learned over different operations.

The overall shape adaptor module ensures that its reshaping factor $s(\alpha)$ can be updated through the updated scaling weights. Thus, we enable differentiability of $s(\alpha)$ in a shape adaptor module as an approximation from the mapping of the derivative of its learnable scaling weight: $\nabla s(\alpha) \approx s(\nabla \alpha)$. This formulation enables shape adaptors to be easily trained with standard back-propagation and end-to-end optimisation.

In our implementation, we use one resizing layer to maintain the incoming feature dimension (an identity layer), and the other resizing layer to change the dimension. If F_2 is the layer which maintains the dimension with $r_2 = 1$, then a shape adaptor module acts as a learnable down-sampling layer when $0 < r_1 < 1$, and a learnable up-sampling layer when $r_1 > 1$.

In Figure 2, we illustrate our learnable down-sampling shape adaptor in two commonly used computational modules: a single convolutional cell in VGG-like neural networks [35], and a residual cell in ResNet-like neural networks [11]. To seamlessly insert shape adaptors into human-designed networks, we build shape adaptors on top of the same sampling functions used in the original network design. For example, in a single convolutional cell, we apply max pooling as the down-sampling layer, and the identity layer is simply an identity mapping. And in a residual cell, we use the 'shortcut' $[1 \times 1]$ convolutional layer as the down-sampling layer, and the weight layer stacked with multiple convolutional layers as the identity layer. In the ResNet design, we double the scaling weights in the residual cell, in order to match the same feature scale as in the original design.

Shape adaptors can also be designed in more than two branches, into a more general manner. This general design enables shape adaptors to be inserted into more complicated network architectures, such as ResNeXt [37] and Xception [4]. The general formation of shape adaptors is further discussed in Appendix A.

3.2 The Optimisation Recipe

Implementations of Shape Adaptors In practice, where we have data whose spatial dimension is an integer multiple, a different rounding method in the implementation would result in different learning dynamics. In order to enable shape adaptors perform at the highest efficiency, we propose two types of implementation aiming for specific use cases. Assuming we insert shape adaptors in every network layer, and with the spatial dimension of input data $\mathcal{D}^{in}$, the output dimensionality of the $k^{th}(k \geq 1)$ shape adaptor module $\mathcal{D}^{(k)}$, with its corresponding reshaping factor $r^{(k)}$, is defined as:

- Local type:

$$\mathcal{D}^{(k)} = \left\lfloor \mathcal{D}^{(k-1)} \cdot r^{(k)} \right\rfloor, \quad \mathcal{D}^{(0)} = \mathcal{D}^{in} \tag{3}$$

- Global type:

$$\mathcal{D}^{(k)} = \left\lfloor \mathcal{D}^{in} \cdot \prod_{i \leq k} r^{(i)} \right\rceil \tag{4}$$

where $\lfloor \cdot \rfloor$ represents a floor function, and $\lfloor \cdot \rceil$ represents a round function.

The local type implementation corresponds to the same implementation in classical resizing layers, that is to compute the current layer dimension by reshaping the output feature dimension from the previous layer. The global type implementation is a new method introduced aiming for *precise resizing*: a shape adaptor reshapes input features by a holistic reshaping factor based on all previous resizing layers. This would be particularly useful when we insert a large number of resizing layers, or when we have training dataset in a small spatial dimension, and both of which could lead to a shape collapse by a local implementation (resulting in a very small shape despite having a large reshaping factor in every resizing layer). The key difference between these two types of implementations is: a local type down-sampling shape adaptor will guarantee to drop at least one spatial dimension, whilst a global type down-sampling shape adaptor can retain the spatial dimension if desired.

For example, suppose we have training data with input spatial dimension $D^{in} = 32$, optimised with a deep network composed with 20 resizing layers by the same reshaping factor $r = 0.95$. The local implementation would produce an output dimension of 6, which would be much smaller than the global implementation producing an output dimension of 11.

Number of Shape Adaptors Theoretically, shape adaptors should be inserted into every network layer, to enable maximal search space and flexibility. In practice, we found that beyond a certain number of shape adaptors, performance actually began to degrade. We therefore designed a heuristic to choose an appropriate number of shape adaptor modules

N, based on the assumption that each module contributes a roughly equal amount towards the network's overall resizing effect. Let us consider that each module resizes its input feature map in the range (r_{min}, r_{max}). The overall number of modules required should be sufficient to reshape the network's input dimension of $\mathcal{D}^{in}$ to a manually defined output dimension $\mathcal{D}^{last}$, by applying a sequence of reshaping operations, where each is $\sim r_{min}$. As such, the optimal number of modules can be expressed as a logarithmic function of the overall ratio between the network's input and output, based on the scale of the reshaping operation in each module:

$$N = \left\lfloor \log_{1/r_{min}}(\mathcal{D}^{in}/\mathcal{D}^{last}) \right\rfloor \tag{5}$$

Initialisations in Shape Adaptors As with network weights, a good initialisation for shape adaptors, i.e. the initial values for α, is important. Again, assuming we have every shape adaptor designed in the same search space $\mathcal{R} = (r_{min}, r_{max})$ with the reshaping factor $s(\alpha) = (r_{max} - r_{min})\alpha + r_{min}$, we propose a formula to automatically compute the initialisations such that the output feature dimension of the initialised shape would map to the user-defined dimension D^{out}. Assuming we want to initialise the raw scaling parameters $\bar{\alpha}$ before sigmoid function $\alpha = \sigma(\bar{\alpha})$, we need to solve the following equation:

$$\mathcal{D}^{in} \cdot s(\sigma(\bar{\alpha}))^N = \mathcal{D}^{out}. \tag{6}$$

Suppose we use N as defined in Eq. 5, then Eq. 6 is only solvable when $D^{last} \leq D^{out}$. Otherwise, we then initialise the smallest possible shape when encountering the case for $D^{last} > D^{out}$. This eventually derives the following:

$$\bar{\alpha} = \begin{cases} \ln\left(-\frac{\sqrt[N]{\mathcal{D}^{out}/\mathcal{D}^{in}} - r_{min}}{\sqrt[N]{\mathcal{D}^{out}/\mathcal{D}^{in}} - r_{max}} + \epsilon\right) & \text{if } \mathcal{D}^{last} \leq \mathcal{D}^{out} \\ \ln(\epsilon) & \text{otherwise} \end{cases}, \quad \epsilon = 10^{-4}. \tag{7}$$

where ϵ is a small value to avoid encountering ±∞ values.

In practice, we need to avoid having the case when $\mathcal{D}^{last} > \mathcal{D}^{out}$, which would become a marginal point from a sigmoid function that eventually would receive a very small gradient.

Shape Adaptors with Memory Constraint During experiments, we observed that shape adaptors tend to converge to a larger shape than the human designed network, which may then require very large memory. For practical applications, it is desirable to have a constrained search space for learning the optimal network shape given a user-defined memory limit. For any layer designed with down-sampling shape adaptors, the spatial dimension of which is guaranteed to be smaller than the one from the previous layers. We thus again use the final feature dimension to approximate the memory usage for the network shape.

Suppose we wish to constrain the network shape with the final feature dimension to be no greater than D^{limit}. We then limit the scaling factors in shape adaptors by use of a penalty value ρ, which is applied whenever the network's final feature dimension after the current update D^{cout} is greater than the defined limit, i.e. when $D^{cout} > D^{limit}$. When this occurs, the penalty term ρ is applied on every shape adaptor module, and we compute ρ dynamically for every iteration so that we make sure $D^{cout} \leq D^{limit}$ in the entire training stage. The penalised scaling parameter α_ρ is then defined as follows,

$$\alpha_\rho = \alpha \cdot \rho + \frac{r_{min}}{r_{max} - r_{min}}(\rho - 1). \tag{8}$$

Then the penalised module's reshaping factor $s(\alpha_\rho)$ becomes,

$$s(\alpha_\rho) = (r_{max} - r_{min})\alpha_\rho + r_{min} = s(\alpha)\rho. \tag{9}$$

Using Eq. 6, we can compute ρ as,

$$\rho = \sqrt[N]{\frac{\mathcal{D}^{limit}}{\mathcal{D}^{cout}}}. \tag{10}$$

Iterative Optimisation Strategy To optimise a neural network equipped with shape adaptor modules, there are two sets of parameters to learn: the weight parameters $\boldsymbol{w} = \{w_i\}$, and the shape parameters $\boldsymbol{\alpha} = \{\alpha_i\}$. Unlike NAS algorithms which require optimisation of network weights and structure parameters on separate datasets, shape adaptors are optimised on the same dataset and require no re-training.

Since the parameter space for the network shape is significantly smaller than the network weight, we update the shape parameters less frequently than the weight parameters, at a rate of once every α_s steps. The entire optimisation for a network equipped with shape adaptors is illustrated in Algorithm 1.

Algorithm 1: Optimisation for Shape Adaptor Networks

```
Define: shape adaptors: $\alpha_s, r_{min}, r_{max}, D^{last}, D^{out}, D^{limit}$
Define: network architecture $f_{\boldsymbol{\alpha},\boldsymbol{w}}$ defined with shape and network parameters
Initialise: shape parameters: $\boldsymbol{\alpha} = \{\alpha_i\}$ with Eq. 5, and Eq. 7
Initialise: weight parameters: $\boldsymbol{w} = \{w_i\}$
Initialise: learning rate: $\lambda_1, \lambda_2$
while not converged do
    for each training iteration $i$ do
        $(x_{(i)}, y_{(i)}) \in (x, y)$                                   ▷ fetch one batch of training data
        if requires memory constraint then
            Compute: $\rho$ using Eq. 10
        else
            Define: $\rho = 1$
        end
        if in $\alpha_s$ step then
            Update: $\boldsymbol{\alpha} \leftarrow \lambda_1 \nabla_{\boldsymbol{\alpha}} \mathcal{L}(f_{\boldsymbol{\alpha}_\rho,\boldsymbol{w}}(x_{(i)}), y_{(i)})$      ▷ update shape parameters
        end
        Update: $\boldsymbol{w} \leftarrow \lambda_2 \nabla_{\boldsymbol{w}} \mathcal{L}(f_{\boldsymbol{\alpha}_\rho,\boldsymbol{w}}(x_{(i)}), y_{(i)})$          ▷ update weight parameters
    end
end
```

4 Experiments

In this section, we present experimental results to evaluate shape adaptors on image classification tasks. Please see the Appendix for further results, and a list of negative results for other experiments we attempted.

4.1 Experimental Setup

Datasets We evaluate on seven different image classification datasets, with varying sizes and complexities to fully assess the robustness and generalisation of shape adaptors. These seven datasets are divided into three categories: i) small (resolution) datasets: CIFAR-10/100 [16], SVHN [7]; ii) fine-grained classification datasets: FGVC-Aircraft (Aircraft) [25], CUBS-200-2011 (Birds) [36], Stanford Cars (Cars) [15]; and iii) ImageNet [5]. Small datasets are in resolution [32×32], and fine-grained classification and ImageNet datasets are in resolution [224×224].

Baselines We ran experiments with three widely-used networks: VGG-16 [35], ResNet-50 [11], and MobileNetv2 [34]. The baseline *Human* represents the original human-designed networks, which require manually adjusting the number of resizing layers according to the resolution of each dataset. For smaller [32×32] datasets, human-designed VGG-16, ResNet-50 and MobileNetv2 networks were equipped with 4, 3, 3 resizing layers respectively, and for [224×224] datasets, all human designed networks have 5 resizing layers.

Implementation of Shape Adaptors For all experiments in this section, since we assume no prior knowledge of the optimal network architecture, we inserted shape adaptors uniformly into the network layers (except for the last layer). We initialised shape adaptors with $D^{last} = 2, D^{out} = 8$, which we found to work well across all datasets and network choices. All shape adaptors use the search space $\mathcal{R} = (0.5, 1)$ with the design in Fig. 2. We applied local type shape adaptors, to have a similar resizing effect from human-designed resizing layers, and with memory constraint on shape adaptors so that the network shape can grow no larger than the running GPU memory. We optimised shape adaptors every $\alpha_s = 20$ steps for non-ImageNet datasets, and every $\alpha_s = 1500$ steps for ImageNet. The full hyper-parameter choices are provided in the Appendix B.

4.2 Results on Image Classification Datasets

First, we compared networks built with shape adaptors to the original human-designed networks, to test whether shape adaptors can improve performances solely by finding a better network shape, without using any additional parameter space. To ensure fairness, all network weights in the human-designed and shape adaptor networks were optimised using the same hyper-parameters, optimiser, and scheduler.

Table 1 shows the test accuracies of shape adaptor and human-designed networks, with each accuracy averaged over three individual runs. We see that in nearly all cases, shape adaptor designed networks outperformed human-designed networks by a significant margin, despite both methods using exactly the same parameter space. We also see that performance of shape adaptor designed networks are stable, with a relatively low variance across different runs. This is similar to the human-designed networks, showing stability and robustness of our method without needing the domain knowledge that is required for human-designed networks. A detailed analysis on robustness and perturbation of shape adaptors compared to other resizing modules is further discussed in Appendix C.

Dataset	VGG-16		ResNet-50		MobileNetv2	
	Human	Shape Adaptor	Human	Shape Adaptor	Human	Shape Adaptor
CIFAR-10	$94.11_{\pm 0.17}$	$\mathbf{95.35}_{\pm \mathbf{0.06}}$	$\mathbf{95.50}_{\pm \mathbf{0.09}}$	$95.48_{\pm 0.17}$	$93.71_{\pm 0.25}$	$\mathbf{93.86}_{\pm \mathbf{0.23}}$
CIFAR-100	$75.39_{\pm 0.11}$	$\mathbf{79.16}_{\pm \mathbf{0.23}}$	$78.53_{\pm 0.11}$	$\mathbf{80.29}_{\pm \mathbf{0.10}}$	$73.80_{\pm 0.17}$	$\mathbf{75.74}_{\pm \mathbf{0.31}}$
SVHN	$96.26_{\pm 0.03}$	$\mathbf{96.89}_{\pm \mathbf{0.07}}$	$96.74_{\pm 0.20}$	$\mathbf{96.84}_{\pm \mathbf{0.13}}$	$96.50_{\pm 0.08}$	$\mathbf{96.86}_{\pm \mathbf{0.14}}$
Aircraft	$85.28_{\pm 0.09}$	$\mathbf{86.95}_{\pm \mathbf{0.29}}$	$81.57_{\pm 0.51}$	$\mathbf{85.60}_{\pm \mathbf{0.32}}$	$77.64_{\pm 0.23}$	$\mathbf{83.00}_{\pm \mathbf{0.30}}$
Birds	$73.37_{\pm 0.35}$	$\mathbf{74.86}_{\pm \mathbf{0.50}}$	$68.62_{\pm 0.10}$	$\mathbf{71.02}_{\pm \mathbf{0.48}}$	$60.37_{\pm 1.12}$	$\mathbf{68.53}_{\pm \mathbf{0.21}}$
Cars	$89.30_{\pm 0.21}$	$\mathbf{90.13}_{\pm \mathbf{0.11}}$	$87.23_{\pm 0.48}$	$\mathbf{89.67}_{\pm \mathbf{0.20}}$	$80.86_{\pm 0.13}$	$\mathbf{84.62}_{\pm \mathbf{0.38}}$
ImageNet	$\mathbf{73.92}_{\pm \mathbf{0.12}}$	$73.53_{\pm 0.09}$	$77.18_{\pm 0.04}$	$\mathbf{78.74}_{\pm \mathbf{0.12}}$	$71.72_{\pm 0.02}$	$\mathbf{73.32}_{\pm \mathbf{0.07}}$

Table 1: Top-1 test accuracies on different datasets for networks equipped with human-designed resizing layers and with shape adaptors. We present the results with the range of three independent runs. Best results are in bold.

Note that shape adaptors presented here are optimised purely to achieve an optimal performance, in a defined representation space, without considering the expense of memory consumption. However, we may also design memory-efficient shape adaptors for network compression which we will present in Section 5.1.

4.3 Ablative Analysis & Visualisations

In this section, we perform an ablative analysis on CIFAR-100 and Aircraft datasets to understand the behaviour of shape adaptors with respect to the number of shape adaptors, and shape adaptor initialisation. We observed that conclusions are consistent across different networks, thus we perform experiments in two networks only: VGG-16 and MobileNetv2. All results are averaged over two independent runs.

4.3.1 Number of Shape Adaptors

We first evaluate the performance by varying different number of shape adaptors used in the network, whilst fixing all other hyper-parameters used in Section 4.2. In Table 2, we show that the performance of shape adaptor networks is consistent across the number of shape adaptors used. Notably, performance is always better than networks with human-designed resizing layers, regardless of the number of shape adaptors used. This again shows

CIFAR-100	Human	Shape Adaptor (with number of)				
		3	4	5	6	8
VGG-16	75.39	79.03	**79.16**	78.56	78.43	78.16
MobileNetv2	73.80	75.39	**75.74**	75.22	74.92	74.86

Aircraft	Human	Shape Adaptor (with number of)				
		5	6	7	8	10
VGG-16	85.28	84.80	**86.95**	86.44	86.72	85.76
MobileNetv2	77.64	81.12	83.00	**83.02**	82.43	80.36

Table 2: Test accuracies of VGG-16 on CIFAR-100 and MobileNetv2 on Aircraft, when different numbers of shape adaptors are used. Best results are in bold. The number produced in Eq. 5 is highlighted in teal.

the ability of shape adaptors to automatically learn optimal shapes without requiring domain knowledge. The optimal number of shape adaptor modules given by our heuristic in Eq. 5 is highlighted in teal, and we can therefore see that this is a good approximation to the optimal number of modules.

In Figure 3, we present visualisations of network shapes in human-designed and shape adaptor designed networks. We can see that the network shapes designed by our shape adaptors are visually similar when different numbers of shape adaptor modules are used. In Aircraft dataset, we see a narrower shape with MobileNetv2 due to inserting an excessive number of 10 shape adaptors, which eventually converged to a local minima and lead to a degraded performance.

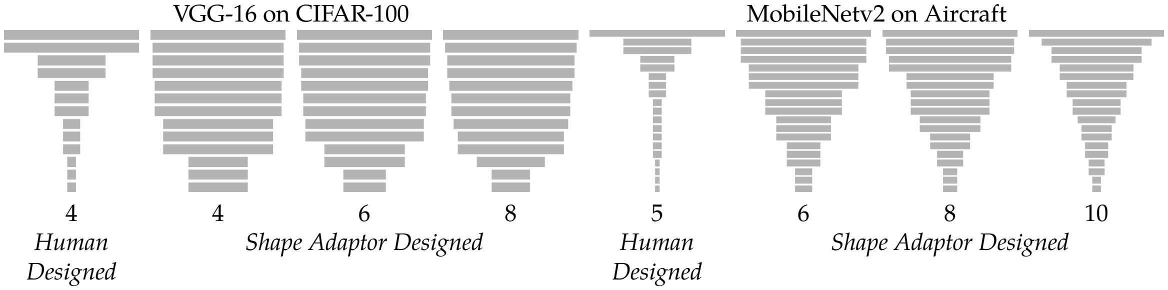


Figure 3: Visualisation of human-designed and shape adaptor designed network shapes. The number on the second row represents the number of resizing layers (or shape adaptors) applied in the network.

4.3.2 Initialisations in Shape Adaptors

Here, we evaluate the robustness of shape adaptors by varying initialisation of α. Initialisation with a “wide" shape (large α) causes high memory consumption and a longer training time, whereas initialisation with a “narrow" shape (small α) results in weaker gradient signals and a more likely convergence to a non-optimal local minima. In Table 3, we can see that the performance is again consistently better than the human-designed architecture, across all tested initialisations. The initialisation in shape adaptor modules given by our formula Eq. 7 is highlighted in teal.

In Figure 4, we present the learning dynamics for each shape adaptor module across the entire training stage. We can observe that shape adaptors are learning in an almost identical

CIFAR-100	Human	Shape Adaptor (with $s(\alpha)$ initialised)			
		0.60	0.70	0.80	0.90
VGG-16	75.39	**79.21**	79.16	78.79	78.53
MobileNetv2	73.80	75.16	**75.74**	74.89	74.74

Aircraft	Human	Shape Adaptor (with $s(\alpha)$ initialised)			
		0.52	0.58	0.62	0.68
VGG-16	85.28	83.90	**86.95**	86.36	86.54
MobileNetv2	77.64	79.64	**83.00**	82.51	81.56

Table 3: Test accuracies of VGG-16 on CIFAR-100 and MobileNetv2 on Aircraft datasets in shape adaptors with different initialisations. Best results are in bold. The initialisation produced in Eq. 7 is highlighted in teal.

pattern across different initialisations in the CIFAR-100 dataset, with nearly no variance. For the larger resolution Aircraft dataset, different initialised shape adaptors converged to a different local minimum. They still follow a general trend, for which the reshaping factor of a shape adaptor inserted in the deeper layers would converge into a smaller scale.

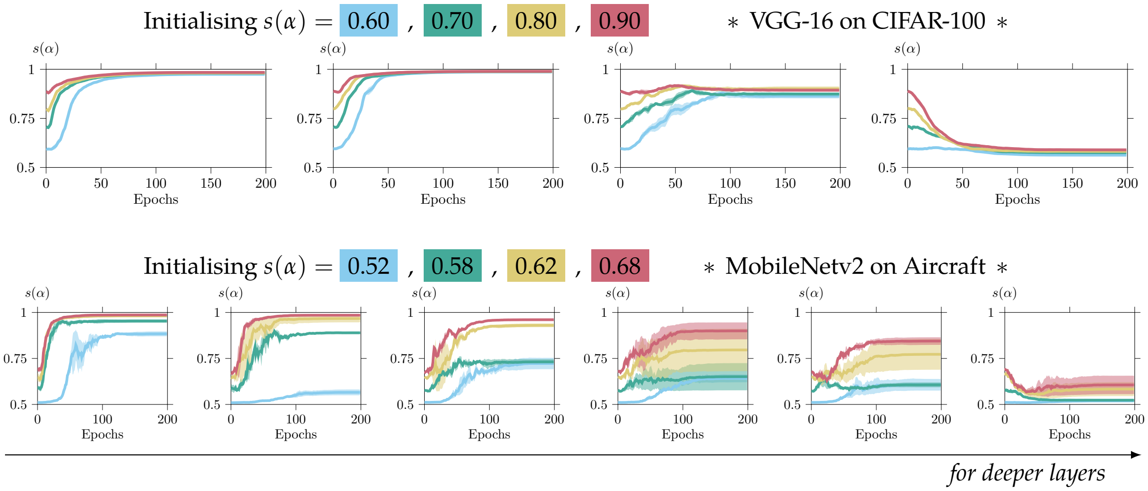


Figure 4: Visualisation of learning dynamics for every shape adaptor module across the entire training stage.

4.4 A Detailed Study on Neural Shape Learning

In this section, we propose a study to analyse different neural shape learning strategies, and the transferability of learned shapes. Likewise, all results are averaged over two independent runs.

We evaluate different neural shape learning strategies by running shape adaptors in three different versions. *Standard*: the standard implementation from previous sections; *Fix (Final)*: a network retrained with a fixed optimal shape obtained from shape adaptors; and *Fix (Large)*: a network retrained with a fixed largest possible shape in the current running GPU memory. The Fix (Final) baseline is designed to align with the training strategy from NAS algorithms [21, 43, 30]. The Fix (Large) baseline is to test whether naively increasing network computational cost can give improved performance.

CIFAR-100	Human	Shape Adaptors		
		Standard	Fix (Final)	Fix (Large)
VGG-16	75.39 314M	79.16 5.21G	78.62 5.21G	78.51 9.46G
MobileNetv2	73.80 94.7M	75.74 923M	75.54 923M	75.46 1.35G

Aircraft	Human	Shape Adaptors		
		Standard	Fix (Final)	Fix (Large)
VGG-16	85.28 15.4G	86.95 50.9G	86.27 50.9G	84.49 97.2G
MobileNetv2	77.64 326M	83.00 9.01G	82.26 9.01G	81.18 12.0G

Table 4: Test accuracies and computational cost (MACs, the number of multiply-adds) on CIFAR-100 and Aircraft datasets trained with different shape learning strategies.

In Table 4, we can observe that our standard version achieves the best performance among all shape learning strategies. In addition, we found that just having a large network would not guarantee an improved performance (VGG-16 on Aircraft). This validates that shape adaptors are truly learning the optimal shape, rather than naively increasing computational cost. Finally, we can see that our original shape learning strategy without re-training performs better than a NAS-like two-stage training strategy, which we assume is mainly due to dynamically updating of network shape helping to learn spatial-invariant features.

In order to further understand how network performance is correlated with different network shapes, we ran a large-scale experiment by training 200 VGG-16 networks with randomly generated shapes.

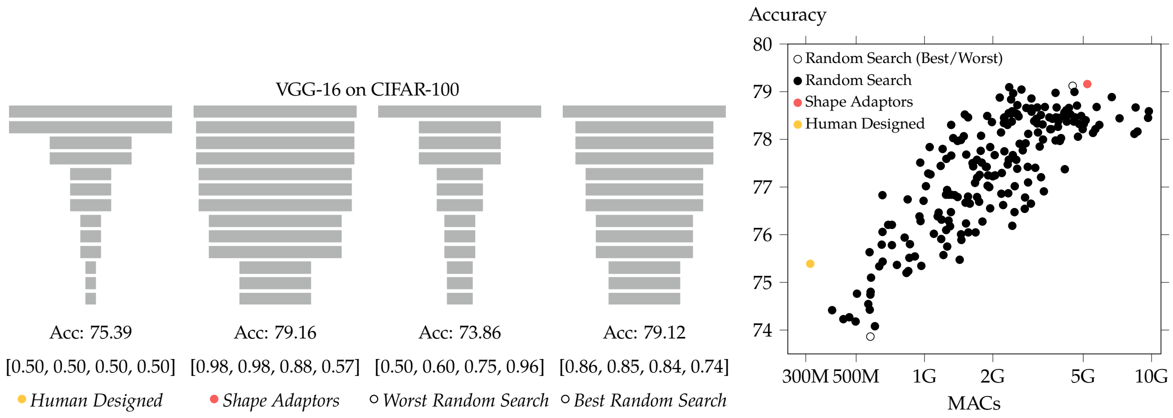


Figure 5: Visualisation and test accuracies of VGG-16 on CIFAR-100 in 200 randomly generated shapes. The second row represents the precise reshaping factor in each resizing layer

In Fig. 5, we visualise the randomly generated network shapes with the best and the worst performance, and compare them to the network shapes in human designed and shape adaptor networks.

First, we can see that the best randomly searched shape obtains a very similar performance as well as a similar structure of shape compared to the ones learned from shape adaptors. Second, the reshaping factors in the worst searched shape are arranged from small to large, which is the direct opposite trend to the reshaping factors automatically learned by our shape adaptors. Third, human-designed networks are typically under-sized, and just by increasing network memory cost is not able to guarantee an improved performance. Finally, we can see a clear correlation between memory cost and performance, where a higher memory cost typically increases performance. However, this correlation ceases after 5G of memory consumption, after which point we see no improved performance. Interestingly, the memory cost of shape adaptors lies just on the edge of this point, which again shows the shape adaptor's ability to learn optimal design.

5 Other Applications

In this section, we present two additional applications of shape adaptors: Automated Shape Compression (AutoSC) and Automated Transfer Learning (AutoTL).

5.1 Automated Shape Compression

In previous sections, we have shown that shape adaptors are able to improve performance by finding the optimal network shapes, but with a cost of a huge memory requirement of the learned network. In AutoSC, we show that shape adaptors can also achieve strong results, when automatically finding optimal memory-bounded network shapes based on an initial human design. Instead of the original implementation of shape adaptors where these are assumed to be the only resizing layers in the network, with AutoSC we attach down-sampling shape adaptors only on top of the non-resized layers of the human-designed architecture, whilst keeping the original human-designed resizing layers unchanged. Here, we insert global type shape adaptors, to be initialised so that the network shape is identical to the human-designed architecture, and thus the down-sampling shape adaptors can only learn to compress the network shape. This guarantees that the learned shape requires no more memory than the human-designed shape.

200/300M MobileNetv2	Params	MACs	Acc.
Human 0.75×	2.6M	233M	69.8
AutoSC 0.85×	2.9M	262M	70.7
Human 1.0×	3.5M	330M	71.8
AutoSC 1.1×	4.0M	324M	72.3

(a) Results on ImageNet

Plain MobileNetv2	Params	MACs	Acc.
Human 1.0×	2.3M	94.7M	73.80
AutoSC 1.0×	2.3M	91.5M	74.81
Human 1.0×	2.3M	330M	77.64
AutoSC 1.0×	2.3M	326M	78.95

(b) Results on CIFAR-100 (up) and Aircraft (down)

Table 5: Test accuracies for AutoSC and human-designed MobileNetv2 on CIFAR-100, Aircraft, and ImageNet. × represents the applied width multiplier.

In Table 5, we present AutoSC built on MobileNetv2, an efficient network design for mobile applications. We evaluate AutoSC on three datasets: CIFAR-100, Aircraft and ImageNet. During training of MobileNetv2, we initialised a small width multiplier on the network's channel dimension to slightly increase the parameter space (if applicable). By doing this, we ensure that this “wider" network after compression would have a similar memory consumption as the human-designed MobileNetv2, for a fair comparison. In all three datasets, we can observe that shape adaptors are able to improve performance, despite having similar memory consumption compared to human-designed networks.

5.2 Automated Transfer Learning

In this section, we present how shape adaptors can be used to perform transfer learning in an architectural level. In AutoTL, we directly replace the human-designed resizing layers with shape adaptors, and initialise them with the reshaping factors designed in the original human-defined architecture, to match the spatial dimension of each pre-trained network layer. During fine-tuning, the network is then fine-tuning with network weights along with network shapes, thus improving upon the standard fine-tuning in a more flexible manner.

We follow the same setting as in PackNet [26] and Piggyback [26], evaluating on 5 fine-grained classification datasets across very different domains. For all tasks, we use input images of resolution [224 × 224], and evaluate them on an ImageNet pre-trained ResNet-50.

	Birds [36]	Cars [15]	Flowers [29]	WikiArt [33]	Sketches [6]
PackNet [27]	80.41	86.11	93.04	69.40	76.17
PiggyBack [26]	81.59	89.62	94.77	71.33	79.91
NetTailor [28]	82.52	90.56	95.79	72.98	80.48
Fine-tune [8]	81.86	89.74	93.67	75.60	79.58
SpotTune [8]	84.03	92.40	**96.34**	75.77	80.20
AutoTL	**84.29**	**93.66**	96.22	**77.47**	**80.74**

Table 6: Test accuracies of transfer Learning methods built on ResNet-50 on fine-grained datasets. Best results are in bold.

The results for AutoTL and other state-of-the-art transfer learning methods are listed in Table 6, for which we outperform 4 out of 5 datasets. The most related methods to our approach are standard fine-tuning and SpotTune [8], which optimise the entire network parameters for each dataset. Other approaches like PackNet [27], Piggyback [26], and NetTailor [28] focus on efficient transfer learning by updating few task-specific weights. We design AutoTL with standard fine-tuning, as the simplest setting to show the effectiveness of shape adaptors. In practice, AutoTL can be further improved, and integrated into other efficient transfer learning techniques.

6 Conclusions & Future Directions

In this paper, we present shape adaptor, a learnable resizing module to enhance existing neural networks with task-specific network shapes. With shape adaptors, the learned network shapes can further improve performances compared to human-designed architectures, without requiring in increase in parameter space. We show that shape adaptors are robust to hyper-parameters, and typically learn very similar network shapes, regardless of the number of shape adaptor modules used. In addition, we show that shape adaptors can also be easily incorporated into other applications, such as network compression and transfer learning.

In future work, we will investigate shape adaptors in a multi-branch design, where the formulation provided in this paper extended to integrating more than two resizing layers in each shape adaptor module. Due to the success of shape adaptors in the other applications we have presented in this paper, we will also study use of shape adaptors for more applications, such as neural architecture search, and multi-task learning.

Acknowledgements

We would like to thank the helpful discussions from Linxi Fan and Bingyu Zhang.

References

[1] Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

[2] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[4] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

[6] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on graphics (TOG)*, 31(4):1–10, 2012.

[7] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[8] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4805–4814, 2019.

[9] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[11] 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.

[12] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

[13] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Automated machine learning-methods, systems, challenges.

[14] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[15] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[16] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[17] Jason Kuen, Xiangfei Kong, Zhe Lin, Gang Wang, Jianxiong Yin, Simon See, and Yap-Peng Tan. Stochastic downsampling for cost-adjustable inference and improved regularization in convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7929–7938, 2018.

[18] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial intelligence and statistics*, pages 464–472, 2016.

[19] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[22] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[23] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[24] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

[25] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[26] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[27] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[28] Pedro Morgado and Nuno Vasconcelos. Nettailor: Tuning the architecture, not just the weights. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3044–3054, 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] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, StockholmsmÃďssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[31] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[33] Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *International Journal for Digital Art History*, Oct. 2016.

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[36] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[37] 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.

[38] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. Mixed pooling for convolutional neural networks. In *International conference on rough sets and knowledge technology*, pages 364–375. Springer, 2014.

[39] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020.

[40] Matthew Zeiler and Robert Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In *Proceedings of the International Conference on Learning Representation*, 2013.

[41] Richard Zhang. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334, 2019.

[42] Yichen Zhu, Xiangyu Zhang, Tong Yang, and Jian Sun. Resizable neural networks, 2020.

[43] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A General Formation of Shape Adaptors

Shape adaptors can be extended into a multi-branch design, into a more general manner. Each shape adaptor module is then composed of $K \geq 2$ resizing layers $F_{i=1:K}$, with fixed reshaping factors $r_{i=1:K} > 0$, and the corresponding learnable scaling weight parameters $\alpha_{i=1:K} \in (0,1)$. In such design, shape adaptor modules can not only learn the optimal network shape, but also the optimal operations contributing to the learned shape.

We first define a set of reshaping factors r_i, and scaling weights α_i in resizing layers F_i:

$$\boldsymbol{r} = \{r_{i=1:K} \,|\, r_i > 0, \, \exists m, n : r_m \neq r_n\}, \quad \text{and} \quad \boldsymbol{\alpha} = \left\{ \alpha_{i=1:K} \,\middle|\, \sum_{i=1}^{K} \alpha_i = 1, \, \alpha_i > 0 \right\}. \tag{11}$$

Then, we design the module's reshaping factor which is mapped from scaling weights $\boldsymbol{\alpha} \to s(\boldsymbol{\alpha})$, lying in the defined search space interval $\mathcal{R} = (\min(\boldsymbol{r}), \, \max(\boldsymbol{r}))$.

The general system for a shape adaptor module is formulated as follows,

$$\texttt{ShapeAdaptor}(x, \boldsymbol{\alpha}, \mathbf{r}) = \sum_{i=1}^{K} \alpha_i \cdot G\left(F_i\left(x, r_i\right), \, \frac{s(\boldsymbol{\alpha})}{r_i}\right), \tag{12}$$

with $s(\boldsymbol{\alpha})$ satisfies

$$s(\boldsymbol{\alpha})_{\alpha_k \to 1} = r_k, \quad \text{and} \quad s(\boldsymbol{\alpha}) \mapsto \mathcal{R}. \tag{13}$$

The weighted generalised mean:

$$s_0(\boldsymbol{\alpha}) = \prod_{i=1}^{K} r_i^{\alpha_i}, \quad \text{and} \quad s_p(\boldsymbol{\alpha}) = \left(\sum_{i=1}^{K} \alpha_i r_i^p\right)^{1/p}, \, p \neq 0 \tag{14}$$

are examples of suitable reshaping function design.

The general design for multi-branch shape adaptors can be inserted into more complicated networks architectures, such as ResNeXt [37] and Xception [4]. It can also be seen as a direct enhancement to spatial pyramid pooling [10, 3], and U-Net [32], to enable them to propagate context information from various, rather than the same, feature dimensions.

B The Complete Hyper-Parameter Table

In this section, for reproducibility, we present a detailed list of hyper-parameter choices, across all networks and datasets evaluated in Table 1. A represents network shape parameters in shape adaptors, and W represents network weight parameters.

	Small Datasets: $[32 \times 32]$			Fine-Grained Datasets: $[224 \times 224]$			ImageNet: $[224 \times 224]$		
	VGG-16	ResNet-50	MobileNetv2	VGG-16	ResNet-50	MobileNetv2	VGG-16	ResNet-50	MobileNetv2
A - Learning Rate	0.1			0.1			0.1		
A - Optimiser	SGD with 0.9 momentum			SGD with 0.9 momentum			SGD with 0.9 momentum		
A - Scheduler	Cosine Annealing			Cosine Annealing			Cosine Annealing		
A - Update Step	20			20			1500		
A - Number	Eq. 2: $\log_2(D^{in}/2)$ (4 for $[32 \times 32]$ images, 6 for $[224 \times 224]$ images)								
A - Initialisation	Eq. 4 with $D^{out} = 8$								
A - Location	Uniformly distributed (across all layers except for the last layer)								
A - Search Space	$(0.5, 1.0)$ (for every shape adaptor module)								
W - Learning Rate	0.1			0.01			0.1	0.1	0.05
W - Optimiser	SGD with 0.9 momentum			SGD with 0.9 momentum			SGD with 0.9 momentum		
W - Weight Decay	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$4 \cdot 10^{-5}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$4 \cdot 10^{-5}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$4 \cdot 10^{-5}$
W - Scheduler	Cosine Annealing			Cosine Annealing			Cosine Annealing		
Batch Size	128			8			32 (per GPU) for 8 GPUs		
Epochs	200			200			120		

Table 7: The complete hyper-parameter applied to reproduce Table 1.

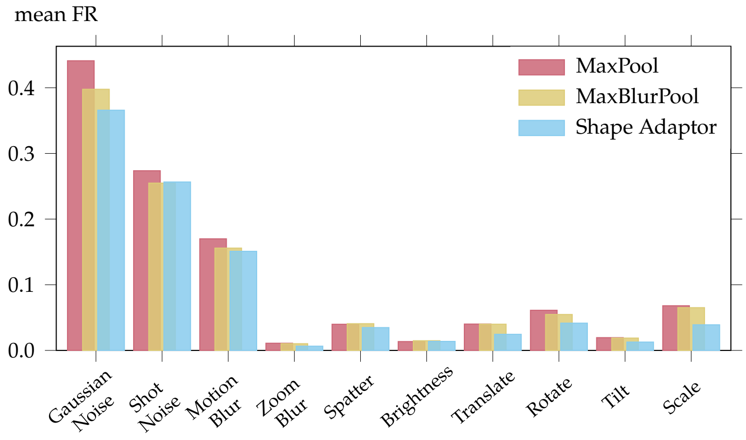
C Corruptions and Perturbations Analysis

In this section, we evaluate the model robustness and uncertainty estimates in networks equipped with shape adaptors, compared with other types of resizing modules.

Metrics We apply two metrics with respect to corruption and perturbation robustness evaluation respectively, introduced in [12]. For corruption analysis, we evaluate with *mean Corruption Error (mean CE)*, which computes an average classification error on a corrupted dataset, composed by corrupting the original dataset with 15 corruption types, and each with additional 5 severity levels. For perturbation analysis, each data in a perturbed dataset becomes a video, to measure prediction stability. We then evaluate by measuring whether video frames prediction match, which we call flip probability. We evaluate with 10 perturbation types, and the mean across these is *mean Flip Rate (mean FR)*.

We apply corruption analysis on CIFAR-10 and CIFAR-100 dataset, which give the corrupted CIFAR-10-C and CIFAR-100-C datasets respectively. We apply perturbation analysis on CIFAR-10 only, for perturbed CIFAR-10-P dataset, with the highest difficulty level 3. All analyses are performed based on VGG-16, and we compare corruption and perturbation robustness for MaxPool (used in original human-designed networks), MaxBlurPool [41] (an anti-aliasing MaxPool), and shape adaptors. All models are trained on the clean dataset.

In Figure 6 Right, we can observe that shape adaptors equipped VGG-16 perform the best among all tested resizing modules by a large margin, in both corrupted and the clean datasets. In Figure 6 Left, we show that shape adaptors are able to improve almost every type of perturbation compared to the results from both MaxPool and the improved MaxBlurPool modules. This is most prominent in *digital type* perturbations (translate, rotate, tilt, scale), which provides approximately 40% of performance improvements compared to MaxPool. These positive results show that shape adaptors not only can improve human-designed networks in accuracy, but also in robustness, by learning spatial-invariant features.



	MaxPool (Vanilla)	Max BlurPool	Shape Adaptors
CIFAR-10 (Clean Err.)	5.89	5.86	**4.65**
CIFAR-10-C (mean CE)	26.78	25.61	**23.02**
CIFAR-100 (Clean Err.)	24.61	25.04	**20.84**
CIFAR-100-C (mean CE)	51.45	51.43	**48.42**

Figure 6: Left: mean flip rate of CIFAR-10-P dataset with difficulty level 3. Right: clean error on CIFAR-10 dataset and mean corruption rate on CIFAR-10-C dataset. All results are trained with VGG-16 on the clean dataset.

D Negative Results

- **Other choices in shape adaptor search space.** We experimented with shape adaptors in the search space $\mathcal{R} = (0.25, 1)$, which we found to converge to a similar overall network shape, but with degraded performance compared to the current setting. We also experimented with shape adaptors using the search space in $\mathcal{R} = (0.5, 2.0)$, which we found to have very unstable learning dynamics, and often with out of memory issues.
- **Other choices in reshaping function design.** We evaluated shape adaptors with its reshaping factor $s(\alpha) = \frac{1}{\alpha/r_1 + (1-\alpha)/r_2}$, a weighted harmonic mean, which we found to have no improvements compared to the current setting.
- **Other optimisation methods.** We experimented with updating network shape parameters and weight parameters based on a different sample in the training dataset, which we found to have a degraded performance compared to the current setting.
- **Learning shape with prior structure knowledge.** We have experimented with directly replacing human-designed resizing layers with shape adaptors, which we found to have a minor effect on final performance compared to the current setting.
- **Alternative shape adaptor design in a residual cell.** We have experimented with an alternate design of the residual cell, with a $[1 \times 1]$ convolution layer as the identity branch, and with the weight layer as the resizing branch. The final performance with such a design achieved worse performance compared to the design defined in Fig. 2.