Architectural Innovations for Efficient Denoising and Classification

A Manual vs. Neural Architecture Search Comparison

MSc Computer Science - AI track

Thomas Markhorst

Delft University of Technology

TU Delft

BOSCH

Architectural Innovations for Efficient Denoising and Classification

A Manual vs. Neural Architecture Search Comparison

by

Thomas Markhorst

Supervisors:	Dr. O.S. Kayhan Dr. J.C. Van Gemert
Graduation committee:	Dr. J.C. Van Gemert (Associate Professor, TU Delft) Dr. E. Demirović (Assistant Professor, TU Delft) Dr. O.S. Kayhan (Bosch Security System, TU Delft)
Project Duration:	December, 2022 - August, 2023
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft BOSCH

Preface

This thesis describes my final work to obtain the title Master of Science at The Delft University of Technology. My thesis was conducted within the Computer Vision Lab at TU Delft in collaboration with Bosch Security Systems B.V.

I would like to thank my supervisor Osman Kayhan, for ensuring this project was a wonderful learning experience and for always bringing a smile to my face. I value that combination more than I can express. I also enjoyed working with Rick Huizer and Dajt Mullaj, who transformed this individual thesis into a team experience. Of course, my thesis would not be possible without my professor Jan van Gemert and the Advanced Development team at Bosch Security Systems B.V. I would like to thank them for the opportunity and guidance. Finally, I would like to express my gratitude to my friends and even more so to my father, mother, and brother, without whom I would not be where I am right now.

Thomas Markhorst
Delft, August 2023

Contents

1

Overview

Understanding the content of an image can become complicated when noise is present. Noise often occurs in dark environments but can also arise inside a camera system. In this thesis, we aim to improve the human perception of noisy images. Part of this improvement can be achieved by using Convolutional Neural Networks (CNNs) as denoisers [10, 33], which try to remove the noise from noisy images. However, denoisers are not able to fully recover a noise-free image. Thus, for images with a high level of noise, denoising might not improve human perception enough. We propose to use a combination of image denoising and classification to improve human image understanding. Image classification is used, as it is a form of machine perception which can assist a human in interpreting the denoised image.

We take a security camera during the night as an example. The security guard is trying to determine if there is a human or animal in the images. Due to the lack of light, the image is noisy, which hinders image understanding. Removing noise from the image assists the security guard, but classifying the content as animal or human improves understanding even further.

We target edge devices, such as the security camera, which have limited computational power. So the CNN, which performs denoising and classification, should require little resources. We, therefore, design a model to be efficient, defined as (i) having low inference time, further referred to as latency, while (ii) retaining denoising performance and classification accuracy. To achieve this, we propose and study two methods to efficiently join denoising and classification in one model.

Using the joint model for different applications requires altering it for different requirements and devices. However, manually adjusting the model for each device and desired latency can take significant effort and requires expert knowledge. These issues have been recently addressed using Neural Architecture Search (NAS) [18, 27, 28, 31], where the design of models is automated to some extent. An additional benefit is the ability of NAS to design a model for specific latency requirements while taking the computational power of the target device into account. We combine NAS with the joint architectures, proposing a seamless and efficient approach to designing joint denoising and classification models for diverse use cases.

This thesis consists of two parts. In Chapter 2, we present our work in the form of a scientific paper, targeting deep learning and computer vision experts. Additionally, Chapter 3 provides background information for the specialized topics in our study. We advise non-experts to first get acquainted with unfamiliar topics by reading Chapter 3. Section 3.1, explains the convolutional operator and its efficient replacement MBConv. In Section 3.2, we discuss noise, noise metrics, training loss, and our denoising baseline UNet. Image classification is discussed in 3.3. Finally, Section 3.4, explains different NAS approaches and then dives into differentiable NAS and the basis of our NAS method TF-NAS [16].

2

Scientific paper

The scientific article starts on the next page due to its fixed full-page CVPR format.

Architectural Innovations for Efficient Denoising and Classification: A Manual vs. Neural Architecture Search Comparison

Thomas Markhorst[1,2] Osman Semih Kayhan[1,2] Jan C.van Gemert[2]
[1] Bosch Security Systems [2] Delft University of Technology

Abstract

In this paper, we combine image denoising and classification, aiming to enhance human perception of noisy images captured by edge devices, like security cameras. Since edge devices have little computational power, we also optimize for efficiency by proposing a novel architecture that integrates the two tasks. Additionally, we alter a Neural Architecture Search (NAS) method, which searches for classifiers [16], to search for the integrated model while optimizing for a target latency, classification accuracy, and denoising performance. Our NAS architectures outperform our manually designed alternatives in both denoising and classification, offering a significant improvement to human perception. Moreover, our approach empowers users to construct architectures tailored to domains like medical imaging, surveillance systems, and industrial inspections.

1. Introduction

The intersection of edge devices, such as security cameras, and deep learning has sparked an interest in optimizing neural networks for inference time, further referred to as latency. Common tasks to optimize for such efficiency are object classification and detection, which mainly aid in machine perception. However, when aiming to improve human perception, the quality of the processed image is equally significant. This importance intensifies particularly for images containing noise, which can arise from various sources such as low-light conditions, sensor noise, or any stage within the image processing pipeline. We focus on using an efficient model to enhance the human perception of noisy images.

Domains relying on human image perception but challenged by noisy images, like medical imaging [21], surveillance systems [29], and industrial inspections [8], can benefit from recently proposed denoising Convolutional Neural Networks (CNNs) [12, 41]. As CNNs denoise better than traditional methods [5, 11]. Fast CNN denoisers [10, 42] are required to accommodate the real-time requirement of the affected domains. However, denoisers are not able to remove all noise, which is not always enough for human image perception.

(a) (b) (c) (d)

Figure 1. We take a noisy image (a), which can be interpreted as an animal (b) or human (c). DC-NAS S is used to denoise and classify (a), aiming to improve human perception (d). Note, in a real application (b) and (c) would not be available, which increases the difficulty of interpreting the noisy image. Artist: Astkhik Rakimova

We further improve human understanding of the image by combining denoising with machine perception, like image classification. For instance, while improving the quality of a noisy image would assist a security guard, the ability to classify the scene content could help interpret the denoised image. This classification could distinguish between a human and a large animal. Therefore, we investigate models which can leverage the benefits of both denoising and classification to enhance human understanding in real-time.

A model combining both denoising and classification is studied in [17], focusing on denoising performance. In addition, we optimize for efficiency, which is required for edge devices, and classification. Our efficiency definition is based on two elements: (i) latency reduction while (ii) retaining denoising performance and classification accuracy. These elements could be optimized using independent classification and denoising models. However, we propose an architecture combining the tasks more efficiently.

First, we employ established model design approaches to enhance independent denoising and classification models, such as model scaling [20, 31] and the introduction of efficient operators [27]. Although the models are optimized, they still operate separately, resulting in unnecessary over-

head. Hence we propose and compare two methods that join both tasks, yielding a novel and efficient architecture.

Adjusting this architecture for each device and desired latency can be laborious and requires expert knowledge. These issues have recently garnered interest, leading to the emergence of new automated architecture search techniques, which have achieved competitive results in image classification [32, 35]. Moreover, recent Neural Architecture Search (NAS) approaches incorporate latency in their loss function, enabling the design of architectures tailored to specific latency requirements. Combining NAS with the proposed architecture provides a seamless and efficient approach to designing denoising and classification models for diverse use cases.

We find that our proposed efficiency-focused architecture consistently outperforms our more straightforward alternative. This is observed for both the manually and NAS designed models. In addition, our NAS models significantly outperform the manually designed ones in denoising and classification performance.

We have the following contributions. (i) We introduce a novel architecture to combine denoising and classification efficiently. The novelty lies in sharing an encoder between the denoiser and the classifier. (ii) We propose modifications to an existing NAS method for classification [16] to stabilize its search, which improves the performance of the found architectures. (iii) We extend an existing NAS method to search for a model which combines denoising and classification, optimized with respect to a target latency, classification accuracy, and denoising performance.

Since no prior work proposes a joint efficient model for denoising and classification, we study the tasks both separately and joint in Section 3. The findings are used as expert knowledge to construct the NAS method in Section 4.

2. Related work

Denoising. Image denoising aims to reconstruct a clean image x from its observed noisy variant y. This relation can be formulated as $y = x + n$, where we assume n to be additive white Gaussian noise (AWGN). Neural network-based denoisers offer faster inference and good performance compared to traditional denoising methods like BM3D [5] and WNNM [11]. The interest in deep learning for denoising started with DnCNN [41], a simple Convolutional Neural Network (CNN). Encoder-decoder architectures became popular due to their efficient hierarchical feature extraction. Specifically, UNet [26] whose skip-connections between the en- and decoder enhance the denoising process as shown in follow-up methods [12, 19, 24]. The interest in the UNet structure continued with transformer architectures [9, 34]. In this paper, our denoisers are based on UNet, ensuring our findings can translate to most related work.

Efficient classification. Optimization for efficiency is generally achieved by either compressing pre-trained networks [23] or designing small networks directly [27, 32]. We focus on efficient design, for which handcrafted models and neural architecture search (NAS) have played essential roles. Studies proposing handcrafted models often introduce efficient operators [14, 27, 43] or scaling methods [31]. These efficient operators are used in NAS methods [32, 35] aiming for the automated design of efficient neural networks. Such an operator is the inverted residual with a linear bottleneck (MBConv) introduced in MobileNetV2 [27]. In our models, we study scaling methods and MBConv's efficiency characteristic.

Neural Architecture Search. The use of reinforcement learning (RL) for neural architecture search introduced efficient architectures with competitive classification performance [13, 25, 30, 32]. However, their discrete search space is computationally expensive. Differentiable NAS (DNAS) methods [1, 18, 35] significantly reduce this cost by relaxing the search space to be continuous using learnable vectors α for selecting candidate operations, which allows for gradient-based optimization. The popularity of DNAS started with DARTS [18], which searches a cell structure. Due to the complex design and repetitiveness throughout the network of the cell structure, follow-up works [16, 35] search operators for every layer instead of constructing repeating cells.

Pitfalls of DNAS are the collapse of search into some fixed operations and a performance drop when converting from the continuous search network to the discretized inference network [4, 38, 39]. TF-NAS [16] addresses these issues with an adaptation in the search algorithm, which lets the search model mimic the discrete behavior of the inference model. In addition, TF-NAS searches an architecture with a target latency by adding a latency loss to the search optimization. Because of these properties, we use TF-NAS as a baseline for our NAS study.

Existing NAS methods for denoising are either not reproducible [22], have a cell-based search space [40], or do not have an encoder-decoder [3] architecture. Instead, we use a layer-based search space and encoder-decoder structure.

Joint classification and denoising. In [36], the positive influence of denoising methods on classification performance is discussed. Moreover, [17] proposed a joint model where a VGG classifier [28] is attached to a denoiser similar to UNet. They report qualitative improvement of the denoised images when adding the classification loss to the denoiser's optimization. Although the model denoises well, the proposed model is not optimized for classification and efficiency, nor is another joining method studied.

3. Gaining expert knowledge on DC-Net

For the separate classification and denoising tasks, we construct a baseline model. Additionally, methods to in-

crease their respective efficiency are studied, resulting in a reduced version of the baseline models. The different sizes of the classifier and denoiser are used to study joining methods and their efficiency.

Dataset & settings. We generate a controlled synthetic data set to study the behavior of the classifier and denoiser when applying model scaling, replacing the convolutional operations, and combining both models. The dataset consists of 30k images, each with a random monotone background in a gray tint [0.1 - 0.3] with two randomly placed non-overlapping MNIST [7] digits. We use two digits to increase the complexity of the denoising task. For experiments including classification, the two digits are extracted from the image using ground truth locations. These extracted digits are separately used as input for the classifier. In the experiments where noise is required, for either denoising or noisy classification, synthetic Gaussian noise is added. This noise is zero mean, and the intensity of the noise is controlled using the standard deviation (σ) of the distribution. Figure 4a shows a sample, and Figure 4b its noisy variant. To test the model behavior on an extensive noise range, every model is trained and tested on eleven σ values evenly spaced on the interval $[0, 1]$.

The models are trained using Adam optimizer with 1E-3 learning rate (LR), plateau LR scheduler, and 100 epochs.

Since the experiments with the controlled data set are not targeted at a specific device, the metric defining efficiency should not depend on a device. Such a metric is computational power, most commonly defined as Floating Point Operations (FLOPs), which we use as the primary metric. Despite being device dependent, we assess latency as a secondary metric. The latency is measured with a batch size of 32, 100 warm-up inference passes and averaged over 1000 inference passes. Denoising performance is quantified using the Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) metrics [33]. For both metrics, higher is better.

3.1. Efficient Classification

Experimental setup. Our baseline classifier (Conv-L) consists of two convolutional, one global max pooling, and a linear layer. Each convolutional layer also has a group normalization [37], max pooling, and ReLU activation function.

To construct the reduced version, we use two methods similar to previous works [27, 31]. In the first method, we replace the second convolutional layer with an MBConv layer. Three expansion rates are used $\{1, 2.5, 4\}$: (i) rate 1 is the lowest possible value, (ii) rate 4 matches the number of FLOPs of the baseline, and (iii) rate 2.5 is in the middle of those two. The second reduction method is to lower the number of filters in the baseline, also called the model width. Using these techniques, models with three different FLOP sizes are constructed, {S, M, L}. We use the following naming scheme, Conv-x and MBe-x, where x represents the FLOP size and e is the expansion rate of the MBConv.

Model	Size	Exp. rate	FLOPs (K) ↓	Lat. (ms) ↓	Acc (%) ↑
Conv-L	L	-	447	0.336	63.2
MB1-S	S	1	177	0.300	56.2
MB2.5-M	M	2.5	350	0.384	64.1
MB4-L	L	4	424	0.468	64.9

Table 1. Classification models using convolutions (Conv) and MBConvs (MB), designed for three different FLOP targets: {S, M, L}, to compare scaling methods. MB models scale down more efficiently than normal Conv models.

The models are trained using Cross Entropy loss. We report the accuracy averaged over all 11 noise levels.

Exp. 1: Conv vs MBConv comparison. According to [27], the MBConv layer should be more efficient than a normal convolutional layer. Therefore, when comparing the two operators in our network, we expect the version with an MBConv layer to need fewer FLOPs for the same accuracy. In Table 1, the MB models with expansion rates 2.5 (MB2.5-M) and 4 (MB4-L) classify better than the Conv-L model with fewer FLOPs. However, with an expansion rate of 1 (MB1-S), the accuracy drops 7% compared to Conv-L. Therefore, [27]'s theory also holds for the noisy classifier, but only for the higher expansion rates.

Conclusion. The MBConv layer can replace the convolutional layers. Additional experiments in Appendix A study the scaling method. We find that compared to the Conv models, the MB models also scale down more efficiently by first reducing the expansion rate, possibly followed by a width reduction. Scaling effectively reduces the number of FLOPs.

MB2.5-M has the second-best accuracy with low FLOPs and a latency close to the baseline, Conv-L. Therefore, MB2.5-M is used as the reduced classifier. We also use MB2.5-M as the new baseline classifier as it outperforms the old baseline, Conv-L, in FLOPs and accuracy.

It is important to note that the reduction in FLOPs instantiated by using MBConvs, does not translate to a latency reduction in these experiments. This issue is discussed previously in [32]. Since the target of these experiments is FLOPs and the latency increase is manageable, we place minimal emphasis on the latency.

3.2. Efficient Denoising

Experimental setup. For denoising, the baseline and reduced version are constructed by performing a hyperparameter study on UNet similar to [20]. Figure 2 shows the UNet architecture along with its hyperparameters to tune. We explore the parameters one at a time, starting with the

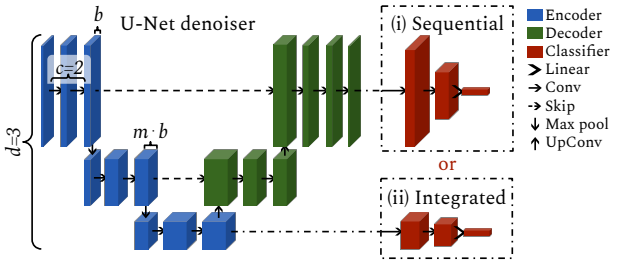


Figure 2. Schematic of the UNet, with hyperparameters base feature map width (b), depth (d), channel multiplier (m) and convolutions per layer (c). For the joint model either attaching the classifier (i) Sequential or (ii) Integrated.

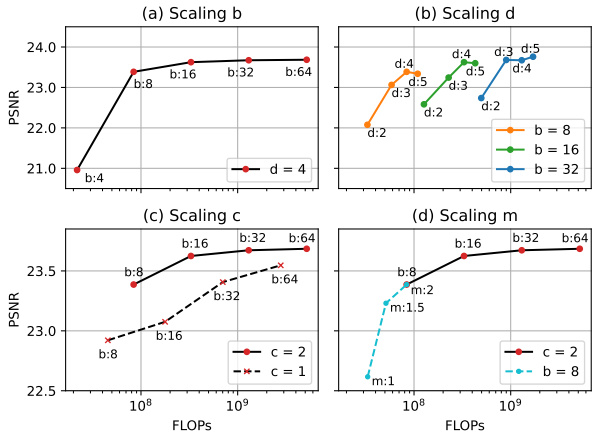


Figure 3. UNet hyperparameter (Figure 2) scaling experiments. Shows how altering a specific hyper-parameter influences denoising performance and FLOPs. We only show PSNR results of σ=0.8, as the other results show the same trend. We find that b and m scale down efficiently, d and c do not.

number of base features maps b, then the UNet depth d, the feature map multiplier m, and the number of convolutional blocks per layer c. In the original UNet: $\{b = 64, d = 5, m = 2, c = 2\}$. Altering these hyperparameters can greatly reduce the model size. Similar to the classification experiments, we also study the ability of the MBConv operator to increase efficiency in the denoiser. The models are trained using Charbonnier loss [2].

Exp. 1: The base feature map width *b*. In this experiment, we aim to find the relevant range of b. Since b is multiplied at every level, the number of feature maps throughout all layers depends on it, which makes it a powerful hyper-parameter. We use $d = 4$ and the other hyperparameters as in the original UNet, then we test $b \in \{4, 8, 16, 32, 64\}$. The trend in Figure 3.a shows that the performance and FLOPs increase with b. We observe that the trend is significantly disrupted by $b = 4$. Conversely, the performance difference between $b = 32$ and $b = 64$ is small,

Model	FLOPs (M) ↓	Lat. (ms) ↓	Metric	Noise level (σ) 0.2	0.4	0.8	1
UNet Baseline	1301.8	7.10	PSNR ↑	33.9	29.5	23.8	22.3
			SSIM ↑	0.99	0.98	0.95	0.92
UNet Reduced	51.2	2.38	PSNR ↑	33.2	28.7	23.3	22.0
			SSIM ↑	0.99	0.98	0.94	0.92

Table 2. Compares Baseline and Reduced UNet denoisers. The reduced model has significantly lower FLOPs and latency yet similar denoising performance.

but the network size quadrupled. Therefore in further experiments, we focus on $b \in \{8, 16, 32\}$.

Exp. 2: The UNet depth *d*. Given the robustness of b, we are interested in how reducing d compares in terms of efficiency. Figure 3.b displays the performance of the architectures with the selected $b \in \{8, 16, 32\}$ testing $d \in \{2, 3, 4, 5\}$. We observe that reducing d causes a drop in denoising performance, whereas b retains performance better, also in Figure 3.a. Therefore b scales down more efficiently. The models with $d = 3$ or 4 denoise most efficient. Especially for the smaller models, $d = 4$ performs well.

Exp. 3: The number of conv blocks per layer *c*. Does reducing c further increase efficiency? To test this, we take the best-performing settings, $d = 4$ and $b \in \{8, 16, 32, 64\}$, and compare $c = 1$ and $c = 2$. Figure 3.c shows that the model with $c = 2$ outperforms $c = 1$. Therefore reducing c does not benefit the model's efficiency.

Exp. 4: The feature map multiplier *m*. We test if our smallest model could be further reduced in size by lowering m. We take $d = 4$ and $b = 8$, and compare $m \in \{1, 1.5, 2\}$. Figure 3.d shows that the reduction to $m = 1.5$ retains performance. For $m = 1$, the performance drops. Reducing m to 1.5 could therefore be used to scale down the model when further reducing b significantly decreases performance.

Conclusion. To construct the reduced and baseline denoiser, we use the smallest and largest value from the found hyperparameter ranges. Resulting in baseline: $\{b = 32, d = 4, m = 2, c = 2\}$ and reduced: $\{b = 8, d = 4, m = 1.5, c = 2\}$. Table 2 compares the two models for a selection of the noise levels. Although the reduced model has significantly fewer FLOPs and lower latency, the denoising performance is relatively similar to the baseline denoiser.

The UNet hyper-parameter experiments are replicated using MBConvs, which lead to similar findings. Moreover, the Conv UNet slightly outperforms the MB model. Therefore, the Conv model is used.

3.3. Joint model: DC-Net

Experimental setup. We construct a baseline and reduced joint model, Denoising-Classifying Network (DC-Net). The baseline uses MB2.5-M as classifier and the baseline UNet as denoiser. Reduced DC-Net also uses MB2.5-M as classifier but the reduced UNet as denoiser.

DC-Net	Type	FLOPs (M) ↓	Lat. (ms) ↓	PSNR ↑	SSIM ↑	Acc. (%) ↑
Baseline	**Int.**	1301.8	**7.14**	**32.8**	**0.97**	88.1
	Seq.	1302.1	7.55	27.1	0.95	**89.6**
Reduced	**Int.**	51.2	**2.41**	**29.9**	**0.97**	86.2
	Seq.	51.5	2.83	25.2	0.92	**87.6**

Table 3. Compares the reduced and baseline joint models, both the Integrated and Sequential method trained on the synthetic noise dataset. Integrated performs significantly better in denoising and slightly worse in classification. Integrated also scales down better.

(a) GT (b) σ=0.8 (c) **Int. S** (d) Seq. S (e) **Int. L** (f) Seq. L

Figure 4. Ground-truth sample (a), which is the target for the denoiser when given noisy image (b). With S for reduced and L for baseline. (c-f) are the cropped denoised outputs for input (b). The red square is a zoomed-in region. For higher noise levels, the denoising performance of Sequential is inferior to Integrated.

For joining the denoiser and classifier, we propose two models: (i) a Sequential model where the classifier is attached after the denoiser (Figure 2.i), and (ii) an Integrated model where the classifier is attached to the UNet encoder (Figure 2.ii). For Integrated, classification and denoising share the encoder, after which two branches are used, the decoder for denoising and another branch for classification.

The benefits of the Integrated model could come in threefold. First, using a shared encoder removes the need for a second large classifier, like in the Sequential method. Second, the decoder and classifier branches could run in parallel compared to running sequentially, which can result in lower latency. Thirdly, the decoder is only optimized for denoising as the optimization of the classifier does not influence it anymore. This should result in better image quality.

The models are trained using a weighted combination of the Cross-Entropy and Charbonnier loss [0.1 - 0.9], respectively weighted by 0.1 and 0.9. We report the metrics averaged over all 11 noise levels.

Exp. 1 Integrated vs. Sequential. Which joining method performs better for the baseline, and does the same hold when reducing its size? We compare the Sequential and Integrated models. In Table 3, we see that for both the baseline and reduced DC-Net models, the Integrated version performs significantly better at denoising, while the Sequential version performs better at classification. For higher noise levels, the Sequential models are not able to reconstruct the digit in detail, as visible in Figure 4.

Conclusion The integrated model has a slightly lower classification accuracy compared to the Sequential model. However, when aiming for improved human perception, it is still required for the human to see the content of the image. Therefore the Integrated model is more suitable.

4. Neural Architecture Search

We follow similar experimentation strategies as in the previous section. TF-NAS is used to construct a classifier, which we use as a basis for our denoiser and joint model.

Dataset & settings. NAS experiments are conducted on Imagenet [6], randomly cropped to 224x224 pixels. To reduce search and training time, 100 classes (Imagenet 100) from the original 1000 classes were chosen, as in [16]. In the experiments requiring noise, Gaussian noise is sampled uniformly with a continuous range of σ in $[0, 1]$.

The models are searched using SGD with momentum, 2E-2 LR with 90 epochs. After that, the found architecture is trained from scratch with 2E-1 LR for 250 epochs. All other settings are similar to [16]. The loss function form depends on the task of the experiment, smooth Cross-Entropy for classification ($\mathcal{L}_{\text{CE}}$), combined Charbonnier-SSIM for denoising ($\mathcal{L}_{\text{Char}}$,$\mathcal{L}_{\text{SSIM}}$), and a weighted combination for the joint model ($\mathcal{L}_{\text{Both}}$).

$$\mathcal{L}_{\text{Both}} = 0.1 \cdot \mathcal{L}_{\text{CE}} + 0.9 \cdot (0.8 \cdot \mathcal{L}_{\text{Char}} + 0.2 \cdot \mathcal{L}_{\text{SSIM}}) \quad (1)$$

Since our NAS method uses a latency look-up table constructed for our device, these experiments target a specific device, GeForce RTX 3090 GPU. Therefore latency is suitable for defining efficiency in the NAS experiments.

4.1. Classification: C-NAS

Experimental Setup. Since TF-NAS [16] learns β's to control the number of convolutional operators per stage, β's can reduce the model size. However, in the models proposed by [16], only 2 out of 24 stages are reduced by β. So the β's have little effect on the found architectures, yet they make the search space more complex. Therefore we propose a version of TF-NAS where the β's are removed so that all convolutional blocks are used.

The candidate operations in the search space of TF-NAS are MBConvs with 8 different configurations, see App. B. The configurations differ in kernel size, expansion rate, and in- or excluding a squeeze- and excitation layer (SE) [15].

The classification experiments are performed using data without noise, as the aim is to examine the NAS method, which is designed for clean images. We investigate key components of TF-NAS and try to improve its stability and classification performance.

Exp 1. Learned vs. Removed β. We conduct an experiment to study the effect of removing β on the search quality. The SE-layer is excluded from the candidate blocks, halving the search space to ensure the number of candidate operations does not cause search instability. We set a low target latency of 6 ms, as learning β should have a positive effect on small networks. For both the learned and removed settings, we run two searches, search 1 and 2.

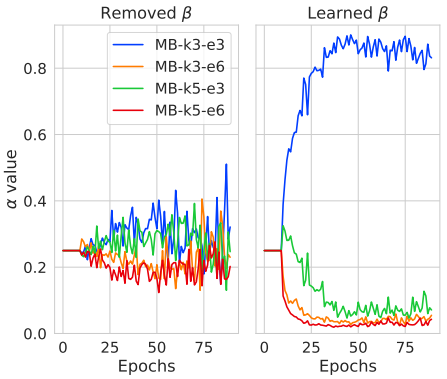


Figure 5. Stage-5:block-4's α values for Removed and Learned β. Search is more stable for Removed β.

Figure 6. Acc for different search spaces, showed for different target latencies. Using fewer operations is most robust.

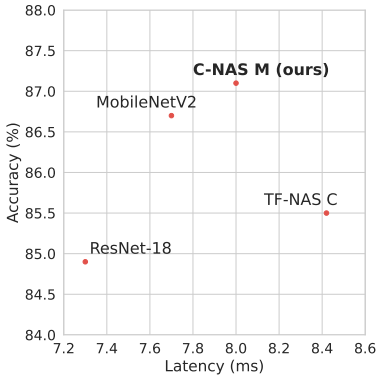


Figure 7. Comparing classifiers with similar latency. Our model classifies best, competing with MBNetV2

Figure 5 shows that when β is learned, the α's oscillate and therefore do not decide on an architecture. Whereas with Removed β, the search is stable. This stability reflects in the performance, as the average accuracy of the Removed β models is 86.3%, compared to 84.2% for Learned β. The separate results for each model are shown in Appendix C.

Exp 2. Number of operators in search space. Does reducing the number of operators during search positively influence the performance of the found models? We test this by comparing the performance of architectures searched with three different search space sizes, {4, 6, or 8} operations, defined in App. B. For each of these search spaces, three different latency targets are used: {6, 8, and 12} ms.

In Figure 6, we see that for lower target latencies, 6 and 8 ms, using fewer operations in the search space does not alter performance significantly. When targeting 12 ms latency, reducing the number of operations in the search space does show a significant improvement. Additionally, we find that when using the larger search spaces, the operators from the small search space are still preferred for lower latencies.

Exp 3. Compare with original TF-NAS. How do architectures found using our proposed changes to TF-NAS perform compared to models with similar latency? We compare our model, C-NAS M, with TF-NAS C, MobileNetV2, and ResNet-18. MobileNetV2 and our model have similar latency, architecture, and operator types. ResNet only differs in that it uses the Conv operator.

Figure 7 shows that the model found using our method (Figure 8.i) has lower latency yet higher accuracy than TF-NAS C as proposed in [16]. The model was searched with target latency 8.0. Therefore, it reached its target. Although ResNet-18 and MobileNetV2 run faster than our model, our classification accuracy is superior, especially when compared to ResNet-18, which only uses Convs.

Conclusion. By Removing β and reducing the number of operators used in the search, the search stability increases, and we find architectures that have better accuracy.

Model	UNet params: d	b	m	Lat. (ms) ↓	PSNR ↑	SSIM ↑
Reduced UNet	4	8	1.5	9.2	25.0	0.69
D-NAS S	-	-	-	9.45	**26.0**	**0.72**
UNet	5	64	2	116.5	**26.6**	**0.74**
D-NAS S upscaled	-	-	-	**109.8**	26.5	0.73

Table 4. Comparison of D-NAS and UNet variants for denoising. The rows are split into two latency ranges. D-NAS outperforms Reduced UNet. The larger variants perform similarly.

An architecture found using our changes classifies better than a TF-NAS architecture with similar latency.

The comparison between our model and ResNet-18 shows that our search space is able to compete with widely accepted Conv-based classifiers. Moreover, our model performs on par with MobileNetV2, a manually designed classifier using MBConvs.

4.2. Denoising: D-NAS

Experimental setup. To construct a denoiser, D-NAS (Figure 8.ii), we use the first six stages of a found C-NAS classifier, which has four levels of resolution. Afterwards, we attach a UNet style decoder by using both a transposed convolution and two normal convolutions for each decoder level. Like UNet, we also add skip connections between the encoder and decoder layers. The decoder is not searched.

Exp 1. D-NAS vs UNet denoiser. Does our denoiser D-NAS perform similarly to the UNet denoisers? Reduced UNet (Section 3.2) is used, $\{d = 4, b = 8, c = 2, m = 1.5\}$, with a latency of 9.2 ms. We compare with D-NAS S, which is found by searching with a target latency of 9.2 ms. In addition, we test the standard UNet architecture. To compare with standard UNet, we increase the number of channels in D-NAS S to match the latency of both architectures.

Table 4 shows that D-NAS S outperforms Reduced UNet by 1.0 dB PSNR and 3% SSIM. The two large models perform similarly.

#	Model	Search Images	Search Loss	Lat. (ms) ↓	PSNR ↑	SSIM ↑	Acc. (%) ↑
1	DC-NAS$_{\text{seq}}$ L	Clean	$\mathcal{L}_{\text{Cls}}$	18.3	25.0	0.69	76.0
	DC-NAS L	Clean	$\mathcal{L}_{\text{Cls}}$	**17.9**	**25.5**	**0.70**	76.0
2	DC-NAS$_{\text{clean}}$ M	Clean	$\mathcal{L}_{\text{Cls}}$	13.9	25.4	0.70	75.7
	DC-NAS$_{\text{noisy}}$ M	Noisy	$\mathcal{L}_{\text{Cls}}$	13.7	25.4	0.70	**76.0**
	DC-NAS$^{\mathcal{L}_{\text{Both}}}_{\text{noisy}}$ M	Noisy	$\mathcal{L}_{\text{Both}}$	13.8	25.4	0.70	75.5
3	C-NAS$_{\text{noisy}}$ M	Noisy	$\mathcal{L}_{\text{Cls}}$	7.9	-	-	75.5
4	DC-NAS$^{\text{mb}}$	Clean	$\mathcal{L}_{\text{Cls}}$	27.7	25.8	0.71	75.5
	DC-NAS$^{\text{mb}}_{\text{1-op}}$	Clean	$\mathcal{L}_{\text{Cls}}$	16.4	25.3	0.70	75.4
	DC-NAS$^{\text{mb}}_{\text{3-lay}}$	Clean	$\mathcal{L}_{\text{Cls}}$	22.1	25.4	0.70	75.1
5	DC-Net$_{\text{reduced}}$	-	-	10.0	24.5	0.68	61.9
	DC-Net$^{\text{tail}}_{\text{reduced}}$	-	-	10.5	24.6	0.69	68.6
	DC-NAS S	Noisy	$\mathcal{L}_{\text{Cls}}$	10.3	**25.4**	**0.70**	**74.3**

Table 5. DC-NAS experiments. 1: Sequential vs Integrated model. Similar latency and denoising performance, yet Integrated denoises better. 2: Different search strategies for DC-NAS. Searching on noisy images with only $\mathcal{L}_{\text{Cls}}$ performs best. 3: C-NAS$_{\text{noisy}}$ M vs DC-NAS$_{\text{noisy}}$ M. The Integrated model classifies better. 4: Using MBConvs in the DC-NAS$_{\text{noisy}}$ M decoder, and two downscaled versions. Original DC-NAS$_{\text{noisy}}$ is more efficient. 5: DC-NAS vs DC-Net. The NAS model outperforms the manually designed ones.

Conclusion. D-NAS outperforms our denoising baseline, UNet, for small models. Therefore D-NAS is a suitable denoising architecture, especially when kept small.

4.3. Joint Model: DC-NAS

Experimental setup. To construct the joint model, we use the Integrated and Sequential setup. The Integrated model, DC-NAS, is constructed similarly to D-NAS. As seen in Figure 8.iii, we connect the decoder after the first six stages of C-NAS but still use the remaining stages as classification branch. Whereas the Sequential model, DC-NAS$_{\text{seq}}$, is constructed by attaching C-NAS to the output of D-NAS, see Figure 8.iv. Therefore, the searched classifier is used twice in DC-NAS$_{\text{seq}}$, which increases its latency. To counter this, a smaller C-NAS model is used in both the encoder and classifier.

Exp 1. Compare Integrated vs. Sequential. We compare DC-NAS and DC-NAS$_{\text{seq}}$ models with similar latency. To get the same latency, C-NAS in DC-NAS$_{\text{seq}}$ L has low latency, only 6.7 ms. Whereas in DC-NAS L, C-NAS has a latency of 12 ms.

In Table 5:1, we see that both models have similar latency and the same classification accuracy, however, DC-NAS L improves denoising performance with 0.5 db PSNR and 1% SSIM.

Exp 2. Encoder search. C-NAS contains the searchable operations within DC-NAS. We test several search approaches: (i) using clean images, and (ii) using noisy images. For both approaches, we search the classifier using only classification loss. Resulting in models (i) DC-NAS$_{\text{clean}}$ M and (ii) DC-NAS$_{\text{noisy}}$ M. In addition, for approach (ii), we construct DC-NAS$^{\mathcal{L}_{\text{Both}}}_{\text{noisy}}$ M by searching using the combined denoising and classification loss. Therefore optimizing C-NAS for both tasks within DC-NAS$^{\mathcal{L}_{\text{Both}}}_{\text{noisy}}$. Regardless of the search method, the found models are trained using noisy images and the combined loss.

Table 5:2, shows that using noisy images during search improves classification accuracy, as DC-NAS$_{\text{clean}}$ M improves 0.3% acc compared to DC-NAS$_{\text{noisy}}$ M. Surprisingly, the denoising performance is the same. Using both the denoising and classification objective during the search, as in DC-NAS$^{\mathcal{L}_{\text{Cls}}}_{\text{noisy}}$, reduces the classification accuracy.

Exp 3. C-NAS vs DC-NAS. We are interested in the difference in noisy classification performance of C-NAS and DC-NAS. We remove the decoder from DC-NAS$_{\text{noisy}}$ to obtain C-NAS$_{\text{noisy}}$ and train it for classification.

We see that DC-NAS$_{\text{noisy}}$, improves classification accuracy by 0.5% compared to C-NAS$_{\text{noisy}}$, in Table 5:2-3.

Exp 4. Decoder tuning. All models found in Experiment 2, have similar denoising performance. These models have the same latency but differ only in the operators that are used in the encoder. We test if the denoising performance is influenced by adjusting the operators in the decoder while retaining the latency. DC-NAS$_{\text{clean}}$ M is used as a basis. We construct three alternatives. (i) DC-NAS$^{\text{mb}}$, for which the convolutional operators in the decoder are replaced with MBConvs (MB-k3-e3), which significantly increases the latency of the model. To account for this, we also test with (ii) DC-NAS$^{\text{mb}}_{\text{1-op}}$, where 1 instead of 2 MBConvs are used per layer and (iii) DC-NAS$^{\text{mb}}_{\text{3-lay}}$ where we reduce the number of decoder layers by one. Method (ii) and (iii) relate to scaling c and d in Section 3.2, where scaling the number of channels (b), was found more effective. Scaling b is not considered here as this would require altering the encoder too.

In Table 5:4, we see that DC-NAS$^{\text{mb}}$ compared to DC-NAS$_{\text{clean}}$ M improves the denoising performance. However, at the cost of 13.8 ms latency increase, only caused by the MB decoder. When reducing the complexity of the MB decoder with DC-NAS$^{\text{mb}}_{\text{1-op}}$ and DC-NAS$^{\text{mb}}_{\text{3-lay}}$, the denoising performance reduces to the original level again, but the latency is still higher than for DC-NAS$_{\text{clean}}$ M.

Exp 5. DC-Net vs DC-NAS How does our model, constructed by search, compare to the model we manually constructed? DC-NAS S is searched on noisy data with a target latency set to the actual latency of DC-Net Reduced, from Section 3.3. Since the classification branch of DC-Net is designed for the simpler synthetic dataset, we also test with DC-Net$^{\text{tail}}$, which uses the same classification branch as DC-NAS S.

Table 5:5, shows that DC-NAS S outperforms both DC-Net models by at least 0.8 dB PSNR, 1% SSIM, and 5.7% accuracy. We display the denoising results of DC-NAS S

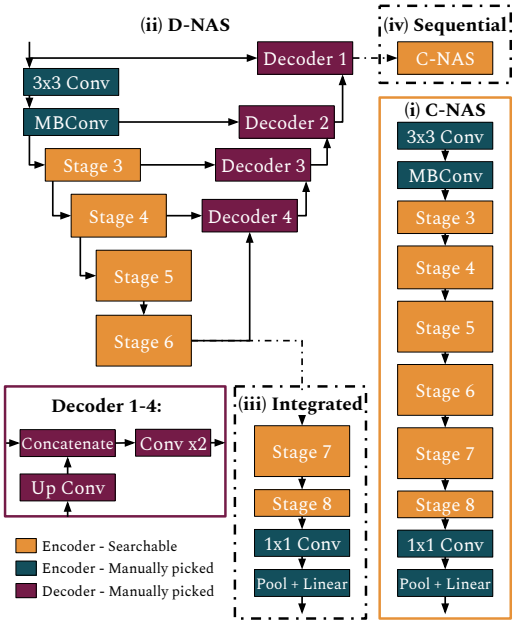


Figure 8. C-NAS and D-NAS architecture. Attach (iii) for DC-NAS and (iv) for DC-NAS$_{\text{seq}}$. Searchable stages consist of a maximum of four MBConvs.

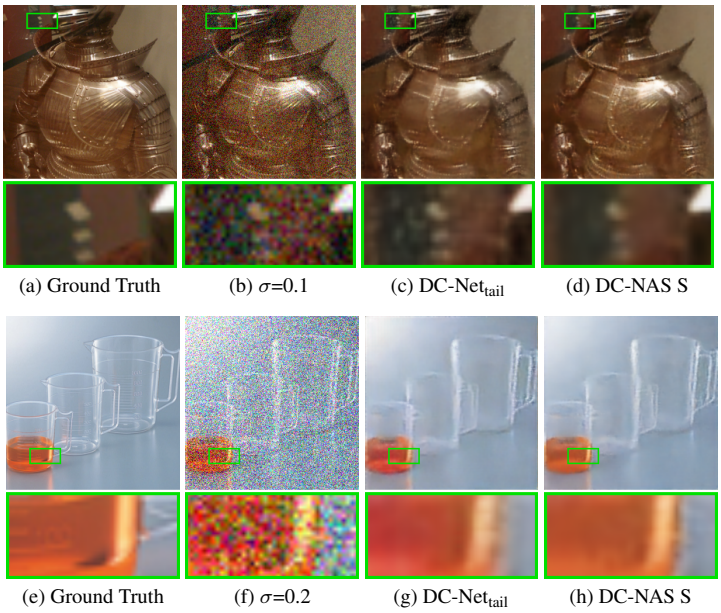


Figure 9. Denoising performance of DC-Net$_{\text{tail}}$ and DC-NAS S. Left to right: clean image, noisy variant, and the denoiser output. Comparing (c) and (d), we see better performance on a smooth plane and slightly sharper edges for (c). With (g) and (h), we see observe a better color reconstruction for (h).

and DC-Net$^{\text{tail}}$ in Figure 9. We observe better denoising on smooth planes, sharper edges and better color reconstruction for DC-NAS S.

Conclusion. We have seen that the Integrated combining method outperforms its Sequential counterpart in denoising. To construct the integrated model, we find that searching for a classifier on noisy data, without taking the denoising objective into account results in the best classification performance. Surprisingly, the search method does not influence the denoising performance. Furthermore, replacing the convolutional blocks in the decoder with MBConvs and then scaling them down does not benefit denoising efficiency either. However, the NAS denoising experiments demonstrate that our denoising setup is competitive and gains performance when scaled significantly. Since up-scaling is not of interest for our models and tuning the decoder operators does not improve performance, our method is focused on searching for only the encoder of the integrated model. The models found by this approach, outperform our manually designed models with similar latency.

5. Limitations & Conclusion

One limitation of our NAS method is its inability to alter the decoder. It is designed this way as manually altering the decoder does not improve efficiency. However, when targeting a significantly different latency, a change in denoising architecture could be optimal. Applying model scaling to the found models is of interest, similar to the EfficientNets [31, 32].

Another limitation is the fixation of β in our NAS method. Although this improved the stability of search and network performance, learning β while retaining a stable search would be preferred. As this would introduce more possibilities in the search space for optimizing efficiency.

In addition, the Integrated models already outperform the Sequential alternatives, but latency could be optimized further for Integrated models by running the denoising and classification branches in parallel.

To conclude, we show that using efficient operators and scaling methods proposed in previous work are relevant for denoising and noisy classification. In addition, we present the integrated model to join the two tasks efficiently. We analyze how the performance of the Integrated model compares to the Sequential model. This shows that the Integrated design is more suitable across various latencies. To simplify the design process of the joint model when targeting a latency, we present a NAS method. We alter an existing NAS method to improve the stability and performance of the search. This method searches a classifier. Using the searched classifier as a basis, we build the Integrated model. We demonstrate that the Integrated model built using our NAS method outperforms the manually constructed model.

References

[1] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *CoRR*, abs/1812.00332, 2018. 2

[2] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud. Two deterministic half-quadratic regularization algorithms for computed imaging. In *Proceedings of 1st International Conference on Image Processing*, volume 2, pages 168–172 vol.2, 1994. 4

[3] Anda Cheng, Jiaxing Wang, Xi Sheryl Zhang, Qiang Chen, Peisong Wang, and Jian Cheng. DPNAS: neural architecture search for deep learning with differential privacy. *CoRR*, abs/2110.08557, 2021. 2

[4] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019. 2

[5] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007. 1, 2

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[7] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 3

[8] Bappaditya Dey, Sandip Halder, Kasem Khalil, Gian Lorusso, Joren Severi, Philippe Leray, and Magdy A. Bayoumi. SEM image denoising with unsupervised machine learning for better defect inspection and metrology. In Ofer Adan and John C. Robinson, editors, *Metrology, Inspection, and Process Control for Semiconductor Manufacturing XXXV*, volume 11611, page 1161115. International Society for Optics and Photonics, SPIE, 2021. 1

[9] Chi-Mao Fan, Tsung-Jung Liu, and Kuan-Hsien Liu. SUNet: Swin transformer UNet for image denoising. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, may 2022. 2

[10] Shuhang Gu, Yawei Li, Luc Van Gool, and Radu Timofte. Self-guided network for fast image denoising. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 1

[11] Shuhang Gu, Lei Zhang, Wangmeng Zuo, and Xiangchu Feng. Weighted nuclear norm minimization with application to image denoising. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2862–2869, 2014. 1, 2

[12] Javier Gurrola-Ramos, Oscar Dalmau, and Teresa E. Alarcón. A residual dense u-net neural network for image denoising. *IEEE Access*, 9:31742–31754, 2021. 1, 2

[13] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. 2

[14] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 2

[15] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. 5

[16] Yibo Hu, Xiang Wu, and Ran He. TF-NAS: rethinking three search freedoms of latency-constrained differentiable neural architecture search. *CoRR*, abs/2008.05314, 2020. 1, 2, 5, 6, 11

[17] Ding Liu, Bihan Wen, Jianbo Jiao, Xianming Liu, Zhangyang Wang, and Thomas S. Huang. Connecting image denoising and high-level vision tasks via deep learning. *CoRR*, abs/1809.01826, 2018. 1, 2

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. 2

[19] Pengju Liu, Hongzhi Zhang, Kai Zhang, Liang Lin, and Wangmeng Zuo. Multi-level wavelet-cnn for image restoration. *CoRR*, abs/1805.07071, 2018. 2

[20] Damian J. Matuszewski and Ida-Maria Sintorn. Reducing the u-net size for practical scenarios: Virus recognition in electron microscopy images. *Computer Methods and Programs in Biomedicine*, 178:31–39, 2019. 1, 3

[21] Sameera V. Mohd Sagheer and Sudhish N. George. A review on medical image denoising algorithms. *Biomedical Signal Processing and Control*, 61:102036, 2020. 1

[22] Marcin Możejko, Tomasz Latkowski, Łukasz Treszczotko, Michał Szafraniuk, and Krzysztof Trojanowski. Superkernel neural architecture search for image denoising, 2020. 2

[23] James O'Neill. An overview of neural network compression. *CoRR*, abs/2006.03669, 2020. 2

[24] Bumjun Park, Songhyun Yu, and Jechang Jeong. Densely connected hierarchical network for image denoising. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2104–2113, 2019. 2

[25] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. 2

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. 2

[27] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 1, 2, 3

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 2

[29] Prabhishek Singh and Achyut Shankar. A novel optical image denoising technique using convolutional neural network and anisotropic diffusion for real-time surveillance applications. *Journal of Real-Time Image Processing*, 18(5):1711–1728, Oct 2021. 1

[30] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. 2

[31] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. 1, 2, 3, 8

[32] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. *CoRR*, abs/2104.00298, 2021. 2, 3, 8, 11

[33] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 3

[34] Zhendong Wang, Xiaodong Cun, Jianmin Bao, and Jianzhuang Liu. Uformer: A general u-shaped transformer for image restoration. *CoRR*, abs/2106.03106, 2021. 2

[35] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *CoRR*, abs/1812.03443, 2018. 2, 11

[36] Jiqing Wu, Radu Timofte, Zhiwu Huang, and Luc Van Gool. On the relation between color image denoising and classification. *arXiv preprint arXiv:1704.01372*, 2017. 2

[37] Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018. 3

[38] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. *CoRR*, abs/1812.09926, 2018. 2

[39] Peng Ye, Baopu Li, Yikang Li, Tao Chen, Jiayuan Fan, and Wanli Ouyang. β-darts: Beta-decay regularization for differentiable architecture search, 2022. 2

[40] Haokui Zhang, Ying Li, Hao Chen, and Chunhua Shen. IR-NAS: neural architecture search for image restoration. *CoRR*, abs/1909.08228, 2019. 2

[41] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising. *CoRR*, abs/1608.03981, 2016. 1, 2

[42] Kai Zhang, Wangmeng Zuo, and Lei Zhang. FFDNet: Toward a fast and flexible solution for CNN-based image denoising. *IEEE Transactions on Image Processing*, 27(9):4608–4622, sep 2018. 1

[43] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 2

A. Efficient Classification: Additional results

In Section 3.1, we compare the performance of MBConv operations with normal convolutions. We present the results for scaling the MBConv and Conv models.

Exp. 2: MBConv width & expansion rate scaling. Since MBConv layers can be used to improve efficiency, we question how to further reduce the MB model's FLOP size. We compare two options: (i) reducing the expansion rate and (ii) scaling the width of the network. We take MB4-L as starting model, as this is our best and largest model.

From the MB models with size S in Table 6, MB1-S performs the worst. It only has a reduced expansion rate from 4 to 1. MB4-S, which is obtained by scaling the width of MB-L, increases classification performance by only 0.4%. However, when slightly reducing MB4-L's width and expansion rate, we derive MB2.5-S, which reaches 58.4% accuracy, significantly outperforming both other S-sized MB models. So the combination of the two methods is most effective.

Exp. 3: Conv width scaling. In this experiment, we compare the width scaling of the Conv-L model. Table 6 shows that all S-sized MB models outperform Conv-S, MB2.5-S even by 3.0%. MB2.5-M also outperforms Conv-M, by 2.7%. Therefore, scaling is more efficient for the MB models than the Conv models when optimizing for FLOPs.

Exp.	Model	Size	Exp. rate	FLOPs (K) ↓	Lat. (ms) ↓	Acc (%) ↑
1-3	Conv-L	L	-	447	0.336	63.2
	MB1-S	S	1	177	0.300	56.2
	MB2.5-M	M	2.5	350	0.384	64.1
	MB4-L	L	4	424	0.468	64.9
2-3	MB2.5-S	S	2.5	178	0.390	58.4
	MB4-S	S	4	188	0.403	56.6
3	Conv-S	S	-	163	0.281	55.4
	Conv-M	M	-	345	0.317	61.4

Table 6. Classification baseline and reduced models, designed for three different FLOP targets: {S, M, L}, to compare scaling methods: expansion rate and model width. Each section of rows is used by the experiments from Sec. 3.1 defined in the *Exp.* column. MB models scale down more efficiently than normal Conv models.

B. Search space

In Section 4.1, different variations of the TF-NAS search space are used [16]. Table 7 displays the candidate operations and for which search space size they are used. The search space with 4 operators is constructed using the MB-Convs without SE-layer, as this is most common in recent NAS methods [32, 35]. For the 6-operator search space, we add the possibility of using an SE layer on the operators where the kernel size is three and the expansion rate is three or six. We use the two smallest operators as they can be used for smaller target latencies too. The search space with 8 operators simply uses all combinations.

Name	Kernel	Expansion rate	SE-layer	4	6	8
MB-k3-e3	3	3	-	✓	✓	✓
MB-k3-e6	3	6	-	✓	✓	✓
MB-k5-e3	5	3	-	✓	✓	✓
MB-k5-e6	5	6	-	✓	✓	✓
MB-k3-e3-se	3	3	✓	-	✓	✓
MB-k3-e6-se	3	6	✓	-	✓	✓
MB-k5-e3-se	5	3	✓	-	-	✓
MB-k5-e6-se	5	6	✓	-	-	✓

Table 7. Overview of the candidate blocks for the different search space sizes $\{4, 6, 8\}$. MBConv operators are used with different kernel sizes k, expansion rate e and in- or excluding the squeeze- and excitation-layer.

C. Learned vs. Removed β: Additional results

In Experiment 1 of Section 4.1, we test the influence of removing β from the search approach. The models with Removed β significantly outperform the models with Learned β in accuracy. Besides, the found models are more similar for Removed than Fixed, Removed β differs only 0.04ms and 0.2% accuracy, while Learned β differs 0.57ms and 1.4% accuracy. This indicates that the search for Removed is more stable.

Type	Search id	LAT (ms) ↓	Acc (%) ↑
Removed β	1	5.85	**86.2**
	2	5.81	**86.4**
Learned β	1	5.04	84.9
	2	4.47	83.5

Table 8. Compares four searched models with target latency 6 ms. Trained on clean images. Two models are searched without β and the other two using learned β. Removed outperforms learned β.

3

Background

3.1. Convolutional & MBConv operator

To understand the MBConv operator, knowledge of the convolutional operator and the depth-wise separable convolution is required. This section will take you through this required knowledge and then explain the MBConv operator.

3.1.1. Convolution

In the field of deep learning, convolutions are mainly used in convolutional layers. Specifically, convolutional layers are used to find patterns in data presented in a grid. Images are a typical example of such a grid, as images are a series of pixels ordered in a grid. The value of the pixel denotes the brightness of that pixel. These grids, the in- and output of the convolutional operators, are also called feature maps.

The convolutional layer uses a matrix called the filter to extract the features from the input images. The filter matrix has dimensions n x n and is used in the dot product with a part of the image with the same dimensions, see Figure 3.1. The spatial dimension of the filter, 3 x 3 in our case, is smaller than the image. To perform the convolutional operation on the entire image, the filter slides over the full image. For every location of the filter on the input, the dot product is added to the result. So the second dot product, created by the green window, is added to the green pixel in the result.

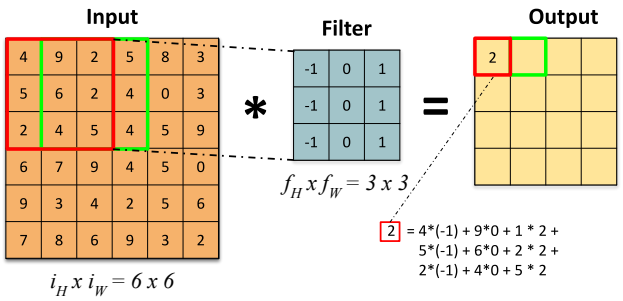


Figure 3.1: Visualization of a convolutional operation with channel dimension = 1.

In the previous example, our image had only one channel, like a grayscale image. For an RGB image, the input would have 3 channels, the filter should match this number of channels. In Figure 3.2, the process is visualized with filter dimension 3 x 3 x 3. To calculate one output pixel, we still take the dot product between the filter and the red highlighted part of the input. Taking the dot product with more channels is similar to taking the dot product between the matching channels (e.g. red input - red filter) and adding the result. In Figure 3.2, the '4' in the output is, therefore, a weighted combination of all values in the red cube. Consequently, a normal convolution fuses spatial and channel information.

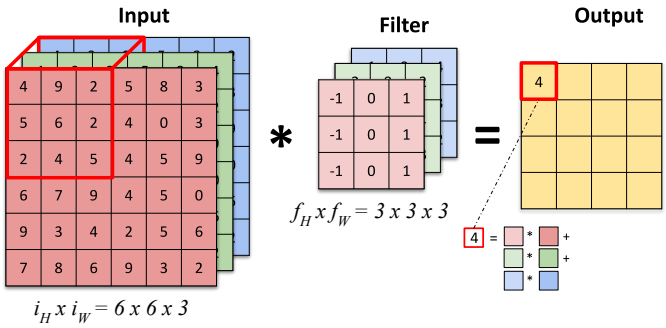


Figure 3.2: Visualization of a convolutional operation with channel dimension = 3.

If the convolutional layer uses multiple filters, the output of the layer will have the same amount of channels. In Figure 3.3, we see that for each filter, the described process is used to create a 4 x 4 matrix. These 2 matrices are stacked and form the output of the layer, with dimension 4 x 4 x 2. Therefore, the number of filters corresponds to the number of channels in the output. At the same time, the number of channels in the filters corresponds to the number of input channels.

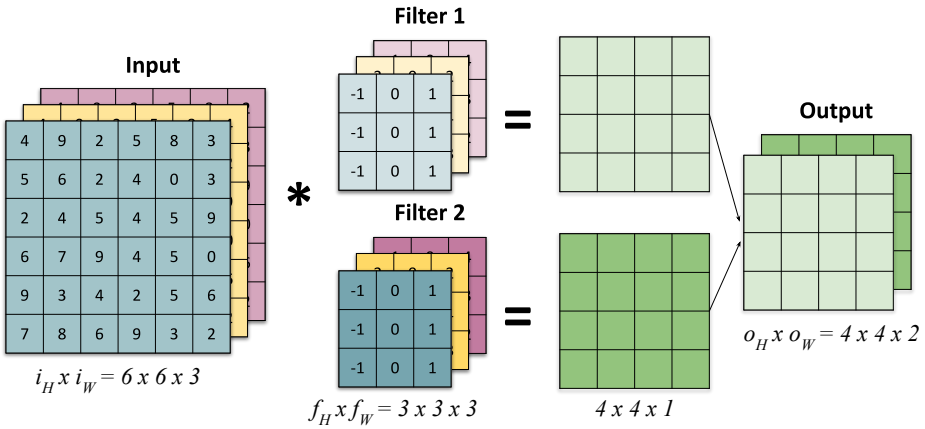


Figure 3.3: Visualization of a convolutional operation with 2 filters and channel dimension = 3.

3.1.2. Depth-wise convolution

In the normal convolution operator, each value in the result fuses input values from different channels and spatial locations. Take Figure 3.2 as an example, the '2' in the result is a combination of all the values in the red box. On the contrary, the depth-wise convolution keeps the channels separate. It does this by taking the dot product of the matching channels (e.g. input green - filter green) and putting it in the corresponding output channel but not adding it together as before. See Figure 3.4. Because the channels are kept separate in the depth-wise convolution, the operation needs significantly less computational power than the normal convolution. The depth-wise convolution only performs spatial information fusion.

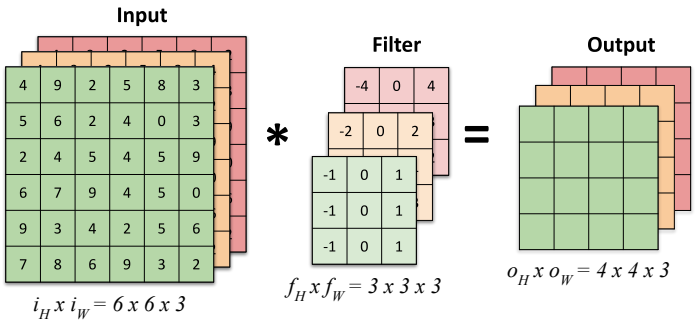


Figure 3.4: Depth-wise convolution visualization, the convolution only combines the values in the same channel.

3.1.3. Point-wise convolution

The point-wise convolution does the opposite of the depth-wise convolution. Instead of keeping the input dimensions separate, it keeps the spatial pixels separate and fuses the channels. This is done by performing a normal convolution with a filter dimension of 1 x 1 x c_i, where c_i is the number of input channels. Because the filters of the point-wise convolution are small, it is a computationally cheap method to increase or decrease the number of channels of the input. The point-wise convolution only performs channel information fusion.

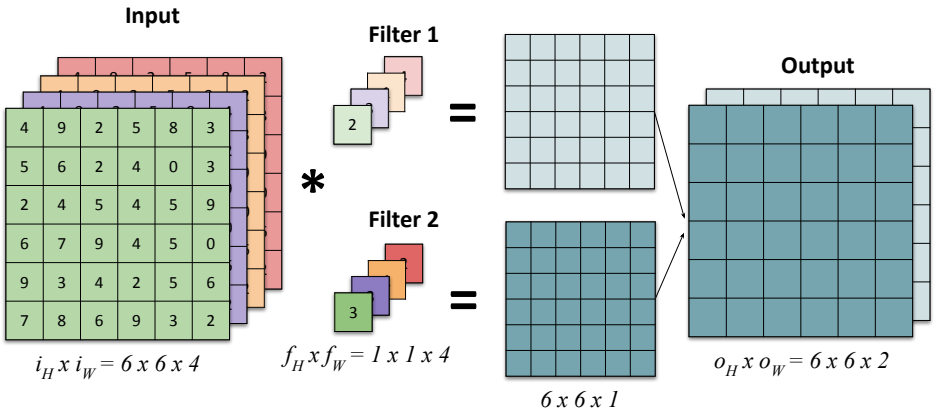


Figure 3.5: Point-wise convolution visualization, the convolution only combines the values with the same spatial location.

3.1.4. MBConv

The MBConv operator is an efficient replacement for a standard convolution and is proposed in MobileNetV2 [23]. Efficient operators aim to reduce the required computations while retaining performance compared to the operator they replace. An MBConv operator is a combination of two point-wise convolutions and one depth-wise convolution. We define the number of input channels c_i, output channels c_o, and expansion rate e. As shown in Figure 3.6, the first point-wise convolution expands the input channels c_i to $c_i \cdot e$. This expansion enlarges the feature space of the feature map and combines the features across channels. These two factors increase the expressiveness of the following depth-wise convolution, as discussed in [13]. The second point-wise convolution is used to reduce and combine the channels again. This reduction forces the information from the larger feature map into a smaller representation, which is more efficient than keeping the larger version. A skip-connection is added between the in- and output feature map, similar to [12].

We emphasize that an MBConv only uses depth- and point-wise convolutions, which separates the fusion of spatial and channel information. This should be inferior to a normal convolution that fuses both spatial and channel information. However, the MBConv is designed under the hypothesis that spatial and channel fusion can be decoupled [5, 14]. This allows the replacement of the convolution with the computationally less expensive depth- and point-wise convolutions.

Adding a non-linear function after a convolutional layer is standard practice to introduce non-linearity in neural networks. In MBConv, the ReLU function [20] is used to this end. However, MBConv does not use a ReLU function in the last point-wise convolution as [23] shows and discusses how adding a non-linear function after the channel reduction destroys information.

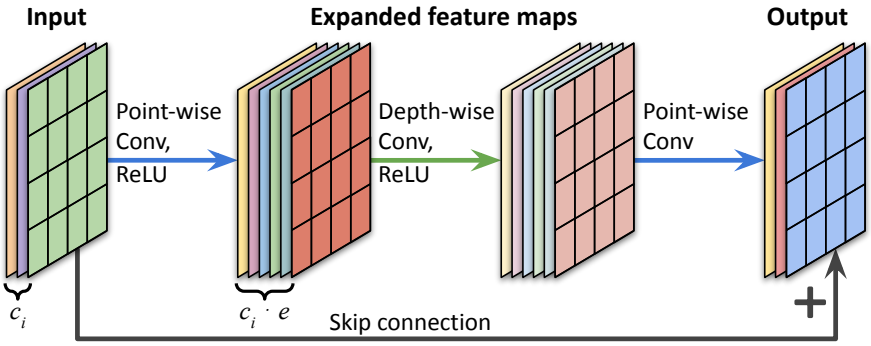


Figure 3.6: MBConv visualization, the convolution combines point- and depth-wise convolutions. With input channels c_i and expansion rate e.

3.2. Image Denoising

3.2.1. Noise

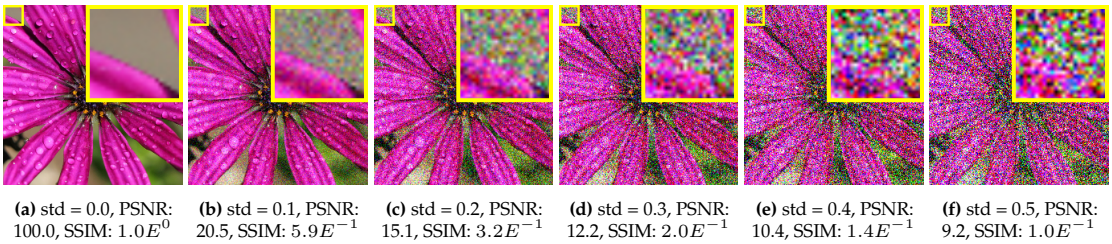
Noise in an image refers to the presence of elements that are non-existent in the actual scene, which can be observed as variations in pixel brightness, often random. Since these variations are not natural, they result in the loss of correct image information. Noisy images are, therefore, harder to interpret for both humans and machines.

Sources of noise in digital images are abundant, and examples are signal transmission errors, photon shot noise in low-light environments, and sensor imperfections. Noise models are used to imitate these real noise sources. Combining a clean image with the noise generated by a noise model results in an image resembling an actual noisy image. This ability to create a noisy version of a clean image is helpful for learning-based denoising approaches since those approaches require a large amount of noisy and clean image pairs. These image pairs are used to train and test the denoising methods, where the noisy image is the input, and the clean image is used as the training target. Since there are different sources of image noise, there are also different methods of modeling them. We discuss the three most commonly studied noise models.

Starting with **signal-dependent noise** [17], for which some knowledge of the workings of a camera is required. Cameras measure light intensity per pixel using the photons traveling from the world to the camera sensor. Each photon that hits a pixel in the camera sensor is converted into a charge, which is measurable. Therefore, the intensity of a pixel is proportional to the average number of photons hitting that pixel in a certain amount of time. This average has an uncertainty stemming from random fluctuations in the arrival time of the photons. The Poisson distribution theoretically describes this type of noise. It is called signal-dependent because the intensity of the pixel is related to the amount of noise that is sampled. This noise model is mainly used to mimic noise in low-light environments.

The other noise model we explain is the one modeling signal-independent noise [17], specifically **read noise**. This noise models the errors that occur while measuring the accumulated charge created by the photons. So, this noise is independent of the signal and is simply a measurement error. Read noise is modeled using a Gaussian distribution with zero mean. The standard deviation of the distribution controls the level of noise.

The signal-independent noise model, which uses a Gaussian distribution, is commonly used to study denoising methods [32, 33]. Although it does not fully capture the complexity of real-world noise sources, it does serve as a reasonable approximation for a wide range of situations. In well-lit environments, the Poisson distribution even becomes a Gaussian [3]. It is therefore used as noise model in this study, visualized in Figure 3.7.



(a) std = 0.0, PSNR: 100.0, SSIM: $1.0E^{0}$ **(b)** std = 0.1, PSNR: 20.5, SSIM: $5.9E^{-1}$ **(c)** std = 0.2, PSNR: 15.1, SSIM: $3.2E^{-1}$ **(d)** std = 0.3, PSNR: 12.2, SSIM: $2.0E^{-1}$ **(e)** std = 0.4, PSNR: 10.4, SSIM: $1.4E^{-1}$ **(f)** std = 0.5, PSNR: 9.2, SSIM: $1.0E^{-1}$

(g) std = 0.6, PSNR: 8.4, SSIM: $7.9E^{-2}$ **(h)** std = 0.7, PSNR: 7.8, SSIM: $6.3E^{-2}$ **(i)** std = 0.8, PSNR: 7.4, SSIM: $5.5E^{-2}$ **(j)** std = 0.9, PSNR: 7.0, SSIM: $4.8E^{-2}$ **(k)** std = 1.0, PSNR: 6.8, SSIM: $4.2E^{-2}$

Figure 3.7: Different noise levels from the Gaussian distribution in the range of standard deviation (std) [0-1]. For each sample, the PSNR and SSIM score (see Section 3.2.2), comparing the sample to the ground truth, are added to the caption. Flower image from Imagenet [6].

3.2.2. Metrics

The quantitative measures to evaluate the performance of the denoiser are SSIM and PSNR [30]. The Peak Signal-to-Noise Ratio (PSNR) compares the noisy image to the clean ground truth image, it is derived from the Mean Squared Error (MSE). The benefit of PSNR over MSE is its increased interpretability caused by the logarithm and maximum range value normalization. A higher PSNR value indicates a better-denoised image. Equation 3.1 defines the PSNR metric, with p the number of pixels in ground-truth image X and denoised image $\hat{X}$. A pixel intensity can be indexed using i with X_i. MAX is the maximum range of pixel intensities which is 1 in our case.

$$\begin{aligned} \text{PSNR} &= 10 \cdot \log_{10}(\frac{\text{MAX}^2}{\text{MSE}}), \\ \text{MSE} &= \frac{1}{p}\sum_{i=1}^{p}(\hat{X}_i - X_i)^2 \end{aligned} \tag{3.1}$$

The Structural Similarity Index (SSIM) measures the similarity between two images [30]. Different from PSNR, which compares images pixel by pixel, SSIM quantifies how similar the visible structures are in the image. This is achieved by calculating the SSIM score using multiple windows of the image. For each window pair, denoised and ground-truth, the SSIM metric is computed. The values of all windows are combined into one SSIM score for the image. SSIM scores can range between -1 and 1, a higher score corresponds to a better-denoised image. The mathematical definition of SSIM is complex, we therefore refer to [30]. Figure 3.7 also shows the PSNR and SSIM score for each noise level.

PSNR and SSIM quantify different visual qualities. [30] shows how a ground-truth image can be distorted in five different ways, visible in Figure 3.8, with each of these 5 distortions resulting in the same PSNR score but a different SSIM score. When compared visually, there is a significant difference in quality between the distorted images. This might indicate that the PSNR score is not able to quantify the image quality in line with human perception. However, PSNR simply scores differently than SSIM.

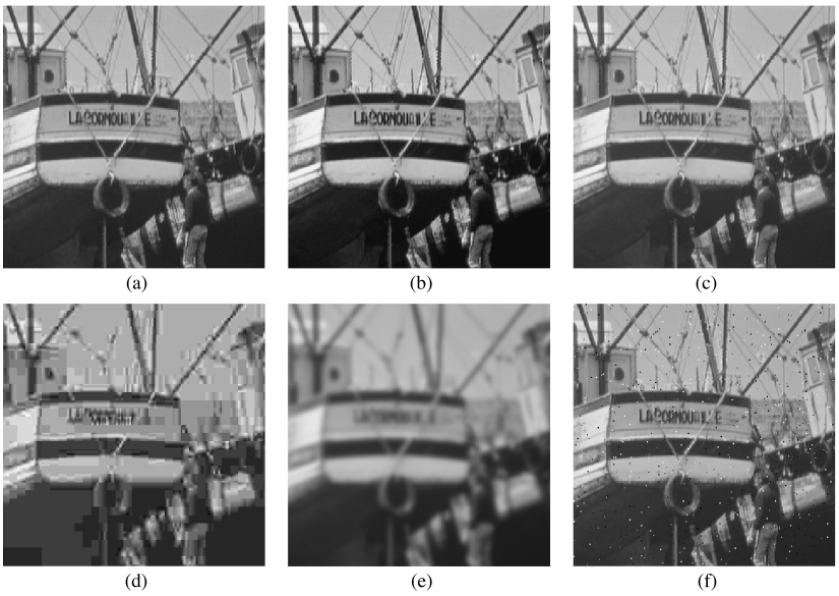


Figure 3.8: (a) Original image, (b-f) Different distortions of the original image. All images have the same PSNR score, yet they are visually different. The SSIM scores are, a: 1.0, b: 0.92, c: 0.99, d: 0.69, e: 0.71. These scores are in line with human perception. Figure from [30].

In [24], an example is given where an original image is altered in two ways, Figure 3.9. First, by adding a patch with a similar structure but visually significantly different. Second, by adding a barely visible noise. The visually significant different image scored better with SSIM than the noisy variant. Therefore, SSIM does not capture completely capture denoising performance either. However, when both SSIM and PSNR are taken into account, they quantify denoising performance well.

Figure 3.9: (a) Original images, (b) Patches of the images are replaced, (c) The original image with uniformly sampled noise added. The SSIM scores with (a) as ground truth are better for (b) than (c). Even though (c) is visually better than (b). Figure from [24].

3.2.3. Loss function

Both $\mathcal{L}_1$ and $\mathcal{L}_2$ loss are popular loss functions for image denoising [17]. These losses compare the ground truth and denoised image pixel by pixel, their definition is given in Equation 3.2. Again p the number of pixels in ground-truth image X and denoised image $\hat{X}$. The pixel intensity can be indexed per pixel i with X_i.

$$\begin{aligned} \mathcal{L}_1 &= \frac{1}{p}\sum_{i=1}^{p}|\hat{X}_i - X_i|, \\ \mathcal{L}_2 &= \frac{1}{p}\sum_{i=1}^{p}(\hat{X}_i - X_i)^2 \end{aligned} \tag{3.2}$$

Similar to the PSNR metric, these losses only compare the ground truth and denoised image pixel by pixel. To address this, [34] proposes a weighted combination of $\mathcal{L}_1$ and SSIM, which is reported to have the best denoising performance. Instead of using $\mathcal{L}_1$ in our combined loss, we use a combination of Charbonnier [4] and SSIM loss. The Charbonnier loss behaves similarly to $\mathcal{L}_1$ loss for larger errors and is smoothed for small errors by using a small constant ϵ, defined in Equation 3.3. This smoothness for smaller values eases differential optimization [8]. The Charbonnier function is used as a loss function in recent denoising studies [1, 9, 29].

$$\mathcal{L}_{\text{Charbonnier}} = \sqrt{(\mathcal{L}_1)^2 + \epsilon^2} \tag{3.3}$$

3.2.4. U-Net

U-Net [22] was originally proposed for medical image segmentation. The model has since become popular for denoising too [2, 7, 11], due to its hierarchical feature maps and skip connections between the encoder and decoder. This section will explain these benefits and the U-Net architecture.

Besides normal convolutions, the U-Net architecture uses *max pool* and *transposed convolution* operators. The *max pool* operator reduces the resolution of the feature map by a factor of two. It does so by dividing

the feature maps into grids with cells of 2 x 2 pixels, as in Figure 3.10. For each of the cells, the maximum pixel value is taken and used in the output feature map. A *transposed convolution*[1] is similar to a normal convolution but increases the resolution of the feature map.

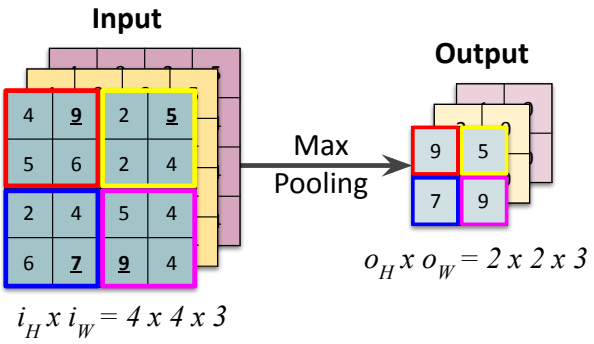


Figure 3.10: Max pooling operation example.

The U-Net encoder has five layers, and its decoder has four, see Figure 3.11. Both the en- and decoder layers follow a structure. All encoder layers have two convolutions operators, where the first convolution doubles the number of channels in the feature map, and the second keeps the number of channels constant. After each of the first four encoder layers, a max pooling layer is added to reduce the resolution. All decoder layers use a transposed convolution to increase the resolution of the feature map. The channels of this feature map are stacked with the channels of the skip connection originating from the corresponding encoder level. This combined feature map is reduced in the number of channels by the first normal convolution of the decoder level and then followed by another normal convolution. After the last decoder level, a point-wise convolution is used to reduce the feature map to one channel for gray-scale images and three channels for RGB images. Since we want to output the denoised image with the same resolution as the input image, all the convolution operators in our U-Nets use padding, which differs with [22].

The hierarchical feature maps form a benefit of U-Net. These are constructed by halving the resolution after every encoder level. This significant reduction in resolution allows the network to capture the context of the full image [22]. The decoder uses a combination of this captured context from the encoder and the finer details preserved in the skip-connections to construct the full-size denoised image [7].

[1]https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11

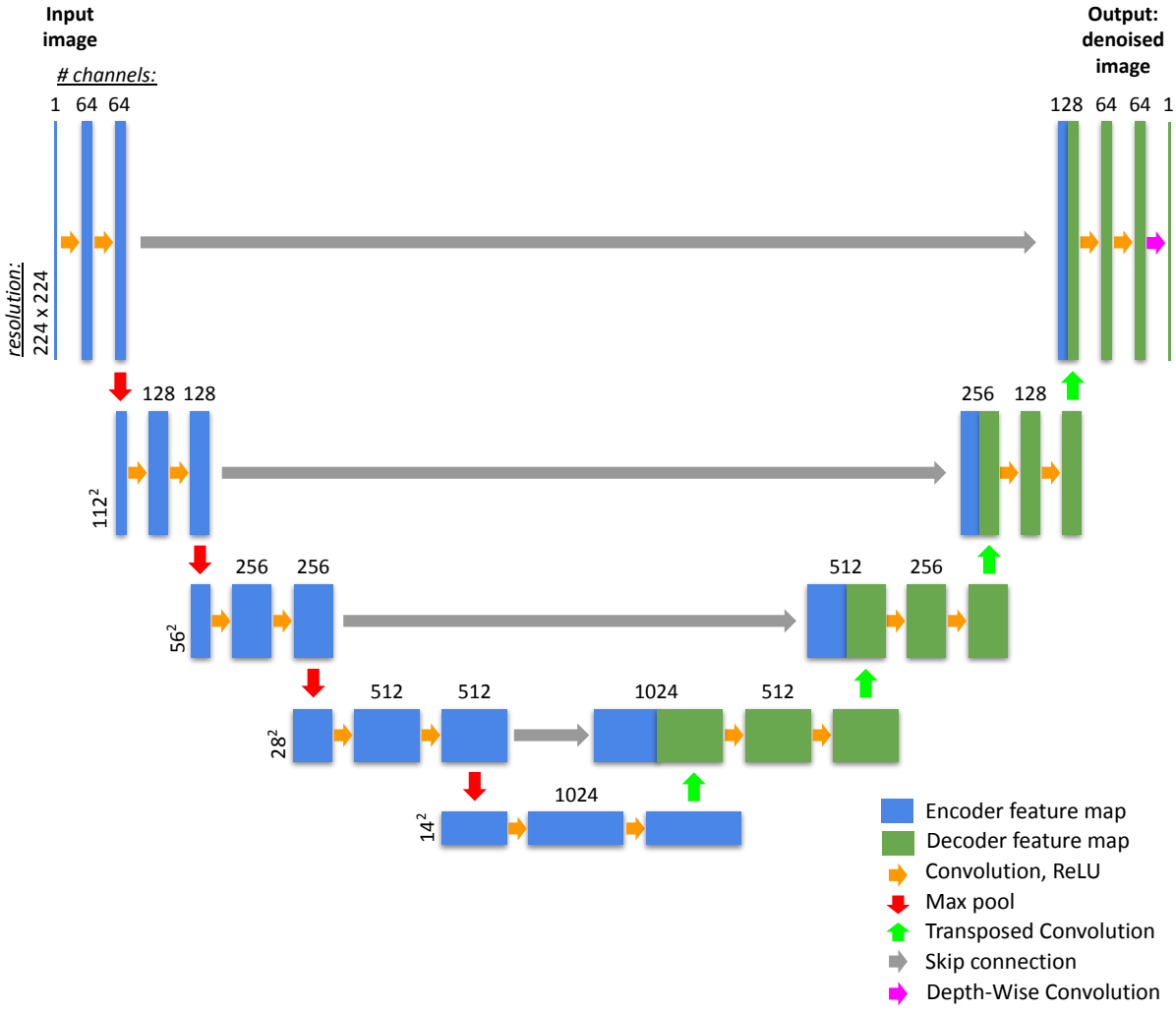


Figure 3.11: U-Net architecture, similar to [22].

3.3. Image Classification

Convolutional Neural Networks (CNNs) are popular models for image classification. In a CNN, convolutional layers are used to detect local features, such as edges and textures in the image. Max pooling layers downsample the feature map to reduce the resolution, which improves the understanding of the full image context. Fully connected layers then combine the feature maps for classification by predicting a probability for each class. A simple classification model could use two convolutional operators, each with a max pooling layer, followed by one fully connected layer. However, popular complex classifiers [12, 25, 27], use dozens of convolutional layers and multiple fully connected layers.

For an image classification model, trained on a dataset with three different classes, the output is a vector with three values. Each value in the vector corresponds to a class and is the predicted probability that the input image is of that class. To construct this probability vector, a softmax is taken over the output of the last fully connected layer. The softmax function normalizes the values between zero and one and ensures that the values add up to one. Equation 3.4 defines the softmax, where $\bar{y}_i$ is the output of the fully connected layer for class i and $\hat{y}_i$ is the corresponding class probability.

$$\hat{y}_i = \frac{exp(\bar{y}_i)}{\sum_j exp(\bar{y}_j)} \tag{3.4}$$

Taking [cat, dog, sheep] as an example, with $\hat{y} = [0.7, 0.1, 0.2]$, then cat has the highest probability. The ground truth of a cat image would be $[1, 0, 0]$. During training, the Cross-Entropy (CE) loss is used to quantify the difference between these predicted probabilities $\hat{y}$ and the ground truth. The CE loss encourages models to assign high probabilities to the correct class. Larger errors in predictions are penalized stronger, which makes the optimization process effective. To compute the CE loss, only the prediction $\hat{y}_k$ corresponding to the correct class k is used. The loss is then the negative logarithm of $\hat{y}_k$. Since $\hat{y}_k$ is supposed to be one, the logarithm is large when $\hat{y}_k$ is close to zero and small when $\hat{y}_k$ is close to one. Equation 3.5 defines the CE loss, where t_i is 1 if the input image is of class i and 0 otherwise, and $\hat{y}_i$ is the predicted value for class i.

$$\mathcal{L}_{CE} = -\sum_i t_i \cdot \log \hat{y}_i \tag{3.5}$$

3.4. Neural Architecture Search

3.4.1. NAS Approaches

Neural Architecture Search (NAS) is used to streamline the process of designing neural networks. The general idea is to search for an architecture in a manually defined search space. When trained, this architecture should score well on the defined metrics. One of the key elements of NAS is the search space definition, as searching in a poor search space cannot lead to finding a proper architecture.

To study the three most common NAS approaches, we propose a simple search space. Figure 3.12 displays the skeleton of the architecture to be searched, for each of the four blocks an operator needs to be selected. In a human-designed architecture, each block would have a manually picked operator assigned. Whereas with NAS, only the set of operators to choose from is human-designed. The options are {*Convolution*, *MBConv*, *Skip-connection*}. The *Skip-Connection* is a common operator candidate as using a skip-connection is analogous to removing the block from the network, thus altering the network architecture. This simple search space already amounts to $3^4 = 81$ architectures.

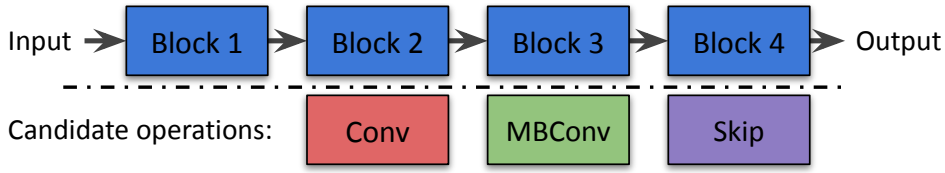


Figure 3.12: Examplary search space for Neural Architecture search, consisting of 4 sequential blocks which need an operator to be assigned. The operators can be selected from the candidate operators: *Conv*, *MBConv* and *Skip-connection*.

The first NAS approach uses a **reinforcement learning** (RL) model, which we will call an agent to avoid confusion. This agent builds an architecture by choosing an operator for each block in the skeleton, Figure 3.12. The generated architecture is trained, and its performance is used as a reward for the agent. Based on the reward, the agent learns what decisions benefit the performance of the architecture. With this knowledge, the agent can design an architecture without searching the entire discrete search space [27, 28].

The second NAS approach is based on **evolutionary algorithms** (EA) [21] such as genetic algorithms [19]. In these approaches, a subset of the architectures in the search space is constructed. This subset called the population, is trained and evaluated. For the next iterations, the population is created by combining the architecture decisions of the best-performing architectures and then training and evaluating the models. When iterating over this process, the non-optimal design choices will be removed, like natural selection. Moreover, the search converges to the best-performing architectures in the discrete search space.

Both of these NAS approaches create one or more architectures, which need to be trained and evaluated, to then iterate extensively over this process. Since training one architecture is already time-consuming, the full search can take over 100 GPU days. **Differentiable** NAS approaches [16, 18, 31] address this issue by changing the search space to be continuous, enabling differential optimization. For which only one large model needs to be optimized to construct an architecture. This results in a search taking 1-2 GPU days. As differentiable NAS is used in this study, we elaborate further on the approach.

3.4.2. Differentiable NAS

The key idea behind differentiable NAS is representing the architectural choices as continuous rather than discrete. To understand this, we first propose a different perspective on the discrete choices made in our example for which we use a search network. In the search network, instead of using one candidate operator in each block, we use all candidate operations in parallel and perform a weighted addition of the results, see Figure 3.13. The weights, α, change the influence each candidate operator has on the outcome of the block. In the discrete search space, the vector α^b for block b sums to 1, and the values α_i^b can either be 0 or 1. When choosing the *Conv* operator in block 1, $\alpha^b = [1, 0, 0]$ which results in the *Conv* being the only operator of which the output is used, visualized in Figure 3.14a.

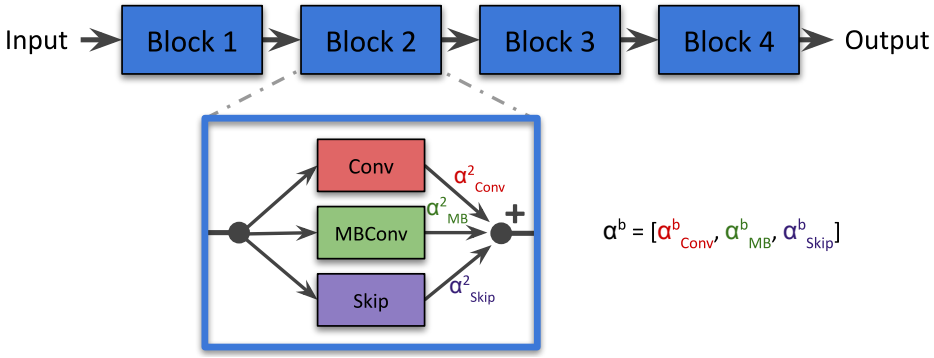
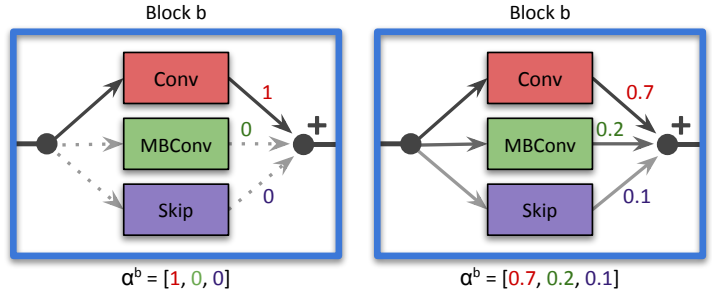


Figure 3.13: The search network where each block executes all candidate operations in parallel and adds their results using a weighted sum. For each block b, this sum is weighted using α^b, and each operator i in block b has its own weight α_i^b.



(a) Discrete representation of the parallel block. α's are such that only the output of Conv is used.

(b) Continuous representation of the parallel block. α's are such that the output of all operations is combined.

Figure 3.14

We relax this representation to be continuous by allowing α_i^b to range between 0 and 1, see Figure 3.14b. The vector α^b should still sum to 1, which is achieved by taking the softmax of the vector, see Equation 3.7. In Equation 3.6, we define the weighted sum, which is also used for the continuous block. For block b, we have the input x^b, output x^{b+1}, candidate operations op_i^b with $i \in \{Conv, MB, Skip\}$ and A_i^b the softmax of α_i^b.

$$x^{b+1} = \sum_i A_i^b \cdot \text{op}_i^b(x^b) \tag{3.6}$$

$$A_i^b = \frac{exp(\alpha_i^b)}{\sum_j exp(\alpha_j^b)} \tag{3.7}$$

When using the definition in 3.6 for all blocks in our search space, finding the optimal values for α is analogous to finding the best candidate operation for each block. The search process for these α values becomes differentiable as Equation 3.6 is differentiable with respect to α, and α is a continuous value. To find the optimal values of α, the candidate operators need to perform well, which requires their weights ω to be trained. Therefore, the search process becomes a bi-level optimization in which ω is optimized. Using the optimized weights ω^*, α is optimized. This process is repeated until convergence. Equation 3.8 shows the bi-level optimization.

$$\begin{aligned} &\min_{\alpha} \mathcal{L}(\omega^*, \alpha) \\ \text{s.t.} \quad &\omega^* = \arg\min_{\omega} \mathcal{L}(\omega, \alpha) \end{aligned} \tag{3.8}$$

Obtaining the optimal weights ω^* for each update of α is time-consuming. When $O(n)$ updates are required to optimize α and ω separately, then $O(n^2)$ updates are required to perform the bi-level optimization. Instead, most differentiable NAS methods [16, 18, 31] approach ω^* using only one update step. Therefore, each α update requires only one ω update, resulting in $O(2n) \Rightarrow O(n)$ steps for the bi-level optimization.

When the bi-level optimization converges, one of the candidate operations for each block needs to be selected. For block b, the candidate operation op_i^b with the highest corresponding α_i^b is selected. Selecting the operator using the max α_i^b value is appropriate, as during search α is optimized using a *softmax*, which is a differentiable alternative to standard *max* operation. The architecture constructed using the selected operators, called the train network, is the network that is proposed by the differentiable NAS method. This train network is subsequently trained with a clean initialization.

3.4.3. TF-NAS

The differentiable NAS (DNAS) method used in this study is TF-NAS [16]. TF-NAS suits our study as it addresses issues in earlier DNAS methods, the classification architecture it proposes is similar to a U-Net encoder, and it optimizes the architecture for a target latency. These three properties will be discussed in the following sections.

Improvements over DNAS

TF-NAS addresses two main issues of earlier DNAS methods [18].

The first issue is called operation collapse. This happens when the ω of a specific operation is optimized faster than competing operations. Such quick-to-optimize operations often have few weights to learn. Since these operations have better-optimized weights, they perform better early in training leading to an increase in their α value. This might cause the search procedure to select and stick with an operator that performs well in the early stages of training but would be outperformed by another operator when trained longer. TF-NAS addresses this issue by altering the optimization procedure of ω. One of the changes is replacing the softmax in Equation 3.7 with a different softmax. Since these changes are used but not further discussed in this study, we refer to [16] for further details.

The second issue with previous work [18, 31] relates to the use of skip connections in the proposed architecture. Skip connections are used as a choice to reduce the number of blocks used in the proposed architecture. If a skip connection has the highest α, the block will be removed from the proposed architecture. Similar to our example search space, many NAS methods use a skip connection to compete directly with the other candidate operations. This introduces instability in the search, where the derived architectures remove too many blocks [16]. Therefore, TF-NAS proposes to extract the skip connection from the parallel block. To replace them, skip connections are introduced, starting after every block and connecting to the same point after the last convolution, where a weighted addition is performed. The addition is weighted by a new parameter β, similar to α. This new configuration, shown in Figure 3.15, lets the skip-connections compete amongst each other instead of against the other candidate operations. In Equation 3.9, the definition is given, the softmax is used again to normalize β.

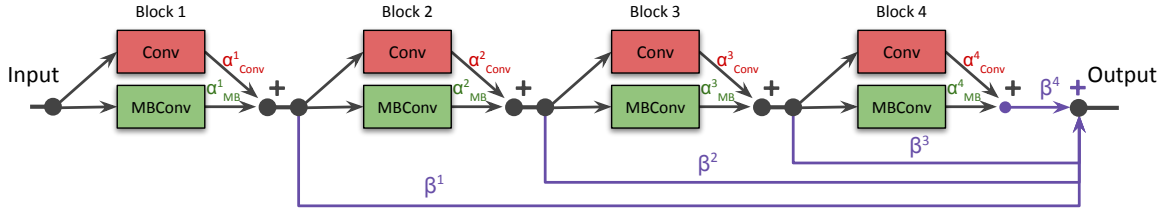


Figure 3.15: Extracting the skip connection from α and using β to weight the new skip connections in the new configuration.

$$x^{out} = \sum_b B^b \cdot x^b,$$
$$B^b = \frac{exp(\beta^b)}{\sum_d exp(\beta^d)} \tag{3.9}$$

Search space

The search space is based on the EfficientNet architecture [26]. Therefore, a classifier is searched, shown in Figure 3.16. The first and last two stages are manually set, while the stages in between are searchable. Each searchable stage contains between 1 and 4 blocks. A block can get any of the candidate operations assigned. The candidate operations are MBConvs with 8 different configurations. Each operation has a kernel size $k = 3$ or $k = 5$ for the depth-wise convolution, an expansion rate $e = 3$ or $e = 6$, and the possibility to add a squeeze- and excitation-layer (SE-layer) [15]. All combinations of these three choices construct the 8 different configurations listed in Table 3.1. The SE-layer is a method that introduces additional weights to a convolutional operator. We do not further explain it, as it will not be used in this study.

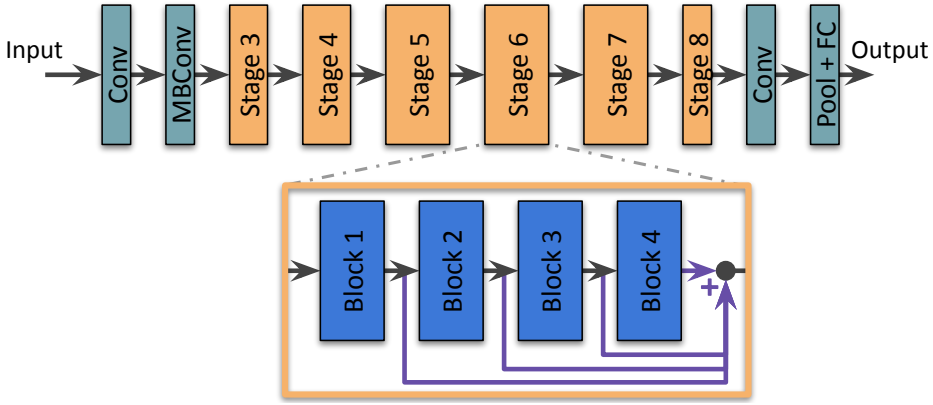


Figure 3.16: The full search space of TF-NAS, the green stages are manually chosen, while the orange stages can be searched. The width of each orange stage indicates the number of searchable blocks it contains. The orange block shows how a stage with 4 blocks looks internally.

Name	Kernel	Expansion rate	SE-layer
MB-k3-e3	3	3	-
MB-k3-e6	3	6	-
MB-k5-e3	5	3	-
MB-k5-e6	5	6	-
MB-k3-e3-se	3	3	✓
MB-k3-e6-se	3	6	✓
MB-k5-e3-se	5	3	✓
MB-k5-e6-se	5	6	✓

Table 3.1: Overview of the candidate MBConv operators with different kernel sizes *k*, expansion rate *e* and in- or excluding the squeeze- and excitation-layer.

Latency optimization

TF-NAS searches an architecture with a configurable target latency. The network latency is controlled by α and β. α controls which operators are used in the network. One operator is slower than the other. β controls the number of blocks in a stage, reducing the operations also reduces the latency. The last parameter influencing the latency, is the scaling factor which slightly alters the expansion rate of the MBConvs. This MBConv scaling is further discussed in [16] and not required knowledge for this study.

The latency is taken into account by adding a latency loss to the bi-level optimization defined in Equation 3.8. For each block, the latency is predicted by performing a weighted addition of the previously measured operator latencies. The weight is α. Equation 3.10 defines the latency prediction LAT^b for block b, with operators op_i^b, $\text{LAT}(\text{op}_i^b)$ which returns the latency of a candidate operation *op* and A_i^b defined in Equation 3.7. $\text{LAT}()$ is a look-up table which can be constructed for a specific device, by measuring the latencies of all operators.

$$\text{LAT}^b = \sum_i A_i^b \cdot \text{LAT}(\text{op}_i^b) \tag{3.10}$$

The latency predicted per block is added for all blocks in a stage, again weighted but this time using β. Combining the predicted latencies of all stages results in a total predicted latency LAT, which is dependent on the architecture parameters α and β. TF-NAS aims to find an architecture for which LAT equals the target latency lat_{target}. Therefore the latency loss should equal 0 when LAT $< lat_{\text{target}}$. Combined with the normalization of LAT, this results in the following function to calculate the latency loss. See Equation 3.11.

$$\mathcal{L}_{\text{lat}} = \max(\frac{\text{LAT}(\alpha, \beta)}{lat_{\text{target}}} - 1, 0) \tag{3.11}$$

The latency loss is added to the bi-level optimization with a weight λ, which is set to 0.5 in this study. Equation 3.12 defines this updated optimization.

$$\begin{aligned} &\min_{\alpha,\beta} \mathcal{L}(\omega^*, \alpha, \beta) + \lambda \cdot \mathcal{L}_{\text{lat}} \\ &\text{s.t.} \quad \omega^* = \arg\min_{\omega} \mathcal{L}(\omega, \alpha) \end{aligned} \tag{3.12}$$

References

[1] Yildiray Anagun, Sahin Isik, and Erol Seke. "SRLibrary: Comparing different loss functions for super-resolution over various convolutional architectures". In: *Journal of Visual Communication and Image Representation* 61 (2019), pp. 178–187. ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2019.03.027. URL: https://www.sciencedirect.com/science/article/pii/S1047320319301336.

[2] Long Bao et al. "Real Image Denoising Based on Multi-Scale Residual Dense Block and Cascaded U-Net With Block-Connection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2020.

[3] C. Bouman and K. Sauer. "A generalized Gaussian image model for edge-preserving MAP estimation". In: *IEEE Transactions on Image Processing* 2.3 (1993), pp. 296–310. DOI: 10.1109/83.236536.

[4] P. Charbonnier et al. "Two deterministic half-quadratic regularization algorithms for computed imaging". In: *Proceedings of 1st International Conference on Image Processing*. Vol. 2. 1994, 168–172 vol.2. DOI: 10.1109/ICIP.1994.413553.

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

[6] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[7] Chi-Mao Fan, Tsung-Jung Liu, and Kuan-Hsien Liu. "SUNet: Swin Transformer UNet for Image Denoising". In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2022. DOI: 10.1109/iscas48785.2022.9937486. URL: https://doi.org/10.1109%2Fiscas48785.2022.9937486.

[8] Cheng-Yang Fu, Mykhailo Shvets, and Alexander C. Berg. "RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free". In: *CoRR* abs/1901.03353 (2019). arXiv: 1901.03353. URL: http://arxiv.org/abs/1901.03353.

[9] Binit Gajera et al. "CT-Scan Denoising Using a Charbonnier Loss Generative Adversarial Network". In: *IEEE Access* 9 (2021), pp. 84093–84109. DOI: 10.1109/ACCESS.2021.3087424.

[10] Javier Gurrola-Ramos, Oscar Dalmau, and Teresa E. Alarcón. "A Residual Dense U-Net Neural Network for Image Denoising". In: *IEEE Access* 9 (2021), pp. 31742–31754. DOI: 10.1109/ACCESS.2021.3061062.

[11] Javier Gurrola-Ramos, Oscar Dalmau, and Teresa E. Alarcón. "A Residual Dense U-Net Neural Network for Image Denoising". In: *IEEE Access* 9 (2021), pp. 31742–31754. DOI: 10.1109/ACCESS.2021.3061062.

[12] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[13] Andrew Howard et al. "Searching for MobileNetV3". In: *CoRR* abs/1905.02244 (2019). arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244.

[14] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[15] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-Excitation Networks". In: *CoRR* abs/1709.01507 (2017). arXiv: 1709.01507. URL: http://arxiv.org/abs/1709.01507.

[16] Yibo Hu, Xiang Wu, and Ran He. "TF-NAS: Rethinking Three Search Freedoms of Latency-Constrained Differentiable Neural Architecture Search". In: *CoRR* abs/2008.05314 (2020). arXiv: 2008.05314. URL: https://arxiv.org/abs/2008.05314.

[17] Saeed Izadi, Darren Sutton, and Ghassan Hamarneh. "Image denoising in the deep learning era". In: *Artificial Intelligence Review* 56.7 (July 2023), pp. 5929–5974. ISSN: 1573-7462. DOI: 10.1007/s10462-022-10305-2. URL: https://doi.org/10.1007/s10462-022-10305-2.

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "DARTS: Differentiable Architecture Search". In: *CoRR* abs/1806.09055 (2018). arXiv: 1806.09055. URL: http://arxiv.org/abs/1806.09055.

[19] Hanxiao Liu et al. "Hierarchical Representations for Efficient Architecture Search". In: *CoRR* abs/1711.00436 (2017). arXiv: 1711.00436. URL: http://arxiv.org/abs/1711.00436.

[20] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[21] Esteban Real et al. "Regularized Evolution for Image Classifier Architecture Search". In: *CoRR* abs/1802.01548 (2018). arXiv: 1802.01548. URL: http://arxiv.org/abs/1802.01548.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[23] Mark Sandler et al. "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

[24] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. "On the Suitability of Lp-Norms for Creating and Preventing Adversarial Examples". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2018), pp. 1686–16868. URL: https://api.semanticscholar.org/CorpusID:3576692.

[25] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[26] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[27] Mingxing Tan and Quoc V. Le. "EfficientNetV2: Smaller Models and Faster Training". In: *CoRR* abs/2104.00298 (2021). arXiv: 2104.00298. URL: https://arxiv.org/abs/2104.00298.

[28] Mingxing Tan et al. "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: *CoRR* abs/1807.11626 (2018). arXiv: 1807.11626. URL: http://arxiv.org/abs/1807.11626.

[29] Zhendong Wang et al. "Uformer: A General U-Shaped Transformer for Image Restoration". In: *CoRR* abs/2106.03106 (2021). arXiv: 2106.03106. URL: https://arxiv.org/abs/2106.03106.

[30] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[31] Bichen Wu et al. "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search". In: *CoRR* abs/1812.03443 (2018). arXiv: 1812.03443. URL: http://arxiv.org/abs/1812.03443.

[32] Kai Zhang, Wangmeng Zuo, and Lei Zhang. "FFDNet: Toward a Fast and Flexible Solution for CNN based Image Denoising". In: *CoRR* abs/1710.04026 (2017). arXiv: 1710.04026. URL: http://arxiv.org/abs/1710.04026.

[33] Kai Zhang et al. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising". In: *CoRR* abs/1608.03981 (2016). arXiv: 1608.03981. URL: http://arxiv.org/abs/1608.03981.

[34] Hang Zhao et al. "Loss Functions for Image Restoration With Neural Networks". In: *IEEE Transactions on Computational Imaging* 3.1 (2017), pp. 47–57. DOI: 10.1109/TCI.2016.2644865.