Deep SELECTOR-JPEG: ADAPTIVE JPEG IMAGE COMPRESSION FOR COMPUTER VISION IN IMAGE CLASSIFICATION AND HUMAN VISION

by

Sepideh Shaterian Bidgoli

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2020

Author’s Declaration

Abstract

Deep Neural Networks (DNNs) demonstrate excellent performance in many Computer Vision (CV) applications such as image classification. To meet storage/bandwidth requirements, the input images to these CV applications are compressed using lossy image compression standards, among which JPEG is the most common. Classical JPEG is designed to consider Human Vision (HV) and pays a little attention to CV, resulting in classification accuracy drop of DNNs, especially at high Compression Ratios (CRs). This work presents Deep Selector-JPEG, an adaptive JPEG compression method that simultaneously targets both image classification and HV. For each image, Deep Selector-JPEG selects a Quality Factor (QF) adaptively to compress the image so that a good trade-off between the Compression Ratio (CR) and DNN classifier Accuracy (Rate-Accuracy performance) can be achieved over a set of images for a variety of DNN classifiers while the PSNR of such compressed image is greater than a threshold value predetermined by HV with a high probability.

Towards this end, Deep Selector-JPEG first defines a set of feasible QFs such that an image compressed at any QF within this set has PSNR greater than a predetermined threshold value with a high probability. For some images, multiple QFs within this set are suitable (ON) for compressing for a DNN classifier, which means compressing at these QFs at least maintains the ground truth rank of the original input for the DNN classifier. For a given image, Deep Selector-JPEG first determines the QFs that are ON among the set of feasible QFs. This problem is represented as a Multi-label Classification (MLC) problem since each image has multiple corresponding suitable QFs. We solve MLC using a binary relevance procedure, which involves training an independent binary DNN classifier for each QF within the feasible set to predict the ON/OFF labeling for each input image. Given a target CR, we empirically derive a subset of feasible QFs for this target CR and select the least QF that is ON in this set.

Experimental results show that in comparison with the default JPEG, Deep Selector-JPEG indeed achieves better Rate-Accuracy performance over the entire ImageNet validation set for all tested DNN classifiers with gains in classification accuracy up to $\approx 1\%$ at the same CRs, while satisfying HV constraints and keeping complexity under control.

Acknowledgements

Above all, I would like to thank my supervisor, Prof. En-hui Yang, for his guidance and support through my graduate studies. His insightful feedback taught me how to think critically and how to formulate a problem precisely. I learned the fundamentals of high-quality research from him that paved my way to establish my research directions. I appreciate his support and care for my current and future success.

I would like to thank Professor AmirKeyvan Khandani, and Professor Zhou Wang, for being my thesis committee members and providing me with valuable feedback.

I would like to thank my colleague in Multimedia Lab, Dr. Hossam Amer, for his help and friendship. His constant support helped me through my research process.

Last but not least, I would like to thank my parents for always encouraging and supporting me.

Dedication

To the passengers of flight PS752.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Motivation and Research Question

Recent forecasts from Cisco reveal that there will be large amounts of images by 2022 that needs to either be stored for or exchanged among Computer Vision (CV) applications. [1] Deep learning (DL) is a key to these CV applications due to its ability to extract desired features from raw pixels of input images without any domain knowledge [14]. To extract these features in the task of image classification for instance, deep neural networks (DNNs) learn the parameters of non-linear activation functions using a backpropagation learning algorithm. These functions progressively transform raw pixels of the input image to produce the output predicted label [14]. With this capability, DL proved a noticeable success in image classification with a steady Top-1 accuracy improvement on the large-scale and high-quality ImageNet dataset from 63.3% to 88.5% [14].

Raw pixels of these large-scale image datasets fed to underlying DNNs typically come from the pipeline of image acquisition, encoding, storage/transmission, and decoding. see Figure 1.1 This implies that these raw pixels are indeed compressed in a lossy manner to meet the storage and bandwidth requirements. Since the late 1980s, a range of image and video lossy codecs from JPEG to HEVC and beyond have been introduced to obtain a better trade-off between compression rate and human perceived quality for images. However, these standard codecs pay a little attention to CV [30]. For example, JPEG is a widely-used image codec that controls the trade-off between compression rate and human's perceived quality of the input image via a parameter called Quality Factor (QF). We have conducted an experiment which shows that if QF = 10 is used to compress all images in ImageNet validation set, a Compression Ratio (CR) 11.1x can be achieved at the expense of a drop of ≈8-10% in terms of classification accuracy of DNN classifiers. Even if the image perceptual quality at QF=10 is deemed acceptable according to Human Vision (HV), the $\approx$8-10% drop in classification accuracy may be too significant to be absorbed for CV. Therefore, it would be desirable to improve the trade-off between the JPEG CR and DNN classification Accuracy (Rate-Accuracy (RA) performance) while maintaining certain perceptual quality for HV. The question is, of course, how? This thesis answers this question.

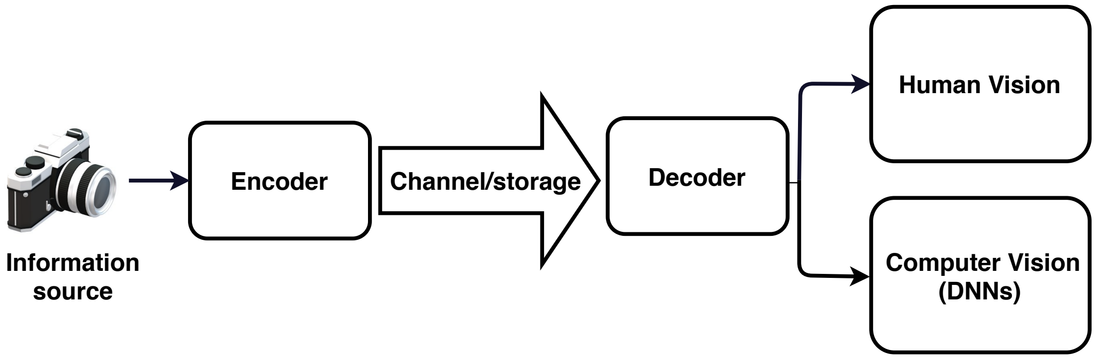


Figure 1.1: Pipeline of image acquisition, encoding, storage/transmission, and decoding for both human and computers.

1.2 Thesis Contributions

In this thesis, we introduce Deep Selector-JPEG, an adaptive JPEG compression method that simultaneously targets both HV and CV by DNN classifiers. For each image, Deep Selector-JPEG selects adaptively a QF from a set of feasible QFs to compress the image. The JPEG compressed image with the selected QF is then fed to a DNN classifier and viewed by humans. The set of feasible QFs is determined according to a PSNR threshold so that the JPEG compressed image with any feasible QF has its PSNR value greater than or equal to the PSNR threshold with a high probability, where the probability is calculated as if the image is taken randomly and uniformly from an image set, say, the ImageNet validation set.

The adaptive selection of a suitable QF from the set of feasible QFs by Deep Selector-JPEG is motivated by the observation that for any original image from the image set, its JPEG compressed images with some feasible QFs, once fed into a given DNN classifier, may actually maintain or even improve the ground truth (GT) rank of the original image for the DNN classifier [8]. Such JPEG compressed images with such feasible QFs are labeled “ON”, meaning that they are suitable as an input image to the given DNN classifier. Other JPEG compressed images with the remaining feasible QFs are labeled “OFF”. For each original image in the image set, the first step of Deep Selector-JPEG is to determine the "ON/OFF" label for each JPEG compressed image with a QF from the set of feasible QFs. Afterwards, the second step of Deep Selector-JPEG is to select the least QF labeled ON for the original image, and use the selected QF to compress the original image.

To solve the first step, we formulate a Multi-Label Classification (MLC) problem, where each original image is associated with a vector of labels that consists of an ON/OFF label of each QF from the set of feasible QFs. This problem is solved via Binary Relevance (BR), which involves training one independent binary DNN classifier for ON/OFF label of each QF from the set of feasible QFs. These binary classifiers have two forms in Deep Selector-JPEG. In the first form, we freeze all layers of a DNN architecture except its last two layers and use the frozen part as a common feature extractor for all binary classification problems. The last two layers of the same architecture are trained to predict the ON/OFF label of each QF inside the feasible set for each binary classification problem. To further increase the RA performance, the second form utilizes the entire DNN architecture to train each binary classification problem. In the first form, we use MobileNet-V2 as a representative of a light-weighted architecture or Inception V3 (IV3) as a representative of a heavy-weighted architecture. The second form utilizes MobileNet-V2 as well.

In the second step, for a target CR, a subset of the feasible QFs is empirically obtained. Compressing all images with each QF member in this subset will lead to a CR near to the given target CR. For each unseen image at test time, Deep Selector-JPEG selects adaptively the least QF that is ON from this subset to obtain an improved RA performance over the default JPG.

Experimental results show that in comparison with the default JPEG, Deep Selector-JPEG with either the light-weighted or heavy-weighted DNN architecture indeed achieves better RA performance over the entire ImageNet validation set for all tested DNN classifiers with gains in classification accuracy up to $\approx 1\%$ at the same CRs, while satisfying HV constraints. In summary, our contributions are listed below:

- We present Deep Selector-JPEG which selects adaptively a QF from a set of feasible QFs for each input image to serve CV in addition to HV while preserving compliance to the widely-used JPEG standard.
- Given the same PSNR threshold, the RA performance gain offered by Deep Selector-JPEG over the default JPEG is consistent across 10 different image classification DNNs which were tested, and the top-5 classification accuracy gain at the same CR can be as high as 1.07%.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 provides some core concepts used in this thesis. First, we provide an overview of JPEG compression. Then we review the concept of computer vision and, specifically, the task of classification. We describe the architectures of two popular DNNs, Inception Version3 (IV3) and MobileNetV2, utilized in the Deep Selector-JPEG design.

Chapter 3 presents Deep Selector-JPEG formation and architecture. Formulation of selection as an MLC problem is discussed. This problem is solved using binary relevance. The proposed solution and architecture are presented. We present the consideration of CV jointly with HV in this chapter. Our training methodology and hyper-parameter tuning are explained.

Chapter 4 explores the RA performance of Deep Selector-JPEG by presenting the experimental results of applying the selector on a set of ten most popular DNNs in the literature for four PSNR constraint. This set includes IV1[27], IV3[28], IV4[26], Resnet-50, Resnet-101 [9], InceptionResnetV2[26], MobileNetV1[10], MobileNetV2[20], Pnasnet-Large [16], Nasnet-Mobile[33]. The experimental training setup, GPU and CPU utilization, and complexity of our architecture are discussed. An example of applying the selector for a specific image illustrates the application of Deep Selector-JPEG.

Chapter 5 concludes the thesis and presents some possible directions to pursue in the future.

Chapter 2

Background

This chapter covers the core concepts used in this research thesis. Section 2.1 reviews JPEG image compression standard; The most widely used image compression standard in the literature. Section 2.2 introduces the computer vision concept and the specific task of classification. Sections 2.3 and 2.4 reviews architecture and details of two popular neural network families used in our design. Last but not least, section 2.4 summarized this chapter.

2.1 Overview on JPEG image Compression Standard

Every day more number of images is being generated, stored, and transmitted. These images are encoded using different compression standards to facilitate the process of storage and transmission. JPEG compression is the most popular one to compress images for Human Vision among different image compression standards. [19], [30] The pip-line of encoding transmission/storage and decoding of data for HV via JPEG compression is shown in Figure. 2.1

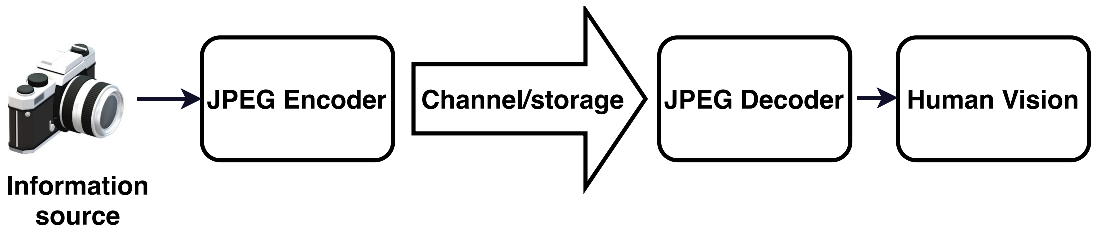


Figure 2.1: Pipeline of encoding transmission/storage and decoding using JPEG

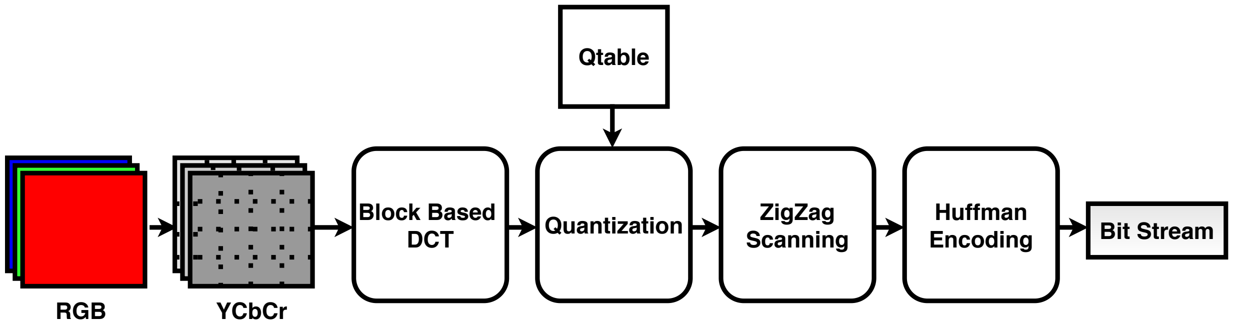


Figure 2.2: JPEG Compression Pipeline.

The pipeline of JPEG compression is shown in Figure 2.2. First, the RGB channels of the input image convert to YCbCr channels to decorrelate intensity information from color information. Y channel represents Luminance, and Cb, Cr channels represent Chrominance. Each channel is divided into 8x8 non-overlapping blocks. The human visual system is more sensitive to Luminance(intensity) than Chrominance (color); thus, CbCr are subsampled, and fewer number of bits are allocated to them. This process is called chroma subsampling.

Each 8x8 block of all channels converts to the frequency domain through Discrete Cosine Transform(DCT) and generates 64 DCT coefficients to exploit spatial redundancy. DCT transform reduces the correlation among pixels inside each 8x8 block in the spatial domain by decorating them to orthogonal basis functions. Equation 2.1 shows the DCT transform.

$$F_{u,v} = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f_{x,y} cos[\frac{2x+1}{u}\pi 16] cos[\frac{2y+1}{v}\pi 16] \tag{2.1}$$

Where $F_{u,v}$ is the DCT coefficient at frequency coordinate (u, v), $u \in [0, 7]$ and $v \in [0, 7]$, and $f_{x,y}$ is pixel value at spatial coordinate (x, y), and $\alpha(.)$ is a normalizing scale factor which is $\frac{1}{\sqrt{2}}$ if its argument equals 0 and 1 otherwise.

DCT transform provides energy compaction that means most of the block's energy resides in low-frequency components; thus, maintaining this component saves most of the information. Removing higher frequency components results in minor distortion in the reconstructed image. Keeping or removing the DCT coefficient through quantization controls the quality of the reconstructed image.

Quantization is a nonlinear, none-invertible operation that adds some distortion to the reconstructed image. JPEG compression utilizes uniform scalar quantization to quantize each of 64 DCT coefficients. A quantization for a DCT coefficient at frequency coordinate (u, v) is specified by its quantization step size $q_{u,v}$. The process of quantization is shown in Equation 2.2.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

Figure 2.3: Left: JPEG default quantization table for Y, Right: JPEG default quantization table for Cb, Cr [8].

$$F'_{u,v} = \lfloor \frac{F_{u,v}}{q_{u,v}} \rceil \tag{2.2}$$

Where $F_{u,v}$ is DCT coefficient at frequency coordinate (u, v) and $F'_{u,v}$ is its corresponding quantized value. Each $q_{u,v}, i, j \in [0, 7]$ for each frequency coordinate is tailored based on HV for all Luminance and Chrominance channels in JPEG compression, and all of them together form default quantization tables of JPEG. These tables for Luminance and Chrominance are shown in Figure 2.3 Left and Right, respectively.

Different quality version of the image can be generated by tuning default quantization tables via a parameter called Quality Factor (QF) Equation 2.3. shows how to tune the quantization table for different QFs.

$$q'_{u,v} = round(\frac{50 + S \times q_{u,v}}{100}) \tag{2.3}$$

Where $q'_{u,v}, u, v \in [0, 7]$ are entries of tuned quantization table to create quantization table at target QF, based on default quantization table entries $q_{u,v}$ shown in Figure 2.3, and $S = 5000/QF$.

After quantization, each 8x8 block will be converted to a 1D array using a zig-zag pattern, such that low-frequency content will appear at the beginning of the sequence, and high-frequency content will appear at the end of the sequence to conduct Run-Length Coding. At the end of the encoding process, each zig-zag ordered sequence would be encoded using the Huffman coding algorithm to produce the final bitstream.

The decoding performs all processes done in encoding in reverse order.

2.2 Overview on the Classification Task in Computer Vision

Computer Vision is the science of enabling computers to gain a high-level understanding of digital image and video data[4][24]. This field of science aims to enable computers to performs tasks that the human visual system does. Computer Vision tasks include processing and analyzing the digital images to extract image descriptors that can turn into decisions. [13], [21]. Some examples of these tasks are Image Classification, Object Detection, Image Segmentation, Visual Relationship Detection, Image Captioning, etc. Among all computer vision tasks, Image Classification is a core problem that forms a basis for many other tasks like object detection and segmentation. Image Classification is the problem of assigning an image to a category among k different categories.

Deep Neural Networks exhibit excellent performance in different areas of Artificial Intelligence, specifically Computer Vision. Their ability to extract high-level features from raw pixels of images makes them universal approximators that can learn any function from a training dataset [7]. For instance, in the task of classification, given a set of training images $X = \{x_1, x_2, ..., x_n\}$ with their corresponding labels $Y = \{y_1, y_2, y_3, ..., y_n\}$ where x_i is the vector of all pixels of an image, and y_i is the corresponding class, DNN learns the correspondence of X, Y by learning parameters of nonlinear activation functions that progressively transform pixels of x_i to predicted label y_i through a backpropagation algorithm. Using the learned function, DNN can predict the class of a new image instance.

Through years of research, different types of DNNs have been developed for different AI applications. The most popular DNN structure to tackle image classification is Convolutional Neural Networks (CNNs). CNNs consists of several convolution layers that act as feature extractors, followed by several fully connected layers in the tail to perform classification.

Since the beginning of the ImageNet Large Scale Visual Recognition Challenge in 2012, many advancements in designing CNN architectures have been introduced. ImageNet is a large dataset of annotated images introduces as a resource for computer vision research. ImageNet training set consists of 1.2 million images categorized by an expert human to 1000 different classes. The validation set contains 50000 images, with 50 images per category, and the test set contains 150000 images.

In our experiments, we use the ImageNet data set to train and test our methodology. As mentioned in chapter 1.2, We use Inception V3 and MobileNetV2 architectures to train both Selector-JPEG forms. The structure and performance of these architectures are discussed in the following sections.

2.3 Inception Architecture

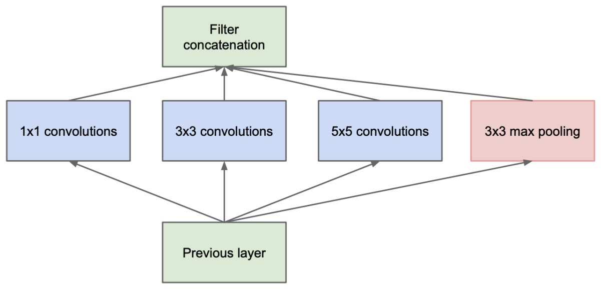
The Emergence of the Inception architecture in 2014 was a breakthrough in designing more efficient CNN architectures with higher classification accuracies. Since its introduction as Inception Version1 (IV1), Inception architectures developed trough time via batch normalization in Inception Version2 (IV2), optimization of inception module in Inception Version3 (IV3), and combination with residual networks in Inception Version4 (IV4). Each of these architectures is discussed in the following subsections.

2.3.1 IV1

IV1 [27] is the first CNN from the inception architecture family. This architecture is a revolutionary model that introduces the concept of modular layers to CNN design. Before Inception, the easiest and safest way to improve a CNN model's performance was stacking more convolutional layers. Although this method was able to produce acceptable results, it suffered from three main problems:

- More number of layers means more parameters that will result in over-fitting, specifically in the case of a limited dataset.
- Training for more number of parameters requires higher computational resources. If the added parameters are used inefficiently (e.g., in cases that most of the weight ends up to be zero), the extra resources are practically wasted.
- Deeper networks face the problem of vanishing gradient. As the gradient backpropagates to deeper layers, its value will gradually decrease, resulting in the intermediate layers not being affected by the network's loss.

IV1 addresses these problems by moving from dense connections to sparse connections, supported with the Heppian principle - neurons that fire together, wire together[3]. Authors of [27] introduce an inception module that is consists of three convolutions with different filter sizes (1x1, 3x3, 5x5) for different levels of spatial concentration, and then combine this information by concatenating the output vector of each convolution to form the final output vector that serves as an input vector to the next layer. To alleviate the extra complexity of 5x5 and 3x3 convolutions, IV1 utilizes dimension reduction and projection whenever the computational cost is high by utilizing 1x1 convolution before expensive 3x3 and 5x5 convolution. Figure 2.4 Left shows the inception module's naive version, while the Right part of the figure shows the inception module after applying reduction/projection layers. Inception V1 is a network consisting of staked layers of inception modules. Lower layers of the architecture stay as traditional convolutional layers, and the inception modules are added to higher layers. IV1 achieves 89.9 Top-5 and 69.8 Top-1 accuracy performance in ImageNet classification challenge.



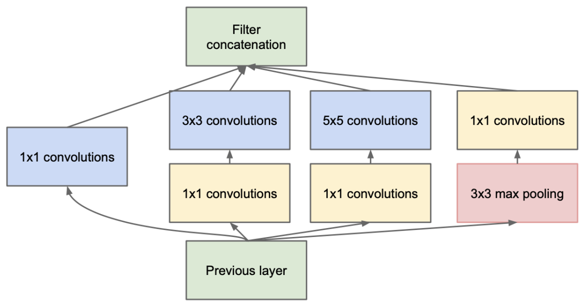


Figure 2.4: Left: Inception module Naive version, Right: Inception module with dimension reduction [27]

2.3.2 IV2

IV2 is an update on IV1 that solves the internal covariate shift problem by utilizing batch normalization[12]. The covariate shift problem happens when the distribution of inputs to a learning system changes, and the learning system has to adapt to this change. While a neural network is training, the parameters change at each step that causes a change in the distribution of output values of each layer that practically are the inputs to the next layer. This problem is called the internal covariate shift. The DNN should adapt itself to the new distribution; this requires a lower learning rate that results in a lower training speed. Batch normalization normalized each scalar features in each dimension of a layer of DNN over a batch of inputs to have zero mean and variance one that reduces the internal covariate shift and speeds up the training process. Another benefit of batch normalization is making the gradient value more independent of initial values and the scale, though the gradient flows through the network easier. Batch normalization prevents the network from stocking in saturation areas since the normalized values are guaranteed to have zero mean and variance. Hence, batch normalization solves the vanishing gradient problem and allows the network to use saturation nonentities like sigmoid. Simple normalization of each layer's input reduces the representational power of the network. To prevent this, authors in [12], introduce two learnable parameters γ and β for each activation x, that scales and shifts the normalized value. BN is summarized in Algorithm 1. IV2 achieves 91.8 % and 73.9 % Top-5 and Top-1 accuracy respectively.

Algorithm 1: BN algorithm [12]

```
**Input**: Input batch $B = x_1, x_2, ..., x_n$, trainable parameter $\beta$ and $\gamma$
**Output**: Normalazed output batch $\{y_i = BN_{\beta,\gamma}(x_i)\}$
$\mu_B \leftarrow \frac{1}{m}\sum_{i-1}^{m} x_i$
$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i-1}^{m}(x_i - \mu_B)^2$
$\hat{x_i} \leftarrow \frac{x_i - \mu_\beta}{\sigma_B^2 + \epsilon}$
$y_i \leftarrow \gamma\hat{x_i} + \beta \equiv BN_{\beta,\gamma}(x_i)$
```

2.3.3 IV3

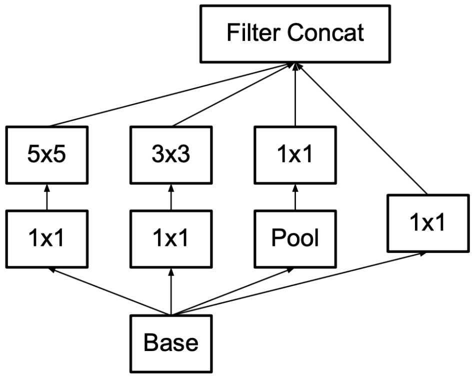
IV3 introduces three optimization ideas to optimize inception architectures efficiently. These ideas consist of factorized convolution, Utilizing auxiliary classifiers, and efficient grid size reduction. We explain each of these ideas in the following paragraphs.[28]

IV3 factorizes the convolution in two manners to reduce the computational costs. The first way is factorizing a convolution into two smaller ones. Inception utilizes convolution with a large kernel size, 5x5, to capture the similarity between input units that are far away from each other, but these convolutions are computationally not efficient. IV3 proposes replacing a convolution with a large kernel size with two progressive convolutions with smaller kernel size and argues that the receptive filed stays the same. Replacing 5x5 convolutions with two 3x3 convolutions, IV3 reduces the complexity by $\frac{9+9}{25}$x that means 28% gain in computational reduction. Figure 2.5 (a) shows the original inception module introduced in [28]. Figure 2.5 (b) shows the optimized inception module by replacing 5x5 convolution with two 3x3 convolutions.

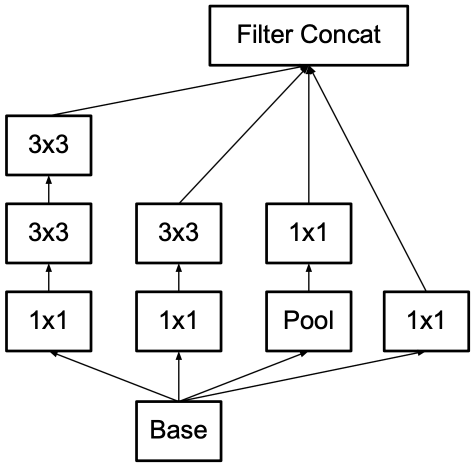
The second approach is spatial factorization to asymmetric convolution. Authors of [28] argue that any nxn convolution layer can be factorized to two layers of nx1 and 1xn convolution resulting in the same receptive field as nxn layer and less number of parameters. For example, factorizing a 3x3 convolution to two layers of 3x1 and 1x3 is 33% cheaper than a 3x3 convolution layer for the same number of input/output filters. Figure 2.6 shows the inception module after factorization of nxn convolutions.

Another optimization method that [28] uses is an auxiliary classifier to improve the gradient flow, specifically at the last training stages. Although the notion of auxiliary classifiers was introduced in [28], to improve the convergence and combating vanishing gradient problem, [28] have shown that auxiliary classifiers act more like a regularize and using auxiliary classifiers in the lower layer does not make any difference.

The third optimization idea proposed in [28] is an efficient grid size reduction. Conventional CNN models use pooling operation to reduce the grid size of feature maps. Typically the



(a) Original Inception Module



(b) Optimized Inception Module

Figure 2.5: Optimizing Inception module: a) Original Inception Module, b) Replacing 5x5 convolution by two progressive 3x3 convolutions [28]

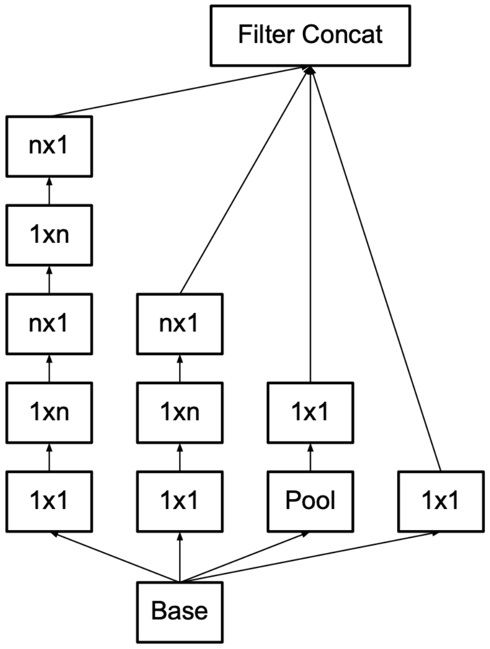


Figure 2.6: Inception module after factorization of nxn convolutions; n chose to be 7 [28]

number of output filter banks increases after a pooling layer to maintain the network's expressiveness, therefore utilizing pooling after a convolution layer increases the network's complexity; On the other hand, applying pooling before the convolution layer reduces the expressiveness of the network. Authors of [28] propose two parallel blocks, one with pooling and the other with convolution with the same number of filters, and then concatenate these two streams to create the final output and decrease the network's complexity together with maintaining its expressiveness. This solution is shown in Figure 2.7 Left. Figure 2.7 Right shows the applied solution to an inception module.

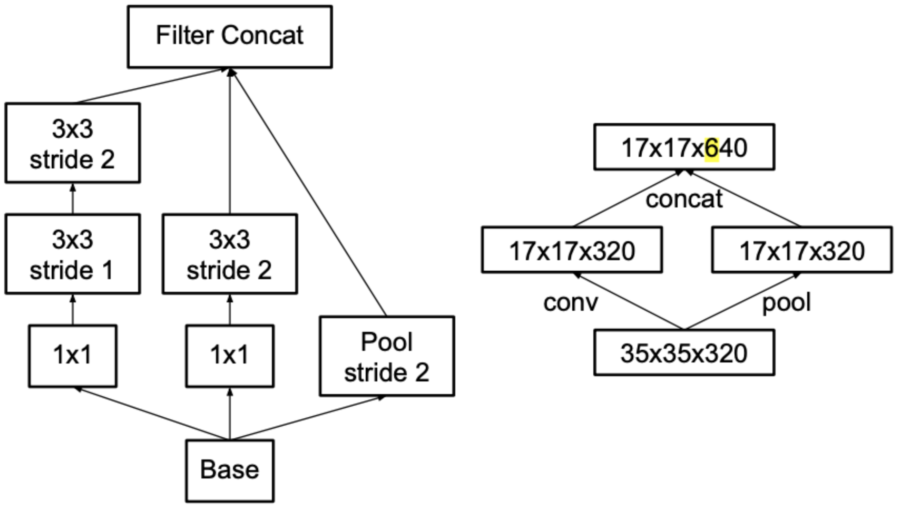


Figure 2.7: Left: Inception module with efficient grid size reduction and expanded filter banks. Right: Convolution stream and pooling that together double output filter bank size.[28]

IV3 architecture consists of layers of convolution in the head of network followed by staked layers of inception modules shown in Figure 2.5 (b), 2.6, and 2.7 and archives 93.8%, 77.6% Top-5 and Top-1 accuracy on ImageNet classification challenge respectively. We have used IV3 architecture to train our Selector-JPEG framework.

2.4 MobileNet Architecture

MobilNets are a class of small and efficient architectures designed for mobile, and embedded vision applications [10]. They have low complexity and are suitable to be implemented on edge devices for Internet of Things (IoT) applications. The details of MobileNetV1 and MobileNetV2 are presented in the following subsections.

2.4.1 MobileNet V1

MobileNetV1 utilizes depthwise-separable convolution instead of standard convolutions to reduce complexity and increase the speed of CNNs [10]. This type of convolution is a factorized version of standard convolution that results in fewer parameters and higher speed. Standard convolutions simultaneously filter and combine different channels of input feature maps to create output feature maps. The standard convolution filters with kernel size D_K and M number of input and N number of output channels is shown in Figure 2.8 (a). Applying standard convolution on input feature maps with spatial dimension D_F results in complexity shown in Equation 2.4.

$$D_K.D_K.M.N.D_F.D_F \tag{2.4}$$

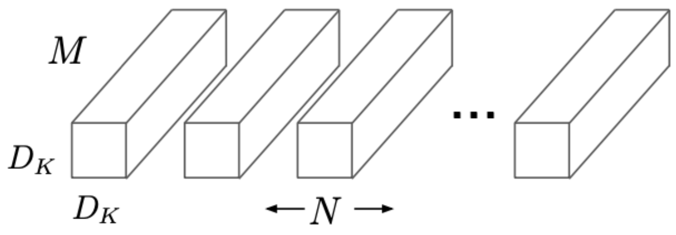
Depthwise-separable convolution split standard convolution into two steps, first filtering each channel and then combining each filtered channel's output to create the output feature map channels. The first step is called depth-wise convolution. The filters of the depth-wise convolution is shown in Figure 2.8 (b). A linear combination of depth-wise convolution outputs through a 1x1 convolution will result in a final output feature-map. This process is called point-wise convolution. Its convolutional filters are shown in Figure 2.8 (c). The complexity of Depthwise-separable convolution applied on feature map with spatial dimension D_F and M, N number of input and output channels is the summation of the complexity of depthwise and point-wise convolution as in Equation 2.5.

$$D_K.D_K.M.D_F.D_F + M.N.D_F.D_F \tag{2.5}$$

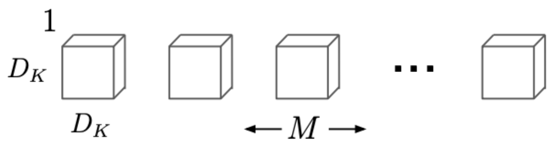
The computational cost of depth-wise separable convolution compared to standard convolution is shown in Equation 2.7

$$\frac{D_K.D_K.M.D_I.D_I + M.N.D_I.D_I}{D_K.D_K.M.N.D_I.D_I} \tag{2.6}$$

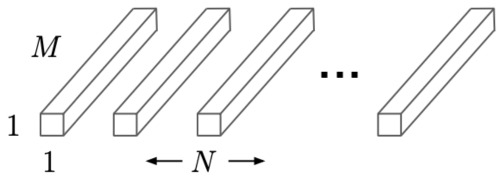
$$= \frac{1}{N} + \frac{1}{D_k^2} \tag{2.7}$$



(a) Standard Convolution



(b) Depthwise Convolution



(c) Pointwise Convolution

Figure 2.8: Standard convolution in (a) is replaced by two steps: (b) depth-wise convolution and (c) point-wise convolution. Rearranged from [10]

MobileNetV1 utilizes 3×3 kernel for depth-wise separable convolution, resulting in 9 times less computational cost than standard convolutions. The building block of MobileNetV1 using depthwise-separable convolution is shown in Figure 2.9 Right. MobileNet consists of 28 layers of depthwise separable convolutions.

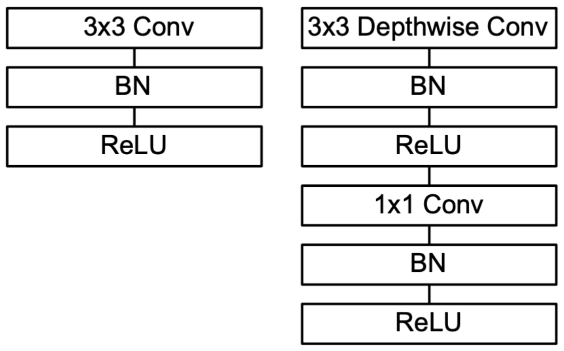


Figure 2.9: Left: Normal convolution with batchnorm and RELU, Right: Depth-wise separable convolution with batchnorm and RELU [10]

MobileNets utilizes two hyperparameters to customize the architecture based on the use case. Tuning these hyperparameters makes it possible to go beyond base MobileNet and achieve lighter and faster models. The first parameter is called ”Width Multiplier.” This parameter scale the size of each layer. Setting Width multiplier to α scales number of input, and output channels from M and N to αM and αN, which result in computational cost as Equation 2.8

$$D_K.D_K.\alpha M.D_F.D_F + \alpha M.\alpha N.D_F.D_F \tag{2.8}$$

Choosing $\alpha \in (0, 1]$ results in α^2 times less computational cost than base MobileNet with $\alpha = 1$. The second parameter is Resolution Multiplier ρ, which controls the input image's resolution and spatial dimensions of subsequent layers. Using width multiplier α and Resolution Multiplier ρ results computational cost as Equation 2.9. Where $\rho \in (0, 1]$ result in ρ^2 reduction in computational cost.

$$D_K.D_K.\alpha M.\rho D_I.\rho D_I + \alpha M.\alpha N.\rho D_I.\rho D_I \tag{2.9}$$

MobileNetV1 base acheives 89.9%, 70.9% Top5 and Top-1 accuracy on ImageNet classification challenge.

2.4.2 MobileNet V2

Inspired by the idea of depthwise separable convolution in [10], MobileNetV2 introduces bottleneck residual blocks to further reduce the complexity of a MobileNetV1. This block mainly consists of a depthwise separable convolution, same as MobileNteV1, preceded by an extra 1x1 convolution layer. In MobileNetV1, the pointwise convolution either keeps or increases the number of output channels; however, in MobileNetV2, this layer reduces the number of output channels creating a bottleneck layer. The last 1x1 convolution in bottleneck residual block is called a projection layer. On the other hand, the first 1x1 convolution layer plays the opposite role and expand the number of channels; this layer is called an expansion layer. The idea behind expansion and projection layers is to keep the low flow of parameters through the network while keeping the network's capacity since the filtering and processing step (depth wise convolution) is done on the expanded data. Another idea proposed in [10] adds residual connections between input and output of the block that allows the gradient flow easier through the network and speed up the training process.

Authors of [20], propose two types of bottleneck blocks: stride one and residual connection and the other with stride two and no residual connection. These two blocks are shown in Figure 2.10. MobileNetV2 consists of 17 of these building blocks in a row followed by a regular 1×1 convolution, a global average pooling layer, and a classification layer. MobileNetV2 achieves 92.5%, 74.9% Top-5 and Top-1 accuracy on ImageNet classification challenge. We used MobileNetV2 as one of our base architectures for designing Deep-Selector-JPEG

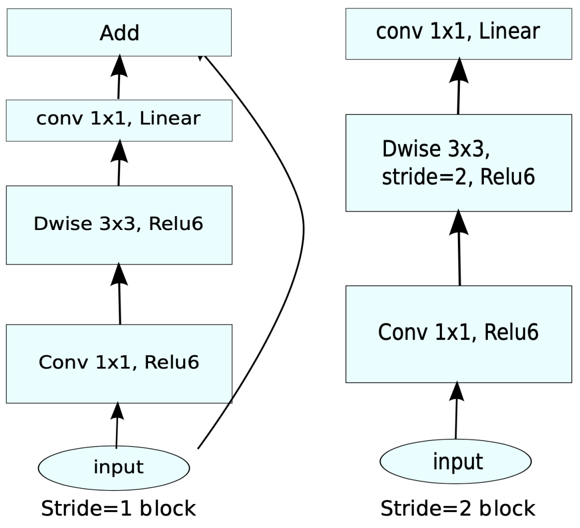


Figure 2.10: MobileNetV2 buildingc blocks Left:Stride 1 and residual connection, Right:Stride 2, no residual connections. [20]

2.5 Summary

This chapter covered an overview of the JPEG compression standard and pointed out that this standard is designed for human vision. We reviewed computer vision and specifically the classification task and introduced the ImageNet dataset, which is the most widely used computer vision research dataset. We covered the architecture and performance of two widely used CNN architectures in computer vision applications used in our Deep Selector-JPEG design, a compression method based on the JPEG standard that considers both human and computer vision. The details of this design are discussed in the next chapter.

Chapter 3

Adaptive JPEG Image Compression for Image Classification and Human Vision

3.1 Overview

As discussed in chapter 1.1, classical compression methods are designed for Human Vision (HV) and pay little attention to Computer Vision (CV). In this chapter we introduce Deep Selector-JPEG, an adaptive JPEG compression method that considers both HV and CV in the image classification task. Section 3.3 provides a case study that motivates our framework, while section 3.2 reviews the literature. Our methodology and design of Deep Selector-JPEG to consider CV are presented in section 3.4. The consideration of HV is discussed in section 3.5. Section 3.6 demonstrates our training methodology and hyper-parameter tuning for the framework. Finally, section 3.7 summarize this chapter.

3.2 Literature Review

Classical compression methods are mostly designed for the HV. With the rapid growth of deep learning-based vision tasks, digital images target CV applications as well as HV. [23]. The classical compression method shows significant degradation in Deep Neural Networks (DNNs) performance, specifically in high Compression Ratios (CRs). Adapting to the rapid growth of CV applications, designing a new compression method, or adapting the existing ones to consider CV perspective becomes critical.

In the literature, there are two directions to address the degradation problem caused by classical compression methods. The first direction is utilizing a compact representation of images to compress for both CV and HV. The second direction adapts the classical compression method for considering the only CV without investigating the effect of new compression on HV. In addition, authors in [32], [8] shows the possibility of improving the classification accuracy of a DNN model by using information from the original image and its compressed version. We review these works in the following paragraphs.

Compress compact representations of images for both computer and human utilization:

Authors in [22] aimed for joint image compression and understanding. They utilized an eight-layer deep residual network to derive a set of feature maps that can be used for both purposes. These feature maps are quantized through a scalar quantizer and entropy coded to derive the bitstream. On the decoder side, the binary stream is decoded, and a DNN is utilized to reconstruct the original image from the encoded feature maps. They trained this framework end-to-end for rate-distortion optimization. A classifier network is then used to perform image understanding. Fixing the compression network, they trained the classifier network on quantized feature maps. Their framework provides the ability to perform classification without decoding the image.

Another approach in [11] introduced vision-driven compact representations of images. These compact representations include sparse edges and color information of reference pixels randomly selected around edges. They compressed this information using entropy coding methods to derive the bitstream. On the other end, a generative model is trained to reconstruct the original image from this compact representation. This reconstructed image is fed for both humans and computers, which in their method they consider face landmark detection as the CV application. They trained the generative model to achieve a specific performance for face landmark detection while satisfying some HV criteria.

Adapt Classical compression for only computer vision utilization: Another line of research is adapting classical compression methods for CV applications. DeepN-JPEG introduced a rectified JPEG Q-table for all images based on frequency component analysis [17]. As a result, a CR=3.5x was obtained while achieving a classification accuracy near the original accuracy of underlying classifier DNNs. Furthermore, authors in [31] modeled the perception of a given DNN classifier via spatial frequency and color information of the input image using the gradient of loss of this DNN w.r.t the input. They proposed GRACE, a DNN aware compression algorithm, based on this perception model that successfully generated one JPEG Q-table for all images to minimize image size with bounded DNN perception loss constraint. However, their experimental classification results were conducted on a small subset of classes of the ImageNet dataset. Authors in [5] designed a bandwidth-efficient quantization for discrete wavelet transform coefficients in JPEG-2000 standard [29] to minimize the image size while minimizing the loss of a specific DNN over a dataset. Authors in [15] trained a reinforcement learning agent to choose the best Quality Factor (QF) of JPEG among a set of QFs to minimize the overall input size while minimizing the accuracy degradation of a cloud-based CV application.

Design new CNN topology using the compressed image information to enhance classification performance:

Authors in [8], [32] have shown the possibility of improving classification accuracy of a DNN model over original accuracy by designing a new CNN topology based on underlying DNN that takes the original input image and its 10 JPEG compressed versions as parallel inputs. The new CNN topology consists of 11 parallel architectures that share the same DNN structure. Each parallel architecture takes a version out of the 10 compressed versions, including the original version itself as input. The last logits block of all parallel architectures are concatenated to form the final logits, which is the input to the fully connected layer with 1000 classes to derive each class's output probabilities in the ImageNet classification challenge. Using IV3 as their underlying DNN, they achieve around 0.4% and 0.3% increase in Top-1 and Top-5 classification accuracy over the original accuracy of IV3 architecture; on the other hand, applying their method using Resnet-50 results in around 0.4% and 0.2% Top-1 and Top-5 accuracy improvement over Resnet-50 original architecture. These results showed the possibility of improving classification accuracy using the information from compressed images and the original image. Their method requires the storage of 11 compressed versions of dataset, which reduces the ability of applying this DNN topology in practice.

Deep Selector-JPEG presents a new compression method based on JPEG compression that jointly serves both human and computer vision. Selector-JPEG adaptively selects a QF for each input image such that gains in overall classification accuracy go up to $\approx$1% at the same CR of JPEG together with satisfying HV constraints. In addition, although Deep Selector-JPEG and the new CNN topology presented in [32] are motivated by the same observation that several QFs of an image can be suitable for a DNN classifier, the goal of Deep Selector-JPEG is designing a new adaptive compression method while the new CNN topology targets showing the possibility of improving classification accuracy of a DNN by using the information of compressed versions of images in the dataset. This observation is discussed in the following section.

3.3 Motivation: Case Study

This section provides observations that motivate us to design Deep Selector-JPEG. First, we review the conventional understanding that feeding a specific DNN classifier with a compressed image at low JPEG QFs always results in worst classification performance than feeding the DNN with the original image. Then we present some image examples for which compression will maintain or even improve classification performance.

The conventional understanding is based on the fact that classifier DNNs are typically trained and tested on high-quality image datasets, and feeding a compressed dataset at low QFs degrades their classification accuracy. Our experiments confirm this phenomenon; feeding Inception V3 with a dataset consists of images in the validation set of the ImageNet dataset all compressed at QF=10 results in $\approx 8\%$ drop in classification accuracy.

It worth noting that classification accuracy is a group notion. If we focus on a specific image, compressing with low QFs may not result in lower DNN classifier performance. The performance of a DNN classifier on a particular image is measured by the rank of the Ground Truth (GT) label of the input image in the sorted output probability vector of the DNN. The best performance is achieved if the GT rank is 1. We have observed that there exist some images that compressing them with lower QFs can achieve similar or better classification performance than the original image while feeding to a classifier DNN; in fact, several JPEG QFs are suitable (”ON”) for compressing these images for feeding to a given classifier DNN. A QF is suitable (ON) for an image if the corresponding JPEG compressed version with this QF at least maintains the Ground Truth (GT) rank of the original image. For instance, the rank of GT lable of original image #651 in ImageNet dataset on IV3 is 2. Compressing this image at different QFs inside set $L = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$ results in corresponding set of ranks $\{1, 2, 1, 2, 1, 1, 1, 2, 2\}$. For this image, all QFs are either keeping the original rank or increasing it, which means all QFs are suitable (ON) for feeding to IV3. Other examples are image numbers 300 and 898 in the validation set of the ImageNet dataset. The GT rank of their original version when applied to IV3 is 1 and 2, respectively. The GT rank outputted by IV3 in response to their compressed versions with QFs inside L, are $\{1, 1, 1, 1, 1, 1, 1, 1, 1\}$, $\{2, 2, 2, 2, 2, 1, 1, 1, 1, 1\}$ respectively. Same as image number 651, all QFs are ON for these two images.

It worth mentioning that different QF versions of these images have different sizes and perceptual qualities. Figure 3.1, 3.2, 3.3 shows the PSNR, GT rank, and CR of QF10, QF40, QF60, and the original version of image number 651, 300, 898 respectively. This observation suggests different QFs could be suitable for compressing an image for a DNN classifier and by selecting a suitable QF to compress an image one can improve the Rate Accuracy (RA) performance of a DNN classifier. For example, selecting QF=10 for images number 651, 300, and 898 in the ImagNet validation set saves bandwidth and achieves the same classification performance as the original image. The adaptive selection process for improving RA performance is discussed in the following section.

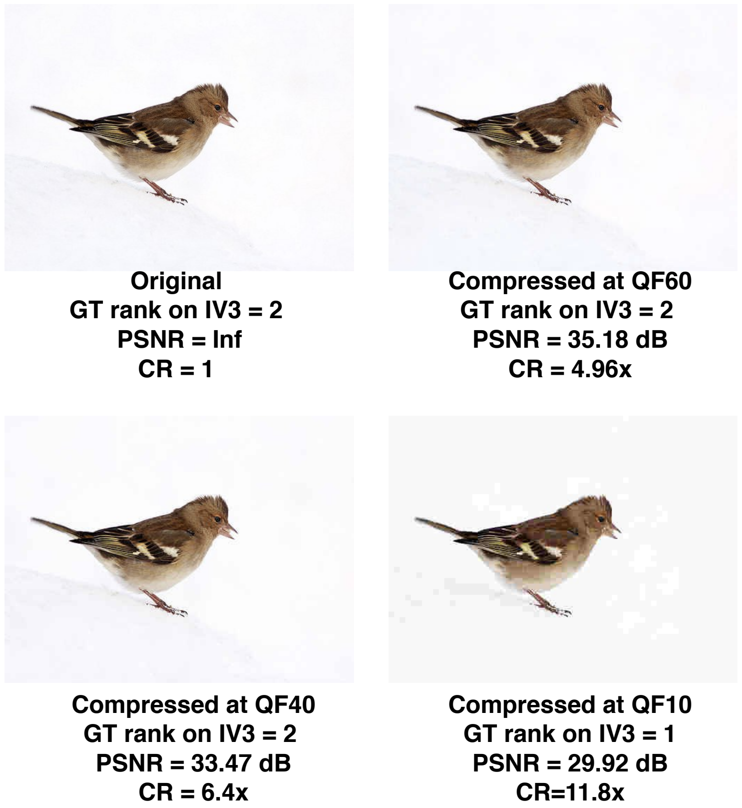


Figure 3.1: GT ranks, PSNRs and CRs of image number 651 in validation set of ImageNet dataset for original version and compressed versions at QF60, QF40, QF10.

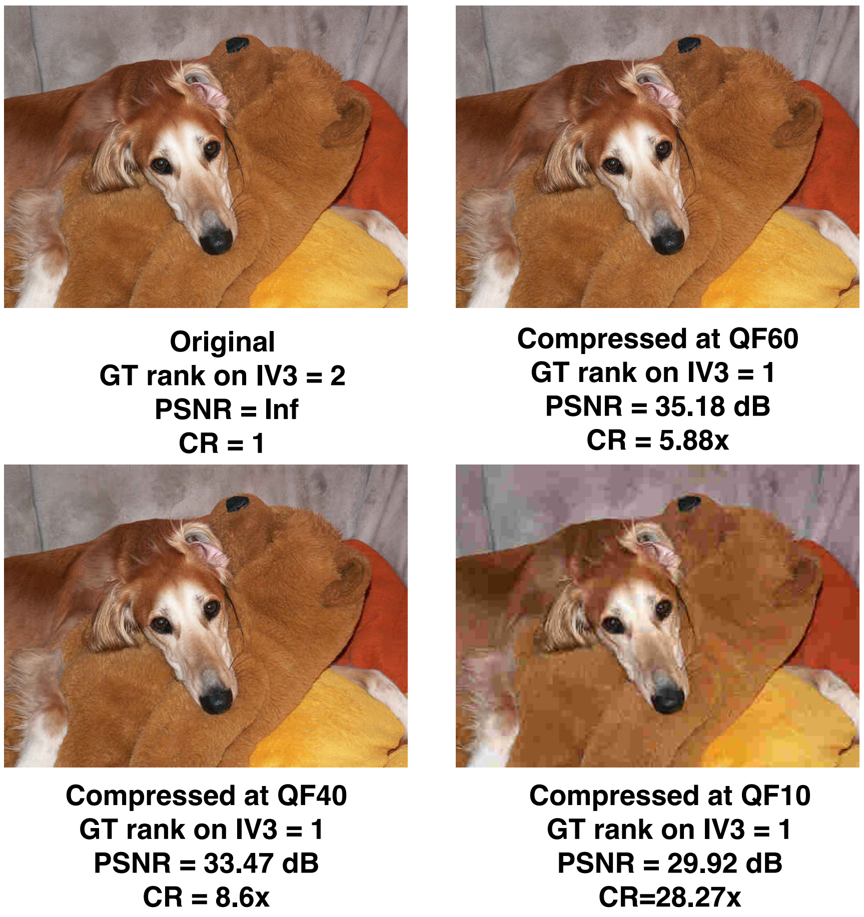
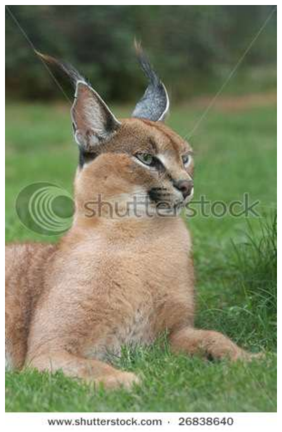
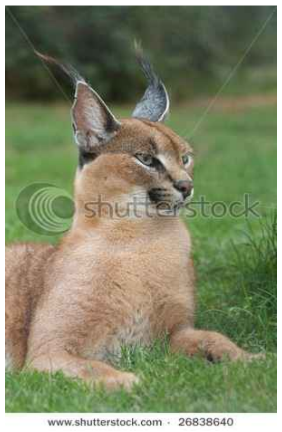


Figure 3.2: GT ranks, PSNRs and CRs of image number 300 in validation set of ImageNet for original version and compressed versions at QF60, QF40, QF10.





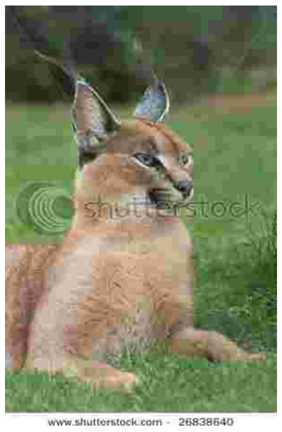


Figure 3.3: GT ranks, PSNRs and CRs of image number 898 in validation set of ImageNet for original version and compressed versions at QF60, QF40, QF10.

3.4 Deep Selector-JPEG: Formation, Architecture and Considering Computer Vision

Deep selector-JPEG goal is to adaptively select a QF for each image among a set of feasible QFs to improve Rate Accuracy (RA) performance of a set of images over the default JPEG. For each image it first detects the QFs that are ON and then among the set of ON QFs selects the one that provides a target CR.

To determine the ON QFs for each original image, Deep Selector-JPEG formulates an MLC problem where each original image is associated with a binary vector of labels that consists of an ON/OFF label of each QF from a finite and non-empty set of feasible QFs, $L = \{QF_1, \cdots, QF_j, \cdots, QF_n\}$. Deep Selector-JPEG tackles this MLC problem using BR learning. This learning method decomposes the given MLC problem with n QFs into n binary classification problems. Thus, each hypothesis y_j among n hypotheses $y_1, \cdots, y_n$ is induced to predict the ON/OFF of each QF_j, $j = 1, 2, \cdots, n$, for each original image $x \in X$, where X denotes the set of all original images. In this way, each y_j: $X \to \{0, 1\}$ is learned independently.

As shown in Figure 3.4, Deep Selector-JPEG determines each y_j: $X \to \{0, 1\}$ via "deep" supervised learning, using two forms of DNN architectures. Form One freezes a pretrained image classification DNN S except for its last two blocks and uses the frozen section as a common Feature Extractor (FE) for all binary classifiers. Based on the common features, each y_j then makes its own independent binary classification decision. In this case, only the function from the common feature to the binary decision (h_j) is learned independently from the training. To further improve RA performance, in the form two DNN architecture, the entire S is learned for each y_j independently.

For training, our training set is $T = \{(x_1, q_1), \cdots, (x_N, q_N)\}$, where $\{x_1, \cdots, x_N\}$ is the set of all original images in the ImageNet training set, and for each original image x_i, q_i is a binary vector $q_i = (q_{i,1}, \cdots, q_{i,n})$ with $q_{i,j} = 1$ indicating that QF_j is ON for the original image x_i. Ideally, the ground truth label $q_{i,j}$ should be determined by humans. That is, given the original image x_i and its JPEG compressed image with QF_j, a human should determine whether the compressed image would lead the human to believe that the GT rank is still at least maintained (i.e ON) with respect to its original image. Because of the sheer size of the image set, such a task is daunting. To overcome this difficulty, we instead replace human labellers with the underlying pretrained DNN classifier S in the Form One DNN architecture. Thus, in our training set, each ON/OFF label $q_{i,j}$ is determined as follows:

$$q_{i,j} = \begin{cases} 1 & R_{GT}(x_{i_{QF_j}}; S) \geq R_{GT}(x_i; S) \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where 1 means ON, and $R_{GT}(x_i; S)$ and $R_{GT}(x_{i_{QF_j}}; S)$ are the ranks of the GT label in the sorted probability vectors produced by S in response to the input image x_i and its corresponding compressed version at QF_j, $x_{i_{QF_j}}$, respectively.

To induce each y_j: $X \to \{0, 1\}$, the following binary cross-entropy loss function for training is utilized to obtain optimized weights W_j^*:

$$W_j^* = \arg\min_{W_j} -\frac{1}{N}\sum_{i=1}^{N} pr_j * q_{i,j} \log(p_j(x_i)) + (1 - q_{i,j}) \log(1 - p_j(x_i)) \tag{3.2}$$

where pr_j is the precision constant, a hyper-parameter to tune the trade-off between the recall and precision of y_j. Lower pr_j implies higher precision and lower recall for y_j. $p_i(x_i)$ is the output of sigmoid function ($\sigma(.)$) and has two forms:

$$\text{Form One: } p_j(x_i) = \sigma(h_j(FE(x_i); W_j^T)) \tag{3.3}$$

$$\text{Form Two: } p_j(x_i) = \sigma(h_j(x_i; W_j^T)) \tag{3.4}$$

From either (3.3) or (3.4), the predicted ON/OFF for a QF_j for each image x_i denoted by $y_j(x_i)$ is:

$$y_j(x_i) = \begin{cases} 1 & p_j(x_i) \geq DT_j \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where DT_j is another hyper-parameter called the decision threshold for QF_j. Higher DT_j implies higher precision and lower recall for y_j. Our training methodology and tuning the hyper-parameters to achieve different CRs are discussed in section 3.6.

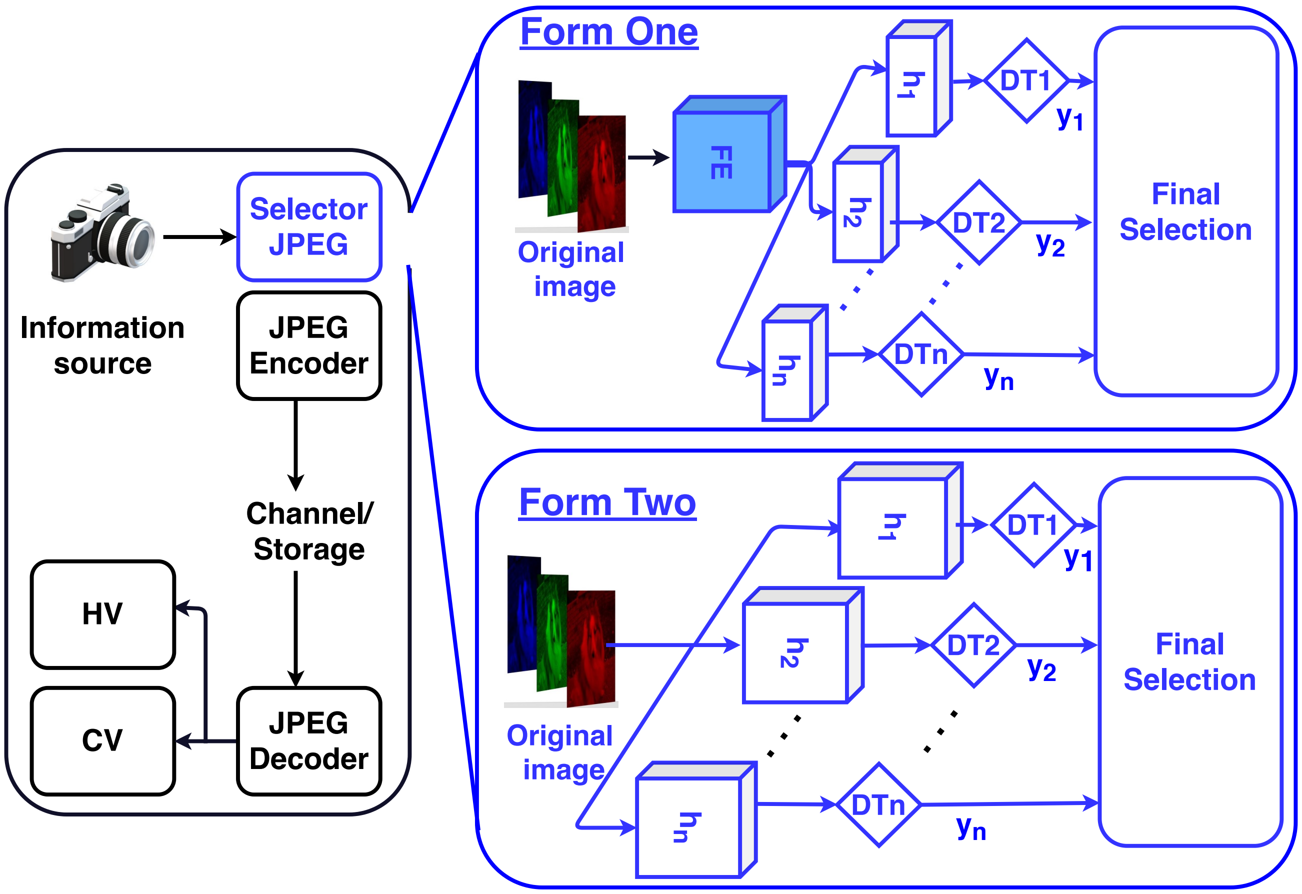


Figure 3.4: Deep Selector-JPEG Formation and Architecture.

3.5 Considering Human Vision

To consider HV into account, the feasible set of QFs, L, discussed in section 3.4 is defined to satisfy a minimum HV criteria. We use classical distortion measure, PSNR as the measure of HV satisfaction. To derive L, we first analyze the effect of QF on PSNR on ImageNet validation dataset. Toward this end, we create a set of QFs between 2 and 30 with step size 2 and between 30 and 90 with step-size 10, $L' = \{QF = 2k \mid 2 \leq QF \leq 30, k \in \mathbb{N}\} \cup \{QF = 10k \mid 30 \leq QF \leq 90k \in \mathbb{N}\}$. We start with a small step size for lower QFs and increase the step size for higher ones since in higher QFs, the differences in PSNR is smaller. We created a cluster of compressed images for each QF inside L'. For all images inside each cluster, the PSNR is calculated. For a target PSNR, we define Hit-Rate as the number of images that have higher PSNR than the target value and calculate the Hit-Rate for each QF.

The Hit-Rate of each QF for target PSNR 26dB, 28dB, 30dB, 32dB are shown in table 3.1, 3.2, 3.3 and 3.4 respectively. For each target PSNR, we choose QF_S, the starting QF in set L, to be the lowest QF with corresponding Hit-Rate higher than ≈90%. For target PSNR 26dB, 28dB, 30dB and 32dB, QF_s are 10, 20, 40, 60 respectively. These are pointed out as the green cells in each table. For each PSNR constraint, the set L is defined as $L = \{QF = QF_s + kr \mid k \in \mathbb{N}, QF_s + kr < m\}$. The parameter r is the step size between QFs in set L and m is the maximum QF. In our experiments we choose m to be 90, and r to be 10. Therefore, for target PSNRs 26dB, 28dB, 30dB and 32dB the feasible set of QFs are $L = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$, $L = \{20, 30, 40, 50, 60, 70, 80, 90\}$, $L = \{40, 50, 60, 70, 80, 90\}$ and $L = \{60, 70, 80, 90\}$ respectively.

Table 3.1: Hit-Rate of each QF for PSNR ≥ 26

PSNR	QF	PSNR	QF
2	41.10	24	98.21
4	62.37	26	98.55
6	79.99	28	98.90
8	87.21	30	99.15
10	90.91	40	99.80
12	93.22	50	99.96
14	94.75	60	100.00
16	95.85	70	100.00
18	96.64	80	100.00
20	97.22	90	100.00
22	97.79	100	100.00

Table 3.2: Hit-Rate of each QF for PSNR ≥ 28

PSNR	QF	PSNR	QF
2	11.64	24	92.19
4	29.81	26	93.19
6	54.77	28	94.09
8	67.54	30	94.86
10	74.77	40	97.49
12	79.71	50	99.04
14	83.14	60	99.84
16	85.73	70	100.00
18	87.89	80	100.00
20	89.44	90	100.00
22	90.90	100.00	100.00

Table 3.3: Hit-Rate of each QF for PSNR ≥ 30

PSNR	QF	PSNR	QF
2	2.15	24	78.87
4	7.41	26	80.72
6	26.37	28	82.47
8	41.43	30	84.00
10	51.33	40	89.92
12	58.42	50	93.89
14	63.69	60	96.98
16	67.95	70	99.63
18	71.42	80	100.00
20	74.11	90	100.00
22	76.71	100.00	100.00

Table 3.4: Hit-Rate of each QF for PSNR ≥ 32

PSNR	QF	PSNR	QF
2	0.46	24	59.38
4	1.32	26	61.79
6	7.55	28	64.34
8	17.67	30	66.46
10	26.46	40	74.86
12	33.84	50	81.43
14	39.91	60	89.68
16	45.08	70	95.00
18	49.48	80	99.94
20	53.11	90	100.00
22	56.44	100.00	100.00

3.6 Training Strategies

3.6.1 Training Methodology

We trained our proposed Deep Selector-JPEG with its two forms using stochastic gradient descent via TensorFlow machine learning platform. Our Deep Selector-JPEG used multi-GPU training via two NVIDIA GeForce RTX 2080 Ti GPUs that evenly splits a training batch size of 100 for 2 epochs, where each epoch consists of 20000 steps. In training, we chose S to be the publicly available IV3 and MobileNet V2 pretrained on ImageNet classification. IV3 is a representative for heavy-weighted DNN architectures and MobileNet-V2 is a representative for light-weighted ones. Precision constants from 0.2 to 0.7 and decision thresholds from 0.5 to 0.9 were tried.

For the Form One DNN architecture of Deep Selector-JPEG, we produced two selectors, namely *IV32L* and *M2L*, with the underlying DNN S to be IV3 and MobileNet V2, respectively. In each of these selectors, we froze all layers except for the last two blocks, which we progressively trained for 2 epochs. For each selector and each y_j, we initialized the learning rate to 0.001 to train the last logits layer with random weights while freezing all its previous layer. Using the same learning rate, we unfreeze the previous layer and initialize it with pre-trained weights while using the weights of the previously trained layer.

For the Form two DNN architecture of Deep Selector-JPEG, we choose S to be MobileNetV2 for each y_j and initialize it with its pre-trained weights except for the last logits block that's initialized with random weights. We call this selector *MFull*, where training each y_j is done independently for 2 epochs with a learning rate set to 0.01. In both forms of Selector-JPEG, Inception V3 and MobileNet V2 model evaluations are performed using a running average of the parameters computed over time.

3.6.2 Hyper-parameter tuning

We tune the precision constants (pr_j) and the decision threshold (DT_j) introduced in Equation (3.2) and (3.5) for each y_j to achieve a trade-off between accuracy and compression ratio. As discussed in section 3.4, lower pr_j and/or higher DT_j implies higher precision and lower recall for h_j.

To improve the accuracy of S at a target CR (CR^*), we tune pr_j and DT_j and then select the least QF that is ON for each x_i from an empirically obtained set of QFs, $G \subseteq L$. To define G for CR^*, let QF_k be the QF inside L that has the closest CR $\leq CR^*$. G is empirically derived as $G = \{QF \in L |\ QF \leq QF_{k+1}\}$. It is worth noting that selectors for lower j need higher precision because a false positive at lower QFs causes more accuracy drops on a DNN than higher ones. As a result, we choose low pr_j and high DT_j for $\forall_{QF \in G} : QF_j < QF_k$ to ensure high precision for these selectors. For an input image, we select the least QF that is ON from G, but if none of the QFs inside G is selected, we finally select QF_{k+2}.

We present the tuned hyper-parameters for two target CRs for PSNR $\geq$ 26 dB to illustrate this process. For target CR=3x corresponding QF_k is 70 since compressing all images on validation set of ImgeNet at QF=70 results in average CR=2.89x. Therefore, the emperically derived set of QFs is $G = \{10, 20, 30, 40, 50, 60, 70, 80\}$. The tuned parameters for each QF_j inside G is shown in table 3.5. For QFs less than 70, the parameters are tuned to achieve high precision, while for $QF_j = 70$, $QF_j = 80$ parameters are tuned to have a higher recall. The resulting precision and recall percentage of these chosen parameters for each binary classification for each of M2L, MFull, and IV3-2L are presented in table 3.6. It can be seen from the table that as QF_j increases, recall increases, and precision decreases. In the test time, for each image, Deep Selector-JPEG selects the least QF that is ON among QFs inside $G =$. If none of QFs inside G are ON Deep Selector-JPEG selects QF90. The distribution of selected QFs for all QFs inside G is shown in Figure 3.5 (a) and confirms the high recall at QF90 and QF80 as well as low recall, thus high precision at QFs less than 70.

To acheive higher target CRs, there is no need to include high QFs that result in low CR in the selection. For example for target CR=5.5x, QF_k is equal to 30 and corresponding G is $G = \{10, 20, 30, 40\}$. Tuned hyper-parameters are shown in table 3.7. The resulting precision and recall percentage for these parameters for M2L, MFull, and IV3-2L are shown in table 3.8. All QF_j have relatively high precision to ensure correct prediction of turned ON QFs. Although the precision constant and decision threshold for all QF_j is almost similar, recall increase as QF_j increase since typically more number of images are turned ON for higher QFs. The distribution of selected QFs is shown in Figure 3.5 (b).

Table 3.5: Hyperparameters at CR=3x and PSNR $\geq 26 \implies QF_s = 10 : QF_k = 70$, $G = \{10, 20, 30, 40, 50, 60, 70, 80\}$; If nothing inside G is ON, QF=90 is selected.

QF	**M2L**		**MFull**		**IV32L**	
	DT	pr	DT	pr	DT	DT
10	0.9	0.4	0.9	0.2	0.9	0.2
20	0.9	0.4	0.9	0.3	0.9	0.2
30	0.9	0.3	0.9	0.3	0.9	0.2
40	0.9	0.3	0.9	0.4	0.9	0.2
50	0.9	0.4	0.9	0.4	0.9	0.3
60	0.9	0.4	0.9	0.5	0.9	0.3
70	0.9	0.4	0.8	0.5	0.8	0.5
80	0.7	0.7	0.7	0.4	0.7	0.7

Table 3.6: Precision and Recall Percentages based on ImageNet validation set at CR=3x and PSNR ≥ 26.

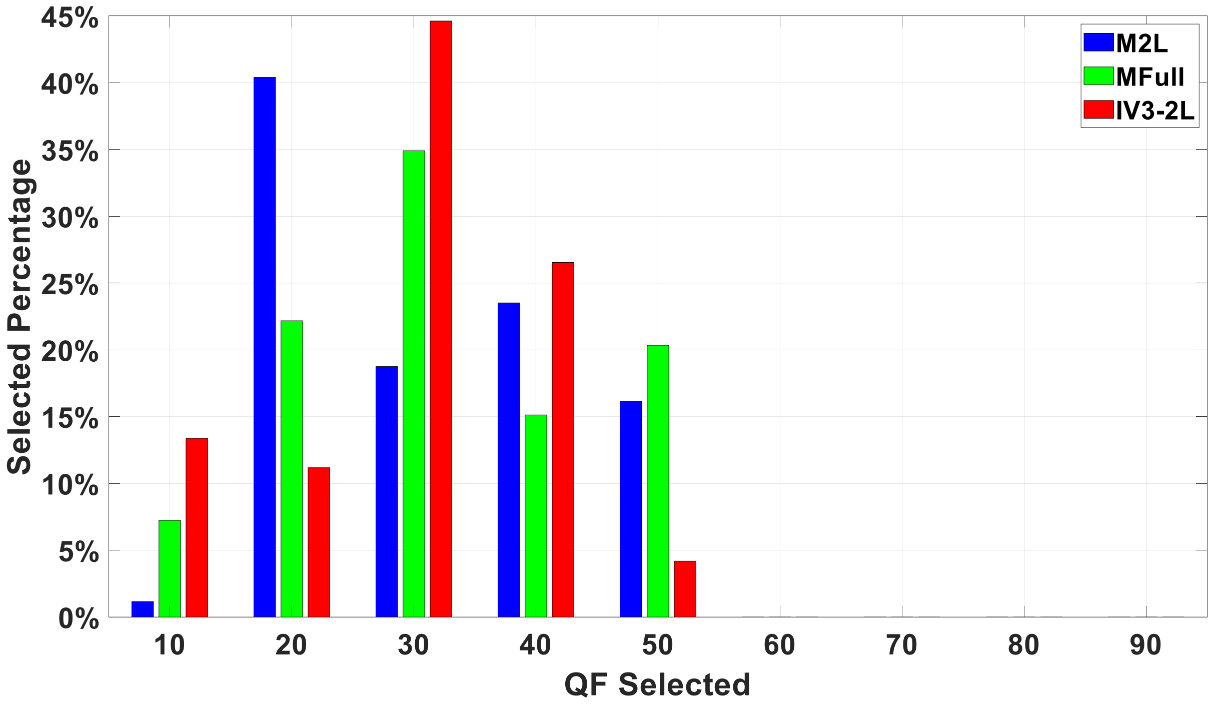
QF	**M2L**		**MFull**		**IV32L**	
	Precision	Recall	Precision	Recall	Precision	Recall
10	97.25%	1.52%	96.58%	12.50%	98.99%	16.65%
20	96.07%	11.73%	97.08%	18.52%	99.28%	26.81%
30	96.71%	10.20%	97.79%	17.94%	99.07%	30.22%
40	96.42%	14.16%	97.88%	20.75%	99.31%	29.79%
50	94.40%	52.45%	97.61%	29.88%	98.81%	41.31%
60	94.08%	62.91%	97.49%	32.50%	98.77%	44.041%
70	94.91%	60.60%	95.17%	71.59%	96.50%	81.79%
80	93.41%	97.66%	93.09%	99.77%	95.33%	98.23%

Table 3.7: Hyperparameters at CR=5x and PSNR $\geq 26 \implies QF_s = 10 : QF_k = 30, G = \{10, 20, 30, 40\}$; If nothing inside G is ON, select QF_{k+2} = 50.

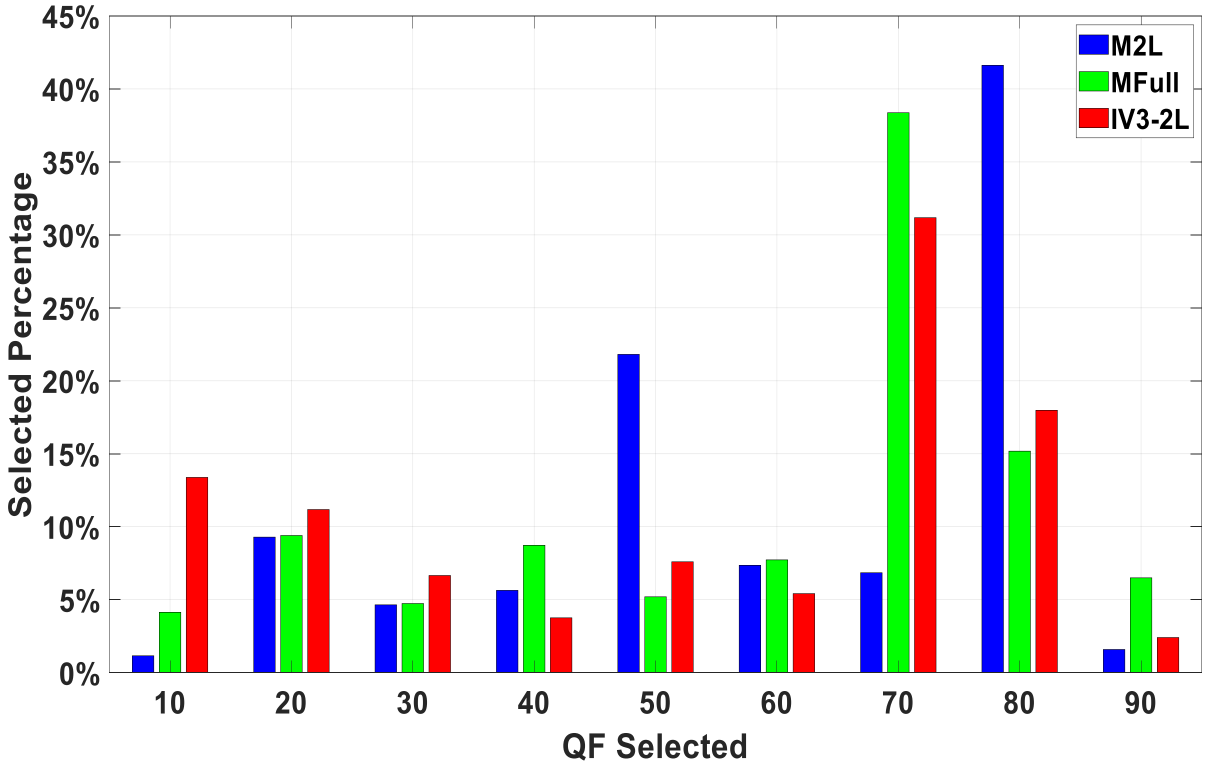
QF	**M2L**		**MFull**		**IV32L**	
	DT	pr	DT	pr	DT	DT
10	0.9	0.3	0.9	0.2	0.8	0.2
20	0.8	0.3	0.8	0.3	0.8	0.2
30	0.8	0.6	0.8	0.3	0.8	0.5
40	0.7	0.7	0.8	0.4	0.8	0.5

Table 3.8: Precision and Recall Percentages based on ImageNet validation set at CR=5x and PSNR $\geq$ 26.

QF	**M2L**		**MFull**		**IV32L**	
	Precision	Recall	Precision	Recall	Precision	Recall
10	97.25%	1.52%	97.35%	9.75%	98.11%	28.74%
20	92.81%	46.26%	95.66%	33.94%	98.36%	39.69%
30	92.75%	58.71%	92.60%	68.86%	95.58%	73.36%
40	91.12%	84.46%	92.05%	82.21%	96.078%	74.11%



(a) CR = 5x



(b) CR = 3x

Figure 3.5: Distribution of Selected QFs based on ImageNet validation set for CR = 5x (Top) and CR=3x (Bottom) at PSNR $\geq$ 26.

3.7 Summary

This chapter introduced Deep Selector-JPEG, an adaptive JPEG compression method tailored for both HV and CV. The architecture and consideration of CV and the consideration of HV were discussed. Our training methodology and the hyper-parameter tuning was presented, and an example of a tuned parameter with their selected distribution illustrated this process.

Chapter 4

Experimental Results

4.1 Overview

In this chapter, we demonstrate the effective ability of the proposed Deep Selector-JPEG to achieve a better Rate Accuracy (RA) trade-off than that of the default JPEG compression while considering human vision. The generality of the proposed method on different Deep Neural Networks (DNNs) is demonstrated by applying the selector on the ten most popular DNNs in the literature. Our experiment setting for training and evaluating Deep Selector-JPEG for four different PSNR constraints is presented in 4.2. Section 4.3 presents the experimental results of applying Deep Selector-JPEG on the tested DNNs for target PSNRs. Section 4.4, discusses the complexity of our design. A demo of applying Deep Selector-JPEG on a specific input image example is presented in section 4.5. Finally, section 4.6 summarize this chapter.

4.2 Experiment Setup

This section demonstrates our experimental set up for training two forms of Deep Selector-JPEG and evaluating its performance for the ten most popular DNNs in the literature.

4.2.1 Training Experiments Pipeline

This subsection reviews our training procedure and pipeline for the two forms of Deep Selector-JPEG. We utilize the ImageNet training dataset to conduct all of our training experiments. Each

Binary Relevance (BR) problem introduced in 3.4 is trained independently for each QF inside the feasible set of QFs as a binary classification problem. Our training pipeline for two forms of the selector is shown in Figure 4.1. The training pipeline's first step is to determine the" ON/OFF" labels for each BR for all images in the training dataset. We use MobileNetV2 to derive ON/OFF labels for training M2L and MFull, the selector form one and two based on MobileNetV2 architecture. Moreover, Inception V3 (IV3) is used for deriving ON/OFF labels for training IV3-2L, the selector form one based on IV3 architecture. All of the training dataset images are first compressed for each QF inside $L = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$ to create nine compressed clusters. We run inference on MobileNetV2 and IV3 for each cluster and label the original image based on Equation 3.1.

In the selector form one, we use a large portion of the MobileNetV2 and IV3 as a common feature extractor for all BR problems and train the last two layers of these architectures to create M2L and IV32L selectors, respectively. The training process for this form is shown in Figure 4.1 right branch. Training the last two layers is done in two runs progressively. In the first run, we train the last logits block (Fully Connected (FC) layer) of each architecture, initialized with random weights. For each BR problem, FC layer training is done for different precision constants (pr) from 0.2 to 0.7 with a learning rate of 0.001 for two epochs. In the second run, for each precision constant, we add an extra block to the training process and load the trained weights from the first training run to initialize the FC layer. The last two layers are trained together with a learning rate of 0.001 for two epochs. In both runs, all layers before the trained layers are kept constant. There are two possible methods to keep part of a DNN constant while training the rest. The first method is to freeze the constant portion, run inference on each batch of training data through the frozen parts at each training step, and train the unfrozen layers. This method adds the overhead of inferencing time through frozen layers at each step of training.

On the other hand, it is possible to run inference on frozen layers once and store the outputs of the last layer of the constant part, called the bottleneck layer. These stored bottleneck layers are then used as the input to the training process. We use the second method to speed up the training process during training selector form one. During each training run, the bottleneck layers for IV3-2L and MobileNetV2 are first stored. To store them, we run inference on IV3 and MobileNetV2 for a certain number of images and store their corresponding bottleneck layers at each training run. During training, stored bottleneck layers are loaded and fed to the training process as the input. Each of these bottleneck layers has numerous parameters and requires a huge storage space. We use the TRrecord file format introduced by the TensorFlow organization to store these bottlenecks.

TFRecord is a binary file format that serializes data and speeds up the reading process significantly. This is beneficial during the training process, specifically when we want to minimize the waiting time caused by reading data from memory. This file format is also compatible with batch training as it provides the ability to load only the required portion of the data. Therefore, there is no need to load the entire dataset to memory, and only the batch of data used for the current training step will be loaded that saves up Central Processing Unit (CPU) memory. Moreover, storing in binary format using TFRecords reduces the required storage space and makes the data portable. Saving the layer" expanded-Conv-16" as the bottleneck layer of MobileNetV2 for 100 images as a NumPy file takes 34Mb space on disk while saving the same bottleneck layer of 125 images through the TFRecord format takes 17Mb. The last two layers of each architecture are trained for each BR. At last, the hyperparameters are tuned based on the process discussed in 3.6.2 to meet the PSNR and CR constraints.

In the second form of the selector, using ON/OFF labels from MobileNetV2, we train all layers of the MobileNetV2 architecture for each BR problem to predict ON/OFF labels of each QF within a set of feasible QFs for each input image. We call this selector MFull. The training process for this form of the selector is shown in Figure 4.1 left branch. We first generate inputs for training in TFrecord file format and load it partially for each training step. We train MFull for all precision constants from 0.2 to 0.7. For a target PSNR and CR, precision constant and decision threshold are tuned based on the procedure described in 3.6.2.

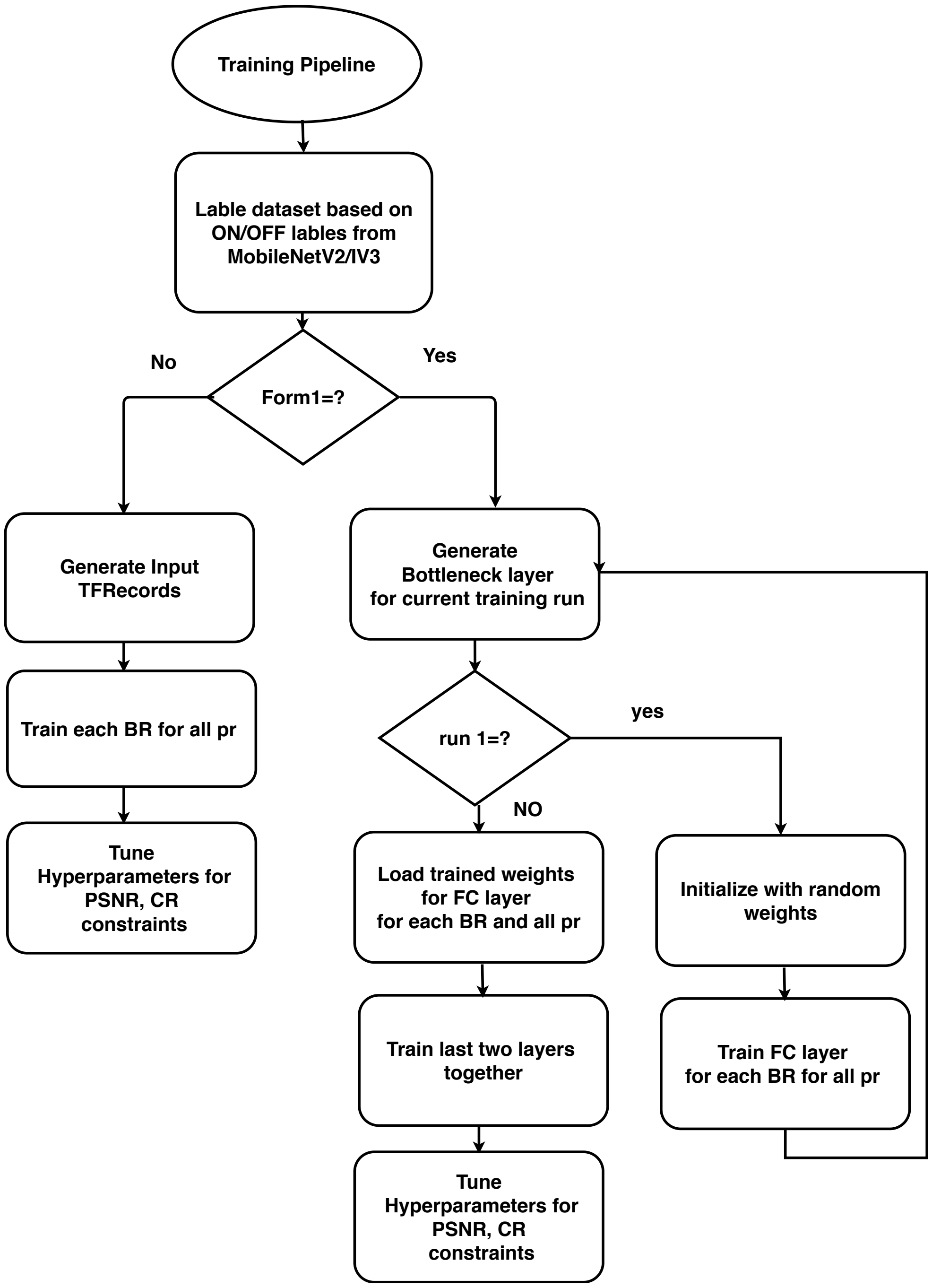

Training Pipeline
Lable dataset based on ON/OFF lables from MobileNetV2/IV3
No
Yes
Form1=?
Generate Input TFRecords
Train each BR for all pr
Tune Hyperparameters for PSNR, CR constraints
Generate Bottleneck layer for current training run
run 1=?
yes
NO
Load trained weights for FC layer for each BR and all pr
Initialize with random weights
Train last two layers together
Train FC layer for each BR for all pr
Tune Hyperparameters for PSNR, CR constraints

Figure 4.1: Training Pipeline for two forms of the selector

4.2.2 Efficient Implementation of the Training Procedure

This subsection reviews some methods for implementing training procedures efficiently. We implement our training on Graphic Processing Units (GPUs) to speed up the training process. GPUs are comprised of hundreds of cores and can handle hundreds of threads simultaneously. The GPU is ideal for processing graphical data and video rendering, where different tasks need to be performed at the same time. Training and inferencing neural networks involve many complex computations, such as big matrix multiplication, that must be done simultaneously to provide efficient training and inference time. Conduction these computations on the CPU, on one single thread, would be very slow. However, GPUs can provide parallel computing and make it possible to perform different matrix multiplications tasks simultaneously, resulting in a significant improvement in training and inferencing speed. For further speed improvement, we utilized multi-GPU processing. Two NVIDIA GeForce RTX 2080 Ti GPUs are used. Each GPU contains a replica of the weights of the Deep Neural Network (DNN) model. A batch of the input data is divided into two groups; each group goes through one GPU during the forward pass. The gradient of the loss with respect to the DNN parameters for each group is calculated on each GPU; at the end of one forward pass, these two sets of gradients are averaged on the CPU to derive the final gradient of the input batch. The final gradient is used in the backward pass, and all wights on both GPUs are updated accordingly. [6]. This process is shown in figure 4.2.

As TFRecord provides the ability to load data partially, the loading and preprocessing of data for training is done on the CPU. Typically, data preparation and training are done sequentially, which means the GPUs should wait for the CPU to load and process data; thus, the CPU becomes the bottleneck for training time. We configure a pipeline to allocate the tasks of training and data preprocessing simultaneously. While training is performed on GPUs for step k, CPU prepares the data for step $k+1$, which provides parallel processing in task allocation. [32]. To further decrease the waiting time for GPUs, we use multi-threaded queues in TensorFlow. The GPUs take the inputs from a queue, and meanwhile, they do the training process, the new inputs would be inserted into the queue. Ensuring that the queue is never empty decreases GPUs' idle time to wait for inputs. Multiple threads prepare the training examples and enqueue them, while a training thread dequeue batches of training data from the queue. [18]. A process called Queue Runner implemented by Tensorflow is used to properly handle the queues to start and stop simultaneously and capture errors. [32]

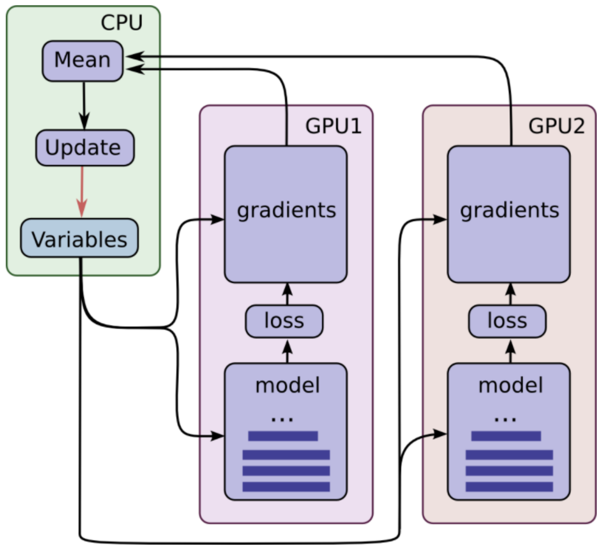


Figure 4.2: Multi-GPU Utilization[6]

4.2.3 Performance evaluation of Deep Selector-JPEG

After the training is finished, we use the validation set of the ImageNet dataset to evaluate Deep Selector-JPEG performance for different test DNNs. Each image is passed through IV32L, M2L, and MFull to select the proper QF. The compressed image using this selected QF is fed to each test DNN to derive the final classification performance. We compare Rate Accuracy (RA) performance of Deep Selector-JPEG with default JPEG for four PSNR constraints. The details of these experiments are discussed in the following paragraphs.

To create the RA curve of default JPEG for a test DNN, we first create nine compressed data clusters. To create each cluster, a constant QF inside $L = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$ is used to compress all images in the validation set of ImageNet dataset. Each cluster is passed through a test DNN, corresponding classification accuracy and average Compression Ratio (CR) over all images in the cluster is recorded. The resulting accuracy and CR for all clusters are plotted to create RA curves of default JPEG.

We create four sets of RA curves for four PSNR constraints to evaluate the performance of Deep Selector-JPEG for both Human Vision (HV) and Computer Vision (CV). These PSNR constraints are PSNR $\geq$ 26dB, PSNR $\geq$ 28dB, PSNR $\geq$ 30dB, PSNR $\geq$ 32dB. They correspond to the four sets of feasible QFs, $L_1 = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$, $L_2 = \{20, 30, 40, 50, 60, 70, 80, 90\}$, $L_3 = \{40, 50, 60, 70, 80, 90\}$ and $L_4 = \{60, 70, 80, 90\}$.

The union of these four sets is L_1, which means there is no need to train new QFs for each PSNR constraint. We train the nine QFs in L_1 once, and limit the selection to the other target PSNRs corresponding feasible sets to meet their constraint. For each target PSNR, we use Deep Selector-JPEG to adaptively select the proper QF among its corresponding feasible set for each input image such that the average CR of all images in the dataset reaches a target CR value. The compressed dataset using the selected QFs for each CR passes through the test DNNs to derive the classification accuracy. The resulting CRs and classification accuracies are plotted to achieve RA curves.

We use IV3 and MobileNetV2 to derive ON/OFF labels for our training set. Moreover, their architectures are used as a base for training the two forms of Deep Selector-JPEG. We apply the QF selections of IV32L, M2L, and MFull at each PSNR and CR constraint to the ten most popular DNN classifiers in the literature [2]. The set of test DNNs includes, IV1, IV4, IV3, ResNet-50, ResNet-101, Inception-ResnetV2, MobileNetV1, MobileNetV2, nasnet-mobile and Pnasnet-Large.

In order to quantify gains of the selector, for each PSNR constraint, we present the accuracy gain at the highest and lowest achievable CRs within the set of valid QFs for the PSNR for M2L, MFull, and IV3-2L on all test DNNs in Tables 4.1, 4.2, 4.3, 4.4. Numbers between parenthesis indicate Top-1 and Top-5 percentage gains, respectively, with respect to JPEG over DNNs under test. An average accuracy gain over all DNNs at target CR is shown in the last row of tables.

The results for each PSNR constraint is discussed in the following section.

4.3 Experimental Results and Discussion

This section discusses our experimental results for each PSNR constraint and compares our results with DeepNJPEG [17].

Discussion of results for target PSNR $\geq$ 26 dB:

The Top-5 and Top-1 vs CR performances of Deep Selector-JPEG for PSNR $\geq$ 26 are shown in 4.3, 4.4 respectively. The dotted black curves represent JPEG, while the red, green, and blue curves represent IV32L, MFull, and M2L, respectively. The legend of each subfigure outlines the order of test DNNs from top to bottom. For example Figure 4.3 (b) shows Top-5 performance of Deep Selector-JPEG for Pnasnet-Large, InceptionResnetV2, nasnet-mobile in the order of legend, the first set of curves on top shows results on Pnasnet Large, and the last set of curves on the bottom shows results on nasnet-mobile.

Looking at Figure 4.3 and 4.4, we can observe the consistent RA performance gains at PSNR $\geq$ 26, which can also be confirmed by Table 4.1. For instance, 0.89%, Top-5 accuracy gain can be achieved using IV32L at CR=10.05x when applied to one of the top-performing DNN classifiers, Pnasnet_Large (see Table 4.1). It is worth noting that test DNNs have different classification accuracy levels and robustness, yet the accuracy gains are consistent. For instance, accuracy gain on MobileNetV2 with 92% original Top-5 accuracy and ≈7% drop in original accuracy at high CRs is comparable to the accuracy gain of Pnsanet Large with 96% original Top-5 accuracy and ≈3% drop in accuracy at high CRs. Overall, the average accuracy gain on all architectures reaches 1% at high CRs (10.05x).

An interesting observation can be made that IV3-2L achieves a better trade-off over MFull and M2L for most test DNNs. This observation is consistent among different PSNR constraints, which reveals that using DNN with higher classification performance for ON/OFF labeling provides better RA performance. (see IV32L vs. other selectors) Due to MFull's high capacity, we can observe that MFull achieves slightly more accuracy gains than M2L.

We can also notice that the performance gain of Deep Selector-JPEG increases slightly when the selector trained via the ON/OFF labelling of the DNN, S, is applied to the same DNN classifier compared to other DNNs. M2L on MobileNet-V2 achieves 0.17%, 0.11% Top-1 and Top-5 accuracy gains at CR=3.53x respectively. At CR=10.05x, M2L attains 0.58%, 1.04% Top-1, Top-5 accuracy gains for MobileNetv2. MFull achieves slightly higher Top-1 and Top-5 accuracy gains of 0.17%, 0.15% on MobileNet-V2 at CR=3.53x . It also achieves 0.79%, 1.34% Top1, and Top-5 accuracy gains at CR=10.05x, respectively. Similar RA performance for IV32L on IV3 can be achieved. AT CR=3.53x both Top-1 and Top-5 gains reach ≈0.1%. On the other hand, Top-1 and Top-5 accuracy gains of 0.84%, 1.28% are achived at CR=10.05x. Table 4.1, green cells point out the accuracy gains of M2L, MFull on MobileNetV2 as well as accuracy gains of IV3-2L on IV3.

Discussion of results for target PSNR $\geq$ 28 dB:

For target PSNR $\geq$ 28, the set of feasible QFs is $L = \{20, 30, 40, 50, 60, 70, 80, 90\}$. Limiting the range of PSNRs limits the range of achievable CRs. The highest achived CR with this setting is 6.9x (see Table 4.2) Top-5, Top-1 performances of Deep Selector-JPEG on all tested DNNs for PSNR $\geq$ 28 are shown in Figures 4.5 and 4.6 respectively. Consistent accuracy gains are achieved among different test DNNs. Table 4.2 presents accuracy gains for this PSNR. These observations show that considering PSNR constraint; Deep Selector-JPEG improves RA performance w.r.t default JPEG for different classification DNNs. The green cells of table 4.2 point out each selector's performance on the DNN classifiers, which is used for deriving ON/OFF labeling for training. Looking at these cells confirms the slight increase in accuracy for these DNNs.

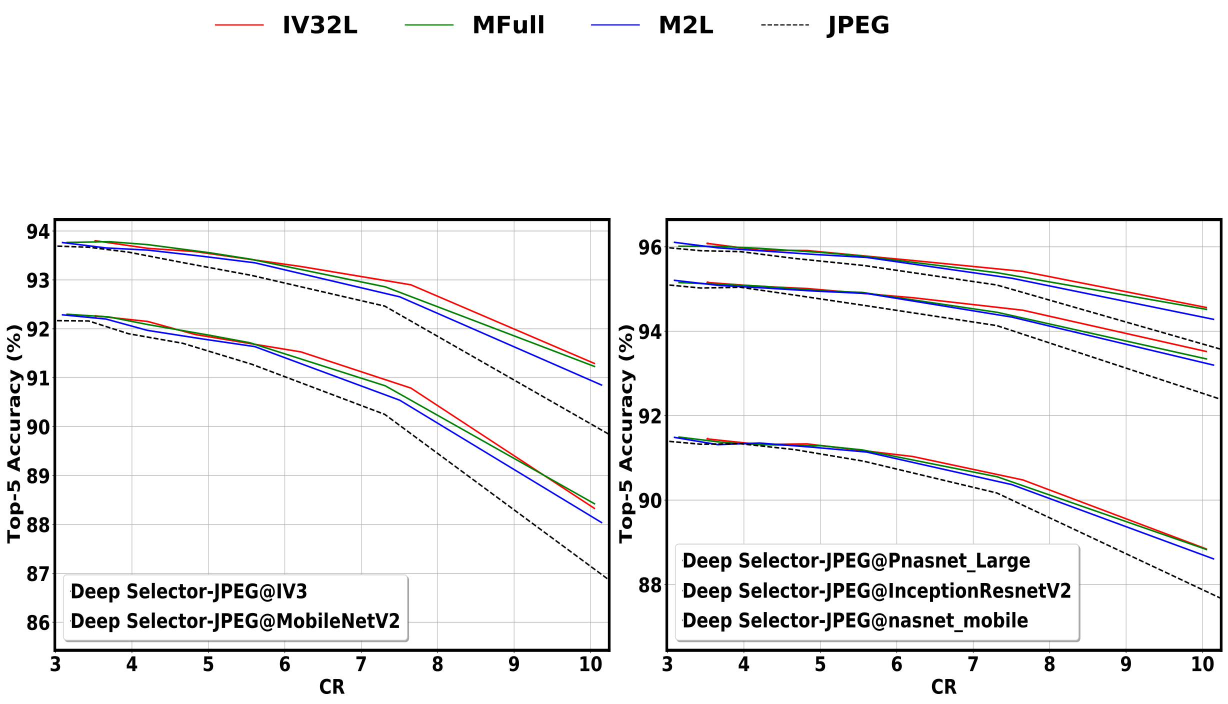
Discussion of results for target PSNR $\geq$ 30 dB , PSNR $\geq$ 32 dB:

For PSNR $\geq$ 30 and PSNR $\geq$ 32 the set of feasible QFs are $L = \{40, 50, 60, 70, 80, 90\}$, $L = \{60, 70, 80, 90\}$ respectively. These sets limit the performance of the selector. Deep Selector-

JPEG is limited to QFs of JPEG, thus it can not go beyond the original accuracy. For high PSNRs compressed images using JPEG itself have high quality and most of them are suitable for DNNs. However, gains up to 0.1% in classification accuracy are still achieved using Deep Selector-JPEG when PSNR$\geq$ 30 or $\geq$ 32. The Top-5, Top-1 performance on all tested DNNs for PSNR $\geq$ 30 are shown in Figure 4.7, 4.8. Moreover, Top5, Top1 performances on all tested DNNs for PSNR $\geq$ 32 are shown in Figure 4.9, 4.10. The accuracy gains for PSNR $\geq q$ 30 and PSNR $\geq$ 32 are shown in Tables 4.3 and 4.4 respectively. The green cells of these tables shows the accuracy gains on the DNNs used for deriving ON/OFF labelling.

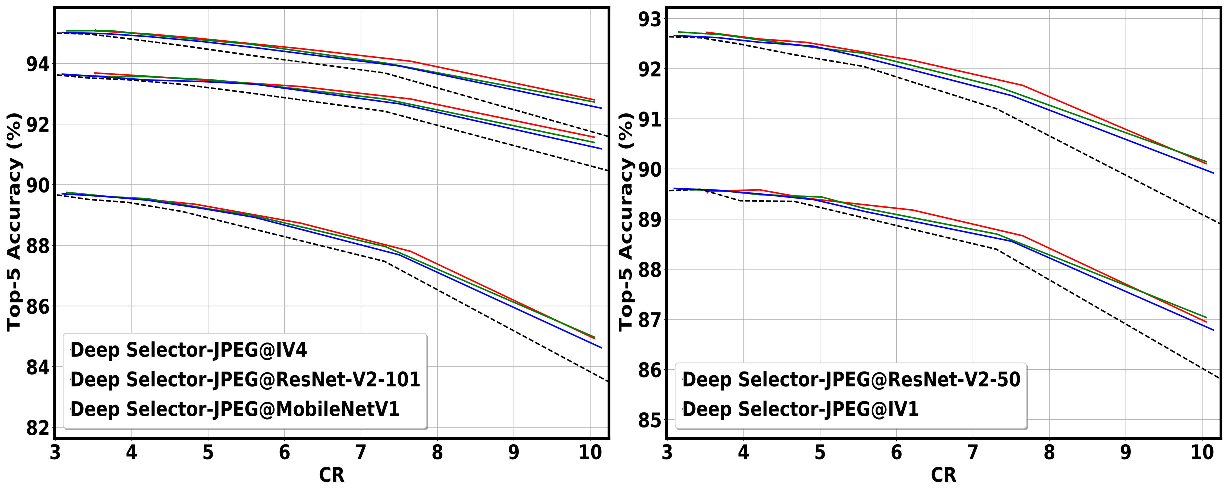
Comparison with DeepNJPEG:

We compare our results with DeepNJPEG [17] reviewed in chapter 3.2 that maintains original classification accuracy at CR 3.5x. Similarly, Deep Selector-JPEG achieves Top-5 accuracy near the original accuracy of the underlying DNN classifiers at CR=3.5x. More specifically, for IV1 and ResNet-50 that are the test DNNs in DeepN-JPEG, we achieve the same Top-5 performance as the original for PSNR $\geq$ 26. (see Figure 4.3 d). It is worth noting that Deep Selector-JPEG improves the overall RA performance of JPEG over different DNNs and achieves Top-5 accuracy near to the original accuracy while taking the human vision into account.



(a) IV3, MobileNetV2

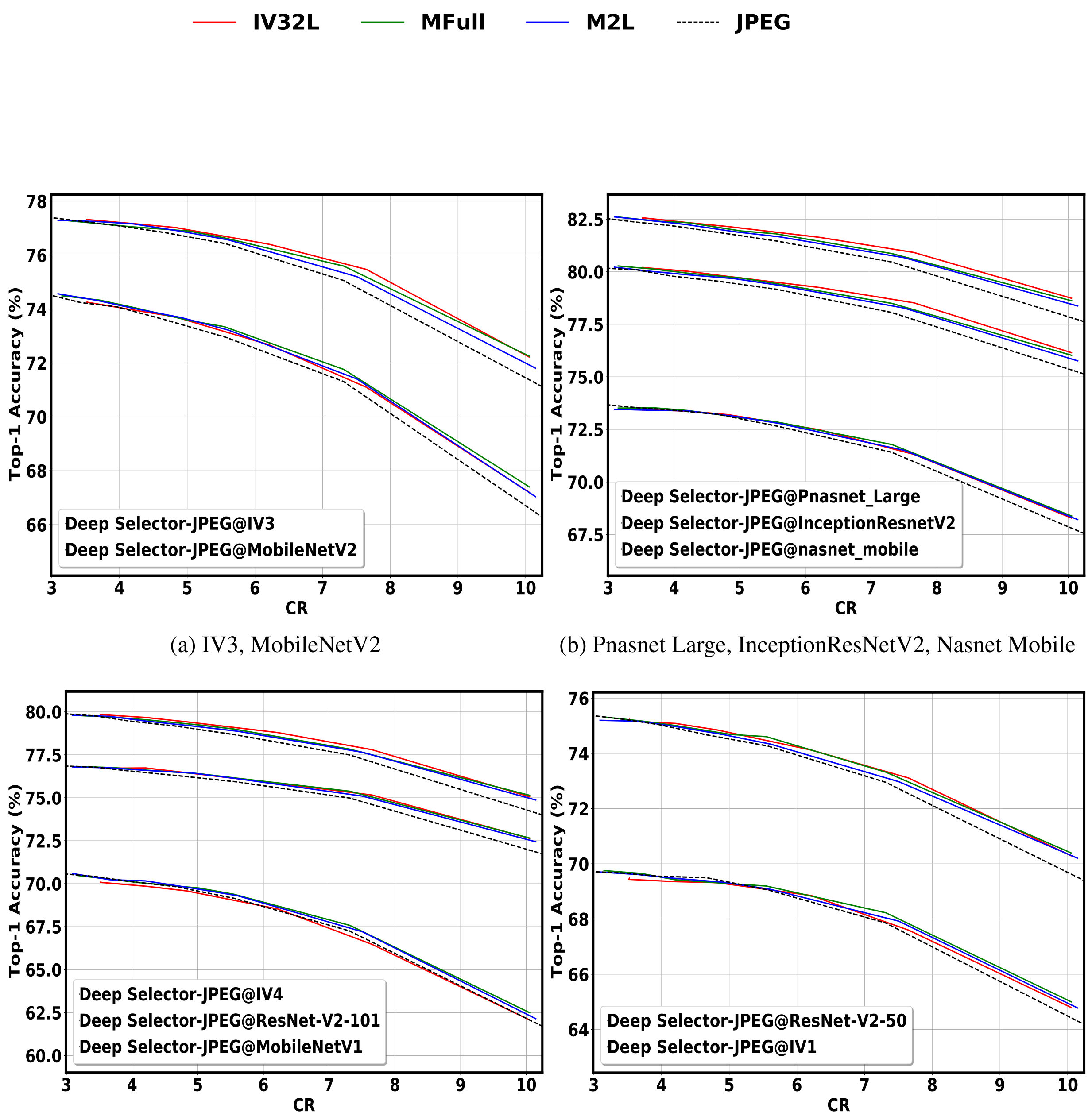
(b) Pnasnet Large, InceptionResNetV2, Nasnet Mobile



(c) IV4, ResNetV2-101, MobileNetV1

(d) ResNetV2-50, IV1

Figure 4.3: RA of Deep Selector-JPEG when applied to all DNNs under test; Top-5 accuracy, PSNR ≥ 26. (Solid: Ours, Dotted: JPEG)



(a) IV3, MobileNetV2

(b) Pnasnet Large, InceptionResNetV2, Nasnet Mobile

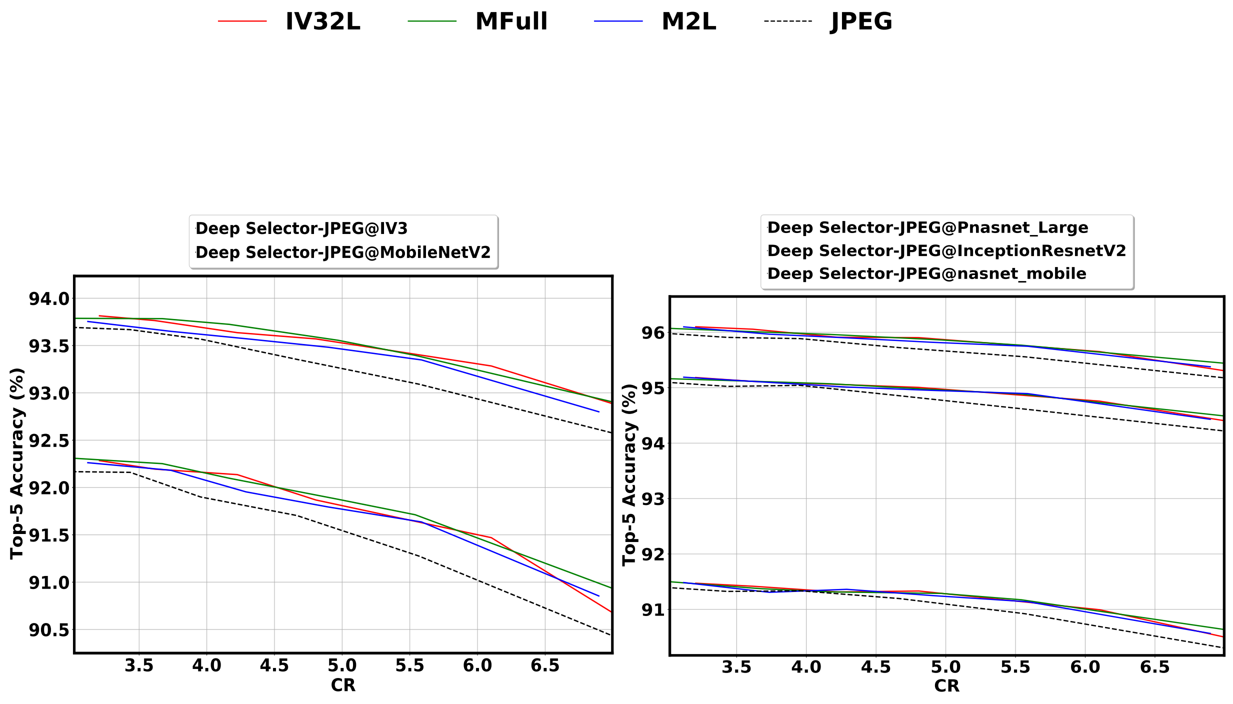
(c) IV4, ResNetV2-101, MobileNetV1

(d) ResNetV2-50, IV1

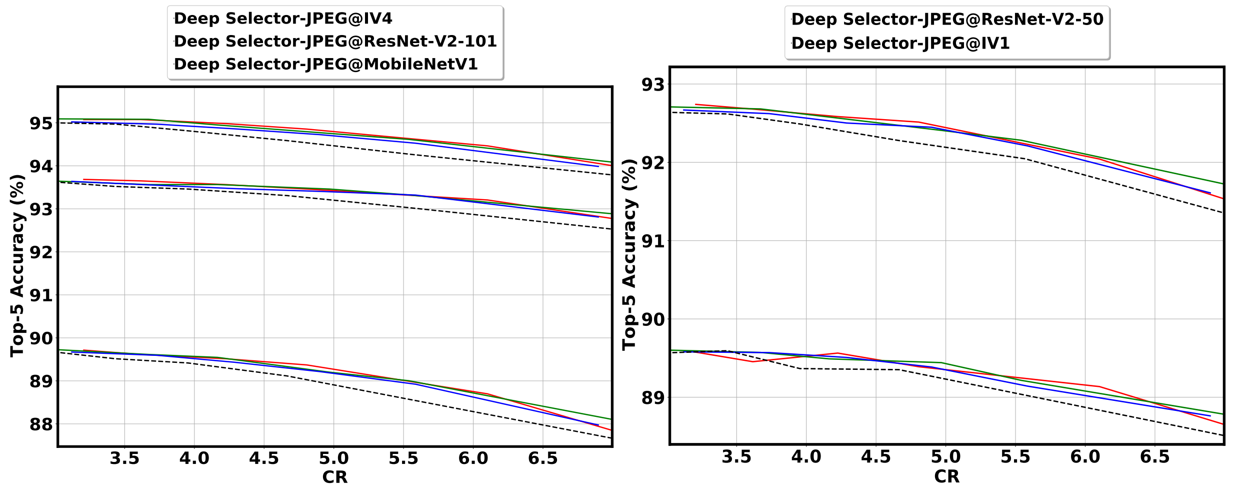
Figure 4.4: RA of Deep Selector-JPEG when applied to all DNNs under test; Top-1 accuracy; PSNR ≥ 26. (Solid: Ours, Dotted: JPEG)

Table 4.1: At CR = 10.05x and CR = 3.52x and PSNR $\geq$ 26, numbers between parenthesis indicate Top-1 and Top-5 percentage gains, respectively, with respect to JPEG over DNNs under test.

DNN	**M2L**		**MFull**		**IV32L**	
	CR=3.53x	CR=10.05x	CR=3.53x	CR=10.05x	CR=3.53x	CR=10.05x
IV3	(0.03, 0.03)	(0.55, 0.90)	(-0.02, 0.13)	(0.88, 1.22)	(0.09, 0.15)	(0.84, 1.28)
MobileNetV2	(0.17, 0.11)	(0.58, 1.04)	(0.17, 0.15)	(0.79, 1.34)	(0.04, 0.15)	(0.59, 1.24)
IV4	(0.04, 0.05)	(0.73, 0.85)	(0.01, 0.14)	(0.91, 1.00)	(0.12, 0.13)	(0.82, 1.08)
ResNet-V2-101	(0.01, 0.06)	(0.58, 0.65)	(0.03, 0.07)	(0.70, 0.81)	(-0.03, 0.17)	(0.69, 0.98)
MobileNet	(-0.05, 0.14)	(0.25, 0.97)	(-0.05, 0.15)	(0.42, 1.22)	(-0.29, 0.14)	(-0.00, 1.17)
ResNet-V2-50	(-0.03, 0.03)	(0.68, 0.92)	(0.02, 0.10)	(0.77, 1.10)	(-0.04, 0.13)	(0.67, 1.05)
IV1	(-0.01, 0.03)	(0.47, 0.87)	(0.04, 0.02)	(0.58, 1.06)	(-0.21, -0.00)	(0.37, 0.97)
Pnasnet_Large	(0.14, 0.10)	(0.64, 0.64)	(0.15, 0.11)	(0.81, 0.85)	(0.24, 0.18)	(0.93, 0.89)
InceptionResnetV2	(0.02, 0.09)	(0.52, 0.74)	(0.13, 0.10)	(0.70, 0.84)	(0.16, 0.13)	(0.82, 1.02)
nasnet_mobile	(-0.09, 0.03)	(0.52, 0.84)	(0.01, 0.08)	(0.58, 1.00)	(-0.08, 0.12)	(0.48, 1.01)
AVERAGE	(0.02, 0.07)	(0.55, 0.84)	(0.05, 0.10)	(0.71, 1.04)	(0.00, 0.13)	(0.62, 1.07)

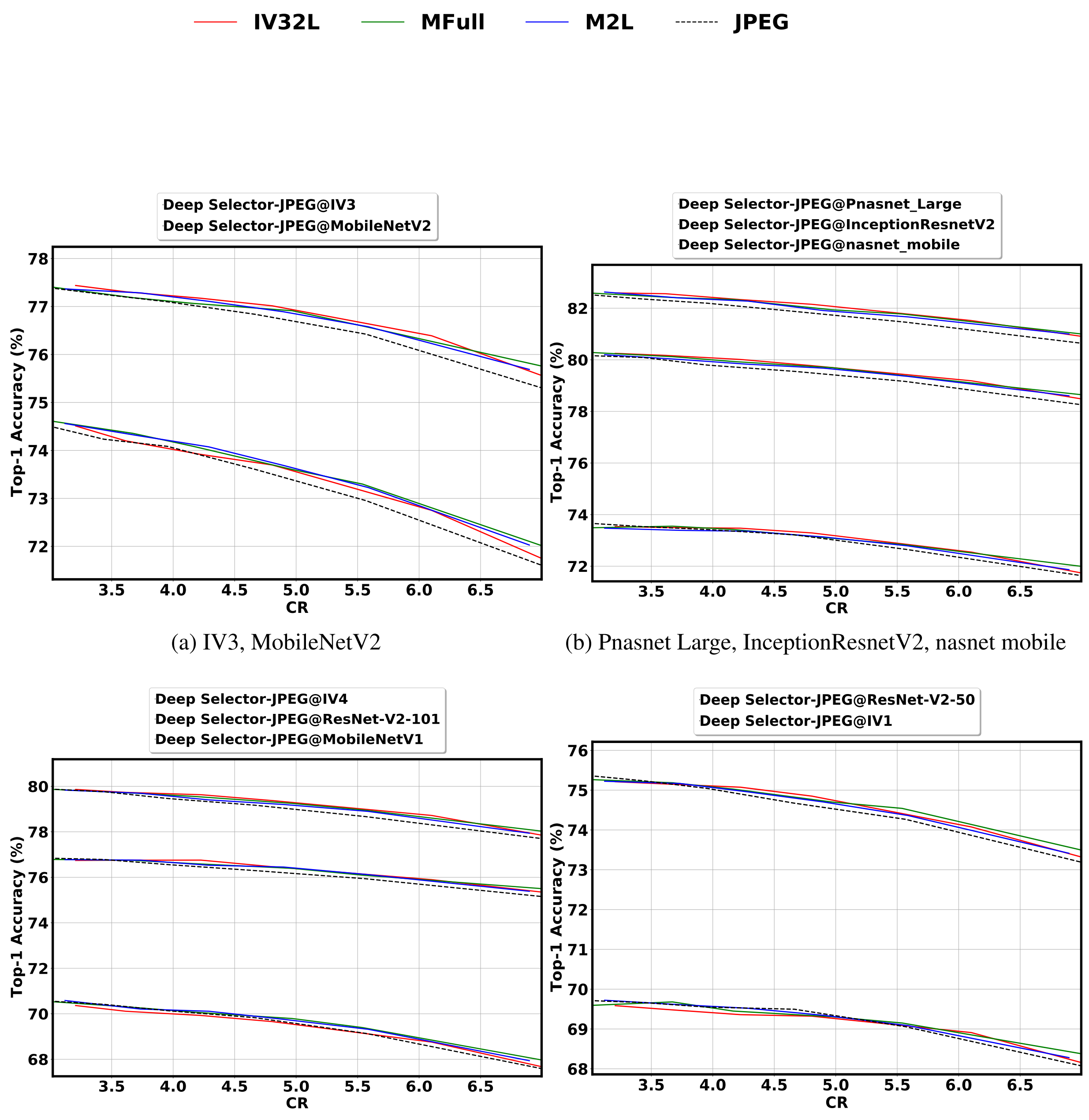


(a) IV3, MobileNetV2 (b) Pnasnet Large, InceptionResnetV2, nasnet mobile



(c) IV4, ResNetV2-101, MobileNetV1 (d) ResNetV2-50, IV1

Figure 4.5: RA of Deep Selector-JPEG when applied to all DNNs under test; Top-5 accuracy; PSNR ≥ 28. (Solid: Ours, Dotted: JPEG)



(a) IV3, MobileNetV2

(b) Pnasnet Large, InceptionResnetV2, nasnet mobile

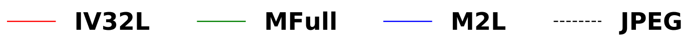
(c) IV4, ResNetV2-101, MobileNetV1

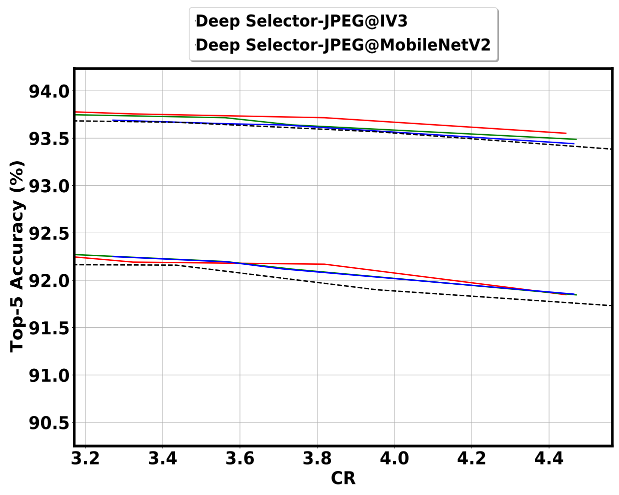
(d) ResNetV2-50, IV1

Figure 4.6: RA of Deep Selector-JPEG when applied to all DNNs under test; Top-1 accuracy; PSNR ≥ 28. (Solid: Ours, Dotted: JPEG)

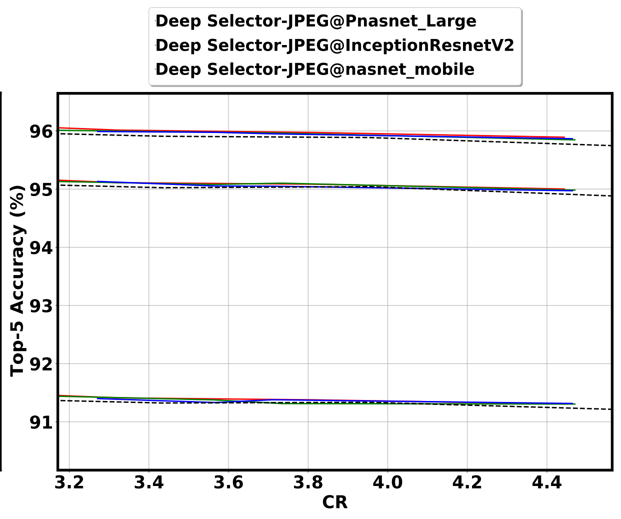
Table 4.2: At CR = 6.9x and CR = 3.21x and PSNR $\geq$ 28, numbers between parenthesis indicate Top-1 and Top-5 percentage gains, respectively, with respect to JPEG over DNNs under test.

DNN	**M2L**		**MFull**		**IV32L**	
	CR=3.21x	CR=6.9x	CR=3.21x	CR=6.9x	CR=3.21x	CR=6.9x
IV3	(0.03, 0.06)	(0.31, 0.19)	(0.02, 0.11)	(0.43, 0.33)	(0.11, 0.13)	(0.27, 0.32)
MobileNetV2	(0.15, 0.09)	(0.33, 0.36)	(0.16, 0.13)	(0.40, 0.50)	(0.14, 0.12)	(0.16, 0.27)
IV4	(-0.01, 0.03)	(0.17, 0.16)	(-0.01, 0.10)	(0.32, 0.30)	(0.04, 0.09)	(0.18, 0.23)
ResNet-V2-101	(-0.04, 0.05)	(0.18, 0.25)	(-0.04, 0.05)	(0.33, 0.35)	(-0.07, 0.11)	(0.20, 0.26)
MobileNet	(0.04, 0.06)	(0.24, 0.25)	(-0.03, 0.10)	(0.37, 0.44)	(-0.13, 0.12)	(0.10, 0.22)
ResNet-V2-50	(-0.09, 0.03)	(0.15, 0.21)	(-0.06, 0.07)	(0.30, 0.36)	(-0.09, 0.11)	(0.14, 0.19)
IV1	(0.02, -0.00)	(0.14, 0.22)	(-0.07, 0.01)	(0.29, 0.27)	(-0.10, -0.00)	(0.10, 0.16)
Pnasnet_Large	(0.15, 0.13)	(0.30, 0.17)	(0.09, 0.11)	(0.35, 0.26)	(0.15, 0.15)	(0.28, 0.14)
InceptionResnetV2	(0.04, 0.11)	(0.28, 0.18)	(0.11, 0.09)	(0.38, 0.27)	(0.12, 0.12)	(0.24, 0.20)
nasnet_mobile	(-0.14, 0.10)	(0.16, 0.22)	(-0.09, 0.10)	(0.35, 0.33)	(-0.06, 0.11)	(0.12, 0.21)
AVERAGE	(0.02, 0.07)	(0.22, 0.22)	(0.01, 0.09)	(0.35, 0.34)	(0.01, 0.11)	(0.18, 0.22)

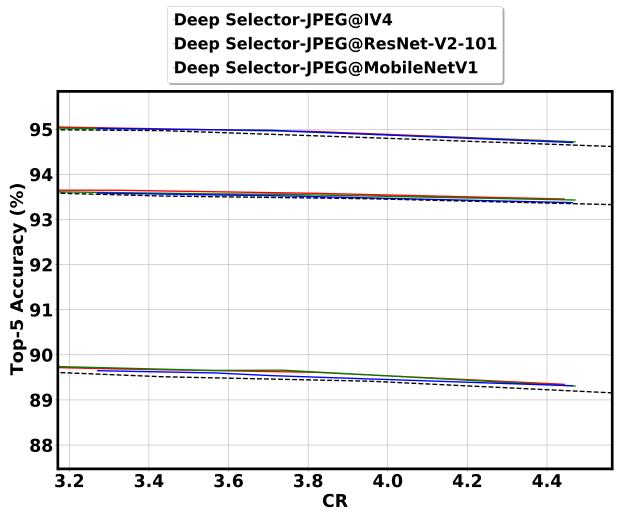




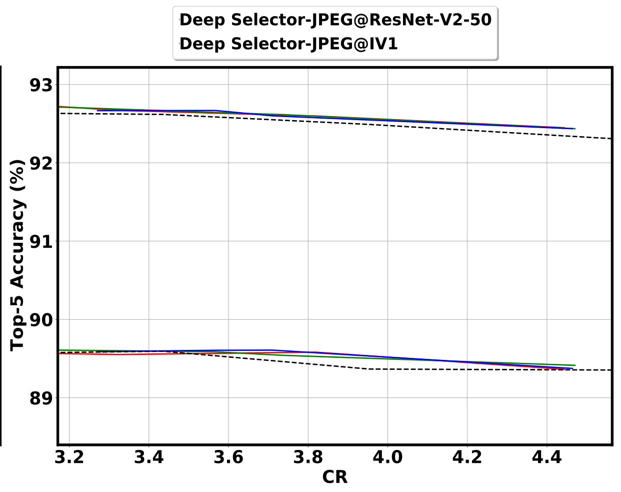
(a) IV3, MobileNetV2



(b) Pnasnet Large, InceptionResnetV2, nasnet mobile

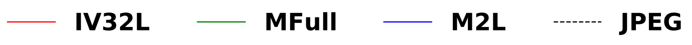


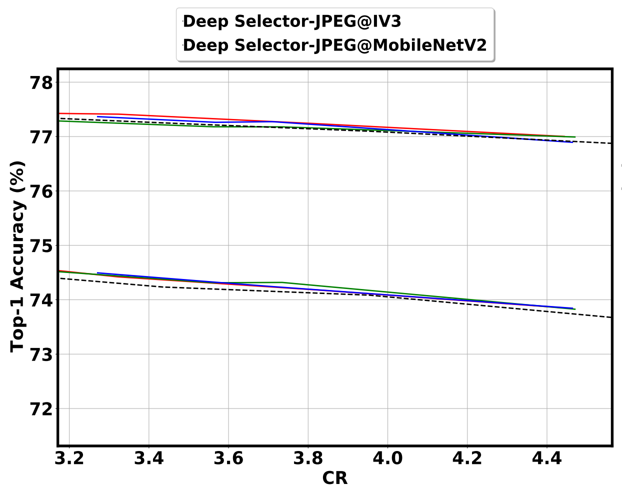
(c) IV4, ResNetV2-101, MobileNetV1



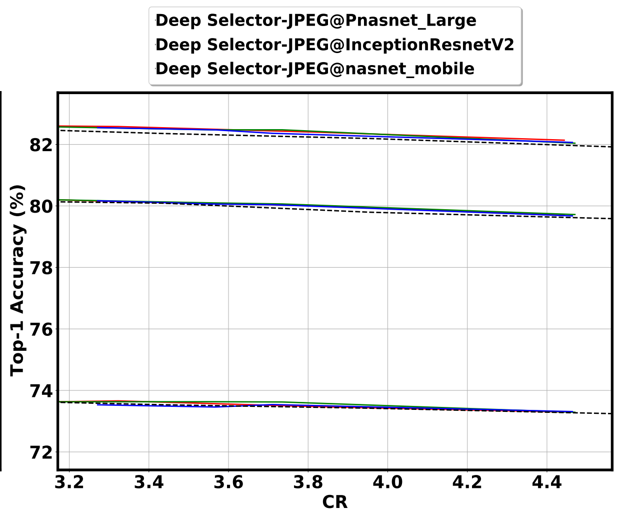
(d) ResNetV2-50, IV1

Figure 4.7: RA performance of Selector-JPEG vs JPEG on unknown DNN classifiers; Top5-accuracy; PSNR $\geq$ 30.

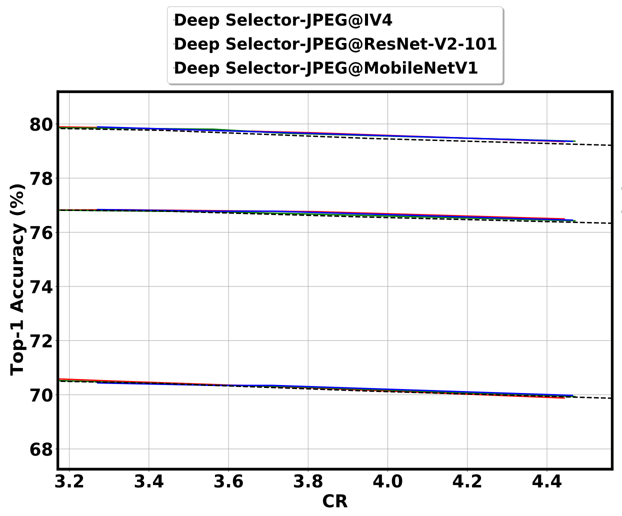




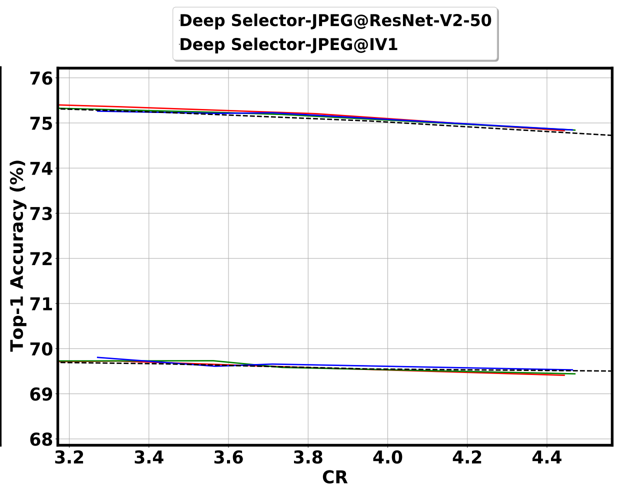
(a) IV3, MobileNetV2



(b) Pnasnet Large, InceptionResnetV2, nasnet mobile



(c) IV4, ResNetV2-101, MobileNetV1

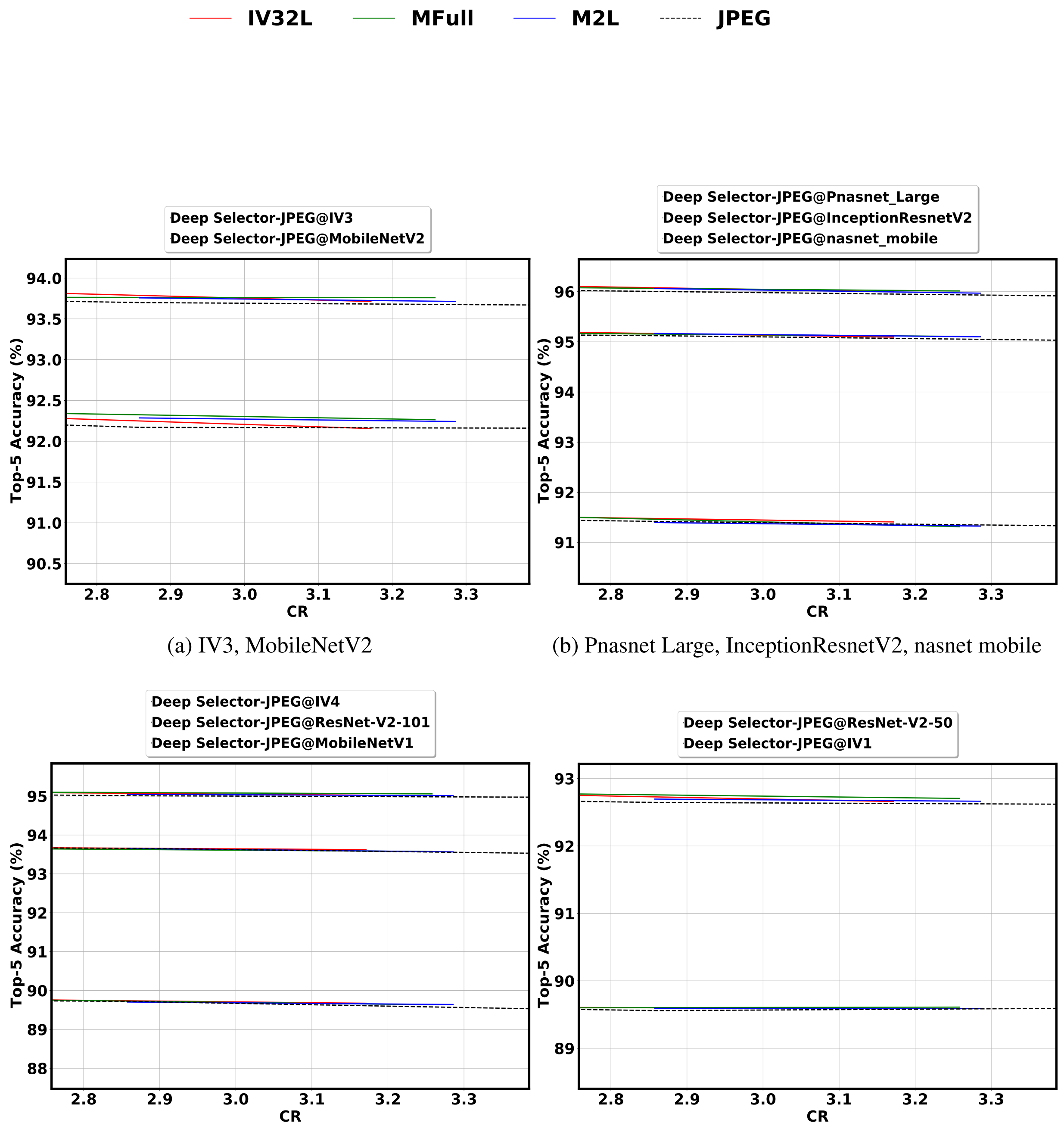


(d) ResNetV2-50, IV1

Figure 4.8: RA of Deep Selector-JPEG when applied to all DNNs under test; Top-1 accuracy; PSNR $\geq$ 30. (Solid: Ours, Dotted: JPEG)

Table 4.3: At CR = 4.44x and CR = 3.27x and PSNR $\geq$ 30, numbers between parenthesis indicate Top-1 and Top-5 percentage gains, respectively, with respect to JPEG over DNNs under test.

DNN	**M2L**		**MFull**		**IV32L**	
	CR=3.27x	CR=4.44x	CR=3.27x	CR=4.44x	CR=3.27x	CR=4.44x
IV3	(0.06, 0.01)	(-0.01, 0.03)	(-0.04, 0.06)	(0.08, 0.07)	(0.11, 0.09)	(0.08, 0.13)
MobileNetV2	(0.16, 0.09)	(0.10, 0.10)	(0.13, 0.09)	(0.09, 0.09)	(0.13, 0.05)	(0.10, 0.08)
IV4	(0.08, 0.04)	(0.10, 0.06)	(0.04, 0.03)	(0.11, 0.07)	(0.06, 0.05)	(0.09, 0.07)
ResNet-V2-101	(0.03, 0.03)	(0.08, 0.02)	(-0.00, 0.04)	(0.06, 0.08)	(0.02, 0.09)	(0.12, 0.10)
MobileNet	(-0.03, 0.08)	(0.06, 0.11)	(0.01, 0.15)	(0.02, 0.11)	(0.06, 0.12)	(-0.04, 0.14)
ResNet-V2-50	(-0.02, 0.04)	(0.07, 0.10)	(0.02, 0.07)	(0.07, 0.10)	(0.09, 0.06)	(0.04, 0.10)
IV1	(0.12, 0.00)	(0.02, 0.02)	(0.05, 0.02)	(-0.07, 0.06)	(0.04, -0.03)	(-0.10, 0.01)
Pnasnet_Large	(0.13, 0.05)	(0.10, 0.09)	(0.13, 0.07)	(0.08, 0.08)	(0.17, 0.09)	(0.16, 0.12)
InceptionResnetV2	(0.05, 0.08)	(0.06, 0.06)	(0.06, 0.07)	(0.10, 0.07)	(0.04, 0.07)	(0.06, 0.09)
nasnet_mobile	(-0.05, 0.05)	(0.04, 0.08)	(0.05, 0.07)	(0.02, 0.07)	(0.07, 0.08)	(0.02, 0.07)
AVERAGE	(0.05, 0.05)	(0.06, 0.07)	(0.04, 0.07)	(0.06, 0.08)	(0.08, 0.07)	(0.05, 0.09)



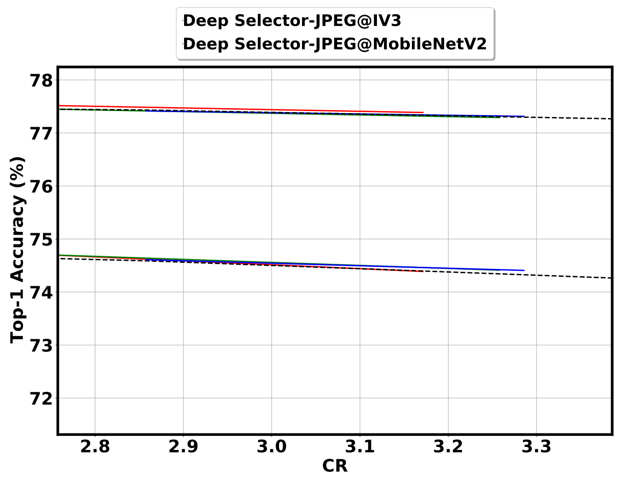
(a) IV3, MobileNetV2

(b) Pnasnet Large, InceptionResnetV2, nasnet mobile

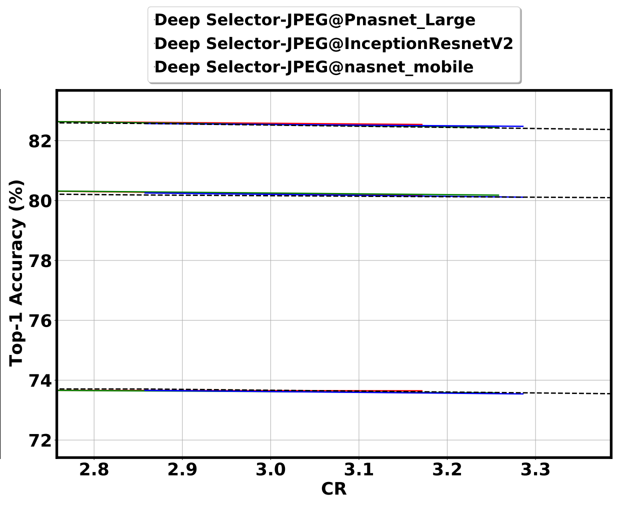
(c) IV4, ResNetV2-101, MobileNetV1

(d) ResNetV2-50, IV1

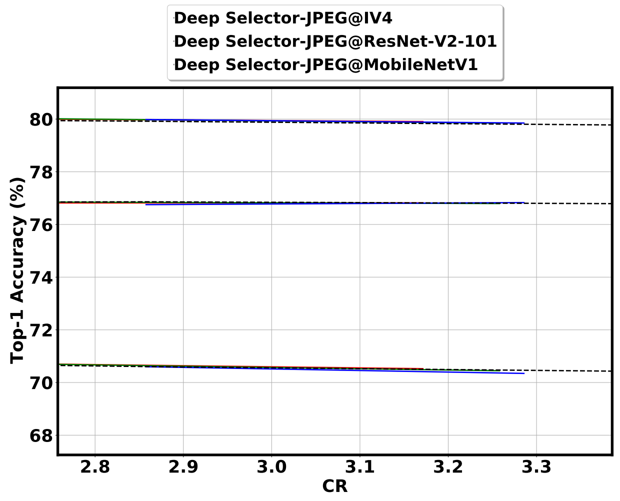
Figure 4.9: RA of Deep Selector-JPEG when applied to all DNNs under test; Top-5 accuracy; PSNR ≥ 32. (Solid: Ours, Dotted: JPEG)



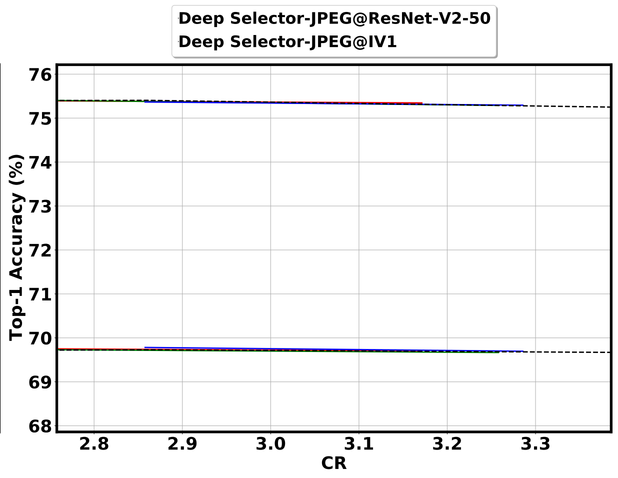
(a) IV3, MobileNetV2



(b) Pnasnet Large, InceptionResnetV2, nasnet mobile



(c) IV4, ResNetV2-101, MobileNetV1



(d) ResNetV2-50, IV1

Figure 4.10: RA of Deep Selector-JPEG when applied to all DNNs under test; Top-1 accuracy; PSNR $\geq$ 32. (Solid: Ours, Dotted: JPEG)

Table 4.4: At CR = 3.17x and CR = 2.96x and PSNR $\geq$ 32, numbers between parenthesis indicate Top-1 and Top-5 percentage gains, respectively, with respect to JPEG over DNNs under test.

DNN	M2L		MFull		IV32L	
	CR=2.96x	CR=3.17x	CR=2.96x	CR=3.17x	CR=2.96x	CR=3.17x
IV3	(-0.02, 0.06)	(0.01, 0.04)	(-0.02, 0.06)	(-0.02, 0.08)	(0.05, 0.09)	(0.05, 0.03)
MobileNetV2	(0.02, 0.12)	(0.07, 0.09)	(0.05, 0.15)	(0.07, 0.11)	(0.03, 0.08)	(-0.01, -0.01)
IV4	(0.06, 0.04)	(0.05, 0.03)	(0.06, 0.08)	(0.04, 0.08)	(0.05, 0.06)	(0.07, 0.04)
ResNet-V2-101	(-0.11, 0.00)	(-0.01, 0.01)	(-0.02, -0.03)	(-0.02, 0.00)	(-0.04, 0.00)	(-0.00, 0.04)
MobileNet	(0.00, -0.02)	(-0.09, 0.05)	(0.04, 0.01)	(-0.01, 0.05)	(0.05, 0.02)	(0.02, 0.06)
ResNet-V2-50	(-0.04, 0.05)	(0.00, 0.04)	(-0.03, 0.11)	(-0.01, 0.09)	(-0.02, 0.08)	(0.03, 0.03)
IV1	(0.05, 0.03)	(0.02, 0.01)	(-0.01, 0.04)	(-0.02, 0.03)	(0.01, 0.04)	(0.01, 0.01)
Pnasnet_Large	(0.00, 0.06)	(0.05, 0.04)	(0.02, 0.06)	(0.02, 0.07)	(0.04, 0.07)	(0.09, 0.05)
InceptionResnetV2	(0.07, 0.04)	(0.02, 0.05)	(0.10, 0.03)	(0.07, 0.05)	(0.09, 0.04)	(0.04, 0.03)
nasnet_mobile	(-0.05, -0.02)	(-0.04, -0.02)	(-0.06, 0.04)	(-0.02, -0.02)	(-0.05, 0.06)	(0.03, 0.04)
AVERAGE	(0.00, 0.04)	(0.01, 0.03)	(0.01, 0.06)	(0.01, 0.05)	(0.02, 0.05)	(0.03, 0.03)

4.4 Complexity Analysis

To analyze the complexity of Deep Selector-JPEG, we calculate the number of Multiply-Accumulate (MACs) operations for each form of Deep Selector-JPEG, which is the standard way to measure the DNN complexity. We train only the last two layers of MobileNetV2 architecture for our M2L selector for each binary relevance problem, while a large part of the architecture is frozen and used as a common feature extractor for all binary relevance problems. The complexity of trained two layers is 47488 MACS, which is not significant compared to the common feature extractor's complexity, which is 541.06 Million MACS(M MACS). Therefore, the complexity of our M2L selector is comparable with the complexity of MobileNetV2. Comparison of the complexity of M2L with MobileNetV2 architectures is presented in table 4.5. The first column shows the number of MACS for the original architectures with 1000 classes, while the second column shows the complexity of Binary MobileNetV2 that has two output classes instead of 1000 classes, and the third column shows the complexity of Deep Selector-JPEG for M2L. This table confirms that the complexity of M2L is comparable with the original MobileNetV2.

The complexity of IV3-2L is higher than the complexity of the original IV3 architecture. We trained the fully connected layer plus the mixed-10 block of IV3 for each binary relevance problem. The complexity of these two layers is 388.50 M MACS. The comparison of the complexity of IV3 original architecture with Binary IV3 that replace 1000 classes with two classes and IV3-2L is shown in table 4.7. The number of binary relevance problems varies with the PSNR and CR constraints as discussed in 3.5 and 3.6.2. Therefore, for different applications with their PSNR and CR constraints, Deep Selector-JPEG provides different complexity levels. Table 4.6 and 4.8, show the overall complexity of M2L and IV3-2L with increasing number of binary relevance problems from 1 to 9 QFs. Increasing the number of QFs, has a minor effect on the complexity of M2L, while the complexity of IV3-2L increases with more trained QFs.

Training all layers of MobileNetV2 as MFull selector results in nine times higher complexity than M2L and contains 541.11*9 M MACs since all layers of the architecture are used for each binary relevance training. Although all layers of MobileNetV2 are trained for MFull selector, its complexity is less than the complexity of IV3-2L. Based on the application and the allowed complexity and accuracy loss, one may choose one of M2L, MFull, or IV3-2L selectors. For example, the complexity of M2L is relatively low; although its RA performance is slightly less than the performance of MFull and IV3-2L, it can be used in employment on edge devices.

Table 4.5: Comparison of complexity of original MobileNetV2 architecture with Binary MobileNetV2 and M2L

MobileNetV2	Binary MobileNetV2	M2L
582.19 M MACS	541.11 M MACS	541.49 M MACS

Table 4.6: Impact of number of trained QFs on Complexity of M2L

num QFs	Number of MACS
1 QFs	541.11 M MACCs
2 QFs	541.16 M MACCs
3 QFs	541.21 M MACCs
4 QFs	541.26 M MACCs
5 QFs	541.30 M MACCs
6 QFs	541.35 M MACCs
7 QFs	541.40 M MACCs
8 QFs	541.45 M MACCs
9 QFs	541.49 M MACCs

Table 4.7: Comparison of complexity of original IV3 architecture with Binary IV3 and IV3-2L.

IV3 Full	Binary IV3	IV3-2L
5713.23 M MACS	5711.17 M MACS	8819.18 M MACS

Table 4.8: Impact of number of trained QFs on Complexity of IV3-2L

num QFs	Number of MACS
1 QFs	5711.17 M MACCs
2 QFs	6099.67 M MACCs
3 QFs	6488.18M MACCs
4 QFs	6876.68 M MACCs
5 QFs	7265.18 M MACCs
6 QFs	7653.68 M MACCs.
7 QFs	8042.18 M MACCs
7 QFs	8430.68 M MACCs
9 QFs	8819.18 M MACCs

4.5 Deep Selector-JPEG Output QF Selection Demonstration

This section demonstrates the performance of Deep Selector-JPEG on image number 651 in the validation set of the ImageNet dataset as an example input image. We compress the original image with the selected QF by Deep Selector-JPEG and present the corresponding PSNR and CR of the compressed image and the Ground Truth (GT) rank of the compressed image on test DNNs.

Output QF Selection for PSNR $\geq$ 26dB:

For target PSNR $\geq$ 26 dB, we present selection for two target CRs. For target CR = 10x, all M2L, MFull and IV3-2L select QF=10 which results in PSNR = 29.92 dB, CR=11.8x for this image. The GT rank of the original image and the compressed version on all test DNNs are shown in table 4.9. We can see that for IV3, using the compressed version improved the GT rank. For other DNNs, the GT rank of the original is kept while CR is significantly increased.

For target CR = 3x, M2L and MFull select QF = 20 that results in PSNR 31.24 dB and CR 8.98x, while IV3-2L choose QF=50 resulting in PSNR 34.27 dB and CR 5.64x. The GT rank of compressed version using the selector's output for all test DNNs is shown in table 4.10. While M2L and MFull maintain the original GT rank, IV3-2L selection improves GT rank.

Output QF Selection for PSNR$\geq$ 28 dB:

For PSNR $\geq$ 28 dB, we demonstrate the selection for two target CRs. For target CR = 7x, all three selectors select QF=20, resulting in PSNR = 31.24 dB and CR = 8.98x. On the other hand, for this PSNR and target CR = 3x, MFull and M2L select QF20, and IV3-2L selects QF=50. The GT rank of the original image and the compressed image for all test DNNs for these settings are shown in 4.11 and 4.12.

Output QF Selection for PSNR $\geq$ 30 dB, PSNR $\geq$ 32 dB:

For PNSR $\geq$ 30 dB and PNSR $\geq$ 32 dB, due to the limited range of CR, we only present selection for target CR=3x. For PNSR $\geq$ 30 dB, all selectors select QF=40 that result in PSNR=33.47 dB and CR=6.39x. The original and selected image's GT ranks are presented in 4.13. On the other hand, for PNSR $\geq$ 32 dB, QF=60 is selected by all selectors resutling in PSNR=35.18 dB and CR=4.96x. The GT ranks are presented in table 4.14. These two PSNR constraints' rank tables reveal that the selected QF always keeps the original GT rank.

These results confirm that for image number 651, for different settings of PSNR and CR, Deep Selector-JPEG always either keeps or improves the original GT rank.

Table 4.9: The GT ranks of the original image and compressed image at the selected QF, output of selection using M2L, MFull, IV3-2L for target PSNR $\geq$ 26 and CR=10x

DNN \ **Selector**	ORG	M2L	MFull	IV3-2L
IV3	2	1	1	1
ResNet-V2-50	1	1	1	1
InceptionResnetV2	1	1	1	1
MobileNetV1	1	1	1	1
IV1	1	1	1	1
IV4	1	1	1	1
ResNet-V2-101	1	1	1	1
MobileNetV2	1	1	1	1
Pnasnet_Large	1	1	1	1
nasnet_Large	1	1	1	1

Table 4.10: The GT ranks of the original image and compressed image at the selected QF, output of selection using M2L, MFull, IV3-2L for target PSNR $\geq$ 26, CR=3x

DNN \ **Selector**	ORG	M2L	MFull	IV3-2L
IV3	2	2	2	1
ResNet-V2-50	1	1	1	1
InceptionResnetV2	1	1	1	1
MobileNetV1	1	1	1	1
IV1	1	1	1	1
IV4	1	1	1	1
ResNet-V2-101	1	1	1	1
MobileNetV2	1	1	1	1
Pnasnet_Large	1	1	1	1
nasnet_Large	1	1	1	1

Table 4.11: The GT ranks of the original image and compressed image at the selected QF, output of selection using M2L, MFull, IV3-2L for target PSNR ≥ 28 and CR=9x

DNN \ Selector	ORG	M2L	MFull	IV3-2L
IV3	2	2	2	2
ResNet-V2-50	1	1	1	1
InceptionResnetV2	1	1	1	1
MobileNetV1	1	1	1	1
IV1	1	1	1	1
IV4	1	1	1	1
ResNet-V2-101	1	1	1	1
MobileNetV2	1	1	1	1
Pnasnet_Large	1	1	1	1
nasnet_Large	1	1	1	1

Table 4.12: The GT ranks of the original image and compressed image at the selected QF, output of selection using M2L, MFull, IV3-2L for target PSNR ≥ 28, CR=3x

DNN \ Selector	ORG	M2L	MFull	IV3-2L
IV3	2	2	2	1
ResNet-V2-50	1	1	1	1
InceptionResnetV2	1	1	1	1
MobileNetV1	1	1	1	1
IV1	1	1	1	1
IV4	1	1	1	1
ResNet-V2-101	1	1	1	1
MobileNetV2	1	1	1	1
Pnasnet_Large	1	1	1	1
nasnet_Large	1	1	1	1

Table 4.13: The GT ranks of the original image and compressed image at the selected QF, output of selection using M2L, MFull, IV3-2L for target PSNR $\geq$ 30 dB and CR=3x

DNN \ **Selector**	ORG	M2L	MFull	IV3-2L
IV3	2	2	2	2
ResNet-V2-50	1	1	1	1
InceptionResnetV2	1	1	1	1
MobileNetV1	1	1	1	1
IV1	1	1	1	1
IV4	1	1	1	1
ResNet-V2-101	1	1	1	1
MobileNetV2	1	1	1	1
Pnasnet_Large	1	1	1	1
nasnet_Large	1	1	1	1

Table 4.14: The GT ranks of the original image and compressed image at the selected QF, output of selection using M2L, MFull, IV3-2L for target PSNR $\geq$ 32 and CR=3x

DNN \ **Selector**	ORG	M2L	MFull	IV3-2L
IV3	1	1	1	1
ResNet-V2-50	1	1	1	1
InceptionResnetV2	1	1	1	1
MobileNetV1	1	1	1	1
IV1	1	1	1	1
IV4	1	1	1	1
ResNet-V2-101	1	1	1	1
MobileNetV2	1	1	1	1
Pnasnet_Large	1	1	1	1
nasnet_Large	1	1	1	1

4.6 Summary

This chapter demonstrated the capability of Deep Selector-JPEG to achieve better RA performance than the default JPEG compression on a set of tested DNNs for different human vision constraints. The selector complexity has been discussed. Moreover, it has been shown that the complexity of M2L is comparable to the complexity MobileNetV2 original architectures while its RA performance is comparable to the RA performance of MFull and IV3-2L. That makes it efficient for employing on edge devices.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This work presents Deep Selector-JPEG, an adaptive JPEG compression method that serves both image classification and Human Vision (HV). Towards this end, for each original image, Deep Selector-JPEG selects adaptively a Quality Factor (QF) to compress the image with its gains in classification accuracy at the same compression ratios as high as $\approx 1\%$ in comparison with the default JPEG. These gains are achieved while preserving compliance with JPEG standard, showing consistent performance across different DNNs, and satisfying HV.

The adaptive selection happens within a predetermined set of feasible QFs to meet a certain PSNR threshold with a high probability to serve HV. Multiple QFs within the feasible set can be suitable (ON) for compressing an image for a Deep Neural Network (DNN). A QF is suitable (ON) to compress an image for a DNN if compressing at this QF maintains the ground truth rank of the original input for the DNN classifier. For a given image, Deep Selector-JPEG first determines the QFs that are ON among the set of feasible QFs by solving a Multilabel Classification Problem (MLC) through Binary Relevance (BR) and then select the least QF that is ON to maintain a certain Compression Ration (CR) constraint. The Binary relevance involves training an independent binary DNN classifier for each QF within the feasible set to predict the ON/OFF labeling for each input image. We have presented two forms of Deep Selector-JPEG; The first form freezes a large portion of a DNN architecture and uses it as a common feature extractor for all binary classification problems, while the rest of the architecture is used for training each binary classification problem. The second form utilizes the entire architecture for training each binary classification problems. MobileNetV2 and Inception V3 are used in the design of the first form. Moreover, MobileNetV2 is used for designing the second form. Given a target CR, we empirically derive a subset of feasible QFs for this target CR and select the least QF that is ON in this set.

For a given PSNR constraint, our results show classification accuracy gains up to $\approx 1\%$ over default JPEG on the same CR. Applying both forms of Selector-JPEG using both IV3 and MobileNetV2 on different DNN models in the literature shows consistent accuracy gains over default JPEG at the same CR, which reveals generality of Selector-JPEG. This shows that considering computer vision as well as human vision in designing a compression method can achieve general RA improvement on different DNNs.

5.2 Future Work

This work can be extended in several ways. We discuss some of these possible extensions in the following subsections.

5.2.1 Reducing Complexity of Deep Selector-JPEG

One of Deep Selector-JPEG's limitations is its complexity, especially for employment on edge devices and the Internet of Things (IoT) applications. Although the M2L presented in chapter 3.6.1 produces acceptable RA results with reasonable complexity 541.11 MACs, the complexity of Deep Selector-JPEG still needs to be decreased for better employment on edge devices. Comparing the RA performance of M2L and MFull shows that comparable RA performance is achieved with less complexity. Since each binary relevance problem is practically a binary classifier, we believe an architecture with fewer parameters would be enough to achieve similar performance. This opens a direction of seeking efficient architectures that fits the Deep Selector-JPEG framework. One possible suggestion is to build up this architecture based on the currently used MobileNetV2. One may freeze the first n layers of MobileNetV2 as a common feature extractor and, instead of training the rest of the architecture, add a fully connected layer after these n layers to reduce the number of parameters. One can decrease n until the lowest number of parameters is achieved, and the RA performance is acceptable.

5.2.2 Applying on Other Codecs, and CV tasks

Deep Selector-JPEG is adaptable for any compression method with a parameter to control the quality of images. That means for a given compression method X with quality parameter y, we can train our architecture to create a selector, called Deep Selector-X to derive ON/OFF labels for a range of y values and select the lowest y that is ON among a set of empirically derived y values for a target CR that is suitable for computer vision and satisfy a HV criteria. (see Figure 5.1). For example, this X can be HEVC [25], one state of the art compression methods for video coding to select the best Quantization Parameter (QP) for each image under a certain PSNR and CR constraint. This may result in better RA performance than Deep Selector-JPEG since HEVC provides better rate-distortion trade off than JPEG; also, it is possible to apply our method on top of tailored compression methods for computer vision, like DeepNJPEG [17] and GRACE [31] introduced in chapter 3.2 and further improve their performance.

DNNs are used for different computer vision tasks like segmentation, object detection, image localization, face detection, etc. Our experiments are limited to classification, which is one popular example of computer vision applications. Our methodology can further be extended to other computer vision tasks like segmentation and object detection to show its generality for different computer vision tasks. This extension is simply done by changing ON/OFF labeling from classifier DNN to ON/OFF labeling based on other computer vision tasks.

Combining these two extensions, the target of Deep Selector-X , would be both human vision and different computer vision applications such as segmentation, object detection etc. This framework is shwon in Figure 5.1.

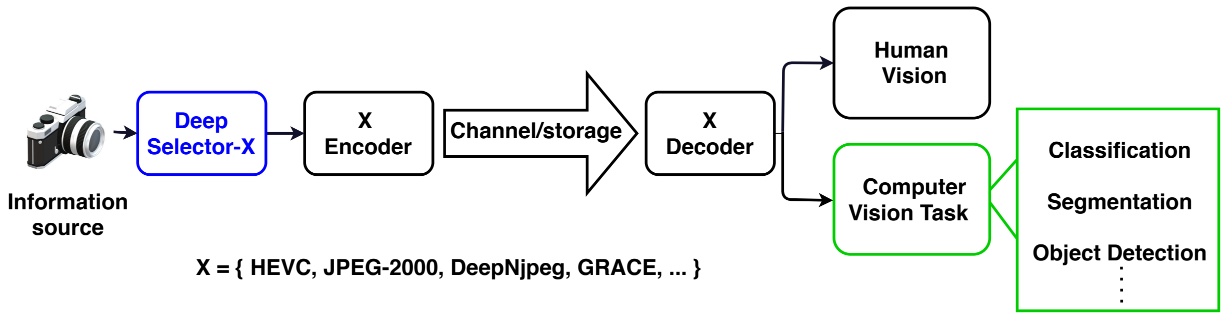


Figure 5.1: Extended frame work of our methodology

5.2.3 Training DNN with the output of Deep Selector-JPEG

For a target CR, Selector-JPEG provides a compressed version of the ImageNet validation dataset that performs better than the compressed dataset with default JPEG at the same CR while inferencing a classifier DNN. One interesting question is the effect of training a classifier DNN with the compressed version of images at selected QF from Deep Selector-JPEG. Authors in [17] showed that training a neural network using a dataset compressed by a specific JPEG QF will increase the inference performance on the dataset that is also compressed with the same QF. We expect that training a DNN using the dataset that is the output of selection done by Deep Selector-JPEG will further improve the RA performance of DNN.

References

[1] Cisco Forecast Report. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] Tensorflow Research Repository. https://github.com/tensorflow/models/tree/master/research/slim/.

[3] S. Arora, A. Bhaskara, R. Ge, and T. Ma. Provable Bounds for Learning Some Deep Representations. In *International Conference on Machine Learning*, pages 584–592, 2014.

[4] DH. Ballard and CM. Brown. Computer Vision Prentice-Hall Inc. *Englewood Cliffs, New Jersey*, pages 313–437, 1982.

[5] L. D. Chamain, S. S. Cheung, and Z. Ding. Quannet: Joint Image Compression and Classification Over Channels with Limited Bandwidth. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 338–343, 2019.

[6] A. Damien. Multi-GPU Training Example. https://wizardforcel.gitbooks.io/tensorflow-examples-aymericdamien/content/6.2_multigpu_cnn.html.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[8] H. Amer. Image/Video Compression: Human and Computer Vision Perspectives, 2020.

[9] K. He, X. Zhang, S. Ren, and J.Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

[11] Y. Hu, S. Yang, W. Yang, L. Duan, and J. Liu. Towards Coding for Human and Machine Vision: A Scalable Image Coding Approach. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2020.

[12] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] R. Klette. *Concise Computer Vision*. Springer, 2014.

[14] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *nature*, 521(7553):436, 2015.

[15] H. Li, Y. Guo, Z. Wang, S. Xia, and W. Zhu. AdaCompress: Adaptive Compression for Online Computer Vision Services. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2440–2448, 2019.

[16] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive Neural Architecture Search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[17] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan. DeepN-JPEG: A Deep Neural Network Favorable JPEG-Based Image Compression Framework. In *Proceedings of the 55th Annual Design Automation Conference*, page 18. ACM, 2018.

[18] Tensorflow Organization. Threading and Queues. https://tensorflow.juejin.im/api_guides/python/threading_and_queues.html.

[19] W.B. Pennebaker and J.L Mitchell. *JPEG: Still Image Data Compression Standard*. Springer Science & Business Media, 1992.

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[21] L. Shapiro and G. Stockman. Computer Vision Prentice Hall. *Inc., New Jersey*, 2001.

[22] Q. Shen, J. Cai, L. Liu, H. Liu, T. Chen, L. Ye, and Z. Ma. CodedVision: Towards Joint Image Understanding and Compression via End-to-End Learning. In *Advances in Multimedia Information Processing – PCM 2018*, pages 3–14. Springer International Publishing, 2018.

[23] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K.B Letaief. Communication-Efficient Edge AI: Algorithms and Systems. *arXiv preprint arXiv:2002.09668*, 2020.

[24] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, Analysis, and Machine Vision*. Cengage Learning, 2014.

[25] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[26] C. Szegedy, S. Ioffe, V. Vanhoucke, and Alex A. Alemi. Inception-V4, Inception-Resnet and the Impact of Residual Connections on Learning. *arXiv preprint arXiv:1602.07261*, 2016.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[29] D. Taubman and M. Marcellin. *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice*, volume 642. Springer Science Business Media, 2012.

[30] G. K. Wallace. The JPEG Still Picture Compression Standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, Feb 1992.

[31] X. Xie and K. Kim. Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[32] Y. Jiang. New Convolutional Neural Network Topology with Compressed Information to Enhance Accuracy for Image Classification Task, 2019.

[33] B. Zoph, V. Vasudevan, J. Shlens, and Q.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.