Tunable Models for Image Processing

Daniel Afrimi

January 21, 2021

1 Introduction

Deep convolutions neural network has demonstrated its capability of learning a deterministic mapping for the desired imagery effect. However, the large variety of user flavors motivates the possibility of continuous transition among different output effects.

For example if our task is image denoisng, the most common method is by supervised models (deep learning), when the input is a noisy image, and the output will be the clean image (fixed pre-determined corruption level). However, models are optimized for only a single degradation level, so if we have a batch of noisy images with different Gaussian noise, our model will not clean the images well (in reality, the degradation level is not known and at inference time, the model can under-perform).

In order to overcome this limitation, continuous-level based models have been proposed. in such models, the output image is based on a target parameter which can be adjusted at inference time.

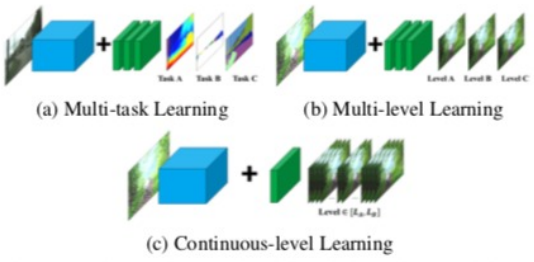


Figure 1: Introduction Description

2 Related Work

2.1 DNI - Deep Interpolation Network

2.1.1 Main Idea

Continuous imagery effect transition is achieved via linear interpolation in the parameter space of existing trained networks (for $numbernets \geq 2$). provided with a model for a particular effect A, they fine-tune it to realize another relevant effect B. DNI applies linear interpolation for all the corresponding parameters of these two deep networks (basically for the filters, normalization (IN) and the biases).

2.1.2 Network Interpolation

If you train two models from scratch on the same task, and perform visualization of the filters (first and last filters), It can be seen that the order of the filters is different but their representation is similar (though not in

the same order). Fine-tuning, however, can help to maintain the filters's order and pattern. The "high correlation" between the parameters of these two networks provides the possibility for meaningful interpolation.

In **Figures 2** it can be seen that the filters for different types of noise is strong correlated and when fine-tune several models for relevant tasks (different types of noise) from a pretrained one (N20), the correspondingly learned filters have intrinsic relations with a smooth transition, measuring the correlation between two filter (similar to the Pearson correlation) results close relationship among the learned filters, exhibiting a gradual change as the noise level increases.

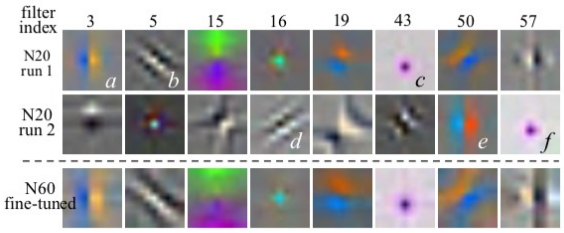


Figure 2: Filter correlations. The first two rows are the filters of different runs (both from scratch) for the denoising (N20) task. The filter orders and filter patterns in the same position are different. The *fine-tuned* model (N60) ($3rd$ row) has a "strong correlation" to the pre-trained one ($1st$ row).

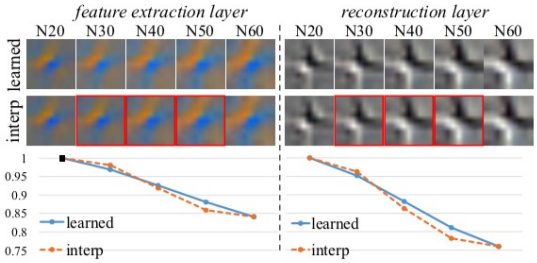


Figure 3: Filters with gradual changes. We show one representative filter for each layer. **$1st$ row**: The fine-tuned filters for different noise level show gradual changes. **$2nd$ row**: The interpolated filters (with red frames) from the N20 and N60 filters could visually fit those learned filters well. **$3rd$ row**: The correlation curves for learned and interpolated filters are also very close.

Figure 2: Filter Visualization (left) and filter gradual changes in different noises (right)

2.1.3 Training

In the paper they trained a model on some task (e.g. denoising), at first they trained a model from scratch on a certain type of noise (Gaussian with some standard deviation), then after the training they did fine-tuning to a different noise (Gaussian with standard deviation - last level).

2.2 AdaFM - Adaptive Feature Modification

2.2.1 Main Idea

The additional module, namely AdaFM layer, performs channel-wise feature modification, and can adapt a model to another restoration level with high accuracy. the aim is to add another layer to manipulate the statistics of the filters, so that they could be adapted to another restoration level.

Adaptive Feature Modification (AdaFM) layers are inspired by the recent normalization methods in deep CNNs, but in contrast AdaFM layer is independent of either batch or instance samples, filter size and position of AdaFM layers are flexible, the interpolation property of AdaFM layers could achieve continual modulation of restoration levels.

Because there are "high correlation" between the filters of the networks (as we saw in DNI), we can think of a function that performs the match between them those filters. To further reveal their relationship, they use a filter to bridge the corresponding filters - g (done by depth-wise convolution layer after each layer).

In addition, the filter g is gradually updated by gradient descent, so we can obtain the intermediate filter to get smooth transition.

In **Figures 3** we can see the architecture of AdaFM-Net.

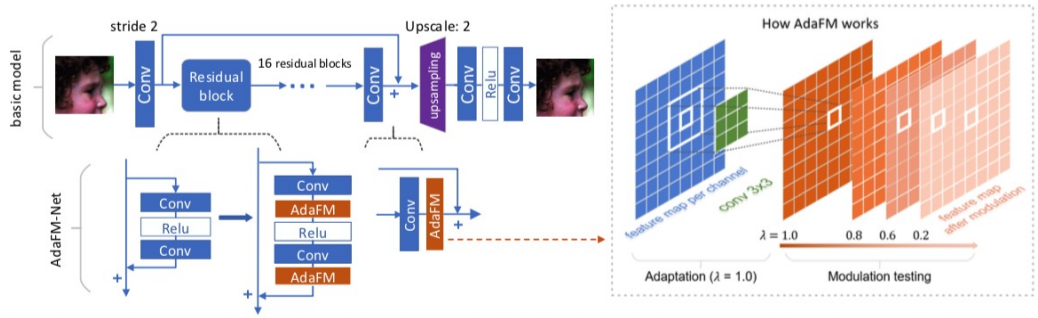


Figure 4. The left part presents the basic model and the AdaFM-Net. The right part shows how AdaFM works in the adaptation process and the modulation testing.

Figure 3: AdaFM-Net

The model mainly deals with restoration tasks (e.g denoisng, SR)

2.2.2 Training

Train a basic model and an adaptive model that could deal with level L_a and L_b, respectively. While in modulation testing, they propose a new network that can realize arbitrary restoration effects between level L_a and L_b by modulating certain hyper-parameters.

1. Train a basic network (CNN) for the start restoration level L_a, we call this network N^a_{bas}.
2. Insert AdaFM layers to N^a_{bas} and obtain N_{ada}. fixing the parameters of N^a_{bas}.
3. Optimize the parameters of AdaFM on the end level L_b (finetuning the AdaFM layers)

2.2.3 Adaptation Accuracy

according the paper as the AdaFM-Net is optimized from L_a to L_b, they name this process as adaptation, and use adaptation accuracy to denote its performance.

There are 3 factors that affect this adaptation accuracy:

- Filter Size - a larger filter size or more parameters will lead to better adaptation accuracy.
- Direction - From easy to hard is a better choice.
- Range - The smaller of the range/gap (L_a - L_b), the better the adaptation accuracy.

In **Figures 4** we can see some results.

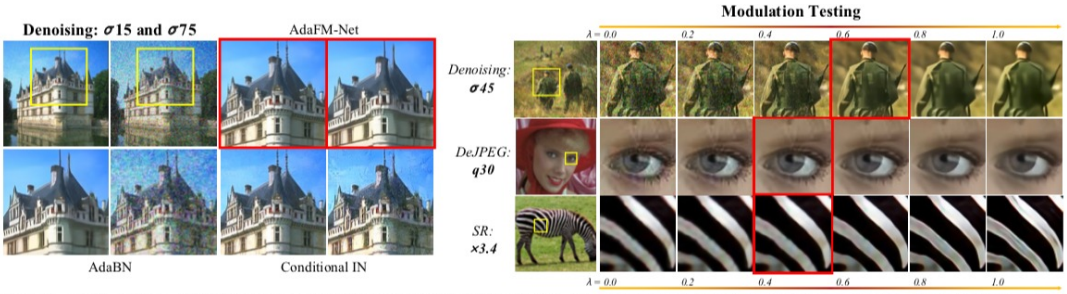


Figure 6. Left: Artifacts on the output images produced by AdaBN and conditional instance normalization. Right: Modulation testing in Denoising (CBSD68), DeJEPG (LIVE1) and Super Resolution (Set14 [21]).

Figure 4: AdaFM-Net Results

2.3 FTN - Filter Transition Network

2.3.1 Main Idea

Filter Transition Network (FTN) which is a non-linear module that easily adapts to new levels (and a method to initialize non- linear CNNs with identity mappings).

FTN takes **CNN filters** as input (instead of image/feature map) and learns the transitions between levels (the modification is non-linear which can better adapt to new level). The FTN layer basically transform the filters of the network.

FTN module in an arbitrary convolutional layer can be described as $Y_i = X_i * \big(f_i^1 \times (1-\alpha) + FTN(f_i^1) \times \alpha\big)$, where X_i and Y_i are the input and output of the i-th convolution layer, f_i^1 is the corresponding kernel, and α is control parameter.

In **Figures 5** we can see the architecture of FTN.

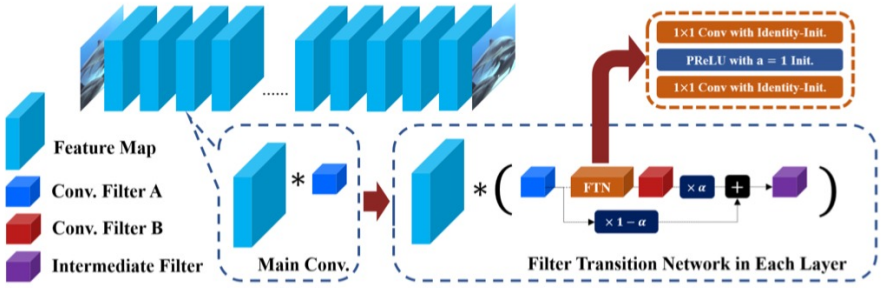


Figure 2: Network architecture of the proposed Filter Transition Network (FTN) when adapted in arbitrary main convolutional networks. Filter of main network (blue) is transformed via FTN for other levels (red). In inference phase, interpolated filter (purple) is used for intermediate levels

Figure 5: FTN

2.3.2 Training

- Train the main network for the initial level with $\alpha = 0$.
- Freeze the main network and train the FTN only for the second level with $\alpha = 1$ (breaks skip-connection).

- At inference time, we can interpolate between two kernels (levels) by choosing $\alpha \in [0, 1]$.

FTN learns the transition itself, and can approximates kernels of the second level (because we optimize only its parameters during the second level)

2.3.3 Initialization of FTN

Init the filter and the activation(PReLU with a=1) as the identity (for keeping the network works for the initial level).

To check with mike how to init the filter - 1 in the middle and the rest is 0?

3 Model Comparison

- **AdaFM** - The layer is a linear transition block with depth-wise convolution. Due to the linear interpolation method, it might also fail to overcome non-linear interpolation requirements. Therefore, it is not appropriate for more complex tasks such as the perception-distortion (PD) trade-off in restoration or style transfer.
- **DNI** - At inference time, it linearly interpolates all the parameters of the networks. can fail when the required interpolation is not linear. fine-tuning the network without any constraint cannot consider the initial level, which might lead to a degraded performance at intermediate levels.
- **FTN** - Enables interpolation of the kernel between initial and final one in a non-linear manner

In **Figures 6** we can see the Model Comparison to 4 categories. it can be seen that:

1. AdaFM works mainly for restoration tasks and for other task (e.g Style Transfer) the adaption won't be good.
2. DNI requires extra memory to save temporary network parameters and requires a third interpolated network for the inference. in addition fine-tuning process of the DNI simply updates the parameters, without considering the initial state. Therefore, the relation between the parameters for the two levels becomes weaker
3. CFS-Net and Dynamic Net propose frameworks that interpolate the feature maps using tuning branches **TODO read this paper** and tuning this branches requires large memory and heavy computations.

	No Extra Memory	Non-linear Adaptation	Efficient	Regularized
AdaFM [12]	✓	✗	✓	✓*
DNI [33,34]	✗	✓	✗	✗
Dynamic-Net [30]	✓	✓	✗	✗
CFSNet [32]	✓	✓	✗	✗
FTN (Ours)	✓	✓	✓	✓

Figure 6: Comparison - Taken from FTN paper

In **Figures 7** we can see that on denoising task DNI works the best (in a CNN network of 10 blocks). However when testing it on Ada-FM-Net (CNN network of 16 blocks) FTN works the best.
In addition we can see that the number of parameters and the computation for FTN and AdaFM (for several tasks) is relative small.

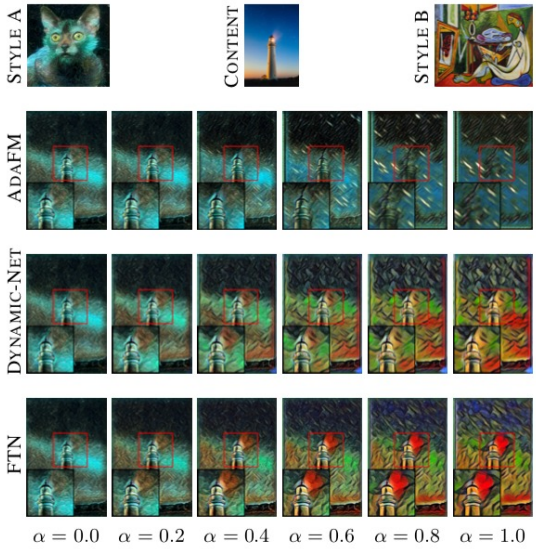
Table 3: Overall computations, relative computations from baseline (in percentage), and number of parameters of the frameworks. Setting and network configurations are described in Sec. 4.1

Task	Denoising		×2 Super-Resolution		Style Transfer	
Network	AdaFM-Net		CFSNet-30		Transform-Net [15]	
	GFLOPs	Params(M)	GFLOPs	Params(M)	GFLOPs	Params(M)
Baseline	25.11	1.41	155.96	2.37	40.42	1.68
+ CFSNet [32]	46.96 (87.02%)	3.06	311.36 (99.64%)	4.93	-	-
+ AdaFM [12]	26.01 (3.58%)	**1.46**	162.50 (4.20%)	**2.47**	41.73 (3.24%)	**1.72**
+ Dynamic-Net [30]	-	-	-	-	62.24 (53.98%)	2.59
+ **FTN**	**25.36 (0.10%)**	1.83	**156.34 (0.02%)**	3.01	**40.86 (1.09%)**	2.06
+ **FTN(G=16)**	**25.13 (0.01%)**	**1.44**	**156.00 (0.00%)**	**2.41**	**40.46 (0.10%)**	**1.70**

Table 2: **Gaussian Denoising Results.** Average PSNR (dB) on CBSD68 test dataset. Unseen noise levels are denoted with *. The best results are **bold-faced** and the second best results are underlined.

	CFSNet-10 (10 Blocks)				AdaFM-Net (16 Blocks)			
Noise Level	20	40*	60*	80	20	40*	60*	80
From Scratch	32.42	28.98	27.13	25.90	32.44	28.90	26.92	25.60
DNI	**32.42**	**28.87**	**27.01**	**25.96**	**32.44**	28.20	26.98	25.97
AdaFM	**32.42**	28.48	26.75	25.84	**32.44**	28.17	26.77	25.96
CFSNet	**32.42**	28.65	26.95	25.93	**32.44**	28.41	26.87	26.00
FTN-gc16	**32.42**	28.77	**27.01**	25.89	**32.44**	**28.78**	**27.05**	25.98
FTN-gc4	**32.42**	28.65	26.90	25.90	**32.44**	28.64	26.95	26.00
FTN	**32.42**	28.45	26.86	25.93	**32.44**	28.48	26.89	**26.03**

Figure 7: Comparison between the models - computation memory (left) Denoising Task (right)



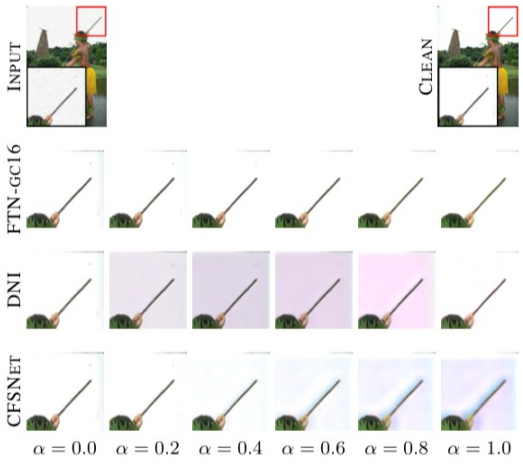


Figure 8: **Denoising results on weak noise level ($\sigma = 20$). When user controls α to a large value, over-smoothing artifacts arise**

Figure 8: FTN gets better result in denoising task (right) and for mixing style (left)

4 Appendix

4.1 PSNR - Peak Signal to Noise

The term peak signal-to-noise ratio (PSNR) is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation.

Using the same set of tests images, different image enhancement algorithms can be compared systematically to identify whether a particular algorithm produces better results. The metric under investigation is the peak-signal-to-noise ratio. If we can show that an algorithm or set of algorithms can enhance a degraded known image to more closely resemble the original, then we can more accurately conclude that it is a better algorithm.

$$PSNR = 20 \log_{10} \left(\frac{MAX_f}{\sqrt{MSE}} \right)$$

Figure 9: PSNR- MAX_f is the maximum signal value that exists in the original image