Exploiting the Potential of Standard Convolutional Autoencoders for Image Restoration by Evolutionary Search

Masanori Suganuma [1 2] **Mete Ozay** [2] **Takayuki Okatani** [2 1]

Abstract

Researchers have applied deep neural networks to image restoration tasks, in which they proposed various network architectures, loss functions, and training methods. In particular, adversarial training, which is employed in recent studies, seems to be a key ingredient to success. In this paper, we show that simple convolutional autoencoders (CAEs) built upon only standard network components, i.e., convolutional layers and skip connections, can outperform the state-of-the-art methods which employ adversarial training and sophisticated loss functions. The secret is to search for good architectures using an evolutionary algorithm. All we did was to train the optimized CAEs by minimizing the ℓ_2 loss between reconstructed images and their ground truths using the ADAM optimizer. Our experimental results show that this approach achieves 27.8 dB peak signal to noise ratio (PSNR) on the CelebA dataset and 33.3 dB on the SVHN dataset, compared to 22.8 dB and 19.0 dB provided by the former state-of-the-art methods, respectively.

1. Introduction

The task of image restoration, which is to recover a clean image from its corrupted version, is usually an ill-posed inverse problem. In order to resolve or mitigate its ill-posedness, researchers have incorporated image priors such as edge statistics (Fattal, 2007), total variation (Perrone & Favaro, 2014), and sparse representation (Aharon et al., 2006; Yang et al., 2010), which are built on intuition or statistics of natural images. Recently, learning-based methods which use convolutional neural networks (CNNs) (LeCun et al., 1998; Krizhevsky et al., 2012) were introduced to overcome the limitation of these hand-designed or simple priors, and have significantly improved the state-of-the-art.

In these studies, researchers have approached the problem mainly from two directions. One is to design new network architectures and/or new loss functions. The other is to develop new training methods, such as the employment of adversarial training (Goodfellow et al., 2014). Later studies naturally proposed more complicated architectures to improve the performance of earlier architectures. Mao et al. (2016) proposed an architecture consisting of a chain of symmetric convolutional and deconvolutional layers, between which they added skip connections (Srivastava et al., 2015; He et al., 2016). Tai et al. (2017) proposed an 80-layer memory network which contains a recursive unit and a gate unit. Yang et al. (2017) proposed an image inpainting framework that uses two networks: one for capturing the global structure of an image, and one for reducing the discrepancy of texture appearance inside and outside missing image regions. While many studies employ the ℓ_2 distance between the clean and recovered images, some propose to use new loss functions such as the perceptual loss to obtain perceptually better results (Johnson et al., 2016; Ledig et al., 2017).

There are also studies on the development of new training methods. A recent trend is to use adversarial training, where two networks are trained in an adversarial setting; a generator network is trained to perform image restoration, and a discriminator network is trained to distinguish whether an input is a true image or a restored one. The first work employing this framework for image inpainting is the *context encoder* of Pathak et al. (2016). They minimize the sum of a reconstruction loss over an encoder-decoder network for restoring intensities of missing pixels and additionally an adversarial loss over a discriminator network. Iizuka et al. (2017) proposed an improved framework in which global and local context discriminators are used to generate realistic images. While the above studies require the shapes of missing regions (i.e., masks) for training, Yeh et al. (2017a) proposed a method which does not need masks for training. Their method first learns a latent manifold of clean images by GANs and search for the closest encoding of a corrupted image to infer missing regions. Despite its

[1]RIKEN, Tokyo, Japan [2]Tohoku University, Sendai, Japan. Correspondence to: Masanori Suganuma <suganuma@vision.is.tohoku.ac.jp>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

success in various application domains, GANs have several issues, such as difficulty of training (e.g., mode collapse), difficulty with evaluation of generated samples (Lucic et al., 2017), and theoretical limitations (Arora et al., 2017).

A question arises from these recent developments: *what is (the most) important of these ingredients, i.e., the design of network architectures, loss functions, and adversarial training?* In this study, we report that convolutional autoencoders (CAEs) built only on standard components can outperform the existing methods on standard benchmark tests of image restoration. We achieve this by employing an evolutionary algorithm (Suganuma et al., 2017) to exploit the potential of standard CAEs, which optimizes the number and size of filters and connections of each layer along with the total number of layers. We did not use adversarial training or any sophisticated loss; all we did was to train the discovered architecture with the standard ℓ_2 loss using the ADAM optimizer (Kingma & Ba, 2015). The contribution of this study is summarized as follows:

- We show that simple CAEs built upon standard components such as convolutional layers and skip connections can achieve the state-of-the-art performance in image restoration tasks. Their training is performed by minimization of a standard ℓ_2 loss; no adversarial training or novel hand-designed loss is used.
- We propose to use an evolutionary algorithm to search for *good* architectures of the CAEs, where the hyperparameters of each layer and connections of the layers are optimized.
- To the best of our knowledge, this is the first study of automatic architecture search for image restoration tasks. Previous studies proposed methods for image classification and tested them on the task.

2. Related Work

2.1. Deep Learning for Image Restoration

Deep networks have shown good performance on various image restoration tasks, such as image denoising (Xie et al., 2012; Zhang. et al., 2017), single image super-resolution (SISR) (Dong et al., 2014; Ledig et al., 2017), deblurring and compressive sensing (Xu et al., 2014; Kulkarni et al., 2016; Mousavi & Baraniuk, 2017), in addition to those mentioned in Section 1. In particular, recent studies tend to rely on the framework of GANs (Goodfellow et al., 2014) for training to improve accuracy or perceptual quality of restored images, e.g., (Pathak et al., 2016; Yeh et al., 2017a).

2.2. Automatic Design of Network Architectures

Neural networks have been and are being designed manually, sometimes with a lot of trial and error. Recently, increasing attention is being paid to automatic design of network architectures and hyperparameters (Miikkulainen et al., 2017; Xie & Yuille, 2017; Liu et al., 2017; Brock et al., 2018; Liu et al., 2018). The recent studies are roughly divided into two categories; those based on evolutionary algorithms and on reinforcement learning.

The employment of evolutionary algorithms for neural architecture search has a long history (Schaffer et al., 1992; Stanley & Miikkulainen, 2002). In the past, the weights and connections of neural networks are attempted to be jointly optimized, whereas in recent studies, only architectures are optimized by evolutionary algorithms, and their weights are left to optimization by SGD and its variants. Real et al. (2017) showed that evolutionary algorithms can explore the space of large-scale neural networks, and achieve competitive performance in standard object classification datasets, although their method relies on large computational resources (e.g., a few hundred GPUs and ten days). Suganuma et al. (2017) proposed a designing method based on cartesian genetic programming (Miller & Thomson, 2000), showing that architectural search can be performed using two GPUs in ten days.

Another approach to neural architecture search is to use reinforcement learning. There are studies that employ the REINFORCE algorithm, policy gradient, and Q-learning to learn network topology (Zoph & Le, 2017; Baker et al., 2017; Zhong et al., 2017; Zoph et al., 2017). These reinforcement learning-based approaches tend to be computational resource hungry, e.g., requiring 10-800 GPUs.

In this study, we employ the method of Suganuma et al. (2017) due to its computational efficiency, although we think that other recent light-weight methods could also be employed. As their method was tested only on classification tasks as in other similar studies, we tailor it to designing CAEs for image restoration tasks. As will be described, we confine the search space to symmetric CAEs, by which we make it possible to design competitive architectures with a limited amount of computational resource (using 1 to 4 GPUs in a few days).

2.3. Evaluation Methods for Image Restoration

There is a growing tendency that perceptual quality rather than signal accuracy is employed for evaluation of image restoration methods (Ledig et al., 2017; Yeh et al., 2017a). The shared view seems to be that employment of adversarial training and/or sophisticated loss such as the perceptual loss tends to deliver sharper and more realistic images, while their pixel-to-pixel differences (e.g., PSNR) from

their ground truths tend not to be smaller (or sometimes even larger). In this study, however, we stick to the pixel-to-pixel difference due to the following reasons. First, which evaluation measure should be used depends on for which purpose we use these "image restoration" methods. For a photo-editing software, looking more photo-realistic will be more important. For the purpose of 'pure' image restoration in which the goal is to predict intensities of missing pixels, it will be more important that each pixel has a value closer to its ground truth (see examples of inpainting images of numbers in Figure 2). Second, popular quality measures, such as mean opinion score (MOS), need human raters, and their values are not easy to reproduce particularly when there are only small differences between images under comparison. Finally, we also note that our method does sometimes provide sharper images compared to existing methods (see examples of inpainting images with random pixel masks in Figure 2).

3. Evolutionary Convolutional Autoencoders

3.1. Search Space of Network Architectures

We consider convolutional autoencoders (CAEs) which are built only on standard building blocks of ConvNets, i.e., convolutional layers with optional downsampling and skip connections. We further limit our attention to *symmetric* CAEs such that their first half (encoder part) is symmetric to the second half (decoder part). We add a final layer to obtain images of fixed channels (i.e., single-channel grayscale or three-channel color images) on top of the decoder part, for which either one or three filters of 3×3 size are used. Therefore, specification of the encoder part of a CAE solely determines its entire architecture. The encoder part can have an arbitrary number of convolutional layers up to a specified maximum. Each convolutional layer can have an arbitrary number and size of (single-size) filters, and is followed by ReLU (Nair & Hinton, 2010). Additionally, it can have an optional skip connection (Srivastava et al., 2015; He et al., 2016; Mao et al., 2016), which connects the layer to its mirrored counterpart in the decoder part. To be specific, the output feature maps (obtained after ReLU) of the layer are passed to and are element-wise added to the output feature maps (obtained before ReLU) of the counterpart layer. We can use additional downsampling after each convolutional layer depending on tasks; whether to use downsampling is determined in advance, and thus is not selected by architectural search, as will be explained later.

3.2. Representation of CAE Architectures

Following (Suganuma et al., 2017), we represent architectures of CAEs by directed acyclic graphs defined on a two-dimensional grid. This graph is optimized by the evolutionary algorithm explained below, where the graph is called phenotype, and is encoded by a data structure called genotype (Eiben & Smith, 2003).

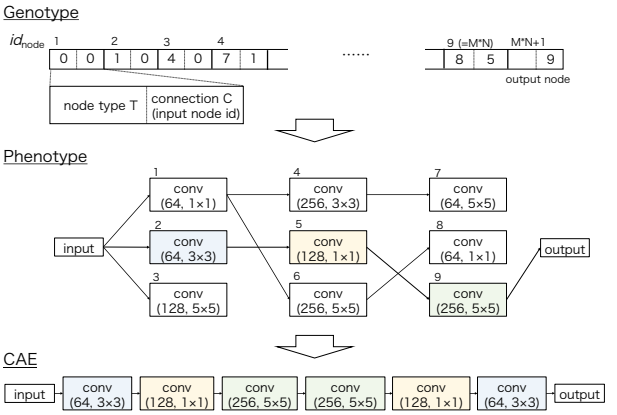


Figure 1. An example of a genotype and a phenotype. A phenotype is a graph representation of a network architecture, and a genotype encodes a phenotype. They encode only the encoder part of a CAE, and its decoder part is automatically created so as to be symmetrical to the encoder part. In this example, the phenotype is defined on the grid of three rows and three columns.

Phenotype A phenotype is a directed acyclic graph defined on a two-dimensional grid of M rows and N columns; see Figure.1. Each node of the graph, which is identified by a unique id_{node} in the range $[1, MN]$ in a column-major order of the grid, represents a convolutional layer followed by a ReLU in a CAE. An edge connecting two nodes represents the connectivity of the two corresponding layers. The graph has two additional special nodes called input and output nodes; the former represents the input layer of the CAE, and the latter represents the output of the encoder part, or equivalently the input of the decoder part of the CAE. As the input of each node is connected to at most one node, there is a single unique path starting from the input node and ending at the output node. This unique path identifies the architecture of the CAE, as shown in the middle row of Figure 1. Note that nodes depicted in the neighboring two columns are not necessarily connected. Thus, the CAE can have different number of layers depending on how their nodes are connected. Since the maximum number of layers (of the encoder part) of the CAE is N, the total number of layers is $2N + 1$ including the output layer. In order to control how the number of layers will be chosen, we introduce a hyper-parameter called level-back L, such that nodes given in the n-th column are allowed to be connected from nodes given in the columns ranging from $n - L$ to $n - 1$. If we use smaller L, then the resulting CAEs will tend to be deeper.

Genotype A genotype encodes a phenotype, and is manipulated by the evolutionary algorithm. The genotype encoding a phenotype with M rows and N columns has $MN + 1$

genes, each of which represents attributes of a node with two integers (i.e., type T and connection C). The type T specifies the number F and size k of filters of the node, and whether the layer has skip connections or not, by an integer encoding their combination. The connection C specifies the node by id_{node} that is connected to the input of this node. The last $(MN+1)$-st gene represents the output node, which stores only connection C determining the node connected to the output node. An example of a genotype is given at the top row of Figure 1, where $F \in \{64, 128, 256\}$ and $k \in \{1\times 1, 3\times 3, 5\times 5\}$.

3.3. Evolutionary Strategy

We use a simple form of the $(1+\lambda)$ evolutionary strategy (Miller & Thomson, 2000) to perform search in the architecture space. In this strategy, λ children are generated from a single parent at each generation, and the best performing child compared to its parent becomes the new parent at the next generation. The performance of each individual (i.e., a generated CAE), called *fitness*, is measured by peak signal to noise ratio (PSNR) between the restored and ground truth images evaluated on the validation dataset. The genotype is updated to maximize the fitness as generation proceeds.

The details are given in Algorithm 1. The algorithm starts with an initial parent, which is chosen to be a minimal CAE having a single convolution layer and a single deconvolution layer.

At each generation, λ children are generated by applying mutations to the parent (line 5). We use a point mutation as the genetic operator, where integer values of the type T and connection C of each gene are randomly changed with a mutation probability r. If a gene is decided to be changed, the mutation operator chooses a value at random for each T and C from their predefined sets.

The generated λ children are individually trained using the training set. We train each child for I iterations using the ADAM optimizer (Kingma & Ba, 2015) with learning rate lr, and a mini-batch size of b (line 6). For the training loss, we use the mean squared error (MSE) between the restored images and their ground truths. After the training phase is completed, the performance of each child is evaluated on the validation set and is assigned to its fitness value (line 7). Finally, the best individual is selected from the set of parent and the children, and replaced the parent in the next generation (line $9-12$). This procedure is repeated for G generations.

We can obtain a single unique path starting from the input node and ending at the output node using our representation. The computed unique path represents the architecture of the CAE. We call nodes on this path functioning nodes. As some (in fact, most) of nodes in a phenotype are not functioning nodes and do not express the resulting CAE, the mutation has the possibility of affecting only non-functioning nodes, i.e., the CAE architecture does not change by the mutation. In that case, we skip the evaluation of the CAE and apply the mutation operator repeatedly until the resulting CAE architecture does change. Moreover, if the fitness values of the children do not improve, then we modify a parent (Miller & Thomson, 2000; Miller & Smith, 2006); in this case, we change only the non-functioning nodes so that the realized CAE (i.e., functioning nodes) will not change (line 14).

Algorithm 1 Evolutionary strategy for a CAE.

1: **Input:** G (number of generations), r (mutation probability), λ (children size), S (Training set), V (Validation set).
2: **Initialization:** (i) Generate a $parent$, (ii) train the model on the S, and (iii) assign the fitness F_p using the set V.
3: **while** $generation < G$ **do**
4: **for** $i = 1$ **to** λ **do**
5: $children_i \leftarrow$ Mutation($parent, r$)
6: $model_i \leftarrow$ Train($children_i, S$)
7: $fitness_i \leftarrow$ Evaluate($model_i, V$)
8: **end for**
9: $best \leftarrow \text{argmax}_{i=1,2,\ldots,\lambda}\ \{fitness_i\}$
10: **if** $fitness_{best} > F_p$ **then**
11: $parent \leftarrow children_{best}$
12: $F_p \leftarrow fitness_{best}$
13: **else**
14: $parent \leftarrow$ Modify($parent, r$)
15: **end if**
16: $generation = generation + 1$
17: **end while**
18: **Output:** $parent$ (the best architecture of CAEs found by the evolutionary search).

4. Experimental Results

We conducted experiments to test the effectiveness of our approach. We chose two tasks, image inpainting and denoising.

4.1. Details of Experiments

4.1.1. Inpainting

We followed the procedures suggested in (Yeh et al., 2017a) for experimental design. We used three benchmark datasets; the CelebFaces Attributes Dataset (CelebA) (Liu et al., 2015), the Stanford Cars Dataset (Cars) (Krause et al., 2013), and the Street View House Numbers (SVHN) (Netzer et al., 2011). The CelebA contains $202,599$ images, from which

we randomly selected $100,000$, 1,000, and 2,000 images for training, validation, and test, respectively. All images were cropped in order to properly contain the entire face, and resized to 64×64 pixels. For Cars and SVHN, we used the provided training and testing split. The images of Cars were cropped according to the provided bounding boxes, and resized to 64×64 pixels. The images of SVHN were resized to 64×64 pixels.

We generated images with missing regions of the following three types: a central square block mask (*Center*), random pixel masks such that 80% of all the pixels were randomly masked (*Pixel*), and half image masks such that a randomly chosen vertical or horizontal half of the image was masked (*Half*). For the latter two, a mask was randomly generated for each training minibatch and for each test image.

Considering the nature of this task, we consider CAEs endowed with downsampling. To be specific, the same counts of downsampling and upsampling with stride $= 2$ were employed such that the entire network has a symmetric hourglass shape. For simplicity, we used a skip connection and downsampling in an exclusive manner; in other words, every layer (in the encoder part) employed either a skip connection or downsampling.

4.1.2. Denoising

We followed the experimental procedures described in (Mao et al., 2016; Tai et al., 2017). We used grayscale 300 and 200 images belonging to the BSD500 dataset (Martin et al., 2001) to generate training and test images, respectively. For each image, we randomly extracted 64×64 patches, to each of which Gaussian noise with different $\sigma = 30, 50$ and 70 are added. As utilized in the previous studies, we trained a single model for all different noise levels.

For this task, we used CAE models without downsampling following the previous studies (Mao et al., 2016; Tai et al., 2017). We zero-padded the input feature maps computed in each convolution layer not to change the size of input and output feature space of the layer.

4.2. Configurations of Architectural Search

For the proposed evolutionary algorithm, we chose the mutation probability as $r = 0.1$, the number of children as $\lambda = 4$, and the number of generations as $G = 250$. For the phenotype, we used the graph with $M = 3$, $N = 20$ and level-back $L = 5$. For the number F and size k of filters at each layer, we chose them from $\{64, 128, 256\}$ and $\{1 \times 1, 3 \times 3, 5 \times 5\}$, respectively. During an evolution process, we trained each CAE for $I = 20$k iterations with a mini-batch of size $b = 16$. We set the learning rate lr of the ADAM optimizer to be 0.001. Following completion of the evolution process, we fine-tuned the best CAE using the training set of images for additional 500k iterations, in which the learning rate is reduced by a factor of 10 at the 200k and 400k iterations. We then calculated its performance using the test set of images. We implemented our method using PyTorch (Paszke et al., 2017), and performed the experiments using four P100 GPUs. Execution of the evolutionary algorithm and the fine-tuning of the best model took about three days for the inpainting tasks and four days for the denoising tasks.

4.3. Comparison with Existing Methods

4.3.1. Inpainting

As mentioned above, we follow the experimental procedure employed in (Yeh et al., 2017a). In the paper, the authors reported the performances of their proposed method, Semantic Image Inpainting (SII), and Context Autoencoder (CE) (Pathak et al., 2016). However, we found that CE can provide considerably better results than those reported in (Yeh et al., 2017a) in terms of both PSNR and visual quality. Thus, we report here PSNR and SSIM values of CE that we obtained by running the authors' code[1]. In order to calculate SSIM values of SII, which were not reported in (Yeh et al., 2017a), we run the authors' code[2] for SII.

In order to further validate effectiveness of the evolutionary search, we evaluate two baseline architectures; one is the architecture generated by a random search (RANDOM), and the other is the architecture having the same depth as the best performing architecture found by our method but having the constant number (64) of fixed size (3×3) filters in each layer with a skip connection (BASE). In the random search, we generate ten architectures at random in the same search space, and report the average PSNR and SSIM values of them. All other experimental setups are the same.

Table 1 shows the PSNR and SSIM values obtained using five methods on three datasets and three masking patterns. Our method (i.e., the CAE optimized by the evolutionary algorithm) is referred as E-CAE. We run the evolutionary algorithm three times, and report the average accuracy values of the three optimized CAEs. As we can see, our method outperforms the other four methods for each of the dataset-mask combinations. It should also be noted that CE and SII use mask patterns for inference; to be specific, their networks estimate only pixel intensities of the missing regions specified by the provided masks, and then they are merged with the unmasked regions of clean pixels. Thus, the pixel intensities of unmasked regions are identical to their ground truths. On the other hand, our method does not use masks; it outputs a complete image such that the missing regions are hopefully inpainted correctly. We then calculate the

[1] https://github.com/pathak22/context-encoder

[2] https://github.com/moodoki/semantic_image_inpainting

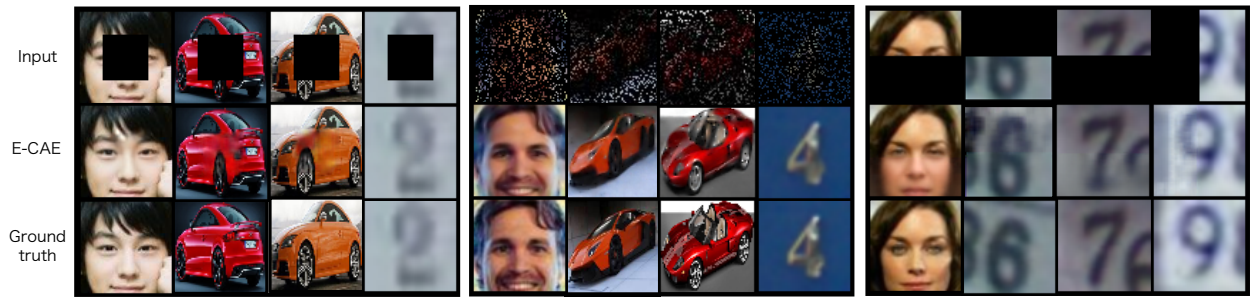


Figure 2. Examples of inpainting results obtained by E-CAE (CAEs designed by the evolutionary algorithm).

Table 1. **Inpainting results.** Comparison of two baseline architectures (RANDOM and BASE), Context Autoencoder (CE) (Pathak et al., 2016), Semantic Image Inpainting (SII) (Yeh et al., 2017a), and CAEs designed by our evolutionary algorithm (E-CAE) using three datasets and three masking patterns.

		PSNR					SSIM				
DATASET	TYPE	RANDOM	BASE	CE	SII	E-CAE	RANDOM	BASE	CE	SII	E-CAE
CELEBA	CENTER	15.3	27.1	28.5	19.4	**29.9**	0.740	0.883	0.912	0.907	**0.934**
	PIXEL	25.5	27.5	22.9	22.8	**27.8**	0.766	0.836	0.730	0.710	**0.887**
	HALF	12.7	11.8	19.9	13.7	**21.1**	0.549	0.604	0.747	0.582	**0.771**
CARS	CENTER	17.1	19.5	19.6	13.5	**20.9**	0.704	0.767	0.767	0.721	**0.846**
	PIXEL	17.0	19.2	15.6	18.9	**19.5**	0.533	0.679	0.408	0.412	**0.738**
	HALF	13.0	11.6	14.8	11.1	**16.2**	0.511	0.541	0.576	0.525	**0.610**
SVHN	CENTER	23.5	29.9	16.4	19.0	**33.3**	0.819	0.895	0.791	0.825	**0.953**
	PIXEL	29.0	40.1	30.5	33.0	**40.4**	0.687	0.899	0.888	0.786	**0.969**
	HALF	11.3	12.9	21.6	14.6	**24.8**	0.574	0.617	0.756	0.702	**0.848**

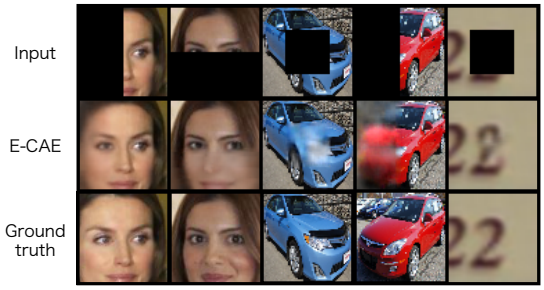


Figure 3. Examples of blurry reconstructions generated by E-CAE.

PSNR of the output image against the ground truth without identifying missing regions. This difference should favor CE and SII, and nevertheless our method performs better.

Sample inpainted images obtained by E-CAE along with the masked inputs, and the ground truths are shown in Figure 2. As we choose the same images as those used in (Yeh et al., 2017a), the readers can easily check differences in visual quality from CE and SII. It is observed overall that E-CAE performs stably; the output images do not have large errors for all types of masks. It performs particularly well for random pixel masks (the middle column of Figure 2); the images are realistic and sharp. It is also observed that E-CAE tends to yield less sharp images for images with a filled region of missing pixels. However, E-CAE can infer their contents accurately, as shown in the examples of inpainting images of numbers (the rightmost column of Figure 2); CE and SII provide either obscure images of numbers which are difficult to recognize, or sharp images of wrong numbers; see Figure 18 and 21 of (Yeh et al., 2017b). Figure 3 shows several examples of difficult cases for E-CAE.

4.3.2. DENOISING

We compare our method with two baseline architectures (i.e., RANDOM and BASE described in Section 4.3.1) and two state-of-the-art methods; RED (Mao et al., 2016) and MemNet (Tai et al., 2017). Table 2 shows PSNR and SSIM values for three versions of the BSD200 test set with different noise levels $\sigma = 30, 50$, and 70, where the performance values of RED and MemNet are obtained from (Tai et al., 2017). Our method again achieves the best performance for all cases except a single case (MemNet for $\sigma = 30$). It is worth noting that the networks of RED and MemNet have 30 and 80 layers, respectively, whereas our best CAE has only 15 layers (including the decoder part and the output layer), showing that our evolutionary method was able to find simpler architectures that can provide more accurate results.

An example of an image recovered by our method is shown

Table 2. **Denoising results on BSD200.** Comparison of results of two baseline architectures (RANDOM and BASE), RED (Mao et al., 2016), MemNet (Tai et al., 2017), and E-CAE.

	PSNR					SSIM				
NOISE σ	RANDOM	BASE	RED	MEMNET	E-CAE	RANDOM	BASE	RED	MEMNET	E-CAE
30	27.25	27.00	27.95	28.04	**28.23**	0.7491	0.7414	0.8019	**0.8053**	0.8047
50	25.11	24.88	25.75	25.86	**26.17**	0.6468	0.6229	0.7167	0.7202	**0.7255**
70	23.50	23.22	24.37	24.53	**24.83**	0.5658	0.5349	0.6551	0.6608	**0.6636**

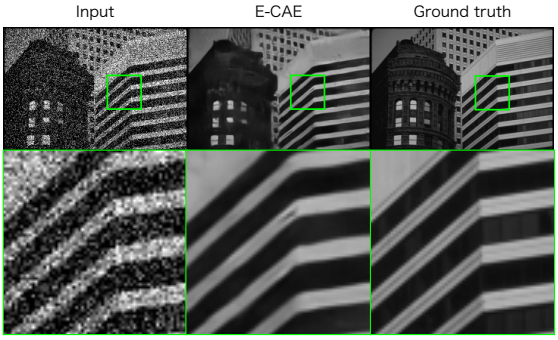


Figure 4. Examples of images reconstructed by E-CAE for the denoising task. The first column shows the input image with noise level $\sigma = 50$.

in Figure 4. As we can see, E-CAE correctly removes the noise, and produces an image as sharp as the ground truth.

4.4. Analysis of Optimized Architectures

Table 3 shows the top five best performing architectures designed by our method for the image inpainting task using center masks on the CelebA dataset and the denoising task, along with their performances measured on their test datasets. One of the architectures best performing for each task is shown in Figure 5. It is observed that although their overall structures do not look very unique, mostly due to the limited search space of CAEs, *the number and size of filters are quite different across layers, which are hard to manually determine*. Although it is difficult to give a general interpretation of why the parameters of each layer are chosen, we can make the following observations: i) regardless of the tasks, almost all networks have a skip connection at the first layer, implying that the input images contain essential information to yield accurate outputs; ii) 1×1 convolution seems to be important ingredients for both tasks; 1×1 conv. layers dominate the denoising networks, and all the inpainting networks employ two 1×1 conv. layers; iii) when comparing the inpainting networks with the denoising networks, we observe the following differences: the largest filters of size 5×5 tend to be employed by the former more often than the latter (2.8 vs 0.8 layers in average), and 1×1 filters tend to be employed by the former less often than the latter (2.0 vs. 3.2 layers in average).

4.5. Effects of Parameters of Evolutionary Search

The evolutionary algorithm has several parameters, two of which, i.e., the mutation probability (r) and the number of children (λ), tend to have particularly large impact on the performance of the optimized E-CAEs. Using the center mask inpainting task on the CelebA dataset, we analyze their impact in detail in this subsection.

Effect of mutation probability Employment of a larger mutation probability (r) will change the structures of CAEs more drastically at each generation, and make the process of architecture search less stable. On the other hand, a large mutation probability will contribute to reduce the possibility of being trapped in local optima. Figure 6 (a) shows the relation between different mutation probabilities and the performances of CAEs obtained by using them; their performances are calculated on the validation set. It is observed from the plots that smaller mutation probabilities tend to deliver lower accuracy at initial generations, but eventually provide higher accuracy after a sufficient number of generations are generated. The best result was obtained for $r = 0.1$.

Effect of number of children Employment of a larger number (λ) of children will enable us to perform search in a wider subspace of the architecture space at each generation, but at the expense of larger computational cost per generation. Figure 6 (b) shows the relation between different λ values ($\lambda = 1, 2, 4, 8$, and 16) and the performances of the optimized CAEs. The best performance is obtained for $\lambda = 4$ using a sufficient number of generations, but there is not much difference in the final PSNR results obtained by different number of children. Interestingly, even the evolution performed using $\lambda = 1$, which uses the minimum computational cost per generation, yields a competitive result. Specifically, it took 1.68 days on one P100 GPU for training, and achieved PSNR $= 29.80$ on the test set after fine-tuning of the model.

5. Conclusion

In this paper, we have first introduced an evolutionary algorithm that searches for *good* architectures of convolutional autoencoders (CAEs) for image restoration tasks. We have then shown that the CAEs designed by our algorithm

Table 3. **Best performing five architectures of E-CAE.** $C(F,k)$ indicates that the layer has F filters of size $k \times k$ without a skip connection. CS indicates that the layer has a skip connection. This table shows only the encoder part of CAEs. For the denoising, the average values of PSNR and SSIM of three noise levels are shown.

Architecture (Inpainting)	PSNR	SSIM
$CS(128,3)-C(64,3)-CS(128,5)-C(128,1)-CS(256,5)-C(256,1)-CS(64,5)$	29.91	0.9344
$C(256,3)-CS(64,1)-C(128,3)-CS(256,5)-CS(64,1)-C(64,3)-CS(128,5)$	29.91	0.9343
$CS(128,5)-CS(256,3)-C(64,1)-CS(128,3)-CS(64,5)-CS(64,1)-C(128,5)-C(256,5)$	29.89	0.9334
$CS(128,3)-CS(64,3)-C(64,5)-CS(256,3)-C(128,3)-CS(128,5)-CS(64,1)-CS(64,1)$	29.88	0.9346
$CS(64,1)-C(128,5)-CS(64,3)-C(64,1)-CS(256,5)-C(128,5)$	29.63	0.9308

Architecture (Denoising)	PSNR	SSIM
$CS(64,3)-C(64,1)-C(128,3)-CS(64,1)-CS(128,5)-C(128,3)-C(64,1)$	26.67	0.7313
$CS(64,5)-CS(256,1)-C(256,1)-C(64,3)-CS(128,1)-C(64,3)-CS(128,1)-C(128,3)$	26.28	0.7113
$CS(64,3)-C(64,1)-C(128,3)-CS(64,1)-CS(128,5)-C(128,3)-C(64,1)$	26.28	0.7107
$CS(128,3)-CS(64,1)-C(64,3)-C(64,3)-CS(64,1)-C(64,3)$	26.20	0.7047
$CS(64,5)-CS(128,1)-CS(256,3)-CS(128,1)-CS(128,1)-C(64,1)-CS(64,3)$	26.18	0.7037

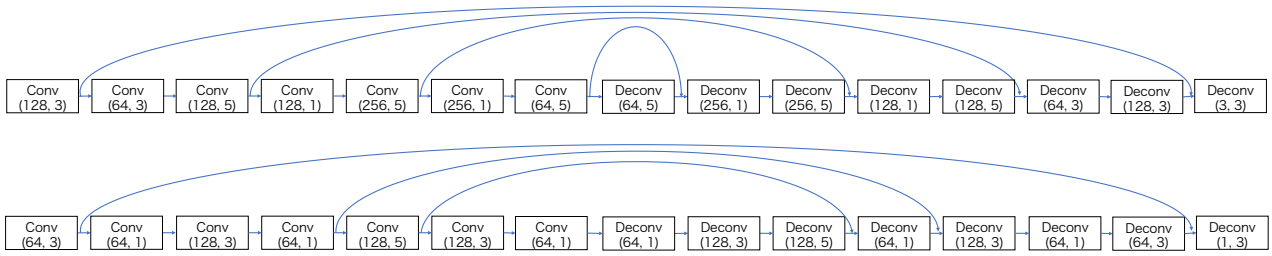
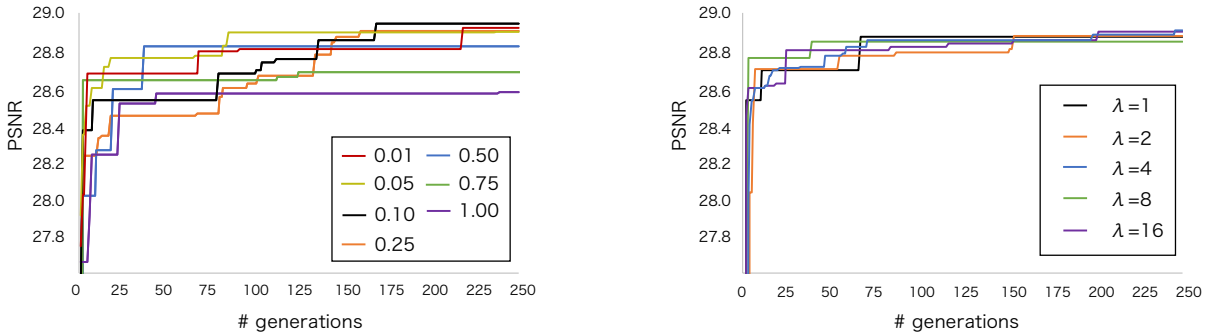


Figure 5. One of the best performing architectures given in Table 3 for inpainting (upper) and denoising (lower) tasks.



(a) Mutation probability (b) The number of children

Figure 6. Improvement of PSNR of E-CAE by increasing number of generations obtained using the evolutionary algorithm for (a) different mutation probabilities, and (b) different number of children. The center mask inpainting task on the CelebA dataset is used. PSNR is calculated using the validation set.

outperform the state-of-the-art networks for image inpainting and denoising, despite the fact that these networks are built on combination of complicated architectures with very deep layers, (multiple) hand-designed losses, and adversarial training; our CAEs consist only of standard convolutional layers with optional skip connections, and they are simply trained by the ADAM optimizer to minimize standard ℓ_2 loss. Although our CAEs have simple architectures, their space is still very high-dimensional; CAEs can have an arbitrary number of layers, each of which has an arbitrary number and size of filters as well as whether to use a skip connection. Our evolutionary algorithm can find good architectures in this high-dimensional space. This implies that there is still much room for exploration of search spaces of architectures of classical convolutional networks, which may apply to other tasks such as single image colorization (Zhang et al., 2016), depth estimation (Eigen et al., 2014; Xu et al., 2017), and optical flow estimation (Ilg et al., 2017).

Acknowledgements

This work was partly supported by CREST, JST Grant Number JPMJCR14D1, and the ImPACT Program "Tough Robotics Challenge" of the Council for Science, Technology, and Innovation (Cabinet Office, Government of Japan).

References

Aharon, M., Elad, M., and Bruckstein, A. k-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

Arora, S., Ge, R., Liang, Y., Ma, T., and Zhang, Y. Generalization and equilibrium in generative adversarial nets (gans). In *ICML*, pp. 224–232, 2017.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Smash: one-shot model architecture search through hypernetworks. In *ICLR*, 2018.

Dong, C., Loy, C. C., He, K., and Tang, X. Learning a deep convolutional network for image super-resolution. In *ECCV*, pp. 184–199, 2014.

Eiben, A. E. and Smith, J. E. *Introduction to Evolutionary Computing*, volume 53. SpringerVerlag, 2003.

Eigen, D., Puhrsch, C., and Fergus, R. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, pp. 2366–2374, 2014.

Fattal, R. Image upsampling via imposed edge statistics. In *SIGGRAPH*, 2007.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NIPS*, pp. 2672–2680, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Iizuka, S., Simo-Serra, E., and Ishikawa, H. Globally and locally consistent image completion. In *SIGGRAPH*, pp. 107:1–107:14, 2017.

Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., and Brox, T. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *CVPR*, 2017.

Johnson, J., Alahi, A., and Fei-Fei, L. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, pp. 694–711, 2016.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Krause, J., Stark, M., Deng, J., and Fei-Fei, L. 3d object representations for fine-grained categorization. In *ICCV Workshops (ICCVW)*, pp. 554–561, 2013.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105, 2012.

Kulkarni, K., Lohit, S., Turaga, P., Kerviche, R., and Ashok, A. Reconnet: Non-iterative reconstruction of images from compressively sensed measurements. In *CVPR*, pp. 449–458, 2016.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, pp. 4681–4690, 2017.

Liu, C., Zoph, B., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. *arXiv:1712.00559*, 2017.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *ICCV*, pp. 3730–3738, 2015.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. Are gans created equal? a large-scale study. *arXiv:1711.10337*, 2017.

Mao, X., Shen, C., and Yang, Y. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *NIPS*, pp. 2802–2810, 2016.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, pp. 416–423, 2001.

Miikkulainen, R., Liang, J. Z., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. Evolving deep neural networks. In *GECCO*, 2017.

Miller, J. F. and Smith, S. L. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

Miller, J. F. and Thomson, P. Cartesian genetic programming. In *EuroGP*, pp. 121–132, 2000.

Mousavi, A. and Baraniuk, R. G. Learning to invert: Signal recovery via deep convolutional networks. In *ICASSP*, pp. 2272–2276, 2017.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *ICML*, pp. 807–814, 2010.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, 2011.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017.

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. Context encoders: Feature learning by inpainting. In *CVPR*, pp. 2536–2544, 2016.

Perrone, D. and Favaro, P. Total variation blind deconvolution: The devil is in the details. In *CVPR*, pp. 2909–2916, 2014.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *ICML*, pp. 2902–2911, 2017.

Schaffer, J. D., Whitley, D., and Eshelman, L. J. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *COGANN*, pp. 1–37, 1992.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Training very deep networks. In *NIPS*, pp. 2377–2385, 2015.

Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

Suganuma, M., Shirakawa, S., and Nagao, T. A genetic programming approach to designing convolutional neural network architectures. In *GECCO*, pp. 497–504, 2017.

Tai, Y., Yang, J., Liu, X., and Xu, C. Memnet: A persistent memory network for image restoration. In *CVPR*, pp. 4539–4547, 2017.

Xie, J., Xu, L., and Chen, E. Image denoising and inpainting with deep neural networks. In *NIPS*, pp. 341–349, 2012.

Xie, L. and Yuille, A. L. Genetic CNN. In *ICCV*, pp. 1379–1388, 2017.

Xu, D., Ricci, E., Ouyang, W., Wang, X., and Sebe, N. Multi-scale continuous crfs as sequential deep networks for monocular depth estimation. In *CVPR*, pp. 5354–5362, 2017.

Xu, L., Ren, J. S., Liu, C., and Jia, J. Deep convolutional neural network for image deconvolution. In *NIPS*, pp. 1790–1798, 2014.

Yang, C., Lu, X., Lin, Z., Shechtman, E., Wang, O., and Li, H. High-resolution image inpainting using multi-scale neural patch synthesis. In *CVPR*, pp. 6721–6729, 2017.

Yang, J., Wright, J., Huang, T. S., and Ma, Y. Image super-resolution via sparse representation. *IEEE transactions on Image Processing*, 19(11):2861–2873, 2010.

Yeh, R. A., Chen, C., Lim, T. Y., Schwing, A. G., Hasegawa-Johnson, M., and Do, M. N. Semantic image inpainting with deep generative models. In *CVPR*, pp. 5485–5493, 2017a.

Yeh, R. A., Chen, C., Lim, T. Y., Schwing, A. G., Hasegawa-Johnson, M., and Do, M. N. Semantic image inpainting with deep generative models. *arXiv:1607.07539v3*, 2017b.

Zhang., K., Zuo, W., Chen, Y., Meng, D., and Zhang, L. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

Zhang, R., Isola, P., and Efros, A. A. Colorful image colorization. In *ECCV*, pp. 649–666, 2016.

Zhong, Z., Yan, J., and Liu, C. Practical network blocks design with Q-Learning. In *arXiv: 1708.05552*, 2017.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. *arXiv:1707.07012*, 2017.