A Appendix

A.1 Training Schedules

CIFAR-10 adversarial training schedule: For our baseline experiments on CIFAR-10, we used the PRN-18 as well as the WRN-28-10 architecture as they give a good trade-off between complexity and feasibility. For the PRN-18 models, we trained for 300 epochs with a batch size of 512 and a circling learning rate schedule with the maximal learning rate 0.2 and minimal learning rate 0. We set the momentum to 0.9 and weight decay to $5e^{-4}$. The loss is calculated via Cross Entropy Loss and as an optimizer, we use Stochastic Gradient Descent (SGD). For the AT, we used the FGSM attack with an ϵ of 8/255 and an α of 10/255 (in Fast FGSM the attack is computed for step size α once and then projected to ϵ). For the WRN-28-10 we used a similar training schedule as for the PRN-18 models but used only 200 epochs and a smaller maximal learning rate of 0.08.

CIFAR-10 clean training schedule: Each model is trained without AT. We used 300 epochs, a batch size of 512 for each training run and a circling learning rate schedule with the maximal learning rate at 0.2 and minimal at 0. We set the momentum to 0.9 and a weight decay to $5e^{-4}$. The loss is calculated via Cross Entropy Loss and as an optimizer, we use Stochastic Gradient Descent (SGD).

CINIC-10 adversarial training schedule: For our baseline experiments on CINIC-10 we used the PRN-18 architecture. We used 300 epochs, a batch size of 512 for each training run and a circling learning rate schedule with the maximal learning rate at 0.1 and minimal at 0. We set the momentum to 0.9 and weight decay to $5e^{-4}$. The loss is calculated via Cross Entropy Loss and as an optimizer, we use Stochastic Gradient Descent (SGD). For the AT, we used the FGSM attack with an epsilon of 8/255 and an alpha of 10/255.

CIFAR-100 adversarial training schedule: For our baseline experiments on CIFAR-100 we used the PRN-18 architecture as it gives a good trade-off between complexity and feasibility. We used 300 epochs, a batch size of 512 for each training run and a circling learning rate schedule with the maximal learning rate at 0.01 and minimal at 0. We set the momentum to 0.9 and a weight decay to $5e^{-4}$. The loss is calculated via Cross Entropy Loss and as an optimizer, we use Stochastic Gradient Descent (SGD). For the AT, we used the FGSM attack with an epsilon of 8/255 and an alpha of 10/255.

ImageNet adversarial training schedule: For our experiment on ImageNet we used the ResNet50 architecture. We trained for 150 epochs with a batch size of 400, and a multistep learning rate schedule with an initial learning rate 0.1, $\gamma = 0.1$, and milestones $[30, 60, 90, 120]$. We set the momentum to 0.9 and weight decay to $5e^{-4}$. The loss is calculated via Cross Entropy Loss and as an optimizer, we use Stochastic Gradient Descent (SGD). For the AT, we used FGSM attack with an epsilon of 4/255 and an alpha of 5/255.

A.2 ImageNet Training Efficiency

When evaluating practical training times (in minutes) on ImageNet per epoch, we can not see a measurable difference in the costs between a ResNet50 with FLC pooling or strided convolution.

We varied the number of workers for dataloaders with clean training on 4 A-100 GPUs and measured $\approx$ 43m for 12 workers, $\approx$ 22m for 48 workers and $\approx$ 18m for 72 workers for both. FGSM-based AT with the pipeline by [42] takes 1:07 hours for both FLC pooling and strided convolutions per epoch. We conclude that training with FLC pooling in terms of practical runtime is scalable (runtime increase in ms-s range) and training times are likely governed by other factors.

The training time of our model should be comparable to the one from Wong et al. [45] while other reported methods have a significantly longer training time. Yet, the clean accuracy of the proposed model using FLC pooling improves about 8% over the one reached by [45], with a 1% improvement in robust accuracy. For example [12] has an increased training time by factor four compared to our model, already on CIFAR10 (see Table 6). This model achieves overall comparable results to ours. The model by Salman et al. [38] is trained with the training schedule from Madry et al. [32] and uses a multi-step adversarial attack for training. Since there is no release of the training script of this model on ImageNet, we can only roughly estimate their training times. Since they adopt the training schedule from Madry et al., we assume a similar training time increase of a factor of four, which is similar to the multi-step times reported for PGD in Table 6.

A.3 Aliasing Free Down-Sampling

Previous approaches like [50,51] have proposed to apply blurring operations before down-sampling, with the purpose of achieving models with improved shift invariance. Therefore, they apply Gaussian blurring directly on the feature maps via convolution. In the following, we briefly discuss why this setting can not guarantee to prevent aliasing in the feature maps, even if large convolutional kernels would be applied, and why, in contrast, the proposed FLC pooling can guarantee to prevent aliasing.

To prevent aliasing, the feature maps need to be band-limited before down-sampling [14]. This band limitation is needed to ensure that after down-sampling no replica of the frequency spectrum overlap (see Figure 3). To guarantee the required band limitation for sub-sampling with a factor of two to $N/2$ where N is the size of the original signal, one has to remove (reduce to zero) all frequency components above $N/2$.

Spatial Filtering based Approaches [50,51] propose to apply approximated Gaussian filter kernels to the feature map. This operation is motivated by the fact that an actual Gaussian in the spatial domain corresponds to a Gaussian in the frequency (e.g. Fourier) domain. As the standard deviation of the Gaussian in the spatial domain increases, the standard deviation of its frequency representation decreases. Yet, the Gaussian distribution has infinite support, regardless of its standard deviation, i.e. the function never actually drops to zero. The convolution in the spatial domain corresponds to the point-wise multiplication in the frequency domain.

Therefore, even after convolving a signal with a perfect Gaussian filter with large standard deviation (and infinite support), all frequency components that were $\neq 0$ before the convolution will be afterwards (although smaller in magnitude). Specifically, the convolution with a Gaussian (even in theoretically ideal settings), can reduce the apparent aliasing but some amount of aliasing will always persist. In practice, these ideal settings are not given: Prior works such as [50,51] have to employ approximated Gaussian filters with finite support (usually not larger than 7×7).

FLC Pooling Therefore, FLC pooling operates directly in the frequency domain, where it removes all frequencies that can cause aliases.

This operation in the Fourier domain is called the *ideal low pass filter* and corresponds to a point-wise multiplication of the Fourier transform of the feature maps with a rectangular pulse $H(\hat{m}, \hat{n})$.

$$H(\hat{m}, \hat{n}) = \begin{cases} 1 & \text{for all } \hat{m}, \hat{n} \text{ below M/2 and N/2} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

This trivially guarantees all frequencies above below M/2 and N/2 to be zero.

Could we apply FLC Pooling as Convolution in the Spatial Domain? In the spatial domain, the ideal low pass filter operation from above corresponds to a convolution of the feature maps with the Fourier transform of the rectangular pulse $H(\hat{m}, \hat{n})$ (by the Convolution Theorem, e.g.[14]). The Fourier transform of the rectangle function is

$$sinc(m, n) = \begin{cases} \frac{sin(\sqrt{m^2+n^2})}{\sqrt{m^2+n^2}} & m, n \neq 0 \\ 1 & m, n = 0 \end{cases} \tag{7}$$

However, while the ideal low pass filter in the Fourier domain has finite support, specifically all frequencies above $N/2$ are zero, $sinc(m, n)$ in the spatial domain has infinite support. Hence, we need an infinitely large convolution kernel to apply perfect low pass filtering in the spatial domain. This is obviously not possible in practice. In CNNs the standard kernel size is 3x3 and one hardly applies kernels larger than 7x7 in CNNs.

A.4 Model Confidences

In Table 10, we evaluate the confidence of model predictions. We compare each model's confidence on correctly classified clean examples to its respective confidence on wrongly classified adversarial examples. Ideally, the confidence on the

Table 10. Evaluation for clean and robust accuracy, higher is better, on AutoAttack [9] with our trained models. The models reported by the original authors may have different numbers due to different hyperparameter selection. We report each models confidence on their correct predictions on the clean data (Clean Confidence) and the models confidence on its false predictions due to adversarial perturbations (Perturbation Confidence). The top row reports the baseline without adversarial training.

Method	Clean Acc	AA Acc	Clean Confidence	Perturbation Confidence
Baseline	95.08	0.00	100.00	97.89
FGSM & early stopping [45]	82.88	11.82	90.50	84.26
FGSM & FLC Pooling (Ours)	**84.81**	38.41	**98.84**	70.98
PGD [27]	83.11	40.35	56.58	75.00
Robustness lib [12]	76.37	32.10	95.22	78.91
AWP [46]	82.61	**49.43**	88.83	**37.98**
MART [44]	55.49	8.63	24.44	50.17
TRADES [48]	81.49	46.91	53.94	50.46

adversarial examples should be lower. The results for the different methods show that FLC yields comparably high confidence on correctly classified clean examples with a 20% gap in confidence to wrongly classified adversarial examples. In contrast, the baseline model is highly confident in both cases. Other, even state-of-the-art robustness models have on average lower confidences but are even less confident in their correct predictions on clean examples than on erroneously classified adversarial examples (e.g. MART [44] and PGD [27]). Only the model from [46] has a trade-off preferable over the one from the proposed, FLC model.

A.5 AutoAttack Attack Structure

In the main paper we showed one example of an image optimized by AutoAttack [9] to fool our model and the baseline in Figure 5. In Figure 6, we give more examples for better visualisation and comparison.

A.6 Ablation Study: Additional Frequency Components

In addition to the low frequency components we tested different settings in which we establish a second path through which we aim to add high frequency or the original information. We either add up the feature maps or contacted them. The procedure of how to include a second path is represented in Figure 7. One approach is to execute the standard down-sampling and add it to the FLC pooled feature map. The other is to perform a high pass filter on the feature map and down-sample these feature maps. Afterwards, the FLC pooled feature maps as well as the high pass filtered and down-sampled ones are added. With this ablation, we aim to see if we do lose too much through the aggressive FLC

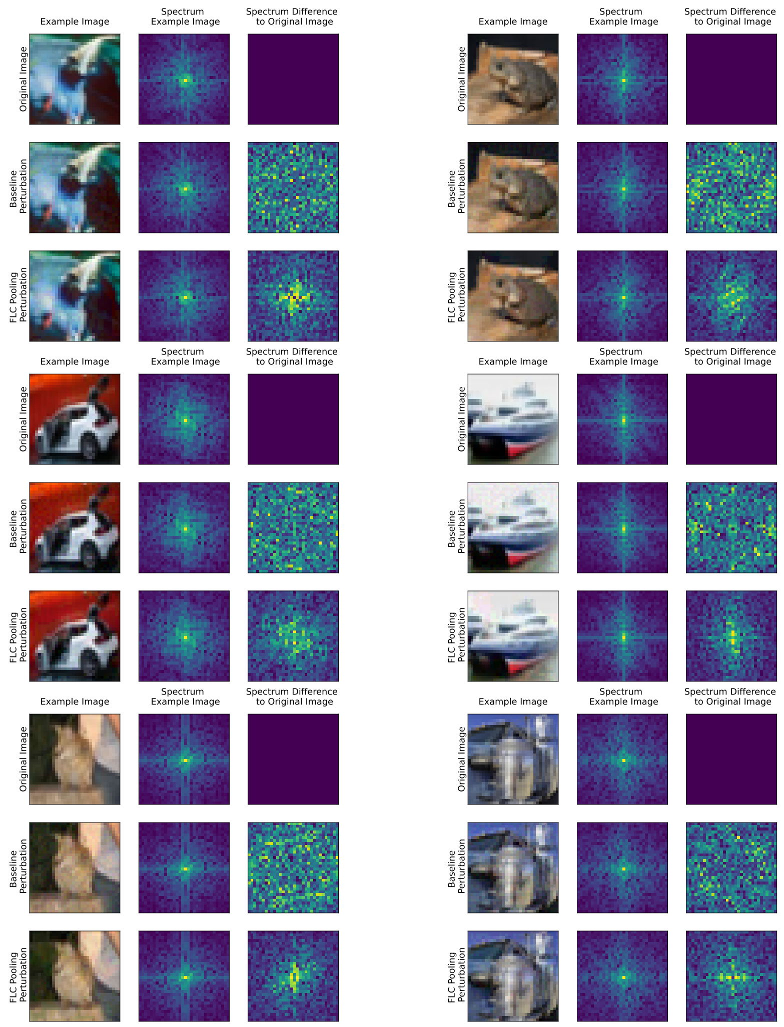

Fig. 6. Spectrum and spectral differences of adversarial perturbations created by AutoAttack with $\epsilon = \frac{8}{255}$ on the baseline model as well as our FLC Pooling. The classes from top left down to the bottom right are: Bird, Frog, Automobile, Ship, Cat and Truck.

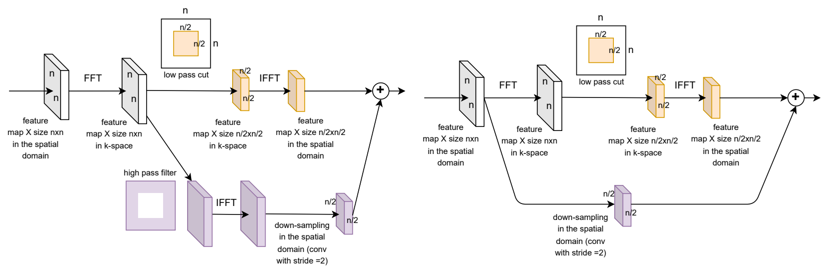
FFT
IFFT
n
n/2
low pass cut
feature map X size nxn in the spatial domain
feature map X size nxn in k-space
feature map X size n/2xn/2 in k-space
feature map X size n/2xn/2 in the spatial domain
high pass filter
down-sampling in the spatial domain (conv with stride =2)

Fig. 7. LC pooling plus, which either includes the original down-sampled signal like it is done traditionally (right) or with the high frequency components filtered by a high pass filter in the Fourier domain and down-sampled in the spatial domain by an identity convolution of stride two (left).

pooling and if we would need additional high frequency information which is discarded through the FLC pooling. Table 11 show that we can gain minor points for the clean but not for the robust accuracy. Hence we did not see any improvement in the robustness and an increase in training time per epoch as well as a minor increase in model size, we will stick to the simple FLC pooling.

Table 11. Accuracies for CIFAR-10 Baseline LowCutPooling plus the original or high freqeuncy part of the featuremaps down-sampled in the spatial domain for FGSM Training. We can see that the additional data does not improve the robust accuracy and gives only minor improvement for the clean accuracy. Due to the additional computations necessary for the high frequency /original part we decided to fully discard them and stick to the pure low frequency cutting.

Method	Clean	PGD L_{inf} $\epsilon = \frac{8}{255}$	Seconds per epoch (avg)	Model size (MB)
FLC pooling	84.81	38.41	34.6 $\pm$ 0.1	42.648
FLC pooling + HighPass pooling	85.38	38.02	45.2 $\pm$ 0.4	42.652
FLC pooling + Original pooling	85.37	38.30	35.4 $\pm$ 0.1	42.652