Refining Architectures of Deep Convolutional Neural Networks

Sukrit Shankar* Duncan Robertson† Yani Ioannou* Antonio Criminisi† Roberto Cipolla*
* Machine Intelligence Lab, University of Cambridge, UK
† Microsoft Research Cambridge, UK
ss965@cam.ac.uk | a-durobe@microsoft.com | yai20@cam.ac.uk | antcrim@microsoft.com | rc10001@cam.ac.uk

Abstract

Deep Convolutional Neural Networks (CNNs) have recently evinced immense success for various image recognition tasks [10, 26]. However, a question of paramount importance is somewhat unanswered in deep learning research - is the selected CNN optimal for the dataset in terms of accuracy and model size?

In this paper, we intend to answer this question and introduce a novel strategy that alters the architecture of a given CNN for a specified dataset, to potentially enhance the original accuracy while possibly reducing the model size. We use two operations for architecture refinement, viz. stretching and symmetrical splitting. Stretching increases the number of hidden units (nodes) in a given CNN layer, while a symmetrical split of say K between two layers separates the input and output channels into K equal groups, and connects only the corresponding input-output channel groups. Our procedure starts with a pre-trained CNN for a given dataset, and optimally decides the stretch and split factors across the network to refine the architecture. We empirically demonstrate the necessity of the two operations.

We evaluate our approach on two natural scenes attributes datasets, SUN Attributes [15] and CAMIT-NSAD [19], with architectures of GoogleNet and VGG-11, that are quite contrasting in their construction. We justify our choice of datasets, and show that they are interestingly distinct from each other, and together pose a challenge to our architectural refinement algorithm. Our results substantiate the usefulness of the proposed method.

1. Introduction

Deep Convolutional Neural Networks (CNNs) have recently shown immense success for various image recognition tasks, such as object recognition [10, 21], recognition of man-made places [26], prediction of natural scenes attributes [19] and discerning of facial attributes [12]. Out of the many CNN architectures, AlexNet [10], GoogleNet [21] and VGG [20] can be considered as the most popular ones, based on their impressive performance across a variety of datasets. While architectures of AlexNet and GoogleNet have been carefully designed for the large-scale ImageNet [4] dataset, VGG can be seen as being relatively more generic in curation. Irrespective of whether an architecture has been hand-curated for a given dataset or not, all of them show significant redundancy in their parameter space [8, 11], i.e. for a significantly reduced number of parameters (sometimes as high as 90% reduction), a given architecture might achieve nearly the same accuracy as obtained with the entire set of parameters. Researchers have utilized this fact to speed up inference by estimating the set of parameters that can be zeroed out with a minimal loss in original accuracy. This is typically done through sparse optimization techniques [11] and low-rank procedures [8].

Envisaging a real-world scenario: We try to envisage a real-world scenario. A user has a new sizeable image dataset, which he wants to train with a CNN. He would typically try out famous CNN architectures like AlexNet, GoogleNet, VGG-11, VGG-16, VGG-19 and then select the one which gives maximum accuracy. In case his application prioritizes a reduced model size (number of parameters) as compared to the accuracy (e.g. applications for mobile platforms and embedded systems), he will try to strike a manual trade-off of how much he wants to sacrifice the accuracy for a reduction in model size.

Once he makes his choice, what if his finally selected architecture could be altered so as to potentially give a better accuracy while also reducing the model size ? In this paper, we aim to target such a scenario. Note that in all cases, the user can further choose to apply one of the sparsification techniques such as [11] to significantly reduce the model size with a slight decrease in accuracy. We now formally define our **problem statement** as follows :

Given a pre-trained CNN for a specific dataset, refine the architecture in order to potentially increase the accuracy while possibly reducing the model size.

Operations for CNN architecture refinement: One may now ask what is exactly meant by the *refinement of a CNN architecture*. On a broader level, refining a CNN

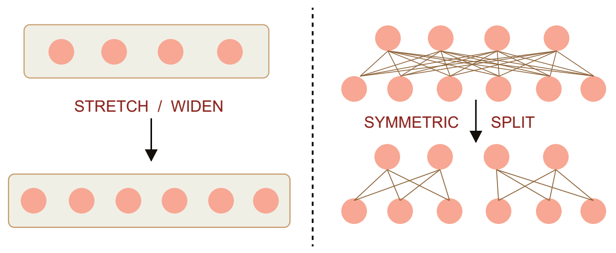


Figure 1. **Operations considered for our approach:** We consider two operations, viz. stretch (*left*) and symmetric split (*right*), for architectural refinement of a CNN. Stretching refers to increase in number of hidden units (nodes) for a given layer, without changing its connection pattern to the previous or the next layer. A stretch by a factor of 1.5 is shown here. A symmetrical split of say K between two layers separates the input and output channels into K equal groups, and the corresponding input and output channel groups are connected. A symmetric split of 2 is shown here. Symmetrical split is implemented as the *group* parameter in Caffe [9].

architecture can involve altering one or more of the following: the number of hidden units (nodes) in any layer, the connection pattern between any two layers, and the depth of the network. On a relatively finer level, one might think of changing the convolution kernel size, pooling strategies and stride values to refine an architecture.

In this paper, we consider the task of CNN architecture refinement on a broader level. Since we embark on such a problem in this work, we only consider two operations, viz. stretch and symmetric split. Stretching refers to increase in number of hidden units (nodes) for a given layer, while a symmetrical split of say K between two layers separates the input and output channels into K equal groups, and the k^{th} input channel group is only connected to the k^{th} output channel group[1]. Please see Fig 1 for an illustration of these operations. We do *not* consider the other plausible operations for architectural refinement of CNN; for instance, arbitrary connection patterns between two layers (instead of just symmetric splitting), reducing the number of nodes in a layer, and alteration in the depth of the network.

Intuition behind our approach: The main idea behind our approach is to best separate the classes of a dataset, assuming a constant depth of the network. Our method starts with a pre-trained CNN, and studies separation between classes at each convolutional layer. Based on the nature of the dataset, separation between some classes may be more at lower layers, while for others, may be lesser at lower layers. Similar variations may be seen at deeper layers. In comparison to its previous layer, a given layer can increase the class separation for some class pairs, while decreasing for others. The number of class pairs for which the class separation increases contributes to the *stretching / widening* of the layer; while the number of class pairs where the class separation decreases contributes to the *symmetric splitting* of the layer inputs. Thus, both stretch and split operations can be simultaneously applied to each layer. The amount of stretch or split is not only decided by how the layer affects the class separation, but also by the class separation capacity of the subsequent layers. Once the stretch and split factors are estimated, they are applied to original CNN for architectural refinement. The refined architecture is then trained again (from scratch) on the same dataset. Section 3 provides complete details of our proposed approach.

Our contribution(s): Our major contributions can now be summarized as follows :

1. For a given pre-trained CNN, we introduce the problem of refining network architecture so as to potentially enhance the accuracy while possibly reducing the required number of parameters.

2. We introduce a strategy that starts with a pre-trained CNN, and uses stretch and symmetric split (Fig 1) operations for CNN architecture refinement.

2. Related Work

Deep Convolutional Neural Networks (CNNs) have experienced a recent surge in computer vision research due to their immense success for visual recognition tasks [10, 26]. Given a sizeable training set, CNNs have proven to be far more robust as compared to the hand-crafted low-level features like Histogram of Oriented Gradients (HOG) [3], color histograms, gist descriptors [13] and the like. For visual recognition, CNNs provide impressive performance for recognition of objects [21], man-made places [26], attributes of natural scenes [19] and facial attributes [12]. However, learning an optimal CNN architecture for a given dataset is largely an open problem. Moreover, it is less known, how to find if the selected CNN is optimal for the dataset in terms of accuracy and model size or not.

Transfer learning with deep nets: With the availability of large scale datasets such as ImageNet [4] and MIT Places [26], researchers have resorted to transfer learning techniques [22] for efficient training of relatively smaller related datasets [14, 27]. During transfer learning, the parameters of the CNN trained with base dataset are duplicated, and some additional layers are attached at the deep end of the CNN which are trained exclusively on the new dataset. In the process, the parameters copied from the net trained on the base dataset might or might not be allowed for slight perturbation. However, none of the transfer learning techniques attempts to refine the CNN architecture ef-

[1]For better understanding, we give an example of symmetric splitting with convolutional layers. Let a convolutional layer $conv_1$ having 96 outputs be connected to $conv_2$ having 256 outputs. Then there are 96×256 input connections for $conv_2$, each connection having a filter of square size $(11 \times 11 \ \ say)$. A splitting of 2 for $conv_2$ divides input connections of $conv_2$ into 2 symmetric groups, such that the first / second 48 outputs of $conv_1$ only get connected to the first / second 128 outputs of $conv_2$.

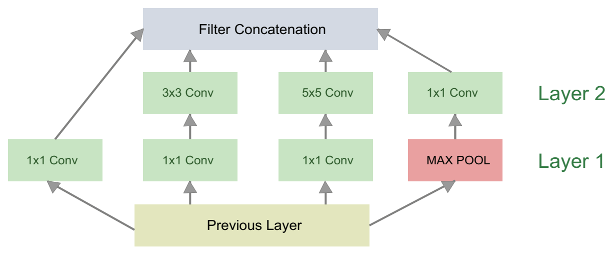


Figure 2. **Inception Module of GoogleNet [21] :** The inception module is an intrinsic component of the GoogleNet architecture. GoogleNet has 9 inception modules named as 3a, 3b, 4a, 4b, 4c, 4d, 4e, 5a, 5b connected one after another. The inception module has two layers and 6 convolutional blocks (green blocks), connected as shown in the figure. As an implementation perspective of our approach with GoogleNet, for a convolutional block l in Layer 1, the subsequent blocks are all convolutional blocks in layer 2, irrespective of the connection pattern. This is done for ease in the computation of (2) and (3). However, for a given convolutional block l in a layer of inception module, its previous convolutional block is considered only to be the one from which l has incoming links. The distinction is made for simplicity in computation, as the statistics of the previous layer is only required in case (b) of our approach (Section 3), for deciding whether any operation should be applied to the current block or not.

fecting an increase in original accuracy and a reduction in model size simultaneously. While transfer learning can be effective when the base dataset has a similar distribution as the target dataset, it might be a deterrent otherwise [19]. We emphasize that our approach can be applied to any pre-trained CNN, irrespective of whether the training has been done through transfer learning or from scratch.

Low-rank and sparsification methods for CNNs: Irrespective of whether a CNN has been hand-designed for a specific dataset or not, all the famous CNN architectures exhibit enormous redundancy in parameter space [11]. Researchers have recently exploited this fact to speed up the inference speeds by estimating a highly reduced set of parameters which is sufficient to produce nearly the same accuracy as the original CNN does with the full set of parameters. While some works like [8, 17, 23] have resorted to low-rank factorization of weight matrices between two given layers, others have used sparsification methods for the same [6]. Recently, [11] has combined the low-rank and sparsification methods to produce a highly sparse CNN with a slight decrease in the original accuracy. The work of [5] can be considered as a pseudo-reduction method for the parameter space of a CNN. It does not sparsify the network, but presents an approach to estimate almost 95% of parameters from only the rest 5%. Thus, they do not claim that most parameters are not necessary, but that most parameters can be estimated by a relatively small set.

It is worthwhile to mention that our approach falls into a different solution paradigm, that can complement various methods developed for deep learning for distinct purposes. All the related works discussed above and some other works that tend to enhance the accuracy with deep learning such as [19] and deep boosting methods [1, 16, 18], assume a fixed architectural model of the CNN. Our approach instead modifies the architecture of the original CNN to potentially enhance the accuracy while possibly reducing the model size. Thus, all the techniques applied to a fixed architecture can be applied to the architecture refined by our method, for a plausibly better performance as per the chosen metric. Also, due to the novel operations that we consider for CNN architectural refinement, our method can complement the various other methods developed for a similar purpose.

3. Approach

Let the dataset contain M classes. Let the CNN architecture have L convolutional layers. At a given convolutional layer $l \in \{1, \dots, L\}$, let there be h_l number of hidden units (nodes). Then for a given input image i, one can generate an h_l dimensional feature vector $\boldsymbol{f_l^i}$ at convolutional layer l, by taking average spatial responses at each hidden unit of the layer [25]. Using this, one can find a mean feature vector of dimension h_l for every class $m \in \{1, \dots, M\}$ at every convolutional layer l by taking the average of $\boldsymbol{f_l^i}\ \ \forall\ \ i \in \boldsymbol{a_m}$, where the set $\boldsymbol{a_m}$ contains images annotated with class label m. Let this average feature vector for class m be denoted by $\boldsymbol{g_l^m}$.

Finding the inter-class separation: For a given dataset and a base CNN architecture, we first train the CNN on the dataset using a given loss function (such as softmax loss, sigmoid cross entropy loss, etc. [9]). From this pre-trained CNN, we compute $\boldsymbol{g_l^m}; l \in \{1, \dots, L\}, m \in \{1, \dots, M\}$. Using $\boldsymbol{g_l^m}$, inter-class correlation matrices $\boldsymbol{C_l}$ of sizes $M \times M$ are found out for every convolutional layer l, where a value at the index-pair $(m, \hat{m}); m, \hat{m} \in \{1, \dots, M\}$ in $\boldsymbol{C_l}$ indicates the correlation between $\boldsymbol{g_l^m}$ and $\boldsymbol{g_l^{\hat{m}}}$. Note that the correlation between two feature vectors of the same length can vary between -1 and 1, inclusive. Examples of $\boldsymbol{C_l}$ can be seen in Fig 4. All $\boldsymbol{C_l}$ are symmetric, since correlation is non-causal. A lesser correlation between classes implies better separation, and vice-versa.

Measuring separation enhancement and deterioration capacity of a layer: The correlation matrices give an indication of the separation between classes for a given convolutional layer. Comparing $\boldsymbol{C_l}$ and $\boldsymbol{C_{l+1}}$, one can know for which class pairs, the separation increased (correlation decreased) in layer $l+1$, and for which ones, the separation deteriorated (correlation increased) in $l+1$. Similar statistics for layer l can be computed by comparing $\boldsymbol{C_{l-1}}$ and $\boldsymbol{C_l}$. For a convolutional layer l, let the number of class pairs where the separation increased in comparison to layer $l-1$ be n_+^l, and where the separation decreased be n_-^l. Let $n_T = M^2$ denote the total number of class pairs. Note that both stretch and split operations can be simultaneously applied to each layer. n_+^l contributes to the *stretching / widening* of the layer l, while n_-^l contributes to the *symmetric splitting* of its inputs.

In the domain of decision tree training [2], information gain is used to quantify the value a node adds to the classification task. However, in the context of our work, this measure would not enable us to estimate both the split and the stretch factors for the same layer. Thus, we resort to the number of class pairs where the separation increases / decreases to measure the separation enhancement and deterioration capacity of a layer respectively. As we will discuss in the next subsection and Section 4, both the stretch and split operations applied to the same layer helps us to optimally reduce the model size and increase accuracy.

Estimating stretch and split factors: By the definition of n_+^l, for a layer l, we define the average class separation enhancement capability of the subsequent layers by the following expression:

$$\xi(l) = \sum_{i=l+1}^{L-1} (n_+^i \ / \ n_T)/(L-l-1) \tag{1}$$

Note that we omit the last layer L in the above expression. This is discussed at the end of this subsection.

For each l, there can be two cases, (a) $n_+^l < n_-^l$, (b) $n_+^l \geq n_-^l$. **Case (a)** implies that the number of class pairs for which separation decreased were more than for which separation increased. This is not a desired scenario, since with subsequent layers, we would want to gradually increase the separation between classes. Thus, in case (a), we do a symmetric split between $l-1$ and l, i.e. the connections incoming to l undergo a split. This is done under the hypothesis that split should minimize the hindering linkages and thus cause a lesser deterioration in the separation of class pairs in layer l. The amount of split is decided by n_-^l and the average separation enhancement potential of the subsequent layers $\xi(l)$. For example, if the subsequent layers greatly increase the separation between classes, a lesser split should suffice since we do not need to improve the separation potential between layers l and $l{+}1$ by a major extent. Doing a high amount of split in this case may be counterproductive, since the efficient subsequent layers might not then get sufficiently informative features to proceed with. Based on this hypothesis, we arrive at the following equation indicating the split factor r_s^l for convolutional layer l under case(a):

$$r_s^l = 2^{\psi\left(\frac{n_-^l}{n_T} \ \xi(l)\right)} \tag{2}$$

where $\psi(x) = \lfloor x/\lambda \rfloor$. λ is a parameter that controls the amount of reduction in model size. Note that the expression is raised to the power of 2 in (2), meaning that we do splits in multiples of 2. This is done to make the implementation coherent for Caffe's [9] group parameter. The group parameter in Caffe is similar to the symmetric split operation considered here (Fig 1). Although *group* parameter can be any integer, it should exactly divide the number of nodes being split. Since, the number of nodes in architecture layers are typically multiples of 2, we raise the expression to the power of 2 in (2). For case (a), no stretching is performed, since that might lead to more redundancy.

For case (b), the number of class pairs experiencing increased separation are greater than those undergoing deterioration. We aim to stretch the layer as well as split its inputs in such a scenario. The stretch factor is based on n_+^l and the average separation enhancement capability of subsequent layers $\xi(l)$. If $\xi(l)$ is significant, stretching in l is done to a lesser extent indicating that l needs to help but only to a limited extent to avoid overfitting; and vice-versa. We thus arrive at the following equation indicating the stretch factor r_e^l for layer l in case (b):

$$r_e^l = 1 + \phi\left(\frac{n_+^l}{n_T} \ \xi(l)\right) \tag{3}$$

where $\phi(x) = \lfloor x/\lambda \rfloor \ \lambda$ is a function that depends on λ. We add 1 in (3), since a stretch factor of say 1.25 indicates that the number of nodes in the respective layer be increased by a quarter. Note that $\phi(x) = \lambda\psi(x)$. This indicates that for say $\lambda = 0.25$, a split factor of 2 might be roughly equivalent to a stretch factor of 1.25 for enhancing the class separation. This is an empirical choice, which helps us to optimally increase the accuracy and reduce the model size. We will delineate the importance of λ in the next subsection.

In case (b), due to n_-^l, there is also some redundancy in the connections between l and $l-1$. Thus the inputs of layer l also need to be split. The split factor in this case is again decided by (2). The operation of splitting along with stretching helps to reduce the model size while also potentially enhancing the accuracy.

In our approach, we do not consider the refinement of fully connected layers, but only refine the convolutional layers. This is motivated by the fact that in CNNs, convolutional layers are mostly present in high numbers, with fully connected layers being lesser in number. For instance, GoogleNet has only one fully connected layer at the end after 21 convolutional layers. However, since fully connected layers can contain a significant amount of parameters in comparison to convolutional layers (like in AlexNet), considering fully connected layers for architectural refinement can be worth exploring.

Since for a layer, our method considers the change in class separation compared to the previous layer, no stretching or splitting is done for the first convolutional layer since it is preceded by the input layer. Also, we notice that the final convolutional layer in general, enhances the separation for most classes in comparison to the penultimate convolutional layer. Thus, stretching it mostly amounts to overfitting, and so, we exclude the last convolutional layer from

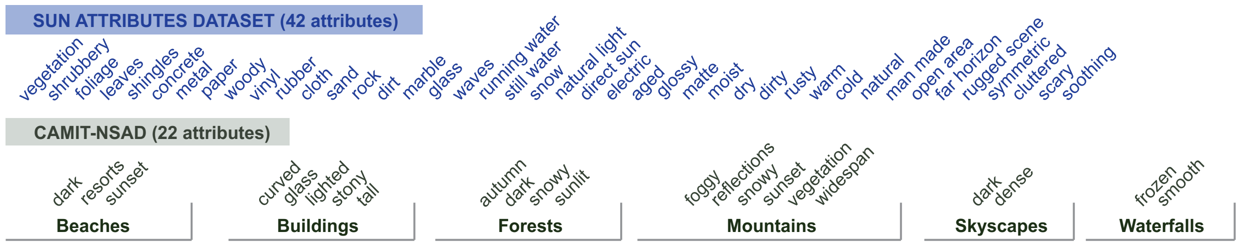


Figure 3. **Datasets and Classes:** The figure shows the classes present in the SUN Attributes Dataset (SAD) as considered in [19] and CAMIT-NSAD dataset [19]. While classes in SAD are purely attributes, classes in CAMIT-NSAD are attribute-noun pairs.

all our analysis. By a similar argument, the last inception unit is omitted from our analysis in GoogleNet.

Once the stretch / split factors are found using a pre-trained architecture, the refined architecture is trained from scratch. Thus, we do not share any weights between the original and the refined architecture. The weight initialization in all cases is done according to [7].

On choice of λ and upper bound: The parameter λ controls the amount of reduction in model size. It is meant to be empirically chosen and the functions $\psi(.), \phi(.)$ have been formulated so that they satisfy our hypotheses mentioned in the previous subsection while taking into account the possible effect of λ. If λ increases, the stretch factors decrease and the splits are smaller (see $\phi(.)$ and $\psi(.)$). If λ decreases, the split factors can be very high along with decent values for stretch factors. However, due to the difference in $\phi(.)$ and $\psi(.)$, the increase in the stretch factor will be limited as compared to the increase in the split factor. This is desired since we do not wish to increase the model size by vast amounts of stretching operations, which may also lead to overfitting. *Hence, with increasing λ, the model size tends to increase, and vice-versa.* For all our experiments, we set an empirically chosen $\lambda = 0.25$.

A natural question to now ask is that what range of values of λ should one try? The lower bound may be empirically chosen based on the maximum split factor that one wishes to support. However, λ can be upper-bounded by a value λ_o above which no split and stretch factor can change. From the definitions of $\psi(.)$ and $\phi(.)$, λ_o can be easily given as follows:

$$\lambda_o = max\left[\left(\frac{n_+^{l'}}{n_T}\ \xi(l')\right), \left(\frac{n_-^{l'}}{n_T}\ \xi(l')\right), \left(\frac{n_-^{l}}{n_T}\ \xi(l)\right)\right] \tag{4}$$

where $n_+^{l'} \geq n_-^{l'}\ \ \forall\, l'; n_+^{l} < n_-^{l}\ \ \forall\, l$.

Refining with GoogleNet: Note that GoogleNet contains various inception modules (Fig 2), each module having two layers with multiple convolutional blocks. While describing our refinement algorithm, wherever we have mentioned the term *convolutional layer*, in context of GoogleNet, it should be considered as a convolutional block. Please see Fig 2 for a better understanding of how do we decide subsequent and previous layers in GoogleNet for refining. Also see Fig 6 for the stretch and split factors obtained after architectural refinement of GoogleNet.

Continuing architectural refinement: Our method refines the architecture of a pre-trained CNN. One can also apply such a procedure to an architecture that has been already refined by our approach. One can stop once no significant difference in the accuracy or model size is noticed for some choices of λ [2].

4. Results and Discussion

Datasets: We evaluate our approach on SUN Attributes Dataset (SAD) [15] and Cambridge-MIT **N**atural **S**cenes **A**ttributes **D**ataset (CAMIT-NSAD) [19]. Both the datasets have classes of natural scenes attributes, whose listing can be found in Fig 3.

The full version of SAD [15] has 102 classes. However, following [19], we discard the classes in the full version of SAD which lie under the paradigm of recognizing activities, poses and colors; and only consider the 42 visual attribute classes, so that the dataset is reasonably homogeneous for our problem. SAD with 42 attributes has 22,084 images for training, 3056 images for validation and 5618 images for testing. Each image in the training set is annotated with only one class label. Each test image has a binary label for each of the 42 attributes, indicating the absence / presence of the respective attribute. In all, the test set contains 53,096 positive labels.

CAMIT-NSAD [19] is a natural scenes attributes dataset containing classes as attribute noun pairs, instead of just attributes. CAMIT-NSAD has 22 attributes and contains 46,008 training images, with at least 500 images for each attribute-noun pair. The validation set and the test set contain 2104 and 2967 images respectively. While each training image is annotated with only one class label, the test

[2] Although our approach does not induce a concrete optimization objective from the correlation analysis, we believe that it is a step towards solving the deep architecture learning problem, and furthers the related works towards more principled directions. The intuition behind our method was established from various experiments, done on diverse datasets with a variety of shallow and deep CNNs.

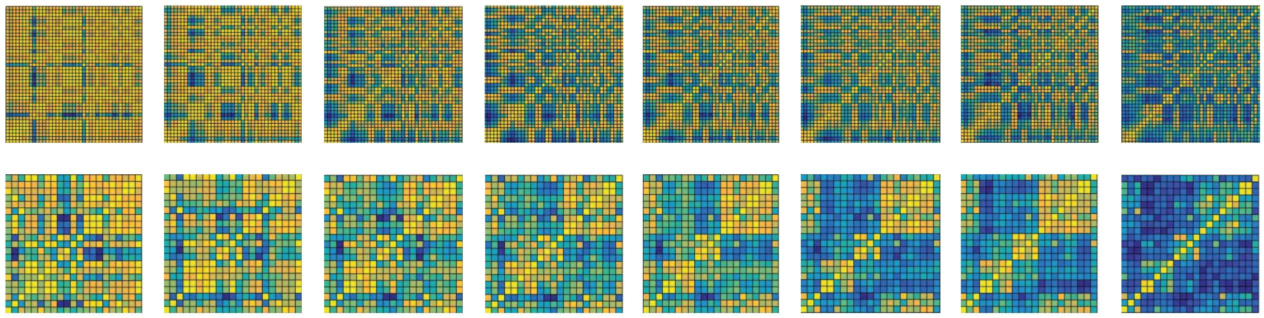


Figure 4. **Correlation Matrices for 8 Convolutional Layers of VGG-11 trained on SAD and CAMIT-NSAD :** Traversed row-wise, correlation matrices $\boldsymbol{C_l} : l \in \{1, \ldots, 8\}$ are shown. Dark Blue color indicates minimum correlation between classes, while a bright yellow color indicates maximum correlation. Thus, all diagonals are bright yellow, since each class is maximally correlated with itself. For each matrix, the attribute classes are ordered as in Fig 3 seen left to right. Note that more correlation implies lesser separation and vice-versa. ***Top Row (SAD)*** **:** The lower layers can separate the classes better as compared to deeper layers. ***Bottom Row (CAMIT-NSAD)*****:** The classes are separated lesser in lower layers and more prominently in deeper layers. This is mainly because classes in SAD are purely attributes, while classes in CAMIT-NSAD are attribute-noun pairs. Due to this distinction, the two datasets have nearly contrasting characteristics which pose a challenge to the architectural refinement problem. For instance, *snow* class in SAD can be separated from *dirt* class mostly by the distinction of white and brown colors; while in CAMIT-NSAD, the class of *snowy forests* cannot be separated from *snowy mountains* just by noticing the color difference, since both forests and mountains are snowy. Infact, in this case, the separation is most likely to appear in deeper layers where the distinction is also made between forests and mountains. The above explanation is made under the widely accepted notion that a CNN learns low-level type features (edges, color patterns, etc.) in lower layers, and more class-specific features in deeper layers [24]. Also note that some classes in the datasets have a natural correlation, e.g. classes of *vegetation*, *shrubbery*, *foliage* and *leaves* in SAD are well correlated, since the presence of leaves is very likely where some vegetation occurs. As a result, separation between these classes may always be low as compared to separation between the classes of *vegetation* and *running water*. A similar analysis can be made for CAMIT-NSAD. *Figure is best viewed in color.*

images contain binary labels for each of the 22 attributes. In all, the test set contains 8517 positive labels. All images in SAD and CAMIT-NSAD are 256×256 RGB.

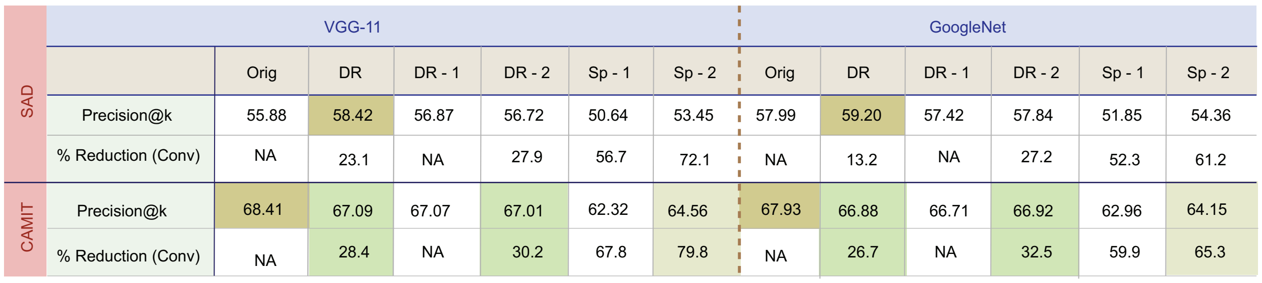
It can be seen that classes in SAD are pure attributes, while that in CAMIT-NSAD are noun-attribute pairs. Due to this distinction, the two datasets have different characteristics which make them challenging for the problem of architectural refinement. Please see Fig 4 for a better understanding of this distinction.

Notice that classes in CAMIT-NSAD can be finally separated to a greater extent as compared to classes in SAD. This is because almost each class in SAD has a variety of outdoor and indoor scenes, since an attribute can exist for both. For instance, both an outdoor and indoor scene can be *glossy* as well as can have *direct sunlight*. However, with noun-attribute pairing as in CAMIT-NSAD, the classes are more specifically defined, and thus significant separation between a greater number of class pairs is achieved at the end.

Choice of Datasets: The choice of the datasets used for evaluation needs a special mention. We chose attribute datasets, since given the type of labels here, it is difficult to establish where should the model parameters be reduced / increased. This we found was in contrast to *object* recognition datasets such as ImageNe, where we observed that refining an architecture by symmetric splitting in the first few layers could increase accuracy. However, we thought this to be very intuitive, since *objects* are generally encoded in deeper layers, and thus, one would expect to reduce parameters in the top layers. We thus evaluate our procedure with the types of datasets, where one cannot easily decide which network layers contribute to the class labels.

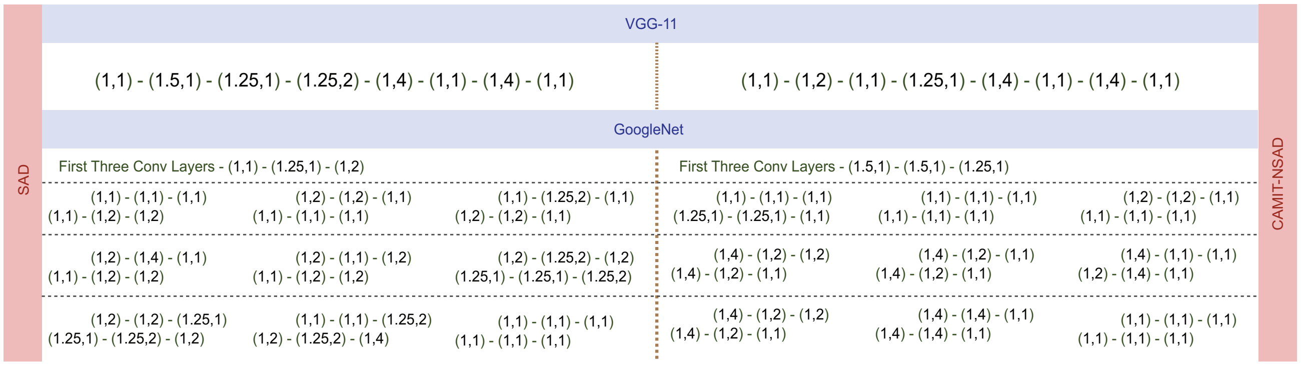
Architectures for Refinement: We choose GoogleNet [21] and VGG-11 [20] as the base CNN architectures, which we intend to alter using our approach. Since GoogleNet and VGG-11 are quite contrasting in their construction, they together pose a considerable challenge to our architectural refinement algorithm. While VGG-11 (which can be loosely considered as a deeper form of AlexNet [10]) has 8 convolutional layers, and 3 fully connected layers, GoogleNet is a 22-layer deep net having 9 inception units after three convolutional layers, and a fully connected layer before the final output. Each inception unit has 6 convolutional blocks arranged in 2 layers. We refer the reader to [21] and [20] for complete details of GoogleNet and VGG-11 respectively. An instance of inception module in GoogleNet is shown in Fig 2.

Baselines: We consider the following baselines to compare with our proposed approach. ***(a) Our approach with only stretching and no splitting*** - We consider refinement with only the stretching operation and no splitting operation, i.e. the CNN architecture is refined by only stretching some layers, but no symmetric splitting between layers is done. This proves the importance of stretching operation for architectural refinement. ***(b) Our approach with only splitting and no stretching*** - Evaluating by only considering the symmetrical split operation and no stretch operation



		VGG-11						GoogleNet					
		Orig	DR	DR - 1	DR - 2	Sp - 1	Sp - 2	Orig	DR	DR - 1	DR - 2	Sp - 1	Sp - 2
SAD	Precision@k	55.88	58.42	56.87	56.72	50.64	53.45	57.99	59.20	57.42	57.84	51.85	54.36
	% Reduction (Conv)	NA	23.1	NA	27.9	56.7	72.1	NA	13.2	NA	27.2	52.3	61.2
CAMIT	Precision@k	68.41	67.09	67.07	67.01	62.32	64.56	67.93	66.88	66.71	66.92	62.96	64.15
	% Reduction (Conv)	NA	28.4	NA	30.2	67.8	79.8	NA	26.7	NA	32.5	59.9	65.3

Figure 5. **Results and Comparisons :** The figure shows results on SAD and CAMIT-NSAD with VGG-11 and GoogleNet using our approach and various other baselines. *Orig* = Original architecture, *DR* = Deep Refined Architecture (our approach), *DR-1* = Deep Refined Architecture with only the Stretch Operation, *DR-2* = Deep Refined Architecture with only the Symmetric Split Operation, *Sp-1* = L1 Sparsified network, *Sp-2* - Sparsified network with [11]. We use *precision@k* as our performance metric, and report that here as a percentage. For SAD, k = 21, while for CAMIT-NSAD, k = 7. Please refer text for details on this. We report the reduction percentage in the parameters of the convolutional layers in comparison to the original architecture. Thus, reporting reduction in model size for $Orig$ is not applicable. Also, since *DR-1* only does a stretching operation over the original architecture, it is bound to increase the model size, and thus % reductions in the model size are not applicable here as well. Note that *DR* performs significantly well for SAD giving a decent reduction in model size with impressive increase in precision. For CAMIT-NSAD, *DR* does not improve the precision of the original architecture. The results here are reported for $\lambda = 0.25$. For CAMIT-NSAD, we also did experiments for higher values of λ but we did not see any increase in precision; rather with increased λ, we got lesser reduction in model size as expected (Section 3) . Nevertheless, for CAMIT-NSAD, *DR* and *DR-2* could produce architectures with a reduced model size producing precision better than the state-of-the-art sparsification techniques of *Sp-1* and *Sp-2*.



	SAD			CAMIT-NSAD		
VGG-11	(1,1) - (1.5,1) - (1.25,1) - (1.25,2) - (1,4) - (1,1) - (1,4) - (1,1)			(1,1) - (1,2) - (1,1) - (1.25,1) - (1,4) - (1,1) - (1,4) - (1,1)		
GoogleNet	First Three Conv Layers - (1,1) - (1.25,1) - (1,2)			First Three Conv Layers - (1.5,1) - (1.5,1) - (1.25,1)		
	(1,1) - (1,1) - (1,1) (1,1) - (1,2) - (1,2)	(1,2) - (1,2) - (1,1) (1,1) - (1,1) - (1,1)	(1,1) - (1.25,2) - (1,1) (1,2) - (1,2) - (1,1)	(1,1) - (1,1) - (1,1) (1.25,1) - (1.25,1) - (1,1)	(1,1) - (1,1) - (1,1) (1,1) - (1,1) - (1,1)	(1,2) - (1,2) - (1,1) (1,1) - (1,1) - (1,1)
	(1,2) - (1,4) - (1,1) (1,1) - (1,2) - (1,2)	(1,2) - (1,1) - (1,2) (1,1) - (1,2) - (1,2)	(1,2) - (1.25,2) - (1,2) (1.25,1) - (1.25,1) - (1.25,2)	(1,4) - (1,2) - (1,2) (1,4) - (1,2) - (1,1)	(1,4) - (1,2) - (1,1) (1,4) - (1,2) - (1,1)	(1,4) - (1,1) - (1,1) (1,2) - (1,4) - (1,1)
	(1,2) - (1,2) - (1.25,1) (1.25,1) - (1.25,2) - (1,2)	(1,1) - (1,1) - (1.25,2) (1,2) - (1.25,2) - (1,4)	(1,1) - (1,1) - (1,1) (1,1) - (1,1) - (1,1)	(1,4) - (1,2) - (1,2) (1,4) - (1,2) - (1,1)	(1,4) - (1,4) - (1,1) (1,4) - (1,4) - (1,1)	(1,1) - (1,1) - (1,1) (1,1) - (1,1) - (1,1)

Figure 6. **Refined Architectures obtained with our approach:** Left column shows the refined architectures for SAD, and the right column for CAMIT-NSAD. The corresponding precision results are reported in Fig 5 under the column *DR*. Each tuple (a, b) indicates that a is the stretch factor for the convolutional layer/block, while b is the split factor for the input of that convolutional layer / block. Entry of $(1, 1)$ implies no stretch and splitting should be done. In Fig 5, for *DR-1* , every value of b is made 1, while for *DR-2* , every value of a is made 1. VGG-11 contains 8 convolutional layers, for which the factors are shown. In Googlenet, the factors for first three convolutional layers are shown in the first row (under GoogleNet). After that , each row under GoogleNet contains the factors of the convolutional blocks in the inception unit of Fig 2. Traversed row-wise, inception units correspond to the ordering 3a, 3b, 4a, 4b, 4c, 4d, 4e, 5a, 5b of GoogleNet architecture [21]. Note that since we do not consider the last convolutional layer (that is connected to the fully connected layer) in our analysis, all factors for that are 1 in VGG-11. A similar argument exists for the last inception unit in GoogleNet.

provides evidence to the utility of splitting. ***(c) L1 Sparsification*** - We consider the L1 sparsification of a CNN as one of the important baselines. Here, the weights (parameters) of a CNN are regularized with an L1 norm, and the regularization term is added to the loss function. Due to the L1 norm, this results in a sparse CNN, i.e. a CNN with a reduced model size. Following [11], all the parameters with values less than or equal to *1e-4* are made zero both during training and testing. This not only ensures maximal sparsity, but also stabilizes the training procedure resulting in better convergence. ***(d) Sparsification with the method of [11]*** - The method of [11] combines the low-rank decomposition [8] and L1 sparsification techniques for better sparsity. However, they mention that the critical step in achieving comparable accuracy with high amount of sparsity, is minimizing the loss function along with L1 and L2 regularization terms upon the weights of the CNN. Low-rank decomposition can increase sparsity with a further decrease in accuracy. Since, in this work, we are interested in an optimal trade-off between accuracy and model size, we evaluate the method of [11] without the low-rank decomposition. This ensures that we obtain maximum possible accuracy with [11] at the expense of some reduced sparsity. ***(e) Original architecture:*** We consider the original architecture without any architectural refinement and sparsification techniques applied. The amount of reduction achieved in the model size with our approach and other baselines along with the recognition performance is compared with this baseline.

Note that for all the above mentioned baselines, the CNN is first trained on the respective dataset with the standard minimization of softmax loss function [9], after which a second training step is done. For baselines (a) and (b), the re-training step is performed on the refined architecture as described in Section 3; while for baselines (c) and (d), re-training is done as a fine-tuning step, where the learning rate of the output layer is increased to 5 times the learning rate of all other layers.

Other plausible baselines: We also tried randomly splitting and stretching throughout the network as a plausible baseline. Here although in some cases, we could reduce the model size by almost similar amounts as our proposed approach, significantly higher accuracy was consistently achieved using our method.

Training: For all datasets and CNN architectures, the networks are trained using the Caffe library [9]. The pre-training step is always performed with the standard softmax loss function [9]. For all the pre-training, refinement and baseline cases, batch size of 32 samples is considered. An *adaptive step policy* is followed during training, i.e. if the change in validation accuracy over a range of 5 consecutive epochs is less than 0.5, the learning rate is reduced by a factor of 10. For SAD, we start with an initial learning rate of 0.01 for both GoogleNet and VGG-11, while for CAMIT-NSAD, a starting learning rate of 0.001 suffices for both the architectures. In all cases, we train for 100 epochs.

Testing: Given a trained CNN, we need to predict multiple labels for each test image in SAD and CAMIT-NSAD. We use *precision@k* as our performance metric. The metric is normally chosen when one needs to predict top-k labels for a test image. Since, our ground-truth annotations contain only binary labels for each class, for a given test image, we cannot sort the labels according to their degree of presence. We thus decide k for each dataset as the maximum number of positive labels present for any image in the test set. For SAD, k is 21, while for CAMIT-NSAD, k is 7. Thus, given a test image, we predict the output probabilities of each class using the trained net, and sort these probabilities in the descending order to produce a vector $\boldsymbol{T}$. If that test image has say 5 positive labels in ground-truth annotations, we expect the first 5 entries of $\boldsymbol{T}$ to correspond to the same labels for a 100% precision. We thus compute the true positives and false positives over the entire test set and report the final precision. This is in line with the test procedure followed by [19].

Discussion of results: Fig 5 shows the precision and model size reduction results obtained with our approach and the baselines, for both the datasets and both the architectures. For understanding intrinsic details of the refined architectures, please refer to Fig 6 and Fig 2. It is clear that for SAD, our approach for both VGG-11 and GoogleNet, offers an increase in original precision while giving a reasonable reduction in model size. The reduction in the number of parameters in convolutional layers holds more importance here, since our method was only applied to the convolutional layers. It is interesting to note from the results on SAD, that the predicted combination of stretch and split is more optimal as compared to only having the split or the stretch operation. This also shows that stretching alone is not always bound to enhance the precision, since it may lead to overfitting. In all cases, the sparsification baselines fall behind the precision obtained with our approach, although they produce more sparsity.

The results on CAMIT-NSAD present a different scenario. Note that our approach is not able to enhance the precision in this case, but decreases the precision by a small amount, while giving decent reduction in model size. However, the precision obtained with a reduced model size by using our approach is still greater than the one obtained by other baseline sparsification methods, though at the expense of lesser sparsity. The inability to increase precision in this case can be attributed to the fact that our approach is greedy, i.e. it estimates the stretch and split factors for every layer, and not jointly for all layers. This affects CAMIT-NSAD since the classes are attribute-noun pairs, and attribute-specific information and noun-specific information are encoded at different layers, which need to be considered together for refinement.

Note that a single metric jointly quantifying both the accuracy increase and the model size reduction is difficult to formulate. In cases where we increase the accuracy as well as decrease the model size (SAD), we offer a win-win situation. However, in cases where we decrease the model size but cannot increase the accuracy (CAMIT-NSAD), we believe that the model choice depends on user's requirements, and our method provides an additional and plausibly a useful alternative for the user, and can also complement the other approaches. One can obtain an architecture using our approach, and then apply a sparsification technique like [11] in case the user's application demands maximum sparsity, and not that good a precision.

5. Conclusion

We have introduced a novel strategy that alters the architecture of a given CNN for a specified dataset for effecting a possible increase in original accuracy and reduction of parameters. Evaluation on two challenging datasets shows its utility over relevant baselines.

References

[1] C. Cortes, M. Mohri, and U. Syed. Deep boosting. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1179–1187, 2014. 3

[2] A. Criminisi, J. Shotton, and E. Konukoglu. *Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning*. Now, 2012. 4

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 2

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE, 2009. 1, 2

[5] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013. 3

[6] B. Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014. 3

[7] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015. 5

[8] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 1, 3, 7

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 2, 3, 4, 8

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012. 1, 2, 6

[11] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015. 1, 3, 7, 8

[12] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. *arXiv preprint arXiv:1411.7766 (To apeear in ICCV)*, 2015. 1, 2

[13] A. Oliva, A. Torralba, et al. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 155, 2006. 2

[14] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1717–1724. IEEE, 2014. 2

[15] G. Patterson and J. Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In *CVPR*, pages 2751–2758. IEEE, 2012. 1, 5

[16] Z. Peng, L. Lin, R. Zhang, and J. Xu. Deep boosting: Layered feature mining for general image classification. In *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, pages 1–6. IEEE, 2014. 3

[17] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE, 2013. 3

[18] S. Shalev-Shwartz. Selfieboost: A boosting algorithm for deep learning. *arXiv preprint arXiv:1411.3436*, 2014. 3

[19] S. Shankar, V. K. Garg, and R. Cipolla. Deep-carving: Discovering visual attributes by carving deep neural nets. In *CVPR*, June 2015. 1, 2, 3, 5, 8

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 6

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 1, 2, 3, 6, 7

[22] L. Torrey and J. Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 1:242, 2009. 2

[23] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH*, pages 2365–2369, 2013. 3

[24] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014. 6

[25] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014. 3

[26] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems*, pages 487–495, 2014. 1, 2

[27] J. T. Zhou, S. J. Pan, I. W. Tsang, and Y. Yan. Hybrid heterogeneous transfer learning through deep learning. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. 2