NetTailor: Tuning the architecture, not just the weights

Pedro Morgado [*]   Nuno Vasconcelos
Department of Electrical and Computer Engineering
University of California, San Diego
{pmaravil,nuno}@ucsd.edu

Abstract

Real-world applications of object recognition often require the solution of multiple tasks in a single platform. Under the standard paradigm of network fine-tuning, an entirely new CNN is learned per task, and the final network size is independent of task complexity. This is wasteful, since simple tasks require smaller networks than more complex tasks, and limits the number of tasks that can be solved simultaneously. To address these problems, we propose a transfer learning procedure, denoted NetTailor[1], *in which layers of a pre-trained CNN are used as universal blocks that can be combined with small task-specific layers to generate new networks. Besides minimizing classification error, the new network is trained to mimic the internal activations of a strong unconstrained CNN, and minimize its complexity by the combination of 1) a soft-attention mechanism over blocks and 2) complexity regularization constraints. In this way,* NetTailor *can adapt the network architecture, not just its weights, to the target task. Experiments show that networks adapted to simple tasks, such as character or traffic sign recognition, become significantly smaller than those adapted to hard tasks, such as fine-grained recognition. More importantly, due to the modular nature of the procedure, this reduction in network complexity is achieved without compromise of either parameter sharing across tasks, or classification accuracy.*

1. Introduction

Real-world applications of machine learning for vision often involve the ability to solve *multiple* recognition tasks. For example, a robot should be able to decide if a door is open or closed, whether an object can be picked up or not, what is the expression on a person's face, among others. However, attempting to design a single recognizer for all tasks is often impractical, since datasets for different tasks are not always available at the same time, and state-of-the-art models use

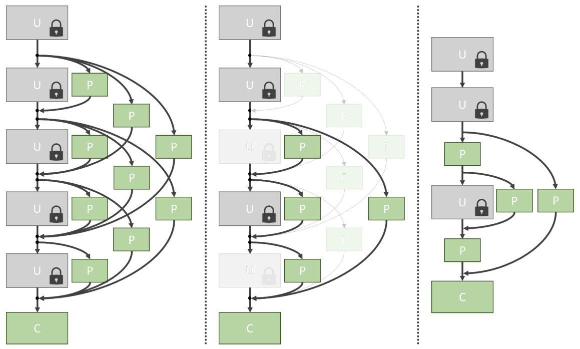


Figure 1: Architecture fine-tuning with NetTailor. Pre-trained blocks shown in gray, task-specific in green. Left: Pre-trained CNN augmented with low-complexity blocks that introduce skip connections. Center: Optimization prunes blocks of poor trade-off between complexity and impact on recognition. Right: The final network is a combination of pre-trained and task-specific blocks.

different training procedures (e.g. face recognition is solved through an embedding approach [57], while object recognition uses a classification loss [20]). Instead, the standard solution is to use an off-the-shelf convolutional neural network (CNN) pre-trained on a large dataset such as ImageNet [9], MS-COCO [33] or MIT-Places [73], and fine-tune it to each task [70, 27, 15].

Although fine-tuning often achieve good performance on the target task [70], this practice is quite wasteful. First, fine-tuning produces a large network per task, *independently* of the task complexity. Hence, computing and storage requirements increase linearly with the number of tasks, with simpler tasks like optical character recognition (OCR) being as demanding as hard tasks like fine-grained recognition. Second, although the resulting networks are derived from a common pre-trained model, they differ in *all* their parameters. Hence, as the robot switches between tasks, large arrays of parameters need to be reloaded, which may hinder operation in real-time. Other smart computing platforms, such as consumer electronics devices, mobile devices or smart cars, also face similar problems.

In this work, we seek a solution to these problems. Layers of a large pre-trained neural network are viewed as universal blocks that can be combined to generate new networks. The universal blocks implement universal filters shared by all tasks. They are

[*]This work was partially funded by graduate fellowship SFRH/BD/109135/2015 from the Portuguese Ministry of Sciences and Education, NRI Grants IIS-1546305 and IIS-1637941, and NVIDIA GPU donations.

[1]Source code and pre-trained models available at: https://pedro-morgado.github.io/nettailor.

complemented by task-specific blocks that enable adaptation to new tasks. Given a new target task, we propose to search for the best *architecture* that combines any number of large pre-trained blocks and small task-specific blocks. While pre-trained blocks are responsible for the bulk of feature extraction, task-specific blocks are used to 1) build the final (classification) layer, 2) simplify or even replace pre-trained blocks when possible, or 3) adjust network activations to compensate for domain differences between the source and target tasks.

Evidence for the feasibility of this idea was recently provided in [49], where a pre-trained network is successfully adapted to multiple tasks without changing its parameters by *adding* a small number of residual adaptation layers. In this work, however, instead of merely adding layers, we seek to *adapt the network architecture*, to tailor the network to the complexity of the new task. Because the process is analogous to a tailor that adjusts a pre-made suit to fit a new customer, we denote the procedure NetTailor. The main idea is illustrated in Fig. 1. First, we augment a pre-trained CNN with low-complexity blocks that introduce skip connections throughout the network, and a soft-attention mechanism that controls the selection of which blocks to use. Then, we train the augmented CNN with a loss that penalizes both classification error and complexity. The complexity penalty favors the small task-specific blocks over the large pre-trained ones, encouraging the minimum amount of computation required by the target task. Good classification performance is promoted with a combination of the cross-entropy loss and a variant of model distillation [21], which encourages the simplified CNN to match the performance of a classically fine-tuned CNN. This optimization eliminates blocks with a poor trade-off between complexity and impact on recognition performance.

In sum, NetTailor seeks an architecture that matches the performance of standard fine-tuning, but that is as small as possible and mostly composed of universal blocks shared by many tasks. This procedure has three important properties. First, it enables the deployment of networks of different complexity for different tasks. For example, in simpler recognition problems such as digit recognition (SVHN dataset), NetTailor removed 73.4% parameters, while in high-level tasks such as the recognition of everyday objects (Pascal VOC dataset) only 36.1% of the parameters are removed. Second, because the majority of the parameters required per task belong to shared pre-trained blocks, NetTailor solves more tasks with the same resources and allows task switching to be more efficient. On average, NetTailor only introduces 8% of new task-specific parameters per task, when compared to the size of the pre-trained network. Third, we show that pre-trained blocks can be discarded *without a significant loss in performance*, achieving accuracy similar to previous transfer learning techniques.

2. Related work

NetTailor is related to various CNN topics.

Transfer learning: CNNs are routinely transferred by fine-tuning. NetTailor is a flexible transfer procedure that adjusts the network architecture (not just the weights) while keeping the majority of the parameters unchanged.

Life-long learning & learning without forgetting Intelligent systems integrate knowledge over time, leveraging what they know to solve new tasks. This ability is known as lifelong learning [62] or never-ending learning [42] and is usually incremental, i.e. with tasks learned sequentially. Fine-tuning has two main problems for lifelong learning. First, since the original weights are modified, the number of parameters increases linearly with the number of tasks. This is wasteful since low and mid-level features can be shared across very different image domains [58]. Second, after fine-tuning, network performance can degrade substantially on the source task [16]. This degradation is known as "catastrophic forgetting" and has been the subject of various recent works, which we categorize into two groups.

The first group forces the CNN to "remember" the source task when training on target data [32, 24, 1]. This is done either by 1) preventing network responses for source classes from changing significantly on images of the new task [32, 1], 2) maintaining an "episodic memory" of images from previous tasks [36, 51], 3) preventing the reconstruction of features crucial to the source task from changing [48], or 4) identifying and protecting weights critical for previous tasks [24, 30]. The second group retains previous task knowledge by freezing the source network and adding a small number of parameters for adaptation to the new task. For example, progressive neural networks [55] and dynamically expandable networks [69] expand the original network by adding hidden units to each layer, and Rebuffi *et al*. [49, 50] add small task-specific layers, denoted residual adapters, that adapt the activations of the source network to the target task. Finally, Mallya *et al*. [39, 38] identify a small set of source weights that can be pruned or retrained to improve performance on the target task.

NetTailor has similarities with the second group, since it freezes pre-trained layers. Also, similarly to some methods in the first group, NetTailor uses source activations of intermediate layers as guidance for the activations of the new network. The main difference is that prior techniques do not seek to adapt the network complexity to the task requirements, which results in wasted computation when target tasks are simpler than the source task.

Multi-task learning Multi-task learning (MTL) aims to improve generalization by leveraging relations between tasks [6]. MTL is widely used for problems like object detection, where sharing representations between object location and classification [15, 52] or even segmentation [19] has led to significant gains. Other examples of successful MTL are head orientation and facial attribute detection [72, 47], scene geometry, instance, and semantic segmentation [23, 41], among others. The main difference between MTL and transfer techniques is that MTL assumes that all tasks are performed on the same domain, usually all operating on the same image. This is not the case for transfer, where the target task belongs to a different domain, possibly very dissimilar from that of the source images.

Domain adaptation Domain adaptation addresses the transfer of a task across two domains. When labels are available for both domains, this is usually done by fine-tuning. NETTAILOR addresses the problem that, depending on the gap between domains, there may be a need to adjust the architecture. This is, however, different from unsupervised domain adaptation [14, 63, 64], where there are no labeled data for the target domain. Unlike general transfer techniques like NETTAILOR, unsupervised domain adaptation is designed to bridge the gap between two datasets with *exactly* the same classes, and to maximize performance on the target (unsupervised) dataset with no concern for source domain performance.

Network compression Network compression aims to reduce the size of a neural network by removing weights. Early works [29, 18] derived near-optimal strategies to identify and remove weights of low impact on network performance. However, because these methods rely on second order derivatives of the loss function, they are impractical for deep networks. Recently, good results have been shown with simpler procedures, such as pruning weights of low magnitude [17] or introducing sparsity constraints during training [74]. These methods reduce model size considerably but do not improve the speed of inference, due to the irregular sparsity of pruned weights. Alternative approaches advocate for "structured sparsity" as a means to remove entire filters [31, 43]. NETTAILOR adopts the standard training methodology of iterative pruning (pre-train, prune, re-train), but takes the concept of structured sparsity one step further, pruning entire layers instead of weights or filters. However, NETTAILOR is not a network compression procedure, as layer pruning is only feasible in transfer learning, specifically when the target task is simpler than the source. Existing compression methods could also be used to compress the pre-trained network, further reducing the complexity of networks fine-tuned with NETTAILOR.

Distillation Model distillation algorithms seek to emulate a model with a simpler, smaller or faster one. In [4], a strong ensemble model is used to label a large unlabeled dataset, which is then used to train a simpler model that mimics the ensemble predictions. Similar ideas have been used to transfer knowledge between networks with different characteristics. For example, Ba *et al*. [2] demonstrate that shallow networks can mimic deeper networks while using the same amount of parameters for stronger parallelization, [21] and [53] replicate complex networks with significantly smaller or thinner ones, and [7] transfers a previous network to a new deeper or wider network without retraining for a faster development workflow. The student teacher paradigm used by NETTAILOR is similar to that of FitNets [53], as teacher supervision is added both at the network output and internal activations. However, instead of training a new network from scratch, NETTAILOR adapts the architecture of a pre-trained network without changing most of its weights.

Cascaded classifiers & Adaptive inference graphs Cascaded classifiers [66], can also significantly accelerate inference, by quickly rejecting easy negatives. Recent works developed these ideas within a deep learning framework, both for classification [61, 22] and detection [5, 68]. By introducing early-exits, the network can classify images as soon as it reaches the desired degree of confidence [22, 61], or anytime the decision output is expected [22]. Closer to NETTAILOR is the work on adaptive inference graphs (AIG) [65, 13], which dynamically adjusts the network topology at test time conditioned on the image alone. Thus, similar to NETTAILOR, both cascades and AIG methods can select which parts of the network to evaluate for each image. However, these methods cannot effectively solve the multi-domain classification problem. When networks are trained independently, a different network is generated per task. Training networks jointly requires simultaneous access to all datasets. This drastically restricts the training of networks by different developers, for different tasks, at different times, since different developers 1) may not have access to each other's data, and 2) usually lack the resources and desire to train for tasks other than their own. NETTAILOR addresses this problem by reusing a set of universal blocks shared across datasets, allowing each developer to focus on the single task of interest. It reduces both inference times and space requirements without the need for joint training on all datasets.

Neural architecture search Neural architecture search (NAS) is devoted to learning new network architectures in a data-driven manner. Typically, this is accomplished using reinforcement learning or evolutionary algorithms to update a model responsible for generating architectures so as to maximize performance [75, 76]. Since the space of possible architectures is extremely large, NAS can be quite slow and recent developments focus on accelerating the search process [34, 35]. NETTAILOR can be seen as a differentiable NAS procedure, since the network architecture is optimized for a given task. However, unlike general NAS, we seek a solution that reuses a set of pre-trained blocks in order to address the storage and computing inefficiencies associated with multi-domain transfer learning problems.

Curriculum learning Curriculum learning techniques use variations of back-propagation to improve learning effectiveness. This can be done by controlling the order in which examples are introduced [3]. Other approaches use a teacher network to enhance the learning of a student network [12, 40]. NETTAILOR uses a replica of the source network, fine-tuned on the target task, as a teacher for the learning of the simplified network.

3. Method

In this section, we introduce NETTAILOR.

3.1. Task transfer

A CNN implements a function

$$f(\mathbf{x}) = (G_L \circ G_{L-1} \circ \cdots \circ G_1)(\mathbf{x}). \quad (1)$$

by composing L computational blocks $G_l(\mathbf{x})$ consisting of simple operations, such as convolutions, spatial pooling, normalization among others. For object recognition, $\mathbf{x}$ is an image from a class $y \in \{1,...,C\}$, and $f(\mathbf{x}) \in [0,1]^C$ models the posterior class probability $P(y|\mathbf{x})$. While the blocks $G_l(\mathbf{x})$ differ with the CNN model, they are often large, both in terms of computation and storage. For example, under the ResNet model, each $G_l(x)$ is formed by two 3×3 convolutions, or in deeper versions a "bottleneck" block containing two 1×1 and one 3×3 convolutions [20].

Since CNN training requires a large dataset, such as ImageNet [9], Places [73] or COCO [33], not available for most applications, CNNs are rarely learned from scratch. Instead, a CNN pre-trained on a large dataset is fine-tuned on a new task. In this case, the original task is denoted as the *source* and the new one as the *target* task. Fine-tuning adjusts the weights of the blocks of (1), while maintaining the network architecture. Hence, independently of the complexity of the new task, the computational and storage complexity remain large. This is undesirable for target tasks simpler than the source task, especially for applications that have computational or storage constraints, such as mobile devices.

3.2. NetTailor

In order to avoid these problems, task transfer should ideally have two properties. First, rather than reusing entire networks, it should reuse network blocks. In particular, it should be possible to add or remove blocks to best adapt the architecture to the new task, not just its weights. This way, if the target task is much simpler than the source task, network size could decrease significantly. Second, new networks should reuse existing pre-trained blocks to the largest possible extent, in order to minimize the number of parameters to be learned. Reusing blocks is particularly crucial for memory constrained implementations (e.g., robotics or mobile devices), because it allows sharing of blocks across tasks. In this case, since only a fraction of (task-specific) parameters need to be switched and stored per task, both the costs of task switching and model storage remain low.

In this work, we introduce a new transfer technique, denoted NetTailor that aims to achieve these goals. The NetTailor procedure illustrated in Fig. 1 can be summarized as follows.

1. Train the *teacher network* by fine-tuning a pre-trained network on the target task.
2. Define the *student network* by *augmenting* the pre-trained network with task-specific low-complexity *proxy layers*.
3. Train the task-specific parameters of the student network on the target task to *mimic* the internal activations of the teacher, while imposing *complexity constraints* that encourage the use of low-complexity proxy layers over high-complexity pre-trained blocks.
4. Prune layers with low impact on network performance.
5. Fine-tune the remaining task-specific parameters.

While we only experimented with teacher networks that are learned by fine-tuning (step 1), NetTailor could also be used

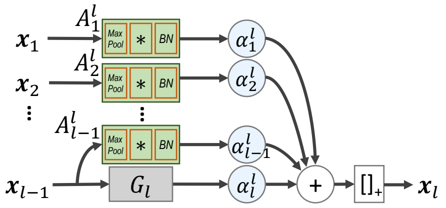


Figure 2: Augmentation of pre-trained block G_l at layer l with multiple proxy layers A_p^l. $\mathbf{x}_i$ represents the network activation after layer i.

with any transfer technique that produces a teacher that preserves the architecture of the pre-trained network. We focused on fine-tuning due to its popularity and high performance for most tasks where a reasonably sized dataset is available for training [27, 70]. Layer pruning (steps 4 and 5) is performed using operations common in the network compression literature [17, 74], and is briefly described in Section 3.5. We now discuss steps 2 and 3 in detail.

3.3. Architecture of the student network

The main architectural component introduced in this work is the augmentation of the pre-trained network $f(\mathbf{x}) = (G_L \circ G_{L-1} \circ \cdots \circ G_1)(\mathbf{x})$ with the complexity-aware pooling block of Figure 2. Starting from the pre-trained model, each layer G_l is augmented with a set of lean proxy layers $\{A_p^l(\cdot)\}_{p=1}^{l-1}$ that introduce a skip connection between layers p and l. As the name suggests, proxy layers aim to approximate and substitute the large pre-trained blocks $G_l(\cdot)$ whenever possible. The output activation $\mathbf{x}_l$ of layer l is then computed by pooling the output of the l^{th} pre-trained block $G_l(\cdot)$ and proxies $A_p^l(\cdot)$

$$\mathbf{x}_l = \alpha_l^l G_l(\mathbf{x}_{l-1}) + \textstyle\sum_{p=1}^{l-1} \alpha_p^l A_p^l(\mathbf{x}_p), \tag{2}$$

where $\{\alpha_p^l\}_{p=1}^l \in [0,1]$ are a set of scalars that enable or disable the different network paths.

Two steps are taken to reduce the number of task-specific parameters. The first is to use proxy layers of low-complexity. Specifically, $A_p^l(\cdot)$ is composed of 1) a spatial max-pooling block that converts activations from the spatial resolution of $\mathbf{x}_p$ into that of $\mathbf{x}_l$, and 2) a 1×1 convolution (with batch normalization) that projects the input feature map $\mathbf{x}_p$ into the desired number of channels for $\mathbf{x}_l$. Thus, in comparison to the standard ResNet block which contains two 3×3 convolutions, each proxy $A_p^l(\cdot)$ contains only $\frac{1}{18}$ of the parameters and performs only $\frac{1}{18}$ of floating point operations. Second, proxy layers are forced to compete with each other to minimize the propagation of redundant information through the network. This is accomplished by introducing a set of auxiliary parameters a_p^l and computing α_p^l as the softmax across all paths merging into layer l

$$\alpha_p^l = \frac{e^{a_p^l}}{\sum_k e^{a_k^l}}. \tag{3}$$

Finally, while the description above implies a dense set of low-complexity proxies, connecting the outputs of all layers $i < l$ to that of layer l, we found this to be often unnecessary (see Section 4.1). Therefore, we limit the number of proxies

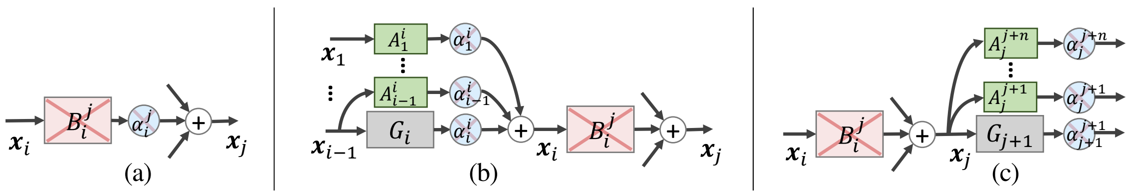


Figure 3: Block removal criteria. (a) Self-exclusion. (b) Input exclusion. (c) Output exclusion.

in (2) to the closest k, and use

$$\mathbf{x}_l = \alpha_l^l G_l(\mathbf{x}_{l-1}) + \sum_{p=\max(l-k,1)}^{l-1} \alpha_p^l A_p^l(\mathbf{x}_p). \tag{4}$$

Figure 1 illustrates the initial student architecture for $k=3$.

3.4. Tailoring the student to the target task

The student network seeks a trade-off of two goals: low complexity and performance similar to the teacher.

3.4.1 Constraining student complexity

In the complexity-aware pooling block of Fig. 2, scalars $\{\alpha_p^l\}_{p=1}^l$ act as a soft-attention mechanism that selects which blocks to use for the target task. Let $B_i^j(\cdot)$ denote the computational block associated with path $i \rightarrow j$, i.e. $B_i^j(\cdot) = G_i(\cdot)$ if $i=j$ or $B_i^j(\cdot) = A_i^j(\cdot)$ otherwise. Then, block $B_i^j(\cdot)$ can be removed if one of three conditions hold:

- Self-exclusion (Fig. 3a): path $i \rightarrow j$ is excluded, i.e. $\alpha_i^j = 0$;
- Input exclusion (Fig. 3b): all paths merging into node i are excluded, i.e. $\alpha_k^i = 0, \forall k \leq i$;
- Output exclusion (Fig. 3c): all paths departing from node j are excluded, i.e. $\alpha_j^k = 0, \forall k > j$ and $\alpha_{j+1}^{j+1} = 0$.

Note that while self-exclusion only allows the removal of a single block, both input and output exclusion remove multiple blocks simultaneously. For example, if all paths merging into node i are excluded, then all blocks departing from this node have no viable input and can be removed. Similarly, if all paths departing from node j are excluded, then all blocks merging into this node will end up being ignored and can be removed as well.

To tailor the architecture to the target task, the set of scalars $\{\alpha_p^l\}_{p=1}^l$ should enable high performance, but minimize the expected network complexity. Let $R_{self}^{i,j}$, R_{inp}^i and R_{out}^j denote the events associated with conditions 1, 2 and 3, respectively, and $\mathcal{C}_i^j$ the complexity of block B_i^j. Then, the expected complexity of block B_i^j is

$$E\left[\mathcal{C}_i^j\right] = \mathcal{C}_i^j \left(1 - P(R_{self}^{i,j} \cup R_{inp}^i \cup R_{out}^j)\right). \tag{5}$$

Under the assumptions that events $R_{self}^{i,j}$, R_{inp}^i and R_{out}^j are disjoint, and events $R_{self}^{i,k}$ are all independent, the probability of (5) is given by

$$P(R_{self}^{i,j} \cup R_{inp}^i \cup R_{out}^j) = P(R_{self}^{i,j}) + P(R_{inp}^i) + P(R_{out}^j) \tag{6}$$

with

$$P(R_{inp}^i) = P\left(\cap_{k \leq i} R_{self}^{k,i}\right) = \prod_{k \leq i} P(R_{self}^{k,i}) \tag{7}$$

$$\begin{aligned} P(R_{out}^j) &= P\left(\cap_{k>j} R_{self}^{j,k} \cap R_{self}^{j+1,j+1}\right) \\ &= P(R_{self}^{j+1,j+1}) \cdot \prod_{k>j} P(R_{self}^{j,k}). \end{aligned} \tag{8}$$

Finally, by modeling the probability of self-exclusion by $P(R_{self}^{i,j}) = r_i^j = 1 - \alpha_i^j$, then (5) becomes

$$E\left[\mathcal{C}_i^j\right] = \mathcal{C}_i^j \left(1 - r_i^j - \prod_{k \leq i} r_k^i - r_{j+1}^{j+1} \prod_{k>j} r_j^k\right), \tag{9}$$

and the expected network complexity

$$E[\mathcal{C}] = \sum_{i,j} E\left[\mathcal{C}_i^j\right]. \tag{10}$$

Although the exclusion events may not be disjoint or independent, the minimization of (10) still provides the desired incentive towards the use of low-complexity proxies. Hence, we use (10) as a differentiable complexity penalty explicitly enforced during training.

3.4.2 Mimicking the teacher

The teacher network is obtained by fine-tuning a pre-trained network for the target task. To transfer this knowledge to the student network, the latter is encouraged to match the internal activations of the teacher, by adding an L_2 regularizer

$$\Omega = \sum_l \|\mathbf{x}_l^t - \mathbf{x}_l\|^2, \tag{11}$$

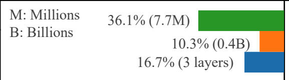
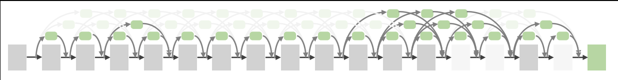
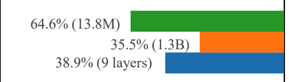
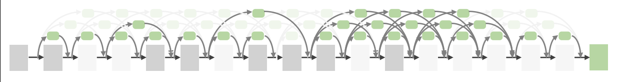
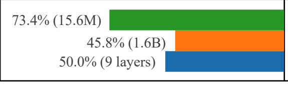
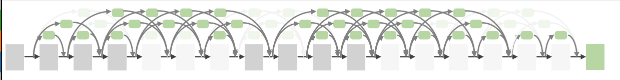
where $\mathbf{x}_l^t$ is the activation of l^{th} block of the teacher network, $\mathbf{x}_l$ the corresponding activation of the student network given by (2), and the sum is carried over all internal blocks as well as network outputs (prior to the softmax).

3.4.3 Loss function

NetTailor optimizes all task-specific parameters of the student network end-to-end to meet three goals: 1) minimize classification loss on the target task, 2) minimize network complexity and 3) minimize the approximation error to the teacher network. Given a target dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}$ of images $\mathbf{x}_i$ and labels y_i, this is accomplished by minimizing the loss function

$$L = \sum_i L_{cls}(f(\mathbf{x}_i), y_i) + \gamma_1 E[\mathcal{C}] + \gamma_2 \Omega, \tag{12}$$

where $f(\cdot)$ denotes the output of the student network, $L_{cls}(f(\mathbf{x}), y)$ is the cross-entropy loss between the network prediction $f(\mathbf{x})$ and ground-truth label y, $E[\mathcal{C}]$ is the expected network complexity of (9) and (10), Ω is the teacher approximation loss of (11), and γ_1 and γ_2 two hyper-parameters that control the importance of each term.

Dataset	Accuracy		Reduction in Complexity	Learned Architecture
	Fine-tuning	NetTailor		
VOC	82.82%	82.90%		
Flowers	95.84%	95.51%		
SVHN	96.59%	96.53%		

🟩 # Parameters Removed 🟧 # FLOPs Removed 🟦 # Layers Removed ⬜ Pre-trained blocks 🟩 Task-specific blocks

Figure 4: Reduction of network complexity and final architecture after adapting ResNet34 to three datasets using NetTailor.

3.5. Pruning and fine-tuning

After training the student network, the magnitude of the scalars α_i^j reflects the importance of each block, with values close to zero indicating a low impact on network performance. Given this observation, we threshold the scalars α_i^j, and use the three exclusion conditions outlined above to remove all unnecessary blocks. In order to enable better control over the trade-off between performance and complexity, proxies and pre-trained blocks are removed using different pruning schemes. Since proxy layers are both small and crucial for the adaptation to the target task, we define a very low threshold θ (typically 0.05) and only remove proxies with $\alpha_i^j < \theta$. As for pre-trained layers, we first rank their importance by the values of α_i^i, and remove the k least important blocks. Finally, in order to recover from the removal of network components, all remaining task-specific layers are fine-tuned to minimize the loss of (12) without complexity constraints ($\gamma_1 = 0$).

4. Evaluation

We conducted a series of experiments to evaluate the NetTailor procedure. Section 4.1 provides an in-depth analysis of the impact of important variables such as the complexity of the target task, the depth of the pre-trained network, the importance of the teacher and the number of skips in the student network. Then, to demonstrate the effectiveness of the proposed procedure, Section 4.2 compares Net-Tailor to prior work on several datasets.

4.1. Analysis

We analyze Net-Tailor using three classification datasets of varying characteristics: SVHN, VGG-Flowers and Pascal VOC 2012. SVHN [45] is a large digit recognition dataset containing 100k images of street view house numbers. VGG-Flowers [46] is a small fine-grained dataset composed by 8k images distributed across 102 flower species. PASCAL VOC 2012 [11] is a dataset for the detection of a small number (20) of common objects. While VOC was designed for object detection, we test our method on the classification task alone. We used ground-truth bounding boxes to crop all objects with a 20 % margin and re-sampled the dataset to avoid large class imbalances. We used standard training and test sets in all cases.

Training details We now describe the standard implementation of NetTailor which, unless otherwise specified, is used throughout our experiments. Global blocks are obtained by pre-training a large CNN model on ImageNet (ResNet34 in most of our experiments) and remain unchanged afterward in order to share them across tasks. For each target task, the teacher is trained by fine-tuning the pre-trained network. The student is assembled by augmenting the pre-trained blocks with three skip connections per layer, and all task-specific parameters (i.e. final classifier, proxy layers and scalars α) are trained to minimize the loss of (12) with $\gamma_1 = 0.3$ (complexity constraints) and $\gamma_2 = 10$ (teacher loss). In the complexity constraints of (9), the complexity $\mathbb{C}_i^j$ is defined as the number of FLOPs of each block normalized by the total number of FLOPs of the pre-trained network. This definition makes pre-trained layers about 20 times more expensive than proxy layers. One critical detail is the initialization of the scalars α to initially favor pre-trained over task-specific blocks. This initialization provided a good starting point for learning (i.e. similar to the pre-trained network alone) and reduced overfitting. Specifically, we set the initial value of a_i^i to 2 for all i (i.e. pre-trained blocks), and a_i^j to -2 for all $i \neq j$ (i.e. proxies). After training the student network, we remove all proxies with $\alpha_i^j < 0.05$ and the k least important pre-trained blocks (as ranked by the values of α_i^i). Finally, we fine-tune the remaining task-specific parameters to minimize the loss of (12) without complexity constraints $\gamma_1 = 0$. The pruning and retraining steps are repeated multiple times with different values of k, and the leanest model that achieves a target accuracy within 0.5% of the teacher network is chosen as the final architecture. All hyper-parameter values were chosen based on early experiments and used for all three datasets, as they tend to provide a good trade-off between accuracy and network complexity. A study of some of these parameters is provided below. As usual with classification problems, we used stochastic gradient descent with momentum in all training steps.

Effectiveness of NetTailor on various datasets: To study the impact of dataset complexity, we tuned the ResNet34 architecture using NetTailor and measured the maximum achievable reduction in network complexity that retains performance similar to fine-tuning. The results are shown in Fig. 4 for three different datasets. We list the total number of layers,

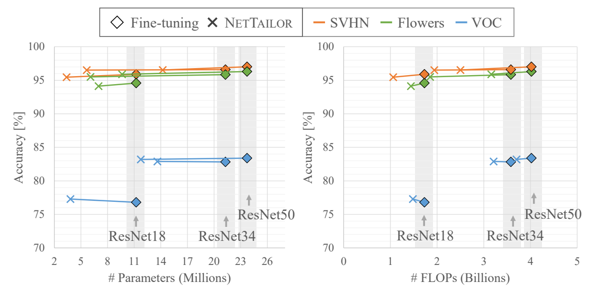


Figure 5: Accuracy vs. complexity of models of increasing depth. Diamonds represent the fine-tuned model and crosses the model obtained with NetTailor. The lines connect fine-tuned models to their adapted counterparts.

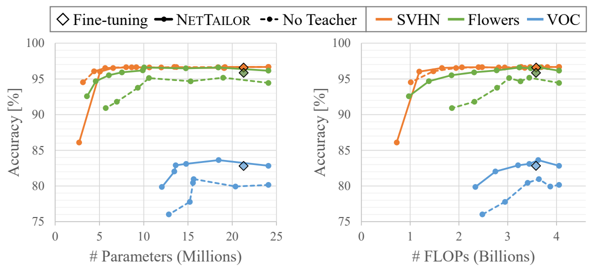


Figure 6: Accuracy vs. complexity of models discovered by NetTailor with and without the teacher. Right-most dots represent unpruned networks and subsequent ones networks with increasing numbers of removed layers.

parameters (global and task-specific) and FLOPs removed from the pre-trained network by NetTailor. We also display the final learned architecture for each task. Fig. 4 shows that networks trained for simpler tasks, such as SVHN, are the most heavily pruned, with 9 out of 18 pre-trained blocks removed. This results in a drastic 73.4 % reduction in total parameters and a 45.8 % reduction in FLOPs. For simpler tasks, most residual blocks are unnecessary and fine-tuning likely converts them into transformations close to the identity, which can be replaced by low complexity proxies. NetTailor also obtains significant reductions for the more complex Flowers and VOC datasets. Overall, the results of Fig. 4 show that, for many applications, large pre-trained networks can be significantly reduced, both in size and speed, without loss of performance. Furthermore, because the pre-trained blocks remain unchanged, only a small number of new parameters is introduced per task: 1.90M (million) for VOC, 1.88M for flowers and 1.85M for SVHN (pre-trained ResNet34 blocks have a total of 21.29M parameters).

Depth of pre-trained model: To understand the effectiveness of NetTailor when applied to networks of increasing depth, we used the ResNet model family: ResNet18, ResNet34, and ResNet50. Unlike ResNet18 and ResNet34, ResNet50 blocks have a bottleneck architecture, where input maps are projected into a low-dimensional space by a 1×1 convolution, then processed by a 3×3 convolution, projected back into the high-dimensional space through a 1×1 convolution, and added to the residual link. Due to the high-dimensionality of the output, this block architecture does not allow the use of proxies as defined in Fig. 2, since 1×1 convolutions in the high-dimensional space are still expensive. Instead, to keep the complexity of each proxy at about $\frac{1}{20}$ of the pre-trained block, we employ a bottleneck structure to the proxy as well, i.e. we employ two consecutive 1×1 convolutions with batch-norm. The first projects the input into a low-dimension space (4 times smaller than the bottleneck of the pre-trained block), and the second restores the input dimensionality.

		Accuracy	# Parameters	# FLOPS	# Blocks
SVHN	Fine-tuning	96.59 %	21.29 M	3.58 B	18
	1-Skip	96.60 %	5.13 M	1.79 B	9
	3-Skip	96.53 %	5.65 M	1.94 B	9
	5-Skip	96.79 %	6.14 M	1.62 B	7
	Dense-Skip	96.13 %	**4.01 M**	**1.22 B**	**5**
Flowers	Fine-tuning	95.84 %	21.29 M	3.58 B	18
	1-Skip	96.05 %	8.06 M	2.66 B	13
	3-Skip	95.51 %	**6.07 M**	**1.85 B**	**9**
	5-Skip	95.56 %	7.12 M	2.45 B	9
	Dense-Skip	95.58 %	6.45 M	2.29 B	11
VOC	Fine-tuning	82.82 %	21.29 M	3.58 B	18
	1-Skip	82.42 %	**12.81 M**	3.13 B	15
	3-Skip	82.33 %	13.54 M	2.98 B	14
	5-Skip	82.56 %	12.94 M	**2.89 B**	**13**
	Dense-Skip	82.56 %	14.85 M	3.30 B	15

Table 1: Effect of initial student architecture.

The results depicted in Fig. 5 show that NetTailor can produce architectures that achieve the performance of a larger CNN (e.g. ResNet50) with the same or fewer parameters as a smaller one (ResNet18). This is especially important for more complex problems, where network depth has a bigger impact on performance. For example, for the VOC dataset, NetTailor is able to reduce ResNet50 to only 11.7M parameters, only 0.5M more than ResNet18, but with much higher performance (83.2 % vs 79.6 % accuracy). Reduction in inference speed, however, was not as drastic for the VOC dataset, since NetTailor mostly removed high-level layers which contain most of the parameters but only account for a small number of operations.

Teacher supervision: Fig. 6 shows the advantage of using a fine-tuned network as the teacher. Each line in Fig. 6 shows the performance achieved by the model after removing different numbers of blocks k (smaller values of k produce models of higher complexity). As can be seen, removing the teacher leads to significant loss in performance, regardless of the number of removed blocks, with the student network never achieving the same performance as fine-tuning. The exception to this trend is the SVHN dataset, which is a simple dataset with a large number of images. This indicates that the skip architecture of Fig. 1 is prone to overfitting in smaller datasets, but teacher supervision provides an effective solution to this problem.

Student architecture: We also compare different student architectures, by augmenting ResNet34 with 1, 3, 5 or a dense set of skip proxies. The results presented in Table 1 show that augmenting the student architecture with a dense set of proxies can be beneficial for simpler datasets like SVHN. This

	CUB [67]			Cars [25]			Flowers [46]			WikiArt [56]			Sketch [10]			Avg	Avg	Avg
	Acc	Params	FLOPs	Acc	Params	FLOPs	Acc	Params	FLOPs	Acc	Params	FLOPs	Acc	Params	FLOPs	Acc	Params	FLOPs
Feature [38]	70.03	23.9	4.11	52.80	23.9	4.11	85.99	23.9	24.0	55.60	23.9	4.11	50.86	23.9	4.11	63.05	23.9	4.11
PackNet $\rightarrow$ [39]	80.31	23.9	4.11	86.11	23.9	4.11	93.04	23.9	4.11	69.40	23.9	4.11	76.17	23.9	4.11	81.01	23.9	4.11
PackNet $\leftarrow$ [39]	71.38	23.9	4.11	80.01	23.9	4.11	90.55	23.9	4.11	70.31	23.9	4.11	78.70	23.9	4.11	78.19	23.9	4.11
Piggyback [38]	81.59	24.3	4.11	89.62	24.3	4.11	94.77	24.3	4.11	71.33	24.3	4.11	79.91	24.3	4.11	83.44	24.3	4.11
NetTailor	**82.52**	**13.7**	**3.31**	**90.56**	**12.9**	**3.31**	**95.79**	**8.5**	**2.37**	**72.98**	**15.4**	**3.55**	**80.48**	**15.1**	**3.44**	**84.47**	**13.1**	**3.20**

Table 2: Accuracy and model complexity for prior transfer learning methods in five datasets. PackNet performance is sensitive to the order in which datasets are presented. $\rightarrow$ indicates the following order: CUB, Cars, Flowers, WikiArt and Sketch. $\leftarrow$ indicates reversed order.

	ImNet [9]	Airc [37]	C100 [26]	DPed [44]	DTD [8]	GTSR [60]	Flwr [46]	Oglt [28]	SVHN [45]	UCF [59]	Mean	Score	Avg Params	Avg FLOPS
LwF [32, 49]	59.87	61.15	**82.23**	92.34	58.83	97.57	83.05	88.08	96.10	50.04	76.93	2515	5.86	0.87
Piggyback [38]	57.69	65.29	79.87	**96.99**	57.45	97.27	79.09	87.63	**97.24**	47.48	76.60	2838	6.04	0.87
DAN [54]	57.74	64.12	80.07	91.30	56.54	98.46	86.05	89.67	96.77	49.38	77.01	2851	6.54	0.97
ResAdapt [50]	60.32	64.21	81.91	94.73	58.83	99.38	84.68	89.21	96.54	**50.94**	78.07	3412	6.44	0.96
NetTailor	**61.42**	**75.07**	81.84	94.68	**61.28**	**99.52**	**86.53**	**90.09**	96.44	49.54	**79.64**	**3744**	**3.67**	**0.61**

Table 3: Accuracy and model complexity of several methods on the visual decathlon challenge.

is because accurate digit classification depends largely on lower-level features that are directly bypassed into the classification layer by proxies that skip a large number of blocks. By contrast, dense skips are unnecessary for harder datasets, such as Flowers or VOC, with NetTailor removing most of the long reach proxies. Also, as shown in Table 1, directly imposing a limit on the number of proxies per layer leads to more significant reductions in complexity for the same performance level.

4.2. Comparison to prior work

Finally, we compare NetTailor to prior transfer learning methods designed for the efficient classification of multiple domains. We follow two experimental protocols. The first protocol described in [38] consists of five datasets: CUB [67], Stanford Cars [25], Oxford Flowers [46], WikiArt [56] and Sketch [10]. Following [38], we use the same train/test set splits, and apply the NetTailor procedure to the same backbone network, ResNet50, with an input size 224x224. The second protocol is the visual decathlon benchmark [49] and consists of ten different datasets including ImageNet, Omniglot, German Traffic Signs, among others. We use the same train/validation/test sets provided in [49] which contain images resized to a common resolution of 72 pixels. Similar to [50], we also use a wide residual network [71] with 26 layers pre-trained on ImageNet. Results are reported using both top-1 accuracy and the "decathlon score" [49] which pools all results in a single metric that accounts for the different difficulty of each task.

Table 2 compares the results of NetTailor to several methods. Feature extraction computes features from a pre-trained network, which are then used to build a simple classifier. While feature extraction shares most weights across datasets, differences between the source and target domains cannot be corrected, thus achieving low performance. More refined methods, such as PackNet [39] and Piggyback [38], try to selectively adjust the network weights in order to remember previous tasks, or freeze the backbone network and learn a small set of task-specific parameters (a set of masking weights in the case of Piggyback) that is used to bridge the gap between source and target tasks. All these methods ignore the fact that source and target datasets can differ in terms of difficulty, and thus the *architecture* itself should be adjusted to the target task, not just the weights. As seen in Table 2, these methods are not competitive with NetTailor, which can significantly reduce the network complexity both in terms of model size and inference speed. NetTailor outperforms all approaches in all datasets, improving the classification accuracy of the second best method in four out of five datasets, while requiring an average of 46% fewer parameters and 22% fewer FLOPS.

Comparisons in the Visual Decathlon benchmark show similar findings. In addition to Piggyback [38], we also compared to Learning without Forgetting (LwF) [32], deep adaptation networks (DAN) [54] and parallel Residual Adapters (ResAdapt) [50]. LwF learns a new network per task that retains the responses of the original ImageNet model. Hence, similar to fine-tuning, the number of parameters in LwF also grows linearly with the number of tasks. Both ResAdapt and DAN address this problem by introducing a small amount of extra parameters that adapt the source network to the target task. This is accomplished by adjusting each layer's activations in the case of ResAdapt, or their parameters directly in the case of DAN. Although both methods can share large blocks across tasks, none try to adjust the model complexity to the target task. As shown in Table 3, NetTailor outperforms ResAdapt by 1.57% across 10 datasets and achieves 332 points higher in the decathlon score. More importantly, NetTailor only uses 3.67×10^6 parameters (43% fewer than ResAdapt) and 0.61×10^9 FLOPs (36% fewer than ResAdapt) per task.

5. Conclusion

In this work, we introduced a novel transfer learning approach, denoted NetTailor, which adapts the *architecture* of a pre-trained model to a target task. NetTailor uses the layers of the pre-trained CNN as universal blocks shared across tasks and combines them with small task-specific layers to generate a new network. Experiments have shown that NetTailor is capable of learning architectures of increasing complexity for increasingly harder tasks, while achieving performances similar to that of transfer techniques like fine-tuning.

References

[1] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

[4] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2006.

[5] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. Learning complexity-aware cascades for deep pedestrian detection. In *International Conference on Computer Vision (ICCV)*, 2015.

[6] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[7] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[8] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.

[10] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[12] Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. *arXiv preprint arXiv:1805.03643*, 2018.

[13] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[14] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning (ICML)*, 2015.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[16] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[18] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1993.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *International Conference on Computer Vision (ICCV)*, 2017.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[22] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018.

[23] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[24] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *National Academy of Sciences*, 2017.

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

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

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[28] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[29] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1990.

[30] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[31] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[32] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.

[34] 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 *European Conference on Computer Vision (ECCV)*, 2018.

[35] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[36] David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[37] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[38] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *European Conference on Computer Vision (ECCV)*, 2018.

[39] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[40] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*, 2017.

[41] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[42] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bo Yang, Justin Betteridge, Andrew Carlson, B Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[43] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[44] Stefan Munder and Dariu M Gavrila. An experimental study on pedestrian classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 28(11):1863–1868, 2006.

[45] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems Workshop (NeurIPS)*, 2011.

[46] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

[47] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[48] Amal Rannen, Rahaf Aljundi, Matthew B Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. In *International Conference on Computer Vision (ICCV)*, 2017.

[49] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[50] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[51] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[52] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[53] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[54] Amir Rosenfeld and John K Tsotsos. Incremental learning through deep adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2018.

[55] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[56] Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015.

[57] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[58] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRw)*, 2014.

[59] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[60] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[61] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR)*, 2016.

[62] Sebastian Thrun. A lifelong learning perspective for mobile robot control. In *Intelligent Robots and Systems*, pages 201–214, 1995.

[63] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *International Conference on Computer Vision (ICCV)*, 2015.

[64] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[65] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *European Conference on Computer Vision (ECCV)*, 2018.

[66] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition (CVPR)*, 2001.

[67] 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.

[68] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[69] Jaehong Yoon, Eunho Yang, et al. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[70] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[71] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016.

[72] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision (ECCV)*, 2014.

[73] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[74] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision (ECCV)*, 2016.

[75] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[76] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.