# Neural Architecture Construction using EnvelopeNets

Purushotham Kamath[*]
SigFind
utham@sigfind.com

Abhishek Singh
Cisco Systems
abhishs8@cisco.com

Debo Dutta
Cisco Systems
dedutta@cisco.com



## ABSTRACT

In recent years, advances in the design of convolutional neural networks have resulted in significant improvements on the image classification and object detection problems. One of the advances is networks built by stacking complex cells, seen in such networks as InceptionNet and NasNet. These cells are either constructed by hand, generated by generative networks or discovered by search. Unlike conventional networks (where layers consist of a convolution block, sampling and non linear unit), the new cells feature more complex designs consisting of several filters and other operators connected in series and parallel. Recently, several cells have been proposed or generated that are supersets of previously proposed custom or generated cells. Influenced by this, we introduce a network construction method based on *EnvelopeNets*. An EnvelopeNet is a deep convolutional neural network of stacked *EnvelopeCells*. EnvelopeCells are supersets (or envelopes) of previously proposed handcrafted and generated cells. We propose a method to construct improved network architectures by restructuring EnvelopeNets. The algorithm restructures an EnvelopeNet by rearranging blocks in the network. It identifies blocks to be restructured using metrics derived from the featuremaps collected during a partial training run of the EnvelopeNet. The method requires less computation resources to generate an architecture than an optimized architecture search over the entire search space of blocks. The restructured networks have higher accuracy on the image classification problem on a representative dataset than both the generating EnvelopeNet and an equivalent arbitrary network.

## 1 INTRODUCTION

In recent years, several neural networks have shown significant improvements on the image classification and object detection problems. Based on the method used to construct the network, they can broadly be classified into bespoke cell based architectures and search/generative based cell/network architectures

Examples of bespoke network architectures include, among others, InceptionNet [19], DenseNet [7], and ResNet [6]. These networks were hand designed and introduced mechanisms such as complex cell structures with parallel paths and skip connections. They were designed based on an understanding of the behavior of a convolutional network. *E.g.* InceptionNet used a carefully custom designed cell, based on the ideas of 1x1 convolutions [10] and multiple filters in parallel. Its cell structure consisted of 6 blocks and the first version of its architecture consisted of 6 layers of the inception cell with 2 widening layers.

Following the success of the cell based architectures, recent work has indicated that search algorithms and generative models can find cell architectures or network architectures that outperform the bespoke architectures. NasNet [21], [22] uses a policy gradient algorithm and a generative network to search for the optimal cell constructed from a limited set of blocks *e.g.* 3x3, 5x5, and 7x7 convolutional filters. The cell that was discovered consisted of 8-10 blocks connected in series and parallel. The network was constructed by stacking several of these cells to form a deep network. The motivation for such an approach comes from the intuition that a search for cell architecture is much faster than the search for a network architecture and the cell could generalize across training sets.

While successive generations of the "cell" based architecture have improved image recognition accuracy on standard test sets, it is somewhat unclear what mechanism in a cell at each layer has driven these improvements. However there is work that indicate that different parts of the network play different roles in the overall classification task. Zeiler et. al. [20] shows that shallower layers extract gross features while deeper layers compose the gross features to extract. Li et. al. [9] show that pruning can be effective in reducing the size of a network without significant degradation in accuracy. Both these works indicate that, at different layers of a network, some filters are more important than others. In other words, there may not exist a single optimal cell structure for all training sets. As a consequence different cell structures may perform better at different depths of the network. This indicates the feasibility of building a network by restructuring the cell structure of a stacked cell network.

An observation of the cell structures from the cell generation methods, indicates that often, the generated cells form an "envelope around" or "superset of", or "partial subset of" the bespoke cells [22]. Influenced by this, we propose a new method of construction that provides a prior to the network design. The prior reduces the search space for finding a large (deep and/or wide) architecture. The resulting search method differs from both purely incremental and reduction methods of construction.

The construction method starts with a uniform stacked network called an *EnvelopeNet* constructed by stacking cells called *EnvelopeCells*. The EnvelopeCells are hand constructed and have several parallel paths using several types of filters and other blocks, forming a superset or envelope which could encapsulate several popular cell architectures. The algorithm trains the envelope network and during training extracts per filter statistics from the EnvelopeNet. It uses theses metrics to restructure the network architecture by pruning some filters based on the metrics obtained from the featuremaps at the output of the filters. It simultaneously deepens the network by adding envelope cells to maintain approximately the same number of parameters as the original EnvelopeNet. The algorithm executes several iterations of restructuring building a deeper network with each iteration. The metrics needed to restructure the network stabilize fairly quickly (compared to the time needed for accuracy to stabilize), allowing for a quick identification of the blocks to be restructured.

[*]Work done while author was at Cisco.

The generated networks are compared to the generating EnvelopeNet and to an ”arbitrary” network *i.e.* a network of the same depth and number of filters whose network structure is chosen arbitrarily, but with approximately the same number of parameters. On the image classification problem using a standard (CIFAR-10) dataset the generated network performs better than both the EnvelopeNet and the arbitrary network.

## 2 RELATED WORK

Recent work in the area of cell based neural network design and architecture search can be broadly classified into three categories:

- Bespoke cell design: In this method, a custom cell is carefully designed based on insight into the performance of different types of blocks. Multiple cells are stacked in series to form a network. *E.g.* InceptionNetv3 [19] uses three different types of cells which they refer to as inception modules.
- Cell design through generative models/optimization: In this method, generative models are used to find close to optimal cell designs. Multiple cells are stacked in series to form a network. *E.g.* NasNet [22] uses an RNN to generate cell structures and reinforcement learning to optimize the cell architecture.
- Network design through search/optimization: In this method a whole network is designed through a combinatorial search, via optimization or evolutionary algorithms from very basic blocks (operators) *E.g.* Neuroevolution [5], AmoebaNet [15]

Examples of bespoke cell based architectures include InceptionNet [19], DenseNet [7], and ResNet [6]. These networks have repeated structures of cells and/or connections. The InceptionNet cell has four parallel paths with 3x3, 5x5, 1x1 (for dimensionality reduction) convolution blocks and a max pooling block. One of its two design motivations was to construct a network such that blocks with high correlated outputs should be clustered together as inputs for the next layer [1]. The resulting cell design was a relatively ”lean” block (as compared to the cell designs that followed). Resnet [6] uses skip connections to connect layers to provide an identity function that improves performance and several bespoke networks incorporated these connections.

The bespoke cell based architectures were followed by models that generated cell designs. Neural Architecture Search [21] uses a RNN that generated the number of filters, filter size, and stride for a convolution network. The resulting performance exceeded that of the state of the art bespoke networks. NasNet[22] uses similar generative techniques with reinforcement learning to search for cell designs composed from a fixed set of operators (blocks) . The approach was motivated by the design of a search space for a cell such that that the complexity of the network architecture is independent of depth and input size. The work indicates that in a network of stacked cells, the search for the best network architecture is reduced to searching for the best cell architecture A very interesting observation of their generated architecture is that it is an envelope over a broad class of human invented architectures. This concept of an ”envelope” architecture has motivated our study. Progressive NAS [11] extends the work by searching for a good cell composed of blocks. The work reduces the search space of operators and finds a reusable cell structure using sequential model based optimization. The work showed that learning can be transfered from a smaller network trained using a CIFAR dataset to a larger network for the ImageNet dataset, addressing concerns that construction methods are susceptible to overfitting. Efficient NAS [14] extends the generative constructions methods further by reducing the computation resources needed through parameter sharing across iterations of generation.

A third method of construction uses a search space that consists of the entire set of neural networks that can be formed using a limited set of blocks (operators) with an optimization algorithm to drive the search process. These methods do not have repeating cell structures. Neuroevolution methods [5] encompass a range of evolutionary algorithms/techniques that discover network architectures. Real et. al. [16] proposed an evolutionary algorithm to pick a combination of architecture and hyperparameters. Their approach is to design a network from a set of basic blocks (Convolutional, Batch Normalization, Relu) through a evolutionary algorithm that grows a network via mutations on a population. The mutations are designed ensure that the performance of the population improves as evolution progresses. A regularized version of this technique called AmoebaNet [15] improves performance further and starts the neuro evolution from a prior. Elsken et. al. [4] attempt a neural architecture search using blocks (operators) rather than standard cells. They use hill climbing to incrementally build neural networks. The resulting networks show gains over other approaches. The computation resources needed to evaluate all combinations of the blocks in these evolutionary methods was one of the motivating factors behind the generative/search based cell architecture design.

There has also been considerable interest in algorithms to reduce neural network size without loss of accuracy. These studies were motivated by the need to improve inference times and the need to run inference on the reduced resources. Li et. al. [9] showed that filter pruning of convolutional networks can reduce parameters, training and inference times without significant degradation in accuracy. Roy et. al. [17], and Molchanov et. al. [13] also use pruning to remove duplicates or to reduce model size. Mittal et. al [12] shows that random pruning can be as effective as algorithmic pruning when performed on a hand designed network (such as ResNet).

The tuning of model hyperparameters has been studied in depth. A number of techniques such as grid search, Bayesian optimization and random search [2] are used. These techniques are effective for continuous valued parameters, although it is harder to apply them to tune the placement and connectivity of a network architecture.

Construction using EnvelopeNets spans the domains of the construction and reduction methods discussed above. It is based on standard cell construction, filter pruning and network restructuring techniques. The algorithm uses a network architecture consisting of stacked standard cells (EnvelopeCells) to begin construction. Note that EnvelopeCells may be non optimal - they are is intentionally chosen to be over provisioned. The network restructuring techniques use metrics extracted during training of the EnvelopeNet and it restructures the network in iterations. Restructuring is done by removing low performing filters and adding new envelope cells to the deep end of the network. By doing this, the method avoids searching over the entire space of network architectures. The network restructuring method also differentiates our work from the neuroevolution methods, where single blocks/filters are added based on a mutation, or optimization technique.

## 3 MOTIVATION

Neural architecture search across all types of operators for a network architecture is computationally expensive [4]. This has lead to research into neural architecture search for cell architectures that perform well. While this, too, is computationally intensive [11], intelligent generation and search algorithms have found cells that have improved upon hand crafted cells. However, both these methods rely on an iterative cycle of incremental construction and performance measurement (via training and evaluation) with an optimization algorithm. The motivation is to find less compute intensive algorithms that can exploit information extracted from a partially trained network to construct improved networks. We propose a construction method based on EnvelopeNets. An *EnvelopeNet* is a deep convolutional neural network of stacked EnvelopeCells. *EnvelopeCells* are supersets (or envelopes) of previously proposed handcrafted and generated cells. This work attempts to understand if networks constructed in this way, can perform significantly better than the envelopes from which they are generated and better than equivalent arbitrarily constructed networks. It also attempts to understand if the constructed network can be identified with less computation resources than needed for an incremental construction approach with full training and optimization.

## 4 HYPOTHESIS

The hypothesis, that envelope networks can be restructured to yield higher performing networks, is based on intuition from related work around cell construction, network search and pruning filters and connections.

The following terminology is used in the discussion of construction:

- Block $B_i$: An operation such as convolution, max pool, concatenation (also called an operator).
- Cell $C_i$: A combination of blocks in series or parallel. May be custom built [6] or generated [21, 22].
- Network $N_i$: A combination of cells and/or blocks. *E.g.* a network may consist of cells stacked in series, or a directed graph of blocks.

We occasionally use the term network *width* to refer to the number of parallel branches of a cell. Where used, the context should be sufficient to identify whether it is network width or the more commonly used channel width.

The problem statement for Envelope construction is: Given a network $N_e$ of depth $L_e$ built by stacking envelope cells $C_e$ (custom built or generated) in series, find the network $N_m$, from the set of all possible networks $N_i$ of depth $L_i > L_e$, that may be constructed by removing $n$ blocks of $N_e$ and adding them back in different locations, that maximizes accuracy $Perf$ subject to constraints on network complexity (number of blocks, parameters and operations $M$).

Find $N_m$ such that

$$Perf(N_m) > Perf(N_i), \forall i \mid M(N_m) < M_{max}$$

Note that, the algorithms described in this work are, in general, non optimal. They do, however, generate constructions whose performance exceeds that of the EnvelopeNet and an arbitrarily constructed network of the same network complexity (same depth, same blocks and approximately the same number of parameters).

There has been considerable work on what the individual layers of a neural network perform using visualization techniques [20]. The studies indicate that after training a network, the shallower layer (layers closer to the head of the network) extract gross features (edges, boundaries, shapes) while deeper layers (layers closer to the tail) compose these into more abstract features or objects (such as meshes, facial features). Their results also show that visual inspection of filter performance can be used for architecture selection and can have a significant impact on performance. Chollet [3] indicates that different filter types can extract different characteristics. By providing 3 parallel paths with different filters, each inception cell can be tuned to extract features at different levels. After training, it was found that for most layers, one of the paths dominated the others, indicating that one path was primarily activated at each layer. ENAS [11] extends the cell architecture by adding blocks in series, and uses a cell that has four parallel paths and one in series. This cell performs better than InceptionNet on common training sets. Pham et. al. [14] improves the construction method and indicates that their cell is locally optimal *i.e.* that removal of blocks from the cell could reduce performance of the network.

The EnvelopeCell used in this work is similar to the InceptionCell but uses blocks that were identified by ENAS. Note that we do not claim that the EnvelopeCell we have designed is either locally or globally optimal, or that the generated network is, either. Our hypothesis is that the generated network can outperform the EnvelopeNet and an equivalent arbitrary network and that the generated network can be identified with less compute resources than an search or generative method.

A number of metrics have been surveyed for choosing which filters to prune [12] to reduce a convolutional network. These include the mean activation, $l_1$ norm, entropy on activations and scaled entropy. We experimented with these metrics and also used another metric based on feature map variance. It is based on the intuitive reasoning that the best filters to prune are those that have the least variance in the featuremaps, computed over a reasonably large training set. The reasoning behind the intuition is that after a reasonable amount of training is complete, filters generally identify the scale of the features which they extract. In this state, filters which have consistently low variance in the distribution of their output featuremap over the training, are contributing less to the classifier's output. Such filters are candidate filters for pruning, and substituted by other filters at locations where they may contribute to more information extraction. The idea is that lower performing filters would be better suited to be placed in a layer where they can contribute more to the classification task. This hypothesis is supported by the results of Zeiler et. al. [20], that show that, after training, some filters in a deep network end up with featuremaps with "dead" features (uniform featuremaps, with consistently low

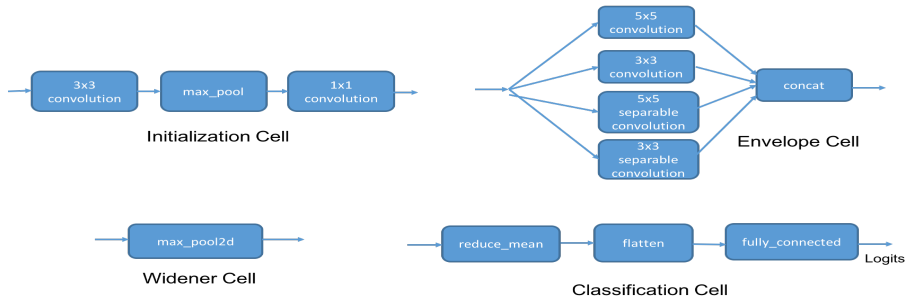


**Figure 1: Cell architectures**

variance over the training set), while other have cleaner distinctive features (featuremaps with high variance).

This work studies a simplified version of the construction algorithm: Instead of restructuring the network, by adding back the $n_b$ lowest performing filters to the network, at different locations, an EnvelopeCell is added to the tail of the network. $n_b$ is set to equal the number of blocks (filters) in an EnvelopeCell. The algorithm runs in iterations. In each iteration $n$ blocks are removed, while a new EnvelopeCell is added to the deep end of the network. This narrows and deepens the network while maintaining the overall network parameter count approximately same. In this work the envelope and generated networks do not have connections across cells (skip connections). (Ongoing work is studying a fully connected envelope *i.e.* the input to every cell is the concatenation of the output every preceding cell (subject to dimension reduction where needed)).

## 5 CONSTRUCTION USING ENVELOPENETS

In this section we discuss the design of the EnvelopeCell, the EnvelopeNet and the construction algorithm. The EnvelopeCell considered in this work is a sample envelope. We do not make the claim that this is an optimal EnvelopeCell, but we use it to justify our claims regarding networks constructed from EnvelopeNets.

### 5.1 EnvelopeCell

The EnvelopeCell is built using the blocks listed below connected in parallel. Each block consists of a convolution block, a Relu unit and a batch normalization unit. The outputs of the blocks are concatenated to form the final output.

- 3x3 convolution
- 3x3 separable convolution
- 5x5 convolution
- 5x5 separable convolution

The reasoning behind the choice of these blocks is that these blocks were the component blocks used in the cell architecture discovered in [14]. In addition we use two additional cells:

- Widener: A widening cell is a maxpool unit that reduces the image dimensions by a factor of two and doubles the channel width. The widener blocks are placed at regular intervals ($wideningfactor$) in the network which is a common design practice [18]. The experiments in this work use $wideningfactor$=4.
- Classification: A classification cell consisting of a average pooling block, a fully connected convolution and a soft max. This cell is common to all networks. No dropout is applied - the reasons are discussed in the following section.

Figure 1 shows the envelope cell and the other 3 basic cells we use.

### 5.2 EnvelopeNet

The EnvelopeNet is comprised of a number of the EnvelopeCells stacked in series. The stem and classification blocks are placed at the head and tail of the network. Wideners are placed at intervals of $widenerinterval$ = 4 envelope cells.

The experiments start with an Envelope network that is 6 layers deep (6 EnvelopeCells and a widener at layer 4). This network is referred to as the *6x4* EnvelopeNet, since the EnvelopeCell is 4 filters wide, *i.e* the network has a width of 4. Networks generated by the construction algorithm are referred to as *6x4-N* networks where *N* is the number of restructuring iterations of the algorithms before generation of the network.

### 5.3 Construction

Algorithm 1 shows the algorithm used to construct the networks.

The algorithm starts with the EnvelopeNet and trains it for $trainingsteps$ steps. During the training, statistics from the featuremaps at the outputs of the filter (sums of elements, squared sums of elements and sample counts) are collected. A running variance of the elements of featuremaps is computed and from this, the average variance per element is computed. The filters are then sorted in order of the variances and the filters with the $maxrestructure$ lowest variances are pruned, subject to the constraint that every layer must have at least one filter. An EnvelopeCell is added to the tail end (deepest end) of the network. The algorithms is run for $restructureiterations$ iterations. We set $maxrestructure$ = 4, so that 4 filters are removed. 4 filters are added back (in the EnvelopeCell), so that the overall number of filters remains constant (and the overall number of parameters remains comparable to that of the EnvelopeNet).

Figure 2 shows the 6x4 EnvelopeNet and the generated network 6x4-5, after $restructureiterations$ = 5 iterations (an 11 layer network is generated).

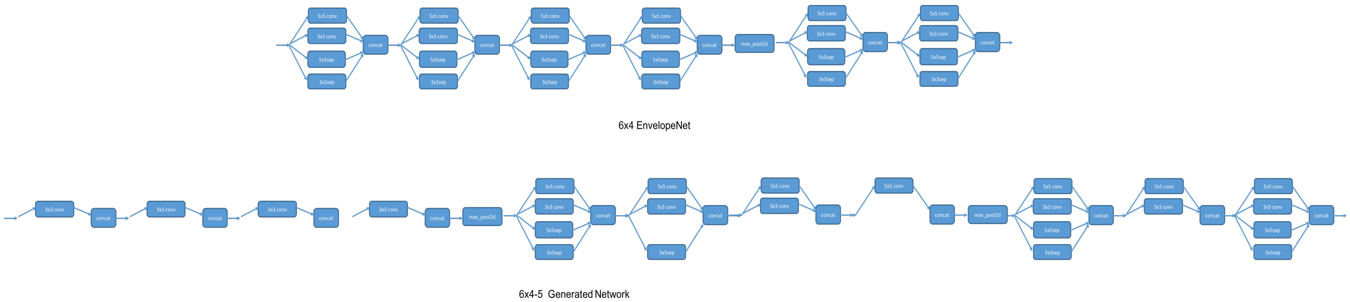


**Figure 2: 6x4 Envelope Network and the 6x4-5 generated network**

**ALGORITHM 1:** Neural Architecture Construction

**Input:** EnvelopeNet *E*, restructureiterations *R*
**Output:** GeneratedNet *network*
$iterations \leftarrow 0$
$network \leftarrow E$
**while** *iterations* ¡ *R* **do**
  //Filter stats are featuremap variances indexed by filter
  //and layer, collected during training
  $filterstats \leftarrow train(network)$
  $evaluate(network)$
  $network \leftarrow construct(network, filterstats)$
  $iterations \leftarrow iterations + 1$
**end**
return *network*
**Function *construct*** *network, filterstats*
  //Sort cells/layer in order of variance
  $sortedfilters \leftarrow sort(filterstats)$
  $restructurefilters \leftarrow []$
  $filtercount \leftarrow getfiltercount(network)$
  **for** *filter c in sortedfilters* **do**
    $layer \leftarrow layer(filter)$
    **if** *restructurefiltercount(layer) + 1 ¿ filtercount(layer)* **then**
      //Do not prune a cell if it is the last cell
      //in the layer
      continue
    **else**
      restructurefilters.add(filter)
      **if** *length(restructuredfilters) ¿ maxrestructuring* **then**
        //Limit restructured filters to 4
        break
      **end**
    **end**
  **end**
  //Remove pruned filters and add envelopecell to end
  //Add necessary wideners
  $prunefilters(network, restructurefilters)$
  $addcell(network, envelopecell)$
  return *network*
**end**

## 6 RESULTS

We ran two sets of experiments: an analysis of the metric chosen to select the filters to be pruned during the restructuring process and an analysis of the performance of the restructured network with ablation.

The experiments covered both construction of the network and evaluation of the EnvelopeNet and the generated network. Both the construction and the evaluation phase train the networks from scratch *i.e* there is no parameter inheritance across iterations of the construction algorithm or from construction to evaluation. The algorithm was evaluated on the image classification problem using the CIFAR-10 dataset [8] Both construction and the evaluation of the generated networks used a common set of hyperparameters which were kept constant for all runs. The training used preprocessing techniques such as random cropping, varying brightness and contrast. The optimization was RMSProp with *momentum*=0.9, *learning rate*=0.01 with an *exponential decay* of 0.94 per 2 epochs. Weight decay with a factor of $4 \times 10^{-5}$ was used.

The batch size was set to 50 for all of the experiments. No hyperparameter tuning was done on the envelope net or the generated networks. The number of restructuring iterations *restructureiterations* was set to 5. The number of *trainingsteps* for the restruturing algorithm was set to 30K. The number of filters to be pruned in an iteration was set to $maxrestructure = 4$. Training and evaluation of the based and generated networks ran for 30-70K training steps. Our experiments were run on NVIDIA GeForce GTX 980/1080 GPU systems on bare metal and on the NVIDIA K80 GPU systems on the AWS cloud running TensorFlow v1.5.0.

### 6.1 Restructuring metric

The restructuring metric chosen was the variance of the featuremaps at the output of the filters. The size of all featuremaps at a given layer is the same, allowing the variance of featuremaps at different filters to be compared. However, to compare the metric across layers, the per filter variance needs to be averaged per element of the featuremap. At every training step, the algorithms collect counts and sums from the featuremap at the output of every filter. The counts are used to calculate a running variance of each element of the featuremap on a per filter basis. The per filter per element variance is obtained by averaging over the elements of the featuremap.

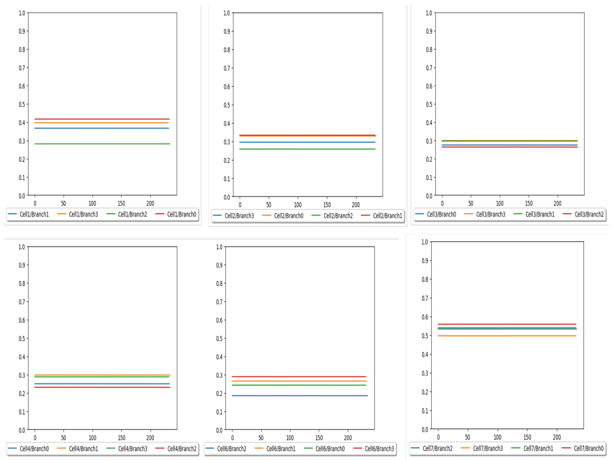

**Figure 3: Running variance of the featuremaps of individual filters vs. training iterations for a 6x4 EnvelopeNet. The filters are grouped by layer. Each graph shows the variance for 4 filters in one layer and 6 layers are shown. The data shows a snapshot of 300 steps after approximately 30K training steps.**

Figure 3 shows the variance vs. training steps. The graph shows a small snapshot (300 training steps) of the evolution of the running featuremap variance for each filter in each layer. After 30K iterations the variances are relatively constant and the filters can be sorted in order of variance to identify the filters to prune. The graphs show that variance stabilizes within 30K iterations for this dataset (the variance stabilization time). This is substantially lower than the number of iterations required to train the network to maximum accuracy (approximately 100K iterations).

## 6.2 Construction

The variance results show that variance has stabilized within 30K iterations, allowing the selection of the filter to be restructured fairly quickly. The total time for the algorithm to run $N$ iterations is $N$ times the variance stabilization time. This compares favorably with both evolutionary methods, where the run time is a function of the total number of possible block combinations for a network, and cell search methods where the search time is a function of the total number of possible block combinations for a cell. At each iteration the accuracy is checked to verify that it exceeds the accuracy of the previous network. If the accuracy drops, the iteration terminates (ongoing work is exploring the techniques to handle this, perhaps by retracing steps and choosing a different set of filters to prune). However in our experiments the accuracy increased with each iteration and the algorithm did not terminate prematurely. Figure 2 shows the generated networks. Table 1 shows the number of filters for each layer, the parameters and flops for the 6x4 EnvelopeNet and the generated network (6x4-5). It also shows the parameters for an "arbitrary" network - a network with the same depth and and number of filters as 6x4-5, but with the structure chosen arbitrarily, called 6x4-5-Random (or the *arbitrary net*). Note tconsider all arbitrary networks,

| Network | Number of filters in each layer | Parameters | Operations (flops) |
|---|---|---|---|
| 6x4 | 4, 4, 4, 4, 4, 4 | 12.18M | 7.82B |
| 6x4-5 | 1, 1, 1, 1, 4, 3, 2, 1, 4, 2, 4 | 16.91M | 18.65B |
| 6x4-5-Random | 4, 4, 4, 2, 1, 1, 2, 2, 1, 2, 1 | 16.41M | 14.47B |

**Table 1: Number of filters at each layer and the number of parameters for the 6x4 EnvelopeNet and the generated networks.**

## 6.3 Performance

The Neural Architecture Construction algorithm was run for 5 iterations. The network used for the first iteration was the base 6x4 envelope network. Each succeeding iteration increased the depth by one EnvelopeCell, while reducing the filters by 4. The final generated architecture was an 11 layer network (not including the wideners).

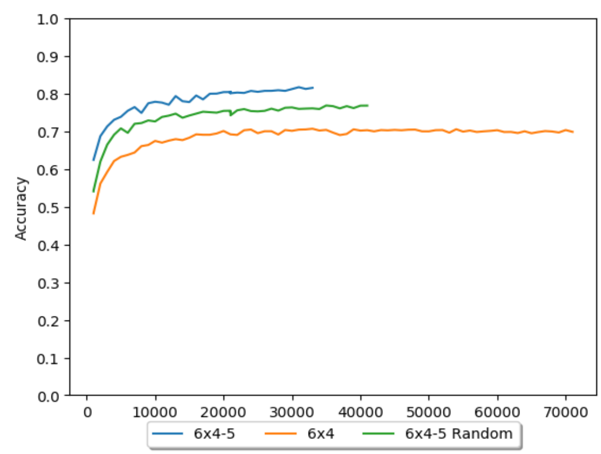


**Figure 4: Accuracy vs. training iterations for a 6x4 EnvelopeNet, a 6x4-5 generated network and an arbitrary network of the same depth, same filters as 6x4-5, but chosen arbitrarily (called 6x4-5-Random) for the image recognition task on the CIFAR-10 data set**

The EnvelopeNet, the generated net and the arbitrary net were evaluated. Each network was trained on the CIFAR-10 dataset for 30-70K steps with performance on the test set evaluated every 5K steps (5 epochs). Figure 4 shows accuracy of the 6x4 EnvelopeNet, the 6x4-5 network and the arbitrary network (6x4-5-Random) on the image classification task using the CIFAR-10 dataset.

The results show the restructured network clearly outperforming the original EnvelopeNet. The performance of the arbitrary network is better than the EnvelopeNet, but lower than the algorithmically generated network. This ablation result indicates that structure of the generated network is responsible for some of the gain, and that the entire gains do not come from deepening the network.

## 7 DISCUSSION

Deep convolutional networks gain a majority of their improvement from deepening the network. Hence, it is valid to question whether the increase in accuracy of the restructuring method, comes from just deepening the network, or from restructuring the filters. This question is important because previous approaches proposed network structures formed by stacking uniform cells where performance is usually proportional depth up to a certain point. Figure 4 indicates that networks with the same depth, but with different structures (the generated net vs. the arbitrary net) can performance differently. One possible reason for this is feature abstraction: Filters at different depths extract features at different layers of abstraction. At particular levels of abstraction, there may be large number of features to extract, which in turn may necessitate a large number of filters at a specific depth. This would explain the difference in the performance of a non uniform vs. a uniform stacked cell architecture. Ongoing work is investigating the reasons for performance differences between uniform and non-uniform architectures.

Previous studies have indicated that neural networks exhibit a form of plasticity, allowing random pruning of filters from a network, with little degradation of accuracy provided the fraction of filters pruned is reasonably small [9]. One counter intuitive result is that random pruning is as effective as algorithmic pruning when it comes to the accuracy of final trained model [12]. This indicates that perhaps an algorithmic approach is unnecessary. However the key difference between our work and the pruning studies is that existing studies consider pruning of a hand crafted network that has already gone through an optimized design. In contrast, our network attempts to prune an EnvelopeNet, which is an over provisioned network. As can be seen from the results in Figure 4, the algorithm generated network, outperforms a random network of equivalent size. The root cause for this may lie in the distribution of the importance of the filters. As we reduce from a large EnvelopeNet to an intermediate network (similar to a handcrafted network) to a pruned network, the benefit obtained from pruning may decrease, possibly hitting a knee around the intermediate network. This would fit in well with our observations as well as results from pruning studies.

The restructuring (pruning), bears some resemblance to the Dropout regularization method. While the motivation behind each method is different, they both remove elements of filters, one probabilistically, during training and the other, physically during construction. During our experiments, we observed that dropout does impact the construction process: it increases the time needed for the variance to stabilize. For this reason, we turn off dropout during the construction process. Ongoing work is evaluating the effect of enabling dropout during the training/evaluation of the generated network.

Compared to existing methods of generating or evolutionary methods, the advantage of our method comes from giving a strong prior to the network construction procedure by setting up initial layout of the architecture. This initial layout of the architecture helps the algorithm to restructure the network rather than use resources for the discovery of a base network structure. Here, we loosely define the base network structure as a network that uses traditional and successful convolutional architecture design practices *e.g.* reducing featuremap's dimension and increasing number of channels periodically after a certain number of layers.

The EnvelopeCell based pruning and addition method of restructuring can be viewed as lying between bottom up incremental methods of construction and top down pure reduction methods of construction. The networks generated could be generated incrementally from scratch or purely by reduction from a much deeper, wider EnvelopeNet. However in either case the resources needed to reach the prior makes both approaches more compute intensive. Eventually through better design of cells, future work may result in a universal EnvelopeNet consisting of multiple stacked layers of a universal EnvelopeCell.

## 8 CONCLUSIONS

In this paper we introduce a method to construct a neural architecture based on restructuring an EnvelopeNet. Neural architecture and search has, so far, focused on incremental construction of networks through cell generation followed by stacking, or on evolutionary search for optimal architectures. In this work, we introduce restructuring based construction methods. They differ from conventional search/generative methods in that they combine cell based construction with network reduction. By training an EnvelopeNet during the construction process we gain insight into the behavior of filters and other blocks that are used to restructure the network. We show that networks generated using the restructuring algorithm can outperform the EnvelopeNets used to generate them and that the generated networks can be discovered in less time than required by an incremental search based method. The results indicate that the structure of the network matters, and that the restructuring algorithm can identify improved structures.

## REFERENCES


[1] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. 2013. Provable Bounds for Learning Some Deep Representations. abs/1310.6343 (2013). arXiv:1310.6343 http://arxiv.org/abs/1310.6343

[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.* 13, 1 (Feb. 2012), 281–305. http://dl.acm.org/citation.cfm?id=2503308.2188395

[3] François Chollet. 2016. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR* abs/1610.02357 (2016). http://arxiv.org/abs/1610.02357

[4] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. 2017. Simple And Efficient Architecture Search for Convolutional Neural Networks. abs/1711.04528 (2017). arXiv:1711.04528 http://arxiv.org/abs/1711.04528

[5] D. Floreano, P. Drr, and C. Mattiussi. 2008. Neuroevolution: from architectures to learning. Evolutionary Intelligence (2008). https://doi.org/10.1007/s12065-007-0002-4

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[7] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely Connected Convolutional Networks. *CoRR* abs/1608.06993 (2016). arXiv:1608.06993 http://arxiv.org/abs/1608.06993

[8] A Krizhevsky and G Hinton. 2009. Learning multiple layers of features from tiny images. 1 (01 2009).

[9] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning Filters for Efficient ConvNets. *CoRR* abs/1608.08710 (2016). arXiv:1608.08710 http://arxiv.org/abs/1608.08710

[10] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network In Network. *CoRR* abs/1312.4400 (2013). arXiv:1312.4400 http://arxiv.org/abs/1312.4400

[11] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. 2017. Progressive Neural Architecture Search. *ArXiv e-prints*. arXiv:cs.CV/1712.00559

[12] Deepak Mittal, Shweta Bhardwaj, Mitesh M. Khapra, and Balaraman Ravindran. 2018. Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks. abs/1801.10447 (2018). arXiv:1801.10447 http://arxiv.org/abs/1801.10447

[13] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning. *CoRR* abs/1611.06440 (2016). arXiv:1611.06440 http://arxiv.org/abs/1611.06440

[14] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. abs/1802.03268 (2018). arXiv:1802.03268 http://arxiv.org/abs/1802.03268

[15] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2018. Regularized Evolution for Image Classifier Architecture Search. *CoRR* abs/1802.01548 (2018). http://arxiv.org/abs/1802.01548

[16] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc V. Le, and Alex Kurakin. 2017. Large-Scale Evolution of Image Classifiers. *CoRR* abs/1703.01041. arXiv:1703.01041 http://arxiv.org/abs/1703.01041

[17] Aruni RoyChowdhury, Prakhar Sharma, Erik G. Learned-Miller, and Aruni Roy. 2018. Reducing Duplicate Filters in Deep Neural Networks.

[18] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper with Convolutions. *CVPR* (2015). http://arxiv.org/abs/1409.4842

[20] Matthew D. Zeiler and Rob Fergus. 2013. Visualizing and Understanding Convolutional Networks. *CoRR* abs/1311.2901 (2013). arXiv:1311.2901 http://arxiv.org/abs/1311.2901

[21] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR 2017*. https://arxiv.org/abs/1611.01578

[22] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning Transferable Architectures for Scalable Image Recognition. *CoRR* abs/1707.07012. arXiv:1707.07012 http://arxiv.org/abs/1707.07012