PatchNAS: Repairing DNNs in Deployment with Patched Network Architecture Search

Yuchu Fang[1], Wenzhong Li[1]*, Yao Zeng[1], Yang Zheng[2], Zheng Hu[2], Sanglu Lu[1]

[1] State Key Laboratory for Novel Software Technology, Nanjing University
[2] TTE Lab, Huawei Technologies Co., Ltd.
{fangyuchu,zengyao}@smail.nju.edu.cn, {lwz,sanglu}@nju.edu.cn, {zhengyang31,hu.zheng}@huawei.com

Abstract

Despite being widely deployed in safety-critical applications such as autonomous driving and health care, deep neural networks (DNNs) still suffer from non-negligible reliability issues. Numerous works had reported that DNNs were vulnerable to either natural environmental noises or man-made adversarial noises. How to repair DNNs in deployment with noisy samples is a crucial topic for the robustness of neural networks. While many network repairing methods based on data argumentation and weight adjustment have been proposed, they require retraining and redeploying the whole model, which causes high overhead and are infeasible for varying faulty cases on different deployment environments. In this paper, we propose a novel network repairing framework called PatchNAS from the architecture perspective, where we freeze the pretrained DNNs and introduce a small patch network to deal with failure samples at runtime. PatchNAS introduces a novel network instrumentation method to determine the faulty stage of the network structure given the collected failure samples. Then a small patch network structure is searched unsupervisedly using neural architecture search (NAS) technique with data samples from deployment environment. The patch network repairs the DNNs by correcting the output feature maps of the faulty stage, which helps to maintain network performance on normal samples and enhance robustness in noisy environments. Extensive experiments based on several DNNs across 15 types of natural noises show that the proposed PatchNAS outperforms the state-of-the-arts with significant improvement as well as much lower deployment overhead.

Introduction

Recent years have witnessed the explosive deployment of Deep Neural Networks (DNNs) in production environments. With the wide range of applications in safety-critical tasks such as autonomous driving and healthcare, there is an urgent need to concern the robustness and trustworthy of DNNs. Numerous works had reported that neural networks suffered from reliability issues (Eykholt et al. 2018; Kurakin, Goodfellow, and Bengio 2018), and they were vulnerable to either natural noises (Hendrycks et al. 2021) or man-made adversarial noises (Goodfellow, Shlens, and Szegedy 2014; Madry et al. 2017). According to the Udacity self-driving car challenge (Tian et al. 2018), the changes in driving conditions such as fog and rain could lead to thousands of erroneous behaviors in three top performing DNNs. The work of (Yu et al. 2021) found that adding 15 different types of natural noises into clean images could seriously harm the classification accuracy of deployed DNNs, where more than 40% performance degradation were reported.

Typically, DNNs were trained with a pre-collected dataset before being deployed into real environment. Due to the fact that the runtime environment contains potential noises and unseen cases, the pre-trained DNNs may fail to make correct classifications or decisions. Therefore, it is important to repair the deployed DNNs with the failure samples collected at runtime. An intuitive method was to finetune the DNNs with the failure samples (Yun et al. 2019; Hendrycks et al. 2020). However, since the amount of available failure samples is usually small, finetuning DNNs with them easily results in underfitting (the networks still focus on original training data and neglect the faulty cases) or overfitting (the networks learn the irrelevant information from the failure samples). Some works proposed to repair DNNs by extracting common features from the failure samples and utilized them as an augmentation direction to create more data samples to retrain the DNNs (Gao et al. 2020; Ren et al. 2020; Yu et al. 2021; Ma et al. 2018; Zhang and Chan 2019; Sohn, Kang, and Yoo 2019).

Despite the great efforts, the existing network repairing methods encounter several drawbacks. Firstly, they required retraining the whole DNNs and redeploying them at runtime whenever faulty cases occur, causing a lot of overhead due to their exceptional large sizes. Secondly, they need to retrain personalized models for individual application whose deployment environments varies from device to device, substantially increasing the difficulty of maintenance. Thirdly, full model update will interrupt the AI services, inevitably influencing the service reliability and user experience.

To address these challenges, we propose a novel lightweight network repairing method called PatchNAS from the architecture perspective, where we freeze the pretrained DNNs (common module) and introduce a small patch network (personalized module) to handle varying faulty cases in runtime environment. Firstly, the proposed framework introduces a *neural network instrumentation* method, similar to program instrumentation in software engineering, that inserts probes into different stages of the neural network struc-

*The corresponding author is Wenzhong Li (lwz@nju.edu.cn).

ture to monitor its operation conditions and locate its weak stages for the runtime failure samples. Then, a *patch gate* module is trained to recognize faulty stages from the DNN, and attached the faulty stages to a *patch network* module for repairment. To cope with the difficulty of data annotation at runtime, the patch structure is automatically learned with an unsupervised neural architecture search (NAS) technique using unlabeled data from the deployment environment. The patch network is further trained with a small number of normal samples and failure samples, so that it learns to correct error output from failure samples and enhances the ability of the original model to combat with faulty cases. Since the patch network does not take part in the forward propagation of all inputs and it only deals with the probable failure samples detected by the patch gate, the proposed method is light-weight and cost-efficient. The advantage of PatchNAS is that the original DNN model remains intact during the repairing process, and only a small patch network need to be updated to deal with faulty cases from the deployment environment. With the proposed PatchNAS, the performance of DNNs on normal samples will not degrade, and their robustness will be enhanced against varying faulty cases.

The contributions of our work are summarized as follows:

- We propose to repair DNNs in deployment from a novel architecture perspective by adding a patch network whose structure is searched unsupervisedly using the neural architecture search (NAS) technique. The patch network repairs the neural network by correcting the output of the faulty stages in the DNNs. To the best of our knowledge, we are the first to incorporate a patch structure with NAS to achieve light-weight network repairing.
- To locate the weak/faulty stages in the neural network structure, we propose a novel network instrumentation method where probes are inserted for each stage of the DNN with a patch gate module trained to determine the probable faulty areas. The PatchNAS design helps to maintain network performance on normal samples and enhance the robustness against failure samples with the intervention of a small patch network structure.
- Extensive experiments showed that the proposed PatchNAS method outperforms the state-of-the-art network repairing methods on four widely-used DNN models across fifteen different types of noises, and the training and deployment costs are significantly reduced.

Related Work

We introduce the following three aspects that are related to our work: neural network robustness, neural network repair, and neural architecture search.

Neural Network Robustness Robustness has become a crucial issue for the application of neural networks. Both man-crafted adversarial noises (Goodfellow, Shlens, and Szegedy 2014; Madry et al. 2017) and noises in the natural environment can mislead the network into making wrong decisions, which imposes great threat to neural network applications like autonomous driving and traffic monitoring (Tian et al. 2018; Hendrycks and Dietterich 2019). Recently, neural network architecture has been found to be related to network robustness and different architectures have different resistance given a specific type of noise (Tang et al. 2021; Devaguptapu et al. 2021).

Neural Network Repair Network repair aims to repair a trained neural network given runtime data samples misclassified by the network. The works of CutMix and AugMix (Yun et al. 2019; Hendrycks et al. 2020) introduced data augmentation techniques to generate training samples following a pre-defined pattern like randomly masking or shearing the images, which can be used with failure samples to finetune the network to enhance robustness. FSGMix adopted a similar approach to AugMix by taking samples from a Gaussian mixture model learned from the failure cases (Ren et al. 2020). Yu et al. introduced a style transfer model to learn the unknown failure patterns from the runtime data and transferred the learned pattern to the existed training dataset to retrain the network (Yu et al. 2021). Another line of works repair the network by adjusting model weights according to their roles in the decision of the faulty cases. Zhang et al. adjusted the direction and magnitude of the model weights toward the average weight of a series identical models trained with reduced subsets of the original training dataset (Zhang and Chan 2019). Sohn et al. performed sensitivity based fault localization to locate the faulty neurons and directly optimized them with Particle Swarm Optimization until their behaviors were corrected (Sohn, Kang, and Yoo 2019).

Neural Architecture Search Neural Architecture Search (NAS) aims to search architectures that perform well on target data and tasks. Despite early efforts to utilize time-consuming reinforcement learning (Zoph et al. 2018) or evolution algorithm (Real et al. 2019) to search the structure, recent NAS methods usually adopt a more efficient way to train a hyper network in the search phase (Liu, Simonyan, and Yang 2019; Bender et al. 2018). The hyper network contains all possible network structures in the search space, and conducting NAS is to find the best sub-network from the hyper network. Though most NAS methods conduct supervised task in the search phase, recent works showed that structures searched using unsupervised tasks like rotation prediction and colorization have comparable performance as those searched with supervised task (Liu et al. 2020; Yan et al. 2020; Duan et al. 2021).

Different from the previous works that rely on augmented datasets for retraining and full model update, this paper proposes a novel idea of repairing DNNs with a patch subnetwork to correct varying failure cases at runtime, which provides a light-weight cost-efficient method to enhance model robustness without retraining the predeployed networks.

PatchNAS Framework

The proposed PatchNAS framework is demonstrated in Fig. 1. As shown in the top left, a DNN is trained with a pre-collected dataset D_t, and then deployed into the real-world environment. During runtime, the DNN is used for inference on a set of unlabeled data D_u from deployment environment, among which a small set of failure samples

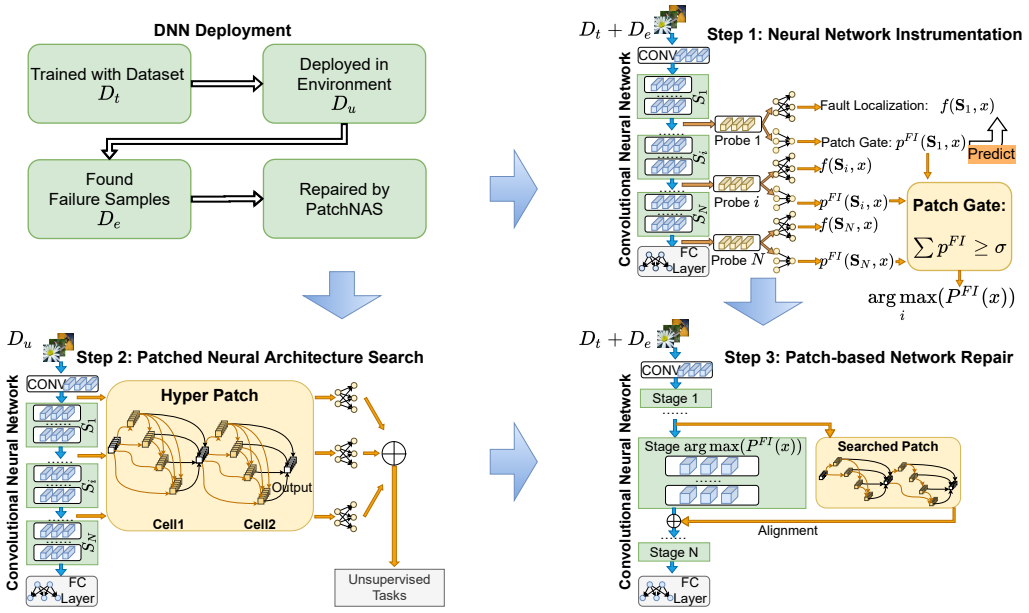

Figure 1: The framework of PatchNAS. Given a few failure samples collected from deployment environment, we repair the DNNs with three steps: (1) perform Neural Network Instrumentation to predict potential failure samples and locate the corresponding faulty stages; (2) search for a patch network structure against the noises in deployment using unsupervised NAS technique; (3) apply the searched patch to correct the output of the faulty stages.

D_e are annotated by the operating engineers for model repair. In PatchNAS, we freeze the pretrained DNN and attach a small patch network architecture to repair the model, i.e., making the network's output distribution p as close as that of the ground truth with the aid of the failure samples.

PatchNAS mainly consists of three steps. Firstly, it performs *neural network instrumentation* (step 1, top right of Fig. 1) to determine the faulty areas by inserting probes into each stage of the network structure. Modern convolutional neural networks are typically composed of several stages where each stage consists of several Conv-BN-ReLU layers or blocks as shown in the green square in Fig. 1. The sizes of output feature maps are usually the same inside the stage while shrinking between adjacent stages. In neural network instrumentation, we intend to locate the stage that most likely causes the incorrect output for failure samples in D_e, and further train a *patch gate* module to determine whether misclassification will occur for an input.

Next, a small *patch network* module is constructed with a *patched neural network architecture search* algorithm (step 2, the bottom left of Fig. 1). Though different failure inputs can be traced down to different network stages, we share the patch network for all weak stages to form a light-weight patch module. We create a hyper network patch and train all its sub architectures on several self-supervised tasks to determine the suitable patch structure.

After obtaining the patch network, *patched-based network repair* (step 3, the bottom right of Fig. 1) can be applied to correct the output of failure samples. Given an input failure sample x, the patch gate will determine the faulty area in the forward propagation of DNN stages and activate the patch network to intervene its own output with the stage's output to repair the erroneous feature map.

Detailed Method

Neural Network Instrumentation

Inspired by program instrumentation (Huang 1978) that inserts probes into the program to monitor the running states in execution, we propose a neural network instrumentation method to locate the most faulty stage inside the network. As demonstrated in the top right of Fig. 1, we insert probes into each stage of the network and acquire the information of useful features extracted by each stage. In DNNs, the probe is in the form of a neural network consisting of a Conv-BN-ReLU layer and a fully-connected (FC) layer. It takes the output feature maps of the stage as input and uses them on classification task to evaluate their contributions to the final results. The process is formulated as:

$$p^{cls}(\mathbf{S}_i, x) = \boldsymbol{softmax}(\mathbf{W}_i^{fc} * \boldsymbol{CBR}(M_i, \mathbf{W}_i^{CBR})), \quad (1)$$

where i is the index of the stage; $\mathbf{S}_i$ represents stage i with output feature maps M_i given input sample x; $p^{cls}(\mathbf{S}_i, x)$ is the output distribution of probe i; $\boldsymbol{CBR}(\cdot)$ represents a Conv-BN-ReLU layer, and $\mathbf{W}_i^{fc}$, $\mathbf{W}_i^{CBR}$ are the weights of the ith probe. The probes are updated with normal samples from the original training dataset D_t with the following loss:

$$\min_{\mathbf{W}_i^{fc}, \mathbf{W}_i^{CBR}} \sum_{x \in D_t} \boldsymbol{CE}(Y, p^{cls}(\mathbf{S}_i, x)), \quad (2)$$

where $\boldsymbol{CE}(\cdot)$ is the standard cross entropy loss.

With the probes, we monitor the network's information extraction process and locate faulty stage of failure samples.

Faulty Stage Localization We define the faulty stage of a network as the most related stage that generates incorrect output for the failure samples. Specifically, we adopt the Fisher Information (Cramér 1999) to estimate the relative importance of each stage on repairing the network. Fisher Information (FI) measures the amount of information that an observable random variable carries about an unknown parameter of a target distribution, (Xu et al. 2021), which is defined as the variance of the score function (the gradient of log-likelihood) with respect to parameter θ:

$$s(\theta) = \nabla_\theta \log p(y|\theta), \tag{3}$$

$$\boldsymbol{F} = \mathbb{E}_{p(y|\theta)}[\nabla \log p(y|\theta) \nabla \log p(y|\theta)^{\mathrm{T}}], \tag{4}$$

where s is the score function, θ is the parameter that models distribution $p(y|\theta)$, and $\boldsymbol{F}$ is the Fisher Information Matrix. In neural networks, Fisher Information measures the importance of the parameters in the network towards the given tasks. Given a network parameter θ, the Fisher Information Matrix is defined as:

$$\boldsymbol{F}(\theta) = \mathbb{E}[(\frac{\partial \log p(y|x;\theta)}{\partial \theta})(\frac{\partial \log p(y|x;\theta)}{\partial \theta})^{\mathrm{T}}]. \tag{5}$$

In practice, Fisher Information of the network parameter is computed through the diagonal elements of the empirical Fisher Information Matrix. For any model parameter $\theta \in \mathbf{S}_i$, its FI in a given batch of the data is computed as:

$$f(\theta, x) = \frac{1}{|B|} \sum_{(\hat{x},\hat{y}) \in B} (\frac{\partial \log p(\hat{y}|\hat{x};\theta;\mathbf{W}_i^{fc};\mathbf{W}_i^{CBR})}{\partial \theta})^2, \tag{6}$$

where $\mathbf{W}_i^{fc}; \mathbf{W}_i^{CBR}$ are the weights of probe i; (x, y) is a pair of samples in the data batch B.

We define a stage's FI to be the average FI of its own parameters. Hence, the process is formulated as:

$$f(\mathbf{S}_i, x) = \frac{1}{|\mathbf{S}_i|} \sum_{\theta \in \mathbf{S}_i} f(\theta, x), \tag{7}$$

where $\mathbf{S}_i$ represents stage i. The higher a stage's FI is, the more importance it is to the network output.

With the above formulation, the stage with the largest FI, $\arg\max_i(f(\mathbf{S}_i, x))$, is considered as the faulty stage for the failure sample x.

Patch Gate Module Intuitively, faulty cases reveal the intrinsic weakness of the neural network. They contain some features that the neural network is vulnerable to. If we can detect such features and locate the faults in forward propagation, we can intervene in the network's propagation to correct the potential failure cases.

Therefore, we propose a patch gate module to detect faults and their locations in the forward propagation of the network. We reuse the Conv-BN-ReLU layer of the probe and replace the original FC layer with a new one $\mathbf{W}_i^{fcG} \in \mathbb{R}^{C_{in} \times 1}$, where C_{in} is the number of input channels determined by the feature map size. For a given input x, the patch gate predicts the normalized FI of a network stage as:

$$p^{FI}(\mathbf{S}_i, x) = \boldsymbol{ReLU}(\mathbf{W}_i^{fcG} * \boldsymbol{CBR}(M_i, \mathbf{W}_i^{CBR}))), \tag{8}$$

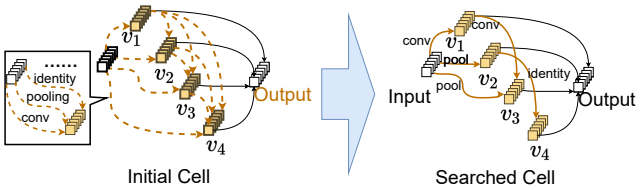

Figure 2: The architecture of a search cell.

where M_i is the output feature maps of stage i given data x.

The patch gate is trained by:

$$\min_{\{\mathbf{W}_i^{fcG}\}_{i \le N}} \frac{1}{N} \sum_{i=1}^{N} (f(\mathbf{S}_i, x) - p^{FI}(\mathbf{S}_i, x))^2, \tag{9}$$

where $f(\mathbf{S}_i, x)$ is the FI of stage i computed through Eq. 7, $p^{FI}(\mathbf{S}_i, x)$ is the prediction of patch gate for FI of stage i, and N is the number of stages in the DNN. In practice, we use equal number of failure samples and normal samples to train the patch gate.

If $\sum_{i \le N} p^{FI}(\mathbf{S}_i, x)$ exceeds a pre-defined threshold σ, we consider that the current forward propagation contains potential fault and the inference of stage $\arg\max_i(p^{FI}(\mathbf{S}_i, x))$ should be intervened.

Patched Neural Architecture Search

As network robustness for certain noise is affected by its architecture (Tang et al. 2021), the pre-defined network structure doesn't always suit for varying deployment environments. We resolve this issue by using a patch network to adjust the output of the faulty stage in the network to enhance its robustness. Since annotating data is infeasible at runtime, we adopt an unsupervised NAS technique to search for the structure of the patch network.

Patch Structure and Search Space As shown in lower left of Fig. 1, we attach a patch network module to all faulty stages detected by neural network instrumentation to correct the output for failure samples. The main body of the patch is a stem containing two cells. These cells are served to extract proper features in noisy environment, whose structures are determined by an unsupervised NAS method.

The initial cell is a fully-connected directed acyclic graph consisting of M nodes $\{v_1, v_2, ...v_M\}$ and $M \cdot (M-1)/2$ directed edges as depicted in Fig. 2. The nodes represent 3-d tensors and edges represent computation operations (e.g., 3×3 convolution) which direct from the input tensors to the output tensors. For nodes v_i, the output is the sum of computation results of its predecessors, which is computed as:

$$v_i = \sum_{j<i} \boldsymbol{norm}(e_{(j,i)}(v_j)), \tag{10}$$

$$\boldsymbol{norm}(x) = \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}}, \tag{11}$$

where the function $\boldsymbol{norm}(\cdot)$ performs normalization to the inputs, and $e_{(j,i)}$ is the edge from v_j to v_i representing a

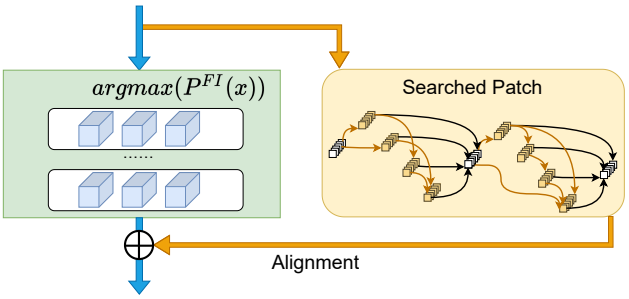

Figure 3: Faulty stage correction with patch network.

compound operation. Each edge contains operations sampled from an operation pool $\mathcal{O}$ (1×1 and 3×3 conv., 1×1 and 3×3 depth-wise separable conv., 3×3 max pooling, 3×3 average pooling, identity), which is calculated as:

$$e_{(j,i)}(x) = \sum_i \boldsymbol{norm}(o_i(x)), \tag{12}$$

where o_i is a sampled operation. The output of the cell is the concatenation of all nodes except for the input. The goal of NAS is to determine the exact operations for each edge.

Architecture Search and Evaluation As labeled data is hard to collect in real deployment environment, we search for the structure of the patch in an unsupervised manner.

Specifically, we construct a set of self-supervised tasks (e.g., predicting image rotation angle, solving jigsaw puzzle) to train the initialized/hyper patch with unlabeled samples. Firstly, we preprocess the raw data according to given task (e.g., random rotated) and feed them into the original DNN. Then we compute the input feature maps of the ith stage of the network and feed them to the patch. The output of the patch network is further fed to the classifiers to solve the given self-supervised tasks. The process is formulated as:

$$U_i(M_i) = \mathbf{W}_i^{un} * \boldsymbol{K}^{\pi(\mathcal{O})}(M_i), \tag{13}$$

where $U_i(M_i)$ is the patch's outputs of the unsupervised task on stage i given inputs M_i, $\mathbf{W}_i^{un}$ is the weight of the classifier on stage i, $\pi(\mathcal{O})$ represents the sampled operations from operation pool $\mathcal{O}$, and $\boldsymbol{K}^{\pi(\mathcal{O})}(\cdot)$ calculates the results of the patch given sampled operations. The objective of training is:

$$\min_{W\in\pi(\mathcal{O}),\{\mathbf{W}_i^{un}\}_{i\le N}} \boldsymbol{CE}(Y, \boldsymbol{Softmax}(\sum_{i=1}^{N} U_i(M_i)), \tag{14}$$

where Y is the corresponding label of the self-supervised task, $W \in \pi(\mathcal{O})$ is the weights in the sampled operations which will be trained by gradient descent.

We train a hyper patch network by randomly sampling sub-patches and update weights of the chosen operations with current data batch. In our implementation, we adopt the same sampling strategy as in (Bender et al. 2018).

After training the hyper patch network, we randomly sample 200 sub-patch structures with the sampling probability $\frac{1}{|\mathcal{O}|}$ and evaluate their performance. The structure of the sub-patch with the best overall performance, denoted by $\mathcal{O}'$, is chosen as the final structure of the patch network.

Patch-Based Network Repair

After acquiring patch gate and the searched patch structure, we incorporate the searched patch to correct the output of the predicted faulty stage of the network.

Repair Method Given an input sample x, the patch gate calculates $p^{FI}(\mathbf{S}_i, x)$ for every stage based on Eq. 8. If the sum of the predicted FI exceeds a pre-defined threshold σ, the patch will intervene the forward of stage $i = \arg\max(p^{FI}(\mathbf{S}_i, x))$. The input of stage i will be fed into the patch and the output of both the patch and stage i will be added up as the input of the next stage $i+1$, i.e.,

$$M'_{i+1} = M_{i+1} + K^{\mathcal{O}'}(M_i), \tag{15}$$

where M_i and M_{i+1} are the input and output feature maps of stage i, M'_{i+1} is the repaired output of stage i as well as the input of stage $i+1$. Notably, the output of the patch and the stage may have different shapes. To solve the problem, we simply add another 1×1 convolution layer after the patch to align the feature maps by its stride and out channels like the projection shortcuts in ResNet (He et al. 2016).

As the structure of the patch has been trained for feature extraction on unsupervised tasks, it could be applied to distinguish the valid information from noisy environment and, in turn, to compensate the forward of the faulty stage.

Patch Training Given a set of training samples, the patch module is trained by minimizing the cross entropy loss between the neural network's final output and the true label through gradient descent. We use both failure samples D_e and the samples in D_t that trigger the patch gate to train the patch for repairment. Since the proportion of samples that trigger the patch gate is quite small (see Evaluation), the training cost of patch network is efficient.

While we update weights in the patch, the original network remains intact during the whole process, which means that it maintains its performance on the normal samples that are not triggered by patch gate in deployment. This characteristic is helpful to repair one big powerful network with a specialized small patch for varying deployment environments (e.g., snowy arctic areas and moist rain-forest areas). Moreover, the deployment cost of the repair is significantly reduced as only small modules need to be transferred to the target environment rather than the full model.

Evaluation

Implementation

Dataset and DNN Models We adopt *CIFAR-10* (Krizhevsky, Hinton et al. 2009) and *Tiny-ImageNet*[1] to evaluate the performance of different repairing methods. Following the works of (Hendrycks and Dietterich 2019; Yu et al. 2021), 15 types of noises [2] in natural environment are added to the datasets' original test images. Each type of

[1]https://tiny-imagenet.herokuapp.com/

[2]Gaussian noise(GN), shot noise(SN), impulse noise(IN), defocus blur(DN), glass blur(GN), motion blur(MB), zoom blur(ZB), snow(SO), frost(FR), fog(FOG), brightness(BR), contrast(CO), elastic transform(ET), pixelate(PL), Jpeg compression(JC)

Model	Methods	Type of Noise															Avg.
		GN	SN	IN	DB	GB	MB	ZB	SO	FR	FOG	BR	CO	ET	PL	JC	
AllConvNet	CutMix	35.98	24.80	38.07	18.41	15.06	18.29	17.36	14.74	14.66	16.48	17.86	24.24	15.55	17.73	16.53	20.38
	AugMix	47.58	41.12	44.88	13.96	29.15	20.04	16.74	19.05	25.59	17.43	19.40	43.32	14.63	21.72	17.48	26.14
	FSGMix	17.95	14.14	25.77	19.73	19.39	17.01	22.47	17.43	19.75	28.36	**21.11**	56.73	19.88	14.74	20.83	22.35
	DeepRepair	41.77	37.17	45.70	19.90	31.42	22.40	22.25	25.92	34.49	32.78	20.18	67.03	**21.03**	20.05	**25.11**	31.15
	PatchNAS	**58.48**	**54.86**	**52.11**	**23.53**	**46.43**	**25.57**	**25.79**	**31.18**	**43.76**	**37.84**	15.37	**67.12**	18.66	**39.60**	20.32	**37.38**
WRN-40-2	CutMix	35.44	25.54	29.26	26.80	20.45	25.48	25.26	26.34	15.05	26.17	30.57	24.15	21.31	23.87	23.67	25.29
	AugMix	26.45	33.54	34.71	23.35	20.93	22.63	23.47	18.55	22.46	21.78	22.50	23.19	19.41	31.17	18.79	24.20
	FSGMix	15.14	16.53	37.72	**31.53**	27.94	29.65	**30.60**	26.01	27.91	31.86	**33.88**	47.05	**25.99**	24.56	24.30	28.71
	DeepRepair	40.09	35.60	**46.73**	26.84	29.61	25.91	27.63	27.87	35.91	32.19	27.52	49.03	25.52	19.41	**29.02**	31.93
	PatchNAS	**56.66**	**50.24**	39.85	23.52	**32.85**	**31.33**	26.65	**29.33**	**39.28**	**32.67**	24.12	**49.12**	21.97	**47.63**	21.70	**35.13**
DenseNet	CutMix	39.16	38.13	35.66	25.81	24.36	19.67	20.53	25.92	19.74	21.68	29.91	24.23	19.61	20.13	19.06	25.57
	AugMix	43.39	39.36	22.80	21.30	28.02	17.27	20.12	19.13	21.80	17.78	21.81	31.96	16.78	39.01	18.83	25.57
	FSGMix	36.55	29.94	36.32	**29.48**	29.14	27.47	**30.99**	26.31	31.84	33.83	**32.17**	49.30	24.56	30.88	26.24	31.67
	DeepRepair	42.30	44.51	**43.44**	25.11	30.30	24.40	26.23	28.21	36.36	31.64	28.53	**53.31**	25.53	15.26	**28.32**	32.23
	PatchNAS	**55.13**	**50.42**	31.70	23.63	**33.16**	**29.18**	26.14	**31.30**	**38.09**	**33.46**	23.72	45.12	24.72	**47.58**	22.20	**34.37**
MobileNet V2	CutMix	**26.27**	**25.51**	**23.66**	23.07	18.04	21.60	25.47	16.63	16.23	13.28	19.05	16.86	20.04	**24.34**	**23.80**	20.92
	AugMix	25.61	23.39	23.49	25.12	**20.05**	23.26	27.42	16.24	17.25	13.99	17.49	17.15	22.28	22.04	23.77	21.24
	FSGMix	15.29	15.31	14.25	23.50	18.06	24.48	26.77	17.84	19.93	17.00	18.15	**23.23**	21.18	21.26	23.19	19.96
	PatchNAS	22.98	23.20	20.54	**29.42**	18.61	**29.00**	**30.51**	**19.53**	**20.48**	**24.39**	**26.35**	17.73	**24.58**	19.36	22.21	**23.26**

Table 1: Comparison of test accuracy on failure samples (i.e., D_s) with single type of noise for AllConvNet, WRN-40-2 and DenseNet for CIFAR-10 and MobileNet-V2 for Tiny-ImageNet.

noises contains five severity levels. We use four well-known DNNs in the experiments: *AllConvNet* (Springenberg et al. 2014), *Wide-ResNet* (Zagoruyko and Komodakis 2016), *DenseNet* (Huang et al. 2017) for CIFAR-10, and *MobileNet-V2* (Sandler et al. 2018) for Tiny-ImageNet.

To simulate runtime environments, we randomly select half samples from each noisy dataset as non-labeled dataset D_u, and use the rest noisy samples to test the DNNs well-trained following (Hendrycks et al. 2020). Based on the results, we partition the failure samples into two sets: a small number of failure samples (100 for CIFAR-10 and 2000 for Tiny-ImageNet) are used as labeled training set D_e, and the rest failure samples are used as test set D_s to evaluate the performance of repair algorithms.

Baselines We compare the proposed PatchNAS against two types of baselines: finetune-based methods (*CutMix* (Yun et al. 2019), *AugMix* (Hendrycks et al. 2020)) and specifically designed network-repair methods (*FSGMix* (Ren et al. 2020), *DeepRepair* (Yu et al. 2021)). For finetune-based methods, failure cases D_e and original training dataset D_t are combined to finetune the neural networks. For network-repair methods, D_e and D_t serve their roles according to specific methods in the literature.

Configuration of PatchNAS The number of nodes M in each cell is set to 4. We adopt three commonly-used unsupervised tasks to search for the architecture of the patch: rotation prediction (Gidaris, Singh, and Komodakis 2018), colorization (Zhang, Isola, and Efros 2016) and solving jigsaw puzzles (Noroozi and Favaro 2016). The structure with best average performance on three tasks is chosen as the architecture of the patch. The experiments are conducted on a server with 2 NVIDIA GeForce RTX 3090.

Performance Comparison

Ability of Repairing a Single Type of Noise We compare the performance of different algorithms on repairing with a single type of noise in the test dataset D_s. Each experiment is repeated three times and the average results are shown in Tab. 1. PatchNAS achieves the best overall results on all DNNs where the performance on failure samples improves up to 82.32% against the second best method. Notably, PatchNAS achieves bigger improvement on noises that create more failure samples like Gaussian noise and glass blur. Some environmental noises like brightness can be effectively resolved by data-augmentation (e.g., FSGMix performs best on BR), and they are less sensitive to DNN structure with a tiny patch. The average performance gain ranges from 6.64% to 20.00% compared with the SOTA method (DeepRepair on CIFAR-10), and a 9.51% performance boost is observed against AugMix on Tiny-ImageNet.

Ability of Repairing Mixture of Noises We then test the algorithms' ability to repair a mixture of different types of noises, and the results are shown in Tab. 2. PatchNAS achieves the best overall results in the experiments where around 25% performance boost can be found against the baselines. It demonstrates that the variety of noises makes it difficult for one single neural network architecture to fit with. Nevertheless, PatchNAS significantly outperforms the baselines in repairing mixture of noises.

Robustness We further use the combined clean and noisy test set to evaluate the robustness of repaired DNNs, average accuracy is compared in Fig. 4. Again, PatchNAS has the highest accuracy on all networks and achieves the biggest improvement on the least robust network (AllConvNet).

Model	Method	Number of Different Noises			
		All 15	4	3	2
AllConvNet	CutMix	28.60	26.54	26.73	26.90
	AugMix	31.05	33.57	37.11	35.84
	FSGMix	26.27	24.09	25.21	18.11
	DeepRepair	30.41	32.92	36.19	32.18
	PatchNAS	**36.88**	**37.06**	**39.34**	**40.46**
WRN-40-2	CutMix	33.73	29.16	28.89	27.45
	AugMix	**34.61**	27.38	26.54	30.00
	FSGMix	34.09	26.24	24.88	20.67
	DeepRepair	29.94	30.61	32.33	30.74
	PatchNAS	30.28	**31.59**	**34.37**	**35.73**
DenseNet	CutMix	34.65	32.21	33.01	34.78
	AugMix	34.01	34.38	35.39	36.90
	FSGMix	38.32	33.44	33.65	31.04
	DeepRepair	36.58	33.48	35.89	34.97
	PatchNAS	**38.42**	**38.01**	**37.91**	**39.30**

Table 2: Comparison of test accuracy on failure samples (i.e., D_s) with multiple noises.

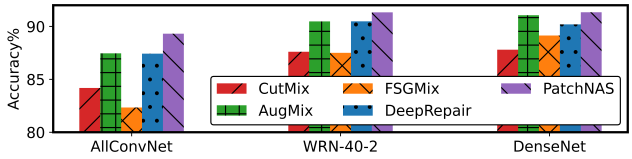

Figure 4: Comparison of accuracy on combined test sets.

Repair and Deployment Overhead

Repair Overhead There are four modules to train in PatchNAS: the probes, the patch gate, the hyper patch, and the searched patch. The training of probes and hyper patch don't require failure samples and they can be prepared in advance. If failure samples are found, it can simply repair the network by training the patch gate and the patch network instantly. We compare the average repair time of different methods and the results are shown in Fig. 5. While many baselines take several hours to retrain the model, the proposed PatchNAS can repair most DNNs within 10 minutes.

The repair efficiency of PatchNAS stems from two aspects: less training samples and smaller model to train. As illustrated in Tab. 3, while other methods adopt the whole original training dataset to finetune the network, PatchNAS only requires a few samples to train the patch network, which is about 5% of the original dataset.

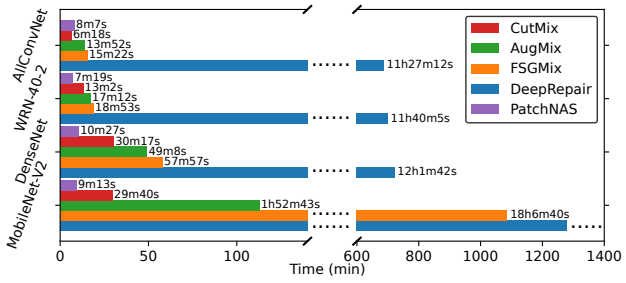

Figure 5: Comparison of repairing time.

Network	Baseline		PatchNAS	
	#Samples	%	#Samples	%
AllConvNet	50100	100%	**2847**	**5.68%**
WRN-40-2	50100	100%	**2675**	**5.34%**
DenseNet	50100	100%	**1682**	**3.36%**
MobileNet-V2	102000	100%	**5495**	**5.39%**

Table 3: Training samples used to repair DNNs.

Model	Number of Parameters			Overhead	
	Original Model	Probes & Patch Gate	Patch	Baseline	PatchNAS
AllConvNet	1.41M	0.09M	0.10M	100%	**13.21%**
WRN-40-2	2.24M	0.06M	0.11M	100%	**7.59%**
DenseNet	0.77M	0.11M	0.07M	100%	**23.18%**
MobileNet-V2	2.48M	0.12M	0.08M	100%	**8.06%**

Table 4: Deployment overhead of different methods.

Deployment Overhead Unlike other repair methods that need to transfer the entire repaired DNN back to deployment environment, PatchNAS only transfers several patch modules. We compare the amount of parameters updated by different methods, and the results are in Tab. 4. The parameter size of PatchNAS is down to 7.59% of those of the baselines, which is light-weight and favorable for fast deployment.

Performance of Patch Gate

We test the performance of patch gate to detect failure samples from the noise datasets. We draw the ROC curves of patch gate and calculate its AUC value for different types of noises. As illustrated in Fig. 6, the patch gate excels in distinguishing failure samples where most AUC values are above 0.9. The worst case is BR noise for AllConvNet where AUC still reaches 0.8, which shows the effectiveness of patch gate.

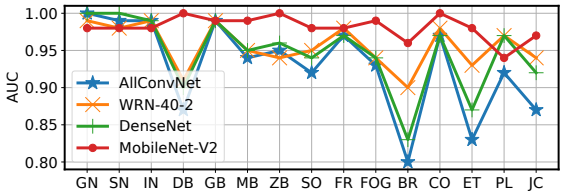

Figure 6: AUC of the patch gate in detecting failure samples.

Conclusion

In this paper, we proposed a patch based method called PatchNAS to repair DNNs in deployment. PatchNAS froze the pretrained DNNs and introduced a small patch network to correct failure samples at runtime. It adopted a network instrumentation method to determine the faulty stage of the network and applied an unsupervised NAS technique to search the optimal patch structure in deployment environment. The patch network repaired the DNNs by correcting the output feature maps of the faulty stage and maintained network performance on normal samples to enhance robustness. Extensive experiments showed that the proposed PatchNAS significant outperformed the state-of-the-arts with much lower deployment overhead.

Acknowledgments

This work was partially supported by the Natural Science Foundation of Jiangsu Province (Project "Research on Frontier Basic Theory and Method of Security Defense for Power Systems with High-dimensional Uncertain Factors", Grant No. BK20222003), the National Natural Science Foundation of China (Grant Nos. 61972196), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Sino-German Institutes of Social Computing.

References

Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; and Le, Q. 2018. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, 550–559. PMLR.

Cramér, H. 1999. *Mathematical methods of statistics*, volume 43. Princeton university press.

Devaguptapu, C.; Agarwal, D.; Mittal, G.; Gopalani, P.; and Balasubramanian, V. N. 2021. On adversarial robustness: A neural architecture search perspective. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 152–161.

Duan, Y.; Chen, X.; Xu, H.; Chen, Z.; Liang, X.; Zhang, T.; and Li, Z. 2021. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5251–5260.

Eykholt, K.; Evtimov, I.; Fernandes, E.; Li, B.; Rahmati, A.; Xiao, C.; Prakash, A.; Kohno, T.; and Song, D. 2018. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1625–1634.

Gao, X.; Saha, R. K.; Prasad, M. R.; and Roychoudhury, A. 2020. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *IEEE/ACM 42nd International Conference on Software Engineering*, 1147–1158. IEEE.

Gidaris, S.; Singh, P.; and Komodakis, N. 2018. Unsupervised representation learning by predicting image rotations. In *Proceedings of the InInternational Conference on Learning Representations*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hendrycks, D.; and Dietterich, T. 2019. Benchmarking neural network robustness to common corruptions and perturbations. In *Proceedings of the InInternational Conference on Learning Representations*.

Hendrycks, D.; Mu, N.; Cubuk, E. D.; Zoph, B.; Gilmer, J.; and Lakshminarayanan, B. 2020. Augmix: A simple data processing method to improve robustness and uncertainty. In *Proceedings of the InInternational Conference on Learning Representations*.

Hendrycks, D.; Zhao, K.; Basart, S.; Steinhardt, J.; and Song, D. 2021. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 15262–15271.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Huang, J. 1978. Program instrumentation and software testing. *Computer*, 11(4): 25–32.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. Technical Report 4, University of Toronto.

Kurakin, A.; Goodfellow, I. J.; and Bengio, S. 2018. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, 99–112. Chapman and Hall/CRC.

Liu, C.; Dollár, P.; He, K.; Girshick, R.; Yuille, A.; and Xie, S. 2020. Are labels necessary for neural architecture search? In *European Conference on Computer Vision*, 798–813. Springer.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. Darts: Differentiable architecture search. In *Proceedings of the InInternational Conference on Learning Representations*.

Ma, S.; Liu, Y.; Lee, W.-C.; Zhang, X.; and Grama, A. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 175–186.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Noroozi, M.; and Favaro, P. 2016. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, 69–84. Springer.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, 4780–4789.

Ren, X.; Yu, B.; Qi, H.; Juefei-Xu, F.; Li, Z.; Xue, W.; Ma, L.; and Zhao, J. 2020. Few-shot guided mix for dnn repairing. In *IEEE International Conference on Software Maintenance and Evolution*, 717–721. IEEE.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (ICCV'18)*, 4510–4520.

Sohn, J.; Kang, S.; and Yoo, S. 2019. Search based repair of deep neural networks. *arXiv preprint arXiv:1912.12463*.

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Tang, S.; Gong, R.; Wang, Y.; Liu, A.; Wang, J.; Chen, X.; Yu, F.; Liu, X.; Song, D.; Yuille, A.; et al. 2021. Robustart:

Benchmarking robustness on architecture design and training techniques. *arXiv preprint arXiv:2109.05211*.

Tian, Y.; Pei, K.; Jana, S.; and Ray, B. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, 303–314.

Xu, R.; Luo, F.; Zhang, Z.; Tan, C.; Chang, B.; Huang, S.; and Huang, F. 2021. Raise a Child in Large Language Model: Towards Effective and Generalizable Fine-tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Yan, S.; Zheng, Y.; Ao, W.; Zeng, X.; and Zhang, M. 2020. Does unsupervised architecture representation learning help neural architecture search? In *Advances in Neural Information Processing Systems*, volume 33, 12486–12498.

Yu, B.; Qi, H.; Guo, Q.; Juefei-Xu, F.; Xie, X.; Ma, L.; and Zhao, J. 2021. DeepRepair: Style-Guided Repairing for Deep Neural Networks in the Real-World Operational Environment. *IEEE Transactions on Reliability*, 1–16.

Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; and Yoo, Y. 2019. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, 6023–6032.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. In Richard C. Wilson, E. R. H.; and Smith, W. A. P., eds., *Proceedings of the British Machine Vision Conference*, 87.1–87.12. BMVA Press. ISBN 1-901725-59-6.

Zhang, H.; and Chan, W. 2019. Apricot: A weight-adaptation approach to fixing deep learning models. In *34th IEEE/ACM International Conference on Automated Software Engineering*, 376–387. IEEE.

Zhang, R.; Isola, P.; and Efros, A. A. 2016. Colorful image colorization. In *European conference on computer vision*, 649–666. Springer.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710.