Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5873

CNNs for JPEGs: Designing Cost-Efficient Stems

Samuel Felipe dos Santos [**Federal University of São Carlos** | ***samuel.felipe@ufscar.br***]
Nicu Sebe [**University of Trento** | ***niculae.sebe@unitn.it***]
Jurandy Almeida [**Federal University of São Carlos** | ***jurandy.almeida@ufscar.br***]

Department of Computing, Federal University of São Carlos (UFSCar), Rod. João Leme dos Santos, km 110 - SP-264, Itinga, Sorocaba, SP, São Carlos, SP, 18052-780, Brazil.

Received: 12 April 2025 • **Accepted:** 14 July 2025 • **Published:** 02 March 2026

Abstract. Convolutional neural networks (CNNs) have achieved astonishing advances over the past decade, pushing the state-of-the-art in several computer vision tasks. CNNs are capable of learning robust representations of the data directly from RGB pixels. However, most image data is usually available in compressed format, of which the JPEG is the most widely used due to transmission and storage purposes. For this motive, a preliminary decoding process that has a high computational load and memory usage is demanded. Image decoding can be a performance bottleneck for devices with limited computational resources, such as embedded devices, even when hardware accelerators are used. For this reason, deep learning methods capable of learning directly from the compressed domain have been gaining attention in recent years. These methods usually extract a frequency domain representation of the image, like DCT, by a partial decoding, and then make adaptation to typical CNN architectures to work with it. In this paper, we perform an in-depth study of the computational cost of deep models designed for the frequency domain, evaluating the cost of decoding and passing images through the network. We notice that previous work increased the model's computational complexity to accommodate for the compressed images, nullifying the speed up gained by not decoding images. We propose to remove the changes to the model that increase the computational cost, replacing it with our designed lightweight stems. This way, we can take full advantage of the speed-up obtained by avoiding the decoding. Our strategies were successful in generating models that balance efficiency and effectiveness, allowing deep models to be deployed in a wider array of devices. We achieve up to 25.91% reduction in computational complexity (FLOPs), while only decreasing accuracy in up to 2.97%. We also propose the efficiency-effectiveness score S_E to highlight models with favorable trade-offs between accuracy, computational cost and number of parameters.

Keywords: JPEG, Compressed Domain, DCT, Cost Efficient Models

1 Introduction

Convolutional neural networks (CNNs) have led to significant advances in computer vision and are used in several applications, including medical imaging, autonomous driving, and road surveillance, among others [Deguerre *et al*., 2019; Li *et al*., 2019; Drews-Jr *et al*., 2021; Ferraz and Betini, 2025]. However, achieving this impressive performance requires large models with millions to billions of parameters, resulting in oversized and redundant networks [Marinó *et al*., 2023]. Simultaneously, acquisition devices are becoming capable of capturing images with increasingly higher resolutions [Zhang *et al*., 2019]. For these reasons, one of the main problems faced by deep learning models is their high computational cost, which makes inference and training very expensive [Ehrlich and Davis, 2019]. The need for high processing power and abundant memory capacity makes it difficult to apply such models to devices whose available computing resources are limited, such as mobile phones and other edge devices [Marinó *et al*., 2023]. Therefore, specialized optimizations are imperative at both the software and hardware levels to develop efficient and effective deep learning-based solutions [Marchisio *et al*., 2019]. Several methods have been proposed in the literature to accelerate CNNs. However, only a few of these works explore avoiding the cost of decoding the images by designing models that work directly with information from the compressed domain.

In most cases, compressed formats such as JPEG, PNG, and GIF are used to store image data because they make storage and transmission easier [Deguerre *et al*., 2021]. There are also end-to-end compression methods that learn a compression format for the data. But to use these strategies, data that is already available would have to be encoded in these formats. From these formats, JPEG has remained the most popular despite advances in image compression and is considered a simple solution to store and transmit visual data [Ehrlich *et al*., 2021]. For this motive, we decided to focus only on the JPEG compression standard, since most images are already available are stored in this compression format.

Typical CNNs commonly use RGB images as inputs. For this motive an additional decoding step is required for working with compressed images. The decoding step can be a performance bottleneck in devices with low computational power, like embedded systems [Wang *et al*., 2023]. It also increases the latency of the model, and may hinder real-time applications. A possible alternative to alleviate this problem is to design CNNs capable of learning with DCT (Discrete Cosine Transform) coefficients rather than RGB pixels [Deguerre *et al*., 2019; Ehrlich and Davis, 2019; Deguerre *et al*., 2021; Ehrlich *et al*., 2021; Wang *et al*., 2023; Gueguen *et al*., 2018; Lo and Hang, 2020; Xu *et al*., 2020; Ehrlich *et al*., 2020; Santos *et al*., 2019; Santos and Almeida, 2020].

The DCT is a representation of the data in the frequency domain that is obtained by partial decoding, saving computational cost and removing decoding latency. Optimizations that can potentially reduce processing delays in critical public infrastructure applications [Duan *et al*., 2023]. Frequency domain image processing can yield advantages like computational efficiency and spatial redundancy removal, being used successfully in several computer vision tasks [He *et al*., 2017; Peng *et al*., 2024; Liu *et al*., 2023; Zhang *et al*., 2023b,a; Ming *et al*., 2024]. The key idea exploited by existing works consists in adapting traditional CNN architectures to facilitate the learning with DCT coefficients rather than RGB pixels. This is usually done by removing the stem, the initial stage of the model that is responsible for reducing the spatial dimension, since the DCT already have a small spatial resolution, and making modifications to the other early stages. In this way, DCT coefficients can be extracted by partially decoding JPEG-compressed data and fed directly into the network. However, the changes in the network generally lead to a significant increase in its computational complexity.

To deal with this limitation, we propose to avoid making modifications that can increase the computational cost of the model. We replace these changes with our cost efficient stems that reduce the amount of input channels to the amount expected by the baseline model. This way, we can achieve both a faster model and the speed up of not decoding images. To achieve this goal, we conducted an in-depth study on the computational efficiency of preprocessing frequency domain data and passing it through the network.

Experiments were conducted on the ImageNet dataset. Results indicate our strategies were able to reduce the computational complexity of existing networks in up to 25.91%, making them faster than the RGB baseline, while only decreasing accuracy in up to 2.97%. In this way, it is possible to maintain the speed up by avoiding the decoding of images without increasing the computational cost of the model, leading to better trade-offs between accuracy and computational efficiency.

The main contributions of this work are: (1) Proposing the use of lightweight stems to adapt a deep model for DCT coefficients instead of making changes to the network that significantly increases the computational cost; (2) Conducting an in-depth study of the computational cost of decoding and passing the images through the model; (3) Proposing the efficiency-effectiveness score metric for evaluating the trade-off between overall computational cost and classification performance.

A preliminary version of this work was presented at the Iberoamerican Congress on Pattern Recognition (CIARP 2021) [Santos and Almeida, 2021], where we presented our approaches to reduce the amount of input channels and computational complexity. Here, we introduce several innovations. We discuss new strategies for reducing the amount of input channels that are fed to the network, adapting our proposed strategies to a different architecture. We propose a new metric for measuring the trade-off between computational efficiency and effectiveness. We also include new experiments evaluating the performance of different models regarding their network inference speed in terms of frames per second (FPS), considering the data preprocessing time and the network time. These experiments were crucial for the discussions proposed in this paper. The inference time measurements provide further insight on the relevance of the decoding step and the importance of our proposed strategies to avoid the increase in network time, showing that our approaches are considerably faster than the other compared methods, a speed up that is crucial in environments with limited computational resources. A brief overview of this work was also presented at the Workshop of Theses and Dissertations from the Conference on Graphics, Patterns and Images (SIBGRAPI 2024 WTD) [Santos *et al*., 2024]. There, our strategies and methods were mentioned, but not defined, explained, and discussed in completion.

The remainder of this paper is organized as follows. Section 2 presents a brief explanation of the JPEG compression standard. Section 3 explores related work. Section 4 describes our approach to reduce the computational cost of CNNs trained directly on compressed data. Section 5 presents the experimental setup. Section 6 reports our results. Finally, Section 7 offers our conclusions and directions for future work.

2 JPEG Compression

The JPEG standard (ISO/IEC 10918) was created in 1992 and is currently the most widely-used image coding technology for lossy compression of digital images. The basic steps of the JPEG compression algorithm are presented in Figure 1.

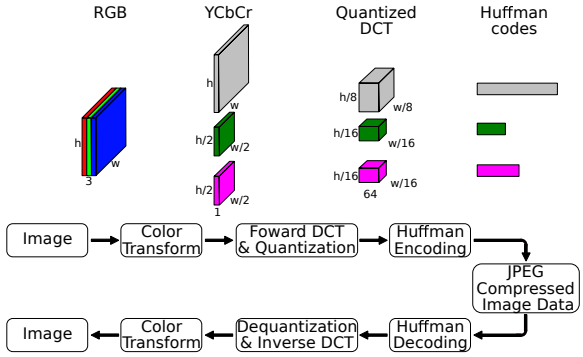


Figure 1. JPEG compression and decompression process [Gueguen *et al*., 2018].

Initially, the representation of the colors in the image is converted from RGB to YCbCr, which is composed of one luminance component (Y), representing the brightness, and two chrominance components, *Cb* and *Cr*, representing the color. Also, the *Cb* and *Cr* components are down-sampled horizontally and vertically by a factor of 2, as human vision is more sensitive to brightness details than to color details. Then, each of the three components is partitioned into blocks of 8×8 pixels and 128 is subtracted from all the pixel values. Next, each block is converted to a frequency domain representation by the forward discrete cosine transform (DCT). The result is an 8×8 block of frequency coefficient values, each corresponding to the respective DCT basis functions, with the zero-frequency coefficient (DC term) in the upper left and increasing in frequency to the right and down. The amplitudes of the frequency coefficients are quantized by dividing each coefficient by a respective quantization value

defined in the quantization tables, followed by rounding the result to the nearest integer. High-frequency coefficients are approximated more coarsely than low-frequency coefficients, as human vision is fairly good at seeing small variations in color or brightness over large areas, but fails to distinguish the exact strength of high-frequency brightness variations. The quality setting of the encoder affects the extent to which the resolution of each frequency component is reduced. If an excessively low-quality setting is used, most high-frequency coefficients are reduced to zero and thus discarded altogether. To improve the compression ratio, the quantized blocks are arranged into a zig-zag order and then coded by the run-length encoding (RLE) algorithm. Finally, the resulting data for all 8×8 blocks are further compressed with a lossless algorithm, a variant of Huffman encoding. For decompression, inverse transforms of the same steps are applied in reverse order. If the DCT computation is performed with sufficiently high precision, quantization and subsampling are the only lossy operations whereas the others are lossless, therefore they are reversible.

3 Related Work

In this section, we present a review of the literature of deep learning methods that use DCT coefficients as input. We divided such works into two categories according to how they are related to our approach: (1) works that aim to improve performance of CNNs by using information from the frequency domain, like DCT coefficients; and (2) works that focus on deep models designed to process data directly from the frequency domain in the form of DCT coefficients.

In the first category, Ehrlich *et al*. [2020] proposed a single model capable of dealing with variable JPEG qualities, called Quantization Guided JPEG Artifact Correction (QGAC) where the CNN is parameterized with the quantization matrix of the JPEG images. Also proposed by Ehrlich *et al*. [2021], the Task-Target Artifact Correction is a method that mitigates the performance penalty of using JPEG images with not ideal compression qualities. The proposed method uses the QGAC [Ehrlich *et al*., 2020] network to correct artifacts in the JPEG image before feeding them to the network. The novelty of the method is the fine-tuning of the artifact correction network in a unsupervised manner, using the L1 distance between the logits of the networks feed with compressed and uncompressed images. Temburwar *et al*. [2022] tackled another task with DCT CNNs, the content-based image retrieval (CBIR), avoiding the cost of decoding.

In the second category, for works that modify CNNs specifically to deal with DCT coefficients, Ehrlich and Davis [Ehrlich and Davis, 2019] reformulated the ResNet architecture to perform its operations directly in the JPEG compressed domain. Lossless linear operations were adapted to operate in the JPEG compressed domain, while the non-linear ones, like ReLU, were approximated. In a different direction, Lo and Hang [2020] explored the task of semantic segmentation directly on the DCT representation of the images. The proposed method uses the Frequency Component Rearrangement (FCR) technique to code the relationship between the DCT coefficients in a new dimension, feeding them to the proposed DCT-EDANet. The location of each DCT coefficient in a 8×8 block represents its frequency index, for this reason, when a convolution is performed, the frequency relationship is misinterpreted as spatial relationship, failing to extract essential features [Lo and Hang, 2020]. The FCR technique rearranges the DCT coefficients, separating the frequencies channel-wise in a new dimension. All the frequency relationships are exclusively represented on this dimension, allowing for the proper use of 2D convolution on the DCT representation. Other works like [Gueguen *et al*., 2018; Deguerre *et al*., 2019, 2021] and the methods we present here also follow Lo and Hang [2020], using FCR to rearrange the DCT. The work also employ manual selection of low frequencies DCT channels to reduce computational complexity.

Xu *et al*. [2020] proposed a method that can be applied to deep models with few modifications to the architecture to use information from higher resolution images, mitigating the loss of salient information caused by downsampling. The method consists of extracting the DCT coefficients of the image to obtain a representation on frequency domain, followed by a training process where the network learns to dynamically select only a subset of the most relevant frequencies jointly with the task.

To accelerate training and inference speed, Gueguen *et al*. [2018] proposed different architectural modifications to apply to the ResNet-50 network [He *et al*., 2016] to accommodate DCT coefficients. These coefficients can be obtained by partial decoding, thus saving the high computational load and memory usage of fully decoding the JPEG images. The modifications made to the model in order to accommodate the DCT consist of skipping the stem, the first stage of the ResNet-50 network, and altering the initial blocks of the second and third stages to mimic the increase in receptive fields and stride of the baseline RGB model. The approach was called Receptive Field Aware (RFA). In order to deal with the lower resolution of the *Cb* and *Cr* Chroma components, several methods were proposed. The Upsampling-RFA upsamples the *Cb* and *Cr* components to match the resolution of the *Y* component and the Deconvolution-RFA do the same process, but with a deconvolution layer instead. On the Late-Concat-RFA (LC-RFA), the *Y* component goes through the modified versions of stages 1 and 2 of the network, while the *Cb* and *Cr* components goes through a separate convolutional block. Then, all components are concatenated and fed to the fourth stage. Gueguen *et al*. [2018] also proposed the LC-RFA-Thinner, a version of the LC-RFA with altered number of channels on the first three convolutional blocks that the *Y* component goes through, respectively, from 1024, 512, 512 channels to 384, 384, 768.

Some works followed Gueguen *et al*. [2018], proposing to skip the stem and some modifications to early stages to adapt them for DCT coefficients. Deguerre *et al*. [2019, 2021] adapted the the Single Shot MultiBox Detector (SSD) [Liu *et al*., 2016] for object detection and later extended it for using the LC-RFA, Deconvolution-RFA and LC-RFA-Thinner as backbones for the SSD. Wang *et al*. [2023] adapted the Tiny-YOLO-v3, ResNet-18 and SqueezeNet networks. These models were used on an FPGA ZYNQ and proposed optimization strategies for the decoding step on this device. Their

results indicate that the decoding step is a bottleneck in the execution time of embedded devices commonly used for CNN acceleration. Not surprisingly, using CNNs designed for frequency domain can lead to speed-ups to 4.29 times faster. Santos *et al*. [2020] proposed the Frequency Band Selection (FBS) technique that manually discard high frequency DCT coefficients, reducing the number of channels in early stages and reducing the computational costs of the network. Abdellatef and Karam [2024] proposed a graph-based method to select the most important DCT frequency channels, discarding unimportant channels.

Recently, with the advent of transformer obtaining state-of-the-art result in several computer vision tasks, Park and Johnson [2023] proposed a strategy to use the ViT-Ti, ViT-S and SwinV2-T architectures directly on JPEG compressed images without the need for any changes to the model. The only adaptations made were to the embedding, where 8×8 DCT blocks are combined to create patches. To address the difference in resolution between the Y and chroma components, sub-block conversion was used in the Y component to increase from from 8×8 frequencies to 16×16, halving the spatial resolution and allowing concatenation with the Cb and Cr components.

There are also other recent works that use information from the compressed domain, but where not included in more details, since they have some key aspects that distance it from the scope of our work, like using a different compression standard than JPEG, only using other data from the compressed domain instead of the DCT coefficients, or being designed for videos instead of images. For instance, Chen *et al*. [2018] used Wavelet-like Auto-Encoder (WAE) to decompose the original image into two sub-images with lower resolution, reducing the computational cost; Jiang *et al*. [2019] propose a method to classify Chinese ink-wash paintings by preprocessing RGB images with grey scale conversion, Sobel edge detection, and morphological operations, converting to DCT, extracting features with a DCT CNN, and classifying using SVM; Similar to the work of Ehrlich and Davis [Ehrlich and Davis, 2019], Ayat *et al*. [2019] proposes to modify the LENET-5 to perform its operations on the spectral (frequency) domain, obtaining this representation with the Fast Fourier Transform (FFT); Sun *et al*. [2020] proposes a CNN designed for the DCT domain for the task of reducing compression defects by mapping relationships between the original images and JPEG compressed images; Qin *et al*. [2021] proposed the FcaNet, a model that uses frequency channel attention, weighting the importance of each input channel. Duan *et al*. [2023] propose to perform vision task directly on compressed images generated by Learned Image Compression (LIC) methods using CNNs, avoiding the decoding step. Zhang *et al*. [2023b] proposes a filter pruning method by measuring the uniqueness of the feature maps by calculating the similarity in the frequency domain, converting them to DCT coefficients to do so. The filters with feature maps of low uniqueness are removed, considerably reducing computational cost and amount of parameters while keeping similar classification performance. Su *et al*. [2024] proposed the DCT-Attention Down-sample (DAD) technique for reducing the resolution of feature maps using DCT and self-attention, using a proposed hybrid model of CNN and Vision transformer (ViT), achieving similar accuracy to other transformers based methods while having lower computational cost. Ji and Karam [Ji and Karam, 2024] adapted the Visual Transformer (ViT) architecture to work with the latent representation of learning-based compressed images, proposing the compressed-domain Vision Transformer (cViT).

Table 1 presents a comparison between the works from the category of models designed specifically for DCT coefficients, taking into consideration the baseline architecture, how the model handles the lower resolution from the chroma components, modifications to the DCT input that are done before feeding them to the model, architectural modification made to the model to adapt it for DCT coefficients and the computational complexity and amount of parameter from the DCT model compared to the RGB baseline.

As we can see, most of the works only focused on improving the effectiveness of a given model. When analyzing models designed for DCT coefficients, it is also very important to take into account their efficiency, since one of their major benefits is the speed-up obtained by partially decoding the images. However, in order to increase accuracy, many works end up increasing the complexity or adding a considerable amount of parameters to the model, losing this key advantage.

For this motive, despite the advances in this emerging field, there is still a need for studies that considers efficiency aspects, such as computational complexity and amount of parameters. Only a few recent works employed models that reduce both the number of parameters and the computational complexity for models designed for DCT Santos *et al*. [2020]; Park and Johnson [2023]; Abdellatef and Karam [2024]. Santos *et al*. [2020] was able to reduce both but suffered with performance drops. Park and Johnson [2023] and Abdellatef and Karam [2024] used transformer architectures and graph-based channel selection, respectively, to achieve this goal.

This works aims join this small and recent group of works by proposing to use efficient stems to reduce the amount of input channels of the network, allowing us to not increase the computational complexity of the models while keeping the speed up of avoiding the decoding, leading to efficient networks with a good balance between accuracy and computational cost. We also propose a new metric to evaluate the efficiency-effectiveness trade off.

4 Learning from the Compressed Domain

The DCT computation of a 8×8 pixel block requires 1920 floating point operations (FLOPs) [Hanzo *et al*., 2007]. Although it seems negligible, computer vision tasks usually involve the processing of a huge amount of images, each one containing many pixel blocks, therefore the total computational cost may be significant. For example, on the ImageNet dataset, considering the resolution of 224×224 as input size, it would be necessary 150.52 GFlops for decoding the 100,000 images of the test set, a considerable amount, and 1,806.25 GFlops for the 1,2 million images of the training set. These are significant values that reflect on the overall speed of the model.

References	Baseline Architecture	Chroma resolution	Input modifications	Model modifications	Complexity	Parameters
Gueguen *et al*. [2018]	ResNet-50	Upsampling, Downsampling Deconvolution and LC	FCR	Skip stem, RFA and thinner model	Lower	Higher
Ehrlich and Davis [2019]	Custom CNN	N.R.	N.R.	Adapted Ops. to frequency domain	N.R.	N.R.
Deguerre *et al*. [2019]	VGG-D	No chroma sub-sampling	Not used	DCT-dedicated first conv., skip stem	Lower	N.R.
Lo and Hang [2020]	EDANet	No chroma sub-sampling	FCR, Manual selection of low frequencies	Skip stem and increase depth	Lower	N.R.
Santos *et al*. [2020]	ResNet-50	Upsampling	FCR, Static channel selection	Skip stem, RFA, thinner model	Lower	Lower
Xu *et al*. [2020]	ResNet-50	No chroma sub-sampling	FCR, high resolution images, dynamic channel selection	Skip stem	N.R.	N.R.
Deguerre *et al*. [2021]	VGG-D and ResNet-50	Deconvolution and LC	FCR, Y channels only	Skip stem, RFA and thinner model	Lower	Higher
Wang *et al*. [2023]	ResNet-18, SqueezeNet, and Tiny-YOLO-v3	Upsampling	FCR	Skip stem and stride reduction	Lower	Higher
Park and Johnson [2023]	ViT-Ti, ViT-S and SwinV2-T	Sub-block conversion	Grouped, Separated and Concatenated embedding, DCT domain augmentations	No modifications	Lower	Lower
Abdellatef and Karam [2024]	ResNet-50	Deconvolution	FCR, Graph-based channel selection	Skip early layers, Skip stem	Lower	Lower
Ours	ResNet-50	Upsampling and LC	FCR, stem to reduce input channels	Skip stem, skip stages, stride reduction	Lower	Lower

Table 1. Comparison between works that design model to process data directly from the frequency domain in the form of DCT coefficients. N.R. denotes Not Reported.

Roughly speaking, the DCT can be seen as a convolution with a specific filter size of 8×8, stride of 8×8, one input channel, 64 output channels, and specific, non-learned orthonormal filters. As both the filter size and stride are equal to 8, spatial information of adjacent blocks do not overlap. In theory, a standard convolutional layer may learn to behave like a DCT, but in practice, this is not trivial, as the learned bases may be undercomplete, complete but not orthogonal, or overcomplete. In spite of that, the use of DCT weights as input for a CNN is feasible, since they can be seen as the outputs of a convolution layer with frozen weights initialized from the DCT filters [Gueguen *et al*., 2018]. Motivated by the aforesaid observations, we examine ways of integrating frequency domain information into CNNs. To present date, little work has been done to exploit the DCT representation widely used in compressed data as input for neural networks [Gueguen *et al*., 2018].

Our starting point were the models from Gueguen *et al*. [2018], which are adapted to facilitate the learning with DCT coefficients rather than RGB pixels. All models had similar accuracy. The main difference was in the way they deal with the lower resolution of the chroma components. Initially, we decided to focus on the Upsampling-RFA, since it is the model with the least amount of changes to the original ResNet-50. Later, in order to attempt to reduce computational cost even further, we adapted our strategies to the LC-RFA, that have a greater amount of architectural changes in comparison with the ResNet-50, but do not upsample the chroma components. A comparison among the original ResNet-50 [He *et al*., 2016], the Upsampling-RFA, the LC-RFA [Gueguen *et al*., 2018], and our proposed techniques is presented in Figure 2.

On the Upsampling-RFA (Figure 2-b), DCT coefficients obtained from the *Cb* and *Cr* components are upsampled in order to match the resolution of the *Y* component. Then, the three components are concatenated channel-wise, passed through a batch normalization layer, and fed to the second stage of the ResNet-50 network. The second and third stages of the ResNet-50 network were changed to accommodate the amount of input channels and to ensure that the number of output channels at the end of these stages is the same of the original ResNet-50 network.

Due to the smaller spatial resolution of the DCT inputs, early blocks of the second stage of the ResNet-50 network were altered to have a smoother increase of their receptive fields. For this reason, their strides were decreased in order to replicate the size increase of receptive fields in the original ResNet-50 architecture.

In the LC-RFA (Figure 2-c), the *Y* component goes through modified versions of the 2^{nd} and 3^{rd} stages, and the first convolutional block of the 4^{th} stage, all of them with the same stride reduction and increased amount of filters as the Upsampling-RFA, meanwhile, the *Cb* and *Cr* components go through a separate convolutional block. They are then joined and fed to the second convolutional block of the 4^{th} stage.

These changes in the ResNet-50 network raised its computation complexity and number of parameters. Although there was a speed-up by partially decoding the images, there was also an increase in the computational cost for passing them through the network once they are loaded in memory. Also, the increase in the amount of parameters indicates that, in order to reach a similar classification accuracy from the RGB model, the Upsampling-RFA and LC-RFA needed to use a more robust model than the original. In this way, the speed-up obtained by partially decoding the images is outweighed by the increase in the computational cost for passing them through the network once they are loaded in memory.

Motivated by the above observations, we examine different ways of dealing with those issues. Our goal was to obtain a model that works directly on DCT coefficients, but also have a similar amount of parameters and computational complexity from the original ResNet-50 for RGB images. Reducing computational complexity and number of parameters, but keeping similar classification performance is paramount on devices with limited resources, like mobile devices and embedded systems. These improvements facilitate the use of deep models on a wider array of different application and environments.

In Section 4.1, we tested our stems to reduce the amount of input channel. Then, in Section 4.2, we studied the effects of

avoiding the upsampling of the chroma components. Finally, in Section 4.3, we explored the effects of reducing the amount of layers of the CNN.

4.1 Reducing Input channels

In traditional CNNs, stem layers work as a compression mechanism over the initial image and are used to reduce the spatial dimensions of the input. The models of Gueguen *et al*. [2018] skip the stem due to the smaller spatial resolution of the DCT inputs, but the number of convolutional filters in their early blocks is increased to accommodate the higher amount of input channels. Motivated by this, we proposed to use stem layers that reduce the amount of input channels instead of spatial resolution.

As we can see in Figure 2-d, our stems reduce the number of input channels of the second stage to 64. To accommodate this amount of input channels, we change the number of convolutional filters of the second and third stages to be the same as the original ResNet-50, but we keep the decreased strides at early blocks, as proposed by Gueguen *et al*. [2018]. Three different stem layers were evaluated to decrease the number of input channels from 192 (i.e., 3 color components $\times$ 64 DCT coefficients) to 64: (1) a linear projection (LP), (2) a local attention (LA) mechanism, and (3) a cross channel parametric pooling (CCPP).

4.1.1 Linear projection (LP)

Linear projection is a simple strategy that is widely used in deep learning. For this reason, it was the first strategy stem we tested to reduce the number of input channels fed to the second stage of the model from 192 to 64. For example, residual networks, like the ResNet-50 [He *et al*., 2016], have shortcut connections in every block of a few stacked layers, as shown in Equation 1, where $F()$ is the residual mapping to be learned by the i-th block of stacked layers, W_i are its parameters, x are the input data, and y are the output feature maps.

$$y = F(x, W_i) + x \tag{1}$$

The $F() + x$ operation indicates the shortcut connection that enables an element-wise addition with the initial input x. The dimensions of $F()$ and x must be equal for the operation to be possible, when they are different, a W_s linear projection can be applied to match the dimension. Equation 2 describes a linear projection layer, where x has n feature maps and W_s is a weight matrix of size $m \times n$. The product $W_s \cdot x$ will generate an output y with m feature maps, where each one is a linear combination of all the n features from x.

$$y = F(x, W_i) + W_s \cdot x \tag{2}$$

The Linear Projection stem allows us to treat the DCT inputs as a whole, regardless of the significance of individual frequency components to the image content. Additionally, the linear transformation preserves the skewness or kurtosis (shape) of their distribution.

4.1.2 Local attention (LA)

Visual attention is the cognitive process where, given a natural scene, the most significant visual information is selected while other redundant information is filtered out, being an important mechanism for the human visual system [Fang *et al*., 2012].

Attention mechanics for CNNs can have many types of variants, like spatial, channel and self-attention, according to the dimensions of the input they focus. In the case of channel attention, importance weights are learned and attached to the input channels [Qin *et al*., 2021].

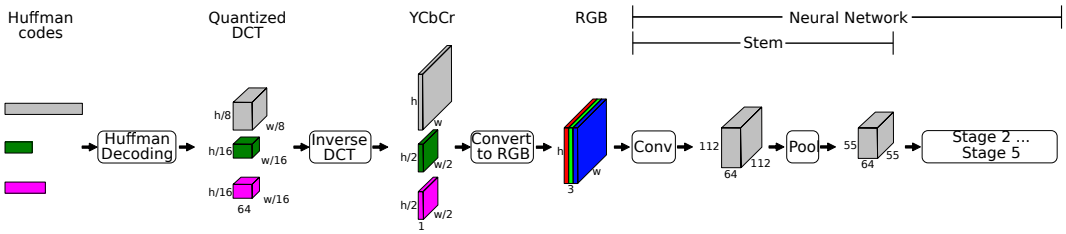
Originally proposed by Luong *et al*. [2015] for the machine translation task, local attention is a soft attention mechanism used in order to analyze a word taking into consideration a small context window of adjacent words. The objective is to learn attention maps that focus on the most relevant parts of the input. In our experiments, we adapt this strategy to be used in a local channel attention stem for DCT coefficients, reducing the number of channels from 192 to 64. Equation 3 describes this strategy, where x is an input with n feature maps, r is its reshaped version partitioning it into m groups of $\frac{n}{m}$ channels, W is a weight matrix of size $m \times (\frac{n}{m})$, y is an output with m feature maps, and $\odot$ is the Hadamard product.

$$\begin{aligned} r &= reshape\left(x, \left[m, \frac{n}{m}\right]\right) \\ s &= W \odot r \\ a_i &= softmax(s_i), \forall i \in \{1 \dots m\} \\ y_i &= a_i \cdot r_i, \forall i \in \{1 \dots m\} \end{aligned} \tag{3}$$

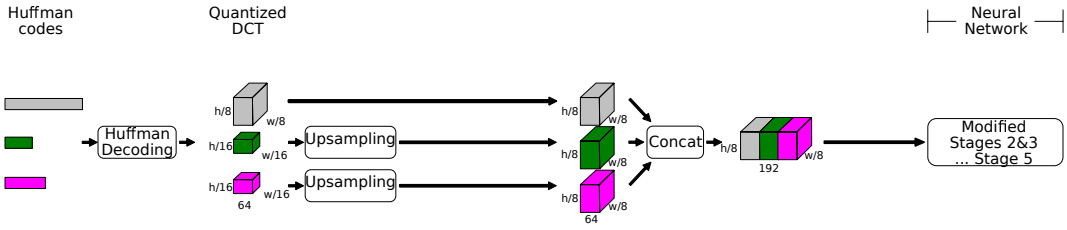
First, the input x is split into m partitions $r = \{r_1, \dots, r_m\}$ with $\frac{n}{m}$ features maps. Then, the Hadamard product between W and r is computed, resulting in the alignment scores s. A individual alignment score s_i is generated for each partition $i \in \{1 \dots m\}$. A softmax function is then used to normalization, generating the attention maps a_i, that are used amplify or attenuate the focus of the distribution of the input data r_i, generating the m feature maps y_i, one for each partition. These feature maps are linear combination of adjacent channels, preserving information of the DCT spectrum for the entire range of frequencies.

4.1.3 Cross Channel Parametric Pooling (CCPP)

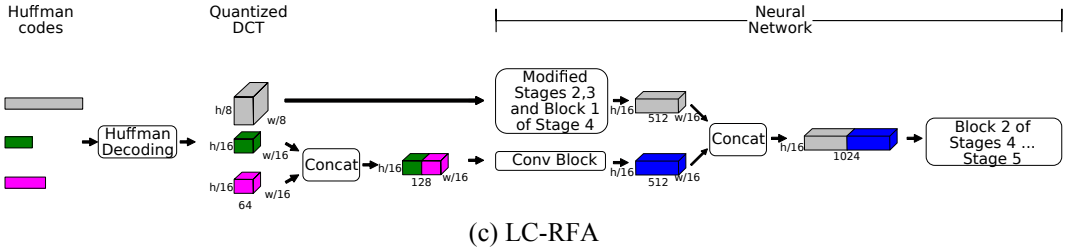
A cross channel parametric pooling layer is composed of a linear recombination of feature maps followed by a rectifier linear unit (ReLU) [Lin *et al*., 2013].



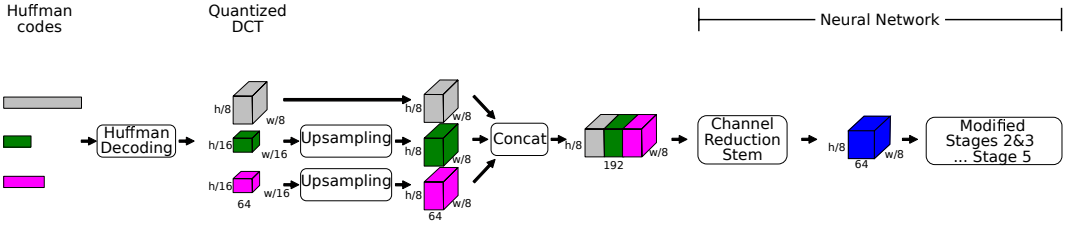
(a) Original ResNet-50 network



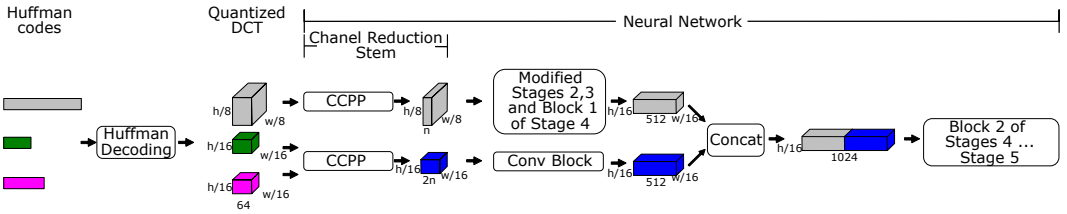
(b) Upsampling-RFA



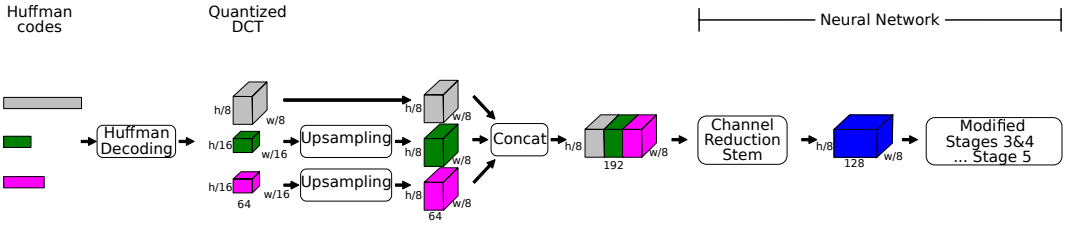
(c) LC-RFA



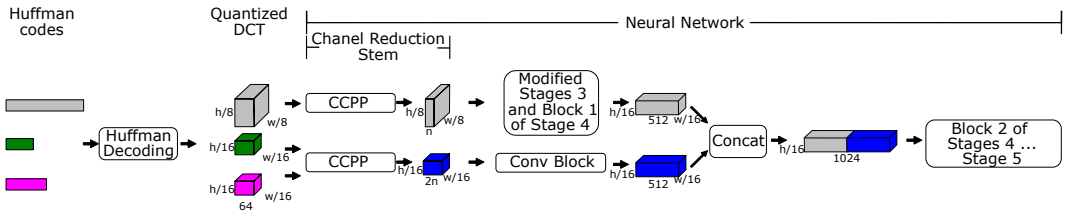
(d) Upsampling-RFA with channel reduction stem (ours)



(e) LC-RFA with channel reduction stem (ours)



(f) Upsampling-RFA with reduced amount of layers (ours)



(g) LC-RFA with reduced amount of layers (ours)

Figure 2. Illustrations of (a) the original ResNet-50 [He *et al.*, 2016], (b) the Upsampling-RFA and (c) the LC-RFA [Gueguen *et al.*, 2018], our improved versions from the (d) Upsampling-RFA and (e) LC-RFA with our stem to reduce the amount of input channels, and our method to reduce the amount of layers on the (f) Upsampling-RFA and on the (g) LC-RFA. Adapted from [Santos *et al.*, 2024].

A cascaded cross channel parametric pooling is a sequence of stacked cross channel parametric pooling layers and it can be used to replace usual convolution layer, since they have enhanced local modeling [Lin *et al*., 2013]. Equation 4 [Lin *et al*., 2013] describe a cascaded cross channel parametric pooling operation, where $f^l_{i,j,k}$ represents the output of the l-th layer, $x_{i,j}$ is the input patch centered at the pixel (i, j), k is used to index the feature maps, $W_{l,k}$ and $b_{l,k}$ are, respectively, weights and biases of the l-th layer for the k-th filter, and N is the number of layers.

$$\begin{aligned} f^1_{i,j,k} &= max\left(0, W^T_{1,k} \cdot x_{i,j} + b_{1,k}\right) \\ &\vdots \\ f^N_{i,j,k} &= max\left(0, W^T_{N,k} \cdot f^{N-1}_{i,j} + b_{N,k}\right) \end{aligned} \tag{4}$$

A cross channel parametric pooling is also equivalent to a pointwise convolution [Chollet, 2017], that consists of a convolution with a kernel size of 1×1. These layers are widely used by several CNNs architectures, being capable of making projections of input feature maps into new channel spaces, altering the amount of channels.

The cross-channel parametric pooling was used to decrease the number of channels from 192 to 64. Similar to the linear projection, this approach overlook the significance of individual DCT coefficients, but the use of non-linear ReLU activation promotes sparse feature maps, mitigating overfitting.

4.2 Adapting Our Strategy to a Different Architecture

All our previous strategies are modifications made to the Upsampling-RFA model proposed by Gueguen *et al*. [2018]. This model was chosen because, of all the strategies introduced by Gueguen *et al*. [2018], it was the one with the least amount of architectural changes compared to the ResNet-50. However, our proposed strategies can also be applied to other models, enabling them to learn from DCT coefficients rather than RGB pixels. With this motivation, we adapted our strategies to the LC-RFA, a model with more complex architectural changes than the Upsampling-RFA, but one that do not upsample the chroma components of the DCT inputs, reducing even further the preprocessing cost.

Originally, the LC-RFA has two input streams, one for the *Y* component of the DCT inputs and other for the concatenation of the *Cb* and *Cr* components. The output of both streams are concatenated and fed to the second convolutional block of the 4^{th} stage. In the *Y* stream, the 2^{nd} and 3^{rd} stages and the first convolutional block of the 4^{th} stage are modified, while the *CbCr* stream is fed to a convolutional block similar to the first of the 4^{th} stage, but with kernel size of 1. The amount of channels and stride of all the blocks are the same as the Upsampling-RFA, with the only difference being the first block of the 4^{th} stage, which has the amount of input and output channels halved, so when concatenated, they have the expected input size of the second block of the 4^{th} stage of the ResNet-50. In our previous experiments with the Upsampling-RFA, we kept the stride reduction in the early blocks, but we used the same amount of convolutional filters as the original ResNet-50. Then, we used different stem layers to reduce the number of channels from 192 to 64, i.e., the expected input shape for the second stage of the original ResNet-50.

In Figure 2-e we present our models based on the LC-RFA. As before, we adopted the same strategy, applying the stride reduction proposed by Gueguen *et al*. [2018], but using the same amount of convolutional filters as the original ResNet-50. If we keep the input depth of *Y* stream equals to the respective layers in the original ResNet-50, it would be 64; and, for the *CbCr* stream, if we use half the input depth of the respective layer, since the LC-RFA halves the input depth and the number of filters in this layer to account for the fact that it is repeated in both streams, it would be 128. These are exactly the sizes already supplied in the DCT inputs, i.e., 64 coefficients for *Y* and 64 each for *Cb* and *Cr*, totaling 128. For this motive, we decided to use CCPP as a stem layer to reduce the amount of input channels of each stream, allowing us to reduce the computational complexity even further.

In our first experiment, two layers of our CCPP were added, one for *Y* stream and other for the *CbCr* stream. They reduce the number of channels to 32 for *Y* and 64 for *CbCr*; or 16 for *Y* and 32 for *CbCr*. The input depth for *CbCr* is the double of that of *Y* stream, since together the *Cb* and *Cr* components have twice the number of coefficients.

4.3 Reducing the Number of Layers

Both the Upsampling-RFA and LC-RFA [Gueguen *et al*., 2018] skips the first stage of the original ResNet-50, feeding the DCT coefficients to the second stage, which is modified to accommodate the amount of input channels. To reduce the complexity and amount of parameters even further, we analyze the effects of skipping the second, third, and fourth stages from the Upsampling-RFA, but maintaining the stride reduction at the early blocks of the initial stage, as Gueguen *et al*. [2018].

Figure 2-f and g shows the architectures of our models that reduce the amount of layers. Our approach differs from Gueguen *et al*. [2018] Upsampling-RFA in that we opted not to increase the number of input channels in the initial stages. Doing so would have significantly increased the network's computational complexity. Instead, we maintained the same number of input channels as the original ResNet-50, that are 64, 128, 256, and 512 input channels for the second, third, fourth, and fifth stages respectively. To adapt the amount of channels for the value expected by the first stage that is kept, we employed ours stems described in the previous sections. It is worth noting that the number of DCT coefficients is similar to the number of input channels in the third stage, requiring a less drastic reduction than that needed for the second stage. The fourth and fifth stages expect a higher number of channels than the DCT inputs, so they are scaled up while maintaining the original data salient features. Since our results on the Upsampling-RFA were promising, we also tested skipping the second stage on our LC-RFA models with our stems to reduce the amount of input channels.

5 Experimental Protocol

All experiments were executed on a machine equipped with an Intel Core i7 6850K 3.6 GHz processor, 64 GBytes of DDR4-memory, and 3 NVIDIA Titan XP GPUs. The machine runs Ubuntu 18.04 LTS (kernel 5.0.0) and the ext4 file system. The models were implemented using torch 1.2.0, and the experiments were conducted on the ImageNet dataset. Since we did not have access to the models proposed by Gueguen *et al*. [2018], we re-implemented them following the specifications available and trained them from scratch. To make a fair comparison, we present the results of our networks trained from scratch, as well as the results obtained from models trained from other works. This allow us to show that the variance present in network training and the small variations in training procedures can generate variance in the results obtained by these models.

To compare our approach with the state-of-the-art, we selected models with similar alterations to the network architecture as our methods, attempting to reduce the amount of channels of the networks designed for DCT coefficients to make them more similar to the original ResNet-50 for RGB images. For this reason, we did not include models like the ones based on the LC-RFA Thinner [Gueguen *et al*., 2018], since they have a smaller amount of channels than the original ResNet-50, although these modifications could be applied to our methods.

The same hyperparameters from Gueguen *et al*. [2018] were used, with batch size of 128, initial learning rate of 0.05, momentum of 0.9, 120 epochs of training, reducing the learning rate by a factor of 10 every 30 epochs. Images were resized so the smallest side is 256 and the crop size is 224×224. For training, random crop and horizontal flip were used for data augmentation. For testing, only a center crop was used.

Initially, we conducted experiments using our stems to reduce the amount of input channels. Then, we performed experiments avoiding the cost of upsampling the chroma components. Finally, on our last experiment, we tested reducing the number of layers from our best models.

To compare our methods with the state-of-the-art, we evaluated them on ImageNet (ILSVRC-2012) [Russakovsky *et al*., 2015], a large-scale image classification dataset widely used in literature. It has 1,000 classes, with 1,281,167 images for training and 50,000 images for testing.

In order to compare the performance of the network on the image classification task, we used the accuracy (Acc) metric, while for comparing the efficiency, we used the amount of trainable parameters, the number of floating point operations (measured in GFLOPs), inference time and the frames per second (FPS). For measuring inference time, we used a single Titan XP GPU and followed Gueguen *et al*. [2018] by doing 10 runs of 200 predictions with batch size of 8. The final inference times are the average of all runs. Following Gueguen *et al*. [2018], we also loaded the data into the memory previously to the time measurements, but differently than these works, we loaded the JPEG compressed data, allowing us to analyze the effects of partially or fully decoding the images. We divided inference time in data preprocessing time and the time for passing the data through the network. Data preprocessing time includes the time for fully or partially decoding the images, applying the necessary operations and loading them into the GPU memory. All images were center cropped previously to the execution of such experiments. We use a libjpeg-turbo based implementation for the partial and full decoding of the JPEG images.

In most of our experiments, FPS is calculated over the average network inference time, since different strategies can be used to speed up the decompression, like GPU acceleration, and most of other work report results this way, like Deguerre *et al*. [2021], except for the ones in Section 6.4, where we also show the FPS based on the total time (i.e., including the preprocessing time), allowing us to see the impact of the decoding step. FPS from other works are also reported, but are not directly comparable to ours, since they use different GPU implementations and, in some cases, different experimental protocol to do the measurements.

In order to analyze the trade-off between computational efficiency and effectiveness, allowing us to select the best models for our goals, we propose the efficiency-effectiveness score S_E metric. It takes in consideration the accuracy, computational complexity and amount of parameters of the model, rewarding the one with a good balance among them, as we can see in Equation 5,

$$S_E = \frac{\frac{Acc}{O} \times \frac{Acc}{P}}{\frac{Acc_{baseline}}{O_{baseline}} \times \frac{Acc_{baseline}}{P_{baseline}}} \tag{5}$$

where Acc is the accuracy of the model, O is the computational complexity, P is the amount of parameters, $Acc_{baseline}$, $O_{baseline}$, and $P_{baseline}$ are respectively the accuracy, computational complexity, and amount of parameters from a baseline model, in this case, the ResNet-50. This way, the S_E metric is used to show the efficiency-effectiveness trade-off of a model compared to the baseline, highlighting the impacts caused by the changes applied to the ResNet-50.

The standard metrics to measure computational efficiency are the computational complexity (GFLOPs), the number of parameters, and FPS. They provide crucial, isolated measurements of computational cost, model size, and practical inference speed, respectively, but they do not capture the influence interplay between these factors have among themselves and with the performance metric of accuracy. For example, a model might achieve a favorable reduction in GFLOPs, but still use a large amount of parameters that make its deployment into a device with limited memory impossible. Or the model might have a significant lower number of parameters, but at an unacceptable cost to its classification accuracy. The proposed S_E score addresses this limitation by providing a single, interpretable score that reflects the quality of the trade-off between accuracy and all of these factors at once, allowing for a more direct comparison.

6 Experimental Results

This section presents the experimental results. Section 6.1 demonstrates the effects our stems for reducing the number of input channels. Section 6.2 discusses the effects of removing the upsampling operation of the model. Section 6.3 shows

the results of our experiments to evaluate the reduction in the number of layers. Finally, Section 6.4 presents a more in-depth analysis on the efficiency of the models, comparing the inference time and FPS of the networks.

6.1 Effects of Reducing the Number of Input Channels

We evaluated the computational complexity, amount of parameters, frames per second, and accuracy of our stems for reducing the amount of input channels from the Upsampling-RFA. Experiments were run on the ImageNet dataset, as it can be seen in Table 2.

Table 2. GFLOPs, number of parameters, FPS based on average network inference time, accuracy and S_E score our Upsampling-RFA based models.

Approach	GFLOPs	Params	FPS	Acc.	S_E
Gueguen *et al*. [2018] training:					
ResNet-50 (3x1)	3.86	**25.6M**	208	75.78	1.06
Upsampling-RFA (3x64)	5.40	28.4M	266	**75.94**	0.69
LC-RFA (3x64)	5.11	27.4M	267	75.92	0.75
Deconvolution-RFA (3x64)	5.39	28.4M	268	76.06	0.69
Deguerre *et al*. [2021] training:					
ResNet-50 (3x1)	3.86	**25.6M**	324	74.73	1.03
LC-RFA (3x64)	5.11	27.4M	318	74.82	0.73
LC-RFA Y (1x64)	5.14	27.6M	329	73.25	0.69
Deconvolution-RFA (3x64)	5.39	28.4M	319	74.55	0.66
Our training:					
ResNet-50 (3x1)	3.86	**25.6M**	588	73.46	1.00
Upsampling-RFA (3x64)	5.40	28.4M	494	72.33	0.62
LC-RFA (3x64)	5.11	27.4M	510	71.67	0.69
Upsampling-RFA + LP (1x64)	**3.20**	**25.6M**	492	69.62	1.08
Upsampling-RFA + LA (1x64)	**3.20**	**25.6M**	626	69.96	**1.09**
Upsampling-RFA + CCPP (1x64)	**3.20**	**25.6M**	**639**	69.73	**1.09**

The three models with the best classification accuracy were the ones trained by Gueguen *et al*. [2018], respectively, the Deconvolution-RFA, the Upsampling-RFA and the LC-RFA. All these models use DCT coefficients as inputs, and were able to achieve better classification accuracy than the RGB baselines. The training of these networks made by Deguerre *et al*. [2021] and us were able to obtain classification accuracy similar to the RGB, but did not surpassed it. In general, when comparing the same models trained by different authors, the best accuracy were obtained by Gueguen *et al*. [2018], followed by Deguerre *et al*. [2021] and us, showing that small differences in procedures and training variance can affect the performance of these models.

The FPS obtained were also different for each work, since different hardware, batch size and other configurations were used. In the experiments of Gueguen *et al*. [2018], the DCT-based models were able to obtain higher FPS than the RGB baseline. While for Deguerre *et al*. [2021], the LC-RFA and Deconvolution-RFA obtained lower FPS than the RGB model by a small margin, 6 and 5 frames, respectively, achieving a similar inference speed. For these reasons, in the remainder of this work, when multiple training results for the same network were available, we used ours to keep a fair comparison.

Among all the DCT-based models trained by Gueguen *et al*. [2018] and Deguerre *et al*. [2021], the one with the best classification accuracy only increased it by 0.36% compared to their RGB baseline, while the one with lowest complexity increased it by 32.38% and has at least 1.8M more parameters. Also from Deguerre *et al*. [2021], LC-RFA and Deconvolution-RFA obtained similar but smaller FPS than the ResNet-50. The results obtained by Deguerre *et al*. [2021] and us show that the modifications made to accommodate DCT coefficients to the ResNet-50, creating the Upsampling-RFA and LC-RFA, increased the computational complexity and number of parameters of the network, making it slower than the original model.

These results also indicate the effectiveness of our proposed S_E metric in highlighting the works with the best balance between efficiency (complexity, parameters and FPS) and effectiveness (accuracy). For example, the Upsampling-RFA from Gueguen *et al*. [2018] obtained the best accuracy and an FPS higher than the baseline ResNet-50 for RGB, but at the cost of significantly higher computational complexity and number of parameters. With those four metrics that measure different aspects of the model performance, it can be hard to have a clear consensus on the model overall qualities. Our proposed S_E metric can be used to deal with this problem, since it takes in consideration the interaction between complexity, parameters and accuracy in a single metric. It clearly highlights that although the Upsampling-RFA obtained higher accuracy than the RGB baseline, the trade-off of higher complexity and number of parameters was not favorable, leading to a considerably lower S_E score.

In our experiments, similar to Deguerre *et al*. [2021], the baseline Upsampling-RFA and LC-RFA obtained FPS lower than the RGB by 94 and 78 frames, respectively. Also, they had a small drop in classification accuracy (around 1.13% and 1.79%), demanded an increase in computational complexity (of approximately 39.9% and 32.38%), had more trainable parameters (2.8M and 1.8M more) and lower S_E score (0.38 and 0.31 points lower).

All our models have similar classification accuracy, computational complexity, number of parameters and FPS. Their amount of parameters were approximately the same as our RGB baseline, the complexity was approximately 17.1% lower and there was a increase 38 FPS for the Upsampling-RFA + LA (1x64) and 51 FPS for the Upsampling-RFA + CCPP (1x64). Upsampling-RFA + LA (1x64) performed slightly better, obtaining classification accuracy 3.5% lower than the RGB baseline, followed closely by CCPP and Upsampling-RFA + LP (1x64), that were 3.73% and 3.84% lower, respectively. The S_E scores of our models were also similar, with all being slightly higher (by 0.08 or 0.09 points) than the ResNet-50.

Since most stems obtained similar computational complexity, number of parameters and accuracy, we decided to use only the CCPP in the other experiments of this work due to it being similar to pointwise convolutions, a strategy commonly used in CNNs.

6.2 Effects Adapting Our Strategy to a Different Architecture

After testing our stems for reducing the number of input channels on the Upsampling-RFA, we attempted to adapt them to the LC-RFA to avoid the cost of applying the upsampling operation to the chroma components of the DCT, decreasing even further the preprocessing cost.

Among the proposed stems, we chose to use the CCPP because it is commonly applied in CNNs in order to obtain channel-wise projections of the feature maps, like in depth-wise separable convolutions [Chollet, 2017].

Two experiments were performed: (*i*) CCPP (1x32|1x64), which used two CCPP layers as stem, one to reduce the amount of channels of *Y* to n=32 and the other to reduce the concatenated *Cb* and *Cr* to 2n=64 channels; and (*ii*) CCPP (1x16|1x32), which reduced the amount of *Y* channels to n=16 and *CbCr* to 2n=32 channels. Table 3 shows the computational complexity, amount of parameters, accuracy on the ImageNet dataset, FPS and S_E score of the LC-RFA and our improved variants.

Table 3. GFLOPs, number of parameters, FPS based on average network inference time, accuracy and S_E score for our Upsampling-RFA and LC-RFA based methods.

Approach	GFLOPs	Params	FPS	Acc.	S_E
ResNet-50 (3x1)	3.86	25.6M	588	**73.46**	1.00
Upsampling-RFA (3x64)	5.40	28.4M	494	72.33	0.62
LC-RFA (3x64)	5.11	27.4M	510	71.67	0.69
Upsampling-RFA + CCPP (1x64)	3.20	25.6M	**639**	69.73	1.09
LC-RFA + CCPP (1x32|1x64)	3.14	**24.7M**	616	71.04	**1.19**
LC-RFA + CCPP (1x16|1x32)	**3.13**	**24.7M**	624	69.84	1.15

Our LC-RFA + CCPP (1x32|1x64) and LC-RFA + CCPP (1x16|1x32) obtained similar results. The one with CCPP (1x32|1x64) obtained slightly higher accuracy and S_E, while the other with CCPP (1x16|1x32) got slightly lower number of parameters, computational complexity, and higher FPS. The similarity in the results is due to the reduction in the number of channels in the LC-RFA with CCPP only affecting the first convolutional layer after the stem of the network from each stream, since the remainder of the model is kept with the same amount of input and output channels as the original ResNet-50. Comparing to the ResNet-50 for RGB images, the LC-RFA + CCPP (1x32|1x64) reduced the computational complexity in 18.65%, had 0.9M fewer parameters, FPS was 28 frames higher, accuracy was 2.42% lower, obtaining an S_E score 0.19 point higher. Similarly, for LC-RFA + CCPP (1x16|1x32), the computational complexity was 18.91% lower, had 0.9M fewer parameters, obtained 36 more FPS, while there was a loss in accuracy of 3.62%, resulting in an S_E score 0.15 point higher. As we can see, the results obtained for these methods were similar to the Upsampling-RFA + CCPP, having slightly better computational cost while the amount of parameters and FPS was slightly worse. The LC-RFA + CCPP (1x16|1x32) version gained 0.11% accuracy over the Upsampling-RFA + CCPP, being very similar, while the CCPP (1x16|1x32) gained 1.31% accuracy, being our DCT-based model with the highest accuracy by a very small margin. According to the S_E metric both LC-RFA + CCPP models were also slightly better than the Upsampling-RFA + CCPP. This was expected, since similar strategies were used to reduce the cost of both of these networks. This way, the LC-RFA models combined with CCPP as stem can offer a good balance between accuracy and computational cost.

6.3 Effects of Reducing the Number of Layers

In this section, we present the results obtained by our strategy to reduce the number of layers of Upsampling-RFA and LC-RFA, proposed in Sections 4.3 and 4.2, respectively. For the Upsampling-RFA, when stages of the network are skipped, we decrease or increase the amount of input channels in order to match the depth expected at the initial stage in which they are provided as input. Since all the strategies presented in Section 4.1 obtained similar results, we decided to use the CCPP as the stem of our networks, as it is commonly used in CNNs for making channel-wise projections [Chollet, 2017].

Table 4 presents the computational complexity and number of parameters from the Upsampling-RFA + CCPP model with different skipped stages. The results indicate that skipping the first and second stages of the network leads to reductions in computational complexity and the number of parameters. However, skipping more stages results in a substantial increase in computational complexity since earlier stages of the network reduce the input resolution. By skipping these stages, inputs with higher resolutions are fed to the network, generating an increase in computational costs. Furthermore, the decreased strides at the initial stage's early blocks also increase the cost of convolution operations. Therefore, in subsequent experiments, we only consider skipping the first and second stages to save computational costs.

Table 4. GFLOPS and number of parameters for our Upsampling-RFA + CCPP when skipping different stages.

Approach	GFLOPs	Params
Skip the 1º stage	3.20	25.6M
Skip the 1º and 2º stages	**2.86**	25.1M
Skip the 1º, 2º, and 3º stages	8.26	23.9M
Skip the 1º, 2º, 3º, and 4º stages	10.76	**15.8M**

A comparison between computational complexity, number of parameters, FPS, accuracy and S_E score between our models presented previously and the Upsampling-RFA + CCPP and LC-RFA + CCPP skipping 1^{st} and 2^{nd} stages is shown in Table 5. Notice that skipping these stages benefited not only computational costs, but also the accuracy. It achieved the best FPS and fourth best classification accuracy among the DCT-based networks. Indeed, its classification accuracy is surpassed only by Upsampling-RFA and LC-RFA [Gueguen *et al*., 2018], whose FPS, computational complexity and number of parameters are worse than the original ResNet-50, and to the LC-RFA + CCPP (3x32) that had higher computational complexity and lower FPS. Comparing the Upsampling-RFA + CCPP skipping 1^{st} and 2^{nd} stages to the original ResNet-50, the classification accuracy was only 2.97% lower, while there was a similar amount of parameters, a reduction of 25.91% in computational complexity and a gain of 183 FPS. Also, it was our model with the highest S_E score, being higher than the ResNet-50 by 0.27 points and, for this motive, this was considered our best model, achieving the best efficiency-effectiveness trade-off.

Similar results were obtained when skipping the first and second stages of the LC-RFA models in terms of computational cost, but their classification accuracy were slightly lower than their counterpart without skip. The model

Table 5. GFLOPs, number of parameters, FPS based on average network inference time, accuracy and S_E score for our strategies for reducing the number of input channels and layers.

Approach	GFLOPs	Params	FPS	Acc.	S_E
ResNet-50 (3x1)	3.86	25.6M	588	**73.46**	1.00
Upsampling-RFA (3x64)	5.40	28.4M	494	72.33	0.62
LC-RFA (3x64)	5.11	27.4M	510	72.75	0.69
Upsampling-RFA + CCPP (1x64)	3.20	25.6M	639	69.73	1.09
LC-RFA + CCPP (1x32|1x64)	3.14	24.7M	616	71.04	1.19
LC-RFA + CCPP (1x16|1x32)	3.13	24.7M	624	69.84	1.15
Upsampling-RFA + CCPP + skipping 1st and 2nd stages (1x128)	2.86	25.1M	**771**	70.49	**1.27**
LC-RFA + CCPP + skipping 1st and 2nd stages (1x32|1x64)	2.91	**24.4M**	735	70.04	1.26
LC-RFA + CCPP + skipping 1st and 2nd stages (1x16|1x32)	**2.90**	**24.4M**	727	69.14	1.24

with CCPP (1x32|1x64) obtained computational complexity 24.61% lower than the original ResNet-50, 1.2M fewer parameters, the lowest amount among all the networks shown in this work, 147 more FPS while having a loss in accuracy of 3.42%, achieving a S_E score 0.26 points higher. The model with CCPP (1x16|1x32) had the same amount of parameters than the CCPP (1x32|1x64), computational complexity 24.87% lower than the ResNet-50, 139 more FPS, accuracy 4.32% lower and S_E score 0.24 point higher. The small differences in FPS can be justified by the variance on the experiments. Overall, these models obtained the lowest amount of parameters from all models we tested, but were outperformed in computational complexity, FPS, accuracy and S_E by the Upsampling-RFA + CCPP skipping the same stages.

6.4 Analyzing the Efficiency of the Models

Table 6 presents the results of our experiment to measure inference time and FPS. As it can be seen, the data preprocessing time is lower than the network time, although it still has a considerable influence, being approximately 16.48% from the total inference time for the Upsampling-RFA, making the total FPS be 82 frames lower than the network FPS. It is important to emphasize that such results are hardware and software dependent and thus may change according to the experimental setup. The base RGB model, the ResNet-50, was the one with the highest data preprocessing time and overall prediction time, since it needs to fully decode the images before feeding them to the network. By using DCT coefficients that can be obtained by partial decoding of the images, the Upsampling-RFA and LC-RFA were able to reduce the data preprocessing time when compared to the ResNet-50. However, as the introduced modifications led to a higher network time, there were drops in FPS for both models, since the increase in the network time was more influential. The LC-RFA obtained considerably lower data preprocessing time than the Upsampling-RFA. This is due to the fact that no upsampling operation is applied to the *Cb* and *Cr* components, avoiding this cost, and that the resolution of these components are halved in relation to the *Y* component, reducing the cost of operations to be applied.

Our stems for reducing the amount of input channels greatly improved the network time while keeping similar data preprocessing time to the model they were added to, regardless of whether Upsampling-RFA or LC-RFA. Skipping stages of the networks yielded gains in data preprocessing time similar to our other models. Indeed, they were the strategies with the best network time among the tested models. The results we obtained here indicate that our stems for reducing input channels are promising, since they greatly reduce network time, being able to greatly speed-up the models.

Figure 3 compares the computational complexity, number of parameters, S_E score, and classification accuracy for the different models presented in this work evaluated on the ImageNet dataset. Although all models have similar accuracies, Upsampling-RFA and LC-RFA significantly increased the computational complexity and number of parameters from the original ResNet-50, leading to a significantly lower S_E score, indicating that the trade-off between efficiency and effectiveness is not beneficial. The use of our stems in the Upsampling-RFA was able to reduce the computational complexity and number of parameters while keeping a similar classification accuracy and greatly increasing the S_E score of the network. When used with the LC-RFA, similar computational complexity and FPS were obtained while the accuracy was slightly higher. In fact, it was the DCT-based model with the third best accuracy, behind only Upsampling-RFA and LC-RFA proposed by Gueguen *et al*. [2018], however, they have a considerably higher computational cost. By using our stem to reduce the number of input channels and skipping the 1st and 2nd stages, we obtained the DCT-based models with the lowest computational complexity, amount of parameters and highest S_E score, while keeping similar accuracy. Among them, the best one was the Upsampling-RFA + CCPP skipping 1st and 2nd stages, since it obtained the best S_E score, indicating that this model is the one with the best trade-off between efficiency and effectiveness.

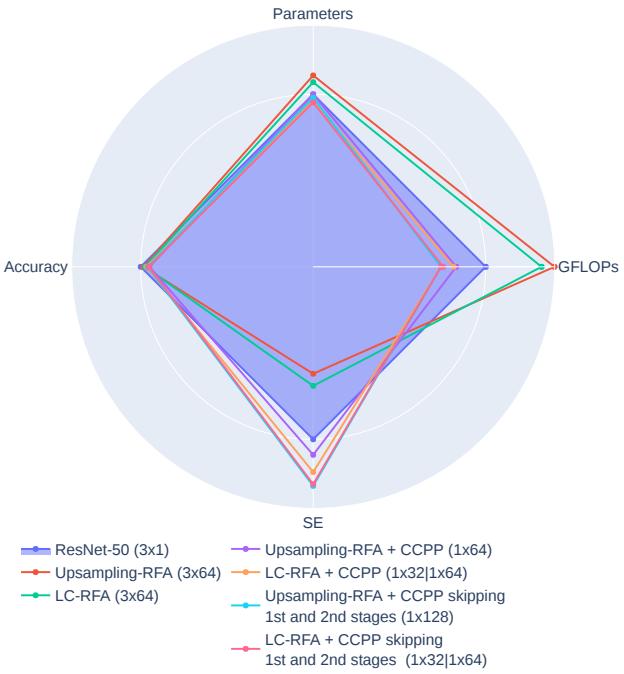


Figure 3. Comparison of computational complexity, number of parameters, S_E score, and classification accuracy from different networks for the ImageNet dataset. Values are shown in proportion to the ResNet-50.

Table 6. Inference time and frames per second for the original ResNet-50 for RGB and models designed for DCT. Inference times are averaged for 10 runs over 25 batches of size 8, ±indicates the standard deviation and frames per second (FPS) are calculated over the average network inference time (Network) and over the average total inference time (Total)

Approach	Inference Times (ms)			FPS		S_E
	Preprocessing	Network	Total	Network	Total	
ResNet-50 (3x1)	85.848 ±2.640	339.704 ±5.428	425.615 ±6.689	588	469	1.00
Upsampling-RFA (3x64)	79.820 ±2.015	404.396 ±2.471	484.268 ±3.396	494	412	0.62
Upsampling-RFA + LP (1x64)	80.240 ±2.082	406.354 ±6.505	486.651 ±7.925	492	410	1.08
Upsampling-RFA + LA (1x64)	82.484 ±2.953	319.416 ±5.744	401.955 ±7.667	629	497	1.09
Upsampling-RFA + CCPP (1x64)	83.099 ±2.718	312.987 ±18.200	396.134 ±20.276	639	504	1.09
Upsampling-RFA + CCPP + skipping 1st and 2nd stages (1x128)	83.232 ±3.599	**259.335 ±7.847**	342.624 ±9.908	**771**	583	**1.27**
LC-RFA (3x64)	69.379 ±5.261	391.433 ±4.049	460.857 ±8.454	510	433	0.69
LC-RFA + CCPP (1x32|1x64)	69.688 ±2.516	324.484 ±17.518	394.219 ±19.123	616	507	1.19
LC-RFA + CCPP (1x16|1x32)	**68.247 ±3.707**	320.320 ±18.981	388.610 ±22.341	624	514	1.15
LC-RFA + CCPP + skipping 1st and 2nd stages (1x32|1x64)	69.770 ±2.338	271.874 ±16.434	**341.687 ±18.625**	735	**585**	1.26
LC-RFA + CCPP + skipping 1st and 2nd stages (1x16|1x32)	70.366 ±3.820	275.031 ±15.022	345.441 ±18.151	727	578	1.24

7 Conclusions

In this paper, we presented a study on CNNs designed to operate directly on frequency domain data, learning with DCT coefficients rather than RGB pixels. These information are readily available in the compressed representation of images, saving the high computational load of fully decoding the data and greatly speeding up the processing time, which is currently a big bottleneck of deep learning, being crucial for environments with limited computational resources, like edge devices and embedded systems. The starting point of our work was the models proposed by Gueguen *et al*. [2018], Upsampling-RFA and LC-RFA, which are modified versions of the ResNet-50 architecture [He *et al*., 2016]. Despite the speed-up obtained by partially decoding JPEG images, their architectural changes raised the computational complexity and the number of parameters of the network. To solve this problem, we propose stems to reduce the amount of input channels, allowing us to avoid the increase in computational complexity of the models designed for DCT inputs, even reduce the overall cost by skipping stages. In this way, we keep the benefits of not decoding the images while the network cost itself is also lower than the RGB baseline, achieving good trade-offs between computational cost and accuracy. Our results showed that our stems are capable of learning how to combine DCT inputs in a data-driven fashion and reduce the computational complexity and number of parameters. As future work, we plan extend our approach for video classification using 3D network architectures, like Res3D [Tran *et al*., 2017] or I3D [Carreira and Zisserman, 2017], test our proposed method with CNNs designed to have low computational MobileNet [Howard *et al*., 2017], EfficientNet [Tan and Le, 2019], and SqueezeNet [Iandola *et al*., 2016], explore auto-encoders to reduce the input channels, following strategies like Chen *et al*. [2018], and evaluate our approach in large-scale datasets, like Kinetics [Kay *et al*., 2017].

Declarations

Authors' Contributions

SFS contributed to conceptualization, data curation, formal analysis, investigation, methodology, project administration, software, validation, visualization, and writing the original draft, review and editing. NS contributed to formal analysis, funding acquisition, project administration, resources, supervision, and review and editing. JA contributed to conceptualization, formal analysis, funding acquisition, investigation, methodology, project administration, resources, supervision, visualization, and review and editing. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Funding

This research was supported by the São Paulo Research Foundation - FAPESP (#2023/17577-0, #2024/04500-2, #2024/22985-3), FAPESP-Microsoft Research Institute (#2017/25908-6), by the Brazilian National Council for Scientific and Technological Development - CNPq (#315220/2023-6, #420442/2023-5, #444982/2024-8), the Coordination for the Improvement of Higher Education Personnel - CAPES (88881.624512/2021-01), by LNCC via resources of the SDumont supercomputer of the IDeepS project, by the EU Horizon project ELIAS (No. 101120237) and by the FIS project GUIDANCE (No. FIS2023-03251).

Availability of data and materials

The datasets and softwares generated and analysed during the current study will be made upon request.

References

Abdellatef, H. and Karam, L. J. (2024). Reduced-complexity convolutional neural network in the compressed domain. *Neural Networks*, 169:555–571. DOI: 10.1016/j.neunet.2023.10.020.

Ayat, S. O., Khalil-Hani, M., Ab Rahman, A. A.-H., and Abdellatef, H. (2019). Spectral-based convolutional neural network without multiple spatial-frequency domain switchings. *Neurocomputing*, 364:152–167. DOI: 10.1016/j.neucom.2019.06.094.

Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. In *IEEE/CVF Conf. on Comput. Vis. and Pattern Recog. (CVPR)*, pages 4724–4733. DOI: 10.1109/CVPR.2017.502.

Chen, T., Lin, L., Zuo, W., Luo, X., and Zhang, L. (2018). Learning a wavelet-like auto-encoder to accelerate deep neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1). DOI: 10.1609/aaai.v32i1.12282.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *IEEE/CVF Conf. on Comput. Vis. and Pattern Recog. (CVPR)*, pages 1800–1807. DOI: 10.1109/CVPR.2017.195.

Deguerre, B., Chatelain, C., and Gasso, G. (2019). Fast object detection in compressed jpeg images. In *IEEE Intell. Transp. Syst. Conf. (ITSC)*, pages 333–338. DOI: 10.1109/ITSC.2019.8916937.

Deguerre, B., Chatelain, C., and Gasso, G. (2021). Object detection in the DCT domain: is luminance the solution? In *IEEE Int. Conf. on Pattern Recog. (ICPR)*, pages 2627–2634. DOI: 10.1109/ICPR48806.2021.9412998.

Drews-Jr, P., Souza, I. d., Maurell, I. P., Protas, E. V., and C. Botelho, S. S. (2021). Underwater image segmentation in the wild using deep learning. *Journal of the Brazilian Computer Society (JBCS)*, 27:1–14. DOI: 10.1186/s13173-021-00117-7.

Duan, Z., Ma, Z., and Zhu, F. (2023). Unified architecture adaptation for compressed domain semantic inference. *IEEE Trans. on Circuits and Systems for Video Technology (TCSVT)*, 33(8):4108–4121. DOI: 10.1109/TCSVT.2023.3240391.

Ehrlich, M., Davis, L., Lim, S.-N., and Shrivastava, A. (2020). Quantization guided jpeg artifact correction. In *European Conf. on Comput. Vis. (ECCV)*, pages 293–309. Springer. DOI: 10.1007/978-3-030-58598-3_18.

Ehrlich, M., Davis, L., Lim, S.-N., and Shrivastava, A. (2021). Analyzing and mitigating jpeg compression defects in deep learning. In *IEEE/CVF Int. Conf. on Comput. Vis. Workshops (ICCVW)*, pages 2357–2367. DOI: 10.1109/ICCVW54120.2021.00267.

Ehrlich, M. and Davis, L. S. (2019). Deep residual learning in the JPEG transform domain. In *IEEE Int. Conf. on Comput. Vis. (ICCV)*, pages 3484–3493. DOI: 10.1109/ICCV.2019.00358.

Fang, Y., Chen, Z., Lin, W., and Lin, C.-W. (2012). Saliency detection in the compressed domain for adaptive image retargeting. *IEEE Trans. on Image Process. (IEEE TIP)*, 21(9):3888–3901. DOI: 10.1109/TIP.2012.2199126.

Ferraz, A. and Betini, R. C. (2025). Comparative evaluation of deep learning models for diagnosis of covid-19 using x-ray images and computed tomography. *Journal of the Brazilian Computer Society (JBCS)*, 31(1):99–131. DOI: 10.5753/jbcs.2025.3043.

Gueguen, L., Sergeev, A., Kadlec, B., Liu, R., and Yosinski, J. (2018). Faster neural networks straight from JPEG. In *Annual Conf. on Neural Information Process. Syst. (NIPS)*, pages 3937–3948. DOI: 10.5555/3327144.3327308.

Hanzo, L., Cherriman, P., and Streit, J. (2007). *Video Compression and Communications: From Basics to H.261, H.263, H.264, MPEG4 for DVB and HSDPA-Style Adaptive Turbo-Transceivers*. John Wiley & Sons, Chichester, SXW, UK, 2 edition. Book.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE/CVF Conf. on Comput. Vis. and Pattern Recog. (CVPR)*, pages 770–778. DOI: 10.1109/CVPR.2016.90.

He, L., Lu, W., Jia, C., and Hao, L. (2017). Video quality assessment by compact representation of energy in 3D-DCT domain. *Neurocomputing*, 269:108–116. DOI: 10.1016/j.neucom.2016.08.143.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861. DOI: 10.48550/arXiv.1704.04861.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *CoRR*, abs/1602.07360. DOI: 10.48550/arXiv.1602.07360.

Ji, R. and Karam, L. J. (2024). Compressed-domain vision transformer for image classification. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 14(2):299–310. DOI: 10.1109/JETCAS.2024.3394878.

Jiang, W., Wang, Z., Jin, J. S., Han, Y., and Sun, M. (2019). DCT–CNN-based classification method for the Gongbi and Xieyi techniques of Chinese ink-wash paintings. *Neurocomputing*, 330:280–286. DOI: 10.1016/j.neucom.2018.11.003.

Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., Suleyman, M., and Zisserman, A. (2017). The kinetics human action video dataset. *CoRR*, abs/1705.06950. DOI: 10.48550/arXiv.1705.06950.

Li, Y., Gu, S., Gool, L. V., and Timofte, R. (2019). Learning filter basis for convolutional neural network compression. In *IEEE Int. Conf. on Comput. Vis. (ICCV)*, pages 5623–5632. DOI: 10.1109/ICCV.2019.00572.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400. DOI: 10.48550/arXiv.1312.4400.

Liu, H., Liu, W., Chi, Z., Wang, Y., Yu, Y., Chen, J., and Tang, J. (2023). Fast human pose estimation in compressed videos. *IEEE Trans. on Multimedia*, 25:1390–1400. DOI: 10.1109/TMM.2022.3141888.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C.-Y., and Berg, A. C. (2016). SSD: single shot multibox detector. In *European Conf. on Comput. Vis. (ECCV)*, pages 21–37. DOI: 10.1007/978-3-319-46448-0_2.

Lo, S.-Y. and Hang, H.-M. (2020). Exploring semantic segmentation on the dct representation. In *ACM MMAsia*, pages 1–6. ACM. DOI: 10.1145/3338533.3366557.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine trans-

lation. In *Conf. on Empirical Methods in NLP (EMNLP)*, pages 1412–1421. DOI: 10.18653/v1/D15-1166.

Marchisio, A., Hanif, M. A., Khalid, F., Plastiras, G., Kyrkou, C., Theocharides, T., and Shafique, M. (2019). Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. In *IEEE Comput. Soc. Annu. Symp. on VLSI (ISVLS)*, pages 553–559. DOI: 10.1109/ISVLSI.2019.00105.

Marinó, G. C., Petrini, A., Malchiodi, D., and Frasca, M. (2023). Deep neural networks compression: A comparative survey and choice recommendations. *Neurocomputing*, 520:152–170. DOI: 10.1016/j.neucom.2022.11.072.

Ming, Y., Zhou, J., Hu, N., Feng, F., Zhao, P., Lyu, B., and Yu, H. (2024). Action recognition in compressed domains: A survey. *Neurocomputing*, 577:127389. DOI: 10.1016/j.neucom.2024.127389.

Park, J. and Johnson, J. (2023). Rgb no more: Minimally-decoded jpeg vision transformers. In *IEEE/CVF Conf. on Comput. Vis. and Pattern Recog. (CVPR)*, pages 22334–22346. DOI: 10.1109/CVPR52729.2023.02139.

Peng, L., Cao, Y., Sun, Y., and Wang, Y. (2024). Lightweight adaptive feature de-drifting for compressed image classification. *IEEE Trans. on Multimedia*, 26:6424–6436. DOI: 10.1109/TMM.2024.3350917.

Qin, Z., Zhang, P., Wu, F., and Li, X. (2021). Fcanet: Frequency channel attention networks. In *IEEE Int. Conf. on Comput. Vis. (ICCV)*, pages 783–792. DOI: 10.1109/ICCV48922.2021.00082.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F.-F. (2015). Imagenet large scale visual recognition challenge. *Int. J. of Comput. Vis. (IJCV)*, 115(3):211–252. DOI: 10.1007/s11263-015-0816-y.

Santos, S. F. and Almeida, J. (2020). Faster and accurate compressed video action recognition straight from the frequency domain. In *Conf. on Graphics, Patterns and Images (SIBGRAPI)*, pages 62–68. DOI: 10.1109/SIBGRAPI51738.2020.00017.

Santos, S. F., Sebe, N., and Almeida, J. (2019). CV-C3D: action recognition on compressed videos with convolutional 3d networks. In *Conf. on Graphics, Patterns and Images (SIBGRAPI)*, pages 24–30. DOI: 10.1109/SIBGRAPI.2019.00012.

Santos, S. F., Sebe, N., and Almeida, J. (2020). The good, the bad, and the ugly: Neural networks straight from jpeg. In *IEEE Int. Conf. on Image Process. (ICIP)*, pages 1896–1900. DOI: 10.1109/ICIP40778.2020.9190741.

Santos, S. F. d. and Almeida, J. (2021). Less is more: Accelerating faster neural networks straight from jpeg. In *Iberoamerican Congress on Pattern Recog. (CIARP)*, pages 237–247. DOI: 10.1007/978-3-030-93420-0_23.

Santos, S. F. d., Sebe, N., and Almeida, J. (2024). Efficient deep learning for image classification: Lighter preprocessing and fewer parameters. In *Conf. on Graphics, Patterns and Images (SIBGRAPI)*, pages 56–62. DOI: 10.5753/sibgrapi.est.2024.31645.

Su, K., Cao, L., Zhao, B., Li, N., Wu, D., Han, X., and Liu, Y. (2024). Dctvit: Discrete cosine transform meet vision transformers. *Neural Networks*, 172:106139. DOI: 10.1016/j.neunet.2024.106139.

Sun, M., He, X., Xiong, S., Ren, C., and Li, X. (2020). Reduction of jpeg compression artifacts based on dct coefficients prediction. *Neurocomputing*, 384:335–345. DOI: 10.1016/j.neucom.2019.12.015.

Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *Int. Conf. on Mach. Learn. (ICML)*, pages 6105–6114. DOI: 10.48550/arXiv.1905.11946.

Temburwar, S., Rajesh, B., and Javed, M. (2022). Deep learning-based image retrieval in the jpeg compressed domain. In *Adv. Mach. Intell. and Signal Process.*, pages 351–363. Springer. DOI: 10.1007/978-981-19-0840-8_26.

Tran, D., Ray, J., Shou, Z., Chang, S.-F., and Paluri, M. (2017). Convnet architecture search for spatiotemporal feature learning. *CoRR*, abs/1708.05038. DOI: 10.48550/arXiv.1708.05038.

Wang, X., Zhou, Z., Yuan, Z., Zhu, J., Cao, Y., Zhang, Y., Sun, K., and Sun, G. (2023). Fd-cnn: A frequency-domain fpga acceleration scheme for cnn-based image-processing applications. *ACM Trans. on Embedded Comput. Syst. (TECS)*, 22(6). DOI: 10.1145/3559105.

Xu, K., Qin, M., Sun, F., Wang, Y., Chen, Y.-K., and Ren, F. (2020). Learning in the Frequency Domain. In *IEEE/CVF Conf. on Comput. Vis. and Pattern Recog. (CVPR)*, pages 1740–1749. DOI: 10.1109/CVPR42600.2020.00181.

Zhang, J., Feng, Y., Wang, C., Shao, M., Jiang, Y., and Wang, J. (2023a). Multi-domain clustering pruning: Exploring space and frequency similarity based on GAN. *Neurocomputing*, 542:126279. DOI: 10.1016/j.neucom.2023.126279.

Zhang, Q., Zhang, M., Chen, T., Sun, Z., Ma, Y., and Yu, B. (2019). Recent advances in convolutional neural network acceleration. *Neurocomputing*, 323:37–51. DOI: 10.1016/j.neucom.2018.09.038.

Zhang, S., Gao, M., Ni, Q., and Han, J. (2023b). Filter pruning with uniqueness mechanism in the frequency domain for efficient neural networks. *Neurocomputing*, 530:116–124. DOI: 10.1016/j.neucom.2023.02.004.