ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200209

Classifying the Classifier: Dissecting the Weight Space of Neural Networks

Gabriel Eilertsen[✝], **Daniel Jönsson**[✝], **Timo Ropinski**[✧], **Jonas Unger**[✝] and **Anders Ynnerman**[✝]

Abstract. This paper presents an empirical study on the weights of neural networks, where we interpret each model as a point in a high-dimensional space – the *neural weight space*. To explore the complex structure of this space, we sample from a diverse selection of training variations (dataset, optimization procedure, architecture, etc.) of neural network classifiers, and train a large number of models to represent the weight space. Then, we use a machine learning approach for analyzing and extracting information from this space. Most centrally, we train a number of novel *deep meta-classifiers* with the objective of classifying different properties of the training setup by identifying their footprints in the weight space. Thus, the meta-classifiers probe for patterns induced by hyper-parameters, so that we can quantify how much, where, and when these are encoded through the optimization process. This provides a novel and complementary view for explainable AI, and we show how meta-classifiers can reveal a great deal of information about the training setup and optimization, by only considering a small subset of randomly selected consecutive weights. To promote further research on the weight space, we release the *neural weight space* (NWS) dataset – a collection of 320K weight snapshots from 16K individually trained deep neural networks.

1 Introduction

The complex and non-linear nature of deep neural networks (DNNs) makes it difficult to understand how they operate, what features are used to form decisions, and how different selections of hyper-parameters influence the final optimized weights. This has led to the development of methods in explainable AI (XAI) for visualizing and understanding neural networks, and in particular for convolutional neural networks (CNNs). Thus, many methods are focused on the input image space, for example by deriving images that maximize class probability or individual neuron activations [42, 48]. There are also methods which directly investigate neuron statistics of different layers [29], or use layer activations for information retrieval [26, 1, 37]. However, these methods primarily focus on local properties, such as individual neurons and layers, and an in-depth analysis of the full set of model weights and the weight space statistics has largely been left unexplored.

In this paper, we present a dissection and exploration of the *neural weight space* (NWS) – the space spanned by the weights of a large number of trained neural networks. We represent the space by training a total of 16K CNNs, where the training setup is randomly sampled from a diverse set of hyper-parameter combinations. The performance of the trained models alone can give valuable information when related to the training setups, and suggest optimal combinations of hyper-parameters. However, given its complexity, it is difficult to directly reason about the sampled neural weight space, e.g. in terms of Euclidean or other distance measures – there is a large number of symmetries in this space, and many possible permutations represent models from the same equivalence class. To address this challenge, we use a machine learning approach for discovering patterns in the weight space, by utilizing a set of meta-classifiers. These are trained with the objective of predicting the hyper-parameters used in optimization. Since each sample in the hyper-parameter space corresponds to points (models) in the weight space, the meta-classifiers seek to learn the inverse mapping from weights to hyper-parameters. This gives us a tool to directly reason about hyper-parameters in the weight space. To enable comparison between heterogeneous architectures and to probe for local information, we introduce the concept of local meta-classifiers operating on only small subsets of the weights in a model. The accuracy of a local meta-classifier enables us to quantify where differences due to hyper-parameter selection are encoded within a model.

We demonstrate how we can find detailed information on how optimization shapes the weights of neural networks. For example, we can quantify how the particular dataset used for training influences the convolutional layers stronger than the deepest layers, and how initialization is the most distinguishing characteristic of the weight space. We also see how weights closer to the input and output of a network faster diverge from the starting point as compared to the "more hidden" weights. Moreover, we can measure how properties in earlier layers, e.g. the filter size of convolutional layers, influence the weights in deeper layers. It is also possible to pinpoint how individual features of the weights influence a meta-classifier, e.g. how a majority of the differences imposed on the weight space by the optimizer are located in the bias weights of the convolutional layers. All such findings could aid in understanding DNNs and help future research on neural network optimization. Also, since we show that a large amount of information about the training setup could be revealed by meta-classifiers, the techniques are important in privacy related problem formulations, providing a tool for leaking information on a black-box model without any knowledge of its architecture.

In summary, we present the following set of contributions:

- We use the neural weight space as a general setting for exploring properties of neural networks, by representing it using a large number of trained CNNs.
- We release the neural weight space (NWS) dataset, comprising 320K weight snapshots from 16K individually trained nets, together with scripts for training more samples and for training meta-classifiers[1].

[✝]Department of Science and Technology, Linköping University, Sweden
[✧]Institute of Media Informatics, Ulm University, Germany

[1] https://github.com/gabrieleilertsen/nws

- We introduce the concept of neural network meta-classification for performing a dissection of the weight space, quantifying how much, where and when the training hyper-parameters are encoded in the trained weights.
- We demonstrate how a large amount of information about the training setup of a network can be revealed by only considering a small subset of consecutive weights, and how hyper-parameters can be practically compared across different architectures.

We see our study as a first step towards understanding and visualizing neural networks from a direct inspection of the weight space. This opens up new possibilities in XAI, for understanding and explaining learning in neural networks in a way that has previously not been explored. Also, there are many other potential applications of the sampled weight space, such as learning-based initialization and regularization, learning measures for estimating distances between networks, learning to prevent privacy leakage of the training setup, learning model compression and pruning, learning-based hyper-parameter selection, and many more.

Throughout the paper we use the term *weights* denoted θ to describe the trainable parameters of neural nets (including bias and batch normalization parameters), and the term *hyper-parameters* denoted ϕ to describe the non-trainable parameters. For full generality we also include dataset choice and architectural specifications under the term hyper-parameter.

2 Related work

Visualization and analysis: A significant body of work has been directed towards explainable AI (XAI) in deep learning, for visualization and understanding of different aspects of neural networks [19], e.g. using methods for neural feature visualization. These aim at estimating the input images to CNNs that maximize the activation of certain channels or neurons [42, 48, 31, 46, 40], providing information on what are the salient features picked up by a CNN. Another interesting viewpoint is how information is structured by neural networks. It is, for example, possible to define interpretability of the learned representations, with individual units corresponding to unique and interpretative concepts [50, 4, 49]. It has also been demonstrated how neural networks learn hierarchical representations [6].

Many methods rely on comparing and embedding DNN layer activations. For example, activations generated by multiple images can be concatenated to represent a single trained model [9], and activations of a large number of images can be visualized using dimensionality reduction [38]. The activations can also be used to regress the training objective, measuring the level of abstraction and separation of different layers [1], and for measuring similarity between different trainings and layers [26, 37], e.g. to show how different layers of CNNs converge. In contrast to these methods we operate directly on the weights, and we explore a very large number of trainings from heterogeneous architectures.

There are also previous methods which consider the model weights, e.g. in order to visualize the evolution of weights during training [12, 13, 27, 30, 2, 11]. Another common objective is to monitor the statistics of the weights during training, e.g. using tools such as Tensorboard. While these consider one or few models, our goal is to compare a large number of different models and learn how optimization encodes the properties of the training setup within the model weights.

By training a very large number of fully connected networks, Novak et al. study how the sensitivity to model input correlates with generalization performance [36]. For different combinations of base-training and fine-tuning objectives, Zamir et al. quantify the transfer-learning capability between tasks [47]. Yosinski et al. explore different configurations of pre-training and fine-tuning to evaluate and predict performance [45]. However, while monitoring the accuracy of many trainings is conceptually similar to our analysis in Section 4.2, our main focus is to search for information in the weights of all the trained models (Section 5).

Privacy and security: Related to the meta-classification described in Section 5 are the previous works aiming at detecting privacy leakage in machine learning models. Many such methods focus on member inference attacks, attempting to estimate if a sample was in the training data of a trained model, or generating plausible estimates of training samples [10, 18, 34]. Ateniese et al. train a set of models using different training data statistics [3]. The learned features are used to train a meta-classifier to classify the characteristics of the original training data. The meta-classifier is then used to infer properties of the non-disclosed training data of a targeted model. The method was tested on Hidden Markov Models and Support Vector Machines. A similar concept was used by Shokri et al. for training an attack model on output activations in order to determine whether a certain record is used as part of the model's training dataset [41]. Our dissection using meta-classifiers can reveal information about training setup of an unknown DNN, provided that the trained weights are available; potentially with better accuracy than previous methods since we learn from a very large set of trained models.

Meta-learning: There are many examples of methods for learning optimal architectures and hyper-parameters [21, 32, 5, 51, 39, 52, 28]. However, these methods are mostly focused on evolutionary strategies to search for the optimal network design. In contrast, we are interested in comparing different hyper-parameters and architectural choices, not only to gain knowledge on advantageous setups, but primarily to explore how training setup affects the optimized weights of neural networks. To our knowledge there has not been any previous large-scale samplings of the weight space for the purpose of learning-based understanding of deep learning.

3 Weight space representation

We consider all weights of a model as being represented by a single point in the high-dimensional neural weight space (NWS). To create such a representation from the mixture of convolutional filters, biases, and weight matrices of fully-connected (FC) layers, we use a vectorization operation. The vectorization is performed layer by layer, followed by concatenation. For the ith convolutional layer, all the filter weights $\mathbf{h}_{i,j}$ from filters $j = 1, ..., K$ are concatenated, followed by bias weights $\mathbf{b}_i$,

$$\theta_i = \theta_{i-1} \shortparallel \text{vec}(\mathbf{h}_{i,1}) \shortparallel ... \shortparallel \text{vec}(\mathbf{h}_{i,K}) \shortparallel \mathbf{b}_i, \tag{1}$$

where $\text{vec}(\cdot)$ denotes vectorization and $\mathbf{v}_1 \shortparallel \mathbf{v}_2$ concatenates vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. For the FC layers, the weight matrices $\mathbf{W}_i$ are added using the same scheme,

$$\theta_i = \theta_{i-1} \shortparallel \text{vec}(\mathbf{W}_i) \shortparallel \mathbf{b}_i. \tag{2}$$

Starting with the empty set $\theta_0 = \emptyset$, and repeating the vectorization operations for all L layers, we arrive at the final weight vector $\theta = \theta_L$. For simplicity, we have not included indices over the 2D convolutional filters $\mathbf{h}$ and weight matrices $\mathbf{W}$ in the notation. Moreover, additional learnable weights, e.g. batch normalization parameters, can simply be added after the biases of each layer. Although

Table 1: Hyper-parameter collection used for random selection of training setup. The convolutional layer width specifies the number of channels of the last convolutional layer, while the FC layer width is the size of the first FC layer. All dataset samples are resized to 32×32×3 pixels. Note that we use the term hyper-parameters to also describe more fundamental choices, such as dataset and architecture. The circled options are used to train the fixed architecture models C_{fixed} in Table 2, where the remaining hyper-parameters are randomly selected.

Parameter	Values
Dataset	MNIST [25], CIFAR-10 [24], SVHN [35], STL-10 [8], Fashion-MNIST [44]
Learning rate	$0.0002 - 0.005$
Batch size	$32, 64, 128, 256$
Augmentation	Off, On
Optimizer	ADAM [23], RMSProp [17], Momentum SGD
Activation	ReLU [33], ELU [7], Sigmoid, TanH
Initialization	Constant, Random normal, Glorot uniform, Glorot normal [14]
Conv. filter size	$3, (5), 7$
# of conv. layers	$(3), 4, 5$
# of FC layer	$(3), 4, 5$
Conv. layer width	$16, (32), 48$
FC layer width	$64, (128), 192$

the vectorization may rearrange the 2D spatial information e.g. in convolutional filters, there is still spatial structure in the 1D vector. We recognize that there is ample room for an improved NWS representation, e.g. accounting for weight space permutations and the 2D nature of filters. However, we focus on starting the exploration of the weight space with a representation that is as simple as possible, and from the results in Section 5 we will see that it is possible to extract a great deal of useful information from the vectorized weights.

4 The NWS dataset

In this section, we describe the sampling of the NWS dataset. Then, we show how we can correlate training setup with model performance by regressing the test accuracy from hyper-parameters.

4.1 Sampling

To generate points in the NWS, we train CNNs by sampling a range of different hyper-parameters, as specified in Table 1. For each training, the hyper-parameters are selected randomly, similarly to previous techniques for hyper-parameter search [5]. It is difficult to estimate the number of SGD steps needed for optimization, since this varies with most of the hyper-parameters. Instead, we rely on early stopping to define convergence. For each training we export weights at 20 uniformly sampled points along the optimization trajectory. In order to train and manage the large number of models, we use relatively small CNNs automatically generated based on the architectural hyper-parameters shown in Table 1. The architectures are defined by 6-10 layers, and in total between ∼20K and ∼390K weights each. They follow a standard design, with a number of convolutional layers and 3 max-pooling layers, followed by a set of fully-connected (FC) layers. The loss is the same for all trainings, specified by cross-entropy. For regularization, all models use dropout [43] with 50% keep probability after each fully connected layer. Moreover, we use batch normalization [22] for all trainings and layers. Without the normalization, many of the more difficult hyper-parameter settings do not converge (the supplementary material contains a discussion around this).

Table 2: The *neural weight space* (NWS) dataset used throughout the paper. We refer to the text and the supplementary material for a description of the capturing process.

Name	Description	Quantity
C_{main}	Random hyper-parameters (including architecture)	13K (10K/3K train/test)
C_{fixed}	Random hyper-parameters (fixed architecture)	3K (2K/1K train/test)

We conduct 13K separate trainings with random hyper-parameters, and 3K trainings with fixed architecture for the purpose of global meta-classification (Section 5.2). Table 2 lists the resulting NWS datasets used throughout the paper. For detailed explanations of the training setup, and extensive training statistics (convergence, distribution of test accuracy, training time, model size, etc.), we refer to the supplementary material.

4.2 Regressing the test accuracy

To gain an understanding on the influence of the different hyper-parameters in Table 1, we regress the test accuracy of the sampled set of networks. Given the hyper-parameters ϕ, we model the test accuracy with a linear relationship,

$$\hat{a}(\phi) = \tau_0 + \sum_{o \in \Omega_O} \tau_o \phi_o + \sum_{c \in \Omega_C} \sum_{i=1}^{K_c} \tau_{c,i} [\phi_c = i], \qquad (3)$$

where τ are the model coefficients and K_c is the number of categories for hyper-parameter ϕ_c. Ω_O is the set of ordered hyper-parameters, and Ω_C is the set of categorical hyper-parameters. The categorical hyper-parameters are split into one binary for each category, as denoted by the Iverson bracket, $[\phi_c = i]$. We fit one linear model for each dataset, which means that we have in total 10 categorical and 1 ordered (learning rate) parameters for each model (see Table 1). Although some of the categorical parameters actually are ordered (batch size, filter size, etc.), we split these to fit one descriptive correlation for each of the categories. In total we have 32 categorical, 1 ordered and 1 constant coefficient, so that the size of the set $\{\tau_0, \tau_o, \tau_{c,i}\}$ is 34.

The distribution of test accuracies on a particular dataset shows two modes; one with successful trainings and one with models that do not learn well. This makes it difficult to fit a linear model to the test accuracy. Instead, we focus only on the mode of models that have learned something useful, by rejecting all trainings with test accuracy lower than a certain threshold in-between the two modes. This sampling reduces the set C_{main} from 13K to around 10.5K. Further, for each dataset, we normalize the test accuracy to have zero mean and unit variance. Thus, a positive model coefficient explains a positive effect on the test accuracy and vice versa, and the magnitudes are similar between different datasets. The results of fitting the model to one dataset, CIFAR-10, is displayed in Figure 1.

On average, the single most influential parameter is the initialization, followed by activation function and optimizer, and it is clear

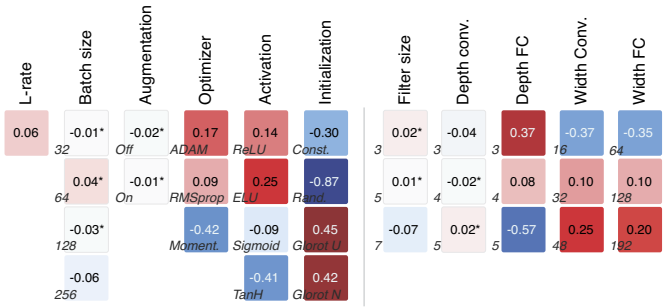

Figure 1: Regression of test accuracy from hyper-parameters on CIFAR-10. Columns represent different hyper-parameters, and the rows categories, as listed in Table 1. * denotes a p-value that is larger than 5%. For results from all datasets, we refer to the supplementary material.

how large impact modern techniques have had on optimization (such as ADAM optimizer, ReLU/ELU activation and Glorot initialization). For architecture specific parameters, a general trend is to promote wider models. However, the number of FC layers has an overall negative correlation. This can be explained by many layers being more difficult to optimize, so that performance suffers when less effective optimizer and initialization is used. Finally, we recognize that a linear model is only explaining some of the correlations, but it helps in forming an overall understanding of the hyper-parameters.

5 Meta-classification

The objective of a meta-classifier is to learn the mapping $g : \theta \rightarrow \phi_c$, i.e. to estimate a specific hyper-parameter ϕ_c from a weight vector θ. The performance of the $g(\theta)$ prediction of ϕ_c gives us a notion for comparing different θ in terms of hyper-parameters. We first give a motivation and definition, followed by examples of global and local meta-classification using meta-classifiers of different complexities.

5.1 Motivation and definition

For a model $f(\theta, x)$, trained using hyper-parameters ϕ, what is specific about the learned weights θ comparing different ϕ? For example, given two sets of weights $\Theta_a = \{\theta_{a,1}, ..., \theta_{a,N}\}$ and $\Theta_b = \{\theta_{b,1}, ..., \theta_{b,M}\}$, trained using different hyper-parameters ϕ_a and ϕ_b, respectively, we expect the weights to converge to different locations in the weight space. However, it is difficult to relate to or reason about these locations based on direct inspection of the weights. For example, the Euclidean inter-distance $||\theta_{a,i} - \theta_{b,j}||$ may very well be smaller than the intra-distance $||\theta_{a,i} - \theta_{a,j}||$, due to the complicated and permutable structure of the weight space. In order to find the decision boundary between Θ_a and Θ_b, we can instead learn it from a large number of samples N and M, using a model $g(\theta)$. The model can thus be used to determine in which region (related to this decision boundary) a new weight sample θ is located. This gives us a notion of quantifying how much of ϕ_a or ϕ_b is encoded in θ.

Given a CNN classifier $f(\theta, x)$, parameterized by the trainable weights θ, and operating on image samples x, a global meta-classifier is described by $g_c(\tau, \theta)$, where θ are static samples of the weight space and τ is the model parameterization. The objective of g_c is to perform classification of the hyper-parameters ϕ_c as shown in Table 1, to determine e.g. which dataset was used in training, if augmentation was performed, or which optimizer was used. g_c takes the vectorized weights θ as input (see Section 3).

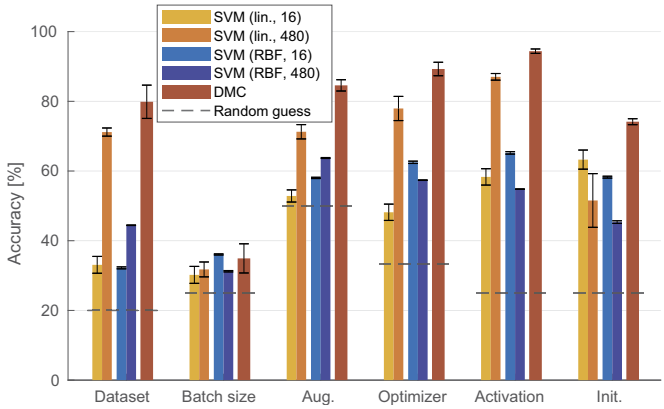

Figure 2: Global meta-classification performance for different hyper-parameters on the C_{fixed} dataset (Table 1). The numbers of the SVM classifiers specify the number of features used. Performance of random guessing is included for reference. The errorbars show standard deviations over 10 independent training rounds.

Models: We consider feature-based meta-classification, as well as deep models applied on the raw weight input. The features are specified from 8 different statistical measures of a weight vector: mean, variance, skewness (third standardized moment), and five-number summary (1, 25, 50, 75 and 99 percentiles). The measures are applied both directly on the weights θ and on weight gradients $\nabla\theta$, for a total of 16 features. The features are used for training support vector machines (SVMs), testing both linear and radial basis function (RBF) kernels. These models provide simple linear and non-linear baselines for comparison with a more advanced deep meta-classifier (DMC). We also tried performing logistic regression on the features, but performance was not better than random guessing.

A DMC is designed as a 1D CNN in order to handle the vectorized weights. Using convolutions on the vectorized weights can be motivated from three different perspectives: 1) there is spatial structure in the weight vector which we can explore, especially for the convolutional layers, 2) for local DMCs we are interested in spatial invariance, so that any subset of neighboring weights can be considered, and 3) for global DMCs we have a large input weight vector from which we need to extract a low-dimensional feature representation before using fully connected components.

Data filtering: Since the objective of a meta-classifier is to explore how hyper-parameters are encoded through the optimization process, we are only interested in models of the weight space that have learned something useful. Therefore, we discard trainings that have not converged, where convergence is defined as specified in Section 4.2.

5.2 Global meta-classification

A global meta-classifier considers all trainable weights from each CNN. We train on the set C_{fixed} in Table 2, where each θ is composed of 92,868 weights. The DMC model consists of 15 1D convolutional layers followed by 6 FC layers. For the SVMs, we consider two methods for extracting the statistical measures mentioned in Section 5.1 – one is to evaluate statistics over the complete set of weights, and the other is to do this layer-by-layer. The layer-wise method extracts separate statistics for multiplicative weights, bias weights, and for each of the batch normalization weights in a layer, yielding a total of 480 training features. We refer to the supplementary material for details on the models and training.

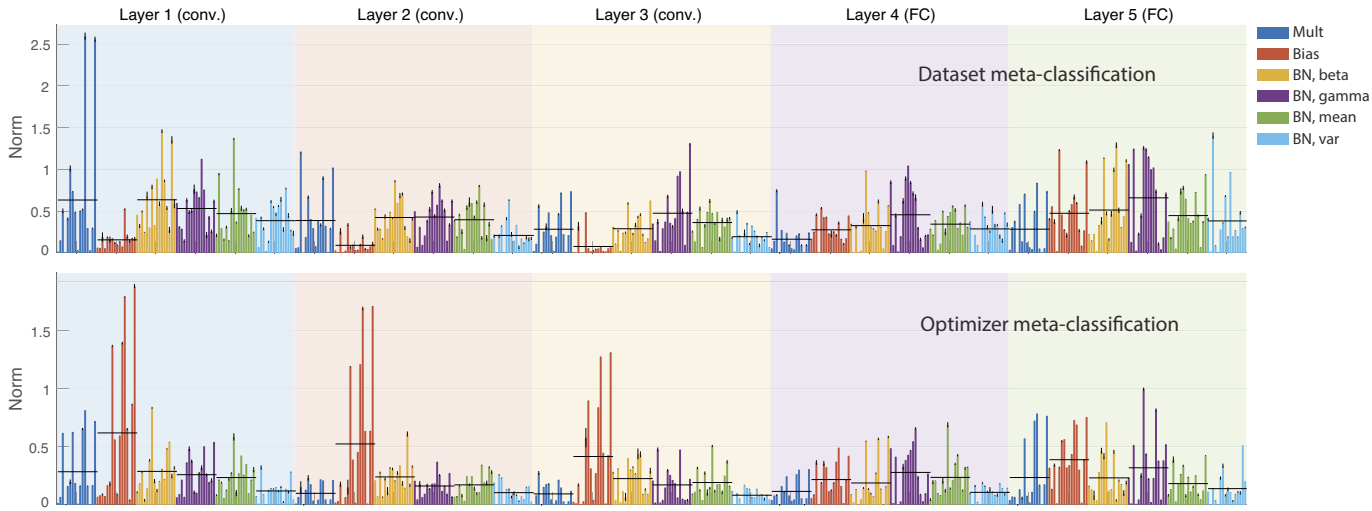

Figure 3: Estimated impact of individual features of linear SVMs for hyper-parameter classification on the set C_{fixed} of the NWS dataset, classifying dataset (top) and optimizer (bottom). Each of the different types of weights of a layer, on the x-axis, are described by 16 features. Horizontal bars denote mean over a type of weights. while error bars show standard deviations over 10 separate training runs. *Mult* represent filters for convolutional layers and weight matrices for fully-connected (FC) layers. *BN* denotes the parameters used for batch normalization.

Figure 2 shows the performance of global meta-classifiers trained on 6 different hyper-parameters. Considering the diversity of the training data, all of the hyper-parameters except for batch size can be predicted with fairly high accuracy using a DMC. This shows that there are many features in the weight space which are characteristic of different hyper-parameters. However, the most surprising results are achieved with a linear SVM and layer-wise weight statistics, with performance not very far from the deep classifiers, and especially for the dataset, optimizer and activation hyper-parameters. Apparently, using separate statistics for each layer and type of weight is enough to give a good estimation on which hyper-parameter was used in training.

By inspecting the decision boundaries of the linear SVMs, we can get a sense for which features that best explain a certain hyper-parameter, as illustrated in Figure 3. Given the coefficients τ_c of a one-versus-rest SVM, it describes a vector that is orthogonal to the hyper-plane in feature space which separates the class c from the rest of the classes. The vector is oriented along the feature axes which are most useful for separating the classes. Taking the norm of τ_c over the classes c, we can get an indication on which features were most important for separating the classes. The most indicative features for determining the dataset are the 1 and 99 percentiles of the gradient of filter weights in the first convolutional layer. As these filters are responsible for extracting simple features, and the gradient of the filters are related to the strength of edge extraction, there is a close link to the statistics of the training images, which explains the good performance of the linear SVM. When evaluating the statistics over all weights, it is not possible to have this direct connection to training data, and performance suffers. For the optimizer meta-classifier, on the other hand, most information comes from the statistics of bias weights in the convolutional layers, and in particular from percentiles of θ and $\nabla\theta$. For activation function meta-classification, the running mean stored by batch normalization in the FC layers contribute most to the linear SVM decisions. For initialization it is more difficult to find isolated types of weights that contribute most, and looking at the SVMs trained on 16 features over the complete θ we can see how initialization is better described by statistics computed globally over the weights.

5.3 Local meta-classification

A local meta-classifier is the model $g_c(\tau, \theta_{[a:b]})$, where $\theta_{[a:b]}$ is the subset of weights between indices a and b. SVMs use features extracted from $\theta_{[a:b]}$, while a local DMC is trained directly on $\theta_{[a:b]}$. A local DMC consists of 12 1D convolutional layers followed by 6 FC layers. The training data is composed of the set C_{main} in Table 2. We use a subset size of $S = 5000$ weights (on average around 5% of a weight vector), such that $b = a + S - 1$. DMCs are trained by randomly picking a for each mini-batch, while SVMs use a fixed number of 10 randomly selected subsets of each θ.

Figure 4 shows the performance of local meta-classifiers g_c, trained on 11 different hyper-parameters ϕ_c from Table 1. For each hyper-parameter, there are three individually trained DMCs based on: subsets of all weights in θ, weights from only the convolutional layers, and only FC weights. In contrast to the global meta-classification it is not possible for an SVM to pinpoint statistics of one particular layer, which makes linear SVMs perform poorly. The RBF kernel improves performance, but is mostly far from the DMC accuracies. Still, the results are consistently better than random guessing for most hyper-parameters, so there is partial information contained in the statistical measures. The best SVM performance is achieved for the initialization hyper-parameter. This makes intuitive sense, as the differences are mainly described by simpler statistics of the weights.

Considering that the local DMCs learn features that are invariant to the architecture, and use only a fraction of the weights, they perform very well compared to global DMCs. This is partly due to the larger training set, but also confirms how much information is stored locally in the vector θ. By comparing DMCs trained on only convolutional or FC weights, we can analyze where most of the features of a certain hyper-parameter are stored, e.g. the dataset footprint is more pronounced in the convolutional layers. For the architectural hyper-parameters, the filter size can be predicted to some extent from only FC weights, which points towards how settings in the convolutional layers affect the FC weights. Compared to the global DMCs, initialization is a more profound local property as compared to e.g. dataset.

Let $\theta_{a,j}$ denote the subset of S weights starting at position a from

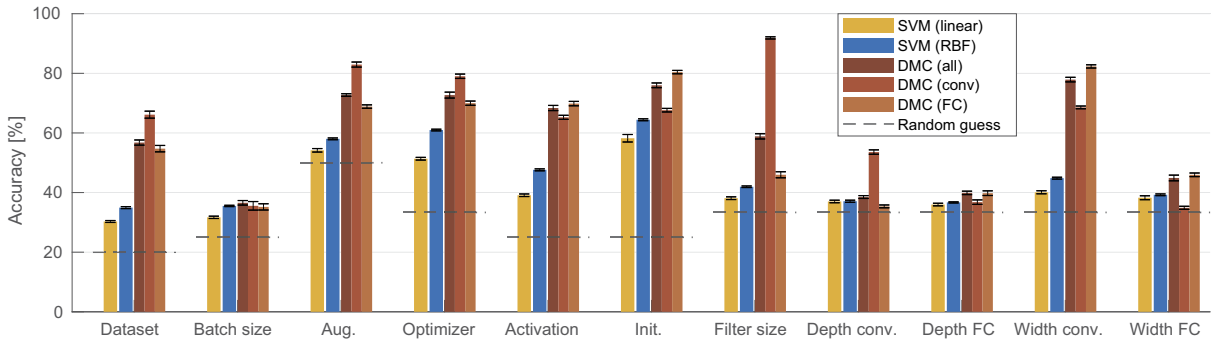

Figure 4: Local meta-classification performance for different hyper-parameters on the C_{main} dataset (Table 2), when using subsets of 5K weights. The SVM classifiers use 16 features from each weight subset. Local DMCs have been separately trained by considering subsets of all/convolutional/FC weights. Performance of random guessing is included for reference. The errorbars show standard deviations over 10 independent training rounds.

optimization step j. The trained model $g_c(\tau, \theta_{a,j})$ can then probe for information across different depths of a model, and track how information evolve during training. Sampling at different a and j, the result is a 2D performance map, see Figure 5, where $(a, j) = (0, 0)$ is in the upper left corner of the map. For the DMC trained to detect the optimizer used, the performance is approximately uniform across the weights except for a high peak close to the first layers. This roughly agrees with the feature importance of the linear SVM in Figure 3, where information about optimizer can be encoded in the bias weights of the early convolutional layers. Inspecting how the DMC performance for initialization decreases faster in the first layers (Figure 5c), we can see how learning faster diverges from the initialization point in the convolutional layers, which agrees with previous studies on how representations are learned [37]. However, we can also see the same tendency in the very last layers. That is, not only convolutional layers quickly diverge from the starting point to adapt to image content, but the last layers also do a similar thing when adapting to the output labels. However, looking at the minimum performance it is still easy to find patterns left from the initialization (see Figure 4), and this hyper-parameter dominates the weight space locally. Connecting to the results in Section 4.2, initialization strategy was also the hyper-parameter of the NWS dataset that showed highest correlation with performance.

6 Discussion

Looking at the results of the meta-classifications, SVMs can perform reasonably well when considering per-layer statistics on the whole set of weights. However, for extracting information from a random subset of weights, the DMCs are clearly superior, pointing to more complex patterns than the statistics used by the SVMs. It is interesting how much information a set of DMCs can extract from a very small subset of the weights, demonstrating how an abundance of information is locally encoded in the weight space. This is interesting from the viewpoint of privacy leakage, but the information can also be used for gaining valuable insight into how optimization shapes the weights of neural networks. This provides a new perspective for XAI, where we see our approach as a first step towards understanding neural networks from a direct inspection of the weight space. For example, in understanding and refining optimization algorithms, we may ask what are the differences in the learned weights caused by different optimizers? Or how does different activation functions affect the weights? Using a meta-classifier we can pinpoint how a majority of the differences caused by optimizer are due to different distributions in bias weights of the convolutional layers. The activation function used when training with batch normalization gives differences in the moving average used for batch normalization of the FC layers. Another interesting observation is the effect of the initialization in Figure 5c and 5f, which points to how convolutional and final layers diverge faster from the initialization point. A potential implication is to motivate studying what is referred to as *differential learning rates* by the *Fastai* library [20]. This has been used for transfer learning, gradually increasing learning rate for deeper layers, but there could be reason to investigate the technique in a wider context, and to look at tuning learning rate of the last layers differently.

6.1 Limitations and future work

We have only scratched the surface of possible explorations in the neural weight space. There is a wealth of settings to be explored using meta-classification, where different combinations of hyper-parameters could reveal the structures of how DNNs learn. So far, we have only considered small vanilla CNNs. Larger and more diverse architectures and training regimes could be considered, e.g. ResNets [16], GANs [15], RNNs, dilated and strided convolutions, as well as different input resolution, loss functions, and tasks.

The performance of DMCs can most likely be improved, e.g. by refining the representation of weights in Section 3. And by aggregating information from the full weight vector using local DMCs, there is potential to learn many things about a black-box DNN with access only to the trained weights. Also, we consider only the weight space itself; a possible extension is to combine weights with layer activations for certain data samples.

While XAI is one of the apparent applications of studying neural networks weights in closer detail, there are many other important applications that would benefit from a large-scale analysis of the weight space, e.g. model compression, model privacy, model pruning, and distance metric learning. Another interesting direction would be to learn weight-generation, e.g. by means of GANs or VAEs, which could be used for initialization or ensemble learning. The model in Section 4.2 was used to show correlations between hyper-parameters and model performance. However, the topic of learning-based hyper-parameter optimization [21, 32] could be explored in closer detail using the weight space sampling. Also, meta-learning could aim to include DMCs during training in order to steer optimization towards good regions in the NWS.

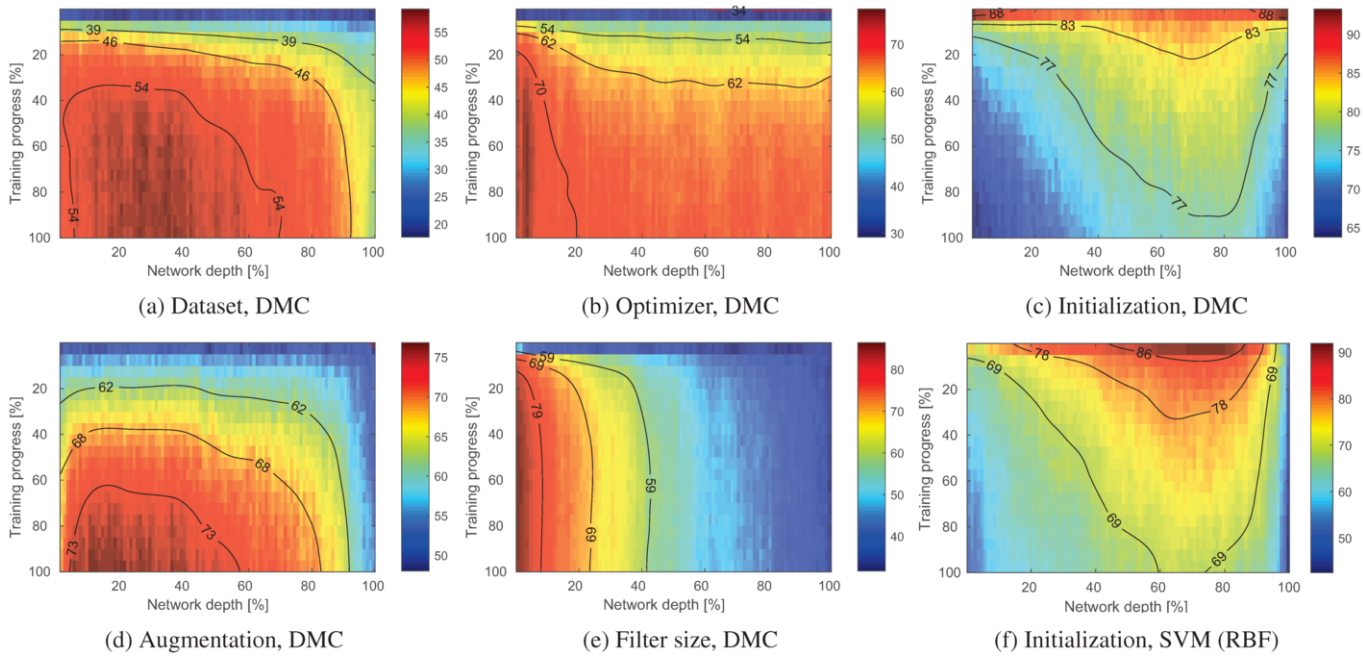

Figure 5: Meta-classification performance maps, visualizing the average test accuracy at different model depths and training progress steps. The y-axis illustrate training progress, from initialization (0%) to converged model (100%), while x-axis is the position of the weight vector where evaluation has been made, from the first weights of the convolutional layers (0%) to the last weights of the fully connected layers (100%). Since the results are evaluated over models with different architectures, it is not possible to draw separating lines between individual layers. (c) and (f) show how similar trends are found by DMCs and SVMs. Note the different colormap ranges (e.g. higher lowest accuracy for initialization). For the results from all DMCs, and individual evaluation on convolutional and fully connected layers, we refer to the supplementary material.

7 Conclusions

This paper introduced the neural weight space (NWS) as a general setting for dissection of trained neural networks. We presented a dataset composed of 16K trained CNN classifiers, which we make available for future research in explainable AI and meta-learning. The dataset was studied both in terms of performance for different hyper-parameter selections, but most importantly we used meta-classifiers for reasoning about the weight space. We showed how a significant amount of information on the training setup can be revealed by only considering a small fraction of random consecutive weights, pointing to the abundance of information locally encoded in DNN weights. We studied this information to learn properties of how optimization shapes the weights of neural networks. The results indicate how much, where, and when the optimization encodes information about the particular hyper-parameters in the weight space.

From the results, we pinpointed initialization as one of the most fundamental local features of the space, followed by activation function and optimizer. Although the actual dataset used for training a network also has a significant impact, the aforementioned properties are in general easier to distinguish, pointing to how optimization techniques can have a more profound effect on the weights as compared to the training data. We see many possible directions for future work, e.g. focusing on meta-learning for improving optimization in deep learning, for example using meta-classifiers during training in order to steer optimization towards good regions in the weight space.

ACKNOWLEDGEMENTS

This project was supported by the Wallenberg Autonomous Systems and Software Program (WASP) and the strategic research environment ELLIIT.

REFERENCES

[1] Guillaume Alain and Yoshua Bengio, 'Understanding intermediate layers using linear classifier probes', *arXiv preprint arXiv:1610.01644*, (2016).

[2] Joseph Antognini and Jascha Sohl-Dickstein, 'PCA of high dimensional random walks with comparison to neural network training', in *Advances in Neural Information Processing Systems (NeurIPS 2018)*, (2018).

[3] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici, 'Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers', *International Journal of Security and Networks (IJSN)*, **10**(3), (2015).

[4] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba, 'Network dissection: Quantifying interpretability of deep visual representations', in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, (2017).

[5] James Bergstra and Yoshua Bengio, 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research (JMLR)*, **13**, (2012).

[6] Alsallakh Bilal, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren, 'Do convolutional neural networks learn class hierarchy?', *IEEE transactions on visualization and computer graphics (TVCG)*, **24**(1), (2018).

[7] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter, 'Fast and accurate deep network learning by exponential linear units (elus)', *arXiv preprint arXiv:1511.07289*, (2015).

[8] Adam Coates, Andrew Ng, and Honglak Lee, 'An analysis of single-layer networks in unsupervised feature learning', in *International conference on artificial intelligence and statistics (AISTATS 2011)*, (2011).

[9] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio, 'Why does unsupervised pre-training help deep learning?', *Journal of Machine Learning Research (JMLR)*, **11**, (2010).

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart, 'Model inversion attacks that exploit confidence information and basic countermeasures', in *ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)*, (2015).

[11] Maxime Gabella, Nitya Afambo, Stefania Ebli, and Gard Spreemann, 'Topology of learning in artificial neural networks', *arXiv preprint arXiv:1902.08160*, (2019).

[12] Marcus Gallagher and Tom Downs, 'Visualization of learning in neural networks using principal component analysis', in *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 1997)*, (1997).

[13] Marcus Gallagher and Tom Downs, 'Weight space learning trajectory visualization', in *Australian Conference on Neural Networks (ACNN 1997)*, (1997).

[14] Xavier Glorot and Yoshua Bengio, 'Understanding the difficulty of training deep feedforward neural networks', in *International conference on artificial intelligence and statistics (AISTATS 2010)*, (2010).

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, 'Generative adversarial nets', in *International Conference on Neural Information Processing Systems (NIPS 2014)*, (2014).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 'Deep residual learning for image recognition', in *IEEE conference on computer vision and pattern recognition (CVPR 2016)*, (2016).

[17] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky, 'Neural networks for machine learning lecture 6a overview of mini-batch gradient descent', (2012).

[18] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz, 'Deep models under the gan: information leakage from collaborative deep learning', in *ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)*, (2017).

[19] Fred Matthew Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau, 'Visual analytics in deep learning: An interrogative survey for the next frontiers', *IEEE transactions on visualization and computer graphics (TVCG)*, (2018).

[20] Jeremy Howard et al. fastai. `https://github.com/fastai/fastai`, 2018.

[21] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *International Conference on Learning and Intelligent Optimization (LION 2011)*, (2011).

[22] Sergey Ioffe and Christian Szegedy, 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', in *International Conference on Machine Learning (ICML 2015)*, (2015).

[23] Diederik P Kingma and Jimmy Ba, 'ADAM: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, (2014).

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

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al., 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, **86**(11), (1998).

[26] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft, 'Convergent learning: Do different neural networks learn the same representations?', in *NIPS Workshop on Feature Extraction: Modern Questions and Challenges*, (2015).

[27] Zachary C Lipton, 'Stuck in a what? adventures in weight space', *arXiv preprint arXiv:1602.07320*, (2016).

[28] Hanxiao Liu, Karen Simonyan, and Yiming Yang, 'DARTS: Differentiable architecture search', in *International Conference on Learning Representations (ICLR 2019)*, (2019).

[29] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu, 'Towards better analysis of deep convolutional neural networks', *IEEE transactions on visualization and computer graphics (TVCG)*, **23**(1), (2017).

[30] Eliana Lorch, 'Visualizing deep network training trajectories with PCA', in *ICML Workshop on Visualization for Deep Learning*, (2016).

[31] Aravindh Mahendran and Andrea Vedaldi, 'Understanding deep image representations by inverting them', in *IEEE conference on computer vision and pattern recognition (CVPR 2015)*, (2015).

[32] Jonas Mockus, *Bayesian approach to global optimization: theory and applications*, volume 37, 2012.

[33] Vinod Nair and Geoffrey E Hinton, 'Rectified linear units improve restricted boltzmann machines', in *International conference on machine learning (ICML 2010)*, (2010).

[34] Milad Nasr, Reza Shokri, and Amir Houmansadr, 'Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks', *arXiv preprint arXiv:1812.00910*, (2018).

[35] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng, 'Reading digits in natural images with unsupervised feature learning', in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, (2011).

[36] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein, 'Sensitivity and generalization in neural networks: an empirical study', in *International Conference on Learning Representations (ICLR 2018)*, (2018).

[37] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein, 'Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability', in *International Conference on Neural Information Processing Systems (NIPS 2017)*, (2017).

[38] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea, 'Visualizing the hidden activity of artificial neural networks', *IEEE transactions on visualization and computer graphics (TVCG)*, **23**(1), (2017).

[39] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin, 'Large-scale evolution of image classifiers', in *International Conference on Machine Learning (ICML 2017)*, (2017).

[40] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra, 'Grad-cam: Visual explanations from deep networks via gradient-based localization', in *IEEE International Conference on Computer Vision (CVPR 2017)*, (2017).

[41] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov, 'Membership inference attacks against machine learning models', in *IEEE Symposium on Security and Privacy (SP)*. IEEE, (2017).

[42] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 'Deep inside convolutional networks: Visualising image classification models and saliency maps', *arXiv preprint arXiv:1312.6034*, (2013).

[43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, 'Dropout: a simple way to prevent neural networks from overfitting', *The Journal of Machine Learning Research (JMLR)*, **15**(1), (2014).

[44] Han Xiao, Kashif Rasul, and Roland Vollgraf, 'Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms', *arXiv preprint arXiv:1708.07747*, (2017).

[45] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson, 'How transferable are features in deep neural networks?', in *International Conference on Neural Information Processing Systems (NIPS 2014)*, (2014).

[46] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 'Understanding neural networks through deep visualization', in *ICML Workshop on Deep Learning*, (2015).

[47] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese, 'Taskonomy: Disentangling task transfer learning', in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018)*, (2018).

[48] Matthew D Zeiler and Rob Fergus, 'Visualizing and understanding convolutional networks', in *European conference on computer vision (ECCV 2014)*. Springer, (2014).

[49] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba, 'Interpreting deep visual representations via network dissection', *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, (2018).

[50] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba, 'Object detectors emerge in deep scene CNNs', in *International Conference on Learning Representations (ICLR 2015)*, (2015).

[51] Barret Zoph and Quoc V. Le, 'Neural architecture search with reinforcement learning', in *International Conference on Learning Representations (ICLR 2017)*, (2017).

[52] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le, 'Learning transferable architectures for scalable image recognition', in *IEEE conference on computer vision and pattern recognition (CVPR 2018)*, (2018).