Hartley Spectral Pooling for Deep Learning

Hao Zhang, Jianwei Ma

Abstract—In most convolution neural networks (CNNs), downsampling hidden layers is adopted for increasing computation efficiency and the receptive field size. Such operation is commonly so-called pooling. Maximation and averaging over sliding windows (*max/average pooling*), and plain downsampling in the form of strided convolution are popular pooling methods. Since the pooling is a lossy procedure, a motivation of our work is to design a new pooling approach for less lossy in the dimensionality reduction. Inspired by the Fourier spectral pooling(FSP) proposed by Rippel et. al. [1], we present the Hartley transform based spectral pooling method in CNNs. Compared with FSP, the proposed spectral pooling avoids the use of complex arithmetic for frequency representation and reduces the computation. Spectral pooling preserves more structure features for network's discriminability than max and average pooling. We empirically show that Hartley spectral pooling gives rise to the convergence of training CNNs on MNIST and CIFAR-10 datasets.

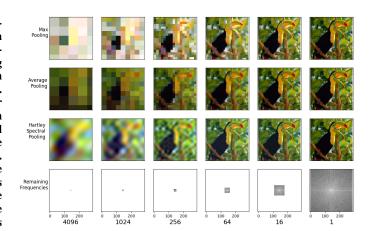*Index Terms*—Deep learning, spectral pooling, Hartley transform

Fig. 1. Downsampling at different scales of dimensionality reduction. Hartley-based spectral pooling project real input onto the Hartley basis and truncates the real frequency representation as desired. This retains significantly more information as well as allows us choose arbitrary output dimension.

I. Introduction

CONVOLUTIONAL neural networks(CNNs) [2]–[4] have been dominant machine learning approach for computer vision, and they also get increasing applications in many other fields. The modern framework of CNNs was established by LeCun et. al. [5] in 1990, with three main components: convolution, pooling, and activation. Pooling is an important component of CNNs. Even before the resuscitation of CNNs, pooling was utilized to extract features to gain dimension-reduced feature vectors and acquire invariance to small transformations of the input in the inpiration of complex cells in animal visual cortex [6].

Pooling is of crucial for reducing computation cost, improving some amount of translation invariance and increasing the receptive field of neural networks. In shallow/mid-sized networks, max or average pooling are most widely used such as in AlexNet [7], VGG [8], and GoogleNet [9]. Deeper networks always prefer strided convolution for architecture-design simplicity [10]. One markable examplar is the ResNet [11]. However, these common poolings sufer from its drawbacks. For example, max pooling implies an amazing by-product of discarding at least 75% of data–the maximum value picked out in each local region only reflects very rough information. Average pooling stretches to the opposite end, resulting in a gradual, constant attenuation of the contribution of individual grid in local region, and ignoring the importance of local structure. These two poolings both sufer from sharp dimensionality reduction and lead to implausible looking results (see the first and second row in Fig.1). Strided convolution may cause aliasing since it simply picks one node in a fixed position in each local region, regarding the significance of its activation [12].

There have been a few attempts to mitigate the harmful effects of max and average pooling, such as a linear combination and extension of them [13], and nonlinear pooling layers [14], [15]. In most common implementations, max or average related pooling layers directly downscale the spatial dimension of feature maps by a scaling factor. L_p pooling [15] provides better generalization than max pooling, with $p = 1$ corresponding to average pooling and $p = \infty$ reducing to max pooling. Yu et. al. [16] proposed mixed pooling which combines max pooling and average pooling, and switches between these two pooling methods randomly. Instead of picking the maximum values within each pooling region, stochastic pooling [17] and S3Pool [18] stochastically pick a node in each pooling region, and the former favors strong activations. In some networks, strided convolutions are also used for pooling. Notably, these pooling methods are all of integer stride larger than 1. To abate the loss of information caused by the dramatic dimension reduction, fractional max-pooling [19] randomly generates pooling region with stride 1 or 2 to achieve pooling stride of less than 2. Recently, Saeedan et. al. [20] propose the detail-preserving pooling aiming at preserving low-level details and filling the gap between max pooling and average pooling [21]. We refer these mentioned pooling methods as *spatial pooling* because they perform pooling in spatial domain.

Recently Rippel et.al. [1] proposed the *Fourier spectral pooling*. It downsamples the feature maps in frequency domain using low-pass filtering. Specifically, it selects pooling region in Fourier frequency domain by extracting low frequency subset(*i.e. truncation*). This approach can alleviates those

Harbin Institute of Technology, China (hao.zhang.hit@gmail.com)
Harbin Institute of Technology, China (jma@hit.edu.cn)

issues that exist in spatial pooling strategies as mentioned above, and it shows good information preserving ability. However, it introduces the processing of imaginary, which should be carefully treated in real CNNs. Besides, the frequency truncation may destroy the *conjugate symmetry* of the Fourier frequency representation of the real input. Rippel et. al. suggest *RemoveRedundancy* and *RecoverMap* algrithms to make sure the downsampled spatial approximation be real(see supplementray of [1]). But they are demonstrated to be time consuming. Following the work in [1], [22] proposes the discrete cosine transform(DCT) pooling layer. Although the DCT pooling layer uses real numbers for frequency representation, it doubles the number of operations compared with FFT pooling layer.

Inspired by the work of [1], we present the Hartley transform-based spectral pooling in this paper. Hartley transform only use real numbers for real input and it has fast discrete algorithm with operations almost the same as Fourier transform. So the presented Hartley spectral pooling layer avoids the use of complex arithmetic and it could be plugged in modern CNNs effortlessly. Moreover, it reduces the calculated amount by dropping out two auxlary steps of the algorithms in [1]. Meanwhile, we provide a useful observation that preserving more information could contribute to the convergence of training modern CNNs.

II. Method

A. Hartley transform

The Hartley transform is an integral transform closely related to the Fourier transform [23], [24]. It has some advantages over the Fourier transform in the analysis of real signals as it avoids the use of complex arithmetic. These advantages attracts researchers to conduct plenty of researches on its application and fast implementation during the 1990s [25]–[28]. In two dimensions, we denote $H(u_1, u_2)$ and $F(u_1, u_2)$ as the Hartley transform and Fourier transform of $f(x_1, x_2)$ respectively, as defined in [27].

$$
\begin{aligned}
H(u_1, u_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x_1, x_2)[cos(2\pi(u_1 x_1 + u_2 x_2)) \\
+ sin(2\pi(u_1 x_1 + u_2 x_2))]dx_1 dx_2
\end{aligned}
\tag{1}
$$

and the inverse transform by

$$
\begin{aligned}
f(x_1, x_2) = \int_{-\infty}^{\infty} H(u_1, u_2)[cos(2\pi(u_1 x_1 + u_2 x_2)) \\
+ sin(2\pi(u_1 x_1 + u_2 x_2))]du_1 du_2
\end{aligned}
\tag{2}
$$

It can be easily derived the relationship between these two transforms, giving[1]

$$
H(u_1, u_2) = \frac{1+i}{2} F(u_1, u_2) + \frac{1-i}{2} F(-u_1, -u_2)
\tag{3}
$$

In the case that function $f(x_1, x_2)$ is real, its Fourier transform is Hermetian, i.e.

$$
F(-u_1, -u_2) = F^*(u_1, u_2)
\tag{4}
$$

[1]Refer to [27] for details.

so the Fourier transform processes some redundancy on the real u_1-u_2 plane, which results in conjugate-symmetry constriction aiming at reducing training parameters in the frequency domain neural networks [1], [29].

The Hermetian property above shows that the Hartley transform of a real function can be written as

$$
H(u_1, u_2) = \mathcal{R}\{F(u_1, u_2)\} - \mathcal{I}\{F(u_1, u_2)\}
\tag{5}
$$

where $\mathcal{R}\{\cdot\}$ and $\mathcal{I}\{\cdot\}$ denote the real and imaginary parts respectively. Note that giving the definition above, the Hartley transform $\mathcal{H}$ is a real linear operator. It is symmetric as well as an involution and a unitary operator

$$
f = \mathcal{H}\{\mathcal{H}f\}
\tag{6}
$$

To Hartley transform, imaginary part and conjuate symmetry no more need to be concerned for real inputs such as images.

a) Differentiation.: Here we discuss how to propagate the gradient through the Hartley transform, which will be used in CNNs. Define $x \in \mathbb{R}^{M \times N}$ and $y = \mathcal{H}(x)$ to be the input and output of a discrete Hartlay transform (DHT) respectively, and $L : \mathbb{R}^{M \times N} \to \mathbb{R}$ a real-valued loss function applied to y. Since the DHT is a linear operator, its gradient is simply the transform matrix itself. During back-propagation, this gradient by the unitarity of DHT, corresponds to the application of Hartley transform:

$$
\frac{\partial L}{\partial x} = \mathcal{H}(\frac{\partial L}{\partial y})
\tag{7}
$$

B. Hartley-based spectral pooling

Spectral pooling preserves considerably more information and structure for the same number of parameters [1](see the third row of Fig. 1) because the frequency domain provides a sparse basis for inputs with spatial structure. The spectrum power of typical input is heavily concentrate in lower frequencies while higher frequencies mainly tend to encode noise [30]. This non-uniformity of spectrum power enables the removal of high frequencies do minimal damage of input information.

To avoid the time consuming *RemoveRedundancy* and *RecoverMap* steps in [1], we suggest the Hartley transform-based spectral pooling. This spectral pooling is straightforward to understand and much easier to implement. Assume we have an input $x \in \mathbb{R}^{H \times W}$, and some desired output map dimensionality $h \times w$. First, we compute the DHT of the input into the frequency domain as $y = \mathcal{H}(x) \in \mathbb{R}^{H \times W}$, and shift the DC component of the input to the center of the domain. Then we crop the frequency representation by maintaining only the central $h \times w$ submatrix of frequencies, denoted as $\hat{y} \in \mathbb{R}^{h \times w}$. Finally, we take the DHT again as $\hat{x} = \mathcal{H}(\hat{y})$ to map frequency approximation back into spatial domain, obtaining the downsampled spatial approximation. The back-propagation of this spectral pooling is similar to its forward-propagation since Hartley transform is differentiable.

Those steps in both forward and backward propagation of this spectral pooling are listed in Algorithm 1 and 2, respectively. These algorithms simplify the spectral pooling by Fourier transform, profited from that the Hartley transform of a real function is real rather than complex. Figure 1 demonstrates
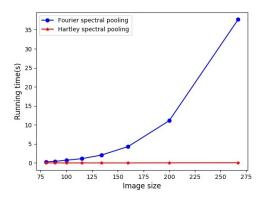
Fig. 2. Comparison between proposed method and Fourier spectral pooling [1] in sum of forward- and backward-propagation running time.

the effect of this spectral pooling for various dimensionality reduction factors.

Algorithm 1 Hartley Spectral pooling

Input: Map $x \in \mathbb{R}^{H \times W}$, output size $h \times w$
Output: Pooled map $\hat{x} \in \mathbb{R}^{h \times w}$
  1: $y \leftarrow \mathcal{H}(x)$
  2: $\hat{y} \leftarrow CropSpectrum(y, h \times w)$
  3: $\hat{x} \leftarrow \mathcal{H}(\hat{y})$

Algorithm 2 Hartley Spectral pooling back-propagation

Input: Gradients w.r.t. output $\frac{\partial L}{\partial \hat{x}}$
Output: Gradients w.r.t. input $\frac{\partial L}{\partial x}$
  1: $\hat{z} \leftarrow \mathcal{H}(\frac{\partial L}{\partial \hat{x}})$
  2: $z \leftarrow PadSpectrum(\hat{z}, H \times W)$
  3: $\frac{\partial L}{\partial x} \leftarrow \mathcal{H}(z)$

C. Comparison.

We compare the efficiency of the proposed pooling layer with that of the Fourier pooling layer in [1] here. Both spectral pooling methods are implemented in plain python with package *numpy*. As shown in Fig.2, the running time of Fourier spectral pooling increases rapidly when image size becomes larger. We claim that this is caused by the time-consuming steps *RemoveRedundancy* and *RecoverMap* when backpropagating in Fourier spectral pooling algorithm(refer to [1]).

III. EXPERIMENTS

We verify the effectiveness of the Hartley spectral pooling through image classification task on MNIST and CIFAR-10 datasets. The trained networks include a toy CNN model (Table I), ResNet-16 and ResNet-20 [11]. The toy network uses max pooling while ResNets employs strided convolutions for downscaling. In these experiments, spectral pooling shows favorable results. Our implementation is based on PyTorch [31].

A. Datasets and configureations

a) MNIST: The MNIST database [32] is a large database of handwritten digits that is commonly used for training various convolutional neural networks. This dataset contains 60000 training examples and 10000 testing examples. All these examples are gray images in size of 28×28. In our experiment, we do not perform any preprocessing or augmentation on this dataset. Adam optimization algorithm [33] is used in all experiments of classification on MNIST, with hyper-parameter $\beta_1 = 0.9$ and $\beta_2 = 0.999$ configured as suggested and a mini-batch size of 100. The initial learning rate is set to 0.001 and is divided by 10 every 5 epochs. Regularization is aborted in these experiments.

b) CIFAR-10: The CIFAR-10 dataset consists of 60000 colored natural images in 10 classes, with 6000 images per class holding 5000 for training and 1000 for testing. Each image is in size of 32×32. For data augmentation we follow the practice in [11], doing horizontal flips and randomly sampling 32×32 crops from image padded by 4pixels on each side. The normalization is performed in data preprocessing by using the channel means and standard deviations. In experiments on this dataset, we use stochastic gradient method with Nesterov momentum and cross-entropy loss. The initial learning rate is started with 0.1, and is multiplied by 0.1 at 80 and 120 epochs. Weight decay is configured to 10^{-4} and momentum to 0.9 without dampening. Mini-batch size is set to 128.

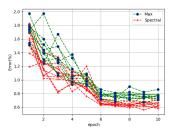*B. Classification results on MNIST*

a) Shallow network: We first use a network in which max pooling is plugged (see Table I). Each convolution layer is followed by a batch normalization and a ReLU nonlinearity. We test Hartley-based spectral pooling by replacing max pooling in this architecture. The train procedure lasts 10 epochs and is repeated 10 times. The classification error on testing set in each epoch is shown in Fig. 3.

Comparing to max pooling, spectral pooling shows strong results, yielding more than 15% reduction on classification error observed in our experiment. As all things equal except the pooling layers, we claim that this improvement is achieved by the better information-preserved ability of Hartley-based spectral pooling.

TABLE I
THE TOY CNN MODEL FOR CLASSIFICATION ON MNIST.

layer name	output size	Max Pooling model	Spectral Pooling model
conv1	28×28	5×5, 16	
pool1	14×14	Max, stride=2	Spectral, 14×14
conv2	14×14	5×5, 32	
pool2	7×7	Max, stride=2	Spectral, 7×7
fc	1×1	10-d fc	

b) ResNet: Next, we use ResNet-20 [11], a deeper network. We don't use more deeper residual net such as ResNet-110 because much more parameters in this architecture may give rise to overfitting. ResNet is composed of numerous residual building blocks. It is a much modern convolutional
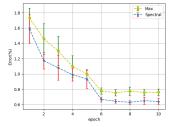
Fig. 3. Classification error on MNIST testing set by networks in Table I. (*Left*) Classification error curves in 10 runs. (*Right*) *mean±std* of ten runs, with best error $0.605\%(0.63 \pm 0.025)$, $0.719\%(0.759 \pm 0.040)$ for spectral pooling and max pooling respectively.
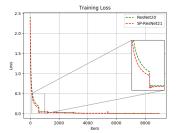
neural network which does not explicitly use pooling layers but instead embeds a stride-2 convolution layer inside some of building blocks for the effectuation of downsampling. For our experiments we replace the stride-2 convolutional layer by spectral pooling and remove the skip connection [9], [34] in those downscaling blocks. Besides, we set the output size of serial spectral pooling layers linearly decreased(reducing 8 in each axis after a spectral pooling layer). The manually tuned network architecture is depicted in Table II[2]. We leave the global average pooling untouched. Each experiment is repeated 5 times and the best result is reported in Table III.

Spectral pooling ResNet21 performs a little better than its ResNet20 counterpart, surprisingly, even though it holds parameters almost in half that of ResNet20. Further, we illustrate one among the five training procedures in Fig. 4. It is observed that the SP-ResNet21 converges faster than ResNet20 (left panel) and performs better in classification (right panel). This indicates the spectral pooling ease the optimization by providing faster convergence at the early stage.

TABLE II
THE ARCHITECTURE OF 21-LAYER SP-RESNET FOR MNIST.

block name	output size	21-layer
conv1	28×28	3×3, 16
conv2	28×28	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 2$
downsample1	20×20	$\begin{bmatrix} SP_{20 \times 20} \\ 3 \times 3, 32 \end{bmatrix}$
conv3	20×20	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 2$
downsample2	12×12	$\begin{bmatrix} SP_{12 \times 12} \\ 3 \times 3, 32 \end{bmatrix}$
conv4	8×8	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 2$
downsample3	4×4	$\begin{bmatrix} SP_{4 \times 4} \\ 3 \times 3, 64 \end{bmatrix}$
conv5	4×4	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
	1×1	avg pool, 10-d fc

[2] Building blocks are shown in brackets, with the numbers of block stacked. The SP stands for the spectral pooling layer, and the footnote $n \times n$ indicates the output size.
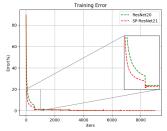

Fig. 4. Training on **MNIST** by ResNet20 and SP-ResNet21. Left panel: training loss; Right panel: classification error on training set.

TABLE III
CLASSIFICATION ERROR ON **MNIST** TESTING SET AFTER TRAINING WITHOUT DATA AUGMENTATION.

method	# params	error(%)
ResNet20	0.27M	0.36
SP-ResNet21	**0.15M**	**0.32**

C. Classification results on CIFAR-10

For the network training on CIFAR-10, the architecture is similar to SP-ResNet21 that trains on MNIST. We set the sizes of output of spectral pooling layers to be 24×24, 16×16, 8×8 sequentially. In experiments we use ResNet16 as a counterpart, since it contains almost the same amount of parameters as SP-ResNet21(see Table IV). The result of spectral pooling plugged network, SP-ResNet21, outperforms that of ResNet16 by nearly 0.25%.

TABLE IV
CLASSIFICATION ERROR ON **CIFAR-10** TESTING SET.

method	# params	error(%)
ResNet16	0.18M	8.87
SP-ResNet21	**0.15M**	**8.63**

IV. CONCLUSION

We present a full real-valued spectral pooling method in this paper. It reduces the computational complexity compared to that of previous spectral pooling work. Based on this approach, we provide some results on several commonly used benchmark dataset (including MNIST, CIFAR-10) by training modified residual nets. We also investigate the contribution of this spectral pooling method to the convergence of training neural networks. We demonstrate spectral pooling yields higher classification accuracy than its counterparts max pooling. And in residual net, it improves the convergence rate in training convolutional neural networks by expanding the space of spatial dimensionality of downsamplings. Also, it is easy to implement and fast to compute during training time.

REFERENCES

[1] O. Rippel, J. Snoek, and R. P. Adams, "Spectral representations for convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2449–2457.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[5] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems (NIPS)*, 1990, pp. 396–404.

[6] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." CVPR, 2015.

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

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[12] J. G. Proakis and D. G. Manolakis, *Digital signal processing*. Pearson Education, 2013.

[13] C.-Y. Lee, P. W. Gallagher, and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree," in *Artificial Intelligence and Statistics*, 2016, pp. 464–472.

[14] C. Gulcehre, K. Cho, R. Pascanu, and Y. Bengio, "Learned-norm pooling for deep feedforward and recurrent neural networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2014, pp. 530–546.

[15] J. Bruna, A. Szlam, and Y. LeCun, "Signal recovery from pooling representations," *arXiv preprint arXiv:1311.4025*, 2013.

[16] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International Conference on Rough Sets and Knowledge Technology*. Springer, 2014, pp. 364–375.

[17] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013.

[18] S. Zhai, H. Wu, A. Kumar, Y. Cheng, Y. Lu, Z. Zhang, and R. Feris, "S3pool: Pooling with stochastic spatial sampling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4970–4978.

[19] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[20] F. Saeedan, N. Weber, M. Goesele, and S. Roth, "Detail-preserving pooling in deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9108–9116.

[21] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 111–118.

[22] J. S. Smith and B. M. Wilamowski, "Discrete cosine transform spectral pooling layers for convolutional neural networks," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2018, pp. 235–246.

[23] R. V. Hartley, "A more symmetrical fourier analysis applied to transmission problems," *Proceedings of the IRE*, vol. 30, no. 3, pp. 144–150, 1942.

[24] R. N. Bracewell, *The Hartley transform*. Oxford University Press, Inc., 1986.

[25] ——, "Discrete hartley transform," *JOSA*, vol. 73, no. 12, pp. 1832–1835, 1983.

[26] ——, "The fast hartley transform," *Proceedings of the IEEE*, vol. 72, no. 8, pp. 1010–1018, 1984.

[27] R. Millane, "Analytic properties of the hartley transform and their implications," *Proceedings of the IEEE*, vol. 82, no. 3, pp. 413–428, 1994.

[28] J. Agbinya, "Fast interpolation algorithm using fast hartley transform," *Proceedings of the IEEE*, vol. 75, no. 4, pp. 523–524, 1987.

[29] H. Pratt, B. Williams, F. Coenen, and Y. Zheng, "Fcnn: Fourier convolutional neural networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 786–798.

[30] A. Torralba and A. Oliva, "Statistics of natural image categories," *Network: Computation in Neural Systems*, vol. 14, no. 3, pp. 391–412, 2003.

[31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[32] E. Kussul and T. Baidyk, "Improved method of handwritten digit recognition tested on mnist database," *Image and Vision Computing*, vol. 22, no. 12, pp. 971–981, 2004.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.