

Shape Adaptor: A Learnable Resizing Module

Shikun Liu¹, Zhe Lin², Yilin Wang², Jianming Zhang², Federico Perazzi², and Edward Johns¹

¹ Department of Computing, Imperial College London

² Adobe Research

{shikun.liu17,e.johns}@imperial.ac.uk
{zlin,yilwang,jianmzha,perazzi}@adobe.com

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.

Keywords: automated machine learning, resizing layer, neural architecture search

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.

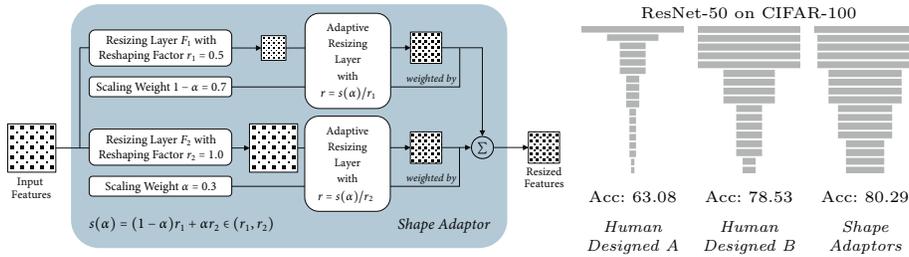


Fig. 1: Left: Visualisation of a shape adaptor module build on top of two resizing layers. Right: Different network shapes in the same network architecture ResNet-50 can result a significantly different performance.

To move beyond the limitations of human-designed network architectures, there has been a growing interest in developing Automated Machine Learning (AutoML) algorithms [9] for automatic architecture design, known as Neural Architecture Search (NAS) [26,2,17,16]. 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 [8], 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 . 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 $s(\alpha) \in (r_1, r_2)$, assuming $r_1 < r_2$. 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.

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 [37] and evolutionary search [27]. Several approaches have been proposed to speed up the search, based on parameter sharing [26], hyper-networks [1], and gradient-based optimisation [17]. But despite their promising performance, these approaches come with controversial debate questioning the lack of reproducibility, and sensitivity to initialisations [15,33].

Architecture Pruning & Compression Network pruning is another direction towards achieving optimal network architectures. But instead of searching from scratch as in NAS, network pruning is applied to existing human-designed networks and removes redundant neurons and connectivities. Such methods can be based on \mathcal{L}_0 regularisation [19], batch-norm scaling parameters [18], and weight quantization [7]. As with our shape adaptors, network pruning does not require the extensive search cost of NAS, and can be 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 [32,14], and with anti-aliased low-pass filters [35]. Other works impose regularisation and adjustable inference by stochastically inserting pooling layers [34,13], and sampling different network shapes [36]. 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 Formulation 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 learnable 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 a function:

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

with reshaping factor $s(\alpha)$, a monotonic mapping which satisfies:

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

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.

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

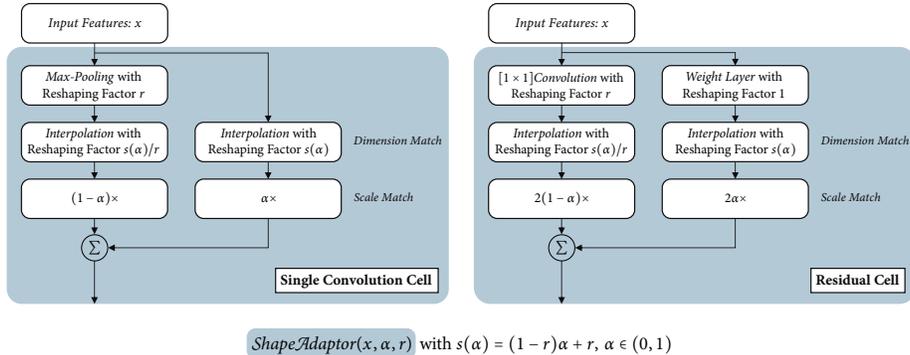


Fig. 2: Visualisation of a down-sampling shape adaptor built on a single convolutional cell and a residual cell with a reshaping factor in the range $\mathcal{R} = (r, 1)$.

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 [10,20], to control the softness. This technique is commonly used in gradient-based NAS methods [17], 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 [30], and a residual cell in ResNet-like neural networks [8]. 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.

3.2 The Optimisation Recipe

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:

$$N = \left\lceil \log_{1/r_{min}} (\mathcal{D}^{in} / \mathcal{D}^{last}) \right\rceil \quad (3)$$

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:

$$\mathcal{D}^{in} \cdot s(\sigma(\bar{\alpha}))^N = \mathcal{D}^{out}. \quad (4)$$

Suppose we use N as defined in Eq. 3, then Eq. 4 is only solvable when $\mathcal{D}^{last} \leq \mathcal{D}^{out}$. Otherwise, we then initialise the smallest possible shape when encountering the case for $\mathcal{D}^{last} > \mathcal{D}^{out}$. This eventually derives the following:

$$\bar{\alpha} = \begin{cases} \ln \left(-\frac{N \sqrt[\mathcal{D}^{out}/\mathcal{D}^{in}]{r_{min}}}{N \sqrt[\mathcal{D}^{out}/\mathcal{D}^{in}]{r_{max}} - r_{min}} + \epsilon \right) & \text{if } \mathcal{D}^{last} \leq \mathcal{D}^{out} \\ \ln(\epsilon) & \text{otherwise} \end{cases}, \quad \epsilon = 10^{-4}. \quad (5)$$

where ϵ is a small value to avoid encountering $\pm\infty$ values.

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,

$$\alpha_\rho = \alpha \cdot \rho + \frac{r_{min}}{r_{max} - r_{min}}(\rho - 1). \quad (6)$$

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

$$s(\alpha_\rho) = (r_{max} - r_{min})\alpha_\rho + r_{min} = s(\alpha)\rho. \quad (7)$$

Using Eq. 4, we can compute ρ as,

$$\rho = \sqrt[N]{\frac{D^{limit}}{D^{cout}}}. \quad (8)$$

Iterative Optimisation Strategy To optimise a neural network equipped with shape adaptor modules, there are two sets of parameters to learn: the weight parameters $\mathbf{w} = \{w_i\}$, 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.

4 Experiments

In this section, we present experimental results to evaluate shape adaptors on image classification tasks.

4.1 Experimental Setup

Datasets We evaluated 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 [12], SVHN [5]; ii) fine-grained classification datasets: FGVC-Aircraft (Aircraft) [21], CUBS-200-2011 (Birds) [31], Stanford

Algorithm 1: Optimisation for Shape Adaptor Networks

```

1 Define: shape adaptors:  $\alpha_s, r_{min}, r_{max}, D^{last}, D^{out}, D^{limit}$ 
2 Define: network architecture  $f_{\alpha,w}$  defined with shape and network
  parameters
3 Initialise: shape parameters:  $\alpha = \{\alpha_i\}$  with Eq. 3, and Eq. 5
4 Initialise: weight parameters:  $w = \{w_i\}$ 
5 Initialise: learning rate:  $\lambda_1, \lambda_2$ 
6 while not converged do
7   for each training iteration  $i$  do
8      $(x_{(i)}, y_{(i)}) \in (x, y)$   $\triangleright$  fetch one batch of training data
9     if requires memory constraint then
10      | Compute:  $\rho$  using Eq. 8
11    else
12      | Define:  $\rho = 1$ 
13    end
14    if in  $\alpha_s$  step then
15      | Update:  $\alpha \leftarrow \lambda_1 \nabla_{\alpha} \mathcal{L}(f_{\alpha,\rho,w}(x_{(i)}, y_{(i)}))$   $\triangleright$  update shape parameters
16    end
17    Update:  $w \leftarrow \lambda_2 \nabla_w \mathcal{L}(f_{\alpha,\rho,w}(x_{(i)}, y_{(i)}))$   $\triangleright$  update weight parameters
18  end
19 end

```

Cars (Cars) [11]; and iii) ImageNet [3]. Small datasets are in resolution $[32 \times 32]$, and fine-grained classification and ImageNet datasets are in resolution $[224 \times 224]$.

Baselines We ran experiments with three widely-used networks: VGG-16 [30], ResNet-50 [8], and MobileNetv2 [29]. 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 $[32 \times 32]$ datasets, human-designed VGG-16, ResNet-50 and MobileNetv2 networks were equipped with 4, 3, 3 resizing layers respectively, and for $[224 \times 224]$ 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. (Other choices of the search space are discussed in the supplementary material.) We applied the 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 supplementary material.

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.

| Dataset | VGG-16 | | ResNet-50 | | MobileNetv2 | |
|-----------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|------------------|----------------------------------|
| | Human | Shape Adaptor | Human | Shape Adaptor | Human | Shape Adaptor |
| CIFAR-10 | 94.11 \pm 0.17 | 95.35\pm0.06 | 95.50\pm0.09 | 95.48 \pm 0.17 | 93.71 \pm 0.25 | 93.86\pm0.23 |
| CIFAR-100 | 75.39 \pm 0.11 | 79.16\pm0.23 | 78.53 \pm 0.11 | 80.29\pm0.10 | 73.80 \pm 0.17 | 75.74\pm0.31 |
| SVHN | 96.26 \pm 0.03 | 96.89\pm0.07 | 96.74 \pm 0.20 | 96.84\pm0.13 | 96.50 \pm 0.08 | 96.86\pm0.14 |
| Aircraft | 85.28 \pm 0.09 | 86.95\pm0.29 | 81.57 \pm 0.51 | 85.60\pm0.32 | 77.64 \pm 0.23 | 83.00\pm0.30 |
| Birds | 73.37 \pm 0.35 | 74.86\pm0.50 | 68.62 \pm 0.10 | 71.02\pm0.48 | 60.37 \pm 1.12 | 68.53\pm0.21 |
| Cars | 89.30 \pm 0.21 | 90.13\pm0.11 | 87.23 \pm 0.48 | 89.67\pm0.20 | 80.86 \pm 0.13 | 84.62\pm0.38 |
| ImageNet | 73.92\pm0.12 | 73.53 \pm 0.09 | 77.18 \pm 0.04 | 78.74\pm0.12 | 71.72 \pm 0.02 | 73.32\pm0.07 |

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.

4.3 Ablative Analysis & Visualisations

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

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

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. 3 is highlighted in teal, and we can therefore see that this is a good approximation to the optimal number of modules.

| CIFAR-100 | Human | SA (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 | SA (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. 3 is highlighted in teal.

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.

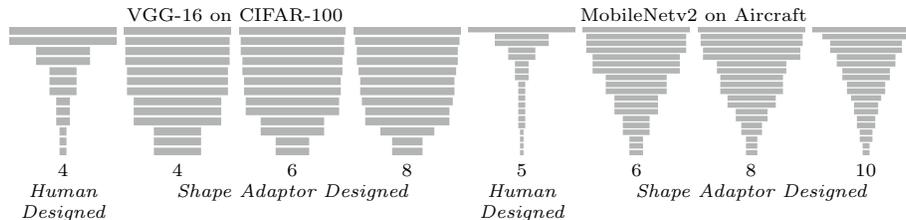


Fig. 3: Visualisation of human-designed and shape adaptor designed network shapes. The number on the second row represents the number of resizing layers (or shape adaptors) applied in the network.

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.

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

| CIFAR-100 | Human | SA (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 | SA (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. 5 is highlighted in teal.

in an almost identical 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.

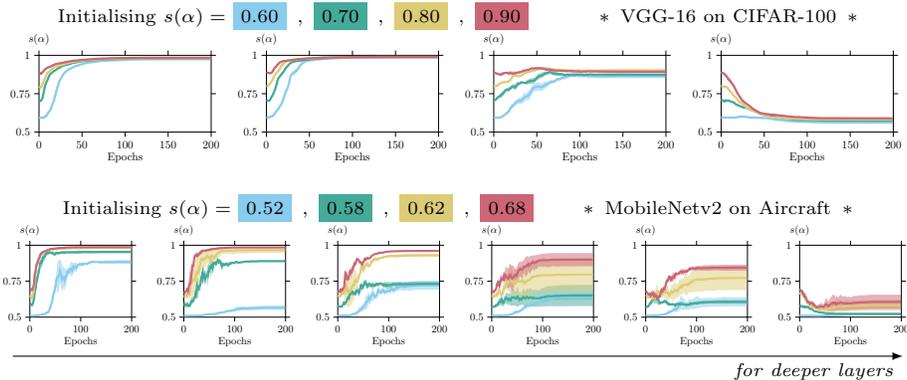


Fig. 4: Visualisation of learning dynamics for every shape adaptor module across the entire training stage.

4.4 A Detailed Study on Neural Shape Learning

In this section, we propose a study to analyse different 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 [17,37,26]. The

Fix (Large) baseline is to test whether naively increasing network computational cost can give improved performance.

| CIFAR-100 | Human | Shape Adaptors | | | Aircraft | Human | Shape Adaptors | | |
|-------------|----------------|----------------|----------------|----------------|-------------|----------------|----------------|----------------|----------------|
| | | Standard | Fix (Final) | Fix (Large) | | | Standard | Fix (Final) | Fix (Large) |
| VGG-16 | 75.39 314M | 79.16 5.21G | 78.62 5.21G | 78.51 9.46G | VGG-16 | 85.28 15.4G | 86.95 50.9G | 86.27 50.9G | 84.49 97.2G |
| MobileNetv2 | 73.80 94.7M | 75.74 923M | 75.54 923M | 75.46 1.35G | MobileNetv2 | 77.64 326M | 83.00 9.01G | 82.26 9.01G | 81.18 12.0G |

Table 4: Test accuracies and 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.

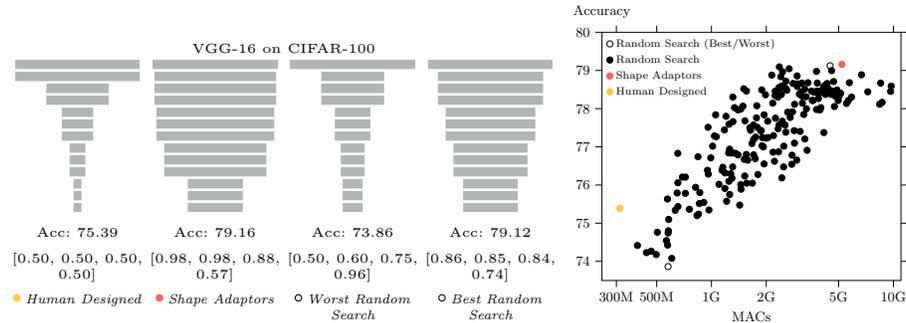


Fig. 5: Visualisation and test accuracies of VGG-16 on CIFAR-100 in 200 randomly generated shapes. The second row represents the precise reshaping factor in each resizing layer.

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

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. We initialise shape adaptors 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.

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 original human-designed resizing layers with shape adaptors, and initialise them with the reshaping factors designed in the original human-defined architecture, to match

| 200/300M | MobileNetv2 | Params | MACs | Acc. | Plain | MobileNetv2 | Params | MACs | Acc. |
|-----------------|-------------|--------|------|------|--------------|-------------|--------|-------|-------|
| Human | 0.75× | 2.6M | 233M | 69.8 | Human | 1.0× | 2.3M | 94.7M | 73.80 |
| AutoSC | 0.85× | 2.9M | 262M | 70.7 | AutoSC | 1.0× | 2.3M | 91.5M | 74.81 |
| Human | 1.0× | 3.5M | 330M | 71.8 | Human | 1.0× | 2.3M | 330M | 77.64 |
| AutoSC | 1.1× | 4.0M | 324M | 72.3 | AutoSC | 1.0× | 2.3M | 326M | 78.95 |

(a) Results on ImageNet

(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.

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.

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 [6], which optimise the entire network parameters for each dataset. Other approaches like PackNet [23], Piggyback [22], and NetTailor

| | Birds [31] | Cars [11] | Flowers [25] | WikiArt [28] | Sketch [4] |
|----------------|---------------|--------------|-----------------|-----------------|---------------|
| PackNet [23] | 80.41 | 86.11 | 93.04 | 69.40 | 76.17 |
| PiggyBack [22] | 81.59 | 89.62 | 94.77 | 71.33 | 79.91 |
| NetTailor [24] | 82.52 | 90.56 | 95.79 | 72.98 | 80.48 |
| fine-tune [6] | 81.86 | 89.74 | 93.67 | 75.60 | 79.58 |
| SpotTune [6] | 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.

[24] 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 practise, AutoTL can be further improved, and integrated into other efficient transfer learning techniques.

6 Conclusions & Future Directions

In this paper, we presented *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 an increase in parameter space. In future work, we will investigate improving shape adaptors in a multi-branch design, where the formulation provided in this paper can be extended to integrating more than two resizing layers in each shape adaptor module. In addition, we will also study use of shape adaptors for more applications, such as neural architecture search, and multi-task learning.

References

1. Brock, A., Lim, T., Ritchie, J., Weston, N.: SMASH: One-shot model architecture search through hypernetworks. In: International Conference on Learning Representations (2018), <https://openreview.net/forum?id=rydeCEhs->
2. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (2019), <https://openreview.net/forum?id=HylVB3AqYm>
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. IEEE (2009)
4. Eitz, M., Hays, J., Alexa, M.: How do humans sketch objects? *ACM Transactions on graphics (TOG)* **31**(4), 1–10 (2012)
5. Goodfellow, I.J., Bulatov, Y., Ibarz, J., Arnoud, S., Shet, V.: Multi-digit number recognition from street view imagery using deep convolutional neural networks. arXiv preprint arXiv:1312.6082 (2013)
6. Guo, Y., Shi, H., Kumar, A., Grauman, K., Rosing, T., Feris, R.: Spottune: transfer learning through adaptive fine-tuning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4805–4814 (2019)
7. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (2016)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
9. Hutter, F., Kotthoff, L., Vanschoren, J.: Automated machine learning-methods, systems, challenges
10. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. In: International Conference on Learning Representations (2017)
11. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3d object representations for fine-grained categorization. In: 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13). Sydney, Australia (2013)
12. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
13. Kuen, J., Kong, X., Lin, Z., Wang, G., Yin, J., See, S., Tan, Y.P.: Stochastic downsampling for cost-adjustable inference and improved regularization in convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7929–7938 (2018)
14. Lee, C.Y., Gallagher, P.W., Tu, Z.: Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In: Artificial intelligence and statistics. pp. 464–472 (2016)
15. Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search. arXiv preprint arXiv:1902.07638 (2019)
16. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)
17. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. In: International Conference on Learning Representations (2019), <https://openreview.net/forum?id=S1eYHoC5FX>
18. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2736–2744 (2017)

19. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. In: International Conference on Learning Representations (2018), <https://openreview.net/forum?id=H1Y8hhg0b>
20. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. In: International Conference on Learning Representations (2017)
21. Maji, S., Kannala, J., Rahtu, E., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. Tech. rep. (2013)
22. Mallya, A., Davis, D., Lazebnik, S.: Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 67–82 (2018)
23. Mallya, A., Lazebnik, S.: Packnet: Adding multiple tasks to a single network by iterative pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7765–7773 (2018)
24. Morgado, P., Vasconcelos, N.: Nettare: Tuning the architecture, not just the weights. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3044–3054 (2019)
25. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing. pp. 722–729. IEEE (2008)
26. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 4095–4104. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018), <http://proceedings.mlr.press/v80/pham18a.html>
27. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI conference on artificial intelligence. vol. 33, pp. 4780–4789 (2019)
28. Saleh, B., Elgammal, A.: Large-scale classification of fine-art paintings: Learning the right metric on the right feature. International Journal for Digital Art History (Oct 2016)
29. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)
30. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
31. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The Caltech-UCSD Birds-200-2011 Dataset. Tech. Rep. CNS-TR-2011-001, California Institute of Technology (2011)
32. Yu, D., Wang, H., Chen, P., Wei, Z.: Mixed pooling for convolutional neural networks. In: International conference on rough sets and knowledge technology. pp. 364–375. Springer (2014)
33. Yu, K., Sciuto, C., Jaggi, M., Musat, C., Salzmann, M.: Evaluating the search phase of neural architecture search. In: International Conference on Learning Representations (2020), <https://openreview.net/forum?id=H1loF2NFwr>
34. Zeiler, M., Fergus, R.: Stochastic pooling for regularization of deep convolutional neural networks. In: Proceedings of the International Conference on Learning Representation (2013)
35. Zhang, R.: Making convolutional networks shift-invariant again. In: International Conference on Machine Learning. pp. 7324–7334 (2019)

36. Zhu, Y., Zhang, X., Yang, T., Sun, J.: Resizable neural networks (2020), https://openreview.net/forum?id=BJe_z1HFPr
37. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (2017)