Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Convolutional Neural Networks based on Discrete Cosine Transform with Applications in Computer Vision

Matej Uličný

April 30, 2021

A Doctoral Thesis submitted in partial fulfilment
of the requirements for the degree of
PhD (Computer Science)

Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Signed: _____ Date: _____

Matej Uličný

Abstract

Convolutional neural networks (CNNs) have become a paradigm for designing vision based intelligent systems. These models are controlled by a vast amount of parameters, which are learned thanks to the availability of annotated datasets. Image data is available in multiple formats including JPEG that uses Discrete Cosine Transform (DCT) coefficients to efficiently encode and compress visual information. We first propose to use directly these DCT coefficients of the JPEG images as input of CNN models, removing the need to completely decode JPEG format before applying CNNs.

Furthermore, we propose to use DCT basis functions to express convolutional filters in any layer of a CNN and we show that this provides an advantageous regularization during the training process. We show that expressing weights within DCT bases can increase performance and speed up the training. We improve several popular models on standard benchmarks such as ImageNet classification accuracy by 1%, MS COCO object detection average precision by 1% and Pascal VOC semantic segmentation IoU score by 1.1%. We propose to exploit properties of natural images by restricting the set of basis functions used during the training. Suppressing the low-frequency component on the first layer can make models insensitive to illumination effects. High-frequency truncation on multiple layers can in turn add stability and efficiently compress a model without any significant loss in accuracy. Using the DCT bases provides a prior that reduces overfitting, specially when compression is applied, and helps with generalization when fewer samples are available.

Lastly, the standard DCT-based compression is modified and extended to be applicable to any weight tensor used in neural networks. We propose to reshape a tensor into a 2-dimensional matrix and reorder its rows based on pairwise distances between the columns in order to make the matrix more coherent. The reordered matrix is transformed via 1-dimensional DCT and high frequencies are truncated. We further correct the scale and bias parameters of batch normalization layers to take into account compression of the preceding layers. Promising results are achieved even without a need for model fine-tuning. The use of a short fine-tuning of one epoch can lead to models with 3-times fewer parameters without a loss in accuracy.

Acknowledgements

First and foremost I would like to express my thanks to my supervisor Professor Rozenn Dahyot for her support and guidance throughout my PhD years. She gave me freedom to explore research areas of my interest and yet provided valuable insights. Secondly, great thanks to Professor Vladimir Krylov for his invaluable help and advice. I would like to thank the ADAPT Centre for providing me with the scholarship funding and an environment for developing research engagement. Many thanks to my colleagues Jing and Chao for compelling discussions over the lunch and coffee breaks. Gratitude to the rest of the team and friends who, thanks to day to day interactions, made the whole process more pleasant: Hana, Fatma, Julie, Reem, Sam, Anwesha, Iman, Lisanne, Jessica, Aisling and the rest of Stack B. Lastly, I would like to thank my whole family, whose support and encouragement has led me on my path.

Contents

List of Figures

List of Tables

Abbreviations

ANN	Artificial Neural Network
AP	Average Precision
BN	Batch Normalization
bs	Batch size
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DNN	Deep Neural Network
DFT	Discrete Fourier Transform
FB	Fourier-Bessel
FFT	Fast Fourier Transform
GD	Gaussian Derivative
Harm	Harmonic
IoU	Intersection over Union
lr	Learning rate
OCR	Optical Character Recognition
PCA	Principal Component Analysis
RBF	Radial Basis Function
RGB	Red Green Blue color space
YCbCr	YCbCr color space

Chapter 1

Introduction

In the beginning of the 21st century, the wide spread of low-cost digital cameras combined with easy access to the internet triggered a boom in online shared visual media content. Data in visual form became the major part of web-related services, yet semantic information contained in images is challenging to analyze. Computer vision incorporates a multitude of tasks, examples of such are classification [12] and detection [13] of objects in images, or segmenting [14] parts of images that have the same meaning (belong to the same object or type of object). Some techniques try to detect entity boundaries [15] or keypoints [16] that can be used for matching parts of images or for human pose estimation. Numerous approaches are focused on enhancing quality of images [17] or suppressing image artifacts [18]. Current trends in analyzing image data relies on data-driven models that learn to extract features representing concepts in images and use these concepts to solve a given problem. The objective of this thesis is to improve the feature extraction effectiveness and efficiency.

1.1 Image recognition pipeline

A vast majority of techniques used in computer vision problems are data dependent. The first step to solve a certain task is to collect a dataset that will be used to tune parameters of a selected model. The dataset can be either created synthetically, or be collected by digital cameras. In supervised learning setting a machine-learning model needs a supervision. It comes as a target, which the model is trying to predict. This target has a form of an image annotation and is usually created by humans. It is often the most costly and tedious activity, specially for large datasets. To predict the target directly from image pixels is often a not feasible task due to characteristics of the investigated problems themselves. For example, the problem of object classification is translation invariant. It would require from a classifier to learn how to recognize an object when it is located on an arbitrary place in the picture. Therefore, a common practise is to collect invariant local features from an image and map them to higher level features that are finally used to train a classifier or

regressor that predicts the target [19].

1.2 Neural networks

Artificial neural network merges those 3 stages into one model that jointly extracts features at different levels of hierarchy and performs classification simultaneously. The network is composed of layers, which sequentially transform the data vector $\mathbf{x} \in \mathbb{R}^n$. A layer consists of linear transformation via matrix multiplication by weight matrix W, bias shift by $\mathbf{b}$ and point-wise nonlinear function ϕ. Output of the layer is another vector $\mathbf{y} \in \mathbb{R}^m$, often referred to as hidden representation:

$$\mathbf{y} = \phi\left(W\mathbf{x} + \mathbf{b}\right). \tag{1.1}$$

Elements of the input and hidden vectors can be interpreted as neurons. Such layer is called a fully-connected layer because it connects each hidden neuron to all input neurons through elements of matrix W. A sequence of these layers constitutes a network. Parameters W and $\mathbf{b}$ are learned during the training process. The non-linear function in each layer is necessary should the network model non-linear input-output relations.

1.2.1 Local learning

A neural network based on fully-connected layers deals only with fixed-size input data. With increased dimensionality of the data vector $\mathbf{x}$ the weight matrix W grows as well. Linear scaling of input and hidden representation dimensionality simultaneously scales the weight matrix quadratically and becomes uncontrollable for large images. Another problem of fully-connected layers is that a certain feature that is detected in one part of an image might not be detected in another part, as those pixels are processed through different weights. This counter-intuitive property is addressed through local learning and weight sharing. The Convolutional Neural Network (CNN) forms weights into filters that extract local correlations at all pixel locations in the image through convolution. An image $\mathbf{x}$ is shaped into 2-dimensional grid. Discrete convolution of an image with filter $\mathbf{w} \in \mathbb{R}^{k \times l}$ is defined as follows:

$$\mathbf{w} * x\left(u, v\right) = \sum_{i=-k/2}^{k/2} \sum_{j=-l/2}^{l/2} w\left(i, j\right) x\left(u - i, v - j\right). \tag{1.2}$$

If a feature occurs at multiple locations in an image, the filter detects all its occurrences. Weight sharing decouples size of the weights from the data size and allows the use of high-resolution images.

1.3 Bases for image data

A high-dimensional input vector $\mathbf{x} \in \mathbb{R}^{n \times m}$ corresponding to an image is a collection of coordinates within certain bases. Conceptually, images are defined within standard bases expressed by a set of orthogonal column unit vectors $\mathbf{e} \in \mathbb{N}^n$1.3.

$$\mathbf{e_i}[u] = \begin{cases} 1 & \text{if } i = u \\ 0 & \text{otherwise.} \end{cases} \tag{1.3}$$

Examples of $\mathbf{e_i} \in \mathbb{N}^3$ are:

$$\mathbf{e_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \mathbf{e_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \mathbf{e_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{1.4}$$

A pixel coordinate is expressed via row and column coordinates $\mathbf{e_i}\mathbf{e_j}^T$ and constitutes the standard bases. Image is a collection of pixel intensities $x_{i,j}$ within those bases 1.5.

$$\mathbf{x} = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} \mathbf{e_i}\mathbf{e_j}^T \tag{1.5}$$

Filters used to convolve these images are defined in space spanned by the same bases. Standard bases however do not have smooth transitions that are often characteristic for signals in natural images. By these we consider images of natural world captured by digital cameras [20]. More compact image representation can be found in compression formats such as JPEG [21]. The compression scheme uses orthogonal Discrete Cosine Transform (DCT, see Section 2.2.2) [22] that maps image patches into complete basis set formed by cosine functions. These functions oscillate at different frequencies. Natural images typically have small coefficients at high-frequency bases. Human perception is also insensitive to details imposed by them [23]. Energy of such image, initially uniformly distributed between standard bases, becomes concentrated into a few DCT bases.

Roots of the DCT transform can be found in Fourier analysis, see the timeline in Figure 1.1. The DCT has been first introduced by Ahmed et al. in 1974 and was shown to be comparable to optimal transforms in multiple applications [22]. On account of its properties it was integrated in JPEG image encoding [21]. DCT has been used to pre-process inputs for classification tasks with RBF neural networks [24] or convolutional networks [25].

In Chapter 3 we propose to adjust CNN filter set that processes the input data to be applicable to inputs consisting of DCT patches. By adjusting the striding parameter accordingly we enforce the first layer parameters to be learned in spectral domain. We have shown that learning from spectral data is beneficial for shallow models. By reading DCT coefficients

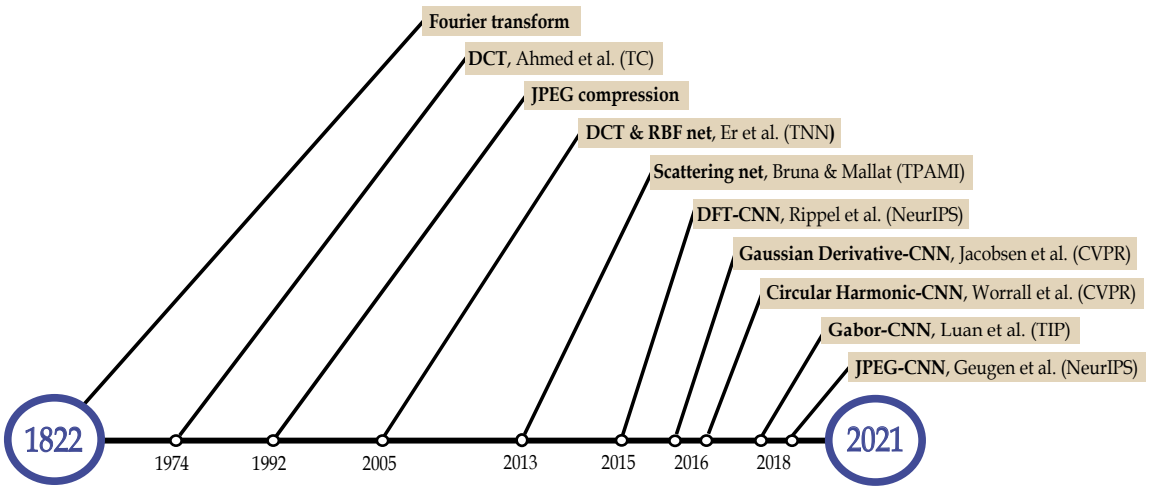


Figure 1.1: Timeline of landmark related works to this thesis.

directly from JPEG images we can avoid unnecessary inverse DCT operation to fully decompress the images. Such approach can also lead to savings in processing memory due skipping convolutions with high-dimensional feature maps.

1.4 Harmonic networks

DCT has proven to provide a suitable basis for natural signals [21]. Filters in CNNs trained on natural images tend to be smooth [26] and convolving natural images with such filters generates features that often preserve some characteristics of natural images. The second research question investigates benefits of representing all convolutional filters in terms of DCT bases. To replace convolutional layers we have proposed harmonic blocks that extract local spectrum of image features through DCT and infer new features by linear combination of the extracted coefficients.

A well established example of deep convolutional network that uses pre-defined filters is the Scattering network [27]. Similar to our proposal, Rippel et al. has represented filter weights in Fourier domain using DFT [28]. In literature, a few recognized works have focused on learnable filter composition from non-standard bases, among them Gaussian derivatives [26], or Circular Harmonics [29]. An analogous approach is to modulate a set of learnable filters by a fixed filter bank, e.g. by Gabor filters [30].

The Chapter 4 describes and analyses our proposed harmonic network. Firstly, representing convolutional filters within the DCT bases has a regularization effect that makes the optimization process easier. Secondly, by truncation of parameters responsible for high frequencies we can efficiently compress spatial filters and decrease parametric complexity and memory footprint of convolutional networks. We have also demonstrated conversion of filters in already trained models from standard bases to DCT bases without a loss of accuracy. This way a compression or further regularization can be applied to any existing

model.

1.5 Channel dimension compression

Many recent CNN architectures sparsify connections and move vast majority of weights into resampling layers implemented through 1×1 convolution [31]. Compressing only filters with spatial extent would have negligible effect on the total parameter count. In Chapter 5 we devise an approach to compress weights of all convolutional and fully-connected layers. A higher dimensional tensor is reshaped to into 2-dimensional matrix. Afterwards, 1-dimensional DCT is performed along one of the dimensions. To transport the energy into lower frequencies, we introduce coherence by reordering the reshaped tensor along this dimension. This is an essential part that prevents the high-frequency truncation from decimating the weights. The decompression is done in an online fashion, where inverse DCT, permutation and reshaping is used to obtain the original weights. Compression rates of up to $4\times$ are obtained with minimum or no loss of classification accuracy with only a modest amount of retraining. For severely compressed models, we suggest a data-free scale and shift parameter correction, based on the proportion of norms of the approximated and the original weight tensors.

1.6 List of Publications

A portion of this work has been published in following articles:

Matej Ulicny, and Rozenn Dahyot. "On using cnn with dct based image data." In *Proceedings of the 19th Irish Machine Vision and Image Processing conference (IMVIP).* 2017.

Matej Ulicny, Vladimir A. Krylov, and Rozenn Dahyot. "Harmonic Networks for Image Classification." In *British Machine Vision Conference (BMVC).* 2019.

Matej Ulicny, Vladimir A. Krylov, and Rozenn Dahyot. "Harmonic networks with limited training samples." In *27th European Signal Processing Conference (EUSIPCO), pp. 1-5. IEEE.* 2019.

Matej Ulicny, Vladimir A. Krylov, and Rozenn Dahyot. "Tensor reordering for CNN compression." In *International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE.* 2021.

1.7 Outline

In this thesis we want to verify the hypothesis that filters in CNNs can be efficiently represented in the frequency domain. We will show that such representation helps in designing

compact CNN models by keeping only the relevant frequencies. Our idea is validated experimentally both on natural and synthetic images. We design a CNN layers in terms of DCT bases and evaluate its performance with and without band-pass compression on a set of image processing tasks.

In the next Chapter, we introduce theoretical background of concepts used in this thesis, namely artificial neural networks, transformation methods, JPEG compression, and we will also review the related work.

Chapter 2

Background

2.1 Artificial Neural Network

Artificial neural network (ANN), also called multilayer perceptron, is a data-driven learning algorithm inspired by biological nervous system. Its origins are dated to 1943, a study of mathematical representations of information processing in biological systems by McCulloch and Pitts [32]. Rosenblatt developed a hypothetical nervous system called perceptron, a two layered network model that used only addition and subtraction (1958) [33]. The neural network can be represented by a directed acyclic graph, embodied of several connected layers (Figure 2.1), each consisting of nodes (or neurons). Such model ordinarily performs weighted summation of inputs followed by a nonlinear scalar transformation. In contrast with shallow models, architectures referred to as "deep" perform multiple levels of such transformations (noted as layers in the computational graph). Weights of the model are found via optimization. Output of each transformation (layer) composes a new feature set representing the input. With increasing depth of the model each following layer produces more abstract features (e.g. from shapes to objects). Neural networks were proved to be adequate function approximators [34, 35].

2.1.1 Optimization

Quality of a prediction is measured by a loss function, expressing how much the prediction differs from the expected value [36]. In classification problems, loss is often measured by Shannon cross-entropy [37] between one-hot class target vector y of length m and posterior probabilities p predicted by the network as

$$\mathcal{L} = -\sum_{i}^{m} y_i \log p_i. \tag{2.1}$$

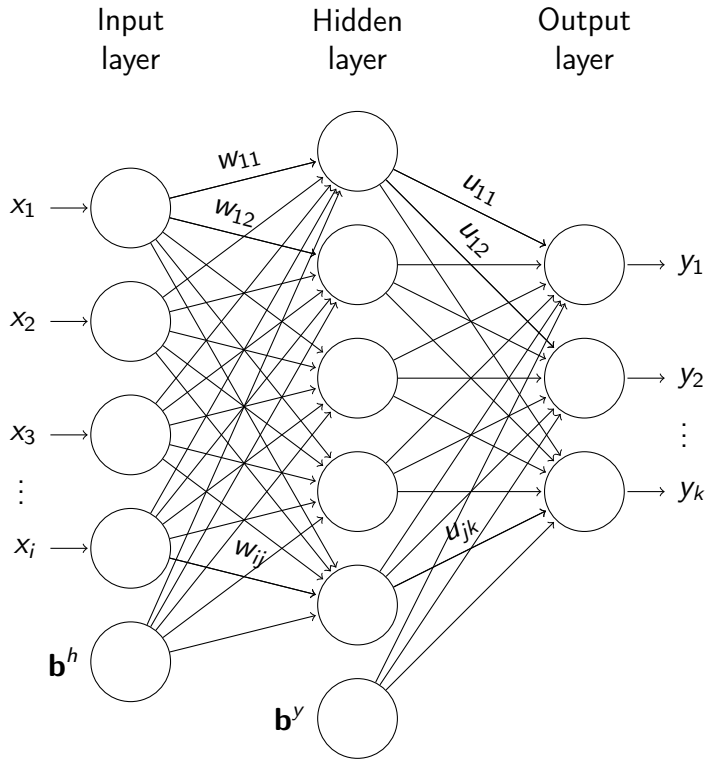


Figure 2.1: A neural network model consisting of input, hidden and output layer, mapping input $\mathbf{x}$ to output $\mathbf{y}$. Inputs of the hidden and the output layers are weighted by weights w_{ij}, u_{jk} and aggregated in individual neurons. Biases $\mathbf{b}^h$ and $\mathbf{b}^y$ are added to the aggregated values at each neuron.

For regression problems the error is usually measured by the $L2$ loss:

$$\mathcal{L} = \sum_{i}^{m} (y_i - p_i)^2 \tag{2.2}$$

Backpropagation

A model prediction is improved by updating its parameters with respect to the gradients calculated by backpropagation of errors estimated by a loss function [38]. The gradients $\nabla_{w_{kj}}$ (2.4) are obtained by the chain rule as partial derivatives of the loss function $\mathcal{L}$ with respect to the weights of the layer w_{kj} given input of the node $\mathbf{x}$ and activation function ϕ.

$$a_j = \phi\left(\sum_{k=1}^{n} w_{kj} x_k\right) \tag{2.3}$$

$$\nabla_{w_{kj}} = \frac{\partial \mathcal{L}}{\partial w_{kj}} = \frac{\partial \mathcal{L}}{\partial a_j} \frac{\partial a_j}{\partial w_{kj}} \tag{2.4}$$

Gradient descent

Gradient descent (2.5) is a first order approximation method that optimizes a function by approaching a local minima by updating parameters w in the direction of the negative

gradient of the loss function with respect to the model parameters. Updates to the w are controlled by a scalar step size referred to as the learning rate η, usually in range $(0, 1)$. In vanilla batch gradient descent, a single parameter update iteration requires gradients for all input instances to be computed. One step is thus computationally expensive and the method tends to fall into a local minima. The Stochastic Gradient Descent (SGD) calculates gradients only for a small randomly sampled subset of the train set to perform a parameter update. SGD has a slower convergence rate than batch gradient descent, but a single parameter update is much less computationally expensive and the optimization procedure tends to avoid poor local minimas. Due to its stochasticity, a wrong choice of starting observations may cause the algorithm to move further from a good solution, thus make the converge problematic.

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \mathcal{L}(w^{(t)}, \mathbf{x}, \mathbf{y}) \tag{2.5}$$

A weighted combination of the current parameter update with a fraction of the previous weight adjustment μ (2.6) is often used to enhance the convergence speed of SGD. Moreover, this technique known in literature as the momentum [39] also prevents the optimization from converging to a poor local minima.

$$\begin{aligned} \Delta w^{t+1} &= \mu \Delta w^{(t)} + \eta \nabla_w \mathcal{L}(w^{(t)}, \mathbf{x}, \mathbf{y}) \\ w^{(t+1)} &= w^{(t)} - \Delta w^{t+1} \end{aligned} \tag{2.6}$$

Nesterov's Accelerated Gradient (NAG) embodies a reformulation of the momentum method. It differs from the basic momentum by calculating gradients at the estimated future position of parameters by updating them by the momentum beforehand (2.7).

$$\begin{aligned} \Delta w^{t+1} &= \mu \Delta w^{(t)} + \eta \nabla_w \mathcal{L}(w^{(t)} - \mu \Delta w^{(t)}, \mathbf{x}, \mathbf{y}) \\ w^{(t+1)} &= w^{(t)} - \Delta w^{t+1} \end{aligned} \tag{2.7}$$

Nesterov's momentum has faster theoretical convergence for convex functions [40]. However, functions approximated by neural networks are rarely convex, thus assumptions under which the faster convergence rate holds are not met.

Adaptive learning rate methods

With small initial learning rate, optimization of a neural network would take too long or cease in a pitfall, while too large learning rate causes a divergence. A key to a good solution is the learning rate decay. By decaying the learning rate over time, a magnitude of updates to parameters decreases, allowing more refined solution to be found. Adaptive learning rate methods automatically adjust the learning rate for each parameter separately, taking its gradient history information into account. Adagrad optimizer [41] performs larger

updates to infrequent parameters and smaller updates for frequent ones. The learning rate is scaled by the square root of the sum of squared gradients. The algorithm was shown to be successful on sparse data, for example when learning word embeddings [42]. However, accumulated sum of gradients scales learning rate monotonically, which often diminishes the learning rate too fast. This drawback is fixed by the RMSprop method [43]. Instead of scaling the learning rate by the sum of squared gradients, an exponentially decaying average of squared gradients is used (2.8).

$$\begin{aligned} s^{(t+1)} &= \beta s^{(t)} + (1-\beta)\,\nabla_w \mathcal{L}(w^{(t)}, \mathbf{x}, \mathbf{y})^2 \\ w^{(t+1)} &= w^{(t)} - \frac{\eta}{\sqrt{s^{(t+1)}}} \nabla_w \mathcal{L}(w^{(t)}, \mathbf{x}, \mathbf{y}) \end{aligned} \tag{2.8}$$

The same approach is exploited in Adadelta method [44], but the learning rate parameter is omitted completely in favor of scaling in terms of running statistics. Literature contains several modifications of the RMSprop algorithms that came to spotlight, among them are Adaptive Momentum optimizer (Adam) [45] that adds the momentum term. First and second moments of gradients replace the weight updates and scale. Instead of ℓ_2 norm based scale estimation, Adamax [45] uses infinity norm and AMSGrad [46] the maximum of the squared past gradients. Nadam [47] introduces Nesterov-acceleration method into Adam.

2.1.2 Activation functions

Approximation of nonlinear functions by neural networks is achieved by point-wise activation functions ϕ that transform weighted sums of inputs. A common choice for an activation function is the logistic sigmoid function (2.9) with range $(0, 1)$:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.9}$$

or its scaled version, the hyperbolic tangent (2.10), with range in interval $(-1, 1)$:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = 2\sigma(2x) - 1. \tag{2.10}$$

Both of these functions have saturated regions that can lead to vanishing and exploding gradients and thus hamper the training process. The Rectified Linear Unit (ReLU) [48] activation solves this problem and its derivative is fast to compute. ReLU is given by the equation:

$$f(x) = \max(0, x). \tag{2.11}$$

In turn, ReLU networks are more sensitive to internal covariate shift in the inputs. Processing only negative input values would map everything to 0, producing so called "dead neuron". If all the input values are positive, no nonlinearity is applied. It was shown the deep networks converge faster if the inputs are normalized [49]. Several modifications of

vanilla ReLU function address this problem. Parametric ReLU (pReLU) [50] prevents dead neurons by allowing a small gradient for non-active neurons (2.12). Parameter a is used to scale the negative values. Its value is either fixed as 0.01 for leaky-relu [51] or is learned during the optimization process.

$$f(x) = \max(0, x) + a \min(0, x) \tag{2.12}$$

Another extension of ReLU is the Exponential Linear Unit (ELU) [52] that combines linear property of ReLU for positive values and maps negative values via exponential function:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0. \end{cases} \tag{2.13}$$

In classification scenario the output of the network implements softmax function that converts an arbitrary real value to a posterior probability of a class c_k in range $(0, 1)$ for given instance x:

$$p(c_k|x) = \frac{e^{a_k}}{\sum_{i=1}^{m} e^{a_i}} \tag{2.14}$$

where m corresponds the number of output nodes (classes) and a_k is the activation value of k-th node. Probabilities of all classes always sum up to 1.

2.1.3 Weight initialization

An essential requirement for the network to converge is a "good" weight initialization. Initializing weights to zero causes symmetry in the network and generates the same gradient for each weight. Desired asymmetry is achieved by randomly sampling weights from the Normal or Uniform distribution with zero mean. However, small weights of a deep model would diminish activations in deeper layers and causing gradients to vanish, large weights exponentially increase activations and cause exploding gradients. For stable information propagation it is desired the variance of activations in preceding layer matches the variance in current layer. A few studies estimated a suitable variance for initial weight distribution.

Assuming the input and the weights have zero mean, to match input-output variances for forward and backward pass, the Xavier initialization [53] proposes to sample the weights from Normal (2.15) or Uniform (2.16) distribution with variance scaled by the sum of input n_{in} and output n_{out} nodes.

$$w \sim \mathcal{N}\left[0, \frac{\sqrt{2}}{\sqrt{n_{in} + n_{out}}}\right] \tag{2.15}$$

$$w \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right] \tag{2.16}$$

Deep networks however often use ReLU activations that necessarily impose positive mean

value in the output distribution. To maintain stable variance propagation through a ReLU network, He initialization [50] is proposed to scale the variance of the weight distribution based only on the number of input weights n_{in}.

2.1.4 Regularization

Neural network optimization poses numerous pitfalls that make the search for an optimal solution difficult. Numerous regularization techniques aim to alleviate some of the optimization issues.

Weight regularization

Weight regularization prevents parameters from having large values. Some of the weights might become much larger than others and prioritize certain features. These factors contribute to overfitting to data, co-adapting to, or memorizing specific inputs, which leads to poor generalization of unseen observations. Penalization of weights by their $\ell 1$ norm promotes feature selection by enforcing sparse solution to the problem, which is less likely to overfit. This penalty is controlled by a hyperparameter λ (2.17).

$$\mathcal{L} = -\sum_{i}^{m} y_i \log p_i + \lambda \sum_{j}^{n} |w_j| \tag{2.17}$$

On the contrary an approach relying on the $\ell 2$ norm, known as the weight decay (2.18), enforces small weight values by penalizing outliers and creates a dense solution.

$$\mathcal{L} = -\sum_{i}^{m} y_i \log p_i + \lambda \sum_{j}^{n} w_j^2 \tag{2.18}$$

Dropout

During the training the Dropout [54] method simulates ensambling by averaging numerous thinned models derived from the original model by setting neuron activations to zero on random. This forces the model to not rely on individual nodes in the decision process.

Batch normalization

As was mentioned in Section 2.1.2, some of the activation functions such as ReLU introduce internal covariate shift. It was found that a network with normally distributed layer inputs converges faster and is more stable to input data distribution. The Batch Normalization (BN) solves this issue by normalizing features across samples in the mini-batch before applying a nonlinear activation [49]. A value of normalized feature y_i is calculated as

$$y_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.19}$$

given the running mean μ of the mini-batch features x_i and the running variance σ and a small value of ϵ for numerical stability. A feature after the normalization has zero mean and unit variance, and is further scaled and shifted by parameters γ, β learned in the training process. Normalizing inputs of each layer grants speed-up of the convergence process, allows usage of larger learning rate due to more stable data distribution, and learning is more robust towards weight initialization.

2.1.5 Convolutional Neural Network

Feed-forward neural networks with fully-connected layers are not scalable to high dimensional inputs. An alternative formulation of neuron connectivity leads to locally connected networks. For instance Convolutional Neural Network (CNN) implements local receptive field and weight sharing to achieve scalability and high performance on image data. In CNNs, input object is represented as a collection of spatial feature maps at multiple scales and levels. They vary from low level features found in local neighborhood to higher level features in form of shapes in global scale. These features are extracted by convolving an input image with learnable filters. Each filter bank produces a feature map that aggregates responses of individual filters on their own input channel. Pooling/subsampling layers are often used to decrease spatial dimensions of features to alleviate computational burden and to increase the receptive field of convolutional filters. The pooling operation substitutes a feature map region with the maximum or average of its activations, which grants invariance to translation and small rotation within the region.

Origins of CNNs are dated back to 80s when multi-layer locally connected networks were used to solve optical character recognition problems (OCR) [55, 56, 57]. The neural network architectures have been known for several decades, optimizing them was, however, a challenging task. Immense increase in computational resources and availability of annotated datasets allowed their expensive optimization. Moreover, several initialization techniques were proposed that posed a key ingredient for convergence of the training procedures. A layer-wise unsupervised pre-training was designed to initialize weights to be closer to their local minima in advance of the training [58]. Techniques described in Section 2.1.3 were found to be cheap alternatives sufficient for network convergence.

CNN models gained their fame they when became the state of the art technique for large scale image classification [12]. Neural network architectures underwent several deign changes, fully-connected layers became absent, the models got deeper [59, 60] and neuron connections more sparse [61]. CNNs have also been used in countless other tasks in computer vision, among the most popular are object detection [13], semantic segmentation [14] or image super-resolution [62].

Shortcut connections

The success of deep networks in object recognition tasks [59, 60] emphasizes the importance of depth. Optimization of deep networks is known to be hampered by vanishing/exploding gradients, addressed by numerous approaches using different activation functions [48], initialization techniques [53] or normalization of activations [49]. Despite all the efforts to maintain stable gradient flow, very deep networks sometimes tend to converge to a worse solution than their shallower counterparts. He et al. used term "degradation" problem when referring to this effect [63]. They claim the effect is not caused by overfitting and is related to the optimization process.

Inspired by gating units of LSTM cells [64], gated shortcut connections were proposed [65] to ease the information flow in very deep networks. These so called *Highway networks* transform layer input $\mathbf{x}$ via three weight matrices and their associated biases $\mathbf{b}$ (2.20).

$$\mathbf{y} = \phi\left(W_h\mathbf{x} + \mathbf{b}_h\right) \cdot \sigma\left(W_t\mathbf{x} + \mathbf{b}_t\right) + x \cdot \sigma\left(W_c\mathbf{x} + \mathbf{b}_c\right) \tag{2.20}$$

Matrix W_h parametrizes the nonlinear transform ϕ of $\mathbf{x}$, which is gated by the logistic sigmoid *transform* gate parametrized by W_t. Portion of the input carried forward is selected by the *carry* gate defined by W_c. The residual learning framework [63] is built on similar principles. A transformation of signal $\mathbf{x}$ in plain feed-forward network $\mathbf{y} = \mathcal{F}\left(\mathbf{x}, W\right)$ parametrized by weights W is reformulated in as

$$\mathbf{y} = \mathcal{F}\left(\mathbf{x}, W\right) + \mathbf{x}. \tag{2.21}$$

The layer is now not learning how to infer $\mathbf{y}$ from $\mathbf{x}$, but only how to generate their difference $\mathbf{y} - \mathbf{x}$, provided $\mathbf{x}$ and $\mathbf{y}$ have the same dimensionality. Note that (2.21) is a special case of (2.20) where $\sigma\left(W_t\mathbf{x} + \mathbf{b}_t\right)$ and $\sigma\left(W_c\mathbf{x} + \mathbf{b}_c\right)$ are matrices of ones. The residual function $\mathcal{F}$ can consist of multiple transformations, which, together with identity mapping, are referred to as *residual* block. A standard *residual* block designed by *He et al.* [63] as depicted on Figure 2.2 is composed of two layers with learned parameters and batch normalization prior to ReLU activation. A stack of such blocks forms the *Residual* network.

2.2 Transformation Methods

Many approaches in image analysis rely on an image representation given by different basis of functions. Transformation method changes the basis in which the signal is represented. An image can be interpreted as the real part of a complex periodic function. This section gives a brief overview of common transformation techniques used on image signals.

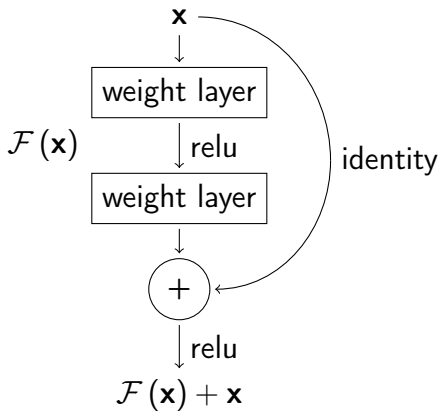


Figure 2.2: Residual block: the residual function $\mathcal{F}$ is composed of learned layers, batch normalization and relu activation. Constitutes the output **y** when summed with block input **x**.

2.2.1 Fourier Transform

The Fourier transformation decomposes image to its spatial frequency spectrum. In continuous form, an image is mapped to a sum of sinusoids with different frequencies. Contribution of each sinusoid towards the whole signal is determined by its coefficient. The Fourier transform in one dimension is expressed as a function $\mathcal{F}^1 : \mathbb{R} \to \mathbb{C}$

$$\mathcal{F}^1(u) = \int_{-\infty}^{\infty} f(x)\, e^{-j2\pi(ux)} dx \tag{2.22}$$

An image can be mapped to Fourier domain in 2 stages, by applying the 1-dimensional transformation along rows and columns subsequently. Alternatively, the 2-dimensional Fourier transform of an image f is defined as a function $\mathcal{F} : \mathbb{R} \to \mathbb{C}$

$$\mathcal{F}(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, e^{-j2\pi(ux+vy)} dxdy \tag{2.23}$$

and its inverse

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathcal{F}(x, y)\, e^{j2\pi(ux+vy)} dudv. \tag{2.24}$$

Applying Euler's formula the basis can be expressed as a sum of sine and cosine functions:

$$e^{-j2\pi(ux+vy)} = \cos[2\pi(ux + vy)] - j \sin[2\pi(ux + vy)]. \tag{2.25}$$

In practical applications images are represented by discrete 2-dimensional pixel grids. The discrete Fourier transform is thus

$$\mathcal{F}(u, v) = \frac{1}{\sqrt{nm}} \sum_{x=0}^{n} \sum_{y=0}^{m} f(x, y) \left(\cos\left[2\pi\left(\frac{ux}{n} + \frac{vy}{m}\right)\right] - j \sin\left[2\pi\left(\frac{ux}{n} + \frac{vy}{m}\right)\right]\right) \tag{2.26}$$

The scale $\frac{1}{\sqrt{nm}}$ makes the transform unitary. Fourier transform is linear and has several important properties such as frequency and space shift property, or scale property.

2.2.2 Cosine Transform

The Fourier cosine transform is defined as $\mathcal{F}_c : \mathbb{R} \to \mathbb{R}$

$$\mathcal{F}_c(u) = \int_{-\infty}^{\infty} f(x) \cos(2\pi u x) dx. \tag{2.27}$$

For image signals lying on a discrete grid, integration is performed by summing responses with fixed step size of one pixel. The formulation of this transformation on discrete signals is often expressed as

$$\mathcal{F}_{DCT}(u) = \sqrt{\frac{\alpha(u)}{n}} \sum_{x=0}^{n-1} f(x) \cos\left[\frac{\pi}{n}\left(x + \frac{1}{2}\right) u\right], \quad \text{where } \alpha(u) = \begin{cases} 1, & u = 0 \\ 2, & \text{otherwise.} \end{cases} \tag{2.28}$$

For a signal of length n it is possible to define a transformation matrix C (2.29), columns of which are formed by the DCT basis functions.

$$C = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \cos\left(\frac{\pi}{2n}\right) & \cos\left(\frac{\pi}{n}\right) & \cos\left(\frac{3\pi}{2n}\right) & \cdots & \cos\left(\frac{\pi(n-1)}{2n}\right) \\ \frac{1}{\sqrt{2}} & \cos\left(\frac{3\pi}{2n}\right) & \cos\left(\frac{3\pi}{n}\right) & \cos\left(\frac{9\pi}{2n}\right) & \cdots & \cos\left(\frac{3\pi(n-1)}{2n}\right) \\ \frac{1}{\sqrt{2}} & \cos\left(\frac{5\pi}{2n}\right) & \cos\left(\frac{5\pi}{n}\right) & \cos\left(\frac{15\pi}{2n}\right) & \cdots & \cos\left(\frac{5\pi(n-1)}{2n}\right) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{\sqrt{2}} & \cos\left(\frac{\pi(2n-1)}{2n}\right) & \cos\left(\frac{\pi(2n-1)}{n}\right) & \cos\left(\frac{\pi(6n-3)}{2n}\right) & \cdots & \cos\left(\frac{\pi\left(2n^2-3n+1\right)}{n^2}\right) \end{bmatrix} \tag{2.29}$$

An image $\mathcal{I} \in \mathbb{R}^{n \times n}$ can be transformed into DCT domain via C by transforming rows and columns of the image matrix separately

$$\mathcal{F}_{DCT} = C^T \mathcal{I} C. \tag{2.30}$$

Images that do not have their width equal to their height can be simply transformed by using two transformation matrices of different sizes.

Alternatively, an image $\mathcal{I} \in \mathbb{R}^{n \times m}$ can be transformed with 2-dimensional DCT as

$$\mathcal{F}_{DCT}(u, v) = \sqrt{\frac{\alpha_u}{n}} \sqrt{\frac{\alpha_v}{m}} \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} \mathcal{I}(x, y) \cos\left[\frac{\pi}{n}\left(x + \frac{1}{2}\right) u\right] \cos\left[\frac{\pi}{m}\left(y + \frac{1}{2}\right) v\right] \tag{2.31}$$

which results in coefficient $\mathcal{F}_{DCT}(u, v)$ representing overlap of an input with sinusoids at frequency u and v in particular dimensions. Basis functions are often normalized by scaling factors $\alpha_k = 1$ if $k = 0$, otherwise $\alpha_k = 2$.

The purpose of DCT is to decorrelate signals. Due to its energy compaction properties on natural images [66] it is widely used for image compression in JPEG image format.

A modification of DCT described in Eq. (2.28) is also used in audio encoding formats. The DCT shows better energy compaction properties on natural images than DFT [67]. Karhunen-loeve transform (KLT) is considered to be optimal in signal decorrelation, however the signal is transformed via specific basis functions that are not separable and need to be estimated for every image. It was also shown that normalized KLT eigenvectors estimated on patches of natural images converge to DCT bases as its correlation coefficient approaches one [68].

2.2.3 Wavelet Transform

Another type of transformation with orthonormal complete basis, the wavelet transform, captures signal responses with respect to frequency and position of the original signal. The advantage over Fourier transform is a sensitivity to frequencies at different scales, useful for non-stationary data. The basis set is obtained by shifting and scaling a mother wavelet ψ.

$$\psi_{uv}(x) = 2^{\frac{u}{2}}\psi\left(2^u x - v\right). \tag{2.32}$$

Coefficient $\mathcal{W}_\psi(u, v)$ for a particular scale u and position v is determined as

$$\mathcal{W}_\psi(u, v) = \int_{-\infty}^{\infty} f(x)\, \frac{1}{\sqrt{2^u}}\psi\left(\frac{x - v2^u}{2^u}\right) dx \tag{2.33}$$

with possible extension to discrete form as for Fourier or Cosine transforms. The signal can be synthesized from the wavelet coefficients as

$$f(x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathcal{W}_\psi(u, v)\, \psi_{uv}(x)\, dudv. \tag{2.34}$$

2D Wavelet transformation on images can be performed by successively transforming image columns and rows at different scales. An example of ψ frequently used for image processing is the *Haar* wavelet, the simplest from the wavelet family. Haar wavelet is defined as a step function

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \leq x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq x < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2.35}$$

One-dimensional discrete Haar transformation of a vector of size n is performed by matrix multiplication by an orthogonal matrix H_n constructed from shifted and scaled step function. Constant n is constrained to be a power of 2. An example of H_2:

$$H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.36}$$

and H_4:

$$H_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}. \tag{2.37}$$

To transform an image block-wise, slices of columns and rows have to be manipulated. Conveniently, a filter bank can be constructed to transform the image by convolution. Rows of H_n can be considered as 1D convolutional filters. H can be represented as a vector of basis functions (rows):

$$H_N = \begin{pmatrix} h_0^T \\ h_1^T \\ \vdots \\ h_{n-1}^T \end{pmatrix}. \tag{2.38}$$

A 2D filter bank is constructed as a set $U = \{H_{u,v} | u \in \langle 0..n-1 \rangle, v \in \langle 0..n-1 \rangle\}$ composed of matrices H_{uv} as products of matrix multiplications between columns of H_n:

$$H_{uv} = h_u h_v^T \tag{2.39}$$

for every $u, v < n$.

Another example of a wavelet family often used in vision problems is Morlet wavelet. Its complex bases are defined as

$$\psi(x) = \frac{1}{\sqrt{\pi u}} e^{-\frac{x^2}{u}} e^{j2\pi vx}. \tag{2.40}$$

2.3 Compression Techniques

This section will give an overview of a few compression techniques that will be further investigated.

2.3.1 JPEG

JPEG coding standard [21] was designed by Joint Photography Experts Group in 1992 and is still one of the most broadly used formats for storing still images, commonly used in cameras or web systems. The format grants good compression rates, however the compression causes information loss. This section will illustrate how this compression works. JPEG uses YCbCr color scheme, consisting of luma component (Y) representing luminance, and two chroma components (Cb and Cr) capturing the blue and the red color difference. For larger compression ratio the color channels (Cb, Cr) are often sub-sampled by factor 2 in horizontal direction, noted as 4:2:2 scheme, or in both horizontal and vertical directions known as 4:2:0. For no chroma sub-sampling, 4:4:4 scheme is used. Each image channel

is split into 8x8 non-overlapping pixel blocks, 128 is subtracted from each pixel value, and the blocks are transformed by 2-dimensional Discrete Cosine Transform.

Discrete Cosine Transform (DCT) decomposes a finite vector into a sum of a set of cosine basis functions. The 8×8 pixel block is transformed via 2-dimensional DCT (2.31) where $m = n = 8$. Result of the transform is a mapping of spatial image block to a set of coefficients in the frequency domain (Fourier space of real numbers). Upper-left corner of the block contains average intensity of the block (DC coefficient), while the other coefficients (AC) capture amplitudes at which the particular cosine functions oscillate, with the highest frequency occurring in the bottom-right corner. The information loss occurs during the quantization, as a result of point-wise integer division of the coefficients with the 8×8 quantization matrix. The matrix is designed to discard high frequencies and its content is dependent on the compression rate.

After the quantization, most of the high-frequency coefficients have typically value zero. From a 2-dimensional matrix they are mapped into 1-dimensional vector, ordering coefficients in an anti-diagonal order starting from the top-left corner with the DC coefficient (see ordering on Figure 2.3). The vector length is compressed by run-length encoding, aggregating all zero elements preceding any non-zero element into one number. To complete the compression process vector values are mapped by Huffman coding.

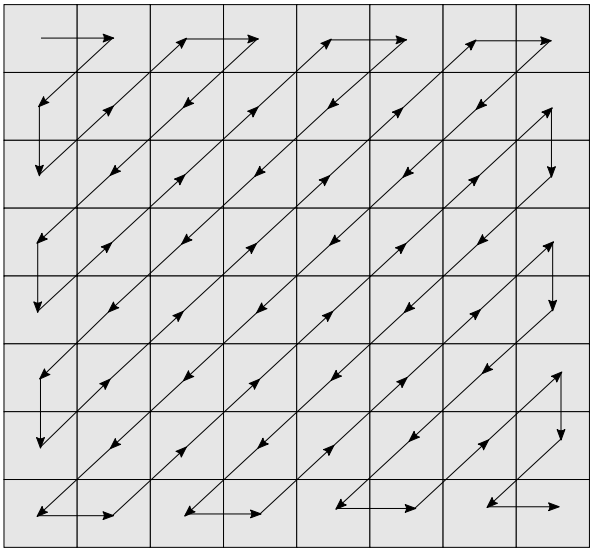


Figure 2.3: Order in which DC and AC coefficients are mapped into 1-dimensional vector.

Huffman coding constructs a mapping table based on symbol/value frequency of occurrence. The most frequent symbols are mapped to the shortest codes and vice versa. The codes are inferred from the Huffman tree, a binary tree with leaf nodes corresponding to symbols with their occurrence frequency, and internal nodes containing summed frequencies of their children. The tree is built by merging the least frequent symbols until all the symbols are included. The final codes are represented via path taken from the root to the leaf node, assigning binary 0 to every left branch taken and binary 1 to the right ones.

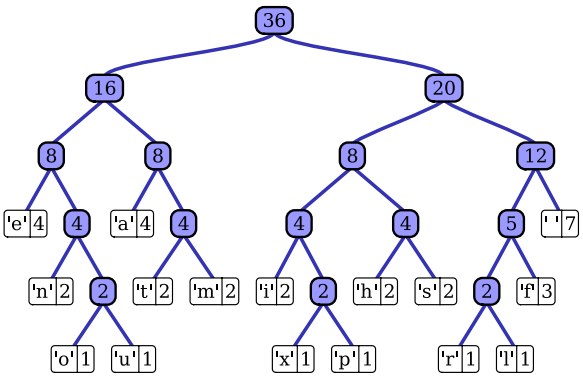


Figure 2.4: Huffman tree is constructed from leaf nodes that represent symbols to be encoded with their occurrance frequency, and internal nodes summing the frequencies of their children. The code for a specific symbol is generated traversing the tree from the root node using 0 for each left branch taken and 1 for the right branches. (source: `https://en.wikipedia.org/wiki/Huffman_coding`)

An image patch is in its final form composed of quantized DCT coefficients flattened into 1D vector in an anti-diagonal order, representing each non-zero coefficient by 3 values: the number of zeros preceding the coefficient, the length of the coefficient encoding, and the Huffman code for the coefficient value. An image is decompressed by performing inverse operations on the compressed coefficients. The principle of splitting an image into 8×8 blocks and transforming them via DCT is also used on MPEG key frames, the only frames that encode static information in videos.

2.3.2 JPEG 2000

The new standard for JPEG created in year 2000 [69] addresses some of the original JPEG format drawbacks. The format allows higher compression ratio while maintaining equal quality. The blocking artefacts caused by tiling are not visible, only ringing artefacts remain, manifested as a blur and rings near edges of an image. The file format allows using any color space, arbitrary bit depth and transparency. Images are transmitted and rendered progressively. JPEG2000 uses arbitrary size tiles and replaces DCT with wavelet transformation (see Section 2.2.3). The Discrete Wavelet Transform (DWT) is used for dyadic image decomposition. Approximation and detail coefficients are extracted at different scales. Cohen–Daubechies–Feauveau (CDF) 9/7 wavelet is used for lossy compression. Lossless compression is also possible through CDF 5/3 wavelet that prevents floating point number rounding errors. The wavelet coefficients are quantized with an integer step value that is inversely proportional to precision. Different quantization can be applied at different parts of an image. The last stage of compression uses arithmetic coder similar to the one used in standard JPEG format. Although JPEG2000 has superior properties to baseline JPEG format, it is not widely adopted by web systems and browsers.

2.4 Scattering networks

The idea of feature extraction from images by convolving them with learned filters is inspired by preceding feature extraction techniques using fixed or so called "hand-crafted" convolutional filters. One such example is the Scattering network [27] that uses wavelet filters to model geometrical variabilities, and can produce representations that are translation and rotation invariant [70]. Scattering networks were shown [70] to achieve comparable classification accuracy to CNNs with filters learned in an unsupervised way.

The scattering transform is realized through a bank of dilated and rotated mother wavelet. Bruna et al. [27] uses morlet wavelet (2.40) to construct multiscale directional filters

$$\psi_{2^j r}(u) = 2^{2j}\psi\left(\frac{2^j u}{r}\right) \tag{2.41}$$

where $j \in \mathbb{Z}$ corresponds to scale and $r \in G$ to rotation of the filter from a discrete finite rotation group G in $\mathbb{R}^2$. The notation can be simplified as $\lambda = 2^j r$. $U[\lambda]\mathbf{x} = |\mathbf{x} * \boldsymbol{\psi}_\lambda|$ is a wavelet transformation and magnitude extraction operator. Output of the network are coefficients computed at frequencies $2^j > 2^{-J}$. A sequence of operations is denoted by a path $p = (\lambda_1, \lambda_2, \ldots, \lambda_m)$. Higher order coefficients $U[p]$ are computed as

$$U[p]\mathbf{x} = U[\lambda_m] \ldots U[\lambda_2]U[\lambda_1]\mathbf{x} = |\,||\mathbf{x} * \boldsymbol{\psi}_{\lambda_1}| * \boldsymbol{\psi}_{\lambda_2}| \ldots \boldsymbol{\psi}_{\lambda_m}|. \tag{2.42}$$

The Gaussian filter at scale 2^J is used to average the coefficients

$$\phi_{2^J}(u) = \frac{1}{2^{2J}}\phi\left(\frac{u}{2^J}\right). \tag{2.43}$$

Aggregation through averaging into coefficients S_J (2.44) gives an invariant representation.

$$S_J[p]\mathbf{x} = U[p]\mathbf{x} * \boldsymbol{\phi}_{2^J} \tag{2.44}$$

Scattering networks implemented with different mother wavelets can also be found in the literature [71].

2.5 CNNs with non-standard bases

Several works have allowed the flexibility of CNN filter banks by restricting its composition from a finite basis set. A filter W is formed by a linear combination of a basis family ψ through learned coefficients α:

$$W = \sum_{i=0}^{k-1} \alpha_i \boldsymbol{\psi}_i. \tag{2.45}$$

One such implementation is the structured receptive field block [26], which motivates the use of Gaussian function derivative family for ψ by scale-space theory [72]. CNN filter $W(u)$

is interpreted as a function and can be rewritten through Taylor expansion of maximum order n

$$G(.,\sigma) * W(u) = \sum_{m=0}^{n} \frac{(G^m(.,\sigma) * W)(a)}{m!} (u-a)^m \tag{2.46}$$

with $G(.,\sigma)$ as the Gaussian kernel and $G^m(.,\sigma)$ its m-th derivative. This decomposition was successful at expressing filters with large spatial kernel and achieved good results with limited samples. Kobayashi [73] has used Gram-Schmidt process to orthogonalize an incomplete set of Gaussian derivative filters and used it as a basis to learn effective filters.

Another type of bases used for filter decomposition were the Fourier-Bessel bases:

$$W_{m,q}(r,\phi) = c_{m,q} J_m(R_{m,q} r)\, e^{im\phi}. \tag{2.47}$$

They are defined on unit disc through polar coordinates r, ϕ, parametrized by angular m and radial q frequency. J_m is the Bessel function of the first kind, $R_{m,q}$ its q-th root and $c_{m,q}$ is used for normalization [74]. By high frequency basis truncation the authors have achieved parameter reduction and stable representations that they have demonstrated on a denoising task [74].

Family of circular harmonics (2.48) has been used to construct a rotation equivariant CNN [29].

$$W_m(r,\phi,R,\beta) = R(r) e^{i(m\phi+\beta)} \tag{2.48}$$

Its filter of rotation order m is complex valued and defined on polar coordinates r, ϕ. The phase offset β and radial profile R, responsible for orientation and shape of the filter, are learned during training. For image $\mathcal{I}$ rotated by θ the filter satisfies (2.49).

$$W_m * \mathcal{I}(r,\phi-\theta) = e^{im\theta} [W_m * \mathcal{I}(r,\phi)] \tag{2.49}$$

Recently, the scattering transform has been extended by learnable parameters [75]. For an input $x(c,\mathbf{u})$ at channel c and vector of spatial coordinates $\mathbf{u}$, output feature $y(f,\mathbf{u})$ is produced by summing $a_{f,\lambda}(c)$ weighted magnitudes of detail coefficients with the sum of $b_f(c)$ weighted approximation coefficients (2.50), where Λ is a set of all scales and orientations.

$$y(f,\mathbf{u}) = \sum_{\lambda\in\Lambda} \sum_{c=0}^{n-1} |x(c,\mathbf{u}) * \psi_\lambda(\mathbf{u})|\, a_{f,\lambda}(c) + \sum_{c=0}^{n-1} \Big(x(c,\mathbf{u}) * \phi_J(\mathbf{u}) \Big) b_f(c) \tag{2.50}$$

Chapter 3

Transforming CNN input with DCT

3.1 Introduction

In the past decade, convolutional neural networks (CNN) have grown in popularity for performing image processing tasks. In CNNs, convolutional filters are trained to extract relevant features and shapes from images to perform classification for instance (c.f. Figure 3.5). Convolutions by small size filters have already been widely adopted for example in VGG [59] or ResNet [63]. A small filter covers only a small part of an image and thus has to be applied numerous times. In early stages of convolutional networks that learn from large images, the spatial resolution of the feature space is rather large and the neural network has to perform a vast amount of operations. This work[1] harnesses the idea to exploit broadly used image compression techniques, in particular JPEG compression format for classification (see Section 3.3). Standard image classification techniques use CNNs on spatial representation of the data, for example RGB pixel intensities. In contrast images in JPEG format are mapped to the frequency domain, which applicability for CNNs is explored in the paper. Section 3.2 firstly introduces the related research works, Section 3.3 gives detailed explanation of the proposed approach, and we show several experimental results in Section 3.4 that confirm that Convolutional Neural Networks can be applied efficiently to image data in the frequency domain.

3.2 Related Work

Several papers have addressed the learning from frequency data. Most of the approaches can be found in forensics or in object recognition tasks.

[1] Part of this chapter has been published in [76].

Forensics

Networks trained on DCT coefficients are frequently used in forensics for detection of tampered parts in images. These parts are assumed to have different distribution of DCT coefficients from the rest of the image. A common practice is to classify histograms of preselected DCT coefficients by 1D convolutional network [77, 78, 79]. Furthermore, this task has been addressed by a multi-branch 2D CNN [80] trained on feature maps spanned by the first 20 AC coefficients (corresponding to non-zero frequencies in DCT) extracted from JPEG images.

Object recognition & classification

A number of studies have investigated the use of spectral image representations given by the DCT for object recognition. The DCT on small resolution images coupled with coefficient truncation was used to speed up the training of fully connected sparse autoencoders [81]. DCT features from entire images were used to train Radial Basis Function Network for face recognition [24]. A significant convergence speedup and case-specific accuracy improvement have been achieved by applying DCT transform to early stage learned feature maps in shallow CNNs [82] whereas the later stage convolutional filters were operating on a sparse spectral feature representation.

In contrast with most of the previous studies, our approach works with frequency information of the image patches, preserving their global location. Under such setting, approaches that exploit spatial dependencies of the data can be used, and in particular convolutional neural networks. CNNs have been successful at processing data that have underlying Euclidean or a regular grid-like structure (e.g. pixel grid), and the size and structure of input data is expected to be fixed to be fed into the networks used [83].

A few subsequent works have also explored the use of CNNs on block-based DCT images. A small-scale CNN was trained on DCT coefficients in Borhanuddin et al. [84]. In Gueguen et al. [25] it was demonstrated how DCT coefficients can be efficiently used to train CNNs for classification, where the DCT coefficients are taken directly from the JPEG image format. DCT coefficients extracted the same way have also been used to train a CNN for object detection [85] and semantic segmentation [86].

We present next our approach that takes advantage of JPEG compressed image format for creating CNN compliant input data to feed into CNN based classifiers.

3.3 Method

Our method is motivated by reusing well performing and broadly used JPEG image compression. Firstly we will briefly illustrate how the compression works. JPEG uses YCbCr color scheme, consisting of luma component (Y), representing luminance, and two chroma

components, (Cb and Cr), capturing the blue and the red color difference. Color channels are sometimes subsampled for larger compression ratio. Each image channel is split into 8x8 pixel patches, 128 is subtracted from each pixel value, and finally the regions are transformed by 2 dimensional Discrete Cosine Transform (DCT).

Discrete Cosine Transform (DCT): 2-dimensional DCT transform (see Section 2.2.2) is applied to input image patches. The result of the DCT transform is a mapping of the patch in frequency domain. Upper-left corner of the patch contains low frequencies while the high frequencies occur in bottom-right part. If the compression is lossy, the patch coefficients are quantized by 8x8 matrix to discard information in high frequencies.

The coefficients in transformed and quantized patches are ordered in an antidiagonal order starting from top-left corner (the lowest frequency). Finally, this one dimensional sequence is compressed with run-length encoding and converted into Huffman code (entropy encoding). The compression is described with more detail in Section 2.3.1. Given the non-homogeneous structure of the Huffman code, its variable length is causing a challenge to fit the data to a fixed-sized input network such as CNN based architectures. Our experiments are focused here on using fixed size outputs of the DCT transform as inputs for our classifier as shown on the Figure 3.1.

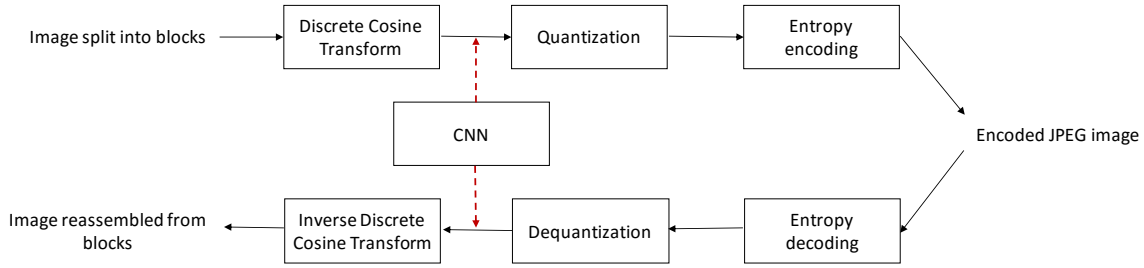


Figure 3.1: Flowchart showing JPEG (de)compression steps: Red dashed arrows indicate where CNN classifier can be plugged-in.

Color information is important for reliable estimation of local features in digital images [87]. Selection of an appropriate color space often plays a big role in tasks such as skin segmentation [88, 89]. Most color space transformations are simple linear operations, a CNN of a sufficient capacity should be able to learn the transformation implicitly. In this chapter we are focusing on the YCbCr color format that has sparse DCT representation in chromatic channels and is used in JPEG images that can be directly used in the proposed approach. We consider an image in YCbCr color space split to patches or windows defined by the window size t. Windows of an arbitrary size are considered, and since the JPEG compression splits image into windows of size 8, the compressed data is not used, instead, each window from the original image data is transformed into DCT space. Alternatively, if $t = 8$ is desired, entropy coded JPEG images can be decoded and dequantized to obtain the input. This process is already performed when extracting RGB information from JPEG images,

and can avoid additional computational costs of performing the inverse DCT. The output of the DCT transform is a set of frequency coefficients, each given by a particular DCT basis function. A set of these coefficients represents a feature space of an image window. The input filter $\mathbf{w} \in \mathbb{R}^{kt \times kt}$ of the proposed approach is designed to cover k adjacent windows in each image direction. The filter application on the input image $\mathbf{x}$ in block-DCT space with block size $t \times t$ can be expressed via following cross-correlation

$$\mathbf{w} \star x\,(ut, vt) = \sum_{i=0}^{kt-1} \sum_{j=0}^{kt-1} w\,(i,j)\, x\,(ut+i, vt+j)\,. \tag{3.1}$$

defined on window coordinates $u, v \in \mathbb{N}$. We will mostly work with 2 window neighborhood ($k = 2$) forming a $2t \times 2t$ kernel. The learnable filter weights can be seen as performing weighted average of DCT coefficients of the input. Spatial location of a weight in the filter determines which frequency in the feature space it is responsible for. To avoid application of weights to coefficients with different frequencies, the convolutional filter slides along the image with a stride of the window size t. Stride is a step size in number of pixels by which the convolutional filter slides along rows and columns of an image. Figure 3.2 provides graphical illustration with the window size $t = 2$ and a neighborhood of 2 blocks (k=2).

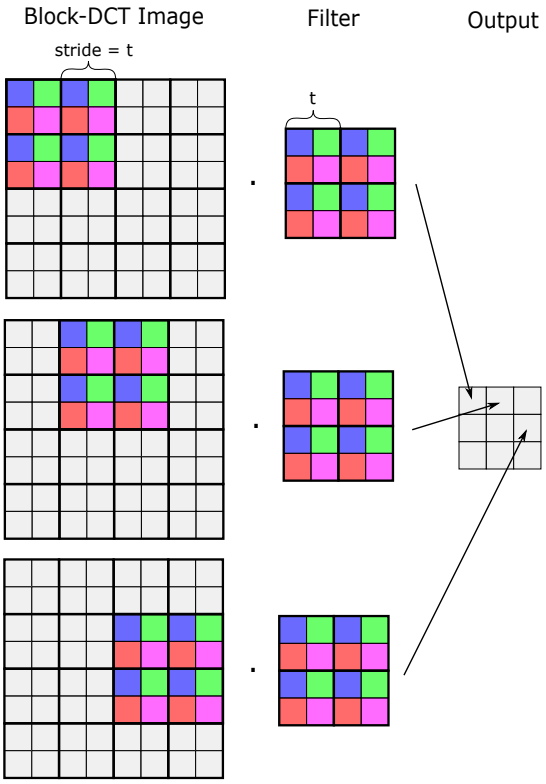


Figure 3.2: Filtering block-DCT image with window size $t = 2$ and filter of size $2t \times 2t$. Filter stride equals t. Individual colors represent particular frequencies in considered DCT spectrum.

For a window size $t = 8$, a filter of 16×16 is used. Note that the window does not need

to have a square size when fed to the CNN, it can have an arbitrary shape composed of t^2 values (e.g. a 1 dimensional vector $1 \times t^2$ as long as relative position of the windows stays intact). Further CNN layers do not need any modification and are applied directly on the output of the first layer. The overall proposed approach outline is depicted on the Figure 3.3.

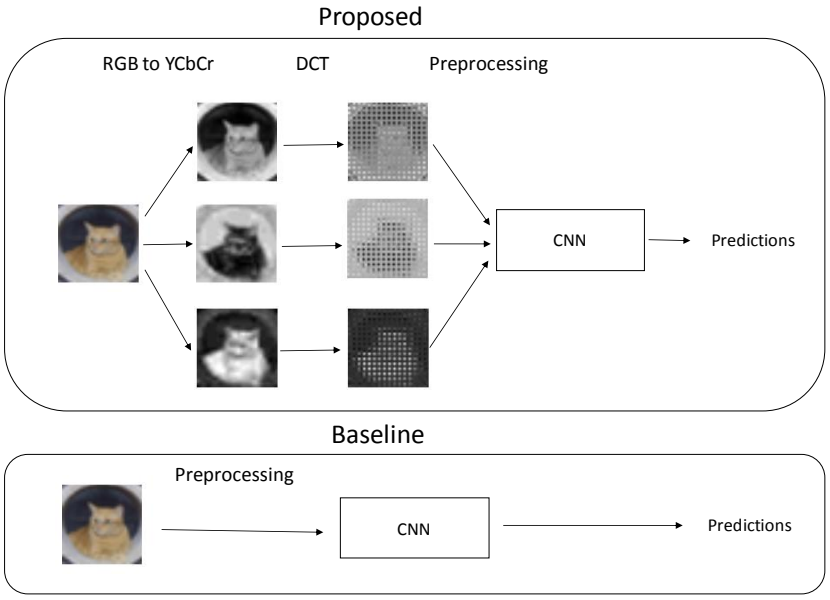


Figure 3.3: Flowchart of the proposed approach compared to the baseline, first converting an RGB image to YCbCr, then slicing it to windows and transforming each window separately prior to classification.

Computational Complexity: The computational complexity of the model is not affected by basis change of input data. The only required modification to any existing CNN architecture is exclusive to the first layer. The complexity of this layer depends on the number of windows k that the convolutional filter covers. The filter is applied with increased stride and therefore for $k \leq 3$ used in our experiments the complexity decreases compared to typical filters used without stride [90]. Our approach, due to the use of stride also decreases the spatial resolution of the feature maps that effectively reduces processing memory requirements.

3.4 Experimental Results

To evaluate the suitability of frequency data representation for training a CNN, experiments are performed on 2 well-known public datasets: CIFAR10 [91] and MNIST [92]. For experimentation we use the official dataset splits into train and test sets. The proposed approach as well as the baselines are trained on the training set and are compared by their classification accuracy achieved on the test set.

3.4.1 CIFAR10

The CIFAR10 dataset of colored natural images of size 32×32 is used to compare low level features learned from frequency representation in contrast with original raw image intensity. This experiment is conducted for two values of t: 4 and 8. For each window size a different shallow network is used, further referred to as CNN-A ($t = 4$) and CNN-B ($t = 8$) with architectural details in Table 3.1. Both of these networks have filters designed to be compatible with window-based DCT inputs as explained in Section 3.3. Features learned by convolutional layers are transformed by batch normalization [49] and activated by ReLU [93]. CNN-A further performs average pooling to downsample the feature space, while CNN-B uses dropout technique to reduce overfitting [54].

4x4 (CNN-A)			8x8 (CNN-B)		
layer name	kernel type	output size	layer name	kernel type	output size
conv	[8x8, 4x4]	7x7, 64	conv	[16x16, 8x8]	3x3, 64
avg-pool	[3x3, 2x2]	3x3, 64	dropout	p=0.25	3x3, 64
softmax		1, 10	softmax		1, 10

Table 3.1: The specification of CNN-A and CNN-B, shallow convolutional networks for learning low level CIFAR10 features. A kernel definition of [8x8, 4x4] denotes a filter with spatial size 8×8 with stride 4×4. Output size of $3 \times 3, 64$ represents 64 feature maps with spatial resolution 3×3. Dropout layer probability is defined by p.

Both shallow architectures (CNN-A, CNN-B) are trained on the original RGB data, YCbCr data representation and on its DCT transform. Stochastic gradient descent with momentum is used to train the network on all 50 000 training images, with learning rate starting at 0.1, reduced by factor 10 after 40 and 60 epochs for a total length of 80 epochs. During the training we use weight decay $\lambda = 0.0001$, and batch size 256. Two input pre-processing techniques are reported: per feature normalization, noted as *mean/std* (for frequency data after performing DCT on non-centered data) and the *center/max* pre-processing by subtraction of 128 from the image pixel values and division by the original maximum value (255 for colors, $t \cdot 256$ for frequency data after performing DCT on centered images). Using the whole test set of 10000 images, the highest classification accuracy was observed for models trained on DCT data with *center/max* pre-processing. The results of 20 runs for each setting, depicted in Table 3.2, demonstrate the window based DCT transform facilitates learning of more discriminative low-level features.

window size	4x4 (CNN-A)		8x8 (CNN-B)	
preprocessing	mean/std ↑	center/max ↑	mean/std ↑	center/max ↑
RGB	66.82 ± 0.39	66.96 ± 0.36	60.36 ± 0.37	60.20 ± 0.31
YCbCr	65.57 ± 0.36	67.07 ± 0.24	59.60 ± 0.34	60.25 ± 0.27
DCT	66.84 ± 0.23	$\mathbf{67.24 \pm 0.26}$	60.25 ± 0.28	$\mathbf{60.87 \pm 0.26}$

Table 3.2: Classification scores (mean $\pm$ std %) of shallow CNN-A and CNN-B networks over 20 runs on CIFAR10.

Empiric results indicate the standard *mean/std* preprocessing technique for RGB data is not well suited for data in YCbCr format. We suspect the imbalance between standard deviations of luma and chroma channels scales down the more important luma channel that is then not exploited sufficiently by a layer normalized with L2 regularization.

The low level features show encouraging results, motivating experiments on a deeper network. A network CNN-C, with details in Table 3.3, is trained with both previously mentioned pre-processing approaches for all 3 previously used data formats. Each layer with trainable parameters (except for the softmax layer) is batch normalized and activated by ReLU. Excluding the first layer, all convolutional layers preserve spatial dimensionality of the features. Due to low resolution of the dataset, window size t is set to 2 to prevent drastic downsampling after the first convolutional layer. The model has more parameters than CNN-A or CNN-B, thus has higher capacity to overfit. To add extra regularization we apply dropout after the second and the fourth convolutional layer with probability 0.25 and on the output of the first fully-connected layer with probability 0.5. We use the same training procedure as for the shallow network with difference in the learning rate scheduler: the network is trained for a length of 300 epochs, having initial learning rate 0.1 reduced by factor 5 after 90, 180 and 240 epochs.

layer name	kernel type	output size
conv1	[4x4, 2x2]	15x15, 64
conv2	[3x3, 1x1]	15x15, 64
dropout1	p = 0.25	15x15, 64
conv3	[3x3, 1x1]	15x15, 64
max-pool1	[3x3, 2x2]	7x7, 64
conv4	[3x3, 1x1]	7x7, 128
max-pool2	[3x3, 2x2]	3x3, 128
dropout2	p = 0.25	3x3, 128
dense1		1, 512
dropout3	p = 0.5	1, 512
softmax		1, 10

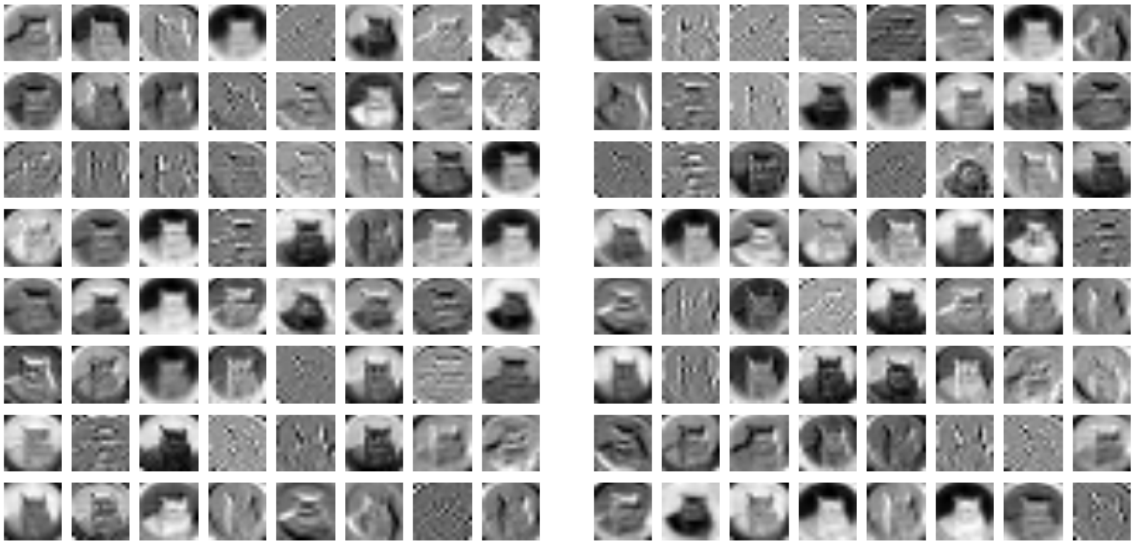
Table 3.3: Convolutional network architecture (CNN-C) used on CIFAR10.

The Table 3.4 reports both median and mean accuracy on the test set after 300 training epochs, both without and with simple augmentation. The images are augmented by random horizontal flipping and random shifting by multiples of t, at most by $2t$, filling missing pixels by zeros. When augmentation is not used, the network trained on DCT achieves slightly higher accuracy, however, RGB representation benefits from the augmentation more than the other representations.

We conduct a visual evaluation of CNN-C network features learned on RGB and DCT data. Firstly, low level first layer activations of the model (*mean/std* and *center/max* preprocessing for RGB and DCT data respectively) for a sample test image are rendered on Figure 3.4 that confirm both networks have learned similarly looking features. Furthermore,

preprocessing	augmentation	mean/std ↑	center/max ↑
RGB		86.11 (86.22 ± 0.25)	86.13 (86.21 ± 0.29)
YCbCr		85.77 (85.47 ± 0.41)	85.96 (86.13 ± 0.49)
DCT		**86.35 (86.25 ± 0.42)**	86.30 (86.27 ± 0.19)
RGB	✓	**90.49 (90.49 ± 0.21)**	90.35 (90.30 ± 0.12)
YCbCr	✓	89.98 (90.08 ± 0.31)	90.28 (90.18 ± 0.27)
DCT	✓	89.97 (90.07 ± 0.27)	90.27 (90.28 ± 0.13)

Table 3.4: Classification accuracy of CNN-C computed as median (mean ± std %) over 5 runs on CIFAR10 dataset.



(a) RGB (b) DCT

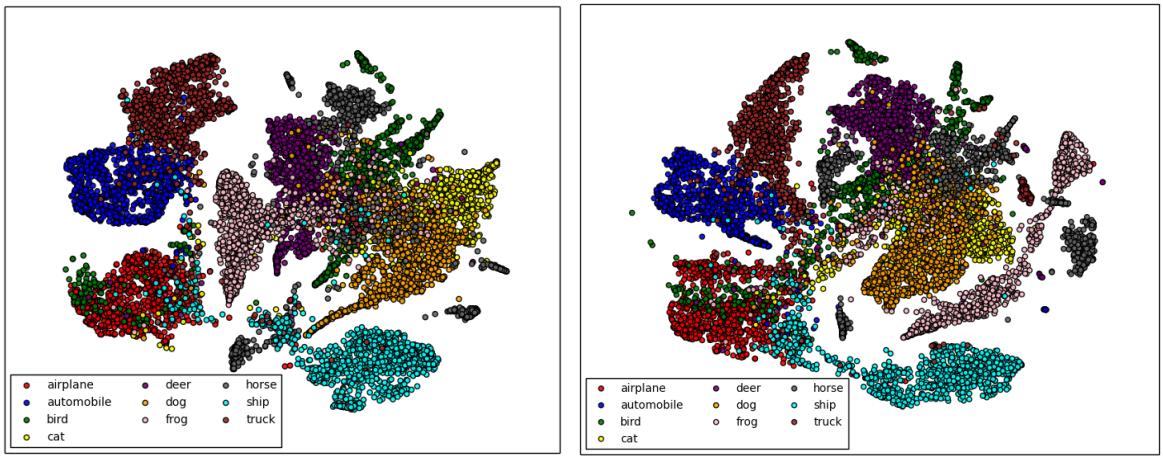
Figure 3.4: First layer activations of a median score CNN-C model trained with augmentation on RGB data (**a**) and DCT representation (**b**), created by inferring the sample image from the Figure 3.3.

discriminative properties of the high level features of CNN-C model trained on DCT are demonstrated by mapping activations of the fully-connected layer ("dense1" in Table 3.3) to the 2D space (Figure 3.5) via t-distributed Stochastic Neighbor Embedding (t-SNE) [94]. Class compactness in the 2D projection is visually similar to the projection of features of the same network trained on RGB data.

A comparison of classification errors made by models trained on RGB and DCT data points out that both networks are making similar mistakes. Figure 3.6 presents the confusion matrices for both networks, showing the most common mistake for both models was confusing dog for a cat. The findings further support the claim that both networks learn similar representations.

3.4.2 MNIST

We perform a similar experiment on MNIST dataset, training a CNN-D network (see Table 3.5) similar to the one used on CIFAR10 data. Unlike CIFAR10, MNIST contains only



(a) RGB (b) DCT

Figure 3.5: A mapping of high-level features of the network trained on standard RGB (**a**) and DCT (**b**) representation of the entire CIFAR10 test set into 2D plane via t-SNE.

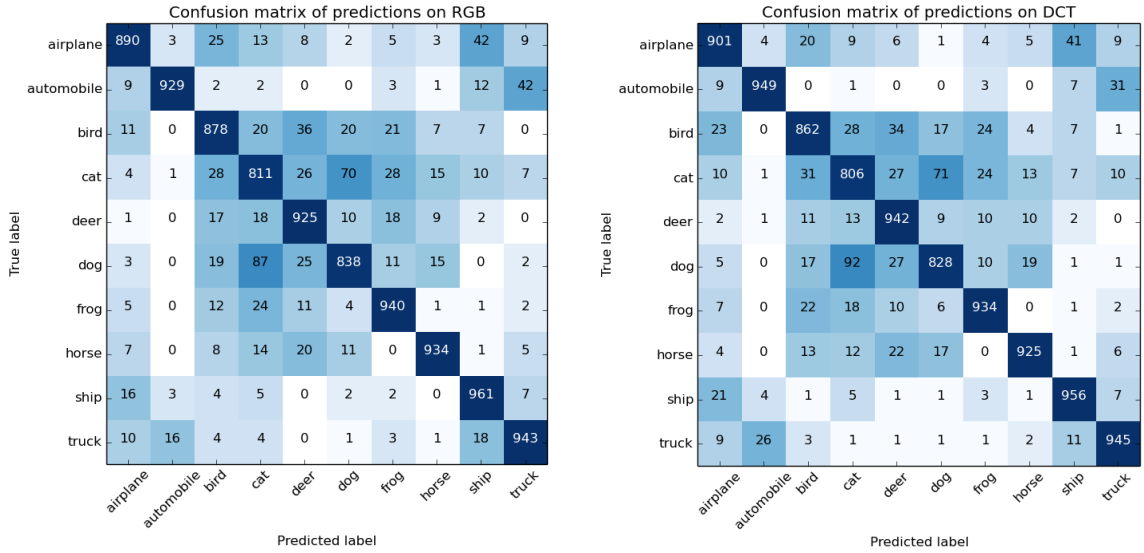


Figure 3.6: A confusion matrix of predictions by CNN-C network trained on RGB (**left**) and DCT representation (**right**) of CIFAR10.

one color channel with dimension 28x28, therefore CNN-C is not directly applicable on this data. MNIST dataset is also less complex and does not require network with that many layers. With the smaller model size we use dropout only in the deeper layers. Here we use $t = 2$ and train the network on the whole 60000 image large train set for 30 epochs (the entire dataset passes) with stochastic gradient descent with batches of 128 images. The initial learning rate is 0.1, which is every 10 epochs reduced by factor 10, and we use momentum of 0.9.

Multiple pre-processing approaches are investigated for the original data representation and for its DCT transform. These consist of subtraction "-" of a constant value and scaling "/" that can be applied pre-DCT or post-DCT. Table 3.6 lists average errors over 20 runs on the whole test set of 10000 images for each pre-processing method. The most successful pre-

layer name	kernel type	output size
conv1	[4x4, 2x2]	13x13, 64
conv2	[3x3, 1x1]	13x13, 64
max-pool1	[3x3, 2x2]	6x6, 64
conv3	[3x3, 1x1]	6x6, 128
max-pool2	[2x2, 2x2]	3x3, 128
dropout1	p = 0.25	3x3, 128
dense1		1, 512
dropout2	p = 0.5	1, 512
softmax		1, 10

Table 3.5: Convolutional network architecture (CNN-D) used on MNIST.

preprocessing	Orig. error (%) ↓	preprocessing	DCT error (%) ↓
/255	0.4455 ± 0.0499	DCT/512	0.4245 ± 0.0332
-128/128	0.4415 ± 0.0273	-128 DCT/256	0.4405 ± 0.0458
-128/255	0.4355 ± 0.0213	-128 DCT/512	$\mathbf{0.4105 \pm 0.0383}$
-mean/std	0.4445 ± 0.0407	DCT-mean/std	0.4320 ± 0.0499
-mean/128	$\mathbf{0.4245 \pm 0.0322}$	DCT-mean/256	0.4360 ± 0.0306
-mean/255	0.4425 ± 0.0360	DCT-mean/512	0.4385 ± 0.0432
		-mean DCT/std	0.4390 ± 0.0391
		-mean DCT/256	0.4460 ± 0.0434
		-mean DCT/512	0.4300 ± 0.0453

Table 3.6: Classification scores as the mean ± std % over 20 runs on MNIST dataset. The original data representation is compared to its DCT transform for different pre-processing techniques: "-" a constant or "mean" value represents subtraction of specified value from every image pixel, "DCT" stands for performing the discrete cosine transform at the particular step, and "/" with a constant or "std" refers to scaling the image by the specified value.

processing technique for original data was the subtraction of mean value and down-scaling by 128 with error 0.4245%, followed by the method that was reported in previous subsection on CIFAR10, subtracting 128 from pixels and scaling by 255, achieving 0.4355% error. The lowest average error of 0.4105% is obtained with DCT representation when subtracting 128 from the original image before performing the DCT and scaling the transformed data with 512 depicted as *center/max* pre-processing in Section 3.4.1. The difference in error in favor of the DCT representation is not significant, however, given noticeably well performing baseline network. There is therefore not much space to observe an improvement.

3.4.3 Implementation details

CNN-{A,B,C,D} are modeled and trained in Keras deep learning framework that uses TensorFlow backend. Models were trained on NVIDIA GTX770 GPU with 2GB of memory. The CNN-C network with roughly 750 thousand parameters, ∼21.5 million of multiply-add operations per image requires about 20 seconds of training per epoch. The full training takes less than 2 hours regardless of input representation, which is passed to the graphic

card in form of 32-bit precision floating point tensor. The DCT transform is computed via provided python implementation in opencv library. Time to transform the whole train set is platform dependent, on Intel processor with 3.7GHz frequency the transform with window size 2 applied to nearly 50 million image patches takes around 4 minutes. When using sizes 4 and 8, substantially less windows are processed leading to roughly 1 minute and 20 seconds long execution time respectively.

3.5 Using JPEG coefficients

Unlike in some related works [81, 95] we also investigate how to use DCT coefficients that are read directly from the JPEG file. Instead of performing the whole decompression pipeline, entropy encoded images undergo only entropy decoding process and dequantization, omitting the inverse DCT transform, a process performed when retrieving RGB channels from JPEG images. We investigate this approach on ImageNet dataset [96] that consists of much larger images compared to CIFAR datasets. Efficient algorithms for computing scaled 2D inverse DCT (IDCT) transform of an 8x8 block require at least 417 add or multiply/add floating point operations [97] to restore an image block. IDCT transform for an image in the ImageNet dataset with an average resolution 400×350 [96] would require 2.75 MFLOPS. Typical batch-size used for training models on ImageNet consists of 256 images requiring 0.7 GFLOPS to transform the whole batch in order to preprocess the images, which is often done using a less parallelizable CPU cores. The concept is related to work of Torfason et al. [98], which shows that compressed images have more abstract feature representation compared to original images and require less convolutions for classification. They however, instead of DCT coefficients, use images compressed by another neural network, a compressive convolutional autoencoder.

3.5.1 Data preparation

The JPEG format allows flexibility of color space and color channel subsampling scheme. In order for this information to be processed by convolutional filter, encoding of all images used for training and inference has to be unified with respect to these parameters. To do so, images encoded in other color space than YCbCr (for example CMYK) are recompressed in desired color format, using quantization matrices corresponding to 95% image quality. We enforce 4:4:4 chroma subsampling on all images as the majority of images in the ImageNet dataset are encoded without color channel subsampling.

3.5.2 Data augmentation

DCT coefficients are extracted directly from the files using the pysteg [99] python package. Upon reading, the coefficients are dequantized with quantization matrix read from the image file, hence the images can be compressed with different compression rates. A small

portion of the images have encoded only the luminance channel (greyscale images). Chroma channels of such images are filled with zeros. During the JPEG encoding process, the intensity value 128 is subtracted from all pixel intensities, therefore their DCT coefficients are expected to be centered around zero. Moreover, sign change of a coefficient affects the spatial structure of the whole underlying image in contrast with only a fluctuation in intensity for a single pixel in spatial domain. To normalize the data, each DCT coefficient is divided by its standard deviation calculated across the whole dataset, separately for each color channel.

An efficient way to enlarge the number of samples for training, the augmentation, is essential to prevent large models from overfitting. Training images in large scale databases come in variable resolutions. A common practice to augment data in ImageNet dataset is to resize an image to a certain size, extract random crop and reflect it horizontally with 0.5 probability [63]. Apart from geometrical transformations, some pipelines use color augmentation to build invariance to illumination [12].

We deploy augmentation directly on block-DCT image representation. It consists of random cropping that preserves entire DCT blocks, and random horizontal mirroring. Mirroring of a block-DCT image is performed in 2 steps: firstly, absolute position of blocks is flipped horizontally, secondly the individual blocks are reflected. A content of a DCT block can be reflected by multiplying its coefficients with matrix M (3.2).

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \tag{3.2}$$

In our experiments, random crops with mirroring were not found to be a sufficient augmentation. While using it, significant overfitting was still observed even for shallower models (e.g. 18 layers). Therefore we investigate using a scale augmentation to increase variability in the data. We have adopted scale and aspect ratio augmentation as described in [60] (followed by random mirroring). Crop area is sampled uniformly from range $\mathcal{U}(8\%, 100\%)$ of the whole image area. Likewise, aspect ratio is sampled from uniform distribution $\mathcal{U}(3/4, 4/3)$. The only constraint is that both, crop width and height is an integer divisible by 8 (JPEG block size). Selected crop area is extracted from random position in the image while not disrupting the DCT blocks. Finally, the crop is resized to desired size.

To resize images with standard interpolation is not possible in DCT domain, hence we have

considered two custom approaches to resize a block DCT image. The first approach [100] proposes to resize an image performing n-point inverse DCT and subsequent m-point forward DCT and coefficient scaling, omitting the need for interpolation in pixel space. When m or n are different from 8, coefficients are either truncated or zero-padded to match desired size. The scale factor is determined by ratio n/m. The limitation of this approach is that scale factor is limited to a small finite set of integer pairs n, m. An arbitrary scaling would require, in extreme case, padding every block to match output image resolution for n and input image resolution for m. Lastly, initial experiments show lower quality of resized images and worse classification results compared to resizing based on interpolation. We have resorted to the second option, using linear interpolation in spatial domain after 8-point inverse DCT and following 8-point forward DCT. The DCT transform is performed by convolving an image with a set of $64, 8 \times 8$ filters (one filter per DCT basis), resulting in image tensor of shape $(64c, h/8, w/8)$, for image of size $h \times w$ with c channels. The original shape of an image is restored by pixel reorganization.

3.5.3 ILSVRC data set

One of the most popular benchmarks for image recognition is provided by the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [101]. The training set is a subset of the ImageNet database [96] composed of 1.28 million images of real wold objects. Performance of the competing models is determined by 50k validation and 150k testing images, which are to be assigned to one of the 1000 possible classes. Images have variable size and are stored in JPEG format.

Figure 3.7: A sample image from ImageNet dataset.

3.5.4 Experimental Setup on Imagenet

Due to architectural constrains imposed in Section 3.3, the choice of a baseline architecture is restricted to models that substantially downsample spatial resolution in early stage convolutional layers. Popular architectures such as VGG [59] or SqueezeNet [102] rely on high resolution feature maps, their adaptation for block DCT data would significantly alter their architecture. A suitable candidate for architecture adaptation is found in ResNet architecture family [63]. It is composed of simple building blocks and shortcut connections. An

layer name	output size	ResNet-18/Naive DCT	dctResNet-14	dctResNet-18	dctResNet-18-ups
conv1	112×112	7×7, 64, stride 2	none		
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	none		8×8, 64, stride 8 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	8×8, 128, stride 8 $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times4$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$
	1×1	average pool, 1000-d fc, softmax			
# of parameters		11.7×10^6	11.6×10^6	14.0×10^6	11.7×10^6
FLOPs		1.81×10^9	1.30×10^9	1.77×10^9	1.73×10^9

Table 3.7: Structure of modified ResNet architectures with respect to the baseline.

input image, when processed by ResNet, is downsampled by factor 4 after the first convolution and pooling layer. Generic architecture of ResNet is composed of root convolutional layer followed by max pooling and a set of residual blocks. There are sets of residual blocks at 4 spatial feature resolutions. All blocks are designed to have the same computational cost, but different memory requirements. For standard crop size 224 $\times$ 224, the first set of blocks is processing features of size 56 $\times$ 56. However, crop of this size consists only of 28 $\times$ 28 DCT blocks, which would constitute the largest possible feature size. Various alternations of the ResNet baseline are considered for modeling blockwise DCT data. Each instance of our architecture is detailed in Table 3.7. The dctResNet-14 skips residual blocks that operate on the first spatial resolution (56 $\times$ 56). dctResNet-18 instead compensates missing depth of the model by increasing the number of residual blocks at lower spatial resolution (14 $\times$ 14) following the same recipe as Torfason et al. [98]. Training one such model is time demanding, thus we resort to comparisons with a single instance of particular model architectures.

3.5.5 Training details

Most of the hyperparameters used in the training procedure follow the original ResNet design by He et al. [63]. Models are trained by stochastic gradient descent with nesterov momentum having initial learning rate 0.1. To speed up the training, the learning rate is decayed 3.75$\times$ faster than in the original schedule, by factor 10 after 8, 16 and 24 epochs, trained for total length of 28 epochs as in Torfason et al. [98]. Gradients are calculated from batches of 256 image crops of size 224$\times$224, augmented as described in Section 3.5.2. Models and the training procedure are implemented in PyTorch framework, adopting official ImageNet example training code [103]. PyTorch framework was chosen for this experimentation due to simple parallelism implementation and the use of dynamic computational graph that was more memory efficient in our experiments. Training is parallelized on 2 Nvidia Titan X (Pascal) GPUs, by splitting the batch.

3.5.6 Results

Standard ResNet-18 model naively trained on block-DCT data extracts low-level features (Figure 3.8) that do not resemble the original image (Figure 3.7). Nevertheless, the model can find patterns in such representation, but gives poor performance. Adapting the input layer for block-DCT representation gives better results (see Table 3.8) despite having fewer layers. However, ignoring blocks at feature size 56×56 diminishes the performance, compared to the baseline, by almost 3%. This "trimmed" ResNet has only 14 layers and the loss of model complexity has certainly affected the performance. Activation maps at the output of the first layer are small and lack detail (Figure 3.9), which may result in overlooking some of the important features.

Figure 3.8: ResNet-18 first layer activation maps inferred from the sample image in block-DCT format (**left**) without any adjustment and features of another model gathered from the same image in RGB format (**right**). Both models are trained on images in their corresponding formats.

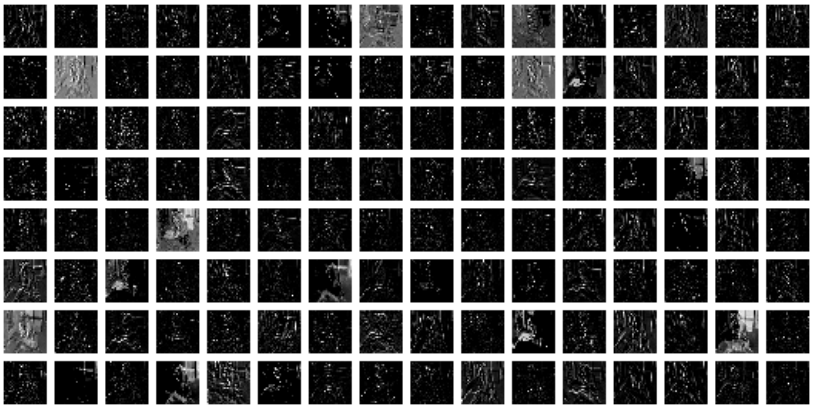


Figure 3.9: dctResNet-14 first layer activation maps inferred from the sample image.

The 18-layered model with the missing 2 blocks inserted deeper in the network narrows the gap but still falls short by over 1% (see Table 3.8) and increases the parameter burden

(Table 3.7). The dctResNets do not have a maxpooling layer and as a result the receptive field of the first layer is reduced. Increasing the filter size to 3 DCT blocks (size 24×24) does not provide any improvement. Another approach how to compensate the missing blocks and higher feature resolution is to deploy the subpixel convolution [104]. To upsample the signal 2 times in each spatial dimension, 4-times more filters are trained in the first layer. The high dimensional feature maps are created by rearranging the lower dimensional outputs of the filter bank. We let the first layer learn 256 8×8 filters and leave the rest of the network as in the baseline model. This method does not surpass performance of dctResNet-18.

model	top-1 ↑	top-5 ↑
ResNet-18	**65.98**	**86.74**
ResNet-18 Naive DCT	61.76	84.00
dctResNet-14	63.20	84.87
dctResNet-18	64.78	85.88
dctResNet-18 (24×24)	64.43	85.72
dctResNet-18 subpixel	64.48	85.88

Table 3.8: Accuracy of ResNet18 models trained on DCT representation.

Instead of resizing the features we focus on larger crop size. Crops of 448×448 consist of 56×56 DCT blocks, which becomes the feature resolution after the first convolutional layer. Due to large stride of the first layer, this expansion has almost no impact on computational complexity of the model. For a fair comparison the baseline network is also adjusted for larger crop size. In the baseline model we can either increase size of the pooling window or stride of the convolutional layer. For a good performance, multiple kernel sizes are tested and the best variant is 9×9 with stride 4. It can be seen in Table 3.9 that accuracy of this model is still behind the model trained on the DCT representation. The improvement is given by data representation rather than model architecture. Identical architecture does not perform as well on RGB data (see Table 3.9). Features of the dctResNet at this crop size contain more details, but the network prefers more high-frequency features (Figure 3.10).

Table 3.10 extends the comparison by deeper ResNet-50 network. As a baseline model on 448×448 RGB crops, kernel size 9×9 and stride 4 is selected. The deeper model trained on DCT data did not surpass the RGB counterpart. Using similar problem formulation, results comparable to the original model were achieved in [25] by increasing the number of features in replaced blocks. This adjustment, although breaches the performance gap, it increases the number of parameters and computational complexity (FLOPs).

Color separation

Images with chroma subsampling need particular consideration. Color channels of such images have lower resolution compared to the luminance channel. Assuming crop size

input	filter/stride	maxpool/stride	crop size	top-1 ↑	top-5 ↑
RGB	7/2	3/2	224	65.98	86.74
RGB	7/2	6/4	448	66.15	**87.04**
RGB	7/4	3/2	448	66.07	87.02
RGB	9/4	3/2	448	**66.34**	87.03
RGB	11/4	3/2	448	66.17	87.08
RGB	14/4	3/2	448	66.12	86.80
RGB	8/8	-	448	65.90	86.92
DCT	8/8	-	448	**66.56**	**87.36**

Table 3.9: Accuracy of ResNet-18 models adjusted for 448×448 crops and comparison with DCT input at this resolution.

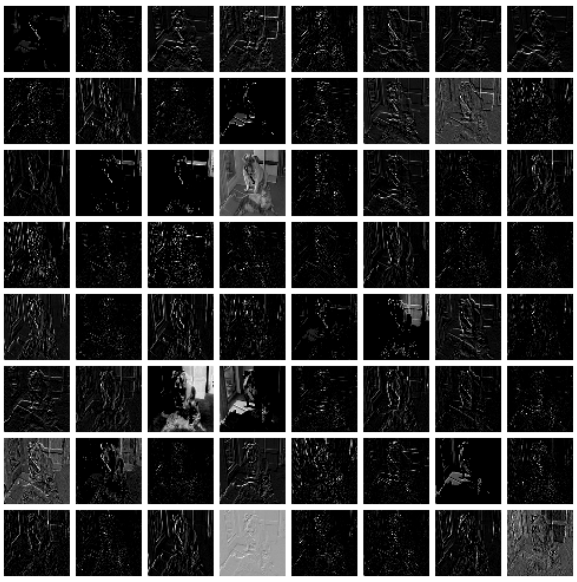


Figure 3.10: dctResNet-18-ups first layer activation maps inferred from the sample image at crop size 448.

224×224, features gathered from subsampled color channels could be at most 14×14. The most natural way to include this information is to mix it with luminance features further along the network when their resolution matches. Color can be considered as higher level information that is not required for shape classification, but may provide valuable semantic information in further stages. This approach is known from the literature as the late fusion of shape and color information and has been successfully used in semantic recognition and object detection tasks [105, 106, 107]. In ILSVRC2012 dataset most images do not use chroma subsampling. It is still worth investigating whether separating color and intensity information might prove to be useful or if less transformations are needed for rather crude color channels. We experiment with dctResNet-50 that has 4 residual blocks with 28×28 features. Some of these blocks are replaced with residual blocks used in separated streams. The streams are aggregated and the remaining (if any) blocks are applied to the combined information such that luma features always pass through 4 blocks. Aggregation through concatenation and summation is tested. In the first case 96 features are used for luma

layers	18				50			
crop size	224		448		224		448	
accuracy	top-1 ↑	top-5 ↑	top-1 ↑	top-5 ↑	top-1 ↑	top-5 ↑	top-1 ↑	top-5 ↑
RGB	**65.98**	**86.74**	66.34	87.03	**71.16**	**90.26**	**71.90**	**90.63**
DCT	64.78	85.88	**66.56**	**87.36**	70.16	89.59	71.63	90.54

Table 3.10: Accuracy of ResNet-18 and ResNet-50 models trained on RGB channels compared to DCT representation at different crop sizes.

channel and 32 for chroma channels. This separation saves at least 37.5% of parameters and computations. When summation is used no spares can be achieved. The equal number of channels in both streams would duplicate the parameter and computation requirements. Results are depicted in Table 3.11.

separation	top-1 ↑	top-5 ↑
no separation	70.16	89.59
concat 2-0	70.28	89.60
concat 4-0	69.54	89.19
concat 2-2	70.34	89.67
concat 4-2	70.06	89.67
concat 4-4	**70.55**	**89.69**
sum 2-0	70.34	89.63

Table 3.11: Accuracy of ResNet-50 models with separate brightness and color streams. The models are described as "aggregation *a*-*b*" where *a* is the number of residual blocks used only for luma features, *b* for chroma features and aggregation is the way the streams are combined.

Separating intensity and color channels seems to bring only a mild improvement, nevertheless reduces parametric and computational complexity without a loss of accuracy. Using no residual blocks to learn from chromatic channels at resolution 28×28 gives the worst performance. The probable reason is that some features in the chromatic channels rely on detail that is lost in lower resolutions.

3.6 Conclusion

In this chapter we have presented an approach for adopting convolutional networks to learn from frequency representations, with motivation to deploy this approach to compressed image data. We showed empirically that low level features learned from window based discrete cosine transform coefficients are comparably or more discriminative than those learned from standard data representation. High level feature analysis shows deeper networks trained on data transformed by DCT learn similar representations and make similar mistakes as their spatial domain counterparts. Furthermore, we have trained convolutional network on DCT coefficients extracted directly from JPEG files of high resolution images. Benefits of color

and intensity separation in the classification network are also demonstrated. Performance of models trained on JPEG coefficients is dependant on the input crop size due to DCT block size that determines subsampling of the first convolutional layer. We will address this issue in the following chapter by performing DCT transform with overlaps directly in the network's computational graph.

Chapter 4

DCT based Harmonic Networks

4.1 Introduction

CNNs have been designed to take advantage of implicit characteristics of natural images, specifically correlation in local neighborhood and feature equivariance. The wide application of features obtained by convolving images with explicitly defined local filters highlights the shift from the extraction of global information towards local learning.

Standard CNNs rely on the learned convolutional filters that allow them to be adjusted flexibly to the data available. In some cases, however, it may be advantageous to revert to preset filter banks. For instance with limited training data, using a collection of preset filters can help in avoiding overfitting and in reducing the computational complexity of the system. An example of such networks with preset (wavelet based) filters is the scattering network, which achieved outstanding results in handwritten digit recognition and texture classification [27].

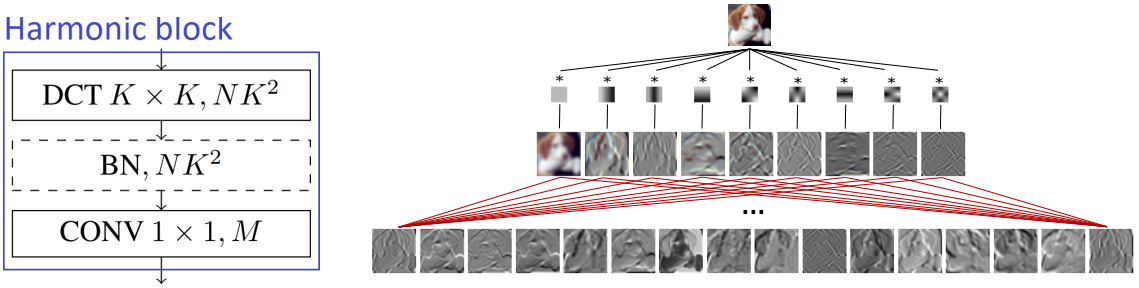


Figure 4.1: Left: Design of the harmonic block. Boxes show operation type, size of the filter (if applicable) and the number of output channels given the block filter size K, number of input channels N and output channels M. Batch normalization (BN) block is optional. Right: Visualization of the harmonic block applied to an input layer.

We propose[1] instead to replace the standard convolutional operations in CNNs by harmonic blocks that learn the weighted sums of responses to the Discrete Cosine Transform

[1] Part of this chapter has been published in [108, 109].

(DCT) filters, see Figure 4.1. DCT has been successfully used for JPEG encoding to transform image blocks into spectral representations to capture the most information with a small number of coefficients [21]. Motivated by frequency separation and energy compaction properties of DCT, the proposed harmonic networks rely on combining responses of window-based DCT with a small receptive field. Our method learns how to optimally combine spectral coefficients at every layer to produce a fixed size representation defined as a weighted sum of responses to DCT filters. The use of DCT filters allows one to easily address the task of model compression. While other works that propose convolutional filters decomposition to particular basis functions [74, 110] have mostly focused on ability to compress the network, we show furthermore that prior information coming from well chosen filter basis can not only be used for compression but can also speed up training convergence and improve performance.

The relevant background is first presented (Section 4.2) and our harmonic network formulation is then explained in Section 4.3. Our proposed harmonic block acts as an elementary unit to encode any neural network. Evaluation of the block is conducted by assessing effect on the performance and parametric complexity of well established architectures when replacing their convolutional layers with harmonic blocks. The proposed models are extensively validated against state-of-the-art alternatives solving a representative set of problems in computer vision that are often addressed by CNNs, image classification (Section 4.4), object detection and segmentation (Section 4.5.1), and finally boundary detection in images (Section 4.5.5). All our architectures are distinct in our reported results with the name starting with *Harm*. The Chapter is concluded with some final remarks (Section 4.6). The PyTorch implementations for our harmonic networks are publicly available at `https://github.com/matej-ulicny/harmonic-networks`.

4.2 Related work

This section provides an overview of related works combining DCT and CNN with application to object classification, works that use other basis functions in CNNs, and will briefly outline works using DCT for CNN compression.

4.2.1 Wavelets & CNNs

Wavelet pre-processing

As an alternative to DCT, scattering Networks (see Chapter 2.4) are built on complex-valued wavelets, for instance Morlet [27] or dual-tree wavelet [111]. The scattering network based on rotation and scale invariant wavelet transform was shown to effectively reduce the input representation while preserving discriminative information for training CNN on image classification [112, 113] and object detection task [114] achieving performance comparable to deeper models. Williams et al. [115] have advocated image pre-processing with wavelet

transform, but used different CNNs for each frequency subband. Wavelet filters were also used as a pre-processing method prior to NN-based classifier [116], and to enhance edge information in images prior to classification [117].

Spectral based CNNs

Other works have used wavelets in CNN computational graphs. Second order coefficients from Fast Wavelet Transform were used [118] to design a wavelet pooling operator. Similar approach was taken by Ripperl et al. who designed spectral pooling [28] based on Fast Fourier Transform of the features and truncation of the high-frequency coefficients. They also proposed to parameterize filters in the Fourier domain to decrease their redundancy and speed up the convergence when training the network. In both works, the pooled features were recovered with Inverse Fast Wavelet or Inverse Discrete Fourier Transform respectively, thus the CNN still operates in the spatial domain.

To address texture classification, Fujieda et al. [119] proposed a Wavelet Convolutional Network that is trained on responses to Haar wavelets and concatenates higher-order coefficient maps along with features of the same dimensionality learned from lower-order coefficients. Similar approach is taken by Lu et al. [120] that learns from both spatial and spectral information that is decomposed from first layer features. The higher-order coefficients are also concatenated along with the lower dimensional feature maps. However, contrary to our harmonic networks, Wavelet CNNs decompose only the input features and not the features learned at intermediate stages.

Robustness to object rotations was addressed by modulating learned filters by oriented Gabor filters [30]. Worrall et al. incorporated complex circular harmonics into CNNs to learn rotation equivariant representations [29]. Similarly to our harmonic block, the structured receptive field block [26] learns new filters by combining fixed filters, a set of Gaussian derivatives with considerably large spatial extent. Additionally, orthogonalized Gaussian derivative bases of small spatial extend have been used by Kobayashi to express convolutional filters [73]. DCFNet [74] expresses filters by truncated expansion of Fourier-Bessel bases, maintaining accuracy of the original model while reducing the number of parameters.

Following our approach Ehrlich et al. [121] has proposed to model CNN operations in JPEG transform domain, including approximating the ReLU activation function. Further extensions include learning the frequency of cosine function that constitutes a part of the filter bank [122].

4.2.2 Compressing DNNs

Numerous works have focused on compressing the size of neural networks and decreasing the inference and training time. Speedup and memory saving for inference can be achieved

by approximating the trained full-rank CNN filters by separable rank-1 filters [123]. It has been shown [124] that a model complexity can be adjusted during the training time: increased via introduction of new filters by rotating and applying noise to existing ones, and reduced by clustering to selectively decrease their redundancy.

Assuming smoothness of learned filters, Frequency-Sensitive Hashed Network (FreshNet) [125] expresses filters by their DCT representation and groups their parameters to share the same value within each group. Wang et al. [126] relaxes this constrain to express each weight by its residual from the cluster center. Weights in this form were quantized and transformed via Huffman coding (used for JPEG compression) for limiting the storage. Convolution was performed in the frequency domain to reduce the computational complexity. Han et al. [127] compressed networks by pruning, clustering and quantizing weights, which are consequently fine-tuned. These approaches however perform full compression pipeline to minimize storage footprint, while our focus will be on parameter count.

4.3 Harmonic Networks

A convolutional layer extracts correlation of input patterns with locally applied learned filters. The idea of convolutions applied to images stems from the observation that pixels in local neighborhoods of natural images tend to be strongly correlated. In many image analysis applications, transformation methods are used to decorrelate signals forming an image [66]. In contrast with spatial convolution with learned kernels, this study proposes feature learning by weighted combinations of responses to predefined filters. The latter extracts harmonics from lower-level features in a region. The use of well selected predefined filters allows one to reduce the impact of overfitting and decrease computational complexity. In this work we focus on the use of DCT as the underlying transformation.

4.3.1 Harmonic blocks

A harmonic block is proposed to replace a conventional convolution operation and relies on processing the data in two stages (see Figure 4.1). Firstly, the input features undergo harmonic decomposition by a transformation method. Conceptually, various transformation methods can be used e.g. wavelets, derivatives of Gaussian, etc. In this study we focus on window-based DCT. In the second stage, the transformed signals are combined by learned weights. The fundamental difference from standard convolutional network is that the optimization algorithm is not searching for filters that extract spatial correlation, rather learns the relative importance of preset feature extractors (DCT filters) at multiple layers.

Harmonic blocks are integrated as a structural element in the existing or new CNN architectures. We thus design harmonic networks that consist of one or more harmonic blocks and, optionally, standard convolution and fully-connected layers, as well as any other structural elements of a neural net. Spectral decomposition of input features into block-DCT

representation is implemented as a convolution with DCT basis functions. A 2D kernel with size $k \times k$ is constructed for each basis function, comprising a filter bank of depth k^2, which is separately applied to each of the input features. Convolution with the filter bank isolates coefficients of DCT basis functions to their exclusive feature maps, creating a new feature map per each input channel and each frequency considered. The number of operations required to calculate this representation can be reduced by decomposing 2D DCT filter into two rank-1 filters and applying them as separable convolution to rows and columns sequentially. We do not implement this separation because computational savings associated with it are minimal for small filters that are used in vast majority of CNNs. Separating the convolution would also have a negative impact on training memory. Despite the convolution with DCT filters being computationally cheaper compared to dense convolutions, the spectral decomposition upsamples the intermediate representation by k^2 factor, thus notably increasing the corresponding training memory requirements.

Each feature map $\mathbf{h}^l$ at depth l is computed as a weighted linear combination of DCT coefficients across all input channels n:

$$\mathbf{h}^l = \sum_{i=0}^{n-1} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w^l_{i,u,v} \boldsymbol{\psi}_{u,v} * \mathbf{h}^{l-1}_i \tag{4.1}$$

where $\boldsymbol{\psi}_{u,v}$ is a u, v frequency selective DCT filter of size $k \times k$, $*$ the convolution operator and $w^l_{i,u,v}$ is learned weight for u, v frequency of the i-th feature. The linear combination of spectral coefficients is implemented via a convolution with 1×1 filter that scales and sums the features, see Figure 4.1. Since the DCT is a linear transformation, backward pass through the transform layer is performed similarly to a backward pass through a convolution layer. Harmonic blocks are designed to learn the *same* number of parameters as their convolutional counterparts. Such blocks can be considered a special case of depth-separable convolution with predefined spatial filters.

DCT is distinguished by its energy compaction capabilities which typically results in higher filter responses in lower frequencies. The undesirable behaviour of relative loss of high frequency information can be efficiently handled by normalizing spectrum of the input channels. This can be achieved via batch normalization that adjusts per frequency mean and variance prior to the weighted combination. The spectrum normalization transforms Eq. (4.1) into:

$$\mathbf{h}^l = \sum_{i=0}^{n-1} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w^l_{i,u,v} \frac{\boldsymbol{\psi}_{u,v} * \mathbf{h}^{l-1}_i - \mu^l_{i,u,v}}{\sigma^l_{i,u,v}}, \tag{4.2}$$

with parameters $\mu^l_{i,u,v}$ and $\sigma^l_{i,u,v}$ estimated per input batch.

4.3.2 Harmonic Network Compression

The JPEG compression encoding relies on stronger quantization of high-frequency DCT coefficients. This is motivated by the human visual system which often prioritises low frequency information over high frequencies [23]. We propose to employ similar idea in the harmonic network architecture. Specifically, we limit the visual spectrum of harmonic blocks to only several most informative low frequencies, which results in a reduction of number of parameters and operations required at each block. The coefficients are (partially) ordered by their relative importance for the visual system in triangular patterns starting at the most important zero frequency at the top-left corner, see Figure 4.2. We limit the spectrum of considered frequencies by hyperparameter λ representing the number of levels of coefficients included perpendicularly to the main diagonal direction starting from zero frequency: DC only for $\lambda = 1$, three coefficients used for $\lambda = 2$, and six coefficients used for $\lambda = 3$. Figure 4.2 illustrates filters used at various levels assuming a 3×3 receptive field. Thus,

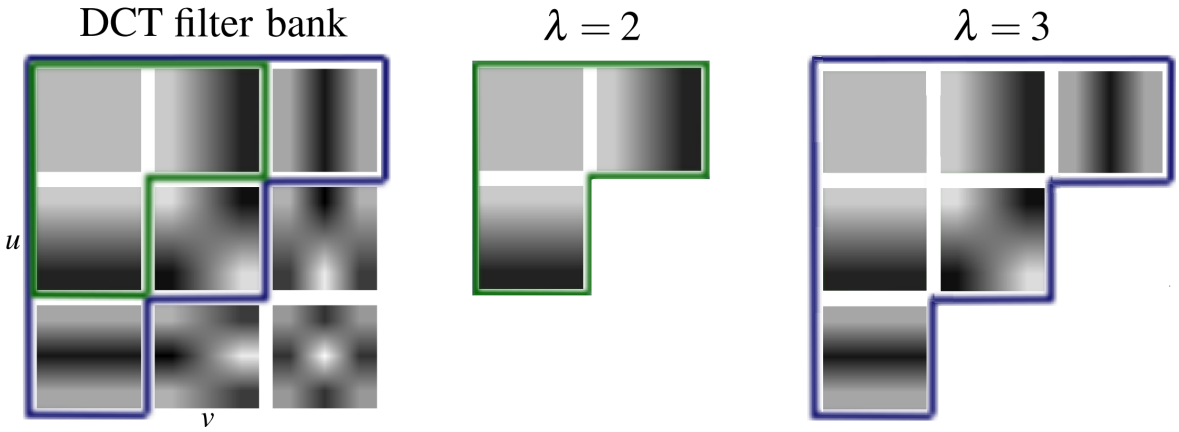


Figure 4.2: 3×3 DCT filter bank employed in the harmonic networks and its compression.

reformulating convolutional layers as harmonic allows one to take advantage of this natural approach to model compression, and in doing also introduce additional regularization into the model. Limiting certain frequencies used for filter representation is however dependent on the task at hand. Some problems e.g. texture analysis require high-frequency variability and restricting it can negatively affect the performance. The empirical impact of harmonic model compression will be investigated in more detail in the experiments below.

4.3.3 Overlapping cosine transform

DCT computed on overlapping windows is also known as Lapped Transform or Modified DCT (MDCT), related to our harmonic block using strides. The overlapped DCT has a long history in signal compression and reduces artefacts at window edges [128]. Dedicated strategies for efficient computations have been proposed [128], including algorithms and hardware optimisations.

DCT transform is equivalent to the discrete Fourier transform of real valued functions with even symmetry within twice larger window. DCT lacks imaginary component given by the

sine transform of real valued odd functions. However, harmonic block allows convolution with DCT basis with an arbitrary stride, creating redundancy in the representation. Ignoring the boundary limitations, sine filter basis can be devised by shifting the cosine filters. Given the equivariant properties of convolution, instead of shifting the filters the same result is achieved by applying original filters to the shifted input. Considering DCT-II formulation for 1D signal $\mathbf{x}$ with n values, the DCT coefficient at frequency u is:

$$f_u = \sum_{a=0}^{n-1} x_a \cos\left[\frac{\pi}{n}\left(a + \frac{1}{2}\right) u\right] \tag{4.3}$$

a corresponding sine transform is

$$g_u = \sum_{a=0}^{n-1} x_a \sin\left[\frac{\pi}{n}\left(a + \frac{1}{2}\right) u\right] \tag{4.4}$$

which is equivalent to

$$g_u = \sum_{a=0}^{n-1} x_a \cos\left[\frac{\pi}{2} + 2\pi z - \frac{\pi}{n}\left(a + \frac{1}{2}\right) u\right]. \tag{4.5}$$

The shift given by $\pi/2 + 2\pi z$ for any $z \in \mathbb{Z}$ can be directly converted to a shift in pixels applied to data X. After simplification, sine transform can be expressed as

$$g_u = \sum_{a=0}^{n-1} x_a \cos\left[\frac{\pi}{n}\left(a - \frac{n(1+4z)}{2u} + \frac{1}{2}\right) u\right] \tag{4.6}$$

which is equivalent to the cosine transform of the image shifted by $\delta = n\,(1+4z)\,/2u$ defined in (4.7).

$$f_u[\delta] = \sum_{a=0}^{n-1} x_{a+\frac{n(1+4z)}{2u}} \cos\left[\frac{\pi}{n}\left(a + \frac{1}{2}\right) u\right]. \tag{4.7}$$

This value represents the stride to shift the cosine filters to capture correlation with sine function. A visualization of a concrete example in one dimension can be found on Figure 4.3.

In other words, by applying DCT with a certain stride it is possible to obtain the feature representation as rich as that obtained with the full Fourier transform.

The harmonic block can also be easily applied with dilations in form of à-trous convolution [129]. Such convolution is performed with a filter that has spacing between its components. Motivation for this approach is to enlarge receptive field of filters without increasing computational cost or decimating signals by stride or pooling. Should dilation be used in harmonic blocks, spacing is inserted between components of the DCT bases as in ordinary filters. We validate this type of harmonic block in Section 4.5.4.

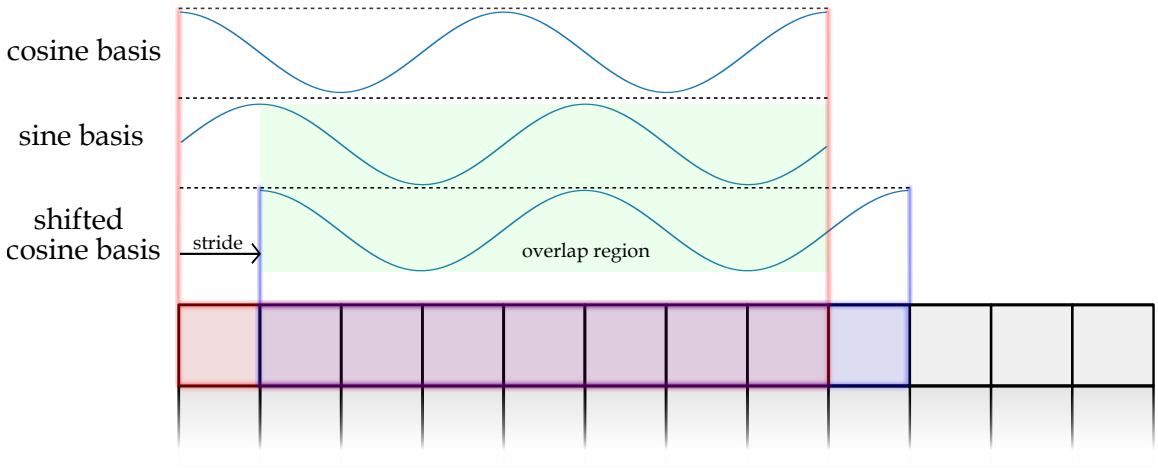


Figure 4.3: Visualization of 1D cosine basis on overlapping image regions. Application of the basis to image regions is highlighted with red and blue colors. In this example the basis length n=8 and frequency u=4 constitute that cosine basis shifted by 1 pixel will become the same function on the region of overlap as the sine basis in the original location. The area where the bases are identical is highlighted with the green color.

4.3.4 Computational Requirements

Harmonic blocks are designed to learn the same number of parameters as their convolutional counterparts. Requirements for the DCT transform scale linearly with the number of input channels and result in a modest increase to the theoretical number of operations. Standard convolutional layer used in many popular architectures that has n input and m output channels with a kernel size $k \times k$ learns nmk^2 parameters and performs nmk^2ab operations if the filter is applied a and b times in particular directions. Harmonic block with k^2 transformation filters of size $k \times k$ upsamples the representation to nk^2 features and then learns one weight for each upsampled to output feature pair hence nk^2m weights. Transform of an $a \times b$ feature set costs nk^2k^2ab on top of the weighted combination nk^2mab that matches the number of multiply-add operations of $k \times k$ convolution. The total number of operations is thus $nk^2ab\,(m + k^2)$. The theoretical number of multiply-add operations over the standard convolutional layer increases by a factor of k^2/m. If we assume a truncated spectrum (use of $\lambda \leq k$) given by $p = \lambda(\lambda + 1)/2$ filters, the proportion of operations becomes $p/k^2 + p/m$.

While keeping the number of parameters intact, a harmonic block requires additional memory during the training and inference to store transformed feature representation. In our experiments with WRN models (Section 4.4.2), the harmonic network trained with full DCT spectrum requires almost 3 times more memory than the baseline. This memory requirement can be reduced by using the DCT spectrum compression.

Despite the comparable theoretical computational requirements, the run time of harmonic networks is larger compared to the baseline models, at least twice slower (on GPU) in certain configurations. This effect is due to the design of harmonic block that replaces a single convolutional layer by a block of 2 sequential convolutions (with individual harmonic filters

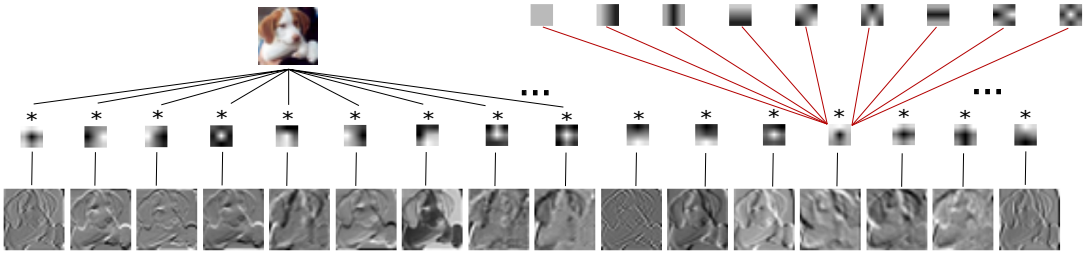


Figure 4.4: Design of the memory efficient harmonic block. Red lines show the learnable weights that combine DCT bases into effective filters used to convolve the input.

and 1x1 convolution). Most blocks do not need BN between the convolutions and thus represent a combined linear transformation. The associativity property of convolutions allows one to reformulate the standard harmonic block defined above so that the DCT transform and linear combination can be effectively merged into a single linear operation:

$$\mathbf{h}^l = \sum_{i=0}^{n-1} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w_{i,u,v} \left(\psi_{u,v} * \mathbf{h}_i^{l-1} \right) = \sum_{i=0}^{n-1} \left(\sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w_{i,u,v} \psi_{u,v} \right) * \mathbf{h}_i^{l-1} \tag{4.8}$$

In other words, equivalent features can be obtained by factorizing filters as a linear combination of DCT basis functions (see visualization on Figure 4.4). We thus propose a faster Algorithm 1 that is a more memory efficient alternative to the standard two-stage harmonic block formulation.

Algorithm 1: Memory efficient harmonic block

Input: $\mathbf{h}^{l-1}$
Define $\mathbf{g} \in \mathbb{R}^{m \times n \times k \times k}$;
for $j \in \{0..m-1\}$ **do**
  for $i \in \{0..n-1\}$ **do**
    $\mathbf{g}_{j,i}^l \leftarrow \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w_{j,i,u,v} \ \psi_{u,v}$;
  end
end
$\mathbf{h}^l \leftarrow \mathbf{g}^l * \mathbf{h}^{l-1}$;
Output: $\mathbf{h}^l$

This reformulation is similar to structured receptive field [26] utilizing another basis functions. The Algorithm 1 overhead in terms of multiply-add operations with respect to the standard convolutional layer is only k^2/ab, where the input image size for the block is $a \times b$. This overhead becomes negligible as the image size increases. The experimental performance of the algorithm is evaluated in Section 4.4.2.

4.4 Image Classification

The performance of the harmonic networks is assessed in image classification on datasets of various sizes from small to large. The method is validated on a set of synthetic and

natural images, on which we also demonstrate several interesting properties of harmonic blocks. Firstly we focus on a synthetic dataset of toys, the small NORB in Section 4.4.1. Its controllable lighting conditions and uniform background makes it useful resource for evaluating lighting invariance properties of harmonic blocks, or model compressibility due to the lack of high-frequency information. We then focus on a set of small images of natural scenes, CIFAR-10 and CIFAR-100 in Section 4.4.2. Apart from demonstrating comparable overall classification accuracy to the baseline model we demonstrate here that high-frequency truncation can also efficiently compress models trained to classify natural images. Moreover, the section shows benefits of overlapping the DCT blocks and robustness of harmonic networks to certain types of noise and shed some light into their convergence speed. The following subsection (Section 4.4.3) looks at the generalization properties of harmonic networks when trained on a limited sample set. Subsets of synthetic (MNIST) and natural (CIFAR-10, STL-10) images are considered in the evaluation. In Section 4.4.4 demonstrates successfulness in the opposite scenario when the amount of data is abundant (ImageNet-1K) by outperforming the baseline as well as many other high-end models. The subsection also focuses on reusing existing models by basis conversion. Lastly, we implement the harmonic block with another sets of basis functions and compare their performance on two representative datasets of natural images (Section 4.4.5). We consider a simple two to three-layer CNNs for the NORB, MNIST datasets, and two typologies of Residual Networks [63, 90] for other datasets as the baselines for substituting the standard convolution operations with harmonic blocks.

4.4.1 Synthetic images

The small NORB dataset [1] was intended for a task of 3D object recognition from its shape. It is a synthetic set of 96 $\times$ 96 binocular images of 50 toys sorted into 5 classes (four-legged animals, human figures, airplanes, trucks, and cars), captured under different lighting and pose conditions (i.e. 18 different angles, 9 elevations and 6 lighting conditions induced by combining different light sources, see example images in Figure 4.5). Training and test sets used in our experiments are retained original [1][2].

We show first that harmonic networks outperform standard and state-of-the-art CNNs in both accuracy and compactness (c.f. Section 4.4.1) and also illustrate how Harmonic networks can be naturally resilient to unseen illumination changes without resorting to using data augmentation (Section 4.4.1).

Comparisons CNN vs. Harmonic Nets

Baseline architectures. The baseline CNN network is selected after an extensive hyper-parameter search. The network (CNN2) consists of 2 convolution layers with 32 5$\times$5 and 64 3$\times$3 filter banks respectively, a fully connected layer with 1024 neurons followed

[2] `https://cs.nyu.edu/~ylclab/data/norb-v1.0-small/`

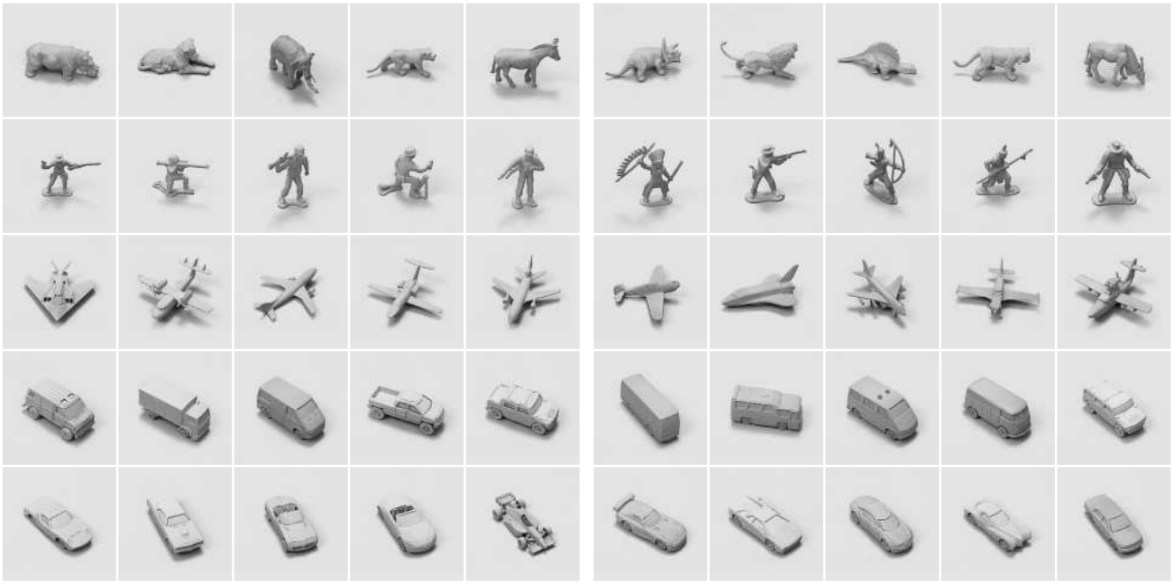


Figure 4.5: Sample images from small NORB showing all 5 instances of each of the 5 categories in training (**left**) and testing (**right**) sets [1].

by a softmax classifier. Filters are applied with stride 2 and features are subsampled by overlapping max-pooling. All hidden layer responses are batch normalized and rectified by ReLU. We also use a slightly deeper network (CNN3) with an additional convolutional layer preceding the first pooling. Details of the architectures are summarised in Table 4.1.

Optimisation. The baseline CNNs are trained with stochastic gradient descent for 200 epochs with momentum 0.9. The initial learning rate 0.01 is decreased by factor 10 every 50 epochs. The network is trained with batches of 64 stereo image pairs and each pair is padded with zeros 5 pixels on each side and a random crop of 96×96 pixels is fed to the network. The optimization is regularized by dropout (p=0.5) on the fully connected layer and the weight decay is set to 0.0005.

Table 4.1: Models used in NORB experiments. Convolution and harmonic operations are denoted as {conv, harm} m, k×k/s with m output features, kernel size k and stride s; similarly for pooling k×k/s and fully connected layers fc m.

Resolution	CNN2	CNN3	Harm-Net 2	Harm-Net 3	Harm-Net 4
96x96	conv 32, 5x5/2	conv 32, 5x5/2	harm 32, 4x4/4	harm 32, 4x4/4	harm 32, 4x4/4
48x48	pool 3x3/2	conv 64, 3x3/2	-	-	-
24x24	conv 64, 3x3/2	pool 3x3/2	harm 64, 3x3/2	harm 64, 3x3/2	harm 64, 3x3/2
12x12	pool 3x3/2	conv 128, 3x3/2	pool 3x3/2	pool 3x3/2	pool 3x3/2
6x6	fc 1024	pool 3x3/2	fc 1024	harm 128, 3x3/2	harm 128, 3x3/2
3x3	-	fc 1024	-	fc 1024	harm 1024, 3x3/3
1x1	dropout 0.5	dropout 0.5	dropout 0.5	dropout 0.5	dropout 0.5
1x1	fc 5	fc 5	fc 5	fc 5	fc 5

Harmonic Networks architectures. Several versions of harmonic networks are considered (Table 4.1), by substituting the first, first two or all three of CNN2 and CNN3 convolution layers by harmonic blocks. Furthermore, the first fully-connected layer can be transformed to a harmonic block by taking a global DCT transform of the activations

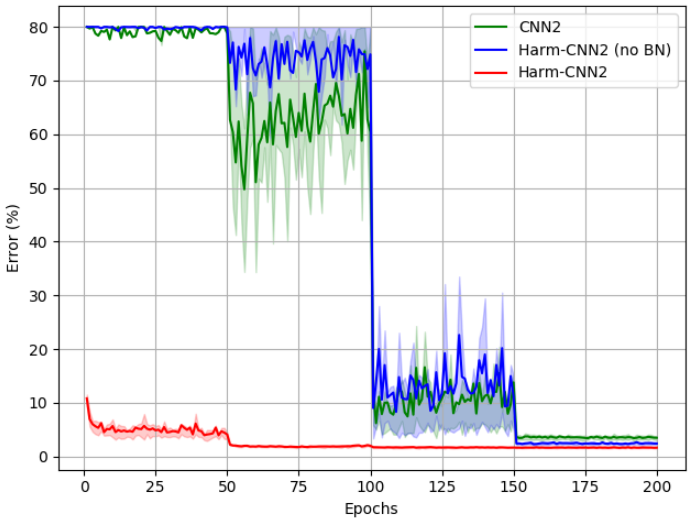


Figure 4.6: Mean classification error on small NORB test set. Weak generalization of CNN (green) and harmonic network (blue) is observed during the early stages of training. Filled areas (best seen in color) show 50% empirical confidence intervals from 20 runs. Batch normalization of DCT spectrum (first block) significantly speeds up convergence of harmonic network (red).

(Harm-Net 4). The first harmonic block uses 4×4 DCT filters applied without an overlap, the further blocks mimic their convolutional counterparts with 3×3 kernels and stride 2. The standard max or average pooling operator is applied between blocks. Using larger input receptive field realized by the DCT transform did not provide any notable advantage.

Performance evaluation. The baseline CNN architecture shows poor generalization performance in early stages of training. The performance on the test set stabilizes only after the third decrease of learning rate, see Figure 4.6. Baseline CNN achieved mean error 3.48%±0.50 from 20 trials, while CNN utilizing harmonic blocks without explicit normalization of harmonic responses exhibits similar behavior resulting in better final solution of 2.40%±0.39. Normalizing DCT responses of input channels at the first block with BN prevents harmonic network from focusing too much on pixel intensity, allows using 10× higher learning rate, significantly speeds up the convergence, improves performance and stability.

We observe the average pooling to work well in combination with harmonic blocks. Error of 1.14%±0.20 is achieved by a hybrid model with one harmonic block followed by 2 standard convolutional layers using overlapping average pooling between them. A variant with 3 harmonic blocks and the same configuration performs comparably with 1.15%±0.22 error (c.f. Table 4.2). The best result was obtained by the model with 3 harmonic blocks replacing the convolutional layers and the fully-connected layer transformed into a harmonic block, misclassifying only 1.10%±0.16 of test samples.

Comparison with state-of-the-art. Table 4.3 shows that these results surpass all previously reported error rates for this dataset to the best of our knowledge. The capsule

Table 4.2: Introduction of harmonic blocks into the baseline architectures and the respective errors on small NORB dataset.

Harmonic layers	Architecture	Pooling type	Error % ↓
	CNN2	overlap max	3.48±0.50
1	Harm-Net2*	max (no BN)	2.40±0.39
1	Harm-Net2*	max	1.63±0.19
1	Harm-Net2*	avg	1.67±0.25
1,2	Harm-Net2	max	1.60±0.18
1,2	Harm-Net2	avg	1.56±0.18
	CNN3	max	3.43±0.31
	CNN3	overlap max	3.89±0.65
1	Harm-Net3*	overlap avg (no BN)	2.56±0.39
1	Harm-Net3*	overlap max	1.16±0.15
1	Harm-Net3*	overlap avg	1.14±0.20
1,2	Harm-Net3*	overlap max	1.21±0.17
1,2	Harm-Net3*	overlap avg	1.15±0.17
1,2,3	Harm-Net3	overlap max	1.18±0.16
1,2,3	Harm-Net3	overlap avg	1.15±0.22
1,2,3,4	Harm-Net4	overlap avg	**1.10±0.16**

*These architectures are constructed by taking the upper part from the Harm-Net and the bottom from CNN baseline with the same number of conv/harm layers.

network [130] claims 1.4% error rate, however estimated under different evaluation protocol, where testing images are resized to 48×48 and predictions for multiple crops of size 32×32 are averaged. The best reported result for a CNN [131] 2.53%±0.40 uses wider CNN architecture and four additional input channels derived from the original input via contrast-extractive filters.

Table 4.3: Comparison with the state-of-the-art on small NORB dataset, showing the proposed method outperforms other reported results.

Method	Parameters	Error % ↓
CNN with input filter [131]	2.7M	2.53 ± 0.40*
Nonlinear SLPP [132]		1.5*
CapsNet [130] multi-crop	310K	1.4*
CapsNet [130] small	68K	2.2*
Harm-Net4	1.28M	**1.10 ± 0.16**
Harm-Net4, fc M=32, no dropout	131K	1.17 ± 0.20
Harm-Net4, fc M=32, no dropout, $\lambda = 3$	88K	1.34 ± 0.21
Harm-Net4, fc M=32, no dropout, $\lambda = 2$	45K	1.64 ± 0.22

*scores reported by authors of the corresponding papers.

Harmonic network compression. We further proceed to designing a very compact version of Harm-Net4 (cf. Table 4.3). To do so, the fully connected layer is reduced to only 32 neurons and the dropout is omitted. The modified network reaches 1.17%±0.20 error

Figure 4.7: Images [1] of an object at two elevations captured with each of the six lighting conditions 0-5 ordered from left to right.

and has 131k parameters. The lack of high-frequency information in the data motivates filter compression by truncation of high-frequencies. When applying compression with $\lambda = 3$ the network reaches 1.34%±0.21 and requires less than 88k parameters[3]. By applying $\lambda = 2$ the total count is less than 45k parameters. This model uses non-overlapping pooling to preserve more of limited high-frequency information. Error of 1.64%±0.22 is achieved on the test set, in contrast with small capsule network [130] with 68k parameters scoring 2.2%.

Harmonic Networks for illumination changes

Spectral representation of input data has a few interesting properties. Average feature intensity is captured into DC coefficient, while other (AC) coefficients capture signal structure. DCT representation has been previously used [24] to build illumination invariant representation. This gives us a strong motivation to test illumination invariance properties of harmonic networks and to compare them with standard CNNs.

Objects in the small NORB dataset are normalized and are presented with their shadows over a uniform background. The six lighting conditions are obtained by various combination of up to 4 fixed light sources at different positions and distances from the objects (see examples on Figure 4.7).

Usual approaches to reduce sensitivity to lighting conditions include image standardization with ZCA whitening [91] or illumination augmentation. Data augmentation is the standard approach in CNNs for compensating the lack of variability in the available training data. Brightness and contrast manipulations encourage the network to focus on features that are independent of the illumination and may as well generate images resembling lighting conditions in the target domain. Contrary to these methods in our approach we achieve the same effect by removing the filter corresponding to the DC component from the first harmonic block. Such network is invariant towards global additive changes in pixel intensity by definition.

Set-up. The dataset is split into 3 parts based on lighting conditions during the image capturing: the bright images (conditions 3,5) dark images (conditions 2,4) and images under standard lighting conditions (0,1) illustrated on Figure 4.7. The models are trained

[3]In this particular setting, the first harmonic block manipulates 4×4 filters and uses $\lambda_{4\times4} = \lambda_{3\times3} + 1$.

(with and without data augmentation) only on data from one split and tested on images from the other two splits that have unseen lighting conditions.

Table 4.4: Mean of classification errors over 10 runs on the test images captured in unseen illumination conditions using small NORB dataset.

Augmentation	**None**		**Brightness & contrast**	
Lighting Condition	**CNN ↓**	**Harmonic ↓**	**CNN ↓**	**Harmonic ↓**
Bright	26.3±2.6	10.2±0.4	17.6±0.7	9.4±0.7
Standard	30.2±1.8	18.0±1.9	22.5±1.1	15.1±1.1
Dark	31.2±1.8	18.9±1.2	20.1±1.3	14.5±1.4

Performance evaluation. Classification errors of the best CNN and harmonic network architectures on the test images (unseen illumination conditions) are reported in Table 4.4. Harmonic networks consistently achieve lower error under various unseen lighting conditions in comparison to baseline CNNs, with and without random brightness and contrast (only for dark images) data augmentation. To take into account the bias caused by better performance of harmonic networks in general conditions, we have also handpicked pairs of harmonic and baseline CNNs that perform comparably on validation set composed from test images with lighting conditions seen during the training. Table 4.5 shows that better performance on unseen conditions is not solely because of a stronger performing model.

Table 4.5: Comparison of generalization to lighting conditions for models with similar validation errors (seen conditions) on small NORB dataset.

Set	**Validation**		**Test**	
Lighting Condition	**CNN ↓**	**Harmonic ↓**	**CNN ↓**	**Harmonic ↓**
Bright	7.80	7.80	17.82	16.40
Standard	7.77	8.23	31.05	26.86
Dark	8.79	9.38	20.21	18.79

4.4.2 Small natural images

The second set of experiments is performed on popular benchmark datasets of small natural images CIFAR-10 and CIFAR-100. Images have three color channels and resolution of 32x32 pixels. The dataset is split into 50k images for training and 10k for testing. Images have balanced labeling, 10 classes in CIFAR-10 and 100 in CIFAR-100.

Baseline. For experiments on CIFAR datasets we adopt WRNs [90] with 28 layers and width multiplier 10 (WRN-28-10) as the main baseline. These improve over the standard ResNets by using much wider residual blocks instead of extended depth. Our implementation extends the PyTorch code[4] from the author. Model design and training procedure are

[4] `https://github.com/szagoruyko/wide-residual-networks/tree/master/pytorch`

kept unchanged as in the original paper [90]. Harmonic WRNs are constructed by replacing convolutional layers by harmonic blocks with the same receptive field, preserving batch normalization and ReLU activations in their original positions after every block.

Results. We first investigate whether the WRN results can be improved if trained on spectral information, i.e. when replacing only the first convolutional layer preceding the residual blocks in WRN by a harmonic block with the same receptive field (Harm1-WRN). The network learns more useful features if the RGB spectrum is explicitly normalized by integrating the BN block as demonstrated in Figure 4.1, surpassing the classification error of the baseline network on both CIFAR-10 and CIFAR-100 datasets, see Table 4.6. We then construct a fully harmonic WRN (denoted as Harm-WRN) by replacing all convolutional layers with harmonic blocks, retaining the residual shortcut projections unchanged. Zagoruyko et al. [90] demonstrated how dropout layers placed inside residual blocks between convolutional layers can provide extra regularization when trained on spatial data [90]. We have observed a similar effect when training on spectral representations, therefore we adopt dropout between harmonic blocks. The harmonic network outperforms the baseline WRN, see Table 4.6. Based on this empirical evidence we always employ BN inside the first harmonic block.

Table 4.6: Settings and median error rates (%) out of 5 runs achieved by WRNs and their harmonic modifications on CIFAR datasets. Number of parameters reported for CIFAR-10.

Method	Dropout	Parameters	CIFAR-10 ↓	CIFAR-100 ↓
WRN-28-10 [90]	✓	36.5M	3.91	18.75
Harm1-WRN-28-10 (no BN)		36.5M	4.10	19.17
Harm1-WRN-28-10		36.5M	3.90	18.80
Harm1-WRN-28-10	✓	36.5M	**3.64**	**18.57**
Harm-WRN-28-10	✓	36.5M	3.86	**18.57**
Harm-WRN-28-10, $\lambda = 3$	✓	24.4M	3.84	18.58
Harm-WRN-28-10, $\lambda = 2$	✓	12.3M	4.25	19.97
Harm-WRN-28-10, progr. λ		15.7M	3.93	19.04
Gabor CNN 3-28 [30]		17.6M	3.88*	20.13*
WRN-28-8 [90]	✓	23.4M	4.01	19.38
WRN-28-6 [90]	✓	13.1M	4.09	20.17

*scores reported by [30].

Analysis of fully harmonic WRN weights learned with 3x3 spectrum revealed that the deeper network layers tend to favour low-frequency information over high frequencies when learning representations. Relative importance of weights corresponding to different frequencies shown in Figure 4.8 motivates truncation of high-frequency coefficients for compression purposes. While preserving the input image spectrum intact, we train the harmonic networks on limited spectrum of hidden features for λ=2 and λ=3 using 3 and 6 DCT filters for each feature respectively. To assess the loss of accuracy associated with parameter reduction we train baselines with reduced widths having comparable numbers of parameters: WRN-28-8

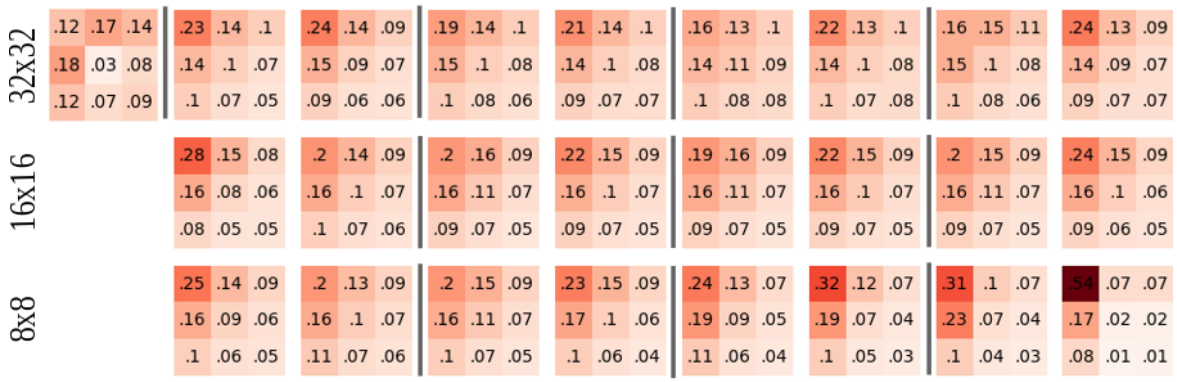


Figure 4.8: Distribution of weights (averaged in each layer) assigned to DCT filters in the first harmonic block (left-most) and the remaining blocks in the Harm-WRN-28-10 model trained on CIFAR-10. Vertical lines separate the residual blocks.

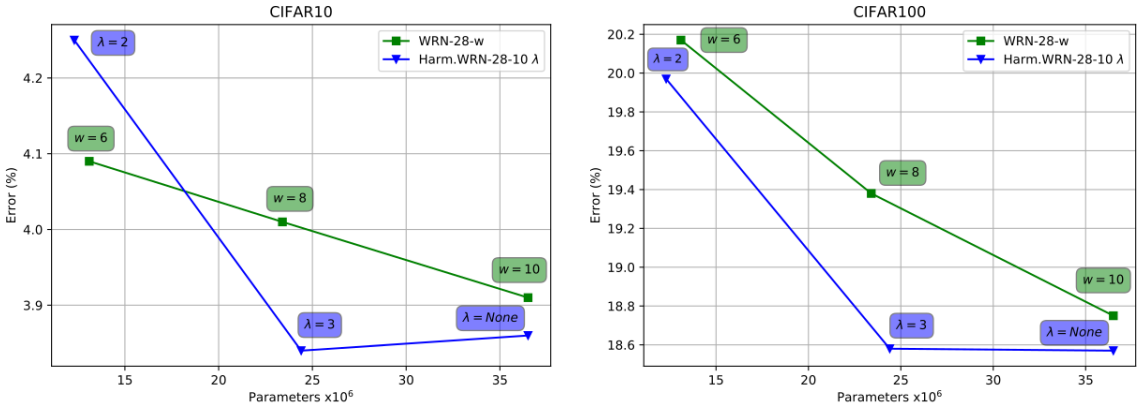


Figure 4.9: Decrease of classification error as a function of model size on CIFAR-10 (left) and CIFAR-100 (right). Parameters of harmonic networks are controlled by the compression parameter λ, and of the WRN baselines by the width multiplier w.

and WRN-28-6, see Figure 4.9. Fully harmonic WRN-28-10 with λ=3 has comparable error to the network using the full spectrum and outperforms the larger baseline WRN-28-10, showing almost no loss in discriminatory information. On the other hand Harm-WRN-28-10 with λ=2 is better on CIFAR-100 and slightly worse on CIFAR-10 compared to the similarly sized WRN-28-6. The performance degradation indicates that some of the truncated coefficients carry important discriminatory information. Detailed comparison is reported in Table 4.6.

We further compare the performance of the harmonic version of WRN-28-10 with the Gabor CNN 3-28 [30] that relies on modulation of the learned filters with oriented Gabor filters. To operate on a similar model we remove dropouts and reduce complexity by applying progressive λ: no compression for 32x32 feature sizes, λ=3 for 16x16, and λ=2 for the rest. With a smaller number of parameters the Harm-WRN-28-10 performs similarly on CIFAR-10 and outperforms Gabor CNN on CIFAR-100.

Harmonic block implementations. Here we compare the standard harmonic block implementation with its memory efficient version introduced in Algorithm 1, see Table 4.12.

The comparison on CIFAR-10 dataset demonstrates that Algorithm 1 provides similar overall performance but reduces both the runtime and memory requirements nearly three times. We will therefore use solely this implementation of the harmonic block except for the root (first) layers, where this implementation is impossible due to the use of BN.

Ablation study. The effect of filter parameterization by DCT basis is investigated by changing particular layers of WRN-16-4 (without dropout), see Table 4.7. The effect is measured as a change in standard error over 5 runs. Firstly, the root convolutional layer of WRN is replaced with harmonic block which adds 0.28 absolute improvement over the baseline error 24.07%. Normalization on the DCT responses gains another 0.12. Using the harmonic blocks only in the residual blocks boosts the accuracy of the baseline by 0.85. Replacing the first layer along with the residual blocks does not provide any empirical benefit. Adding normalization to the root harmonic block gives only a subtle improvement but decreases the variance by half. These observations correspond to the results obtained on the NORB dataset, see Table 4.2. We will always be employing BN as part of the root harmonic block.

Table 4.7: Modifications of the WRN-16-4 baseline on CIFAR-100: mean classification errors and standard deviations from 5 runs.

Root block	Harmonic root BN	Residual blocks	Error % ↓
			24.07 ± 0.24
✓			23.79 ± 0.24
✓	✓		23.67 ± 0.12
		✓	23.22 ± 0.28
✓		✓	23.25 ± 0.25
✓	✓	✓	23.21 ± 0.11

Harmonic network compression. Section 4.3.2 describes how convolutional filters in certain layer can be approximated with fewer parameters. So far we have only considered uniform coefficient truncation by truncating the same frequencies in all the layers or a simple progressive compression. This scheme omits higher number of frequencies in deeper layers, but the same subset of coefficients is used in all harmonic blocks applied to feature maps of particular size. We believe better compression-accuracy trade-off can be achieved by using more elaborate coefficient selection at each layer. In this experiment we start with the WRN-28-10 baseline trained without dropout which has been converted to harmonic WRN-28-10 net (omitting BN in the first harmonic block) by re-expressing each 3×3 filter as a combination of DCT basis functions. The first harmonic block is kept intact (no compression in DCT representation), while all other blocks are compressed. We compare three different coefficient selection strategies:

- *Uniform selection*: at every layer the same λ is used;

- *Progressive selection*: the level of compression is selected based on the depth of the layer $\lambda_{progr} = \max(\alpha, \min(2k-1, \lfloor t/Depth \rfloor))$ for $\alpha = 1$ or 2, constant t, and k is the size of filter ($\lambda = 2k-1$ corresponds to no compression);

- *Adaptive selection*: the compression level is selected adaptively for each layer; a basis filter is excluded if its ℓ_1 norm compared to norms of the other frequencies in the same layer is too low. Specifically, if $\|\mathbf{w}_{i,j}\|_1 / \sum_{u,v=0}^{k-1} \|\mathbf{w}_{u,v}\|_1 < t$ then the coefficient is truncated.

The results reported in Figure 4.10 confirm the behavior observed before (see Figure 4.8), i.e. the high frequencies appear to be more relevant in the early layers of the network compared to deeper layers. The uniform compression fails to adjust and discards the same amount of information in all the layers, and is surpassed by other compression strategies. By using progressive or adaptive coefficient selection a model can be compressed by over 20% without loss in accuracy. The best progressive method loses less than 1% of accuracy when compressed by 45% without a need for finetuning. Note that frequencies are discarded per layer, the selective approach could benefit from more fine-grained compression tailored to individual filters.

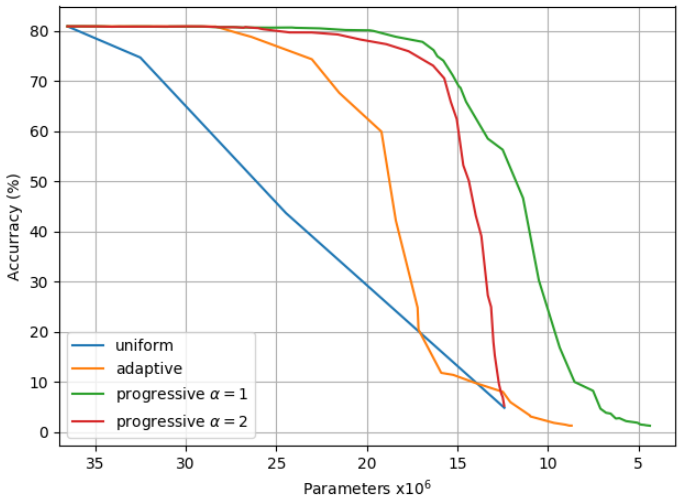
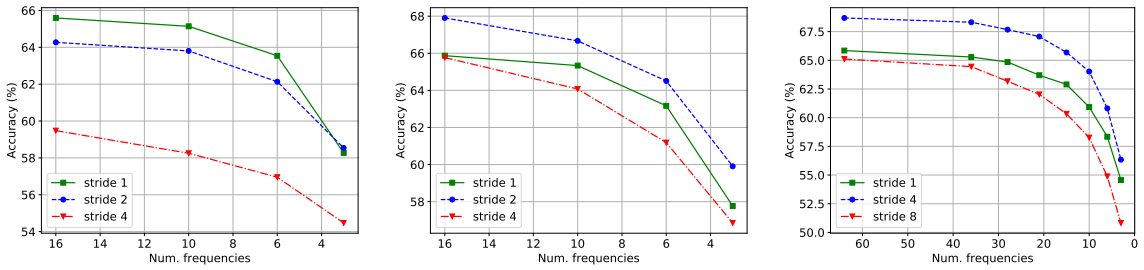


Figure 4.10: Accuracy of compressed harmonic WRN-28-10 on CIFAR100 using different coefficient truncation strategies.

Overlapping DCT In Section 4.3.3 we have demonstrated that the discrete sine transform can be inferred from the DCT on overlapping blocks. Here we show experimentally the benefits of DCT transform with overlapping windows by using overcomplete representation with strides of 1 pixel or fixing the stride to half of the window size. Effect of striding is evaluated on a shallow harmonic network composed of only one normalized harmonic block with 4x4 receptive field, followed by a Rectified Linear Unit (ReLU) activation and connected to a fully connected layer with softmax classifier. This simple architecture allows one to clearly see the contribution of striding. The network is trained with SGD using

learning rate 0.01, Nesterov momentum 0.9, weight decay 0.0005 and batch size 128 for 30 epochs, decaying the learning rate by factor 10 halfway. Since the striding reduces spatial resolution of the features, to match the model complexity, lower dimensional features are resized to have the size of features produced with stride 1. As expected, the network without overlapping windows performs notably worse even when using the full spectrum (see Figure 4.11a).

In order to compare models with similar numbers of parameters, instead of replicating features, networks with larger stride employ a higher number of output features: 200 for non-overlapping, 50 for half-window overlap in contrast to 16 when using stride 1. The same experiment is performed using 8x8 filters learning 625, 200 and 16 feature maps respectively. In this setting the network with stride 1 and the one with full window stride perform comparably on full spectrum as can be seen on Figure 4.11b and Figure 4.11c, but performance degrades more rapidly for non-overlapping filters as the visual spectrum shrinks. The best result was obtained when using a half window stride.



(a) 4x4 replicated features (b) 4x4 balanced block (c) 8x8 balanced block

Figure 4.11: Accuracy degradation of models with different strides when truncating number of DCT coefficients. Stride 1 (green), half window stride (blue) and full window stride (red) are compared. Reported values are averaged over 5 runs.

Lastly, we compare the effect of window function on performance of DCT filters with increased stride. The Gaussian window

$$\mathcal{G}(a) = \frac{1}{\sigma\frac{k}{2}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{a-\frac{k-1}{2}}{\sigma\frac{k}{2}}\right)^2} \tag{4.9}$$

is used to scale the $k \times k$ filter bases in each direction, where a is a spatial index within the filter. Hyperparameter σ controls the spread and is independent of the filter size. Examples of smoothed DCT bases can be seen on Figure 4.12. Figure 4.13 demonstrates that using compact windows with low σ has negative impact specially when no overlap occurs.

Robustness to noise. Corruption in an input image may lead to faulty prediction. Harmonic networks with frequency truncation on early layers are more robust to certain types of noise. The noise is usually concentrated in high-frequency components. By restricting the filters to only low frequencies, harmonic block becomes less responsive to spikes in the image intensities. We have investigated how compressed harmonic WRN-28-10 networks

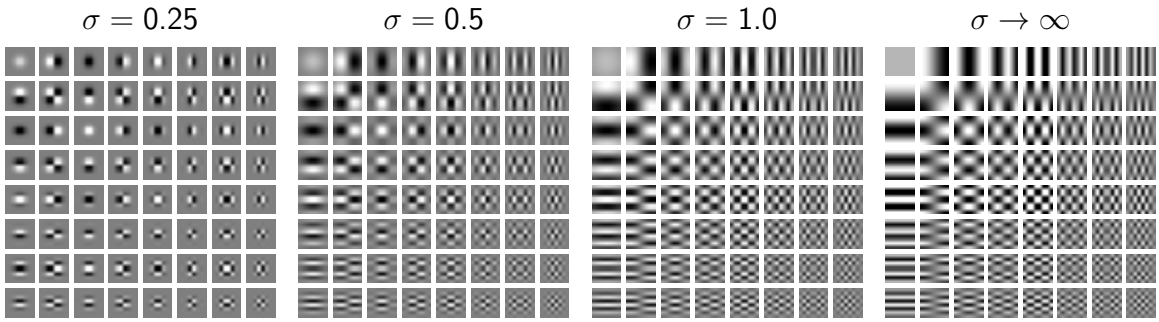


Figure 4.12: The effect of Gaussian window with various σ (4.9) on DCT bases, visualized for 8×8 filters.

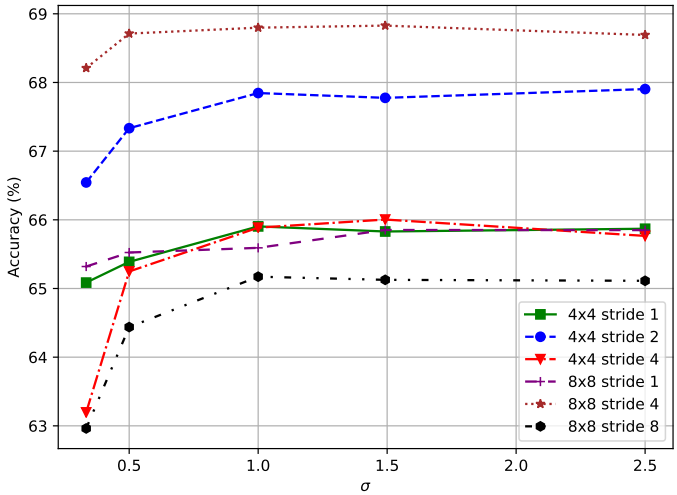
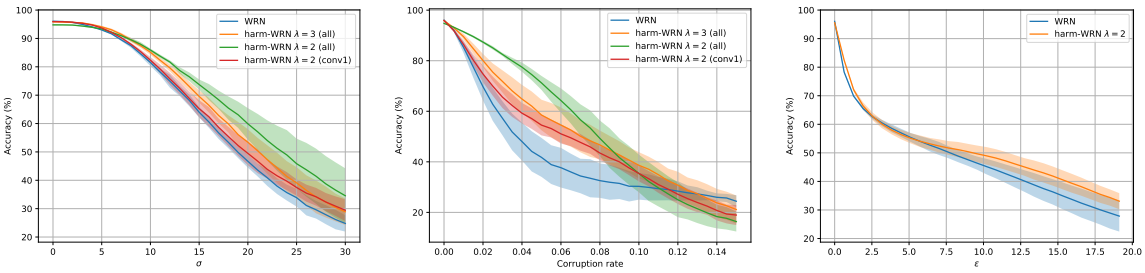


Figure 4.13: Effect of Gaussian window function on filter bases when filters are applied with increased stride. Values shown are the average of 5 runs.

cope with additive noise while classifying CIFAR-10 images. Figure 4.14 shows a comparison with the baseline network on various levels of (a) additive white Gaussian and (b) salt & pepper noise. Compressing only the first layer is in principle similar to low-pass filter preprocessing: it provides a slight improvement, but a much more robust model, especially against the salt&pepper noise, is achieved when all the layers are compressed. This analysis shows that harmonic networks with compression perform non-trivial denoising of high-frequency noise.

We also investigate robustness of harmonic nets to the fast gradient sign adversarial noise [133]. An image $\mathbf{x}$ is corrupted with noise $\epsilon \operatorname{sign}\left(\nabla_x \mathcal{L}\left(\mathbf{w}, \mathbf{x}, \mathbf{y}\right)\right)$, having model parameters $\mathbf{w}$, label $\mathbf{y}$ and a loss function $\mathcal{L}$. Magnitude of the noise is controlled by ϵ. This noise in turn tends to be correlated in local regions, compressing only the first layer provides no gain. A mild robustness to this noise is however obtained by compressing all the other layers, see Figure 4.14. Similar discovery was made by Wang et al. [134] who achieved increased robustness by smoothing the filters.

Convergence speed. High-frequency components have generally smaller responses in



Gaussian noise | Salt & pepper noise | Adversarial noise

Figure 4.14: Robustness of WRN-28-10 to noise: compressed harmonic networks show better performance than the baseline. Confidence intervals of 2 standard deviations from tests of 5 model instantiations shown in color.

both forward and backward passes. Lets revise standardly used Stochastic gradient descent with momentum μ:

$$\begin{aligned} \Delta w &= \mu \Delta w + \nabla_w \mathcal{L}(w, \mathbf{x}, \mathbf{y}) \\ w &= w - \eta \Delta w. \end{aligned} \tag{4.10}$$

The learning rate η is constant for all weights and magnitude of the weight update $|\Delta w|$ depends only on the magnitude of the exponential average of the past gradients $\nabla_w \mathcal{L}(w, \mathbf{x}, \mathbf{y})$. We have taken harmonic WRN-16-8 model to study gradient evolution throughout the training process. Focusing on the last convolutional layer of this model, we have estimated the sum of gradient magnitudes grouped by the DCT bases ($\{\psi_{u,v}; 0 \leq u, v \leq 2\}$) which they correspond to, and normalized them such that sums of all bases add up to 1. Naturally, the highest gradient magnitude belongs to the zero frequency $\psi_{0,0}$. If the parameter w is representing a high-frequency component ($\lambda > 3$), its updates diminish in time compared to the parameters of lower frequencies as shown on Figure 4.15 left.

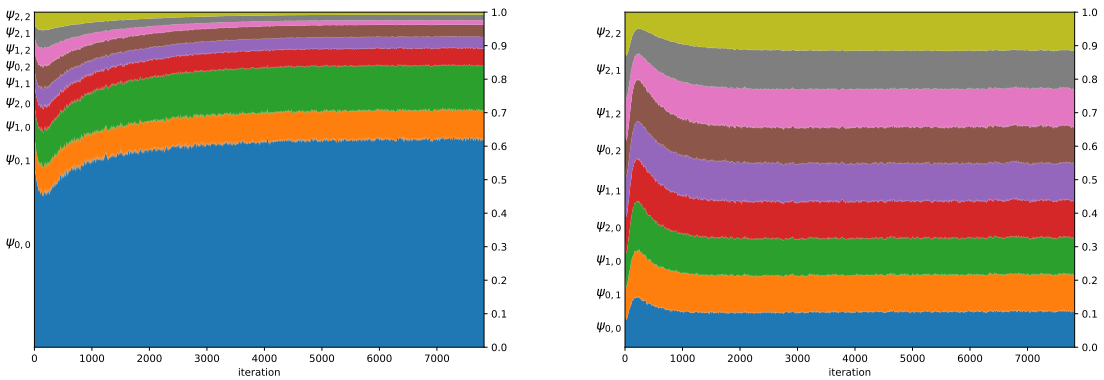


Figure 4.15: Proportion of weight updates between bases in harmonic block using SGD (**left**) and Adam (**right**).

A different behaviour was observed when using the Adaptive momentum optimizer (Adam):

$$\begin{aligned} m &= \beta_1 m + (1-\beta_1)\,\nabla_w \mathcal{L}(w, \mathbf{x}, \mathbf{y}) \\ v &= \beta_2 v + (1-\beta_2)\,\nabla_w \mathcal{L}(w, \mathbf{x}, \mathbf{y})^2 \\ w &= w - \eta \frac{m}{\sqrt{v} + \epsilon} \end{aligned} \tag{4.11}$$

The weight update for a parameter w is given by a ratio of the first m and the second v moment. While the training is unstable, this ratio is small for high frequency parameters, see iteration 200 in Figure 4.15 right. As the training starts to converge, this ratio increases. The updates become close to uniformly spread between parameters of different frequencies. This distinction does not occur for filters optimized within standard bases. As a result, during the initial stage of the training, the harmonic network converges faster, and smaller variance is observed between different models. A comparison of convergence has been conducted with 6 different learning rates (Figure 4.16).

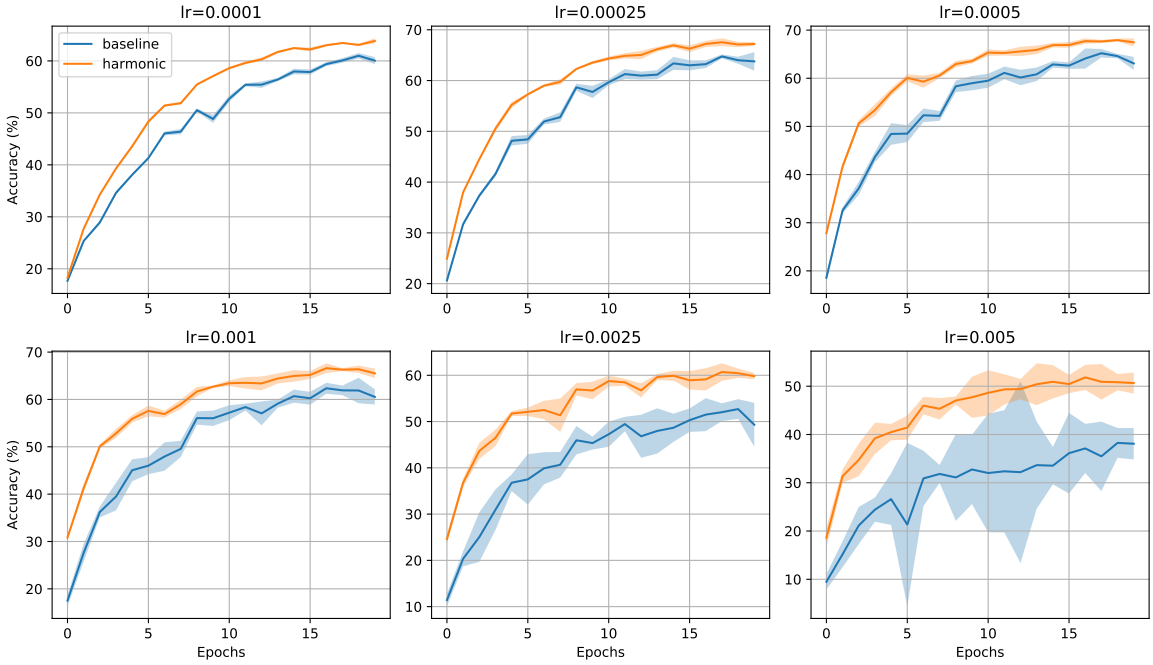


Figure 4.16: Convergence speed of harmonic and baseline WRN-16-8 model with Adam optimizer during the first 20 epochs. The confidence intervals show 2 standard deviations from 5 trials.

4.4.3 Limited data

Deep neural networks require abundant data to achieve high accuracy. It has been shown [27, 112] that scattering network can, by using geometric priors, learn better discrimination boundaries when presented with a small subset of training samples. We demonstrate capabilities of harmonic networks when learning from limited subsets of data on three classification datasets MNIST, CIFAR-10 and STL-10.

MNIST

Bruna and Mallat [27] have chosen the dataset of handwritten digits to test their fully handcrafted scattering network with respect to classification performance on data subsets. We compare our harmonic network to the "classical" CNN, learned depth-separable convolution network and to the fully handcrafted scattering network (as reported by [27]). Table 4.8 shows the harmonic network achieves lower classification error for all sizes of the training set. The baseline network is composed of 3 convolution layers with 32, 64 and 128 3×3 filters, respectively, and with overlapping average pooling between them. Convolutional layers are followed by a fully connected layer with 512 neurons. Batch normalization and ReLU are applied after each layer. The harmonic network uses the same configuration, replacing convolution with harmonic block, while using additional BN in the first block. Harmonic networks are also compared to the depth-separable convolution network that has the same structure but has randomly initialized learnable filters instead of DCT filters. Training is done with SGD for 30 epochs with learning rate 0.1 reduced after every 10 epochs by a factor 10. Weight decay ranges from 0.0005 (for training size 60000) to 0.05 (training size 300). Harmonic networks outperform other networks in all configurations, see Table 4.8.

Table 4.8: Classification errors in % (median of 21 runs) on subsets of MNIST dataset for the harmonic network and benchmarks.

Training size	Scatter [27] ↓	Conv ↓	Sep. conv ↓	Harm ↓
300	4.7	3.9	4.67	**3.71**
1000	2.3	1.88	1.91	**1.84**
2000	1.3	1.39	1.35	**1.21**
5000	1.03	0.97	1.06	**0.86**
10000	0.88	0.7	0.76	**0.65**
20000	0.58	0.59	**0.57**	**0.57**
40000	0.53	0.48	0.47	**0.45**
60000	0.43	0.44	0.46	**0.38**

CIFAR-10

We replicate the experiment from Oyallon at al. [112] and train harmonic network on random subsets of CIFAR-10 dataset with sizes 100, 500 and 1000 samples, preserving the equal number of labels per class. Harmonic WRN 16-8 with dropout rate 0.2 is trained as in [112]. Harmonic layers relying on combinations of fixed filters provide advantage on limited data compared to fully learned CNNs and to scattering CNN hybrids[5] except for the smallest training dataset, see Table 4.9.

[5]The exact subsets used to train the scattering CNN hybrids are not known, we report numerical results from [112].

Table 4.9: Average classification accuracy ± standard deviation of 5 runs on subsets of CIFAR-10.

Method	100 ↑	500 ↑	1000 ↑	Full ↑
WRN 16-8	34.4±1.8	52.2±1.8	62.8±0.7	**95.6**
Scat + WRN [112]	**38.9±1.2**	54.7±0.6	62.0±1.1	93.1
Harm WRN 16-8	37.7±1.9	58.2±1.4	67.0±0.4	**95.6**
Harm WRN 16-8 $\lambda = 3$	37.9±2.4	**58.4±0.9**	**67.2±0.5**	**95.6**
Harm WRN 16-8 $\lambda = 2$	37.2±1.7	57.0±1.0	65.9±0.8	95.3

STL-10

STL-10 [135] is a natural image dataset similar to CIFAR-10. Images are 96×96 and only 5000 training images are labeled. The large set of provided unlabeled images is not utilized in this experiment. We design harmonic WRN 16-8 model (based on Algorithm 1) for this task with several necessary modifications. The first layer uses stride 2, and the feature resolution at the final stage is 12×12. We apply dropout with probability 0.3 inside residual blocks and train the network on the whole training set with learning rate of 0.1 decayed by factor 0.2 after 300, 400, 600, 800 epochs, and stop the training after epoch 1000. The baseline network design and training procedure is similar to [136] that uses additional cutout regularization and reports 87.26% ± 0.23 accuracy on the test set containing 8000 images when trained on batches of 128 images. The harmonic WRN 16-8 achieves 88.1% ± 0.23 trained with the same settings. Decreasing the batch size to 32 improves our result to 90.45% surpassing the deeper scattering WRN [112] by nearly 3%. Furthermore, when only predefined folds of 1000 samples serve as the training data, we obtain the best accuracy by progressively reducing the number of used frequencies along with the spatial resolution: full filter bank is applied on features of size 48×48, filters with $\lambda = 3$ on 24×24 and finally $\lambda = 2$ if features are 12×12. The results of STL-10 experiments are summarised in Table 4.10.

Table 4.10: Average classification accuracy ± standard deviation of 5 runs on STL-10 for various sizes of training data (batch size 32).

Method	1000 imgs. ↑	5000 imgs. ↑
WRN 16-8	73.50 ± 0.87	87.29 ± 0.21
Scat + WRN [112]	76.00 ± 0.60	87.60
Harm WRN 16-8	76.95 ± 0.93	**90.45 ± 0.12**
Harm WRN 16-8 $\lambda = 3$	76.65 ± 0.90	90.39 ± 0.08
Harm WRN 16-8 progressive λ	**77.19 ± 1.02**	90.28 ± 0.20

4.4.4 Large data

In this section we present results obtained on ImageNet-1K classification task. To deploy harmonic networks on large-scale datasets a few adjustments are applied to the harmonic

blocks: Firstly, in the absence of BN inside harmonic block we merge the linear operations together (feature extraction with DCT bases and their weighted combination, by use of Algorithm 1). Secondly, we normalize DCT filters by their ℓ_1 norm.

Table 4.11: Classification errors on ImageNet validation set using central crops after 100 epochs of training.

Model	Parameters	Top-1 % ↓	Top-5 % ↓
VGG-16-BN	138.4M	26.33	8.26
Harm-VGG-16-BN	138.4M	25.55	8.01
ResNet-50 (no maxpool)	25.6M	23.81	6.98
Harm1-ResNet-50	25.6M	**22.97**	**6.48**
Harm-ResNet-50	25.6M	23.11	6.63
Harm-ResNet-50 (avgpool)	25.6M	23.1	6.53
Harm-ResNet-50, progr. λ	19.7M	23.12	6.61
Harm-ResNet-101	44.5M	21.48	5.75
Benchmarks			
ResNet-50 (maxpool) [137]	25.6M	23.85	7.13
ScatResNet-50 [114]	27.8M	25.5	8.0
JPEG-ResNet-50 [25]	28.4M	23.94	6.98
ResNet-101 (maxpool) [137]	44.5M	22.63	6.44

ResNet [63] with 50 layers is adopted as the baseline. Following Gross and Wilber [138] we apply the stride on 3x3 convolution instead of the first 1x1 convolution in the residual block when downsampling is realized. As an alternative to maxpooling, we used stride 4 in the first convolution layer to reduce memory consumption; we refer to this modification as ResNet-50 (no maxpool). The following harmonic modifications refer to this baseline without the maxpooling. Similarly to the above reported CIFAR experiments we investigate the performance of three harmonic modifications of the baseline: (i) replacing solely the initial 7x7 convolution layer with harmonic block (with BN) with 7x7 DCT filters, (ii) replacing all convolution layers with receptive field larger than 1x1 with equally-sized harmonic blocks, (iii) compressed version of the fully-harmonic network. Each model is trained with stochastic gradient descent with learning rate 0.1, reduced 10 times every 30 epochs, reporting the final accuracy at epoch 100. We employ batch size of 256, weight decay 0.0001 and random scale, aspect ratio & horizontal flip augmentation as recommended in [60], producing 224×224 crops.

Table 4.11 reports error rates on ImageNet validation set using central 224×224 crops from images resized such that the shorter side is 256. All three harmonic networks have similar performance and improve over the baseline by $0.6 - 1\%$ in top1 and $0.4 - 0.6\%$ in top5 accuracy. We observe similar progress of the three modifications during the training, see Figure 4.17. ResNet-50 architecture has only 17 layers with spatial filters (other layers have 1×1 filters), which correspond to 11M parameters. We reduce this number by using progressive λ compression: λ=3 on 14x14 features and λ=2 on the smallest feature maps.

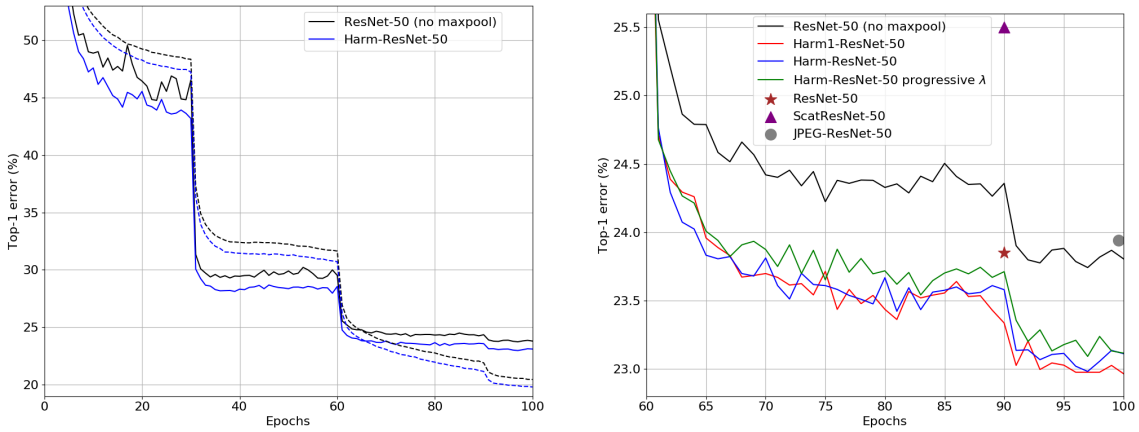


Figure 4.17: Training of harmonic networks on ImageNet classification task. Left: comparison with the baseline showing validation error (solid line) and training error (dashed). Right: tlast 40 epochs of training for all the ResNet-50 based models, including scores reported for the benchmark models.

This reduces the number of weights in compressible layers roughly by half, in total by about 23% of the whole network size. The compressed network loses almost no accuracy and still clearly outperforms the baseline. It should be noted that harmonic networks without bottleneck residual blocks can be compressed more efficiently. Even with compression, the proposed Harm-ResNet-50 confidently outperforms the standard ResNet-50 (maxpool), as well as the more recent ScatResNet-50 [114] and JPEG-ResNet-50 [25], see Table 4.11. Furthermore, we also observe a substantial improvement of 1.15 in top-1 error % associated with the introduction of harmonic blocks into a deeper ResNet-101 architecture. Note that similarly to ResNet-50, the harmonic version is not using pooling and relies on striding instead to reduce the computational complexity. Finally, Harm-ResNet-50 (avgpool) demonstrates a largely similar quantitative performance to the Harm-ResNet-50, which employs striding after the first layer, at the cost of about 10% computational overhead (see Table 4.12). This experiment further validates the selection of nonpooling-based harmonic architecture as the main model.

We validate the use of harmonic blocks on an architecture without residual connections as well. To this end we employ a VGG-16 [139] architecture with BN layers. Harm-VGG-16-BN obtained by replacing all convolutional layers by harmonic blocks yields an improvement of approximately 0.8% in top-1 classification error. This demonstrates that the improvement by harmonic networks is not limited to residual connection-based architectures.

Finally, we evaluate conversion of weights of a pretrained non-harmonic network to those of its harmonic version. To this end each learned filter in the pretrained baseline (ResNet-50 without maxpooling after 90 epochs of training) is expressed as a combination of DCT filters. We skip BN inside the first harmonic block since the related parameters are not available. The direct conversion resulted in the exact same numerical performance due to the basis properties of DCT. We observe a similar pattern of relative importance of DCT filters to the one reported in Figure 4.8. We then finetune the converted model for another

Table 4.12: GPU training memory requirements and speed of harmonic block implementations on CIFAR-10 and ImageNet. All ImageNet models use harmonic blocks based on Alg. 1. Values are measured on Nvidia RTX6000 using batch size 128.

Model	GPU memory ↓	Images/s ↑ train.	infer.	Error % ↓
CIFAR10				
WRN-28-10 [90]	4.6GB	606.4	1876.9	3.89
Harm-WRN-28-10 (2-stage)	14.1GB	211.0	600.4	3.71
Harm-WRN-28-10 (Alg. 1)	4.8GB	573.3	1736.5	3.78
ImageNet				
ResNet-50 (no maxpool)	11.2GB	306.2	820.5	23.81
ResNet-50 (maxpool)	12.1GB	292.9	790.1	23.85
HarmResNet-50	11.4GB	296.3	766.5	23.11
HarmResNet-50 (avgpool)	15.9GB	268.9	680.9	23.10
ResNet-101 (maxpool)	17.4GB	174.1	526.7	22.63
HarmResNet-101	16.9GB	174.4	507.9	21.48

Table 4.13: Performance of the converted harmonic networks (error on ImageNet).

Training	Epochs	Model	Top-1 % ↓	Top-5 % ↓
full	90	ResNet-50 (no maxpool)	24.36	7.34
full	90	Harm-ResNet-50	23.58	6.91
finetuned	90+5	ResNet-50 (no maxpool)	24.34	7.30
finetuned	90+5	ResNet-50 $\Rightarrow$ Harm-ResNet-50	24.15	7.15
finetuned	90+5	ResNet-50 $\Rightarrow$ Harm-ResNet-50, progr. λ	24.60	7.43

5 epochs with the learning rate of 0.001, which results in the top1 (top5) performance improvement of 0.21% (0.19%) over the pretrained baseline, see Table 4.13. We also investigate the conversion to a harmonic network with progressive λ compression. After casting the pretrained filters into the available number of DCT filters (from full basis at the early layers to 3 out of 9 filters at the latest layers), the top1 performance degrades by 6.3% due to loss of information. However, if we allow finetuning for as few as 5 epochs the top1 (top5) accuracy falls 0.24% (0.09%) short of the baseline, while reducing the number of parameters by 23%. This analysis shows how the harmonic networks can be used to improve the accuracy and / or compress existing pretrained CNN models.

Comparison with the state-of-the-art techniques. Here we verify the use of DCT-based harmonic blocks in the more elaborate state-of-the-art models. To this end we modify ResNeXt architecture [31], which is similar to ResNets and uses wider bottleneck and grouped convolution to decrease the amount of floating point operations (FLOPs) and the number of parameters. The model is further boosted using several state-of-the-art adjustments: *(i)* identity mapping in blocks that downsample features are extended by average pooling to prevent information loss; *(ii)* squeeze and excitation blocks (SE) [140] are used after every residual connection. The network is further regularized by stochastic

depth [141] and dropout on the last layer. Training is performed via stochatic gradient decent with learning rate 0.1 and batch size 256, with the former decayed according to one cosine annealing cycle [142]. In addition to mirroring and random crops of size 224, images are augmented with rotations and random erasing [143].

Table 4.14: SE-ResNeXt networks: harmonic vs. baseline errors and comparison with the state of the art on ImageNet.

Model	Param	224×224			320×320 / 331×331		
		FLOPS	Top-1↓	Top-5↓	FLOPS	Top-1↓	Top-5↓
ResNeXt-101 (RNX):							
RNX (64x4d) [31]	83.6M	15.5B	20.4	5.3	31.5B	19.1	4.4
SE-RNX(32x4d)	49.0M	8.0B	19.74	4.90	16.3B	18.80	4.19
same, reported in [140]	49.0M	8.0B	19.81	4.96	-	-	-
Harm-SE-RNX(32x4d)	49.0M	8.1B	19.55	4.79	16.5B	18.72	4.23
Harm-SE-RNX(64x4d)	88.2M	15.4B	**18.36**	**4.37**	31.4B	**17.34**	**3.71**
Benchmarks							
PolyNet [144]	92M	-	-	-	34.7B	18.71	4.25
DualPathNet-131 [145]	79.5M	16.0B	19.93	5.12	32.0B	18.55	4.16
EfficientNet-B4 [146]	19.3M	-	-	-	4.2B	17.4	3.7
SENet-154 [140]	115.1M	20.7B	**18.68**	**4.47**	42.3B	17.28	3.79
NASNet-A [147]	88.9M	-	-	-	23.8B	17.3	3.8
AmoebaNet-A [148]	86.7M	-	-	-	23.1B	17.2	3.9
PNASNet [149]	86.1M	-	-	-	25.0B	17.1	3.8
EfficientNet-B7* [146]	66M	-	-	-	37B	**15.6**	**2.9**

*model trained on 600×600 crops.

Our ResNeXt modification with 101 layers and 32 groups per 4 convolutional filters in residual blocks (expressing filters in both standard bases and DCT bases) is trained for 120 epochs. Use of DCT bases provides a subtle improvement of 0.2% over the standard bases. Furthermore, we upscale the network to use 64 groups of filters, replace max-pooling in the first layer by increased stride and train this network for 170 epochs. From Table 4.14 we conclude that our model outperforms all other "handcrafted" architectures that do not use extra training images and performs comparably to the networks of similar complexity found via neural architecture search. It should be noted that these models were also trained on larger image crops than our harmonic network, whereby such training typically improves the accuracy.

4.4.5 Comparison with other bases

So far we have used the harmonic block built upon DCT bases. As it was mentioned previously, it can conceptually be used with an arbitrary basis function set. We compare harmonic block implementations on other basis sets where the coefficients in the blocks are either optimized from random initialization or are expressing already existing filters.

Training from Random Initialization

In this section we have conducted a comparison of DCT with other basis functions listed in literature for filter decomposition. The baseline models are the large WRN-28-10 and much smaller WRN-16-4 trained to classify CIFAR-100 images. The comparison is conducted with the Fourier-Bessel (FB) bases [74] and the Gaussian derivative (GD) bases [26], for details see Section 2.5. All the filters are 3×3, and we compare in scenarios with 9, 6 or 3 basis functions, always maintaining 9 in the first block, which also uses BN throughout the experiment. Some of the FB and GD filters discretized on 3×3 grid are correlated with filters of different orders. To compose the set of 9 filters we use a subset such that those higher-order filters that have the smallest correlation with with the lower-order filters are included.

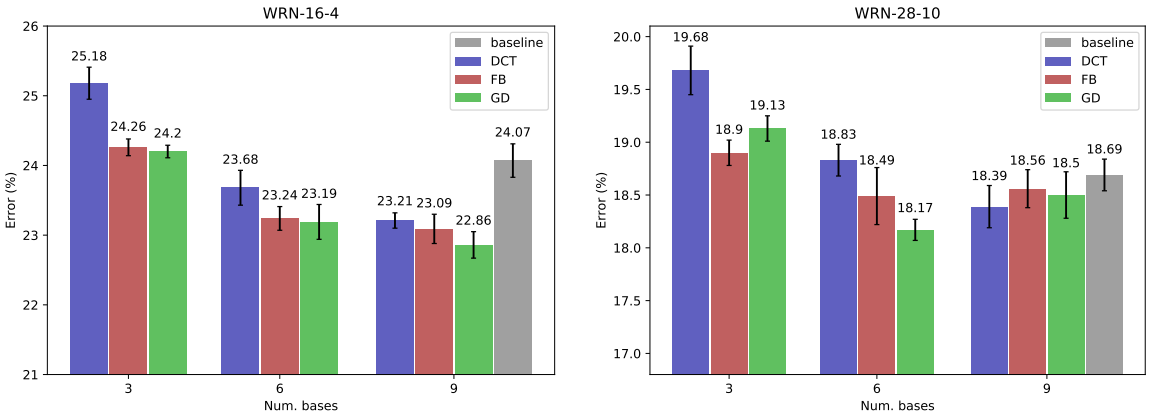


Figure 4.18: Comparison of DCT bases with Fourier-Bessel (FB) and Gaussian derivative (GD) bases on CIFAR-100 classification task.

We train all models from random initialization within the bases listed above. DCT was found to be equivalent to the other bases on both models with full spectrum, however both GD and FB bases perform better when truncated. The 3 leading FB bases have significant correlation with higher frequency DCT bases, using which generally improves the results (see gram matrix between normalized FB and DCT on Figure 4.19).

It is worth to investigate whether Gaussian window function $\mathcal{G}$:

$$\mathcal{G}(a) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{a-\frac{k-1}{2}}{\sigma}\right)^2} \tag{4.12}$$

can induce correlation with higher frequencies in the 3 lowest frequency bases and thus improve the classification. DCT bases $\{\psi_{u,v}; u+v \leq 1\}$ are modulated by $\mathcal{G}$ to create windowed bases $\phi_{u,v}$ as follows:

$$\phi_{u,v}(a,b) = \psi_{u,v}(a,b)\,\mathcal{G}(a)\,\mathcal{G}(b). \tag{4.13}$$

We compare various spread levels σ and find that value in range $\langle 0.5, 0.8 \rangle$ even surpasses

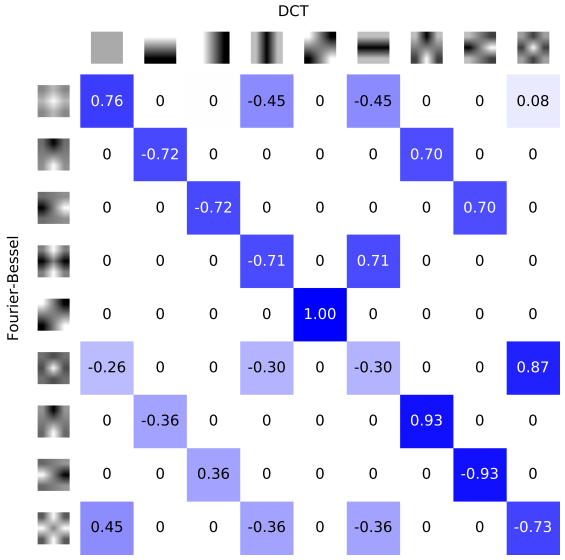


Figure 4.19: Gram matrix between ℓ_2 normalized Fourier-Bessel and DCT bases measured on 3×3 grid.

the performance of FB bases (Figure 4.20).

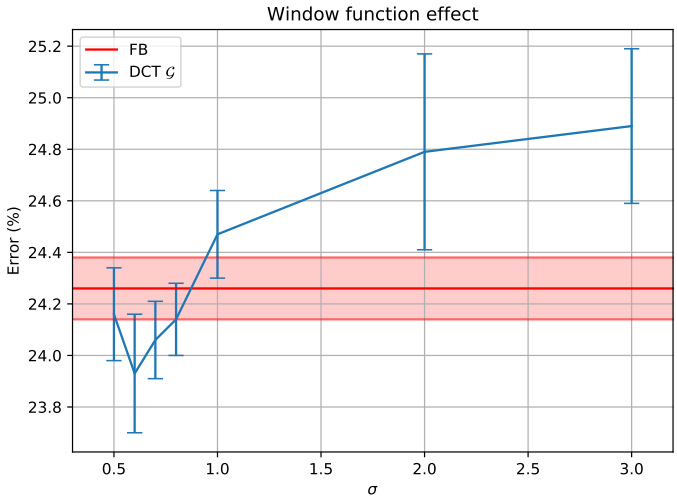


Figure 4.20: Comparison of Fourier-Bessel (FB) bases and DCT bases transformed with window $\mathcal{G}$ with different σ on CIFAR-100 classification task. Only 3 basis functions are used and the confidence intervals show standard deviations from 5 trials.

Expressing Existing Filters

Here we extend the comparison to evaluate ability of certain basis sets to express filters of trained popular CNN architectures. The goal of this experiment is to assess the capability of certain truncated bases to retain necessary information. We compare DCT bases with the Gaussian Derivative bases (GD), Fourier-Bessel bases (FB) and PCA bases, which are estimated on a particular CNN layer. GD and FB bases are however not complete orthonormal sets and exact reconstruction would not be possible. Kobayashi [73] has constructed an orthogonal but incomplete set of Gaussian Derivative filters through Gram-Schmidt pro-

cess. We construct orthonormal versions of the two sets with QR decomposition. However, it should be noted that after the orthogonalization process even low-order bases can be different from the original bases. See the differences for a set of 7×7 GD and FB bases on Figure 4.21. An example of PCA basis set estimated from the first layer of ResNet-50 with 7×7 filters and the DCT bases are depicted on Figure 4.22.

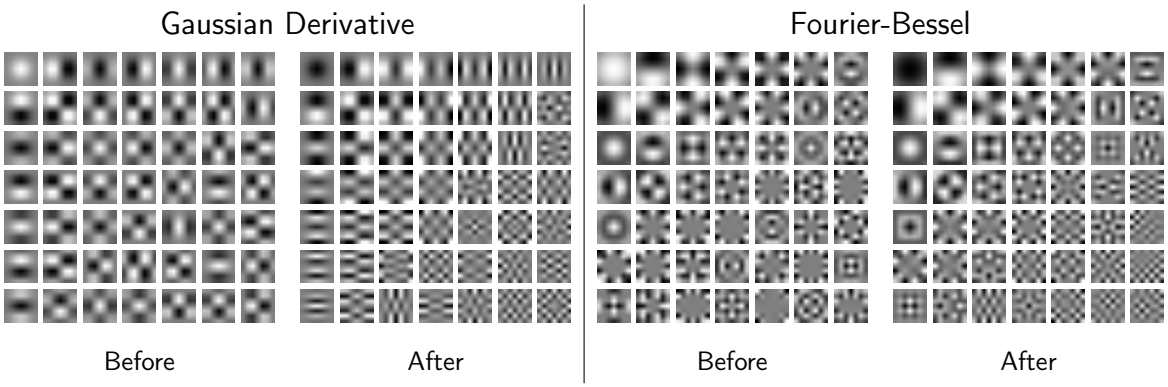


Figure 4.21: Result of othogonalization process on Gaussian Derivative and Fourier-Bessel basis sets.

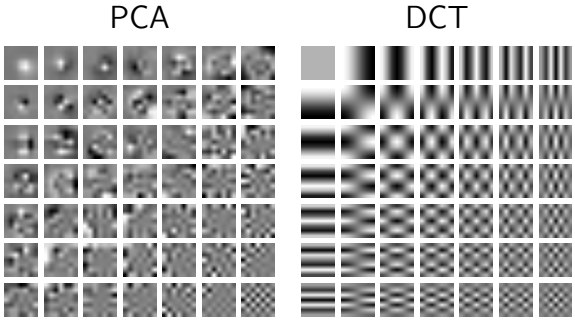


Figure 4.22: PCA bases estimated from the input layer of trained ResNet-50, next to DCT basis set.

We reconstruct filters $\{\mathbf{w}_i; 0 \leq i < n\}$ using a basis set $\{\psi_u; 0 \leq u < k\}$ and following Kobayashi [73], we measure the reconstruction error as the left-out energy ϵ:

$$\epsilon = 1 - \frac{\sum_i^n \left\| \sum_u^k \text{vec}\left(\psi_u\right) \text{vec}\left(\psi_u\right)^T \text{vec}\left(\mathbf{w}_i\right) \right\|_2^2}{\sum_i^n \left\| \text{vec}\left(\mathbf{w}_i\right) \right\|_2^2} \tag{4.14}$$

where $\text{vec}\left(.\right)$ is a vectorization operator. We do the evaluation on filters of different resolutions taken from five layers of various CNN models. The 11×11 and 5×5 filters are taken from the first and the second layer of AlexNet [12] respectively. We also compare on 3×3 filters from the first layer of VGG-16 [59] and from last block of ResNet-50 [50]. The 7×7 filters are also taken from the first layer of ResNet-50.

The reconstruction is evaluated at various truncation levels, see Figure 4.23. The PCA bases achieve far the lowest reconstruction error due to their ability to adapt to the data. The FB and GD bases show similar results, leaving the DCT bases behind when reconstructing

all of the early layers (11×11, 7×7, 5×5 and 3×3 on VGG-16). DCT however shows reconstruction tailing the PCA bases by only a small margin when used on the last layer of ResNet. This layer has far less high-frequency information compared to the early layers, which might explain the different behaviour. The next stage is a comparison of classification errors on ImageNet validation set when replacing trained filters by the filters approximated with a subset of basis functions. The results (Figure 4.23) show almost the opposite trend, except for the 5×5 filters, the truncated DCT bases lead to the lowest errors. This indicates that filter reconstruction error is not directly linked with classification performance. Gaussian derivative filters of low orders have small support on filter edges, which may deform the filter pattern and have uneven spread of the reconstruction error.

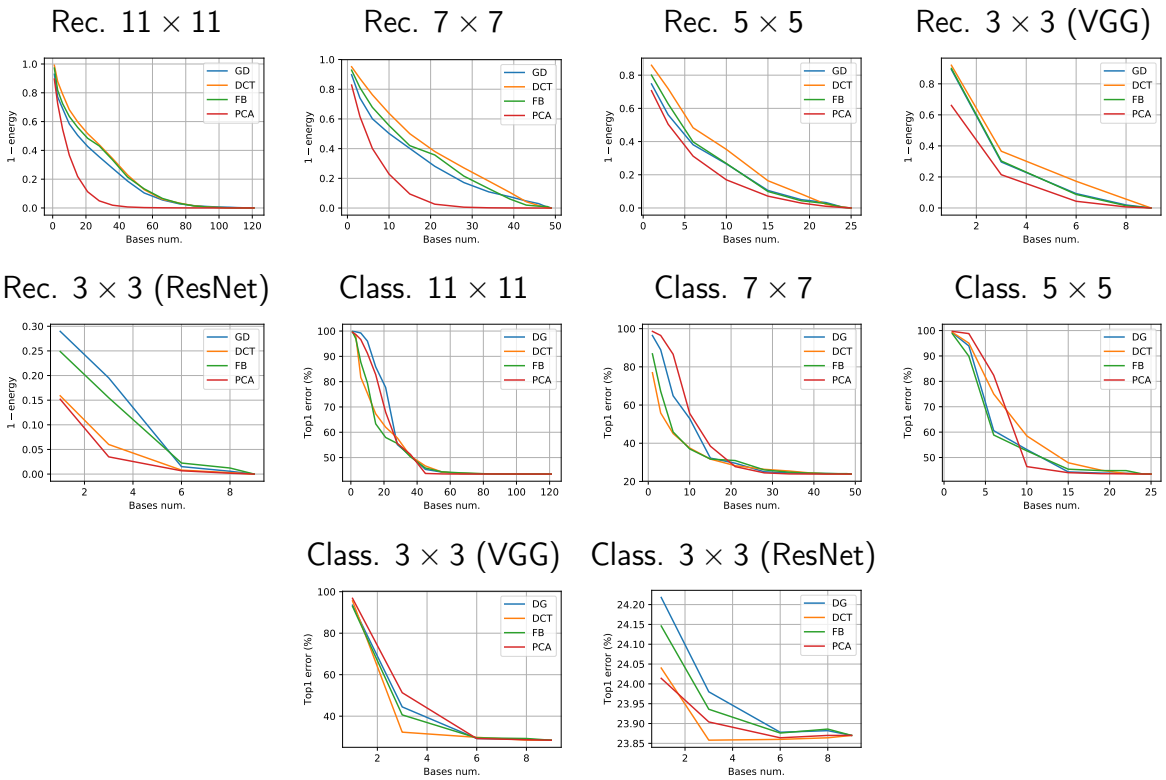


Figure 4.23: Reconstruction and classification (ImageNet val. set) results of filter compression with DCT, Gaussian Derivative (GD), Fourier-Bessel (FB) and PCA bases. Filters are taken from trained AlexNet (11×11, 5×5), ResNet-50 (7×7, 3×3) and VGG-16 (3×3).

Despite the fact that DCT is efficient at compressing some particular layers, if we extend this experiment to all 3×3 filters in VGG-16 or ResNet-50, the DCT compression resulted in inferior results to the other bases, see Figure 4.24. We also show that progressive selection of bases where deeper layers have fewer bases (dashed lines on Figure 4.24) provides notably better accuracy/size trade-off.

4.5 Detection and Segmentation

Classification represents only a subset of problems in computer vision. CNNs are often used for tasks that require more fine-grained predictions or tasks that do not need semantic

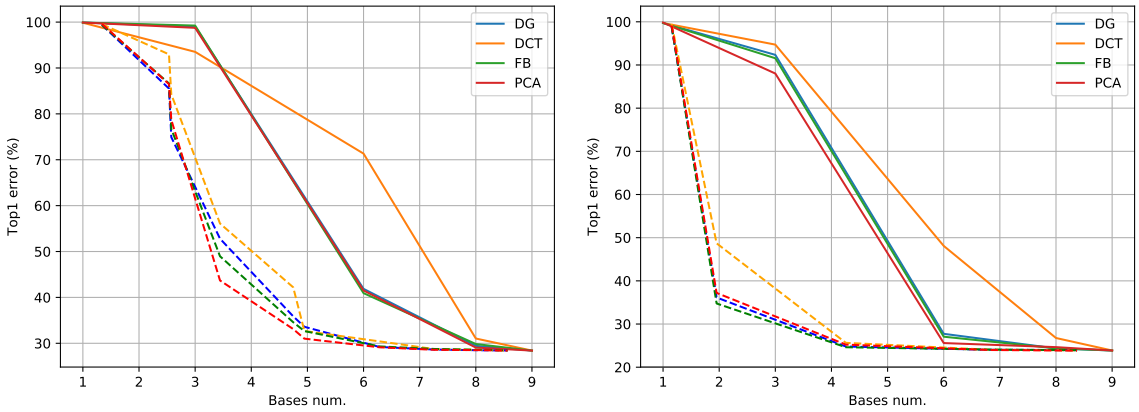


Figure 4.24: Classification results of filter compression with DCT, Gaussian Derivative (GD), Fourier-Bessel (FB) and PCA bases. All 3×3 filters are compressed, showing VGG-16 (**left**) and ResNet-50 (**right**). Dashed lines show progressive selection of bases, where deeper layers have the same or lower number of bases compared to earlier layers.

semantic meaning. A common approach to solve other complex tasks is to transfer the knowledge learned by classification models. Models trained on ImageNet or another large dataset are used to initialize models that require a finer prediction, and are consequently fine-tuned to the task. In this section we aim to demonstrate that representations learned from features expressed via harmonic bases are versatile and can be transferred to other related tasks the same way as standard CNNs. We assess here the performance of harmonic networks in object detection (Section 4.5.1), instance segmentation (Section 4.5.3), semantic segmentation (Section 4.5.4) and boundary detection tasks (Section 4.5.5).

4.5.1 Object Detection and Segmentation

For detecting objects in images well established single stage RetinaNet [150] and multistage Faster [13] (for scenarios when detecting only bounding boxes) and Mask R-CNN [151] (when masks are available as well) frameworks are built upon our harmonic ResNet backbones. A set of experiments is conducted on the datasets Pascal VOC [152] (Section 4.5.2) and MS COCO [153] (Section 4.5.3).

4.5.2 Pascal VOC

We extend PyTorch implementation provided by Chen et al. [154] and train Faster R-CNN model based on our harmonic ResNets with 50 and 101 layers. The first layer and the set of residual blocks used on the highest feature resolution are frozen (not updated) during the training. The region proposal network (RPN) is applied on the feature pyramid [155] constructed from the network layers. RPN layers as well as regression and classification heads are randomly initialized and use standard (non-harmonic) convolution/fully connected layers. Images are resized to set their shortest sides at 600 pixels. The Faster R-CNN is trained with the learning rate $lr = 0.01$ and batch size $bs = 16$ when the backbones have

50 layers, and $lr = 0.0075$ and $bs = 12$ in case of 101 layered configuration. Models are trained on the union of VOC 2007 training and validation sets with about 5000 images for 17 epochs, decreasing the learning rate by a multiplicative factor of 0.1 after epoch 15. We train the networks with original and harmonic backbones using the same setting. Additionally, these models are also trained on the combination of training sets of VOC 2007 and VOC 2012, consisting of about 16 500 images, for 12 epochs with learning rate dropped at epoch 9. All models are tested on VOC 2007 test set and the official evaluation metric, the mean average precision (AP), is averaged over 5 runs. The final results are reported in Table 4.15 for different depths of ResNet backbones, detector architectures and configurations of the datasets. The specific number of epochs and learning schedule were selected to reduce the impact of overfitting. In particular, when trained on VOC 2007 the larger ResNet-101 and its harmonic version reported a decrease in test performance of approximately 0.5 AP if the training progressed to epoch 20.

Table 4.15: Mean average precision of Faster R-CNN models after 5 runs on Pascal VOC07 test set. ResNet-101-based models are trained once.

Backbone	**Box AP↑ VOC07 17 epochs**	**Box AP↑ VOC07+12 12 epochs**
ResNet-50	73.8 ± 0.3	79.7 ± 0.3
Harm-ResNet-50	**75.0 ± 0.4**	**80.7 ± 0.2**
Harm-ResNet-50-AVG	74.7 ± 0.3	**80.7 ± 0.2**
ResNet-101	76.1	82.1
Harm-ResNet-101	**77.4**	**82.9**

From Table 4.15 we conclude that the models built on our harmonic backbones surpass their conventional convolution-based counterparts in all configurations as well as on both training sets. In Figure 4.25 we demonstrate the performance of Faster R-CNN at different stages of training on VOC 2007 and VOC 2007+2012 datasets. Harmonic ResNet-50 with average pooling following the root layer yields comparable results, yet has a mildly higher computational complexity compared to ResNet that uses striding instead of pooling. We observe a consistent improvement due to the Harmonic architecture: by 1% mAP for ResNet-50 and 0.8% mAP in case of ResNet-101 using the Faster R-CNN architecture.

4.5.3 MS COCO

MS Common Objects in COntext (COCO) dataset poses a greater challenge due to a higher variety of target classes and generally smaller object sizes. Higher input resolution is needed for good accuracy. The networks are trained following the standard procedure, images resized so that their shortest size is 800 pixels. The learning rate is initialized by linear scaling method $lr = 0.02 \times (bs/16)$ using the default hyperparameters set up by Chen et al [154]. Batch size $bs = 10$ and learning rate $lr = 0.0125$ are used for shallower models while deeper models required batches of size $bs = 6$ with the corresponding learning rate of

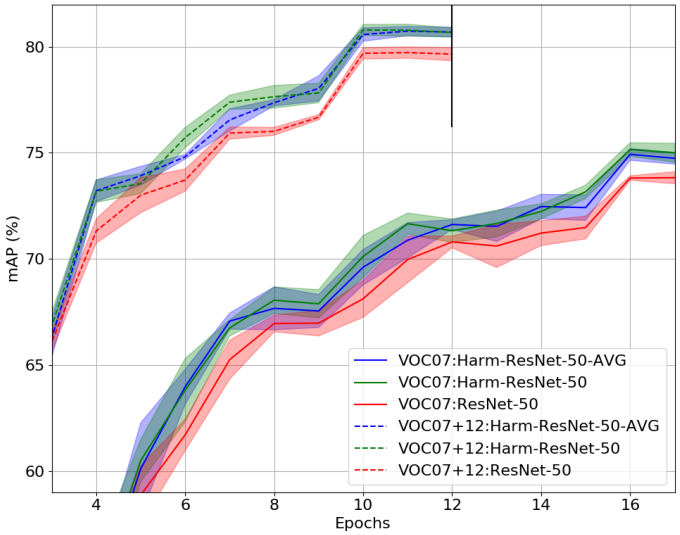


Figure 4.25: Training Faster R-CNN with harmonic network backbone on Pascal VOC detection task. Mean average precision on VOC 2007 validation set reported with standard deviation after training on VOC 2007 train (solid) and VOC 2007 + VOC 2012 train (dashed) sets.

$lr = 0.0075$. All models are trained with standard 12 (24) epochs schedules with learning rate decreased by 10 after epochs 8 (16) and 11 (22). Table 4.16 shows that the use of our harmonic backbones consistently improves both single-stage RetinaNet and multi-stage Faster and Mask R-CNN detectors by 0.7-1.3 AP with identical training procedures employed.

The state-of-the-art detectors rely on a cascade of detection heads with progressively increasing IoU thresholds, which refines the bounding boxes and thus improves localization accuracy [156]. In Table 4.17, we report comparisons achieved with the Cascade R-CNN architecture, trained using the 20-epoch schedule suggested in [156]. The use of our harmonic ResNet-101 provides a 1.0 AP improvement for object detection similar to Faster & Mask R-CNNs, and it also improves instance segmentation average precision by 0.7 AP (see Table 4.17). Moreover, a similar improvement of 1.1 AP is observed for hybrid task cascade R-CNN [157] that alters the mask refinement procedure and exploits semantic segmentation information to incorporate additional contextual information. We show several visual results of this model in Appendix A.

These experiments on object detection and localization demonstrate that the harmonic versions of the backbones provide a meaningful improvement of about 1.0 AP in terms of both bounding boxes and masks to the state-of-the-art detection architectures. Our harmonic networks retain this improvement purely from the classification task through the transformation to the Feature Pyramid Networks (FPNs). The variety of tested detectors indicate that harmonic network backbones have a potential to also improve other detection models and can be transferred to other datasets.

Table 4.16: Mean average precision for different backbones and detector types on MS COCO 2017 validation set. All backbones are transformed to FPNs.

Backbone	Type	Box AP ↑		Mask AP ↑	
		12 epochs	24 epochs	12 epochs	24 epochs
ResNet-50	Faster	36.4	37.7*	-	-
Harm-ResNet-50	Faster	37.2	**38.4**	-	-
Harm-ResNet-50-AVG	Faster	**37.3**	38.2	-	-
ResNet-50	Retina	35.6*	36.4*	-	-
Harm-ResNet-50	Retina	36.3	36.8	-	-
Harm-ResNet-50-AVG	Retina	36.3	37.1	-	-
ResNet-101	Faster	38.5	39.3	-	-
Harm-ResNet-101	Faster	**39.7**	**40.3**	-	-
ResNet-101	Retina	37.7*	38.1*	-	-
Harm-ResNet-101	Retina	39.0	39.2	-	-
ResNet-50	Mask	37.3*	38.5*	34.2*	35.1*
Harm-ResNet-50	Mask	**38.1**	**38.9**	34.7	**35.5**
Harm-ResNet-50-AVG	Mask	**38.1**	38.8	**35.0**	35.4
ResNet-101	Mask	39.4*	40.3*	35.9*	36.5*
Harm-ResNet-101	Mask	**40.7**	**41.5**	**36.8**	**37.3**

*scores reported by [154].

4.5.4 Semantic Segmentation

Semantic image segmentation is related to instance segmentation. The difference lies in classifying each image pixel to its exclusive class, and all entities of the same type share the same label. We have tested the harmonic network capabilities for semantic segmentation task on a Pascal VOC 2012 benchmark. Training images are augmented into a set of 10,582 samples [158]. Performance is measured in terms of intersection over union (IoU) on 1449 images large validation set. For segmentation engine DeepLabV3 model [159] is used. Our code is extending a third party PyTorch implementation[6], which reaches comparable results to those reported in the paper [159]. We have retrained the baseline model with ResNet-50 and ResNet-101 backbones for 30,000 iterations with batch size 16, learning rate 0.1 and output stride parameter equal to 16. The model also uses atrous spatial pyramid pooling (ASPP) but no multi-grid [159]. We have replaced the backbone model of the segmentation network with harmonic ResNets based on orthogonal DCT either converted from the original backbone models or by harmonic models (from Table 4.11) pre-trained on ImageNet (90 epochs). The DeepLabV3 models with harmonic backbones pretrained on ImageNet improve IoU scores by more than 1.1% (Tab 4.18). This experiment also validates the application of harmonic blocks with dilated convolution. DeepLabV3 with harmonic backbone improves on average classes "chair" by 9.5%, "sheep" by 3.9%, "boat" by 3.2%, "cow" and "tvmonitor" both by 2.9%, while lacks at classes "pottedplant" by 3.4% and "aeroplane" by 0.8%. Some visual examples are depicted on Figure 4.26.

[6] `https://github.com/VainF/DeepLabV3Plus-Pytorch`

Table 4.17: Mean average precision on Cascade R-CNN architecture on MS COCO 2017 validation set. All backbones are transformed to FPNs.

Cascade R-CNN Backbone	Type	Box AP ↑ 20 epochs	Mask AP ↑ 20 epochs
ResNet-101	Faster	42.5*	-
Harm-ResNet-101	Faster	43.5	-
ResNet-101	Mask	43.3*	37.6*
Harm-ResNet-101	Mask	44.3	38.3
ResNet-101	Hybrid	44.9*	39.4*
Harm-ResNet-101	Hybrid	**46.0**	**40.2**

*scores reported by [154].

Table 4.18: Intersection over Union (IoU) of DeeplabV3 architecture semantic segmentation on Pascal VOC 2012 validation set. IoU is shown as median of 5 trials ± empirical standard deviation.

DeeplabV3 Backbone	IoU ↑
ResNet-50	76.31 ± 0.07
Harm-ResNet-50 converted	76.65 ± 0.07
Harm-ResNet-50	77.40 ± 0.08
Harm-ResNet-50 (avgpool)	77.62 ± 0.37
ResNet-101 [159]	77.21
ResNet-101	78.31 ± 0.07
Harm-ResNet-101 converted	77.92 ± 0.11
Harm-ResNet-101	79.49 ± 0.29

4.5.5 Boundary Detection

In this section we demonstrate the flexibility of DCT bases for a non-classification related task, namely object contour regression. The challenge here is to distinguish meaningful edges from edges representing a texture. Data and annotations are provided by the Berkeley Segmentation Data Set and Benchmarks (BSDS500) [160]. Our model is built on top of Holistically Nested Edge Detection (HED) [15], which is based on VGG-16 network [139]. Our code originated from a revised third party PyTorch implementation[7]. This model learns multi-scale representations, and at each scale a boundary is predicted as a linear combination of features from the deepest layer at that scale. Predictions generated from the subsampled features are upsampled with bilinear interpolation to match the ground-truth resolution. The final prediction is estimated as a weighted average of all the side outputs. A cross-entropy between each side output and the ground-truth contributes to the loss function. Following Xie et al. [15], annotations from different annotators are merged based on majority vote. Images are augmented into 3 scales, 16 rotations and two horizontally mirrored versions. Training is performed with Adaptive Momentum Optimizer.

[7] `https://github.com/meteorshowers/hed`

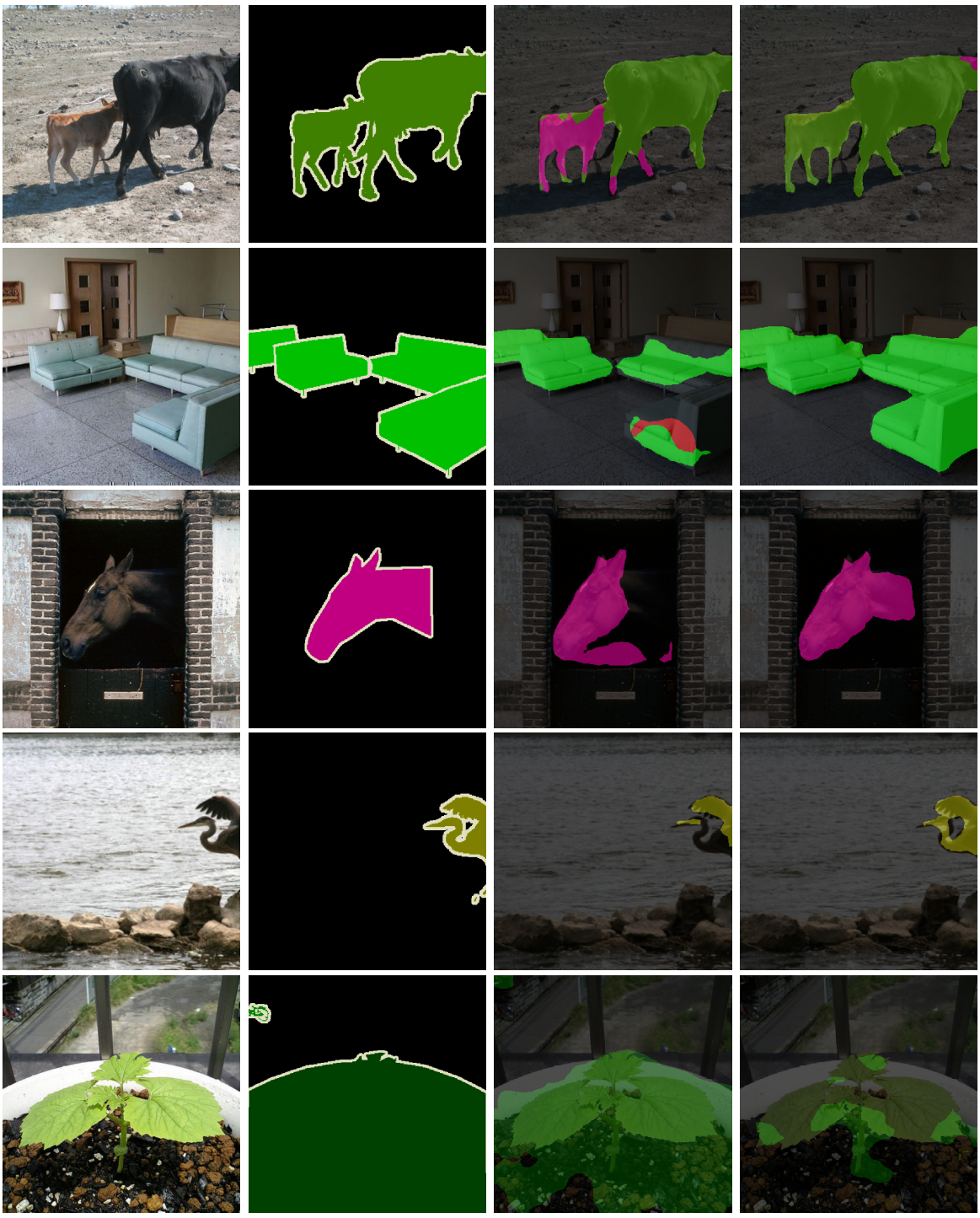


Figure 4.26: Examples of semantic segmentation on Pascal VOC 2012 validation images. The first 4 rows show where harmonic network is more successful than the baseline, while the last row displays case where it fails. DeepLabV3 with ResNet-101 backbone is used.

Overlap of the predictions with human annotations is measured as an F-score on a precision-recall curve. Two metrics are standardly reported on this dataset: the Optimal Dataset Scale (ODS) using fixed contour threshold, and the Optimal Image Scale (OIS) that is calculated selecting image-best thresholds. Following [15], edges are thinned with non-maximum suppression prior to evaluation. Our re-implementation of HED with random weight initialization scores 0.761 ODS and 0.778 OIS (see HED in Table 4.19). Replacing

Table 4.19: Boundary detection results measured by ODS and OIS (higher is better). Our proposed models are denoted as Harm-HED.

Method	Layers/Chan.	Params	ODS ↑	OIS ↑
Random initialization				
HED	13 / 64	14 626k	0.761	0.778
Harm-HED	13 / 64	14 626k	**0.770**	**0.789**
HED	10 / 7	113k	0.738	0.756
Harm-HED	10 / 7	113k	0.743	0.763
Harm-HED λ=2	10 / 7	38k	0.740	0.758
Harm-HED progr.λ	10 / 10	118k	0.751	0.770
H-Net [29]	10 / -	116k	0.726*	0.742*
DynResNet [161]	-	-	0.732*	0.751*
Pretrained				
HED	13 / 64	14 626k	0.777	0.796
Harm-HED	13 / 64	14 626k	0.782	0.803
HED [15]	13 / 64	14 626k	0.790*	0.808*

*scores reported by authors of the corresponding papers.

all the filters with harmonic blocks based on Alg. 1 improves both metrics: ODS to 0.770, and OIS to 0.789 (see Harm-HED in Table 4.19). The qualitative comparison presented in Figure 4.27 demonstrates how the harmonic version of HED handles better the suppression of texture-related edges, preserving the relevant object boundaries. Initialization with a pretrained model typically plays an important role in this task. The quantitative comparison is repeated after initializing weights using the VGG-16 weights learned on ImageNet. Harmonic HED was initialized by the same set of weights using the network conversion to DCT bases. This initialization provided a minor gain in performance, still falling behind the results reported by [15], see scores in Table 4.19. Interestingly, our Harmonic HED still reaches better performance even when both models are initialized the same way.

Moreover, comparisons with two recent works based on steerable filters for boundary detection are included. First, locally adaptive filtering for boundary detection was employed using a shallow residual network with dynamic steerable blocks [161]. These blocks leverage steering properties of Gaussian derivative bases which linearize certain transformations. Second, Worrall at al. [29] have modified HED architecture with complex circular harmonic bases to detect boundaries via representations equivariant to rotations. Both of these implementations use notably smaller models. To match these we downscale our Harmonic HED architectures by limiting the number of channels and network depth similarly to [29]. Using DCT bases slightly improves the small baseline model that already performs reasonably well, see Table 4.19. Note, that different hyperparameters have been used to train these harmonic nets. The equivariant harmonic network [29] has complex bases that require complex convolution and learning parameters for imaginary components as well. Increasing the number of channels (from 7 to 10) in our Harmonic HED while using progressive compression leads to a model with comparable number of parameters yet better performance

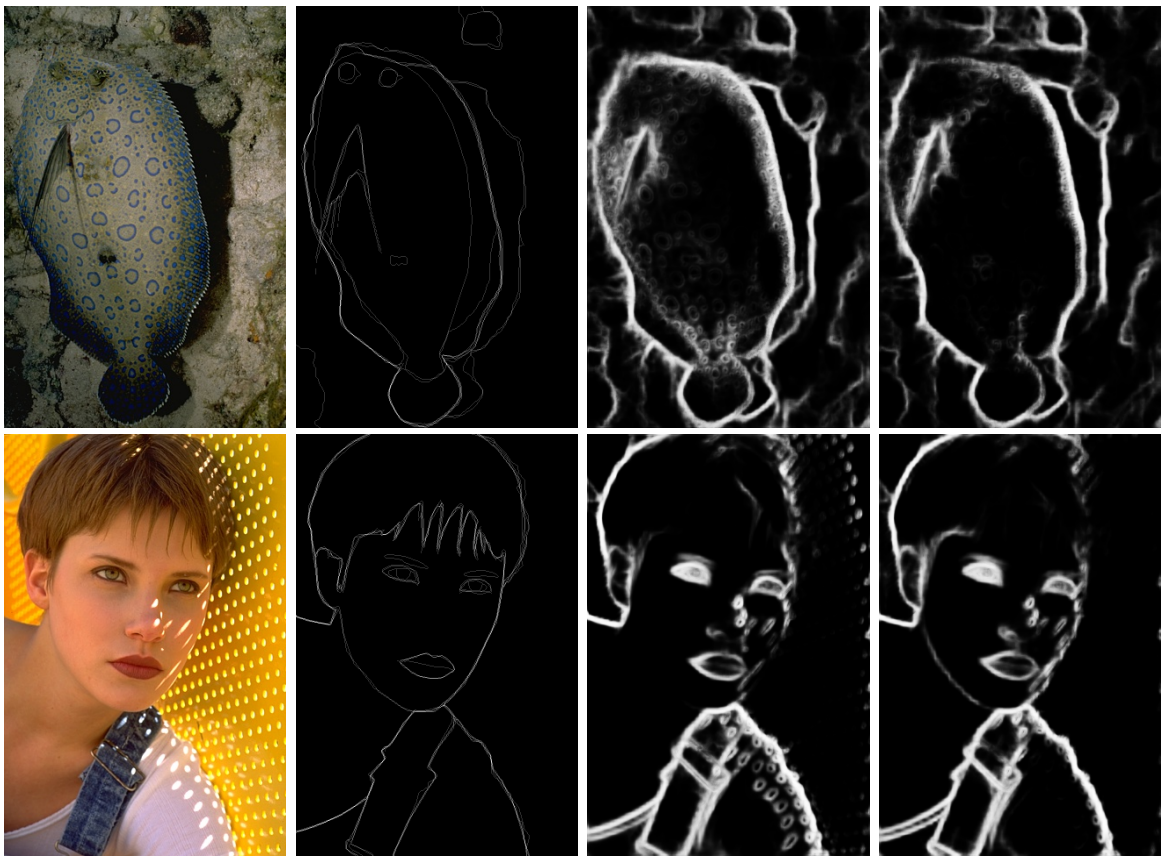


Figure 4.27: Qualitative results of harmonic HED model trained from randomly initialized weights. Our approach (Harm-HED) is better at suppressing texture-imposed edges.

due to more rich feature space. We observe that our 'static' approach based on DCT bases without explicitly built-in invariance can detect boundaries well compared to the benchmark methods.

From the consistent improvement of harmonic networks over the beseline models on a variety of tasks we can expect that the proposed method can be successfully applied to other related tasks as well.

4.6 Conclusion

We have presented a novel approach to explicitly incorporate spectral information from DCT into CNN models. We have empirically evaluated the use of our harmonic blocks with the well-established state-of-the-art CNN architectures, and shown that our approach improves results (e.g. accuracy & complexity) for a range of applications including image classification, segmentation and boundary detection. We also ascertain that harmonic networks can be efficiently set-up by converting the pretrained CNN baselines. The use of DCT allows one to order the harmonic block parameters by their significance from the most relevant low frequency to less important high frequencies. This enables efficient model compression by parameter truncation with only minor degradation in model performance.

This has been shown to be particularly useful for tasks with limited training samples.

Chapter 5

Compressing along the Channel Dimension

5.1 Introduction

A wide variety of methods have been proposed in the recent years to perform structured (removing entire blocks of filters) and unstructured (suppressing individual weights) pruning of CNNs. Such studies are central to addressing the design of memory- and computationally-efficient implementations of the state-of-the-art CNNs. It has been pointed out in multiple works [2, 3, 162] that standard CNN architectures are typically highly over-parameterized such that pruning of some 5-20% of parameters may often result in networks performing better than the original architectures. This impact can be associated with better optimization and regularization. Iterative pruning [4, 163, 164] and access to substantial fine-tuning [5, 6, 165] or even re-training from scratch [7, 163] allows many techniques to arrive at the (nearly) original levels of performance at the cost of additional 30% to 300% of the original amount of training. This is achieved after dropping between 30 and 70% of the weight parameters in general-purpose architectures like VGG [59] and ResNets [63], and 5-20% on architectures designed to run on devices with limited resources, like MobileNets [61].

In Chapter 4 we have introduced frequency-based structured pruning of local layers trained on certain domains such as natural images. This chapter proposes unstructured pruning that is independent of the data distribution.

The vast majority of the existing pruning methods pursue the reduction of computational complexity related to training or deployment of CNNs as the main goal and consider the decrease of parameter count (and hence memory footprint) of the pruned architectures as secondary [2, 162]. Contrary to these approaches in this Chapter we focus specifically on the parameter footprint aspect and consider scenarios when very limited or no retraining may be available to obtain better performing compressed models. The motivation for such goal is

two-fold. Firstly, we consider the storage implications (and overheads related to compressing the original models) for models both in trained architecture repositories, i.e. model zoos, and mobile devices. Secondly, by reducing the trainable parameter count in the models we expect to retain the general modeling capacity and increase the degree of regularization for optimizing the compressed architecture thus reducing the impact of over-fitting. Unlike most of the state-of-the-art methods we propose a non-iterative technique and resort to a strictly limited amount of fine-tuning to compensate for the impact of pruning: we let all our compressed networks train for a single epoch post-compression.

For the general state-of-the-art review on CNN pruning we refer the readers to more comprehensive recent overviews conducted in [162, 166]. Several recent studies [8, 166] have also investigated full-cycle compression including on top of CNN-specific elements the general instruments like quantization and Huffman coding thus achieving the reduction of memory footprint of 10-50 times for AlexNet, VGG16, ResNet-50. In our work we do not investigate specifically the memory footprint but rather the parameter count, assuming that the stronger parameter reduction generally leads to better compressibility. Note however that both quantization and Huffman coding may be used to produce memory efficient representation for our results.

Several works have explored links between Convolutional Neural Networks (CNNs) and wavelet decomposition [27, 112], for details see Section 2.4. Compact filters were designed by their interpretation in terms of specific bases that we have reviewed in Section 2.5. In Chapter 4 we have achieved efficient CNN compression by casting spatial filters into DCT bases. In this chapter we aim to reduce redundancies due to correlation between filters in the same network layer, and to this end we use spectral representation where we perform the weights compression. We demonstrate that DCT can be used to compress more efficiently if deployed along the channel dimensions (Section 5.2) compared to the more common spatial filter compression. Our approach relies on channel reordering to maximise the representation capacity of DCT encoding such that efficient compression can be achieved by suppressing high frequencies, see Figure 5.1. The proposed scheme requires a decompression step, which is encoded via CNN operations at run-time. In case when the resources are limited or no training data is available we propose a correction for the batch normalisation parameters to adjust to the pruned CNN without any fine-tuning (Section 5.7). Frequency based pruning [167] has been previously used to compress convolutional filters based on their spatial correlation, however did not attempt to compress layers with 1×1 kernels or fully connected layers. The approach is dynamic and prunes DCT coefficients based on their magnitude. Our static approach is truncating coefficients only by their frequency and does not require extensive fine-tuning.

The implementation of the proposed compression technique is assessed in terms of accuracy vs. number of parameters, against a panel of state-of-the-art pruning approaches including [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. We present strong results with our novel CNN

compression approach when applied to ResNet-50 [63] and MobileNet-V2 [61] architectures for the task of image classification on ImageNet ILSVRC-2012 dataset [101] (Section 5.3). This selection of these two state-of-the-art architectures on a large challenging dataset provides a suitable validation for the proposed compression technique. We achieve the original ResNet-50 performance with 32% of the original parameter footprint (with 6% of trainable parameters) with just one epoch of post-compression fine-tuning. Furthermore, our compressed models without any fine-tuning achieve lower but competitive results on the par with some recent compression methods involving ample fine-tuning. For MobileNet-V2 architecture we achieve almost 50% parameter reduction with just 0.3% drop in top-1 accuracy.

5.2 Tensor compression via DCT

The vast majority of parameters representing the convolutional neural network are concentrated in convolutional filters or weight matrices of the fully connected layers. A set of filters representing a convolutional layer is typically expressed as a 4-dimensional tensor $\mathbf{w} \in \mathbb{R}^{m \times n \times k \times k}$ and is used to convolve each of the n input feature maps with its own filter of size $k \times k$, spanning m output feature maps:

$$\mathbf{y}_j = \sum_{i=0}^{n-1} \mathbf{x}_i * \mathbf{w}_{j,i}. \tag{5.1}$$

Many of the layers in recent architectures are so called resampling layers, whose filters have 1×1 spatial extent ($k = 1$), i.e., they merely reweight existing features. Fully-connected layers can also be treated as a special case of a convolutional layer with $k = 1$ that resample features consisting of a single scalar. Our aim is to compress the weights of a layer by deploying one-dimensional DCT transform. The columns of the transform matrix $C \in \mathbb{R}^{n \times n}$ represent the DCT basis functions, with elements $C(a, u)$:

$$C(a, u) = \sqrt{\frac{\alpha(u)}{n}} \cos\left[\frac{\pi}{n}\left(a + \frac{1}{2}\right) u\right], \quad \text{where } \alpha(u) = \begin{cases} 1, & u = 0 \\ 2, & \text{otherwise.} \end{cases} \tag{5.2}$$

Given a weight tensor $\mathbf{w} \in \mathbb{R}^{m \times n \times 1 \times 1}$, the output channel j ($j = 0, ..., m-1$) is calculated from the input using a weight vector $\mathbf{w}_j \in \mathbb{R}^n$. In $\mathbf{w}_j$ each element represents the weight of an input channel. The DCT matrix C is orthogonal, and therefore $\mathbf{w}_j = \mathrm{C}\mathrm{C}^T\mathbf{w}_j$. The weight vector $\mathbf{z}_j$ in the DCT domain $\mathbf{z}_j = \mathrm{C}^T\mathbf{w}_j$ has also n coefficients. We apply compression to each vector $\mathbf{w}_j$ by retaining its $t < n$ lowest frequencies. The number of DCT basis functions $t = \left\lfloor \frac{n}{r} \right\rfloor$ used to transform the weights is controlled by a hyperparameter r that represents the fraction of the complete basis set used after compression and is referred to as compression rate in the following. The reduced transformation matrix $\mathrm{C}_t \in \mathbb{R}^{n \times t}$ will produce reduced coefficient representation $\mathbf{z}_j^t = \mathrm{C}_t^T\mathbf{w}_j$. The tensor $\mathbf{z}^t$ is representing

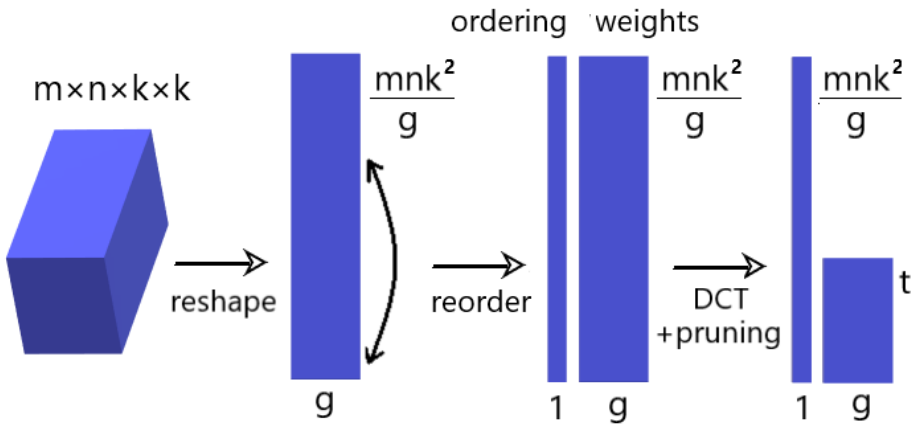


Figure 5.1: Tensor reordering and DCT compression.

our compressed layer in DCT domain, which is then used at runtime to reconstruct an approximation $\tilde{\mathbf{w}}$ of the original weight tensor $\mathbf{w}$ via $\tilde{\mathbf{w}}_j = \mathsf{C}_t \mathbf{z}_j^t = \mathsf{C}_t \mathsf{C}_t^T \mathbf{w}_j$.

It is well-known that concise Fourier and DCT representations are achieved on smooth signals, whereby discontinuities require a substantial number of additional coefficients in the representation. Similarly, compression based on frequency truncation assumes smooth transitions between neighboring elements. Unlike parameters at neighboring spatial locations [26], the weights corresponding to neighboring channels of a CNN are not expected to be correlated. From equation (5.1) it is clear however that the exact channel position does not matter as long as the filter responses are summed. For this reason, to apply DCT to such channels, we boost coherence between the input channels by reordering them. Our expectation is that the more similar the neighboring samples are, the fewer basis functions are required to encode the signal accurately.

Considering a 2D weight matrix $\mathbf{w} \in \mathbb{R}^{m \times n}$, the 1D DCT transform is to be applied m times for every row-vector $\mathbf{w}_{j,\cdot}$, $j \in \{0..m-1\}$ of size n. To boost the smoothness of $\mathbf{w}$, we deploy the ordering procedure outlined in Algorithm 2. Specifically, we reorder the weight matrix $\mathbf{w}$ along the second dimension of size n. The new sequence starts with the column-vector $\mathbf{w}_{\cdot,i}$, $i \in \{0..n-1\}$ with the largest magnitude. Each following vector is chosen to have the smallest distance $\mathcal{D}$ to its predecessor.

Algorithm 2: Weight matrix reordering

Input: $\mathbf{w} \in \mathbb{R}^{m \times n}$ (weight matrix)
$\mathbf{s} \in \mathbb{N}^n$;
$\mathbf{s}_0 \leftarrow \arg\max_{i \in \{0..n-1\}} \|\mathbf{w}_{\cdot,i}\|_2$;
$\mathbf{p} \leftarrow \{0..n-1\} \setminus \{\mathbf{s}_0\}$;
for $j \in \{1..n-1\}$ **do**
  $\mathbf{s}_j \leftarrow \arg\min_{i \in p} \mathcal{D}\left(\mathbf{w}_{\cdot,\mathbf{s}_{j-1}}, \mathbf{w}_{\cdot,i}\right)$;
  $\mathbf{p} \leftarrow \mathbf{p} \setminus \{\mathbf{s}_j\}$;
end
Output: $\mathbf{s}$ (ordering)

In case of output dimension $m = 1$ this procedure establishes the standard ordering from the largest to the smallest. The purpose of the proposed ordering is to boost smoothness between columns of the reordered tensor. This heuristic is based on an assumption that monotonic sequence has high correlation with the low frequency bases of the DCT. A more informed heuristics can be devised based on ordering the sequence to mimic one or a few of the basis functions or by considering order in individual rows. The smoothness is low for large values of the first dimension m, corresponding to the size of vectors being sorted. The natural approach to decrease m is to reshape $\mathbf{w}$ to $\mathbf{w}' \in \mathbb{R}^{g \times mn/g}$ such that the first dimension is $g < m$ - the number of groups we reshape the layer into. The reshaping operation is defined as $\mathcal{R}_{m \times n \to g \times mn/g}\mathbf{w}$ and can be formally described as inverse vectorization $\text{vec}^{-1}_{g,mn/g} \circ \text{vec}\ \mathbf{w}$. This transformation allows higher smoothness of transitions inside the reordered $\mathbf{w}$ for lower values of g. Algorithm 3 describes the compression procedure with reshaping, applied to arbitrary square kernel of size k. Figure 5.1 shows that reordering the weight tensor requires storing the ordering (index vector) to reverse this manipulation. The size of this vector is mn/g, thus small values of g span long index vector $\mathbf{s}$. This means that attaining higher levels of smoothness requires storing more additional information.

Algorithm 3: Weight matrix compression

Input: $\mathbf{w} \in \mathbb{R}^{m \times n \times k \times k}$ (weight matrix)
$\mathbf{w}' \leftarrow \mathcal{R}_{m \times n \times k \times k \to g \times mnk^2/g}\mathbf{w}$; (reshape)
$\mathbf{s} \leftarrow$ Alg. 2 $(\mathbf{w}')$;
$\mathbf{w}' \leftarrow \mathbf{w}'_{\cdot,\mathbf{s}}$; (ordering)
$\mathbf{z}^t \in \mathbb{R}^{g \times t}$;
for $j \in \{0..g-1\}$ **do**
  $\mathbf{z}^t_j \leftarrow C_t^T \mathbf{w}'_j$;
end
Output: $\mathbf{z}^t, \mathbf{s}$ (DCT weights, ordering)

To perform inference and training it is essential to decompress the weight tensor into $\tilde{\mathbf{w}}$ by applying the inverse DCT (iDCT) to the stored coefficients, reordering the vectors to their original position and reshaping the weight tensor to its original shape, see Algorithm 4. The procedure consists of differentiable operations and allows gradient backpropagation. Reordering, permutation and the inverse DCT transformation generate some additional overhead on top of convolutions, and this overhead is usually not required for the structured pruning. Small values of g increase complexity of the iDCT, but we employ fft algorithm having complexity $\mathcal{O}\left(mnk^2 \log \frac{mnk^2}{g}\right)$ that has only a subtle impact on the total number of operations. It is also important to note that the post-compression parameter footprint created by our approach consists of trainable parameters and order parameters $\mathbf{s}$. The latter are not changing throughout the fine-tuning phase. The fraction of trainable parameters can be between 15% (low g) to 90% (high g) of the total.

Algorithm 4: Weight matrix decompression

Input: $\mathbf{z}^t \in \mathbb{R}^{g \times t}$ (DCT weights)
$\mathbf{s} \in \mathbb{N}^{mnk^2/g}$ (ordering)
$\mathbf{w}' \in \mathbb{R}^{g \times mnk^2/g}$;
for $j \in \{0..g-1\}$ **do**
  $\mathbf{w}'_j \leftarrow \mathsf{C}_t\ \mathbf{z}^t_j$; (inverse DCT)
end
$\mathbf{w}' \leftarrow \mathbf{w}'_{\cdot,\mathbf{s}^{-1}}$; (inverse ordering)
$\tilde{\mathbf{w}} \leftarrow \mathcal{R}_{g \times mnk^2/g \rightarrow m \times n \times k \times k}\mathbf{w}'$; (reshape)
Output: $\tilde{\mathbf{w}}$ (decompressed weights)

5.3 Implementation

To evaluate the proposed compression scheme outlined in Algorithm 3 we perform experiments with recent ResNet-50 and MobileNet-V2 models trained on ImageNet-1k dataset for the task of image classification. We investigate the application of compression to spatial $k \times k$ and 1×1 layers with the two core compression strategies: uniform, whereby the same compression rate r and number of groups g are used for all layers, and progressive, that adjusts r and g according to the size of the layer.

We have used PyTorch framework to implement the compressed models and the online layer decompression outlined in Algorithm 4. The pretrained models used in our experiments come from Torchvision model repository [137]. The empirical runtime overhead (due to compression) during training is measured to be 37% for ResNet-50 and 17% for MobileNet-V2.

Ablation studies and performance analysis for the designed compression are investigated separately for 1×1 layers (Section 5.4) and spatial layers (Section 5.5). In Section 5.6 we present the full network compression strategy and outline the batch normalization correction in Section 5.7.

5.4 Compressing resampling layers

We start by investigating the compression capacity of the resampling 1×1 layers. Firstly we validate the hypothesis that sorted vectors are approximated more efficiently. A subset consisting of the first 9 input and 9 output channels (81 scalar values) is dissected from weights in the first 1×1 layer of ResNet-50. Vectorization of this sequence in the default order is visualized in Figure 5.2 (**left**). Instead of trying to reconstruct it in this form we rather attempt to approximate its sorted version (Figure 5.2 **right**).

Figure 5.3 clearly shows that decent reconstruction of the sorted sample sequence can be achieved by using as few as 2 basis functions, while to get comparable results on the original sequence at least half of the basis functions are needed. The common magnitude pruning (ℓ_1 prune) in space spanned by the standard bases require many more parameters for a

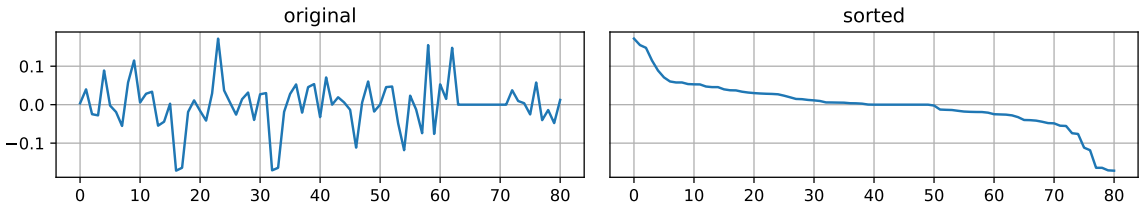


Figure 5.2: Vectorized subset of weights of the first 1×1 layer of pre-trained ResNet-50 in the default order (**left**) and after reordering (**right**). The subset consists of the first 9 input and the first 9 output channels (out of 64).

good reconstruction.

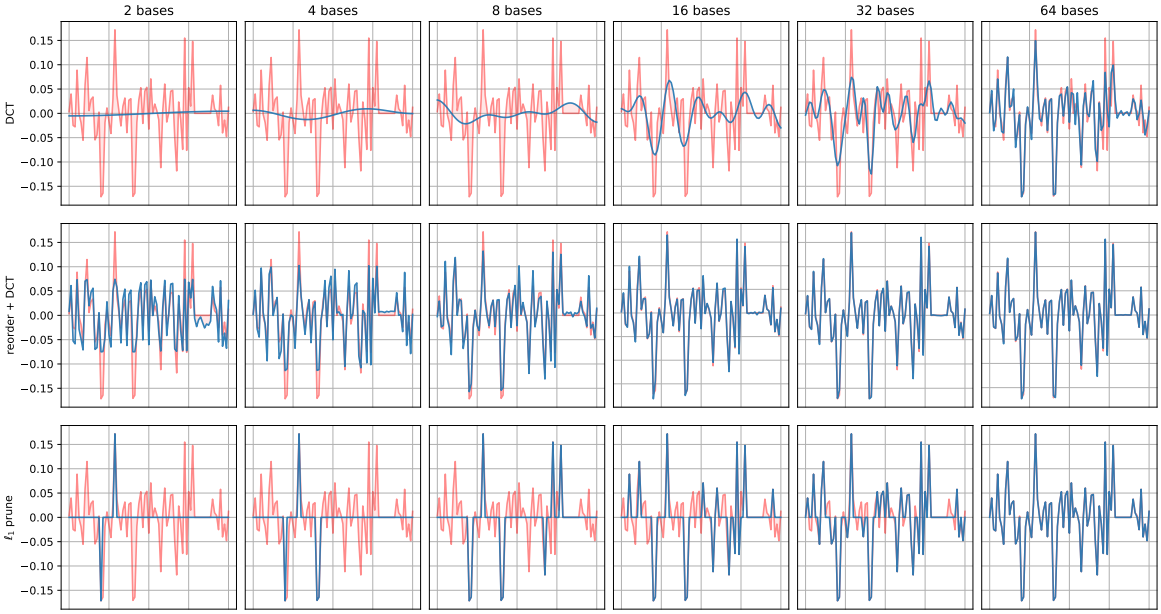


Figure 5.3: An approximation (blue) of the vectorized subset of ResNet-50 1×1 layer weights (red). Subsets of basis functions are used to represent weights in the DCT space without (top) and with (middle) reordering, and are compared to standard bases magnitude ℓ_1 pruning (bottom).

It is worth to investigate whether DCT is a good choice for a transformation method on ordered signals. We conduct a comparison with a few popular transformation methods, these include Discrete Fourier Transform (DFT)

$$C_{\mathrm{DFT}}(a,u) = \sqrt{\frac{1}{n}} e^{\frac{-2\pi j a u}{n}} = \sqrt{\frac{1}{n}} \left[\cos\left(\frac{2\pi a u}{n}\right) - j \cdot \sin\left(\frac{2\pi a u}{n}\right) \right], \tag{5.3}$$

Discrete Sine Transform (DST)

$$C_{\mathrm{DST}}(a,u) = \sqrt{\frac{\alpha(u)}{n}} \sin\left[\frac{\pi}{n}\left(a+\frac{1}{2}\right)(u+1)\right], \quad \text{where } \alpha(u) = \begin{cases} 1, & u = n-1 \\ 2, & \text{otherwise.} \end{cases} \tag{5.4}$$

and a basis set constructed from the Gaussian function

$$g(a) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2} \tag{5.5}$$

by collecting higher-order derivatives (GD). Gaussian derivatives are not orthogonal, we orthogonalize the transformation matrix by QR factorization. Shape of this basis depends on parameter σ. For illustration the first 6 frequencies/orders are plotted in Figure 5.4.

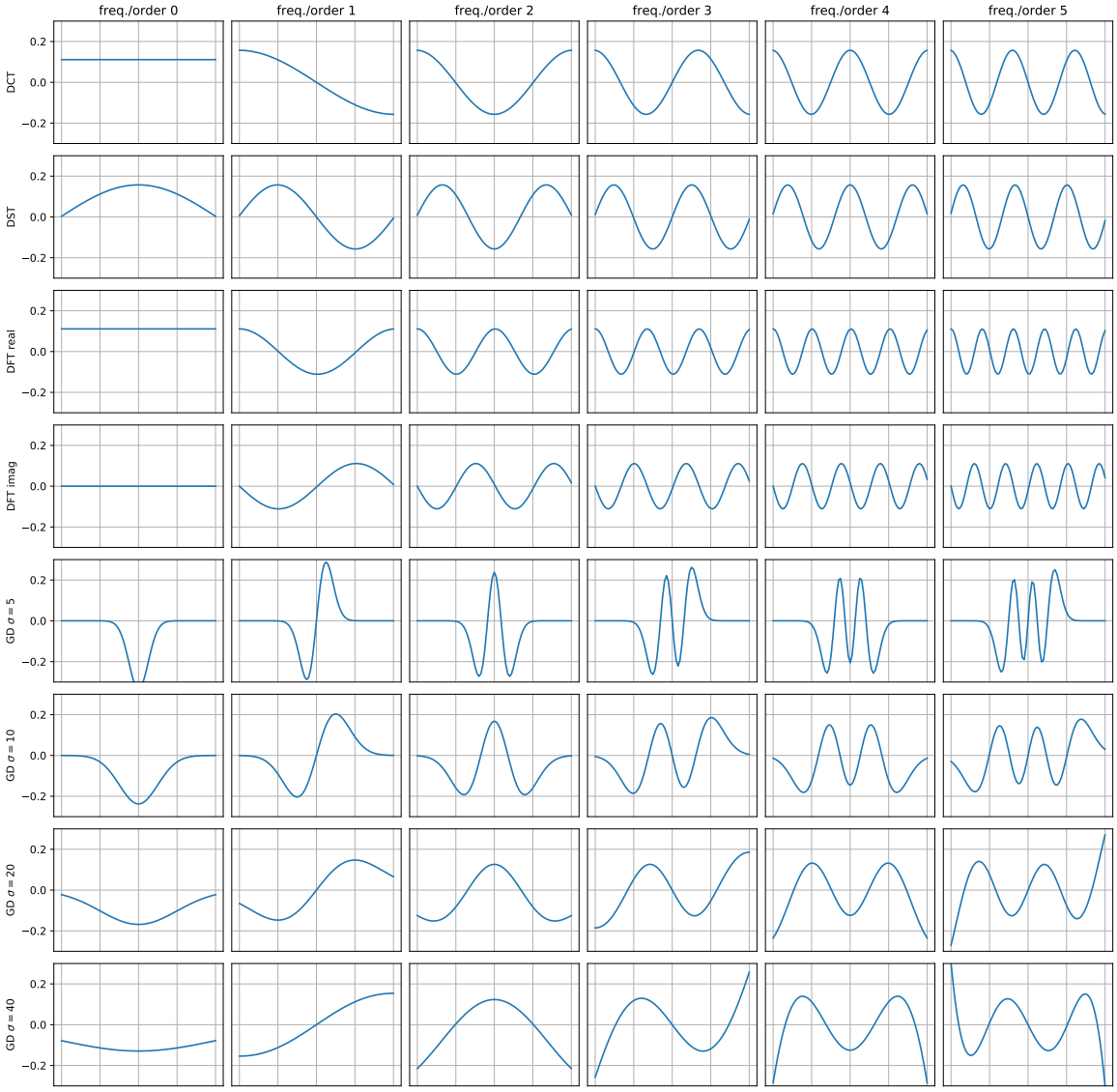


Figure 5.4: The first 6 basis function with the lowest frequency/derivation order of DCT, DST, DFT (real and imaginary part) and Gaussian derivative (GD) bases.

To compare DCT bases with the other mentioned bases we reconstruct the sorted weights by 2, 4, 8, 16, 32 and 64 basis functions. The implementation of DFT uses only the first half of the frequencies, keeping their symmetric versions aside. Since the DFT bases are complex-valued, multiple coefficient selection strategies can be devised. We have tested three: "real" that firstly selects all real-valued coefficients and continues with imaginary part, "imag" prioritizes imaginary component (ignoring the zero frequency) and "complex" selects

both according to their frequency. We have observed (Figure 5.5) that for a good result the imaginary component is more important than the real component. The reconstruction quality seems to be dependent on shape of the basis function representing the frequency 1, which gives the best results if is monotonic, hence resembles shape of the sorted vector. DCT fulfills this criteria and shows the most accurate reconstructions (see Figure 5.5). Comparable results are achieved by Gaussian derivative bases with large σ. As the sigma increases, shape of the basis functions more resembles DCT bases, which in turn can be calculated using efficient algorithms.

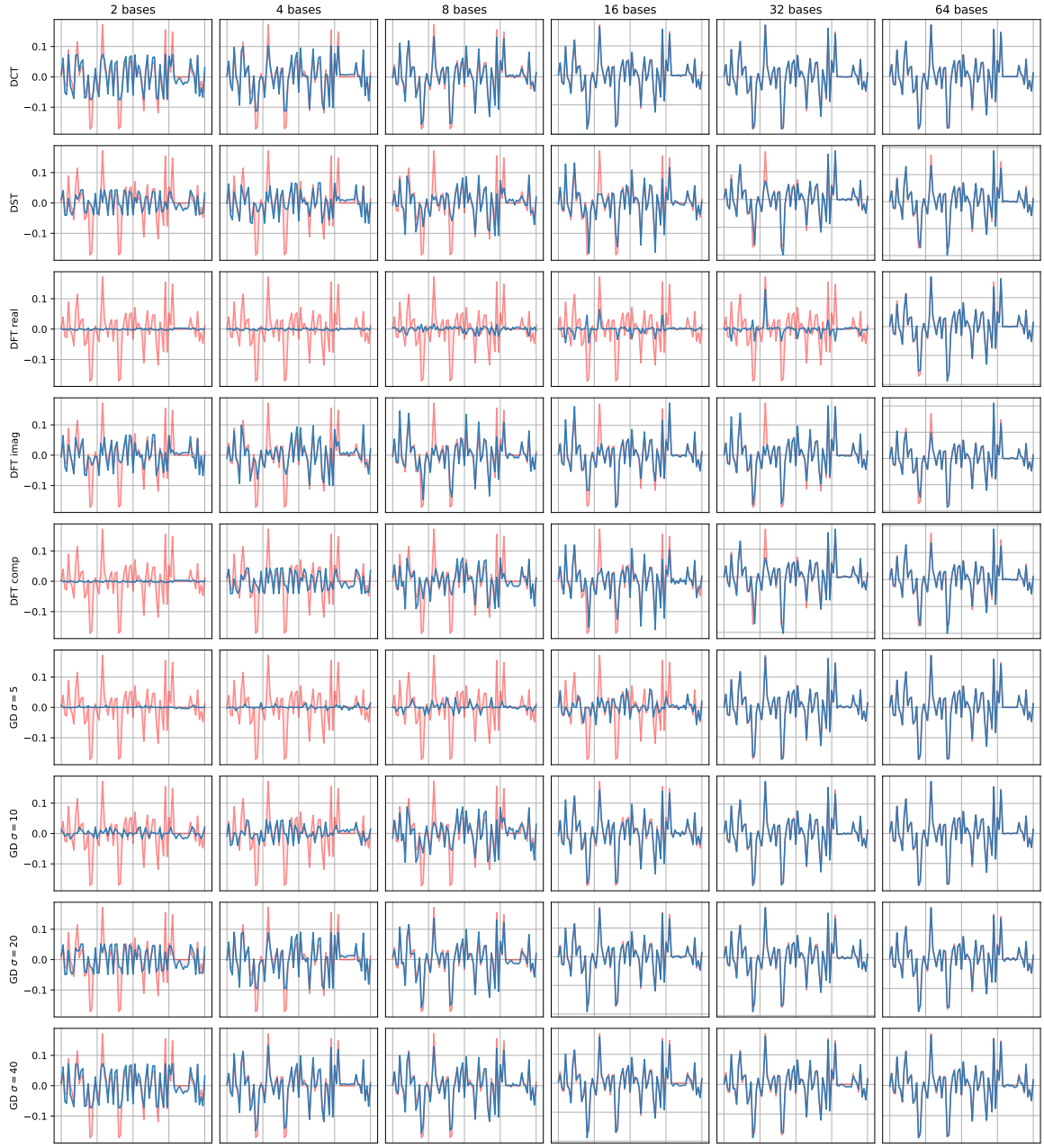


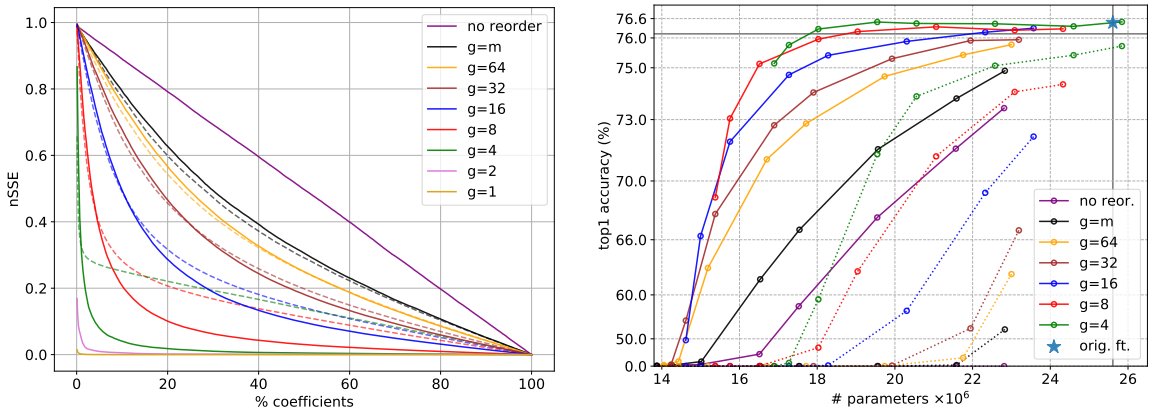
Figure 5.5: Approximation of the vectorized subset of ResNet-50 1×1 layer weights. The reconstructed signal is depicted with (blue) color and original signal with (red). Reconstructions are done with DCT, DST, DFT (prioritizing real, imaginary, or complex numbers) and Gaussian derivatives (GD) with various values of σ.

Various measures can be used to assess quality of compression. We employ the following two metrics: reconstruction error as normalized summed squared error nSSE (5.6), and accuracy loss (overall network performance) on validation set.

$$nSSE(\mathbf{w}, \tilde{\mathbf{w}}) = \frac{\|\text{vec}(\mathbf{w}) - \text{vec}(\tilde{\mathbf{w}})\|_2^2}{\|\text{vec}(\mathbf{w})\|_2^2}, \tag{5.6}$$

5.4.1 Reconstruction error

We experiment with one of the 1×1 layers of trained ResNet-50. The nSSE measured for the baseline compression without reordering the weights shows linear degradation as the fraction of used DCT coefficients decreases (Figure 5.6a), signifying that the energy is not concentrated in lower frequencies as in many natural signals. The reordering procedure results in achieving lower nSSE given the same coefficient budget. Reshaping the tensor and decreasing the number of vectors g, which are transformed by DCT, further improves the result. We have experimented with ℓ_1 and ℓ_2 norms for selection of the initial point for reordering, to find that both worked comparably, thus we have adopted the ℓ_2. For the distance $\mathcal{D}$ that drives the reordering procedure we have considered Euclidean, Manhattan and Cosine distance. Manhattan distance shows slightly inferior results to the Euclidean. Cosine distance proved to be the best for large dimensional vectors $g \geq 32$, while for smaller values of g a better nSSE is obtained using Euclidean distance.



(a) Reconstruction error

(b) Classification accuracy

Figure 5.6: Analysis of compression on 1×1 layers. (a) Reconstruction error of a decompressed 256(output) $\times$ 1024(input) $\times$ 1 $\times$ 1 weight tensor as a function of a total number of DCT coefficients used. The (solid) lines represent reordering according to Euclidean distance and (dashed) according to Cosine distance. (b) ResNet-50 top-1 accuracy on ImageNet validation set with compressed (dotted) and fine-tuned (solid) 1×1 layers. The grey line denotes the model before any compression. (Exponential y-scale.) We observe that lower values of g consistently produce better results.

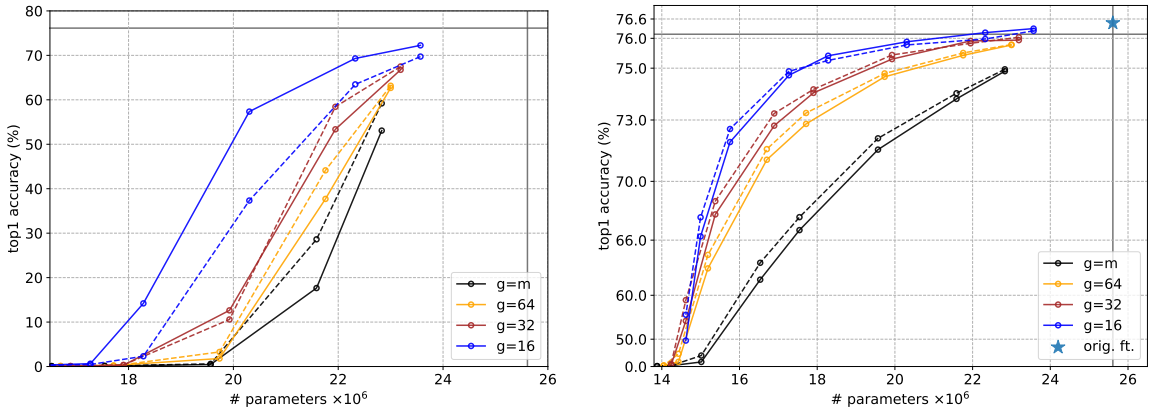
5.4.2 Overall accuracy

The ordering operation employed in our compression requires the order vector **s** to be stored. Since small values of g require storing very long order vectors (and thus effectively reducing the compression ratio), we study only $g \geq 4$. We investigate several compression rates r: $1.3\times, 1.5\times, 2\times, 3\times, 4\times, 8\times, 16\times$ and $32\times$ to uniformly reduce the number of coefficients in all layers in ResNet-50 with 1×1 kernels, using Euclidean distance to drive the reordering. The quality of compression is measured in terms of classification accuracy. It should be noted that these compression rates r do not take into account the storage associated with the index vector **s**, but results in tables and figures are reporting the number of parameters including **s**, unless stated otherwise. Figure 5.6b shows how the classification accuracy evolves when the proposed compression rate increases; the horizontal axis reports the total number of parameters in the model. This figure further confirms the reconstruction error analysis observation: smaller values of g produce more accurate compressed models. The model without weight reordering loses all of its performance even after a mild compression, while model that uses half of the DCT coefficients with $g = 4$ is only 1% short of the original model performance.

We establish that a minor amount of fine-tuning enables us to approach the original level of accuracy (achieved by the non-compressed ResNet-50) even after substantial weight compression in the DCT domain. The fine-tuning is performed for one epoch only on central crops of the resized training images without using any augmentation. The learning rate is set to $0.001\times$batch_size$/256$. As can be seen in Figure 5.6b, one epoch is sufficient even for the $32\times$ DCT compression with $g = 4$ to recover performance just 1% below that of the original, while the model without any reordering similarly compressed drops the accuracy by more than 17%. Similarly to reconstruction error results (Figure 5.6a), the Cosine distance is more suitable for the large values of $g > 16$, see Figure 5.7. Since the Euclidean distance provides better results for lower values of g and is cheaper to calculate, we use it in all the following experiments.

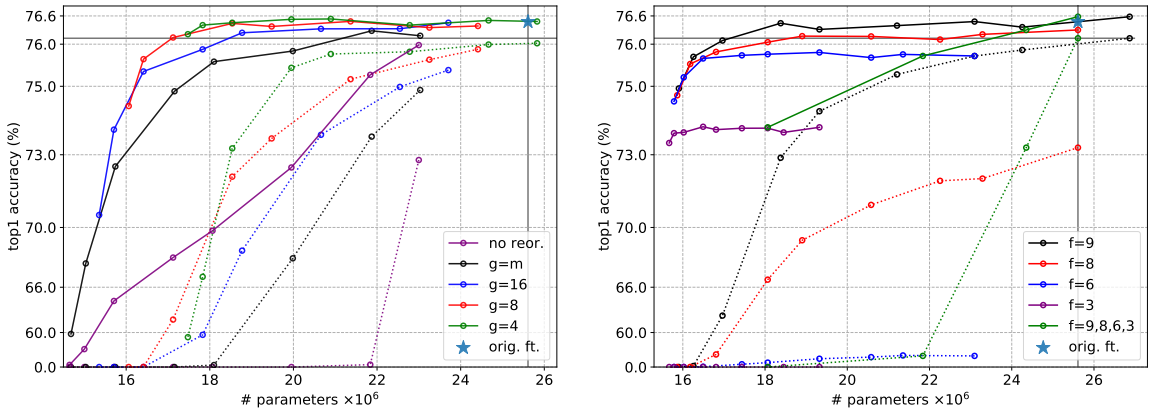
5.5 Compressing layers with spatial kernels

We now employ the same compression procedure to layers with spatial extent, $k > 1$, by reshaping the 4-dimensional weight tensor into a 2-dimensional matrix. We complement the experiment in Section 5.4 by compressing only all the layers with spatial kernels $k = 3$. Fine-tuning can recover original ResNet-50 performance without any loss even when coefficients in these layers are reduced $32\times$ if g=4 or $8\times$ when g=8, see Figure 5.8a. Compressing these layers provides a notably better accuracy-size trade-off. This effect can be explained by the fact that filters in these layers are expected to be more correlated than 1×1 layers to start with. It is worth mentioning the number of spatial layers undergoing compression in ResNet-50 is lower compared to layers with $k = 1$.



(a) Compressed (b) Compressed + fine-tuned

Figure 5.7: Comparison of Cosine (dashed) and Euclidean (solid) distance for compression of 1×1 layers. The curves show ResNet-50 top-1 accuracy on ImageNet validation set where 1×1 layers are (a) compressed, (b) compressed and fine-tuned (Exponential y-scale). The Cosine distance works better for $g > 16$.



(a) Channel dimension (b) Spatial dimension

Figure 5.8: We report ResNet-50 top-1 performance of compressed (dotted) and fine-tuned (solid) models on ImageNet validation set. The compression is applied along (a) channel and (b) spatial dimension for various levels of g. For spatial compression $g = f$. The green curve in (b) only compresses filters separately via 2D-DCT. (Exponential y-scale.)

5.5.1 Selecting dimension for computing DCT

The compression scheme depicted in Alg. 3 executes DCT along the second dimension that initially corresponds to the number of input channels. It is also meaningful to investigate applying compression to the dimension corresponding to output channels. Tensor reshaping produces a matrix with internal ordering dependant on the original position of dimensions in the weight tensor. By denoting transposition as T and reshape operations as R, we consider the following reshaping procedures:

(i) R: $\mathcal{R}_{m \times n \times k \times k \to g \times mnk^2/g} \mathbf{W}$,

(ii) TR: $\mathcal{R}_{n \times m \times k \times k \to g \times mnk^2/g} \mathbf{W}^{T_{0,1}}$,

(iii) RT: $\left(\mathcal{R}_{m\times n\times k\times k\to mnk^2/g\times g}\mathbf{w}\right)^T$

(iv) TRT: $\left(\mathcal{R}_{n\times m\times k\times k\to mnk^2/g\times g}\mathbf{w}^{T_{0,1}}\right)^T$.

The target shape always remains the same $\mathbf{w}' \in \mathbb{R}^{g\times mnk^2/g}$. TR, RT and TRT procedures require slight adjustments to the reconstruction procedure. The experimental analysis demonstrates (Table 5.1) that the use of R procedure often delivers best performance for both 1×1 and 3×3 layers with $g=4$ and $g=8$.

g	reshape type	1×1				3×3			
		1.3×	2×	4×	8×	1.3×	2×	4×	8×
no reorder	R	**0.4**	0.1	0.1	0.1	**72.8**	1.0	0.3	0.2
	TR	0.1	0.1	0.1	0.1	72.5	**1.1**	0.4	**0.3**
	RT	-	-	-	-	46.6	0.4	0.2	0.2
	TRT	-	-	-	-	38.9	0.8	**0.5**	**0.3**
m | n	R	**53.1**	**0.4**	0.1	0.1	**74.9**	**68.2**	5.3	0.3
	TR	39.0	0.3	0.1	0.1	74.2	66.9	**8.2**	**0.5**
	RT	-	-	-	-	70.4	34.7	0.6	0.3
	TRT	-	-	-	-	71.8	53.0	0.9	0.3
16	R	**72.3**	**57.4**	0.6	0.1	75.4	73.6	59.5	3.2
	TR	69.3	35.0	0.5	0.1	**75.6**	**74.2**	**64.6**	**19.8**
	RT	70.9	40.6	**0.9**	**0.2**	75.1	72.6	57.6	6.6
	TRT	71.5	46.3	0.8	0.1	75.5	73.6	53.7	9.5
8	R	74.4	**71.3**	**45.9**	**0.7**	**75.9**	**75.2**	**72.2**	**62.3**
	TR	72.1	58.7	16.4	0.2	75.8	75.0	71.9	58.9
	RT	74.2	69.9	40.1	0.4	**75.9**	75.1	71.9	52.4
	TRT	**74.7**	65.9	35.7	1.5	**75.9**	75.1	70.8	59.0
4	R	**75.7**	**75.1**	**71.4**	**59.3**	**76.0**	**75.8**	**75.5**	73.2
	TR	75.3	73.4	67.9	55.7	**76.0**	**75.8**	75.3	**74.3**
	RT	75.3	72.8	64.8	33.2	**76.0**	**75.8**	75.4	72.9
	TRT	75.2	74.0	71.0	48.5	**76.0**	75.7	74.9	73.1

Table 5.1: Comparison of the four reshaping strategies in terms of ResNet-50 accuracy on ImageNet validation set post-compression (without fine-tuning) of all 1×1 or all 3×3 layers. Results depict several compression rates r and grouping factors g. In case of 1×1 layers $g=m$ is used for R and $g=n$ for TR, while for all 3×3 layers in ResNet-50 it holds that $m=n$.

5.5.2 Combining spatial and channel compression

A common approach to compress convolutional filters with spatial extent is to decompose each filter separately into 2-dimensional basis functions such as 2D DCT [109]. This approach however ignores the channel dimension, hence here we extend it to see whether the representation of weight tensors via separable 3-dimensional DCT bases can improve compression-accuracy trade-off. The compression consists of 2 stages. Firstly, each $k\times k$ filter is expressed with $f\leq k^2$ 2D DCT basis functions with the lowest frequency. Secondly,

the tensor is reshaped by RT procedure with $g = f$ that ensures the channel and spatial dimensions are transformed individually, prior to pruning.

We compress uniformly every layer of ResNet-50 with $k = 3$ to retain $f \in \{9, 8, 6, 3\}$ spatial frequencies: $f = 9$ corresponds to full data with no compression of 3×3 filters (only casting to DCT domain), and $f = 3$ to the strongest degree of spectral pruning (by retaining 3 DCT coefficients out of 9). We then employ the same compression rates for channel-dimension as in previous experiments. Figure 5.8b demonstrates the achieved performances and the corresponding model sizes with and without channel dimension compression. The uniform spatial compression gives poor results compared to the channel dimension compression proposed in Alg. 3. The combination of the two approaches improves these results but is still inferior to relying solely on Alg. 3.

5.6 Full network compression

Finally we proceed with compressing all the convolutional layers at once. Compressing one layer at a time gives us an insight that compressing shallower layers increases classification error more than compressing deeper layers by the same ratio (cf. Figure 5.9). These 'hard-to-compress' layers have also substantially fewer parameters and compressing them with high ratios brings almost no change in model size but leads to a substantial drop in network performance. It is expected that after sorting, larger weight matrices produce less sparse sequence with regular structure that can be approximated with proportionally fewer basis functions. We tackle the performance/size trade-off by deriving compression parameters from the size of the layer. We omit the first layer that has only under 10k parameters and treat the output layer as a 1×1 convolutional layer. We consider 2 strategies: "progressive-r" that adjusts compression ratio r (fixed g throughout the network), and "progressive-g" that optimizes g (fixed r). For the first strategy, a user provided hyperparameter r' is used as a ratio increase $r_{l_1} = 1 + r'$ for a reference layer l_1 that we chose to be the smallest compressed layer. Let $p_{l_i} = m_{l_i} n_{l_i} k_{l_i}^2$ be the number of parameters of the weight tensor of layer l_i. The compression ratio r_{l_i} for any layer l_i is then

$$r_{l_i} = 1 + r' \frac{\sqrt{p_{l_i}}}{\sqrt{p_{l_1}}}. \tag{5.7}$$

The proportion of square roots of parameter counts scales linearly with the number of channels. The second strategy determines g_{l_i} as the closest exponent of 2 that is lower or equal to the weight size proportion

$$g_{l_i} = \max\left(2, 2^{\left\lfloor \log_2 \frac{\sqrt{p_{l_i}}}{\sqrt{p_{l_1}}} \right\rfloor}\right). \tag{5.8}$$

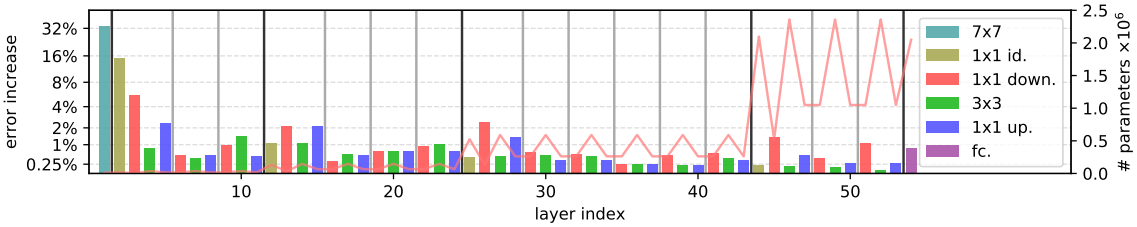


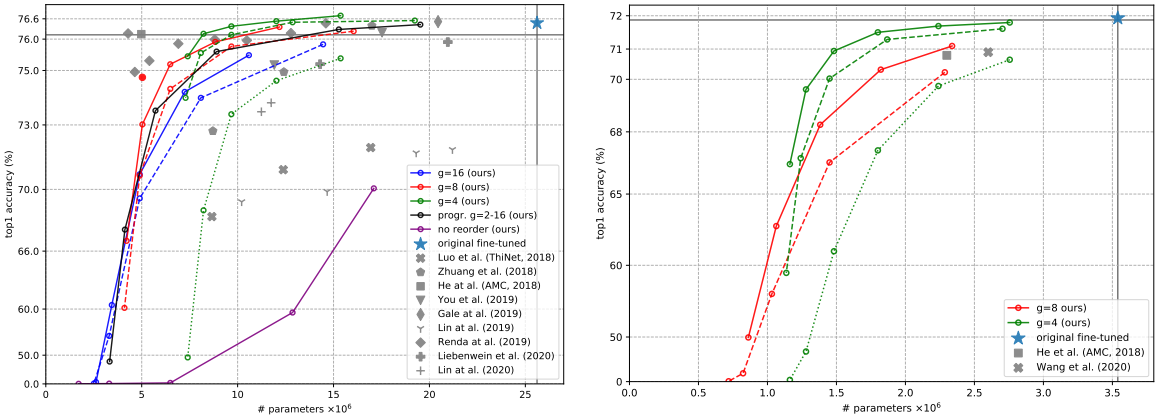
Figure 5.9: Top-1 classification error increase induced by compressing a particular layer of ResNet-50 with $g = 16$, $r = 4\times$. Vertical grey lines indicate borders of residual blocks, black ones show where feature resolution is decreased. Red curve illustrates parametric complexity of a particular layer. Deeper layers with more parameters exhibit lower errors. (Logarithmic error-scale.)

	ResNet-50 [63, 137]							MobileNet-V2 [61, 137]					
metric	Orig.	$g=4$			$g=8$			Orig.	$g=4$				
		$r'=1/8$	$r'=1/4$	$r'=1$	$r'=1/2$	$r'=1$	$r'=1$*		$r'=1/4$	$r'=1/2$	$r'=1$	$r'=2$	$r'=4$
#params	25.6M	15.4M	12.0M	8.2M	6.5M	5.0M	5.0M	3.5M	2.8M	2.2M	1.8M	1.5M	1.3M
#trainable	25.6M	8.9M	5.5M	1.7M	3.2M	1.7M	1.7M	3.5M	1.7M	1.2M	0.8M	0.4M	0.2M
top1 % ↑	76.15	76.69	76.53	76.16	75.21	73.02	74.77	71.87	71.80	71.70	71.51	70.94	69.64
top5 % ↑	92.87	93.27	93.20	92.96	92.36	91.37	92.27	90.29	90.47	90.38	90.34	89.89	89.18

Table 5.2: Our progressive-r compressed ResNet-50 and MobileNet-V2 results on ImageNet after one epoch of fine-tuning post-compression (*model is fine-tuned 20 epochs).

5.6.1 ResNet-50 architecture

We have tested the compression strategy on ResNet-50 model with $g \in \{4, 8, 16\}$ and $r \in \{2, 4, 8, 16, 32\}$ for uniform compression (the same r in every layer) as well as for progressive-g compression and hyperparameter $r' \in \{0.125, 0.25, 0.5, 1, 2\}$ for progressive-r compression, see Figure 5.10. The performance of the progressive-r compression exceeds that of the uniform for all values of g. One epoch fine-tuning of a model uniformly compressed with $g = 4$, $r = 8\times$ can recover the original model performance and consists of 38% of the original model's size. Comparable accuracy is obtained with progressive-r strategy ($r' = 1$) with a model that has only 32% the original amount of parameters. The drawback of this strategy is the large vector of indices used to reorder the weights that sets a lower bound to the maximal possible compression. Progressive-g strategy addresses this issue by shrinking this vector for large layers and thus allows for higher compression rates without excessively corrupting the early layers. Table 5.2 summarises the best numerical results obtained with our compression strategy. If we extend the fine-tuning time to 20 epochs (decreasing learning rate by 10 at epoch 15), a severely pruned model can improve its performance by at least another percentage point, see Table 5.2. Comparisons with several state-of-the-art techniques, presented in Figure 5.10, demonstrate high efficiency of the proposed spectral domain pruning approach. Note that the models outperforming the proposed approach for stronger levels of compression [7, 164], see Figure 5.10, involve very substantial amount of fine-tuning of 50-300 epochs.



(a) ResNet-50 compression on Imagenet (b) MobileNet-V2 compression on Imagenet

Figure 5.10: Top-1 accuracy vs. parameter count on ImageNet validation set after one epoch of fine-tuning for (a) ResNet-50 and (b) MobileNet-V2 architectures: progressive-r compression strategy (solid) clearly outperforms uniform (dashed). The model without fine-tuning (dotted) outperforms some of the related works [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. The red dot in (a) shows performance after 20 epoch fine-tuning. Grey vertical and horizontal bars correspond to the original model size and accuracy. (Exponential y-scale.)

Comparison with magnitude pruning. We now compare the proposed spectral domain pruning to the standard magnitude-based pruning using $\ell 1$-norm. Alg. 3 can be simply adjusted to perform pruning of tensors in standard bases, bypassing the DCT transform. The elements of size g are sorted according to their ℓ_1 norm, keeping only a relevant fraction of those with the largest norm. Table 5.3 clearly demonstrates the superiority of DCT domain-compression for preserving network performance. Minor fine-tuning of one epoch is only 2% worse than original model at 32× DCT compression of all 1×1 and 3×3 layers, while the model pruned based on $\ell 1$ norm cannot recover any of the original performance.

reorder	compress					compress & fine-tune				
	2×	4×	8×	16×	32×	2×	4×	8×	16×	32×
ℓ_1 reorder	16.2	0.1	0.1	0.1	0.1	74.4	68.2	54.1	22.5	7.0
Alg. 2 reorder + DCT	74.8	70.4	49.5	14.4	0.2	76.6	76.5	76.1	75.6	74.0

Table 5.3: Top-1 classification accuracy on ImageNet validation set of ResNet-50 compressed with $g = 4$. Combination of reordering and DCT transform improves over magnitude-based pruning (ℓ_1).

5.6.2 MobileNet-V2 architecture

We now investigate the capability of our method to compress models with compact design. MobileNet-V2 model [61] sparsifies the weights by using depth-separable convolutions. A vast portion of its weights thus resides in its resampling layers (1×1 convolutions) and the output layer. We focus only on these layers, leaving the first layer and the following 7 blocks intact. The fine-tuning is done with batch size 64 and learning-rate 0.0001. The progressive

compression approach enabled us to compress the network into 42% of its original size within 1% accuracy drop (see Figure 5.10b). Note that up until 50% parameter compression the top-5 accuracy remains above the original level (Table 5.2).

5.7 Post-compression batch normalization correction

In some situations data might not be available to fine-tune the compressed model, or fine-tuning might be prohibitively expensive. In such scenarios, a simple scale correction mechanism can come in handy. Replacing the tensor $\mathbf{w}$ by its approximation $\tilde{\mathbf{w}}$ alters output of the convolution (5.1) proportionally to the approximation error magnitude. These differences can however be partially diminished by rescaling and shifting the approximated weight tensor to correct its mean and magnitude. For this purpose we reuse parameters that carry out normalization of activations during the training. The batch normalization operation that follows every convolutional layer in many modern architectures is formalized as

$$\tilde{x}_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{5.9}$$

with statistical parameters mean $\mu = \sum_{i=0}^{n-1} x_i/n$ and variance $\sigma^2 = \sum_{i=0}^{n-1} (x_i - \mu)^2/n$ estimated during training over a batch with n samples. The scale γ and bias β parameters are updated via back-propagation. For deterministic results during the inference stage the running statistical estimates of μ and σ^2 are used instead. Serving only as an extra affine function, these parameters can be incorporated into weight and bias parameters of the preceding layer. The weight tensor $\mathbf{w}$ and bias b can be transformed to $\mathbf{w}'$ and b' that incorporate the estimated scaling of batch normalization layer as follows:

$$\mathbf{w}' = \frac{\gamma \mathbf{w}}{\sqrt{\sigma^2 + \epsilon}}, \qquad b' = b + \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \epsilon}}. \tag{5.10}$$

Similarly, to implement the scale correction, we replace γ by a new scale $\tilde{\gamma} = \rho\gamma$ controlled by parameter ρ. Value of ρ is selected such that norm of the merged convolutional and batch normalization layer $\mathbf{w}'$ equals the norm of the merged reconstructed layer $\tilde{\mathbf{w}}'$, i.e. $\|\mathbf{w}'\|_2 = \|\tilde{\mathbf{w}}'\|_2$. Using equation (5.10) and the definition of $\tilde{\gamma}$ we establish that

$$\frac{\gamma \|\mathbf{w}\|_2}{\sqrt{\sigma^2 + \epsilon}} = \frac{\rho\gamma \|\tilde{\mathbf{w}}\|_2}{\sqrt{\sigma^2 + \epsilon}} \qquad \Longrightarrow \qquad \rho = \frac{\|\mathbf{w}\|_2}{\|\tilde{\mathbf{w}}\|_2}. \tag{5.11}$$

Introduction of the new scale entails a change in the bias parameter. Specifically, when the bias correction term δ is introduced, the new bias is expressed as $\tilde{\beta} = \beta + \delta$. Comparing bias of the original and approximated layer provides the following relationship:

$$\beta - \gamma \frac{\mu}{\sqrt{\sigma^2 + \epsilon}} = \beta + \delta - \rho\gamma \frac{\mu}{\sqrt{\sigma^2 + \epsilon}} \qquad \Longrightarrow \qquad \delta = \gamma\mu \frac{\rho - 1}{\sqrt{\sigma^2 + \epsilon}}. \tag{5.12}$$

Weight approximation might also shift the weight tensor mean $\mu_w = \sum_{i=0}^{n-1}\sum_{u=0}^{k-1}\sum_{v=0}^{k-1}\mathbf{w}_{i,u,v}$ that as the result shifts the activations of the layer's output. Comparing bias of the original and approximated layer when propagating the difference between μ_w and the mean of the reconstructed weight tensor $\mu_{\tilde{w}}$ establishes a relationship:

$$\gamma\frac{\mu_w-\mu}{\sqrt{\sigma^2+\epsilon}}+\beta=\rho\gamma\frac{\mu_{\tilde{w}}-\mu}{\sqrt{\sigma^2+\epsilon}}+\beta+\delta \qquad \Longrightarrow \qquad \delta=\gamma\frac{\mu_w-\mu-\rho\left(\mu_{\tilde{w}}-\mu\right)}{\sqrt{\sigma^2+\epsilon}}, \quad (5.13)$$

that gives us an estimate (5.13) for δ that combines both weight and bias correction.

Results. We have tested this approach on several compressed versions of ResNet-50 using the progressive strategy. The results in Table 5.4 show that this cost-free method improves the severely compressed models. However, this approach can be applied only to layers followed by Batch Normalization.

scale fix	$g=8$				$g=4$				
	$r'=1/8$	$r'=1/4$	$r'=1/2$	$r'=1$	$r'=1/8$	$r'=1/4$	$r'=1/2$	$r'=1$	$r'=2$
# parameters	12.2M	8.8M	6.5M	5.0M	15.4M	12.0M	9.7M	8.2M	7.4M
no correction	72.28	66.22	35.77	0.84	75.40	74.65	73.42	68.81	49.16
correction Eq. (5.12)	73.08	69.19	54.05	14.78	74.85	73.99	72.60	68.53	51.84
correction Eq. (5.13)	73.19	69.39	54.84	15.68	74.93	74.16	72.86	68.92	52.52

Table 5.4: Effects of Batch Normalization parameter correction on ImageNet top-1 classification accuracy for ResNet-50 models compressed using progressive-r strategy without fine-tuning.

5.8 Conclusion

We have proposed a novel DCT-based compression approach for effectively reducing the CNN parameter footprint. Considering the very limited fine-tuning that we used in this work, our method demonstrates state-of-the-art performance as applied to ResNet-50 [63] and MobileNet-V2 [61] architectures for the task of image classification on ImageNet (Figure 5.10). Our model also has the capability to produce strongly performing compressed networks with zero-retraining. In the future we envisage investigating how to reduce the number of computations in addition to limiting the number of parameters based on spectral representations.

Chapter 6

Conclusion

Convolutional neural networks have become a core part of many solutions used to address numerous computer vision problems. The trends are pushing their deployment into many challenging areas such as medical image recognition or autonomous driving. Such domains need reliable, transparent and fast models. CNNs are however often large, computationally intensive and not interpretable. In this thesis we have tackled some of these issues by using knowledge integrated in image compression techniques.

6.1 Summary of contributions

In Chapter 3 we have explored applicability of image representation consisting of blocks created by the Discrete Cosine Transform (DCT) for CNN training. Data in this form is represented via set of coefficients corresponding to magnitudes of cosine functions with different frequencies. We have adjusted filters of the input layer to process the non-overlapping blocks and thus learn the weights in the DCT domain. We have found that a single layer model can this way learn more discriminative features than from spatial image data. The deeper models trained on block-DCT data representation retain comparable performances. We have also managed to train a CNN model, with the necessary adjustments, on DCT coefficients calculated by JPEG compression algorithm. Our pipeline did not need to fully decompress images, only the decoded coefficient were necessary. It can be useful for fast inference on large images or as a way for improving accuracy of shallow CNN models. The drawback of this approach is the fixed size of DCT blocks which drastically limits the size of feature maps extracted from small resolution images. The blocks are also non-overlapping, hence some features at block boundaries may be overlooked.

We address both of these issues in Chapter 4 by designing "harmonic blocks". We propose to directly integrate DCT in the CNN computational graph, to factorize the convolutional filters. By doing so it can be used with blocks of arbitrary size and stride. Flexible striding allows convolution with the DCT filters without decimating the signals. In some cases the

optimization could be eased by normalizing the DCT spectrum with batch normalization. Normalizing DCT coefficients can however be harmful for image restoration tasks that require only a subtle change to the input distribution. With filter factorization into DCT bases we have improved classification accuracy on CIFAR10/100 and ImageNet datasets. By transferring the knowledge from classification task we could construct object detectors and segmentation networks with harmonic backbones that improve detection metrics on MS COCO and Pascal VOC benchmarks. We have shown the harmonic network to be also suitable for boundary estimation tasks by improving over the standard CNN baseline.

Properties of natural signals gave us an insight how DCT representation can be used to enforce certain properties on CNNs. By removing zero-frequency component from the representation we have made CNNs insensitive to illumination changes and surpassed standard CNN models when validating on unseen lighting conditions. CNN filters trained on natural images have been observed to be smooth. By suppressing high-frequency components we have elegantly compressed the number of parameters in convolutional networks without sacrificing performance. We have also shown that models compressed this way can be more robust to certain types of high-frequency noise. The compression makes harmonic networks less over-fitted to data, which also helps the generalization when only limited amount of samples is available for training. Unlike most of other works that make use of filter decomposition we use a complete basis set and can decompose existing CNNs into DCT bases that allow exact reconstruction. This facilitates building harmonic networks for fine-tuning or compression of existing models without a need for pre-training. Encouraging results on image data motivates further research with extrapolating harmonic networks to 1D convolutional networks used on audio inputs or to 3 dimensions when processing voxel representations or spatio-temporal data.

The compression via spectrum truncation in harmonic blocks can only be applied to individual filters, and is limited solely to layers that have filters with spatial extent. In many modern CNN architectures a majority of weights are however concentrated in layers with 1×1 filters and thus cannot be compressed via harmonic block compression. Therefore we have devised another approach to compress any weight tensor used in CNNs. Each weight tensor undergoing compression is firstly reshaped into two dimensional matrix. Columns of this matrix are considered to be separate vectors that are ordered based on their distances, starting from vector with the largest magnitude. This step is crucial as it compacts the energy of each row into a few coefficients as the 1D DCT transform is applied. Following truncation of a large portion of high-frequency coefficients has almost no effect on the total energy. The decompression consists of performing the inverse DCT, inverting the weight matrix reordering, and reshaping the matrix to the original shape. Furthermore, we have proposed to fix scale and shift parameters of batch normalization layers that follow compressed layers based on the compression artefacts.

We have successfully applied the proposed compression to layers of ResNet-50 and MobileNet-

V3 and obtained well performing models even without fine-tuning the weights. Fine-tuning of the reduced set of weights can lead to models with only a small portion of trainable parameters and three times smaller total size with respect to the original model, yet maintaining its performance. Success of this approach can be boosted by minimizing length of the column-vectors. For reconstruction of the weights, however, exact ordering of the vectors in the original tensor has to be stored. Reshaping the matrix such that columns are shorter will increase length of the rows and thus of the index vector necessary to reverse the ordering. In future work this issue can be solved by iterative compression of the weight matrix by shrinking column length at every iteration. This compression scheme can also be applied to other models apart from 2D CNNs, for instance convolutional networks of other dimensions or in recurrent networks, which are more sensitive to compression as they are used repeatedly during the inference.

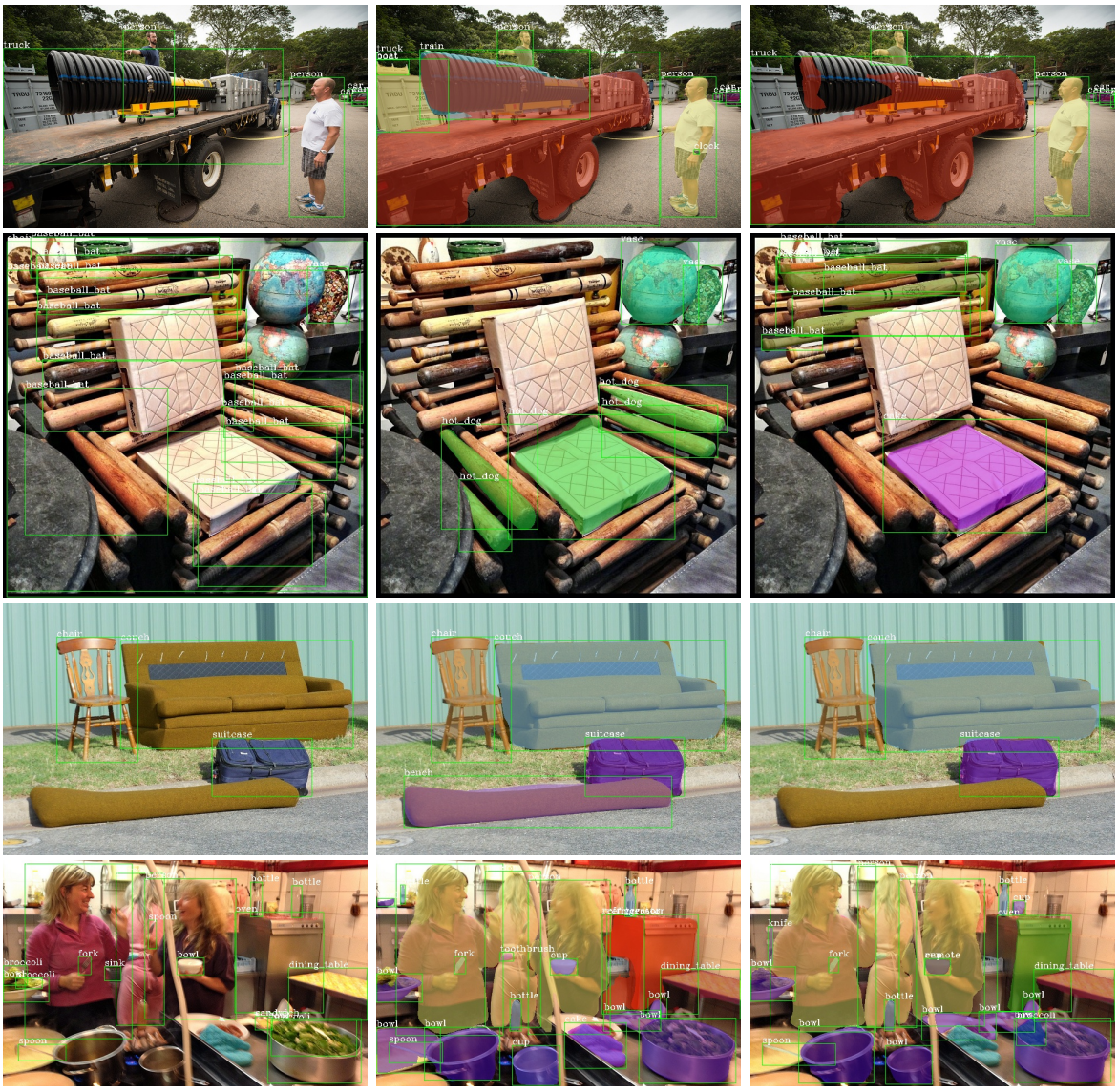
6.2 Insights and future work

This thesis has been challenging the paradigm of representing convolutional filters via standard bases. Instead, we have proposed filter factorization with the DCT basis set. Our experimental evaluation has shown that use of pre-defined filters in CNNs can lead to efficient and versatile models. We suggest to use the filter decomposition in the early layers, together with spectrum normalization for classification-related tasks, in order to improve models' performance. Deeper layers can be almost always efficiently compressed through spectrum truncation. We have mostly used the decomposition to either improve the performance or the model size. There are many avenues how filter decomposition into pre-defined bases can be exploited in order to encode more knowledge into CNNs. Using factorized filters can have interesting effect on generative learning where layers' level of detail can be hardcoded by bandpassing the filter bank. Our approach opens more possibilities to combine well known computer vision techniques with learnable algorithms. Certain layers can be regularized to encode only a particular information. Fixed bases can be used to model equivariance to certain geometrical transformations. CNNs with fixed bases can also lead to development of models more robust to various input deformations. Lastly, filter decomposition is a way to build up their comprehension and thus enhance interpretability of the whole model.

Appendix A

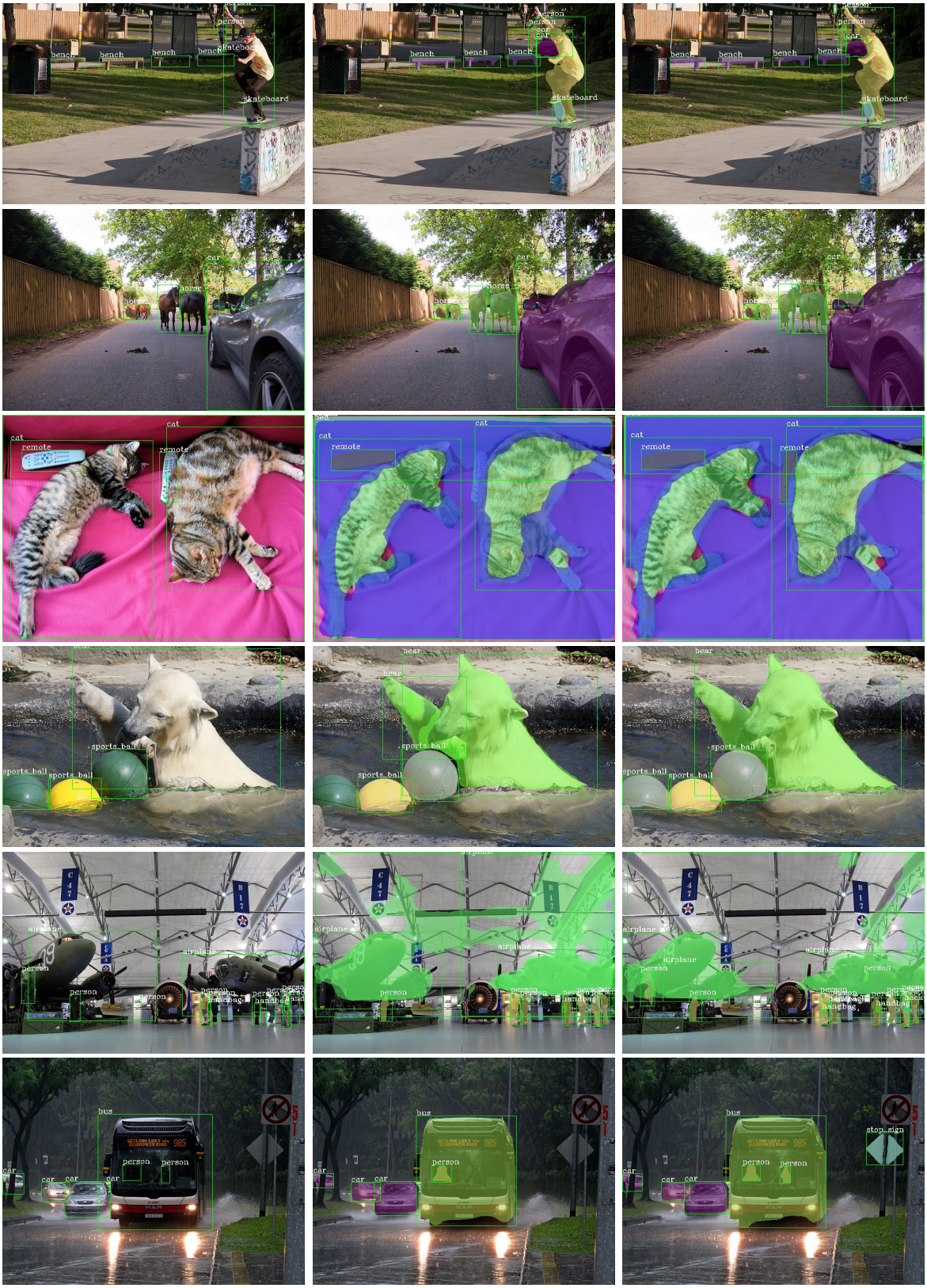
Detection results

In this Appendix chapter we provide qualitative results for the object detection and instance segmentation task presented in Section 4.5.3. We focus on segmentation results of the state-of-the-art object detection model, the hybrid task cascade (HTC) R-CNN with ResNet-101 backbone network. On Figure A.1 we display several images with false-positive detections given by the baseline HTC model. Several objects, or part of objects are classified to incorrect classes. For some of these objects the HTC with harmonic backbone has either made the correct prediction or has ignored them. The Figure A.2 in turn shows images with false negative detections. A few of the instances the HTC has missed are correctly detected by the harmonic HTC.



Ground truth HTC Harm-HTC

Figure A.1: Instance segmentation on MS COCO validation images. The figure shows ground truth bounding boxes and predictions of the hybrid task cascade (HTC) R-CNN model based on standard and harmonic ResNet-101 backbone. Images depict some of the false positives of HTC that are correctly classified by the model with harmonic backbone.



Ground truth HTC Harm-HTC

Figure A.2: Instance segmentation on MS COCO validation images, showing ground truth bounding boxes and predictions of the HTC model based on standard and harmonic ResNet-101. Depicted some of the false negatives of HTC that are detected by the harmonic HTC.

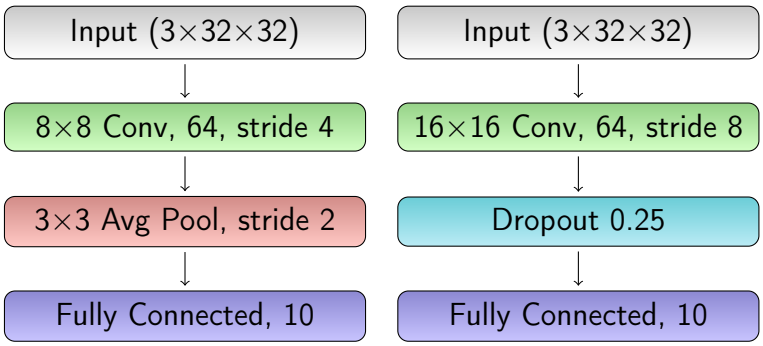
Appendix B

Model list

In this chapter we provide a list of neural network models used in this thesis in alphabetical order. Each model description contains reference to a diagram of its architecture or a reference where the model has been first introduced.

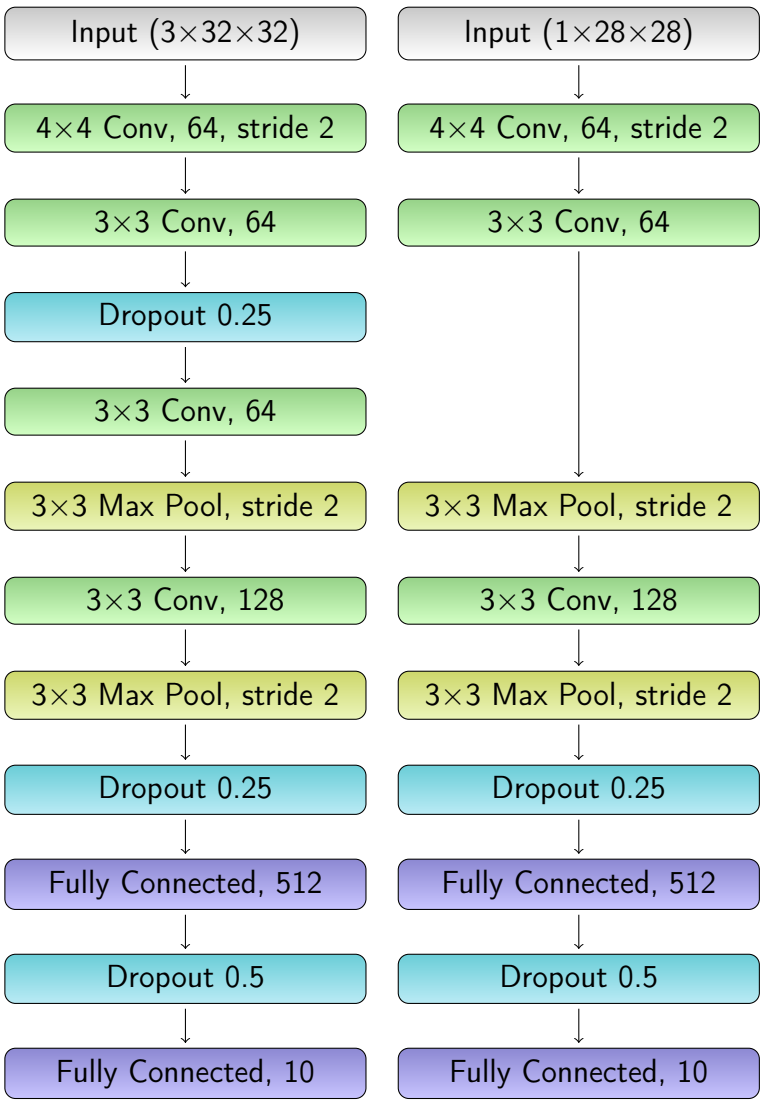
AlexNet	First CNN trained on ImageNet, Krizhevsky et al. [12]
CNN2	A 2 convolutional layer CNN designed for small NORB dataset, for details see diagram on Figure B.3a
CNN3	A 3 convolutional layer CNN designed for small NORB dataset, for details see diagram on Figure B.4a
CNN-A	A small CNN designed for 4×4 DCT-block input from CIFAR dataset, for details see diagram on Figure B.1a
CNN-B	A small CNN designed for 8×8 DCT-block input from CIFAR dataset, for details see diagram on Figure B.1b
CNN-C	A CNN designed for 2×2 DCT-block input from CIFAR dataset, for details see diagram on Figure B.2a
CNN-D	A CNN designed for 2×2 DCT-block input from MNIST dataset, for details see diagram on Figure B.2b
Cascade R-CNN	Region-based CNN with cascaded head for object detection, Cat and Vasconcelos [156]
DeepLab	Deep CNN for semantic segmentation with atrous convolution, Chen et al. [159]
Faster R-CNN	Region-based CNN for object detection, Ren et al. [13]
Harm-Net2	A 2 harmonic layer network designed for small NORB dataset, for details see diagram on Figure B.3b

Harm-Net3	A 3 harmonic layer network designed for small NORB dataset, for details see diagram on Figure B.4b
HED	Holistically-nested edge detection, Xie and Tu [15]
HTC	Hybrid task cascade for instance segmentation, Chen et al. [157]
Mask R-CNN	Region-based CNN with mask prediction head for instance segmentation, He at al. [151]
MobileNet	Efficient CNN for mobile vision applications, Sandler et al. [61]
ResNet	Residual network, He at al. [63]
ResNeXt (RNX)	Aggregated residual transformations network, Xie et al. [31]
SE-RNX	Aggregated residual transformations network with squeeze and excitation blocks, Hu et al. [140]
VGG	Very deep convolutional network, Simonyan and Zisserman [59]
WRN	Wide residual Network, Zagoruyko and Komodakis [90]



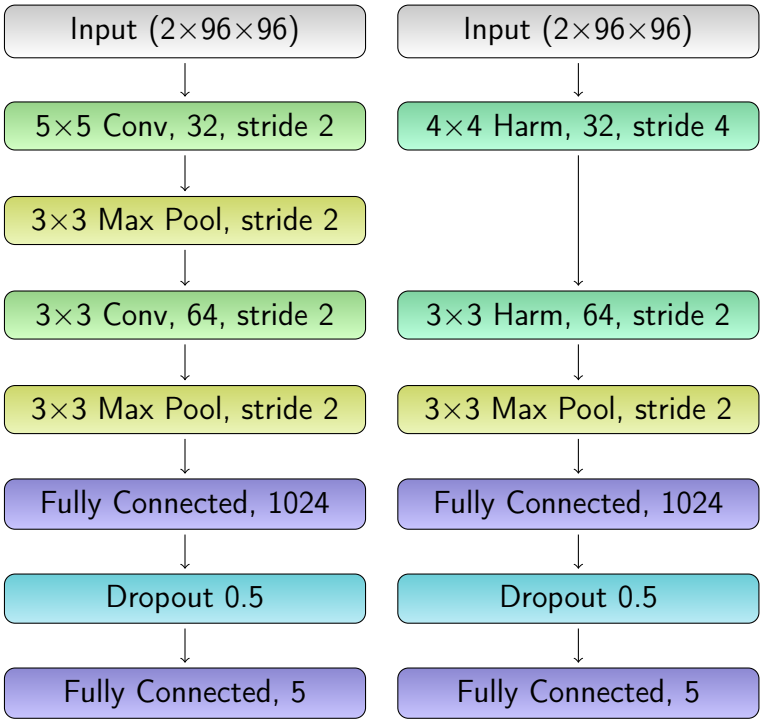
(a) CNN-A (b) CNN-B

Figure B.1: Detailed architecture diagrams of CNN-A and CNN-B. Convolution layers are followed by batch normalization and ReLU activation.



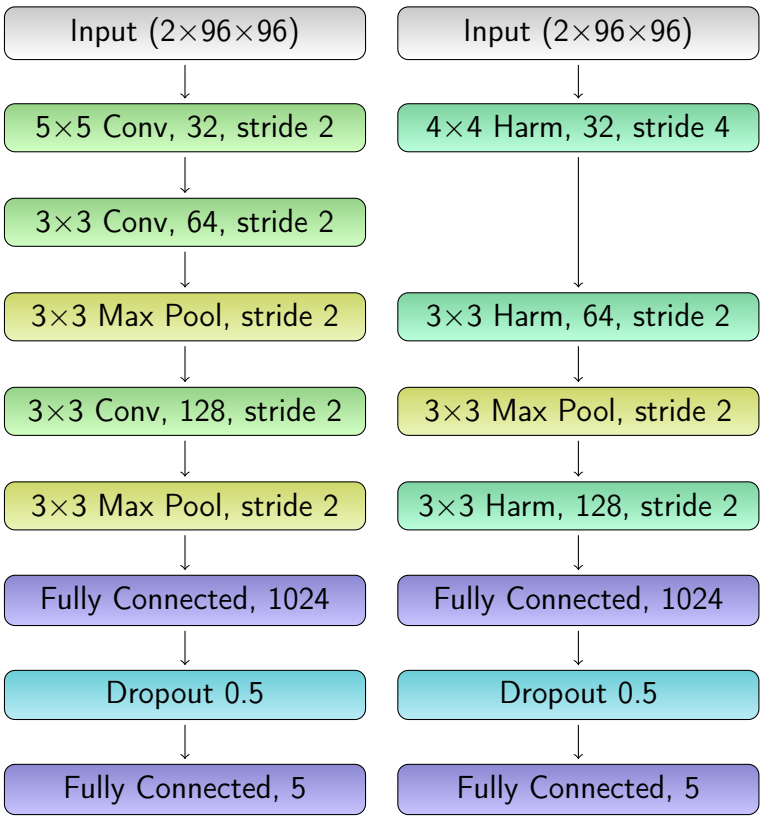
(a) CNN-C (b) CNN-D

Figure B.2: Detailed architecture diagrams of CNN-C and CNN-D. Convolution and fully connected layers (except for the output layer) are followed by batch normalization and ReLU activation.



(a) CNN2 (b) Harm-Net2

Figure B.3: Detailed architecture of CNN2 and Harm-Net2. Convolution, harmonic and fully connected layers (except for the output layer) are followed by batch normalization and ReLU.



(a) CNN3 (b) Harm-Net3

Figure B.4: Detailed architecture of CNN3 and Harm-Net3. Convolution, harmonic and fully connected layers (except for the output layer) are followed by batch normalization and ReLU.

Bibliography

[1] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proc. of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–104. IEEE, 2004.

[2] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2234–2240, 2018.

[3] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. In *Joint Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations (ODML-CDNNR) (with ICML 2019)*, 2019. arXiv:1902.09574.

[4] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, and D. Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations ICLR*, 2020.

[5] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. Thinet: pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, 41(10):2525–2538, 2018.

[6] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. *arXiv preprint arXiv:2001.08565*, 2020.

[7] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2019.

[8] A. Dubey, M. Chatterjee, and N. Ahuja. Coreset-based neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 454–470, 2018.

[9] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via

generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019.

[10] Z. Zhuang, B. Tan, M.and Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018.

[11] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. In *AAAI Conference on Artificial Intelligence*, 2020.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[14] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[15] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1395–1403, 2015.

[16] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006.

[17] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010.

[18] Beat Münch, Pavel Trtik, Federica Marone, and Marco Stampanoni. Stripe and ring artifact removal with combined wavelet—fourier filtering. *Optics express*, 17(10): 8567–8591, 2009.

[19] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2007.

[20] Daniel L Ruderman. The statistics of natural images. *Network: computation in neural systems*, 5(4):517–548, 1994.

[21] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[22] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974. doi: 10.1109/T-C.1974.223784.

[23] N. Jayant, J. Johnston, and R. Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81(10):1385–1422, 1993. doi: 10.1109/5.241504.

[24] Meng Joo Er, W. Chen, and Shiqian Wu. High-speed face recognition based on discrete cosine transform and rbf neural networks. *Trans. Neur. Netw.*, 16(3):679–691, May 2005. ISSN 1045-9227. doi: 10.1109/TNN.2005.844909. URL `http://dx.doi.org/10.1109/TNN.2005.844909`.

[25] Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. Faster neural networks straight from jpeg. In *Advances in Neural Information Processing Systems*, pages 3933–3944, 2018.

[26] Jorn-Henrik Jacobsen, Jan van Gemert, Zhongyu Lou, and Arnold WM Smeulders. Structured receptive fields in CNNs. In *Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2610–2619, 2016.

[27] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE Trans. Pat. Anal. Mach. Intel.*, 35(8):1872–1886, 2013.

[28] Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. In *Advances in neural information processing systems*, pages 2449–2457, 2015.

[29] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037, 2017.

[30] Shangzhen Luan, Chen Chen, Baochang Zhang, Jungong Han, and Jianzhuang Liu. Gabor convolutional networks. *IEEE Trans. Image Process.*, 27(9):4357–4366, 2018.

[31] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[32] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 5(1-2):115–133, 1943. doi: 10.1007/BF02478259.

[33] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958. doi: 10.1037/h0042519.

[34] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[35] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/0893-6080(89)90020-8. URL http://www.sciencedirect.com/science/article/pii/0893608089900208.

[36] Vladimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[37] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. doi: 10.1038/323533a0. URL http://dx.doi.org/10.1038/323533a0.

[39] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.

[40] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[41] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159, 2011.

[42] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[43] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[44] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[45] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[46] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

[47] Timothy Dozat. Incorporating nesterov momentum into adam. http://cs229.stanford.edu/proj2015/054_report.pdf, 2016.

[48] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, April 2011.

[49] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[51] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing (WDLASL)*, 2013. URL http://web.stanford.edu/~{}awni/papers/relu_hybrid_icml2013_final.pdf.

[52] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *4th International Conference on Learning Representations, ICLR*, 2016. URL http://arxiv.org/abs/1511.07289.

[53] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[55] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. ISSN 0340-1200. doi: 10.1007/BF00344251. URL http://dx.doi.org/10.1007/BF00344251.

[56] Yann LeCun, Bernhard Boser, John S. Denker, Don Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. URL `http://dx.doi.org/10.1162/neco.1989.1.4.541`.

[57] Kunihiko Fukushima and Nobuaki Wake. Handwritten alphanumeric character recognition by the neocognitron. *Neural Networks, IEEE Transactions on*, 2(3):355–365, May 1991. ISSN 1045-9227. doi: 10.1109/72.97912.

[58] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[60] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[61] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[62] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[64] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[65] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[66] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, Upper Saddle River, N.J., 3rd edition, 2008.

[67] Leonid P Yaroslavsky. Fast transforms in image processing: compression, restoration, and resampling. *Advances in Electrical Engineering*, 2014, 2014.

[68] RJ Clarke. Relation between the karhunen loeve and cosine transforms. In *IEE Proceedings F (Communications, Radar and Signal Processing)*, volume 128, pages 359–360. IET, 1981.

[69] Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi. The jpeg2000 still image coding system: an overview. *IEEE transactions on consumer electronics*, 46(4):1103–1127, 2000.

[70] Edouard Oyallon and Stephane Mallat. Deep roto-translation scattering for object classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[71] James DB Nelson, Alexander J Gibberd, Corina Nafornita, and Nick Kingsbury. The locally stationary dual-tree complex wavelet model. *Statistics and Computing*, 28(6): 1139–1154, 2018.

[72] Jan J Koenderink. The structure of images. *Biological cybernetics*, 50(5):363–370, 1984.

[73] Takumi Kobayashi. Analyzing filters toward efficient convnet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5619–5628, 2018.

[74] Qiang Qiu, Xiuyuan Cheng, A. Robert Calderbank, and Guillermo Sapiro. Dcfnet: Deep neural network with decomposed convolutional filters. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4195–4204. PMLR, 2018. URL `http://proceedings.mlr.press/v80/qiu18a.html`.

[75] Fergal Cotter and Nick Kingsbury. A learnable scatternet: Locally invariant convolutional layers. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 350–354. IEEE, 2019.

[76] M. Ulicny and R. Dahyot. On using CNN with DCT based image data. In *Irish Machine Vision and Image Processing Conference*, 2017.

[77] Qing Wang and Rong Zhang. Double JPEG compression forensics based on a convolutional neural network. *EURASIP J. on Information Security*, 2016(1):23, Oct 2016.

[78] I. Amerini, T. Uricchio, L. Ballan, and R. Caldelli. Localization of JPEG double compression through multi-domain convolutional neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1865–1871, July 2017.

[79] M. Barni, L. Bondi, N. Bonettini, P. Bestagini, A. Costanzo, M. Maggini, B. Tondi, and S. Tubaro. Aligned and non-aligned double JPEG detection using convolutional neural networks. *J. Vis. Comun. Image Represent.*, 49(C):153–163, November 2017.

[80] Bin Li, Haoxin Zhang, Hu Luo, and Shunquan Tan. Detecting double jpeg compression and its related anti-forensic operations with cnn. *Multimedia Tools and Applications*, 78(7):8577–8601, 2019.

[81] Xiaoyi Zou, Xiangmin Xu, Chunmei Qing, and Xiaofen Xing. High speed deep networks based on discrete cosine transformation. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 5921–5925. IEEE, 2014.

[82] A. Ghosh and R. Chellappa. Deep feature extraction in the DCT domain. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3536–3541, Dec 2016.

[83] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[84] B. Borhanuddin, N. Jamil, S. D. Chen, M. Z. Baharuddin, K. S. Z. Tan, and T. W. M. Ooi. Small-scale deep network for dct-based images classification. In *2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, pages 1–6, 2019.

[85] B. Deguerre, C. Chatelain, and G. Gasso. Fast object detection in compressed jpeg images. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 333–338, 2019.

[86] Shao-Yuan Lo and Hsueh-Ming Hang. Exploring semantic segmentation on the dct representation. In *Proceedings of the ACM Multimedia Asia*, pages 1–6. 2019.

[87] Joost van de Weijer et al. *Color features and local structure in images*. University of Amsterdam, 2004.

[88] B. D. Zarit, B. J. Super, and F. K. H. Quek. Comparison of five color models in skin pixel classification. In *Proceedings International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems. In Conjunction with ICCV'99 (Cat. No.PR00378)*, pages 58–63, 1999. doi: 10.1109/RATFG.1999.799224.

[89] M. C. Shin, K. I. Chang, and L. V. Tsap. Does colorspace transformation make any difference on skin detection? In *Sixth IEEE Workshop on Applications of Computer Vision, 2002. (WACV 2002). Proceedings.*, pages 275–279, 2002. doi: 10.1109/ACV.2002.1182194.

[90] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proc. of British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

[91] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[92] Yann Lecun and Corinna Cortes. The mnist database of handwritten digits, 2017. http://yann.lecun.com/exdb/mnist/.

[93] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010. URL http://www.icml2010.org/papers/432.pdf.

[94] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[95] Dan Fu and Gabriel Guimaraes. Using compression to speed up image classification in artificial neural networks. 2016.

[96] J. Deng, W. Dong, R. Socher, L. J. 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, June 2009. doi: 10.1109/CVPR.2009.5206848.

[97] E. Linzer and E. Feig. New scaled dct algorithms for fused multiply/add architectures. In *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, pages 2201–2204 vol.3, Apr 1991. doi: 10.1109/ICASSP.1991.150851.

[98] Róbert Torfason, Fabian Mentzer, Eiríkur Ágústsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Towards image understanding from deep compression without decoding. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkXWCMbRW.

[99] Steganography and steganalysis in python. http://www.ifs.schaathun.net/pysteg/. Accessed: 2018-05-11.

[100] C. L. Salazar and T. D. Tran. On resizing images in the dct domain. In *Image Processing, 2004. ICIP '04. 2004 International Conference on*, volume 4, pages 2797–2800 Vol. 4, Oct 2004. doi: 10.1109/ICIP.2004.1421685.

[101] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115 (3):211–252, 2015.

[102] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*, 2016.

[103] Pytorch examples. `https://github.com/pytorch/examples`. Accessed on 27/2/2021.

[104] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[105] Cees G. M. Snoek, Marcel Worring, and Arnold W. M. Smeulders. Early versus late fusion in semantic video analysis. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MULTIMEDIA '05, page 399–402, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930442. doi: 10.1145/1101149.1101236.

[106] Noha M. Elfiky, Fahad Shahbaz Khan, Joost van de Weijer, and Jordi Gonzàlez. Discriminative compact pyramids for object and scene recognition. *Pattern Recognition*, 45(4):1627–1636, 2012. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2011.09.020.

[107] F. S. Khan, R. M. Anwer, J. van de Weijer, A. D. Bagdanov, M. Vanrell, and A. M. Lopez. Color attributes for object detection. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3306–3313, 2012. doi: 10.1109/CVPR.2012.6248068.

[108] M. Ulicny, V. A Krylov, and R. Dahyot. Harmonic networks with limited training samples. In *European Signal Processing Conference (EUSIPCO)*, Sep. 2019. URL `https://arxiv.org/abs/1905.00135`.

[109] M. Ulicny, V. A. Krylov, and R. Dahyot. Harmonic networks for image classification. In *Proc. of British Machine Vision Conference (BMVC)*, Sep. 2019.

[110] Muhammad Tayyab and Abhijit Mahalanobis. Basisconv: A method for compressed representation and learning in cnns. *arXiv preprint arXiv:1906.04509*, 2019.

[111] Amarjot Singh and Nick Kingsbury. Dual-tree wavelet scattering network with parametric log transformation for object classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2622–2626. IEEE, 2017.

[112] E. Oyallon, S. Zagoruyko, G. Huang, N. Komodakis, S. Lacoste-Julien, M. Blaschko, and E. Belilovsky. Scattering networks for hybrid representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2208–2221, Sep. 2019. ISSN 0162-8828. doi: 10.1109/TPAMI.2018.2855738.

[113] Amarjot Singh and Nick Kingsbury. Efficient convolutional network learning using parametric log based dual-tree wavelet scatternet. In *Computer Vision Workshop (ICCVW), 2017 IEEE International Conference on*, pages 1140–1147. IEEE, 2017.

[114] Edouard Oyallon, Eugene Belilovsky, Sergey Zagoruyko, and Michal Valko. Compressing the input for CNNs with the first-order scattering transform. In *European Conference on Computer Vision (ECCV)*, 2018.

[115] T. Williams and R. Li. Advanced image classification using wavelets and convolutional neural networks. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 233–239, Dec 2016.

[116] S. Said, O. Jemai, S. Hassairi, R. Ejbali, M. Zaied, and C. Ben Amar. Deep wavelet network for image classification. In *2016 IEEE International Conference on Systems, Man, and Cybernetics*, pages 922–927, Oct 2016.

[117] D. D. N. De Silva, S Fernando, I. T. S. Piyatilake, and A. V. S Karunarathne. Wavelet based edge feature enhancement for convolutional neural networks. In *Eleventh International Conference on Machine Vision (ICMV 2018)*, volume 11041, page 110412R. International Society for Optics and Photonics, 2019.

[118] Travis Williams and Robert Li. Wavelet pooling for convolutional neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[119] Shin Fujieda, Kohei Takayama, and Toshiya Hachisuka. Wavelet convolutional neural networks for texture classification. *arXiv preprint arXiv:1707.07394*, 2017.

[120] H. Lu, H. Wang, Q. Zhang, D. Won, and S. W. Yoon. A dual-tree complex wavelet transform based convolutional neural network for human thyroid medical image segmentation. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 191–198, June 2018.

[121] Max Ehrlich and Larry S. Davis. Deep residual learning in the jpeg transform domain. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[122] Adrià Ciurana, Albert Mosella-Montoro, and Javier Ruiz Hidalgo. Hybrid cosine based convolutional neural networks. *CoRR*, abs/1904.01987, 2019. URL `http://arxiv.org/abs/1904.01987`.

[123] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*. BMVA Press, 2014.

[124] Minyoung Kim and Luca Rigazio. Deep clustered convolutional kernels. In *Feature Extraction: Modern Questions and Challenges*, pages 160–172, 2015.

[125] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing convolutional neural networks in the frequency domain. In *Proc. of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1475–1484, New York, NY, USA, 2016. ACM.

[126] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. CNNpack: Packing convolutional neural networks in the frequency domain. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 253–261. Curran Associates, Inc., 2016.

[127] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[128] H. S. Malvar. Lapped transforms for efficient transform/subband coding. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(6):969–978, June 1990. doi: $10.1109/29.56057$.

[129] P. Dutilleux. An implementation of the "algorithme à trous" to compute the wavelet transform. In Jean-Michel Combes, Alexander Grossmann, and Philippe Tchamitchian, editors, *Wavelets*, pages 298–304, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. ISBN 978-3-642-97177-8.

[130] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

[131] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proc.-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

[132] E. M. Rehn and H. Sprekeler. Nonlinear supervised locality preserving projections for visual pattern discrimination. In *2014 22nd International Conference on Pattern Recognition*, pages 1568–1573, Aug 2014.

[133] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6572`.

[134] Haohan Wang, Xindi Wu, Zeyi Huang, and Eric P Xing. High-frequency component helps explain the generalization of convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8684–8694, 2020.

[135] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[136] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[137] Torchvision models. `https://pytorch.org/docs/stable/torchvision/models.html`. Accessed on 27/2/2021.

[138] Sam Gross and Michael Wilber. Training and investigating residual nets. *Facebook AI Research*, 2016. URL `https://github.com/facebook/fb.resnet.torch`.

[139] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint:1409.1556*, 2014.

[140] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019. doi: 10.1109/TPAMI.2019.2913372.

[141] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[142] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[143] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

[144] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 718–726, 2017.

[145] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *Advances in Neural Information Processing Systems*, pages 4467–4475, 2017.

[146] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/tan19a.html`.

[147] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[148] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[149] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[150] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proc. of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[151] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proc. of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[152] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[153] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[154] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Mmdetection:

Open mmlab detection toolbox and benchmark. *CoRR*, abs/1906.07155, 2019. URL http://arxiv.org/abs/1906.07155.

[155] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proc. of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[156] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proc. of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[157] K. Chen, J. Pang, J. Wang, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Shi, W. Ouyang, C. C. Loy, and D. Lin. Hybrid task cascade for instance segmentation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4969–4978, June 2019. doi: 10.1109/CVPR.2019.00511.

[158] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 447–456, 2015.

[159] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[160] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, May 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.161. URL http://dx.doi.org/10.1109/TPAMI.2010.161.

[161] Jorn-Henrik Jacobsen, Bert De Brabandere, and Arnold Smeulders. Dynamic steerable blocks in deep residual networks. In Gabriel Brostow Tae-Kyun Kim, Stefanos Zafeiriou and Krystian Mikolajczyk, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 145.1–145.13. BMVA Press, September 2017. ISBN 1-901725-60-X. doi: 10.5244/C.31.145. URL https://dx.doi.org/10.5244/C.31.145.

[162] D. Blalock, Jose J. Gonzalez O., J. Frankle, and J. Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020*, pages 129–146. 2020.

[163] J. Frankle, Gintare K. Dziugaite, D. M. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning (ICML)*, 2020. arXiv:1912.05671.

[164] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[165] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2130–2141, 2019.

[166] Yunhe Wang, Chang Xu, Chao Xu, and Dacheng Tao. Packing convolutional neural networks in the frequency domain. *IEEE transactions on pattern analysis and machine intelligence*, 41(10):2495–2510, 2018.

[167] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. Frequency-domain dynamic pruning for convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1043–1053, 2018.